

src/mvp/aaalloc.f

```
1:      subroutine AAALOC
2:      c
3:      C/IF .NOT.DYNAMIC
4:      c
5:      C=<MVP>=====
6:      C Purpose: memory size determination in non-dynamic (static) memory
7:      C      allocation mode.
8:      c
9:      C      In environment such as FACOM-M series or VP series,
10:     C      users can set actual memory size as follows;
11:     C      * copy this routine.
12:     C      * modify size parameters (MAXMEM & LMAX)
13:     C      * compile & link before execution.
14:     c
15:     C called in : MAIN
16:     C=====
17:     c
18:     c === size of task shared memory (word) ==
19:     c
20:     c      parameter (MAXMEM = 6000000)
21:     c      parameter (MAXMEM = 18000000)
22:     c      parameter (MAXMEM = 28000000)
23:     c      parameter (MAXMEM = 120000000)
24:     c
25:     c === size of task local memory (word) for multitasking ==
26:     c
27:     C/IF PARA
28:     *      parameter (LMAX = 120000000)
29:     C/ENDIF
30:     c
31:     common /ARRAY/ LIMIT,IDUM,A(MAXMEM)
32:     common /ARRAYZ/ IAINFL(3)
33:     c
34:     c ... for single tasking mode, array A & H are equivalent.
35:     c
36:     C/IF .NOT.PARA.OR.PARA(VPP)
37:     REAL H(1)
38:     integer IAINFL(3)
39:     equivalence (LIMITL,LIMIT), (IAINF,IAINFL), (H,A)
40:     C/ENDIF
41:     c
42:     c ... for multi tasking mode ....
43:     c
44:     C/IF PARA( SX* )
45:     *      local common /LARRAY/ LIMITL, ILDUM, H(LMAX)
46:     *      local common /LARRAYZ/ IAINFL(3)
47:     C/ELSEIF PARA( CRAY )
48:     *      task common /LARRAY/ LIMITL, ILDUM, H(LMAX)
49:     *      task common /LARRAYZ/ IAINFL(3)
50:     C/ENDIF
51:     c
52:     c -----
53:     c
54:     LIMIT = MAXMEM
55:
56:     C/IF PARA( CRAY SX* )
57:     *      LIMITL = LMAX
58:     C/IF ENDF
59:
60:     C/ENDIF
61:     return
62:     end
```

src/mvp/acaloc.f

```
1:      subroutine ACALOC
2:      C
3:      C=<MVP>=====
4:      C Purpose: memory size setting for character memory
5:      C called in : MAIN
6:      C-----
7:      C
8:      C Estimation of variable sized character area requirement for MVP.
9:      C (November 1997)
10:     C
11:     C IZNAM(NINPZ)*12 : input zone name
12:     C NUCID(NUC)*16   : nuclide ID
13:     C MATP(NUC)*136   : neutron library comment
14:     C MATP(NUC)*136   : photon library comment
15:     C KREGS(NREG)*(LNAM) : list of region/frame names
16:     C TNAME(*)*(LNAM)   : list of region/frame names
17:     C
18:     C LNAM=12 or 16 depending on system.
19:     C
20:     C << required size (bytes) >>
21:     C
22:     C NINPZ*12 + NUC*( 16 + 136 + 136) + NREG*LNAM + <number of names>*LNAM
23:     C
24:     C For a large problem (LNAM=16);
25:     C
26:     C   #of input zone : 2000
27:     C   #of nuclide   : 300
28:     C   #of region    : 2000
29:     C   #of names     : 2000
30:     C
31:     C size is 174400 bytes = 43600 * 4 bytes
32:     C
33:     C So 60000*4 bytes may be sufficient for most problem, I hope.
34:     C
35:     C=====
36:     C
37:     C === size of character memory ( 4 bytes as a unit) ==
38:     C
39:     C   PARAMETER (LCMAX = 8000000)
40:     C
41:     C ... character data area (task shared) ....
42:     C
43:     C   integer LIMITC,IDUMC
44:     C   common /CARRAY/ CHA(LCMAX)
45:     C   common /CARRAYZ/ LIMITC, IDUMC, IAINFC(3)
46:     C   character*4 CHA
47:     C   integer IAINFC
48:     C
49:     C ... character data area (task local) .... for future use
50:     C
51:     C   integer LIMITC,IDUMC
52:     C   character*4 CHH
53:     C   common /CLARRAY/ CHH(1)
54:     C   common /CLARRAYZ/ LIMITLC, ILDUMC, IAINFLC(3)
55:     C
56:     C
57:     C   LIMITC = LCMAX
58:     C
59:     C   IAINFC(1) = 1
60:     C   IAINFC(2) = IAINFC(1)
61:     C   IAINFC(3) = 0
62:     C
63:     C   return
64:     C   END
```

src/mvp/action.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine ACTION( TITLE, A, H, CHA )
3: C=<MVP>=====
4: C PURPOSE:  Control of monte carlo run in MVP-code
5: C CALLED IN: ACTSGL,ACTMPP,ACTVPP,ACTSMP
6: C CALLS:  SEAOONE MIRROR RSKIP HEADER SOURCE CPUTM FLIGHT MNTCPU LATICE
7: C          TALSUM VMNTRF SELONE NEUTR SEARCH MEMMEM FLIONE PHOTR VMNTR0
8: C          VMNTR2
9: C
10: C=====
11:      character TITLE(2)*72
12:      real A(*)
13:      real I(*)
14:      character*4 CHA(*)
15: C
16:      include 'INC/_KPIDS'
17:      include 'INC/_NGPS'
18: C
19:      include 'INC/_FLAGS'
20:      include 'INC/_shared/INC/_SIZES'
21:      include 'INC/_SIZES2'
22:      include 'INC/_XBANK'
23:      include 'INC/_FBANK2'
24:      include 'INC/_SBANK'
25:      include 'INC/_STACK'
26:      include 'INC/_XWORK'
27:      include 'INC/_shared/INC/_PGEOM'
28:      include 'INC/_CXSEC'
29:      include 'INC/_PXSEC'
30:      include 'INC/_shared/INC/_PVRED'
31:      include 'INC/_PSOUR'
32:      include 'INC/_shared/INC/_PTALY0'
33:      include 'INC/_PTALY'
34:      include 'INC/_PTALY2'
35:      include 'INC/_shared/INC/_PTLSP'
36:      include 'INC/_shared/INC/_STALY'
37: C
38:      include 'INC/_shared/INC/_COUNTS'
39:      include 'INC/_shared/INC/_TIMEDT'
40:
41: C/#IF PARA(PVM)
42:      include 'INC/_shared/INC/_PVMPARA'
43: C/#ELSEIF PARA(MPI)
44: *      include 'mpif.h'
45: C/#ELSEIF PARA(VPP)
46:      include 'INC/_shared/INC/_VPPPARA'
47: C/#ENDIF
48:      include 'INC/_shared/INC/_TASKDT'
49: C
50: C..... WORKING ARRAY POINTERS .....
51: C
52:      include 'INC/_WKSOU'
53:      include 'INC/_WKFSS'
54:      include 'INC/_WKFL1'
55:      include 'INC/_WKSE1'
56:      include 'INC/_WKCOL'
57:      include 'INC/_WKMIR'
58:      include 'INC/_WKLAT'
59:      include 'INC/_WKFLA'
60:      include 'INC/_WKSEA'
61:      include 'INC/_WKTLS'
62:      include 'INC/_WKPHT'
63: C
64:      include 'INC/_WKNXT'
65:      include 'INC/_shared/INC/_IOUNIT'

```

```

66:      include 'INC/_IOUNIT2'
67: C..... PERTURBATION POINTERS ..... ! pert
68:      include 'INC/_PERT' ! pert
69: C
70: C ... local variables ...
71: C
72: C/#IF INTEGER8
73: *      integer*8 NTGEN
74: C/#ELSE
75:      integer NTGEN
76: C/#ENDIF
77:      logical JMEM
78: C
79:      real*8 DT, WSUMB
80:      integer KYPOS(3) ! scat-mtx
81: C
82: C-----
83: C
84: C ... elapsed time & CPU time at startup
85: C
86:      call TOKEI(TE0,0)
87:      call CPUTM(T0)
88: C
89: C =====
90:      call HEADER(IOW,'MONTE CARLO RUN')
91: C =====
92: C
93: C ... update special event counters ( COMMON /COUNTS/ )
94: C
95:      MTANG = 0
96:      MDEFR = 0
97: C .... for endless loop detection ..
98:      DESUM = 0
99:      DESUM2 = 0
100:      JLOOP = 0
101:      NLOOP = 0
102: C
103: C ... print batch monitor of sources , keff's if nonzero
104: C
105:      JBPRNT = 1
106: C
107: C .... for FNSSC2 routine ..
108: C
109:      IFNSSC = 0
110: C
111:      write(IOW,7000) TITLE
112: 7000 format(/1X,' PROBLEM TITLE : ',A72/21X,A72)
113: C
114: C/#IF PARA(SX* CRAY*)
115: *      call MVPSYNC_LOCK(2)
116: C/#ENDIF
117: C
118:      write(IOW,'/' TASK ',i5,' START ==== CPU TIME ',lp,
119: & E11.4,' (SEC) ELAPSED ==== ',E11.4,' (SEC)')')
120: & IDTASK,T0,TE0
121: C
122: C/#IF PARA(SX* CRAY*)
123: *      call MVPSYNC_UNLOCK(2)
124: C/#ENDIF
125: C
126: C ... synchronize all task for safety ....
127: C
128: C/#IF PARA(SX* CRAY*)
129: *      call MVPSYNC_BARRIER( 1 )
130: C/#ENDIF

```

src/mvp/action.f

```

131: C
132: C
133: ccccc LH = IAINFL(1)
134:     LH = 1
135: c##<2007/03/14:PN3:
136:     NUC_MAX = max(NUC, NUCPN)
137: c##>
138:
139:     KY = 0
140:     KYPOS(1) = 0
141:     KYPOS(2) = 0
142:     KYPOS(3) = 0
143:     if ( JSCTM.ne.0.or.JSCMU.ne.0 ) then
144:         if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) then
145:             KY = KY + 1
146:             KYPOS(1) = KY
147:         end if
148:         if ( JMICE(6).ne.0.or.JMACE(6).ne.0 ) then
149:             KY = KY + 1
150:             KYPOS(2) = KY
151:         end if
152:         if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) then
153:             KY = KY + 1
154:             KYPOS(3) = KY
155:         end if
156:     end if
157:     KJSCTM = max( 1, JSCTM )
158: C/#IF ARGSAVE
159: *
160: C-----
161: C .... INITIALIZATION .....
162: C-----
163: C/# IF SOURCE(NEW)
164:
165:     call SOURC0(IOW,A,H,H,
166:     N NTGEN, NDEAD, NFISB, JBRPNT, H(LNLOST), NEVENT, NMKREG,
167:     X MCX,MCXP, NUC,NPATOM,NSMIC,
168:     & H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),
169:     & H(LKKP),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LLEE),
170:     & H(LIBREG),H(LIBSPC),H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),H(LXXXF),
171:     & H(LYYYF),H(LZZZF),H(LIZZF),H(LLEEF),H(LINUF),H(LLEVL),
172:     & H(LLZZ),H(LLPOS),H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),
173:     & H(LIBRGF),H(LIBSPF),H(LKLSFF),H(LXIM),
174:     & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK, MIBNK,
175:     & H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK, MIFBK,
176:     & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),
177:     & A(LKZAA),A(LIPCEL),A(LLTYP),A(LXIMP),
178:     & H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSDED),H(LIZLAT),
179:     & H(LNXL),H(LSRCSP),A(LSOUR),A(LLENGYB),A(LTIMEB),
180:     & A(LWGTFF),H(LWSUM),H(LXSOC),H(LNCNTR),H(LWCNTR),EBOTX,ETOPX,
181:     & H(LXSXV),H(LXAVT),H(LAAVT),H(LXAVT),H(LXAVT),H(LXAVT),
182:     & A(LCX),A(LCX),A(LKLIB1),A(LKLIB1),A(LKLIB2),A(LKLIB2S),
183:     & A(LKLIB2),NNK,NMT,EINCD,A(LCRES),A(LKRES),H(LSGTAL),MCRES,
184:     & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LKLB2),NMT,
185:     D NPDET, NPLEN, A(LJPUSD),A(LXPDET),A(LIPDET),A(LIPDT2),
186:     S A(LJSPDT),NIMPT, NDIMPT, H(LIMSFL),H(LIMNFL),
187:     S H(LIMZFL),H(LIMDED),H(LIMSLT),H(LIMNLT),H(LIMZLT),
188:     B H(LLZZI), H(LLPOS),H(LLCRSI),H(LOPTI), H(LPATH), H(LKDETP),
189:     B H(LKLSPI),H(LSMACI),H(LMMACI),
190:     & A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK,MXREJ,
191:     & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
192:     & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
193: c##<2007/03/14:PN3:PN4:
194: c## & H(P1(14)),H(P1(15)))
195: & H(P1(14)),H(P1(15)),

```

```

196: & NUCPN, NUCPNI, NSMICPN, MCXPN, NUC_MAX, H(LSMICPN), NMTPN,
197: & A(LCXPN), A(LCXPN), A(LKLBPN1), A(LKLBPN2), A(LXLBPN1), EBOTLPN,
198: & H(LKPNFG) )
199: c##>
200:
201: C/# ELSE
202:
203: *     call SOURC0(IOW,A,H,H,IRAND,NTGEN,NBANK,NDEAD,NGROUP,NGP1,
204: *     & NGP2,NZONE,NFBANK,NFBANK0,NFISB,NEST,JBRPNT,H(LNLOST),NEVENT,NLBZ,
205: *     & NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,MCX,MCXP,NUC,NPATOM,NSMIC,NMKREG,
206: *     & H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
207: *     & H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LLEE),H(LIBREG),H(LIBSPC),
208: *     & H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),H(LXXXF),H(LYYYF),H(LZZZF),
209: *     & H(LIZZF),H(LLEEF),H(LINUF),H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),
210: *     & H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),H(LIBRGF),H(LIBSPF),
211: *     & H(LKLSFF),H(LXIM),H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
212: *     & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),
213: *     & A(LKZAA),A(LIPCEL),A(LXIMP),H(LLSFFL),H(LNFFL),H(LIZFFL),
214: *     & H(LLSDED),H(LSLAT),H(LIZLAT),H(LNXL),
215: *     * A(LKSOUR),A(LISZON),A(LSOUR),A(LPSPAC),A(LPENRG),A(LKENRG),
216: *     * A(LLENGYB),A(LLENGPB),A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),
217: *     * A(LNSANG),A(LSANG),A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),
218: *     * A(LWGTFF),H(LWSUM),H(LXSOC),H(LNCNTR),
219: *     & H(LWCNTR),EBOT,ETOP,NLENG,H(LXSXV),H(LXAVT),H(LAAVT),H(LXAVT),
220: *     & A(LCX),A(LCX),A(LKLIB1),A(LKLIB1),A(LKLIB2),A(LKLB2S),A(LXLIB2),
221: *     & NNK,NMT,EINCD,A(LCRES),A(LKRES),H(LSGTAL),MCRES,NSTAL,
222: *     & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LKLB2),NMT,EBOT,ETOP,
223: *     & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
224: *     & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
225: *     & H(P1(14)),H(P1(15)) )
226:
227: C/# ENDIF
228: C
229: C
230: C*     if( JALLZ.eq.1 .or. JONEZ.eq.0 ) then
231: C
232: C*     call FLIGH0( IOW, NBANK,NZONE,NSDA,NZDA,NGROUP,NGP1,NGP2,
233: C*     N NREG,NRESP,NCOLS,NDEAD,H(LNLOST),
234: C*     N IRAND,DINF,NEST,NLATT,NLBZ,NMAT,MB,NUC,NSMAC,NSMIC,NEMIC,NSTAL,
235: C*     B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
236: C*     B H(LWWW),H(LIGG),H(LTTT),H(LLEVL),H(LLZZ),H(LLPOS),
237: C*     B H(LLCRS),H(LSMAC),A(LLMAC),H(LKMAC),H(LMMAC),A(LLMIC),
238: C*     B H(LSGTAL),A(LLPDEN),A(LINUCT),A(LDENST),H(LDNFLX),H(LLSFFL),
239: C*     S H(LNFFL),H(LIZFFL),H(LLSCOL),H(LLSSRC),H(LNNXT),H(LIZNXT),
240: C*     G H(LLSDED),A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),
241: C*     G A(LNKZAA),A(LDAL),A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),
242: C*     G A(LKSLAT),A(LLTYP),A(LMLBZZ),A(LKLATT),A(LCELSZ),
243: C*     T A(LRESP),H(LFLTR),H(LRETR),H(LRMIC),H(LRSTR),H(LNLEAK),
244: C*     H(LWLEAK),H(LNCNTR),H(LWCNTR),A(LDNZON),A(LLEMIC),H(LIBREG),
245: C*     W H(P6(1)),H(P6(2)),H(P6(3)),
246: C*     W H(P6(4)),H(P6(5)),H(P6(6)),H(P6(7)),H(P6(8)),H(P6(9)),
247: C*     W H(P6(10)),H(P6(11)),H(P6(12)),H(P6(13)),H(P6(14)),H(P6(15)),
248: C*     W H(P6(16)),H(P6(17)),H(P6(18)),H(P6(19)),H(P6(20)),H(P6(21)),
249: C*     W H(P6(22)),H(P6(23)),H(P6(24)),H(P6(25)),H(P6(26)),H(P6(27)),
250: C*     W H(P6(28)),H(P6(29)),H(P6(30)) )
251: C
252: C*     call SEARCO( IOW, NBANK,NZONE,NSDA,NZDA,H(LNLOST),
253: C*     N NDEAD,NREFS,NGROUP,NGP1,NGP2,NEVENT,NREG,NTIME,
254: C*     N IRAND,NLBZ,NEST,DEPS,NSMAC,NSMIC,MB,NSTAL,NUC,NSPACE,NSUZON,
255: C*     B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
256: C*     B H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LLEE),H(LXIM),H(LKLSF),
257: C*     B H(LSMAC),H(LSMIC),H(LSGTAL),H(LKMAC),H(LMMAC),H(LKSPI),
258: C*     L H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LIBREG),H(LIBSPC),
259: C*     H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSSRC),H(LNNXT),H(LIZNXT),
260: C*     S H(LLSDED),H(LSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LKSREF),

```

src/mvp/action.f

```

261: C*      * H(LLSLAT), H(LIZLAT), H(LNXLT) ,
262: C*      G A(LSDA ), A(LKZMAT), A(LKZREG), A(LKZDA ), A(LKZAA ), A(LNKZAA),
263: C*      * A(LKCELL), A(LIPCEL), A(LMLBZZ), A(LISUSP) ,
264: C*      T A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM, H(LNKILD),H(LWKILD),
265: C*      T H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
266: C*      W H(P7(1)), H(P7(2)), H(P7(3)), H(P7(4)),H(P7(5)),H(P7(6)),
267: C*      W H(P7(7)),H(P7(8)),H(P7(9)),H(P7(10)),H(P7(11)),H(P7(12)),
268: C*      W H(P7(13)),H(P7(14)),H(P7(15)),H(P7(16)),H(P7(17)),H(P7(18)),
269: C*      W H(P7(19)),H(P7(20)),H(P7(21)),H(P7(22)),H(P7(23)),H(P7(24)),
270: C*      W H(P7(25)),H(P7(26)),H(P7(27)),H(P7(28)),H(P7(29)),H(P7(30)) )
271: C
272: C*      else if( JALLZ.eq.0 .or. JOMEZ.eq.1 ) then
273:
274: *      call FLION0(IOW,NDEAD,NCOLS,NCOLP,H(LNLOST),
275: *      N NKTCS,MB,NSMAC,NSMIC,NEMIC,NEVENT,H(LXXX),H(LYYY),H(LZZZ),
276: *      B H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LKKP),H(LIZZ),H(LLEE),H(LIGG),
277: *      B H(LTTT),H(LTTT),H(LLEVL),H(LLZZ),H(LLPOS),
278: *      B H(LLCRS),H(LKLSF),H(LIBSPC),H(LIBREG),H(LSMAC),A(LLMAC),H(LKMAC),
279: *      B H(LMMAC),H(LSMIC),A(LLMIC),H(LSGTAL),H(LKSPI),H(LDBNK),KDBNK,
280: *      B MDBNK,H(LIBNK),KIBNK,MIBNK,A(LLPDEN),A(LINUCT),A(LDNSTP),
281: *      M H(LDNFLX),A(LLPDNP),A(LIATMT),A(LDNSTP),H(LLSFFL),H(LNFFL),
282: *      S H(LIZFFL),H(LLSCOL),H(LLSCLP),H(LLSSRC),H(LNNXT),H(LIZNXT),
283: *      S H(LLSDED),A(LSDA),A(LKZMAT),A(LKZREG),A(LKSPCU),A(LKTCSP),
284: *      G A(LKZDA),A(LKZAA),A(LDALT),A(LKDALT),A(LNVLAT),A(LSZLAT),
285: *      G A(LIPLAT),A(LIPCEL),A(LKSLAT),A(LLTYP),A(LMLBZZ),A(LKZLBZ),
286: *      G A(LKCELL),A(LICTYP),A(LKLATT),A(LCLLSZ),A(LPNND),A(LKNND),
287: *      G A(LPPPF),A(LKPPF) ,
288: *      T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
289: *      T A(LPSALP),A(LPSXYZ),H(LDTALY),A(LRESP),H(LFLTR),A(LRETR),
290: *      T H(LRMIC),H(LRSTR),H(LTFLH),H(LNCNTR),H(LWCNTR),A(LDNZON),
291: *      T A(LLEMIC),A(LTIMEB),NMKREG,A(LMKREG),NTSRF,A(LKTSRF),A(LITSRF),
292: *      T A(LIDSRF),A(LANGLB),NANGLE,A(LENGYB),H(LDNFLXSM),H(LRMICSM),
293: *      & A(LCX),A(LKLIB1),A(LKLIB2),A(LXLIB1),A(LXLIB2),
294: C##<2007/03/14:PN3:
295: C##*      & A(LCXP), A(LKLB1), A(LKLB2), A(LXLB2))
296: *      & A(LCXP), A(LKLB1), A(LKLB2), A(LXLB2),
297: *      & NSMICPN, NUC_MAX, H(LSMICPN), A(LMNUCPN), A(LLPIDPN), A(LIZTBNP),
298: *      & A(LDNSTPN), A(LMNEUTPN), NNCSTPN, A(LINCSTPN), NMTMPN, A(LMTMPN),
299: *      & NSTALY, H(LMTCUT), H(LWTCUT), A(LCXP), A(LKLBPN1), A(LKLBPN2),
300: *      & A(LXLBPN1), H(LKPNFG) )
301: C##>
302: C
303: *      call SEANO0(IOW,A,CHA,IDEFER,
304: *      N H(LNLOST),NDEAD,NEVENT,DEPS,NSMAC,NSMIC,MB,NUC,NMKREG,
305: *      B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LKKP),
306: *      B H(LIZZ),H(LIGG),H(LTTT),H(LTTT),H(LKLSF),H(LLEE),H(LXIM),
307: *      B H(LSMAC), H(LSMIC), H(LSGTAL), H(LKMAC), H(LMMAC), H(LKSPI),
308: *      B H(LLEVL), H(LLZZ), H(LLPOS), H(LLCRS), H(LIBREG),H(LIBSPC),
309: *      B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
310: *      S H(LLSFFL), H(LNFFL), H(LIZFFL), H(LLSSRC), H(LNNXT), H(LIZNXT),
311: *      S H(LLSDED), H(LLSREF), H(LISREF), H(LIZREF), H(LNBREF), A(LKSREF),
312: *      S H(LLSLAT), H(LIZLAT), H(LNXLT),
313: *      G A(LSDA), A(LKZMAT), A(LKZREG), A(LKZDA), A(LKZAA), A(LKSFB),
314: *      G A(LKCELL), A(LIPCEL), A(LMLBZZ), A(LISUSP), A(LMKREG),
315: *      T A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
316: *      T H(LNKILD),H(LWKILD),H(LNSURV),
317: *      T H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
318: *      T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
319: *      T H(LDTALY),A(LRESP),
320: *      W H(P2(1)),H(P2(2)),H(P2(3)),H(P2(4)),H(P2(5)),H(P2(6)),
321: C##<2007/03/14:PN3:PN4:
322: C##*      W H(P2(7)),H(P2(8)),H(P2(9)),H(P2(10)),H(P2(11)),H(P2(12)))
323: *      W H(P2(7)),H(P2(8)),H(P2(9)),H(P2(10)),H(P2(11)),H(P2(12)),
324: *      & NUCPN, NSMICPN, H(LSMICPN), H(LKPNFG) )
325: C##>

```

```

326:
327: C*      endif
328: C
329: *      if( JNEUT.ne.0 ) then
330: C
331: *      call NEUTR0( IOW ,NPKIND,NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,
332: *      1 NBANK,NFBANK,NFBNK0,
333: *      5 BANKP,NEVENT, NEST, NUC,NNK,NMT, MB,NSMAC,NSMIC, NSTAL, MCX,
334: *      6 A(LKZMAT), A(LKZREG), A(LXIMP), A(LWKIL),
335: *      6 A(LWSRV),A(LWGTF),A(LWGTFI),A(LWGTP),A(LWTIME),WLLIM,MXPGEN,
336: *      * A(LRESP), ETOP, EBOT, ETHMAX, AMLIM, EWCUT,
337: *      8 A(LENGYB),NNCST,A(LINCST),A(LCX),A(LKLIB1),A(LKLIB2),
338: *      8 A(LKLB2S), A(LKLIB2), H(LSMAC), A(LLMAC), H(LKMAC),
339: *      9 H(LMMAC), H(LSMIC), A(LLMIC), H(LKSPI), H(LSGTAL), IRAND,
340: *      A(NPATOM,NMTP,ETOPP,EBOTP,
341: *      * A(LCXP), MCXP, A(LKLB1),A(LKLB2),A(LXLB2),
342: *      B A(LDNZON), A(LLEMIC), NEMIC, A(LCRES), A(LKRES), MCRES,
343: *      B NMKREG,A(LMKREG) )
344: C##<2007/03/14:PN3:PN4:
345: C##*      B NMKREG,A(LMKREG) )
346: *      B NMKREG,A(LMKREG),
347: *      & NUC_MAX, TCUT, A(LTIMEB), NNCSTPN, A(LINCSTPN), NMTMPN,
348: *      & A(LMTMPN), NSTALY, NUCPN, NUCPNI, NMTPN, NSMICPN, MCXPN,
349: *      & A(LCXP), A(LKLBPN1), A(LKLBPN2), A(LXLBPN1), H(LSMICPN),
350: *      & EBOTLPN, KPNPRD, H(LKPNFG) )
351: C##>
352: C
353: *      endif
354: C
355: *      if( JPHOT.ne.0 ) then
356: *      call PHOTR0
357: *      F ( IOW,IRAND, NPKIND,
358: *      N GROUP,NZONE,NMAT,NREG,NTIME,NRESP,NBANK,BANKP, NEVENT,
359: *      N NEST,NUC,NMTP,NPATOM,MB,NSMAC,NSMIC, NSTAL,
360: *      N NEE, MTOP,
361: *      T A(LKZMAT),A(LKZREG),A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
362: *      T A(LRESP),
363: *      X A(LCXP),A(LCXP),MCXP ,A(LKLB1),A(LXLB1),A(LKLB2),A(LXLB2),
364: *      X H(LSMAC),A(LLMAC),H(LKMAC),H(LMMAC),H(LSMIC),A(LLMIC),
365: *      X H(LKSPI),H(LSGTAL),ETOPP,EBOTP,A(LENGYB),NNCST,A(LINCST),
366: *      X A(LDNZON),
367: *      X A(LXLB1),A(LEEL),A(LPBT),A(LRKT),A(LEBT),A(LEBTP),A(LIEBTP),
368: *      X A(LWWAG),A(LEEDG),A(LEEEK),
369: *      X A(LLEMIC),NEMIC, A(LCRES),A(LKRES),MCRES, NMKREG,A(LMKREG),
370: *      T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
371: C##<2007/03/14:PN3:PN4:
372: C##*      T H(LDTALY) )
373: *      T H(LDTALY),
374: *      & NUCPNA, NUC_MAX, A(LDNSTP), A(LLPDNP), A(LIATMT), A(LNATMT),
375: *      & A(LDNSTPN), A(LMNUCPN), A(LLPIDPN), A(LIZTBNP), CHA(LNCIDPN),
376: *      & A(LMNEUTPN), A(LWGTPN), A(LPNBPR), A(LPFCNR), NPNBPR, NPFCN,
377: *      & A(LPMT), NPMT, MXPGEN, EBOTX, NSTALY, TCUT, A(LTIMEB), A(LCX),
378: *      & MCX, A(LKLIB1), A(LKLIB2), A(LXLIB1), NMT, NUCPN, NUCPNI,
379: *      & NMTPN, NSMICPN, MCXPN, A(LCXP), A(LCXP), A(LKLBPN1),
380: *      & A(LKLBPN2), A(LXLBPN1), A(LXLBPN2), H(LSMICPN), ETOPLPN,
381: *      & EBOTLPN, NNCSTPN, A(LINCSTPN), NMTMPN, A(LMTMPN), KPNPRD,
382: *      & NMOPNPW, A(LMOPNPW), H(LNMPNWT), H(LWMPNWT), H(LKPNFG) )
383: C##>
384: *      endif
385: *
386: C/##ENDIF
387:
388: C
389: C-----
390: C .... CHECK CPU TIMES FOR MONITOR ROUTINE CALL .....

```

src/mvp/action.f

```

391: C-----
392:       if( JVMNT.ne. 0 ) call MNTCPU
393: C
394: C
395: C-----
396: C .... START MONTE CARLO RUN. GENERATE PARTICLES IN BANK .....
397: C       NTGEN: TOTAL NUMBER OF GENERATED PARTICLES IN THIS RUN
398: C       NGENE: GENERATED PARTICLES IN A BATCH
399: C       NTHIST: TOTAL NUMBER OF GENERATED PARTICLES IN SUCCESSIVE RUNS
400: C             (EFFECTIVE IN RESTRT RUN)
401: C-----
402: C
403:       JTERM = 0
404: C
405: C-----
406: C IF NPART < OR = TO NTHIST, NO RANDOM WALK IN THIS RUN !
407: C-----
408: C
409:       if(NPART.le.NTHIST) then
410:         write(IOW,7020) NPART,NTHIST
411: 7020      format(//1X,10('%%%%%%%%%')//
412: &      1X,'  NUMBER OF HISTORIES TO BE RUN (NPART=',I10,') IS ',
413: &      'LESS THAN THAT OF ALREADY RUN (NTHIST=',I10,') !!'//
414: &      1X,'  NO RANDOM WALK IS PERFORMED!'//
415: &      /1X,10('%%%%%%%%%') )
416:         go to 9999
417:       end if
418: C
419: C
420:       NTGEN = NTHIST
421: C
422: C ... NGENE : history to be generated in sub-batch
423: C ... NGBAT : remaining history to be processed in a batch
424: C       (this should be task shared variable or synchronized on all
425: C       tasks in paralell mode)
426: C ... NHIST1: histories of current batch (<= NHIST)
427: C ... NDEAD : unused particle in bank
428: C
429:       NGENE = 0
430:       NGBAT = MIN( NPART - NTGEN, NHIST )
431:       NHIST1 = NGBAT
432:       NGENE0 = 0
433:       NDEAD = NBANK
434:       WSUMB = 0.0D0
435: C
436: C
437: C
438: C if( JALLZ.eq.0 .or. JONEZ.eq.1 ) goto 2222
439: C
440: C
441: C *****
442: C ***** ALL-ZONE FLIGHT & SEARCH *****
443: C *****
444: C
445: C
446: C-----
447: C .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
448: C-----
449: C
450: C1000 continue
451: C
452:       if(JVMNT.ne.0) call VMNTR0(2,IOW)
453: C       call SELALZ( MACT, NZONE, NBANK, NHIST, NTGEN, NPART ,NGENE,
454: C       &      H(LNFFL), H(LNNXT), NCOLS, NDEAD ,H(LNBREF),NREFS,JREFL,
455: C       &      JLATT , H(LNXLTL), NLBZ , NCOLP, JNEUT, JPHOT, JTERM )
456: C       if(JVMNT.ne.0) call VMNTR2(2)

```

```

456: C
457: C-----
458: C MACT = 0 : GENERATE NEXT BATCH OF PARTICLES.
459: C       = 1 : FREE FLIGHT
460: C       = 2 : NEXT ZONE SEARCH
461: C       = 3 : COLLISION
462: C       = 4 : REFLECTION
463: C       = 5 : LATTICE-SEARCH
464: C       = 99 : END OF RUN
465: C-----
466: C
467: C       if(MACT.eq.0) then
468: C-----
469: C ..... TAKE SUMMATION OF TALLY ARRAYS EXCEPT FOR THE FIRST BATCH
470: C       (NBATCH IS INCREMENTED IN THIS ROUTINE.)
471: C-----
472: C       if( NGENE0.gt.0 ) then
473: C         call TALSUM ( IOW, A, H, CHA, NBATCH, NSKIP, NBPINT, JBPRNT,
474: C       3 NGROUP,NGP1,NGP2,NPKIND,NREG,NRESP,NGENE0, NUC, NEMIC, NEMAC,
475: C       4 NSTAL , NFISB , NEVENT, NTREG, A(LIPTRG), A(LLPTRG),
476: C       5 H(LWSUM), H(LWLEK), H(LFLTR ), H(LFLCL ),
477: C       6 H(LSFLTR), H(LSFLCL), A(LRESP), H(LSRETR), H(LSRECL),
478: C       6 H(LXSOC ), H(LXKEF) , H(LNCNTR), H(LWCNTR), H(LWCXTY),
479: C       7 H(LRMIC ), H(LSRMIC), H(LRMICR), H(LXMIC ), H(LDNFLX),
480: C       7 H(LRMAC ), H(LRMACR), H(LXMAC) ,
481: C       8 H(LDMAC) , H(LRSTR), H(LRSCL), H(LSRSTR), H(LSRSCCL),
482: C       8 A(LLEMIC), A(LLEMAC),H(LTFLH), A(LTFLHS),
483: C       T H(LIETAL),NETALY,A(LIDTAL),NDTALY,H(LETALY),NLETAL,
484: C       T H(LDTALY),NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
485: C       W H(P8(1)), H(P8(2)), H(P8(3)), H(P8(4)), H(P8(5)), H(P8(6)),
486: C       W H(P8(7)), H(P8(8)), H(P8(9)), H(P8(10)) )
487: C       end if
488: C-----
489: C ..... OPTIMIZATION OF KMEMO ARRAY .....
490: C-----
491: C       if(NBATCH.le.NMEMOP) then
492: C         call MEMMEM(
493: C       &      IOW, H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
494: C       end if
495: C-----
496: C ..... CHECK CPU TIME
497: C-----
498: C/CC/IF PARA(SX*)
499: C*      call PTCLOCK(DT)
500: C*      T1 = DT
501: C/CC/ELSEIF PARA(CRAY*)
502: C*      call TSECND(T1)
503: C/CC/ELSE
504: C*      call CPUTM(T1)
505: C/CC/ENDIF
506: C       NBT1 = NBATCH+1
507: C
508: C/CC/IF PARA(SX* CRAY*)
509: C*      call MVPSYNC_LOCK(2)
510: C/CC/ENDIF
511: C
512: C       if ( JBPRNT.ne.0 ) then
513: C         write(IOW,7120) IDTASK, NBT1, T1
514: C       end if
515: C
516: C/CC/IF PARA(SX* CRAY*)
517: C*      call MVPSYNC_UNLOCK(2)
518: C/CC/ENDIF
519: C
520: C       if( TCPU.ne.0.0 .and. T1.gt.TCPU*60.0 ) then

```

src/mvp/action.f

```

521: C          write(IOW,7124) NPART,T1,TCPU
522: C          go to 9999
523: C          end if
524: C-----
525: C .... SKIP RANDOM NUMBER IF NECESSARY .....
526: C-----
527: C          if( NRSKIP.lt.0 .or. (NBATCH.eq.0 .and. NRSKIP.gt.0) ) then
528: C              call RSKIP(NRSKIP, IRAND)
529: C          end if
530: C-----
531: C .... GENERATE PARTICLES .....
532: C-----
533: C          if(JVMNT.ne.0) call VMNTR0(1,IOW)
534: CC/#IF ARGSAVE
535: C*          call SOURCE( NGENE , NGENEO ,NBATCH, JBPRT )
536: CC/#ELSE
537: C
538: CC/# IF SOURCE(NEW)
539: C          call SOURCE(IOW,A,H,H,IRAND,NTGEN,NBANK,NDEAD,
540: C          & NGROUP,NGP1,NGP2,NZONE,NFBANK,NFBANK0,NFISB,NEST,JBPRT,
541: C          & H(LNLOST),NEVENT,NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,MCX,MCXP,
542: C          & NUC,NPATOM,NSMIC,NTIME,TCUT,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),
543: C          & H(LBBB),H(LCCC),H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LKLSF),
544: C          & H(LEEE),H(LIBREG),H(LIBSPC),H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),
545: C          & H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),H(LEEEF),H(LINUF),H(LLEVL),
546: C          & H(LLZZ),H(LLPOS),H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),
547: C          & H(LLCRSF),H(LIBRGF),H(LIBSPF),H(LKLSFF),H(LXIM),
548: C          & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK, A(LMLBZZ),
549: C          & A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),A(LKZAA),
550: C          & A(LIPCEL),A(LXIMP),H(LLSFFL),H(LNFFFL),H(LIZFFL),H(LLSDED),
551: C          & H(LLSLAT),H(LIZLAT),H(LNXLT),H(LSRCSP),A(LSOUR),A(LENGYB),
552: C          & A(LENGPB),A(LTIMEB),A(LWGTFF),H(LWSUM),H(LXSOC),
553: C          & H(LNCNTR),H(LWCNTR),EBOT,
554: C          & ETOP,NLENG,H(LXSXV),H(LXAVT),H(LAAVT),H(LEAVT),A(LCX),A(LCX),
555: C          & A(LKLIB1),A(LXLIB1),A(LKLIB2),A(LKLB2S),A(LXLIB2),NNK,NMT,EINCD,
556: C          & A(LCRES),A(LKCRES),H(LSGTAL),MCRES,NSTAL,
557: C          & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LXLB2),NMTP,EBOTP,ETOPP,
558: C          D NPDET, NPLEN, A(LJPUSD),A(LXPDET),A(LIPDET),A(LIPDT2),
559: C          S A(LJSPDT),NIMPT, NDIMPT, H(LIMSFL),H(LIMNFI),
560: C          S H(LIMZFL),H(LIMDED),H(LIMSLT),H(LIMNLT),H(LIMZLT),
561: C          B H(LLZZ),H(LLPOS),H(LLCRSI),H(LOPTI),H(LPATH),H(LKDETF),
562: C          B H(LKLSFI),H(LSMACI),H(LMMACI),
563: C          & A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK,MXREJ,
564: C          & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
565: C          & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
566: C          % NGENE , NGENEO ,NBATCH )
567: CC/# ELSE
568: C*          call SOURCE(IOW,A,H,H,IRAND,NTGEN,NBANK,NDEAD,
569: C          & NGROUP,NGP1,NGP2,NZONE,NFBANK,NFBANK0,NFISB,NEST,JBPRT,
570: C          & H(LNLOST),NEVENT,NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,
571: C          & NMEMS,MCX,MCXP,NUC,NPATOM,NSMIC,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),
572: C          & H(LBBB),H(LCCC),H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LKLSF),
573: C          & H(LEEE),H(LIBREG),H(LIBSPC),H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),
574: C          & H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),H(LEEEF),H(LINUF),H(LLEVL),
575: C          & H(LLZZ),H(LLPOS),H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),
576: C          & H(LLCRSF),H(LIBRGF),H(LIBSPF),H(LKLSFF),H(LXIM),
577: C          & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK, A(LMLBZZ),
578: C          & A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),A(LKZAA),
579: C          & A(LIPCEL),A(LXIMP),H(LLSFFL),H(LNFFFL),H(LIZFFL),H(LLSDED),
580: C          & H(LLSLAT),H(LIZLAT),H(LNXLT),
581: C          * A(LKSOUR),A(LISZON),A(LSOUR),A(LPSPAC),A(LPENRG),A(LKENRG),
582: C          * A(LENGYB),A(LENGPB),A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),
583: C          * A(LNSANG),A(LSANG),A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),
584: C          * A(LWGTFF),H(LWSUM),H(LXSOC),H(LWCNTR),H(LWCNTR),EBOT,
585: C          & ETOP,NLENG,H(LXSXV),H(LXAVT),H(LAAVT),H(LEAVT),A(LCX),A(LCX),

```

```

586: C*          & A(LKLIB1),A(LXLIB1),A(LKLIB2),A(LKLB2S),A(LXLIB2),NNK,NMT,EINCD,
587: C*          & A(LCRES),A(LKCRES),H(LSGTAL),MCRES,NSTAL,
588: C*          & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LXLB2),NMTP,EBOTP,ETOPP,
589: C*          & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
590: C*          & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
591: C*          % NGENE , NGENEO ,NBATCH )
592:
593: CC/# ENDIF
594:
595: CC/#ENDIF
596: C          if(JVMNT.ne.0) call VMNTR2(1)
597: C          NTHIST = NTHIST + NGENE
598: C          JMEM = NBATCH .LT. NMEMOP
599: C-----
600: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT .....
601: C-----
602: C          else if(MACT.eq.1) then
603: C              if(JVMNT.ne.0) call VMNTR0(3,IOW)
604: CC/#IF ARGSAVE
605: C*          call FLIGHT
606: CC/#ELSE
607: C          call FLIGHT( IOW, NBANK,NZONE,NSDA,NZDA,NGROUP,NGP1,NGP2,
608: C          N NREG,NRESP,NCOLS,NDEAD,H(LNLOST),
609: C          N IRAND,DINF,NEST,NLATT,NLBZ,NMAT,MB,NUC,NSMAC,NSMIC,NEMIC,NSTAL,
610: C          B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
611: C          B H(LWWW),H(LIGG),H(LTTT),H(LLEVL),H(LLZZ),H(LLPOS),
612: C          B H(LLCRS),H(LSMAC),A(LLMAC),H(LKMAC),H(LSMIC),A(LLMIC),
613: C          B H(LSGTAL),A(LLPDEN),A(LINUCT),A(LDENST),H(LDNFLX),H(LLSFFL),
614: C          S H(LNFFFL),H(LIZFFL),H(LLSCOL),H(LLSSRC),H(LNNXT),H(LIZNXT),
615: C          G H(LLSDED),A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),
616: C          G A(LNKZAA),A(LDALT),A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),
617: C          G A(LKSLAT),A(LLTYP),A(LMLBZZ),A(LKLATT),A(LCELSZ),
618: C          T A(LRESP),H(LFLTR),H(LRETR),H(LRMIC),H(LRSTR),H(LNLEAK),
619: C          T H(LWLEAK),H(LNCNTR),H(LWCNTR),A(LDNZON),A(LLEMIC),H(LIBREG),
620: C          W H(P6(1)),H(P6(2)),H(P6(3)),
621: C          W H(P6(4)),H(P6(5)),H(P6(6)),H(P6(7)),H(P6(8)),H(P6(9)),
622: C          W H(P6(10)),H(P6(11)),H(P6(12)),H(P6(13)),H(P6(14)),H(P6(15)),
623: C          W H(P6(16)),H(P6(17)),H(P6(18)),H(P6(19)),H(P6(20)),H(P6(21)),
624: C          W H(P6(22)),H(P6(23)),H(P6(24)),H(P6(25)),H(P6(26)),H(P6(27)),
625: C          W H(P6(28)),H(P6(29)),H(P6(30)) )
626: CC/#ENDIF
627: C          if(JVMNT.ne.0) call VMNTR2(3)
628: C-----
629: C .... FIND NEXT ZONE TO ENTER .....
630: C-----
631: C          else if(MACT.eq.2) then
632: C              if(JVMNT.ne.0) call VMNTR0(4,IOW)
633: CC/#IF ARGSAVE
634: C*          call SEARCH( JMEM, NMEMO,H(LKMEMO),H(LMEMC),H(LMEMZ))
635: CC/#ELSE
636: C          call SEARCH( IOW, NBANK,NZONE,NSDA,NZDA,H(LNLOST),
637: C          N NDEAD,NREFS,NGROUP,NGP1,NGP2,NEVENT,NREG,NTIME,
638: C          N IRAND,NLBZ,NEST,DEPS,NSMAC,NSMIC,MB,NSTAL,NUC,NSPACE,NSUZON,
639: C          B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
640: C          B H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LEEE),H(LXIM),H(LKLSF),
641: C          B H(LSMAC),H(LSMIC),H(LSGTAL),H(LKMAC),H(LMMAC),H(LKSPI),
642: C          L H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LIBREG),H(LIBSPC),
643: C          S H(LLSFFL),H(LNFFFL),H(LIZFFL),H(LLSSRC),H(LNNXT),H(LIZNXT),
644: C          S H(LLSDED),H(LSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LKSREF),
645: C          * H(LLSLAT),H(LIZLAT),H(LNXLT),
646: C          G A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LNKZAA),
647: C          * A(LKCELL),A(LIPCEL),A(LMLBZZ),A(LISUSP),
648: C          T A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,H(LNKILD),H(LWKILD),
649: C          T H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
650: C          W H(P7(1)),H(P7(2)),H(P7(3)),H(P7(4)),H(P7(5)),H(P7(6)),

```

src/mvp/action.f

```

651: C      W H(P7(7) ),H(P7(8) ),H(P7(9) ),H(P7(10)),H(P7(11)),H(P7(12)),
652: C      W H(P7(13)),H(P7(14)),H(P7(15)),H(P7(16)),H(P7(17)),H(P7(18)),
653: C      W H(P7(19)),H(P7(20)),H(P7(21)),H(P7(22)),H(P7(23)),H(P7(24)),
654: C      W H(P7(25)),H(P7(26)),H(P7(27)),H(P7(28)),H(P7(29)),H(P7(30)) ,
655: C      %                                JMEM, NMEMO,H(LKMEMO),H(LMEMC),H(LMEMZ))
656: CC/#ENDIF
657: C                                     if(JVMNT.ne.0) call VMNTR2(4)
658: C -----
659: C .... COLLISION .....
660: C -----
661: C      else if(MACT.eq.3) then
662: C                                     if(JVMNT.ne.0) call VMNTR0(5,IOW)
663: C      if ( JPTDT.ne.0 ) then
664: C          call NXTNR( IOW, A, H, HLLSCOL,
665: C      B      H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) ,
666: C      B      H(LBBB) , H(LWWW) , H(LIZZ) , H(LIGG) , H(LTTT) , H(LITT) ,
667: C      B      H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) ,
668: C      B      H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
669: C      B      KDBNK(0) , KIBNK(0) , H(LNCNTR) , H(LWCNTR) ,
670: C      B      H(LNZZI) , H(LLPOSI) , H(LLCRSI) ,
671: C      B      H(LOPTI) , H(LPATH) , H(LKDETP) , H(LKLSFI) , H(LSMACI) , H(LMMACI) ,
672: C      G      A(LKZMAT) , A(LKZREG) , A(LTIMEB) , A(LXPDET) , A(LIPDET) , A(LIPDT2) ,
673: C      S      A(LENGYB) , A(LJPUSD) , H(LIMSUF) , H(LIMNFI) , H(LIMZFI) ,
674: C      S      H(LIMDED) , H(LIMSLT) , H(LINZLT) , H(LIMNLT) ,
675: C      W      H(PC(1)) , H(PC(2)) , H(PC(3)) , H(PC(4)) , H(PC(5)) , H(PC(6)) ,
676: C      W      H(PC(7)) , H(PC(8)) , H(PC(9)) , H(PC(10)) , H(PC(11)) , H(PC(12)) ,
677: C      W      H(PC(13)) , H(PC(14)) , H(PC(15)) , H(PC(16)) , H(PC(17)) , H(PC(18)) ,
678: C      W      H(PC(19)) , H(PC(20)) , H(PC(21)) , H(PC(22)) , H(PC(23)) , H(PC(24)) ,
679: C      W      H(PC(25)) , H(PC(26)) , H(PC(27)) , H(PC(28)) , H(PC(29)) , H(PC(30)) ,
680: C      W      H(PC(31)) , H(PC(32)) , H(PC(33)) , H(PC(34)) , H(PC(35)) , H(PC(36)) ,
681: C      W      H(PC(37)) , H(PC(38)) , H(PC(39)) , H(PC(40)) , H(PC(41)) , H(PC(42)) ,
682: C      W      H(PC(43)) , H(PC(44)) , H(PC(45)) , H(PC(46)) , H(PC(47)) ,
683: C      end if
684: CC/#IF ARGSAVE
685: C*      call NEUTR(
686: C*      2 H(LLSFFL) , H(LNFFL) , H(LIZFFL) , H(LLSCOL) , NCOLS ,
687: C*      3 H(LLSDED) , NDEAD , NFISB , H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) ,
688: C*      4 H(LBBB) , H(LCCC) , H(LWWW) , H(LEEE) , H(LTTT) , H(LITT) , H(LXIM) , H(LKLSF) ,
689: C*      4 H(LIZZ) , H(LIGG) , H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) ,
690: C*      5 H(LXXXF) , H(LYYYF) , H(LZZZF) , H(LLEEF) , H(LINUF) , H(LIZZF) ,
691: C*      6 H(LLEVLF) , H(LLZZF) , H(LLPOSF) , H(LLCRSF) ,
692: C*      6 H(LIBREG) , H(LIBSPC) , H(LIBRGF) , H(LIBSPF) ,
693: C*      B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
694: C*      7 H(LFLCL) , H(LRECL) ,
695: C*      7 H(LNCLSN) , H(LWCLSN) , H(LNABSB) , H(LWABSB) , H(LNECUT) , H(LWECUT) ,
696: C*      7 H(LNKILD) , H(LWKILD) , H(LNSURV) , H(LWSURV) , H(LNSPLT) , H(LWSPLT) ,
697: C*      8 H(LNCNTR) , H(LWCNTR) , H(LRMIC) , H(LDNFLX) , H(LRSLC) ,
698: C*      9 H(P3(1)) , H(P3(2)) , H(P3(3)) , H(P3(4)) , H(P3(5)) , H(P3(6)) ,
699: C*      9 H(P3(7)) , H(P3(8)) , H(P3(9)) , H(P3(10)) , H(P3(11)) , H(P3(12)) ,
700: C*      9 H(P3(13)) , H(P3(14)) , H(P3(15)) , H(P3(16)) , H(P3(17)) , H(P3(18)) ,
701: C*      A H(P3(19)) , H(P3(20)) , H(P3(21)) , H(P3(22)) , H(P3(23)) , H(P3(24)) ,
702: C*      A H(P3(25)) )
703: CC/#ELSE
704: C      call NEUTR( IOW, NGROUP,NGP1,NGP2,NZONE,NMAT,NREG,NTIME,NRESP,
705: C      1 NBANK,NFBANK,NFBNK0,
706: C      5 BANK,NEVENT, NEST, NUC,NNK,NMT, MB,NSMAC,NSMIC, NSTAL, MCX,
707: C      6 A(LKZMAT) , A(LKZREG) , A(LXIMP) , A(LWKIL) ,
708: C      6 A(LWSRV) , A(LWGTFF) , A(LWGTFF) , A(LWGTFF) , A(LWTIME) , WLLIM, MXPGEN,
709: C      7 A(LRESP) , ETOP, EBOT, ETHMAX, AMLIM, EWCUT,
710: C      8 A(LENGYB) , A(LCX) , A(LKLIB1) , A(LXLIB1) , A(LKLIB2) ,
711: C      8 A(LKLB2S) , A(LKLIB2) , A(LSMAC) , A(LLMAC) , H(LKMAC) ,
712: C      9 H(LMMAC) , H(LSMIC) , A(LLMIC) , H(LKSPI) , H(LSGTAL) , IRAND,
713: C      A NPATOM,NMTP,ETOPP, EBOTP, A(LENGPB) ,
714: C      * A(LCXP) , MCXP, A(LKLB1) , A(LKLB2) , A(LKLB2) ,
715: C      B A(LDNZON) , A(LLEMIC) , NEMIC, A(LCRES) , A(LKCRES) , MCRES,

```

```

716: C      % H(LLSFFL) , H(LNFFL) , H(LIZFFL) , H(LLSCOL) , NCOLS ,
717: C      3 H(LLSDED) , NDEAD , NFISB , H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) ,
718: C      4 H(LBBB) , H(LCCC) , H(LWWW) , H(LEEE) , H(LTTT) , H(LITT) , H(LXIM) , H(LKLSF) ,
719: C      4 H(LIZZ) , H(LIGG) , H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) ,
720: C      5 H(LXXXF) , H(LYYYF) , H(LZZZF) , H(LLEEF) , H(LINUF) , H(LIZZF) ,
721: C      6 H(LLEVLF) , H(LLZZF) , H(LLPOSF) , H(LLCRSF) ,
722: C      6 H(LIBREG) , H(LIBSPC) , H(LIBRGF) , H(LIBSPF) ,
723: C      B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK , H(LFLCL) , H(LRECL) ,
724: C      7 H(LNCLSN) , H(LWCLSN) , H(LNABSB) , H(LWABSB) , H(LNECUT) , H(LWECUT) ,
725: C      7 H(LNKILD) , H(LWKILD) , H(LNSURV) , H(LWSURV) , H(LNSPLT) , H(LWSPLT) ,
726: C      8 H(LNCNTR) , H(LWCNTR) , H(LRMIC) , H(LDNFLX) , H(LRSLC) ,
727: C      9 H(P3(1)) , H(P3(2)) , H(P3(3)) , H(P3(4)) , H(P3(5)) , H(P3(6)) ,
728: C      9 H(P3(7)) , H(P3(8)) , H(P3(9)) , H(P3(10)) , H(P3(11)) , H(P3(12)) ,
729: C      9 H(P3(13)) , H(P3(14)) , H(P3(15)) , H(P3(16)) , H(P3(17)) , H(P3(18)) ,
730: C      A H(P3(19)) , H(P3(20)) , H(P3(21)) , H(P3(22)) , H(P3(23)) , H(P3(24)) ,
731: C      A H(P3(25)) )
732: CC/#ENDIF
733: C                                     if(JVMNT.ne.0) call VMNTR2(5)
734: C -----
735: C .... REFLECTION .....
736: C -----
737: C      else if(MACT.eq.4) then
738: C                                     if(JVMNT.ne.0) call VMNTR0(6,IOW)
739: C          call MIRROR( JMNTR, JDEBG, JVMNT, JLAT,
740: C      N NBANK, NREFS, NSDA, IRAND, NLBZ, NZONE,
741: C      B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) ,
742: C      B H(LWWW) , H(LIZZ) ,
743: C      S H(LNCNTR) , H(LWCNTR) , H(LLSFFL) , H(LNFFL) , H(LIZFFL) ,
744: C      S H(LLSSRC) , H(LNNXT) , H(LIZNXT) ,
745: C      S H(LLSREF) , H(LISREF) , H(LIZREF) , H(LNBREF) , A(LSDA) , A(LKKREF) ,
746: C      S H(LLSLAT) , H(LIZLAT) , H(LNXLT) , A(LMLBZZ) ,
747: C      W H(P4(1)) , H(P4(2)) , H(P4(3)) , H(P4(4)) ,
748: C      W H(P4(5)) , H(P4(6)) , H(P4(7)) , H(P4(8)) , H(P4(9)) , H(P4(10)) ,
749: C      W H(P4(11)) , H(P4(12)) , H(P4(13)) , H(P4(14)) , H(P4(15)) , H(P4(16)) ,
750: C      W H(P4(17)) , H(P4(18)) , H(P4(19)) , H(P4(20)) , H(P4(21)) , H(P4(22)) ,
751: C      W H(P4(23)) )
752: C                                     if(JVMNT.ne.0) call VMNTR2(6)
753: C -----
754: C .... LATTICE SEARCH .....
755: C -----
756: C      else if(MACT.eq.5) then
757: C                                     if(JVMNT.ne.0) call VMNTR0(7,IOW)
758: C          call LATTICE(IOW, JDEBG, JVMNT, JTLT, JHLAT, H(LNLOST) ,
759: C      N NBANK, NZONE, NLATT, NCELL, NLBZ, NEST, NSPACE, NUNV, DEPS,
760: C      B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) , H(LIZZ) , H(LLEVL) ,
761: C      B H(LLZZ) , H(LLPOS) , H(LLSDED) , NDEAD , H(LLCRS) , H(LIBREG) , H(LIBSPC) ,
762: C      S H(LLSSRC) , H(LIZNXT) , H(LNNXT) , H(LLSLAT) , H(LIZLAT) , H(LNXLT) ,
763: C      G A(LKZMAT) , A(LKCELL) , A(LKZLBZ) , A(LMLBZZ) , A(LCELSZ) , A(LSZLAT) ,
764: C      G A(LIDLAT) , A(LLPLAT) , A(LKLATT) , A(LKSLAT) , A(LNLVAT) , A(LIPCEL) ,
765: C      G A(LLTYP) , A(LICTYP) , A(LKDALT) , A(LDALT) , A(LKZREG) ,
766: C      G A(LKSPSU) , A(LKTCSP) , NKTCSP , A(LKHLAT) ,
767: C      W H(P5( 1)) , H(P5( 2)) , H(P5( 3)) , H(P5( 4)) , H(P5( 5)) , H(P5( 6)) ,
768: C      W H(P5( 7)) , H(P5( 8)) , H(P5( 9)) , H(P5(10)) )
769: C                                     if(JVMNT.ne.0) call VMNTR2(7)
770: C -----
771: C      else if(MACT.eq.6) then
772: C                                     if(JVMNT.ne.0) call VMNTR0(12,IOW)
773: C          if ( JPTDT.ne.0 ) then
774: C              call NXPHR( IOW, A, H, H(LLSCLP) ,
775: C      B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) ,
776: C      B H(LLEE) , H(LWWW) , H(LIZZ) , H(LIGG) , H(LTTT) , H(LITT) ,
777: C      B H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) ,
778: C      B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
779: C      B KDBNK(0) , KIBNK(0) , H(LNCNTR) , H(LWCNTR) ,
780: C      B H(LLZZI) , H(LLPOSI) , H(LLCRSI) ,

```


src/mvp/action.f

```

781: C      B      H(LOPTI), H(LPATH), H(LKDETP),H(LKLSFI),H(LSMACI),H(LMMACI),
782: C      G      A(LKZMAT),A(LKZREG),A(LTIMEB),A(LXPDET),A(LIPDET),A(LIPDT2),
783: C      S      A(LENGYB),A(LJPUSD),H(LIMSFL),H(LIMNFI),H(LIMZFL),
784: C      S      H(LIMDED),H(LIMSLT),H(LIMZLT),H(LIMNLT),
785: C      W      H(PG(1)),H(PG(2)),H(PG(3)),H(PG(4)),H(PG(5)),H(PG(6)),
786: C      W      H(PG(7)),H(PG(8)),H(PG(9)),H(PG(10)),H(PG(11)),H(PG(12)),
787: C      W      H(PG(13)),H(PG(14)),H(PG(15)),H(PG(16)),H(PG(17)),H(PG(18)),
788: C      W      H(PG(19)),H(PG(20)),H(PG(21)),H(PG(22)),H(PG(23)),H(PG(24)),
789: C      W      H(PG(25)),H(PG(26)),H(PG(27)),H(PG(28)),H(PG(29)),H(PG(30)),
790: C      W      H(PG(31)),H(PG(32)),H(PG(33)),H(PG(34)),H(PG(35)),H(PG(36)),
791: C      W      H(PG(37)),H(PG(38)),H(PG(39)),H(PG(40)),H(PG(41)),H(PG(42)),
792: C      W      H(PG(43)),H(PG(44)),H(PG(45)),H(PG(46)),H(PG(47)) )
793: C      end if
794: C      CC/IF ARGSAVE
795: C*      call PHOTR (
796: C*      S      H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSCLP),NCOLP,H(LLSDED),NDEAD,
797: C*      B      H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
798: C*      B      H(LWWW),H(LEEE),H(LTTT),H(LITT),H(LXIM),H(LKLSF),H(LIZZ),H(LIGG),
799: C*      B      H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LIBREG),H(LIBSPC),
800: C*      B      H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
801: C*      T      H(LFLCL),H(LRECL),H(LNCLSN),H(LWCLSN),H(LNABSB),H(LWABSB),
802: C*      T      H(LNECUT),H(LWECUT),H(LNKILD),H(LWKILD),H(LNSURV),H(LWSURV),
803: C*      T      H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),H(LRMIC),H(LDNFLX),
804: C*      T      H(LRSCL),
805: C*      W      H(PP(1)),H(PP(2)),H(PP(3)),H(PP(4)),H(PP(5)),H(PP(6)),
806: C*      W      H(PP(7)),H(PP(8)),H(PP(9)),H(PP(10)),H(PP(11)),H(PP(12)),
807: C*      W      H(PP(13)),H(PP(14)),H(PP(15)),H(PP(16)),H(PP(17)),H(PP(18)),
808: C*      W      H(PP(19)),H(PP(20)),H(PP(21)),H(PP(22)),H(PP(23)),H(PP(24)),
809: C*      W      H(PP(25)),H(PP(26)),H(PP(27)),H(PP(28)),H(PP(29)) )
810: C      CC/ELSE
811: C      call PHOTR( IOW,IRAND,
812: C      N      NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,NBANK,BANKP, NEVENT,
813: C      N      NGP1, NGP2, NEST,NUC,NMTP,NPATOM,MB,NSMAC,NSMIC, NETAL,
814: C      N      NEE , MTOP,
815: C      T      A(LKZMAT),A(LKZREG),A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
816: C      T      A(LRESP),
817: C      X      A(LCXP),A(LCXP),MCXP ,A(LKLBp1),A(LXLBP1),A(LKLBp2),A(LXLBP2),
818: C      X      H(LSMAC),A(LLMAC),H(LKMAC),H(LMMAC),H(LSMIC),A(LLMIC),
819: C      X      H(LKSPI),H(LSGTAL),ETOPP ,EBOTP ,A(LENGFB),A(LONZON),
820: C      X      A(LXLBE1),A(LEEL),A(LPBT),A(LRKT),A(LEBT),A(LEBTP),A(LIEBTP),
821: C      X      A(LWWAG),A(LEEDG),A(LEEEK),
822: C      X      A(LLEMIC), NEMIC, A(LCRES), A(LKRES), MCRES,
823: C      %      H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSCLP),NCOLP,H(LLSDED),NDEAD,
824: C      B      H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
825: C      B      H(LWWW),H(LEEE),H(LTTT),H(LITT),H(LXIM),H(LKLSF),H(LIZZ),H(LIGG),
826: C      B      H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LIBREG),H(LIBSPC),
827: C      B      H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
828: C      T      H(LFLCL),H(LRECL),H(LNCLSN),H(LWCLSN),H(LNABSB),H(LWABSB),
829: C      T      H(LNECUT),H(LWECUT),H(LNKILD),H(LWKILD),H(LNSURV),H(LWSURV),
830: C      T      H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),H(LRMIC),H(LDNFLX),
831: C      T      H(LRSCL),
832: C      W      H(PP(1)),H(PP(2)),H(PP(3)),H(PP(4)),H(PP(5)),H(PP(6)),
833: C      W      H(PP(7)),H(PP(8)),H(PP(9)),H(PP(10)),H(PP(11)),H(PP(12)),
834: C      W      H(PP(13)),H(PP(14)),H(PP(15)),H(PP(16)),H(PP(17)),H(PP(18)),
835: C      W      H(PP(19)),H(PP(20)),H(PP(21)),H(PP(22)),H(PP(23)),H(PP(24)),
836: C      W      H(PP(25)),H(PP(26)),H(PP(27)),H(PP(28)),H(PP(29)) )
837: C      CC/ENDIF
838: C      if(JVMNT.ne.0) call VMNTR2(12)
839: C
840: C
841: C      else if(MACT.eq.99) then
842: C-----
843: C      .... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR INITIAL BATCH
844: C-----
845: C      call TALSUM( IOW, A, H, CHA,NBATCH, NSKIP, NBPINT, JBRPNT,

```

```

846: C      3 NGROUP,NGP1,NGP2,NPKIND,NREG,NRESP,NGENE0, NUC, NEMIC, NEMAC,
847: C      4 NSTAL, NFISB , NEVENT, NTREG, A(LIPTRG), A(LLPTRG),
848: C      5 H(LWSUM), H(LWLEK) , H(LFLTR ), H(LFLCL ),
849: C      6 H(LSFLTR), H(LSFLCL), A(LRESP), H(LSRETR), H(LSRECL),
850: C      6 H(LXSOC ), H(LXKEF) , H(LNCNTR), H(LWCNTR), H(LWCXTY),
851: C      7 H(LRMIC ), H(LSRMIC), H(LRMICR), H(LXMIC ), H(LDNFLX),
852: C      7 H(LRMAC ), H(LRMACR), H(LXMAC) ,
853: C      8 H(LDMAC), H(LRSTR), H(LRSCL), H(LSRSTR), H(LRSCL),
854: C      8 A(LLEMIC), A(LLEMAC),H(LTFLH), A(LTFLHS),
855: C      T      H(LIETAL),NETALY,A(LIDTAL),NDTALY,H(LETALY),NLETAL,
856: C      T      H(LDTALY),NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
857: C      W      H(P8(1)), H(P8(2)), H(P8(3)), H(P8(4)), H(P8(5)), H(P8(6)),
858: C      W      H(P8(7)), H(P8(8)), H(P8(9)), H(P8(10)) )
859: C-----
860: C      .... OPTIMIZATION OF KMEMO ARRAY ....
861: C-----
862: C      if(NBATCH.le.NMEMOP) then
863: C      call MEMMEM(
864: C      &      IOW, H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
865: C      end if
866: C      goto 9999
867: C      end if
868: C
869: C      goto 1000
870: C
871: C
872: C      ***** ONE-ZONE FLIGHT & SEARCH *****
873: C      *****
874: C
875: C      call TOKEI(TE1,0)
876: C      C/IF PARA(SX*)
877: C      *      call PTCLOCK(DT)
878: C      *      t1 = dt
879: C      C/ELSEIF PARA(CRAY*)
880: C      *      call TSECND(T1)
881: C      C/ELSE
882: C      call CPUTM(T1)
883: C      C/ENDIF
884: C      NBT1 = NBATCH+1
885: C
886: C      C/IF PARA(SX* CRAY*)
887: C      *      call MVPSYNC_LOCK(2)
888: C      C/ENDIF
889: C
890: C      ... print message for starting batch ...
891: C
892: C      write(IOW,7060) IDTASK, NBT1, T1, TE1
893: C      if(JTLST.ne.0) write(IOTL) 'BATCH ', NBT1 ! time-list
894: C
895: C      C/IF PARA(SX* CRAY*)
896: C      *      call MVPSYNC_UNLOCK(2)
897: C      C/ENDIF
898: C
899: C-----
900: C      .... Select next action (stack length check etc.) .....
901: C-----
902: C      2222 continue
903: C      if(JVMNT.ne.0) call VMNTR0(2,IOW)
904: C
905: C      call SELONE( MACT, JTERM, MZONE, NTGEN, NGENE, NGENE0, NGBAT,
906: C      &      H, H(LNFFL),H(LNNXT),H(LNBREF),H(LNXLT),
907: C      &      IMMODE, H(LIMNFI), H(LIMNFX), H(LIMNLT),
908: C      &      H(LNCNTR), H(LWCNTR), NEVENT)
909: C
910: C      if(JVMNT.ne.0) call VMNTR2(2)

```

src/mvp/action.f

```

911: C
912: C .....
913: C   MACT = 0 : Generate next batch of particles (sub-batch)
914: C   = 1 : Free flight
915: C   = 2 : Next zone search
916: C   = 3 : Collision (Neutron)
917: C   = 5 : Reflection
918: C   = 6 : Lattice-search
919: C   = 7 : Collision (Photon)
920: C   = 8 : Next event estimator
921: C   = 88 : End of batch
922: C   = 99 : End of run
923: C   MZONE : Selected zone number or zero
924: C .....
925: C
926: C   if( MACT.eq.0 ) then
927: C
928: C -----
929: C .... SKIP RANDOM NUMBER IF NECESSARY .....
930: C -----
931: C   if ( NGENE0.eq.0 ) then
932: C     if( NRSKIP.lt.0 .or. (NBATCH.eq.0 .and. NRSKIP.gt.0) ) then
933: C       call RSKIP(NRSKIP, IRAND)
934: C     end if
935: C   end if
936: C -----
937: C ..... GENERATE PARTICLES .....
938: C -----
939: C                                     if(JVMNT.ne.0) call VMNTR0(1,IOW)
940: C /# IF ARGSAVE
941:
942: C /# IF SOURCE(NEW)
943:
944: *       call SOURCE( NGENE,NGENE0,NHIST1,NGBAT,H(LIFISB),WFFACT,
945: * &               H(LRNU), WSUMB )
946: C /# ELSE
947:
948: *       call SOURCE( NGENE,NGENE0,NBATCH,WSUMB )
949:
950: C /# ENDIF
951: C /# ELSE
952:
953: C /# IF SOURCE(NEW)
954:
955:       call SOURCE(IOW,A,H,H,
956: N   NTGEN, NDEAD, NFISB, JBPRT, H(LNLOST), NEVENT, NMKREG,
957: X   MCX,MCXP, NUC,NPATOM,NSMIC,
958: & H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LKKF),
959: & H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LKLSF),
960: & H(LLEE),H(LIBREG),H(LIBSPC),H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),
961: & H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),H(LLEEF),H(LINUF),H(LLEVL),
962: & H(LLZZ),H(LLPOS),H(LLCRS),H(LLEVLF),H(LLZZF),H(LLPOSF),
963: & H(LLCRSF),H(LIBRGF),H(LIBSPF),H(LKLSFF),H(LXIM),
964: & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
965: & H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
966: & A(LMLBZZ),
967: & A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),A(LKZAA),
968: & A(LIPCEL),A(LITYP),A(LXIMP),
969: & H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSDDED),
970: & H(LLSLAT),H(LIZLAT),H(LNXLT),H(LSRCSP),A(LSOUR),
971: & A(LENGYB),A(LTIMEB),A(LWGT),H(LWSUM),H(LXSOC),
972: & H(LNCNTR),H(LWCNTR),EBOTX,ETOPX,H(LXSXV),
973: & H(LXAVT),H(LAAVT),H(LEAVT),NNCST,A(LINCST),
974: & A(LCX),A(LCX),
975: & A(LKLIB1),A(LXLIB1),A(LKLIB2),A(LKLB2S),A(LXLIB2),NNK,NMT,EINCD,

```

```

976: & A(LCRES),A(LKCRES),H(LSGTAL),MCRES,
977: & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LXLB2),NMTP,
978: D   NPDET, NPLEN, A(LJPUSD),A(LXPDET),A(LIPDET),A(LIPDET2),
979: S   A(LJSPDT),NIMPT, NDIMPT, H(LIMSFL),H(LIMNFL),
980: S   H(LIMZFL),H(LIMDED),H(LIMSLT),H(LIMNLT),H(LIMZLT),
981: B   H(LLZZI), H(LLPOSI),H(LLCRSI),H(LOPTI), H(LPAT), H(LKDETP),
982: B   H(LKLSFI),H(LSMACI),H(LMMACI),
983: & A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK,MXREJ,
984: & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
985: & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
986: & H(P1(14)),H(P1(15)),NGENE,NGENE0,NHIST1,NGBAT,H(LIFISB),WFFACT,
987: & H(LRNU), WSUMB,
988: & H(P1(16)),H(P1(17)),H(P1(18)),H(P1(19)),
989: & A(LCXP1),H(LSMICP),
990: & NPTDS, H(LWWD), H(LWD0), H(LWCTRP),
991: & H(LWWD2), NPTCS, H(LWWR), H(LWR0),
992: & H(LWWT), H(LWT0),H(LWT0A),H(LWT0B), NUCPT,
993: & NGSP, H(LWSD), NTPT, H(P1(23)), H(LWSC), H(P1(24)), NORDDS,
994: & NOPSDS,
995: & H(LIMTF), H(P1(25)), H(P1(26)),
996: c##<2007/03/14:PN3:PN4:
997: & NUCPN, NUCPNI, NSMICPN, MCXPN, NUC_MAX, H(LSMICPN), NMTPN,
998: & A(LCXPN), A(LCXPN), A(LKLBPN1), A(LKLBPN2), A(LXLBPN1), EBOTLPN,
999: & H(LKPNFG),
1000: c##>
1001: c+beff2
1002: & H(LWLB),H(LWB0),H(LWSB),H(P1(22)),NPTBE,
1003: & H(LWBD),H(LWBD0),H(LWSBD),H(P1(21)),
1004: & H(LWWLD),H(LWLD0),H(LWSLD),H(P1(20)))
1005: c-beff2
1006:
1007: C /# ELSE
1008:
1009: *       call SOURCE(IOW,A,H,H,IRAND,NTGEN,NBANK,NDEAD,
1010: * & NGROUP,NGP1,NGP2,NZONE,NFBANK,NFBNK0,NFISB,NEST,JBPRT,
1011: * & H(LNLOST),NEVENT,NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,MCX,MCXP,
1012: * & NUC,NPATOM,NSMIC,NMKREG,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),
1013: * & H(LBBB),H(LCCC),H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LKLSF),
1014: * & H(LLEE),H(LIBREG),H(LIBSPC),H(LSMIC),A(LLMIC),H(LKSPI),H(LMMAC),
1015: * & H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),H(LLEEF),H(LINUF),H(LLEVL),
1016: * & H(LLZZ),H(LLPOS),H(LLCRS),H(LLEVLF),H(LLZZF),H(LLPOSF),
1017: * & H(LLCRSF),H(LIBRGF),H(LIBSPF),H(LKLSFF),H(LXIM),
1018: * & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK, A(LMLBZZ),
1019: * & A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),A(LKZDA),A(LKZAA),
1020: * & A(LIPCEL),A(LXIMP),H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSDDED),
1021: * & H(LLSLAT),H(LIZLAT),H(LNXLT),H(LSRCSP),
1022: * & A(LKSOUR),A(LISZON),A(LSOUR),A(LPSPAC),A(LPENRG),A(LKENRG),
1023: * & A(LENGYB),A(LENGPB),A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),
1024: * & A(LNSANG),A(LSANG),A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),
1025: * & A(LWGT),H(LWSUM),H(LXSOC),H(LNCNTR),
1026: * & H(LWCNTR),EBOT,ETOP,NLENG,H(LXSXV),H(LXAVT),H(LAAVT),H(LEAVT),
1027: * & A(LCX),A(LCX),
1028: * & A(LKLIB1),A(LXLIB1),A(LKLIB2),A(LKLB2S),A(LXLIB2),NNK,NMT,EINCD,
1029: * & A(LCRES),A(LKCRES),H(LSGTAL),MCRES,NSTAL,
1030: * & A(LCXP),A(LCXP),A(LKLB1),A(LKLB2),A(LXLB2),NMTP,EBOT,ETOP,
1031: * & H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
1032: * & H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),H(P1(13)),
1033: * & H(P1(14)),H(P1(15)),NGENE, NGENE0, NBATCH, WSUMB )
1034:
1035: C /# ENDIF
1036:
1037: C /# ENDIF
1038:
1039: NTHIST = NTHIST + NGENE
1040: JMEMP = NBATCH .LT. NMEMP

```

src/mvp/action.f

```

1041: C
1042: C-----
1043: C << Free flight >>
1044: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT .....
1045: C-----
1046: C
1047:       else if( MACT.eq.1 ) then
1048:
1049:       if(JVMNT.ne.0) call VMNTR0(3,IOW)
1050: C/IF ARGSAVE
1051: *       call FLIONE( MZONE,
1052: *       W H(P0(1) ),H(P0(2) ),H(P0(3) ),H(P0(4) ),H(P0(5) ),H(P0(6) ),
1053: *       W H(P0(7) ),H(P0(8) ),H(P0(9) ),H(P0(10) ),H(P0(11) ),H(P0(12) ),
1054: *       W H(P0(13) ),H(P0(14) ),H(P0(15) ),H(P0(16) ),H(P0(17) ),H(P0(18) ),
1055: *       W H(P0(19) ),H(P0(20) ),H(P0(21) ),H(P0(22) ),H(P0(23) ),H(P0(24) ),
1056: *       W H(P0(25) ),H(P0(26) ),H(P0(27) ),H(P0(28) ),H(P0(29) ),H(P0(30) ),
1057: *       W H(P0(31) ),H(P0(32) ),H(P0(33) ),H(P0(34) ),H(P0(35) ),H(P0(36) ),
1058: *       W H(P0(37) ),H(P0(38) ),H(P0(39) ),H(P0(43) ),H(P0(44) ),
1059: c##<2007/03/14:PN3:
1060: *       & H(LRNU) )
1061: c##>
1062: C/#ELSE
1063:       call FLIONE(IOW,NDEAD,NCOLS,NCOLP,H(LNLOST),
1064: N NKTCSF,MB,NSMAC,NSMIC,NEMIC,NEVENT,
1065: N H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),
1066: B H(LKKP),H(LIZZ),H(LLEE),H(LIGG),H(LITT),H(LLTT),H(LLEVL),H(LLZZ),
1067: B H(LLPOS),H(LLCRS),H(LKLSF),H(LIBSPC),H(LIBREG),H(LSMAC),A(LLMAC),
1068: B H(LKMAC),H(LMMAC),H(LSMIC),A(LSGTAL),H(LKSRI),
1069: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1070: M A(LLPDNP),A(LIATMT),A(LDNSTP),H(LLSFFL),H(LNFFL),H(LIZFFL),
1071: S H(LLSCOL),H(LLSCLP),H(LLSSRC),H(LNNXT),H(LIZNXT),H(LLSDED),
1072: G A(LSDA),A(LKZMAT),A(LKZREG),A(LKSPSU),A(LKTCSP),
1073: G A(LKZDA),A(LKZAA),A(LDALT),A(LKDALT),
1074: G A(LNVLAT),A(LSZLAT),A(LIPLAT),A(LIPCEL),A(LKSLAT),A(LLTYP),
1075: G A(LMLBZZ),A(LKZLBZ),A(LKCELL),A(LICTYP),A(LKLATT),A(LCELSZ),
1076: G A(LPNND),A(LKNND),A(LPPPF),A(LKPPF),
1077: T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
1078: T A(LPSALP),A(LPSXYZ),H(LDTALY),A(LRESP),H(LFLTR),H(LRETR),
1079: T H(LRMIC),H(LRSTR),H(LTFLH),H(LWCNTR),H(LWCNTR),
1080: T A(LDNZON),A(LLEMIC),A(LTIMEB),NMKREG,A(LMKREG),
1081: T NTSRF,A(LKTSRF),A(LITSRF),A(LIDSRF),A(LANGLB),NANGLE,
1082: T A(LENGYB),H(LDNFLXSM),H(LRMICSM),
1083: & A(LCX),A(LKLIB1),A(LKLIB2),A(LXLIB1),A(LXLIB2),
1084: & A(LCXP),A(LKLB1),A(LKLB2),A(LKLB2),
1085: W H(P0(1) ),H(P0(2) ),H(P0(3) ),H(P0(4) ),H(P0(5) ),H(P0(6) ),
1086: W H(P0(7) ),H(P0(8) ),H(P0(9) ),H(P0(10) ),H(P0(11) ),H(P0(12) ),
1087: W H(P0(13) ),H(P0(14) ),H(P0(15) ),H(P0(16) ),H(P0(17) ),H(P0(18) ),
1088: W H(P0(19) ),H(P0(20) ),H(P0(21) ),H(P0(22) ),H(P0(23) ),H(P0(24) ),
1089: W H(P0(25) ),H(P0(26) ),H(P0(27) ),H(P0(28) ),H(P0(29) ),H(P0(30) ),
1090: W H(P0(31) ),H(P0(32) ),H(P0(33) ),H(P0(34) ),H(P0(35) ),H(P0(36) ),
1091: W H(P0(37) ),H(P0(38) ),H(P0(39) ),H(P0(43) ),H(P0(44) ),
1092: c & MZONE )
1093: & MZONE,
1094: & H(P0(40) ),H(P0(41) ),H(P0(42) ),
1095: & H(LSMICP),H(LSMACP),
1096: & NPTDS,H(LWWD),A(LJPTTR),A(LJPTNU),NPTCS,H(LWWR),MAXDA,A(LDELA),
1097: & H(LWWT),NUCPT,A(LMNUC),NGSP,H(LWSC),NORDD,
1098: c+beff1
1099: & H(LRNU),NEBEF,H(LWCBEF),
1100: c-beff1
1101: c##<2007/03/14:PN3:PN4:
1102: & NSMICPN,NUC_MAX,H(LSMICPN),A(LNATMT),A(LMNUCPN),A(LLPDNP),
1103: & A(LIZTBN),A(LDNSTPN),A(LMNEUTPN),NNCSTPN,A(LINCSTPN),
1104: & NMTMPN,A(LMTMPN),NSTALY,H(LNCTUT),H(LWTCUT),A(LCXPN),
1105: & A(LKLBPN1),A(LKLBPN2),A(LXLBPN1),H(LKPNFG),

```

```

1106: c##>
1107: & NPTBE,H(LWWLD)) ! lambda
1108: C/#ENDIF
1109:
1110:       if(JVMNT.ne.0) call VMNTR2(3)
1111: C-----
1112: C << Next zone search >>
1113: C .... FIND NEXT ZONE TO ENTER .....
1114: C-----
1115:       else if( MACT.eq.2 ) then
1116:       if(JVMNT.ne.0) call VMNTR0(4,IOW)
1117: C/IF ARGSAVE
1118: *       call SEAONE( MZONE, JMEM, H(LKMEMO),H(LMEMC),H(LMEMZ))
1119: C/#ELSE
1120:       call SEAONE(IOW,A,CHA,IDEFER,
1121: N H(LNLOST),NDEAD,NEVENT,DEPS,NSMAC,NSMIC,MB,NUC,NMKREG,
1122: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LKKP),
1123: B H(LIZZ),H(LIGG),H(LITT),H(LITT),H(LKLSF),H(LLEE),H(LXIM),
1124: B H(LSMAC),H(LSMIC),H(LSGTAL),H(LKMAC),H(LMMAC),H(LKSPI),
1125: B H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LIBREG),H(LIBSPC),
1126: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1127: S H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSSRC),H(LNNXT),H(LIZNXT),
1128: S H(LLSDED),H(LLSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LKSREF),
1129: S H(LLSLAT),H(LIZLAT),H(LNXL),
1130: G A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LKSFB),
1131: G A(LKCELL),A(LIPCEL),A(LMLBZZ),A(LISUSP),A(LMKREG),
1132: T A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
1133: T H(LNKILD),H(LWKILD),H(LNSURV),
1134: T H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
1135: T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
1136: T H(LDTALY),A(LRESP),
1137: W H(P2(1) ),H(P2(2) ),H(P2(3) ),H(P2(4) ),H(P2(5) ),H(P2(6) ),
1138: W H(P2(7) ),H(P2(8) ),H(P2(9) ),H(P2(10) ),H(P2(11) ),H(P2(12) ),
1139: % MZONE, JMEM, H(LKMEMO),H(LMEMC),H(LMEMZ),
1140: & NUCPN,NSMICPN,H(LSMICPN),H(LKPNFG), ! photonuc
1141: & H(LSMICP),H(LSMACP),NPTDS,NPTCS,NGSP,H(LWWD),H(LWWD2), ! pert
1142: & H(LWWR),H(LWSD),H(LWSC),NORDD,NOPSD, ! pert
1143: & H(LWWD),H(LWSB),NPTBE,H(LWWBD),H(LWSBD),H(LWWLD),H(LWSLD), ! beff
1144: & H(P2(13) ),IOTL) ! time-list
1145: C/#ENDIF
1146:
1147:       if(JVMNT.ne.0) call VMNTR2(4)
1148: C-----
1149: C .... COLLISION (NEUTRON) .....
1150: C-----
1151: C
1152:       else if( MACT.eq.3 ) then
1153:       if(JVMNT.ne.0) call VMNTR0(5,IOW)
1154:       call NXTNR( IOW, A, H, H(LLSCOL),
1155: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
1156: B H(LLEE),H(LWWW),H(LIZZ),H(LIGG),H(LITT),H(LITT),
1157: B H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),
1158: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1159: B KDBNK(0),KIBNK(0),H(LNCNTR),H(LWCNTR),
1160: B H(LLZZI),H(LLPOSI),H(LLCRSI),
1161: B H(LOPTI),H(LPATH),H(LKDETP),H(LKLSFI),H(LSMACI),H(LMMACI),
1162: G A(LKZMAT),A(LKZREG),A(LTIMEB),A(LXPDET),A(LIPDET),A(LIPDET2),
1163: S A(LENGYB),A(LJPUSD),H(LIMSFL),H(LIMNFI),H(LIMZFL),
1164: S H(LIMDED),H(LIMSLT),H(LIMZLT),H(LIMNLT),
1165: W H(PC(1) ),H(PC(2) ),H(PC(3) ),H(PC(4) ),H(PC(5) ),H(PC(6) ),
1166: W H(PC(7) ),H(PC(8) ),H(PC(9) ),H(PC(10) ),H(PC(11) ),H(PC(12) ),
1167: W H(PC(13) ),H(PC(14) ),H(PC(15) ),H(PC(16) ),H(PC(17) ),H(PC(18) ),
1168: W H(PC(19) ),H(PC(20) ),H(PC(21) ),H(PC(22) ),H(PC(23) ),H(PC(24) ),
1169: W H(PC(25) ),H(PC(26) ),H(PC(27) ),H(PC(28) ),H(PC(29) ),H(PC(30) ),
1170: W H(PC(31) ),H(PC(32) ),H(PC(33) ),H(PC(34) ),H(PC(35) ),H(PC(36) ),

```

src/mvp/action.f

```

1171:      W      H(PC(37)),H(PC(38)),H(PC(39)),H(PC(40)),H(PC(41)),H(PC(42)),
1172:      W      H(PC(43)),H(PC(44)),H(PC(45)),H(PC(46)),H(PC(47)),
1173:      W      H(PC(48)) ) ! photonuc
1174:      end if
1175: C/#IF ARGSAVE
1176: *      call NEUTR(
1177: *      2 H(LLSFFL), H(LNFFL), H(LIZFFL), H(LLSCOL), NCOLS,
1178: *      3 H(LLSDED), NDEAD, NFISB,H(LXXX), H(LYYY), H(LZZZ), H(LAAA),
1179: *      4 H(LBBB),H(LCCC),H(LWWW),H(LEEE),H(LTTT),H(LITT),H(LXIM),H(LKLSF),
1180: *      4 H(LKKP),H(LIZZ), H(LIGG), H(LLEVL),H(LLZZ), H(LLPOS),H(LLCRS),
1181: *      5 H(LXXXF),H(LYYYF),H(LZZZF),H(LLEEF),H(LINUF),H(LIZZF),
1182: *      6 H(LLVLF),H(LLZZF),H(LLPOSF),H(LLCRSF),
1183: *      6 H(LIBREG),H(LIBSPC),H(LIBRGF),H(LIBSPF),
1184: *      B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1185: *      B H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
1186: *      7 H(LFLCL),H(LRECL),
1187: *      7 H(LNCLSN),H(LWCLSN),H(LNABSB),H(LWABSB),H(LNECUT),H(LWECUT),
1188: *      7 H(LNKILD),H(LWKILD),H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),
1189: *      8 H(LNCNTR),H(LWCNTR),H(LRMIC),H(LDNFLX),H(LRSCL),
1190: *      & KYPOS, H(LDNFLXSM),H(LRMICSM),H(LRMIMU),H(LWMIMU),
1191: *      T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
1192: *      T H(LDTALY),H(LRNU),
1193: *      9 H(P3(1)), H(P3(2)), H(P3(3)), H(P3(4)), H(P3(5)), H(P3(6)),
1194: *      9 H(P3(7)), H(P3(8)), H(P3(9)), H(P3(10)), H(P3(11)), H(P3(12)),
1195: *      9 H(P3(13)),H(P3(14)),H(P3(15)),H(P3(16)), H(P3(17)), H(P3(18)),
1196: *      A H(P3(19)),H(P3(20)),H(P3(21)),H(P3(22)), H(P3(23)), H(P3(24)),
1197: c##<2007/03/14:PN4:
1198: c##*      A H(P3(25)),H(P3(26)) )
1199: *      A H(P3(25)),H(P3(26)),H(P3(34)) )
1200: c##>
1201: C/#ELSE
1202:      call NEUTR( IOW ,NPKIND,NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,
1203: 1 NBANK,NFBANK,NFBNK0,
1204: 5 BANKP,NEVENT,NEST,NUC,NNK,NMT,MB,NSMAC,NSMIC,NSTAL,MCX,
1205: 6 A(LKZMAT),A(LKZREG),A(LXIMP),A(LWKIL),A(LWSRV),
1206: 6 A(LWGTFF),A(LWGTFF),A(LWGTFF),A(LWTIME),WLLIM,MXPGEN,
1207: 7 A(LRESP),ETOP,EBOT,ETHMAX,AMLIM,EWCUT,
1208: 8 A(LENGVB),NNCST,A(LINCST),A(LCX),A(LKLIB1),A(LKLIB1),A(LKLIB2),
1209: 8 A(LKLIB2S),A(LXLIB2),H(LSMAC),A(LLMAC),H(LKMAC),
1210: 9 H(LMMAC),H(LSMIC),A(LLMIC),H(LKSPI),H(LSGTAL),IRAND,
1211: A NPATOM,NMTP,ETOPP,EBOTP,
1212: * A(LCX),MCXP,A(LKLB1),A(LKLB2),A(LKLB2),A(LDNZON),
1213: * A(LLEMIC),NEMIC,A(LCRES),A(LKRES),MCRES,NMKREG,A(LMKREG),
1214: % H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSCOL),NCOLS,
1215: 3 H(LLSDED),NDEAD,NFISB,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),
1216: 4 H(LBBB),H(LCCC),H(LWWW),H(LEEE),H(LTTT),H(LITT),H(LXIM),H(LKLSF),
1217: 4 H(LKKP),H(LIZZ),H(LIGG),H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),
1218: 5 H(LXXXF),H(LYYYF),H(LZZZF),H(LLEEF),H(LINUF),H(LIZZF),
1219: 6 H(LLVLF),H(LLZZF),H(LLPOSF),H(LLCRSF),
1220: 6 H(LIBREG),H(LIBSPC),H(LIBRGF),H(LIBSPF),
1221: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1222: B H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
1223: 7 H(LFLCL),H(LRECL),
1224: 7 H(LNCLSN),H(LWCLSN),H(LNABSB),H(LWABSB),H(LNECUT),H(LWECUT),
1225: 7 H(LNKILD),H(LWKILD),H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),
1226: 8 H(LNCNTR),H(LWCNTR),H(LRMIC),H(LDNFLX),H(LRSCL),
1227: & KYPOS, H(LDNFLXSM),H(LRMICSM),H(LRMIMU),H(LWMIMU), ! scat-mtx
1228: T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
1229: T H(LDTALY),H(LRNU),
1230: 9 H(P3(1)), H(P3(2)), H(P3(3)), H(P3(4)), H(P3(5)), H(P3(6)),
1231: 9 H(P3(7)), H(P3(8)), H(P3(9)), H(P3(10)), H(P3(11)), H(P3(12)),
1232: 9 H(P3(13)),H(P3(14)),H(P3(15)),H(P3(16)), H(P3(17)), H(P3(18)),
1233: A H(P3(19)),H(P3(20)),H(P3(21)),H(P3(22)), H(P3(23)), H(P3(24)),
1234: A H(P3(25)),H(P3(26)),
1235: & H(P3(27)),H(P3(28)),H(P3(29)),H(P3(30)), ! pert

```

```

1236: & A(LCXPL),H(LSMICP),H(LSMACP), ! pert
1237: & NPTDS, H(LWWD), H(LWD0), H(LWCTRP), H(LWD0A), H(LWD0B), ! pert
1238: & NBATCH, NSKIP, A(LJPTTR), A(LJPTNU), H(LWWD2), NPTCS, H(LWWR), ! pert
1239: & H(LWR0), H(LWR0A), MAXDA, A(LDELA), ! pert
1240: & H(LWWT),H(LWT0),H(LWT0A),H(LWT0B),NUCPT, ! pert
1241: & A(LMNUC), A(LLPDEN), A(LINUCT), A(LDENST), H(P3(31)), ! pert
1242: & H(P3(32)), H(P3(33)), ! pert
1243: & NGSP, H(LWSD), H(LWSC), NOCS, NTPT, NORDD, NOPS, ! pert
1244: & H(LIMTF), NEBEF, H(LWCBEF), H(P3(35)), H(P3(36)), H(P3(37)), ! beff
1245: & NUC_MAX, TCUT, A(LTIMEB), NNCSTPN, A(LINCSTPN), NMTPN, ! photo-nuc
1246: & A(LMTMPN), NSTALY, NUCPN, NUCPN1, NMTPN, NSMICPN, MCXPN, ! photo-nuc
1247: & A(LCXPN), A(LKLBPN1), A(LKLBPN2), A(LXLBPN1), H(LSMICPN), ! photo-nuc
1248: & EBOTLPN, KPNPRD, H(LKPNFG), H(P3(34)), ! photo-nuc
1249: & H(LWWB), H(LWB0), H(LWSB), NPTBE, ! beff
1250: & H(LWWD), H(LWB0), H(LWSBD), H(LWWD), H(LWLD0), H(LWLD), ! beff
1251: & IOTL) ! time-list
1252: C/#ENDIF
1253: if(JVMNT.ne.0) call VMNTR2(5)
1254: C-----
1255: C << reflective boundary condition >>
1256: C ... REFLECTION .....
1257: C-----
1258: else if( MACT.eq.5 ) then
1259: if(JVMNT.ne.0) call VMNTR0(6,IOW)
1260: call MIRROR( JMNTR, JDEBG, JVMNT, JLATT,
1261: N NBANK, NREFS, NSDA, IRAND, NLBZ, NZONE,
1262: B H(LXXX), H(LYYY), H(LZZZ), H(LAAA), H(LBBB), H(LCCC),
1263: B H(LWWW), H(LIZZ), H(LIBNK),KIBNK,MIBNK,
1264: S H(LNCNTR), H(LWCNTR), H(LLSFFL), H(LNFFL), H(LIZFFL),
1265: S H(LLSSRC), H(LNNXT), H(LIZNXT),
1266: S H(LLSREF), H(LISREF), H(LIZREF), H(LNBREF), A(LSDA), A(LKKREF),
1267: S H(LLSLAT), H(LIZLAT), H(LNXLT), A(LMLBZZ),
1268: W H(P4(1)), H(P4(2)), H(P4(3)), H(P4(4)),
1269: W H(P4(5)), H(P4(6)), H(P4(7)), H(P4(8)), H(P4(9)), H(P4(10)),
1270: W H(P4(11)), H(P4(12)), H(P4(13)), H(P4(14)), H(P4(15)), H(P4(16)),
1271: W H(P4(17)), H(P4(18)), H(P4(19)), H(P4(20)), H(P4(21)), H(P4(22)),
1272: W H(P4(23)) )
1273: if(JVMNT.ne.0) call VMNTR2(6)
1274: C-----
1275: C << lattice geometry handling >>
1276: C ... LATTICE SEARCH .....
1277: C-----
1278: else if( MACT.eq.6 ) then
1279: if(JVMNT.ne.0) call VMNTR0(7,IOW)
1280: call LATTICE( JDEBG, JVMNT, JTLT, JHLAT, JFISX, H(LNLOST),
1281: N NBANK, NZONE, NREG, NLATT, NCELL, NLBZ, NEST, NSUZON, NSPACE, NUNV,
1282: N DINF, DEPS, IRAND, NMKREG,
1283: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LIZZ),
1284: B H(LLEVL),H(LLZZ),H(LLPOS),H(LLSDED),NDEAD,H(LLCRS),
1285: B H(LKLSF), H(LIBREG),H(LIBSPC),
1286: B H(LDBNK), KDBNK,MDBNK,H(LIBNK), KIBNK, MIBNK,
1287: S H(LLSSRC), H(LIZNXT), H(LNNXT), H(LLSLAT), H(LIZLAT), H(LNXLT),
1288: S H(LLSFFL), H(LNFFL), H(LIZFFL),
1289: G A(LSDA), A(LKZDA), A(LKZAA),
1290: G A(LKZMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
1291: G A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
1292: G A(LLTYP), A(LICTYP), A(LKDALT), A(LDALT), A(LKZREG),
1293: G A(LISUSP),A(LKSPSU), A(LKTCSP), NKTCSP, A(LKHLAT),A(LMKREG),
1294: G A(LPPPF), A(LKPPF), NPPPF,
1295: W H(P5(1)), H(P5(2)), H(P5(3)), H(P5(4)), H(P5(5)), H(P5(6)),
1296: W H(P5(7)), H(P5(8)), H(P5(9)), H(P5(10)),
1297: W H(P5(11)), H(P5(12)), H(P5(13)), H(P5(14)),H(P5(15)), H(P5(16)))
1298: if(JVMNT.ne.0) call VMNTR2(7)
1299: C-----
1300: C ... COLLISION (PHOTON) .....

```

src/mvp/action.f

```

1301: C-----
1302:     else if( MACT.eq.7 ) then
1303:                                     if(JVMNT.ne.0) call VMNTR0(12,IOW)
1304:     if ( JPTDT.ne.0 ) then
1305:         call NXPHR( IOW, A, H, H(LLSCLP),
1306:             B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) ,
1307:             B H(LEEE) , H(LWWW) , H(LIZZ) , H(LIGG) , H(LTTT) , H(LITT) ,
1308:             B H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) ,
1309:             B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
1310:             B KDBNK(0) , KIBNK(0) , H(LNCNTR) , H(LWCNTR) ,
1311:             B H(LLZZI) , H(LLPOSI) , H(LLCRSI) ,
1312:             B H(LOPTI) , H(LPATH) , H(LKDETP) , H(LKLSFI) , H(LSMACI) , H(LMMACI) ,
1313:             C A(LKZMAT) , A(LKZREG) , A(LTIMEB) , A(LXPDET) , A(LIPDET) , A(LIPDT2) ,
1314:             S A(LENGYB) , A(LJPUSF) , H(LIMSFL) , H(LIMNFI) , H(LIMZFL) ,
1315:             S H(LIMDED) , H(LIMSLT) , H(LIMNLT) , H(LIMZLT) ,
1316:             W H(PG(1)) , H(PG(2)) , H(PG(3)) , H(PG(4)) , H(PG(5)) , H(PG(6)) ,
1317:             W H(PG(7)) , H(PG(8)) , H(PG(9)) , H(PG(10)) , H(PG(11)) , H(PG(12)) ,
1318:             W H(PG(13)) , H(PG(14)) , H(PG(15)) , H(PG(16)) , H(PG(17)) , H(PG(18)) ,
1319:             W H(PG(19)) , H(PG(20)) , H(PG(21)) , H(PG(22)) , H(PG(23)) , H(PG(24)) ,
1320:             W H(PG(25)) , H(PG(26)) , H(PG(27)) , H(PG(28)) , H(PG(29)) , H(PG(30)) ,
1321:             W H(PG(31)) , H(PG(32)) , H(PG(33)) , H(PG(34)) , H(PG(35)) , H(PG(36)) ,
1322:             W H(PG(37)) , H(PG(38)) , H(PG(39)) , H(PG(40)) , H(PG(41)) , H(PG(42)) ,
1323:             W H(PG(43)) , H(PG(44)) , H(PG(45)) , H(PG(46)) , H(PG(47)) )
1324:     end if
1325: C/#IF ARGSAVE
1326: *      call PHOTR (
1327: *      S H(LLSFFL) , H(LNFFL) , H(LNZFFL) , H(LLSCLP) , NCOLP , H(LLSDED) , NDEAD ,
1328: *      B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) , H(LWWW) , H(LEEE) ,
1329: *      B H(LTTT) , H(LITT) , H(LXIM) , H(LKLSF) , H(LKKP) , H(LIZZ) , H(LIGG) ,
1330: *      B H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) , H(LIBREG) , H(LIBSPC) ,
1331: *      B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
1332: *      T H(LFLCL) , H(LRECL) , H(LNCLSN) , H(LWCLSN) , H(LNABSB) , H(LWABSB) ,
1333: *      T H(LNECUT) , H(LWECUT) , H(LNKILD) , H(LWKILD) , H(LNSURV) , H(LWSURV) ,
1334: *      T H(LNSPLT) , H(LWSPLT) , H(LNCNTR) , H(LWCNTR) , H(LRMIC) , H(LDNFLX) ,
1335: *      T H(LRSCL) ,
1336: *      W H(PP(1)) , H(PP(2)) , H(PP(3)) , H(PP(4)) , H(PP(5)) , H(PP(6)) ,
1337: *      W H(PP(7)) , H(PP(8)) , H(PP(9)) , H(PP(10)) , H(PP(11)) , H(PP(12)) ,
1338: *      W H(PP(13)) , H(PP(14)) , H(PP(15)) , H(PP(16)) , H(PP(17)) , H(PP(18)) ,
1339: *      W H(PP(19)) , H(PP(20)) , H(PP(21)) , H(PP(22)) , H(PP(23)) , H(PP(24)) ,
1340: *      W H(PP(25)) , H(PP(26)) , H(PP(27)) , H(PP(28)) , H(PP(29)) ,
1341: *      & H(LSMICP) , H(LSMACP) , NPTDS , NPTCS , NGSP , H(LWWD) , H(LWWD2) ,
1342: *      & H(LWWR) , H(LWSD) , H(LWSC) , NORDDS , NOPSDDS ,
1343: c##<2007/03/14:PN3:PN4:
1344: *      & H(PP(30)) , H(PP(31)) , H(LRNU) , A , H ,
1345: *      & A(LXPDET) , A(LIPDET) , A(LIPDT2) , H(LIMSFL) , H(LIMNFI) ,
1346: *      & H(LIMZFL) , H(LIMDED) , H(LIMSLT) , H(LIMNLT) , H(LIMZLT) ,
1347: *      & H(LKZZI) , H(LLPOSI) , H(LLCRSI) , H(LOPTI) , H(LPATH) ,
1348: *      & H(LKDETP) , H(LKLSFI) , H(LSMACI) , H(LMMACI) , H(PP(46)) )
1349: c##>
1350: C/#ELSE
1351:     call PHOTR( IOW,IRAND, NPKIND,
1352:     N NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,NBANK,BANKP, NEVENT,
1353:     N NEST,NUC,NMTP,NPATOM,MB,NSMAC,NSMIC, NSTAL,
1354:     N NEE , MTOP,
1355:     T A(LKZMAT) , A(LKZREG) , A(LXIMP) , A(LWKIL) , A(LWSRV) , A(LWTIME) , WLLIM,
1356:     T A(LRESP) ,
1357:     X A(LCXPC) , A(LCXPC) , MCXP , A(LKLBp1) , A(LXLBp1) , A(LKLBp2) , A(LXLBp2) ,
1358:     X H(LSMAC) , A(LLMAC) , H(LKMAC) , H(LMMAC) , H(LSMIC) , A(LLMIC) ,
1359:     X H(LKSPI) , H(LSGTAL) , ETOPP , EBOTP , A(LENGYB) , NNCST , A(LINCST) ,
1360:     X A(LDNZON) ,
1361:     X A(LXLB1) , A(LEEL) , A(LPBT) , A(LRKT) , A(LEBT) , A(LEBTP) , A(LIEBTP) ,
1362:     X A(LWWAG) , A(LEEDG) , A(LEEEK) ,
1363:     X A(LLEMIC) , NEMIC , A(LCRES) , A(LKCRS) , MCRES , NMKREG , A(LMKREG) ,
1364:     % H(LLSFFL) , H(LNFFL) , H(LIZFFL) , H(LLSCLP) , NCOLP , H(LLSDED) , NDEAD ,
1365:     B H(LXXX) , H(LYYY) , H(LZZZ) , H(LAAA) , H(LBBB) , H(LCCC) , H(LWWW) , H(LEEE) ,

```

```

1366:     B H(LTTT) , H(LITT) , H(LXIM) , H(LKLSF) , H(LKKP) , H(LIZZ) , H(LIGG) ,
1367:     B H(LLEVL) , H(LLZZ) , H(LLPOS) , H(LLCRS) , H(LIBREG) , H(LIBSPC) ,
1368:     B H(LDBNK) , KDBNK , MDBNK , H(LIBNK) , KIBNK , MIBNK ,
1369:     T H(LFLCL) , H(LRECL) , H(LNCLSN) , H(LWCLSN) , H(LNABSB) , H(LWABSB) ,
1370:     T H(LNECUT) , H(LWECUT) , H(LNKILD) , H(LWKILD) , H(LNSURV) , H(LWSURV) ,
1371:     T H(LNSPLT) , H(LWSPLT) , H(LNCNTR) , H(LWCNTR) , H(LRMIC) , H(LDNFLX) ,
1372:     T H(LRSCL) ,
1373:     T A(LJDTRG) , A(LIDTAL) , A(LJTEVE) , A(LLTEVE) , A(LKTEVE) , NTEVE ,
1374:     H(LDTALY) ,
1375:     W H(PP(1)) , H(PP(2)) , H(PP(3)) , H(PP(4)) , H(PP(5)) , H(PP(6)) ,
1376:     W H(PP(7)) , H(PP(8)) , H(PP(9)) , H(PP(10)) , H(PP(11)) , H(PP(12)) ,
1377:     W H(PP(13)) , H(PP(14)) , H(PP(15)) , H(PP(16)) , H(PP(17)) , H(PP(18)) ,
1378:     W H(PP(19)) , H(PP(20)) , H(PP(21)) , H(PP(22)) , H(PP(23)) , H(PP(24)) ,
1379:     W H(PP(25)) , H(PP(26)) , H(PP(27)) , H(PP(28)) , H(PP(29)) ,
1380:     & H(LSMICP) , H(LSMACP) , NPTDS , NPTCS , NGSP , H(LWWD) , H(LWWD2) ,
1381:     & H(LWWR) , H(LWSD) , H(LWSC) , NORDDS , NOPSDDS ,
1382: c##<2007/03/14:PN3:PN4:
1383:     & NUCPNA , NUC_MAX , A(LDNSTP) , A(LLPDNP) , A(LIATMT) , A(LNATMT) ,
1384:     & A(LDNSTPN) , A(LMNUCPN) , A(LLPIDPN) , A(LIZTBNP) , CHA(LNCIDPN) ,
1385:     & A(LMNEUTPN) , A(LWGTPN) , A(LPPNBR) , A(LPFCNR) , NPPNBR , NPFCN ,
1386:     & A(LPMT) , NPMT , MXPGEN , EBOTX , NSTALY , TCUT , A(LTIMEB) , A(LCX) ,
1387:     & MCX , A(LKLIB1) , A(LKLIB2) , A(LXLIB1) , NMT , NUCPN , NUCPNI ,
1388:     & NMTPN , NSMICPN , MCXPN , A(LCXPN) , A(LCXPN) , A(LKLBPN1) ,
1389:     & A(LKLBPN2) , A(LXLBPN1) , A(LXLBPN2) , H(LSMICPN) , ETOPLPN ,
1390:     & EBOTLPN , NNCSTPN , A(LINCSTPN) , NMTPN , A(LMTMPN) , KPNPRD ,
1391:     & NMOPNPW , A(LMONPNW) , H(LNMPNWGT) , H(LWMPNWGT) , H(PP(30)) ,
1392:     & H(PP(31)) , H(LRNU) , H(LKPNFG) , A , H ,
1393:     & A(LXPDET) , A(LIPDET) , A(LIPDT2) , H(LIMSFL) , H(LIMNFI) ,
1394:     & H(LIMZFL) , H(LIMDED) , H(LIMSLT) , H(LIMNLT) , H(LIMZLT) ,
1395:     & H(LKZZI) , H(LLPOSI) , H(LLCRSI) , H(LOPTI) , H(LPATH) ,
1396:     & H(LKDETP) , H(LKLSFI) , H(LSMACI) , H(LMMACI) , NPDET ,
1397:     & NPLEN , IMPMAX , H(PP(46)) ,
1398: c##>
1399: c+beff2
1400:     & H(LWWD) , H(LWSB) , NPTBE ,
1401:     & H(LWWD) , H(LWSB) , H(LWWD) , H(LWWD)
1402: c-beff2
1403: C/#ENDIF
1404:                                     if(JVMNT.ne.0) call VMNTR2(12)
1405: C
1406: C-----
1407: C .... NEXT EVENT ESTIMATOR .....
1408: C-----
1409:     else if(MACT.eq.8) then
1410:         call NXTEE( IMODE, A, H, NDIMPT, NIMPT )
1411: C
1412: C-----
1413: C end of a batch (and have remaining batch yet)
1414: C-----
1415:     else if( MACT.eq.88 ) then
1416: C
1417: C ..... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR THE FIRST BATCH
1418: C
1419:         call TALSUM( IOW,A,H, CHA, NBATCH, NSKIP, NTMINT, NBPINT,
1420:         2 JBPRT,NGROUP,NPKIND,NREG,NRESP,NGENEO,WSUMB,NUC,NEMIC,NEMAC,
1421:         4 NSTAL , NFISB , NEVENT , NTREG , A(LIPTRG) , A(LLPTRG) ,
1422:         5 H(LWSUM) , H(LWLEK) , H(LFLTR) , H(LFLCL) ,
1423:         6 H(LSFLTR) , H(LSFLCL) , A(LRESP) , H(LSRETR) , H(LSRECL) ,
1424:         6 H(LXSOC) , H(LXKEF) , H(LNCNTR) , H(LWCNTR) , H(LWCXY) ,
1425:         7 H(LRMIC) , H(LSRMIC) , H(LRMICR) , H(LXMIC) , H(LDNFLX) ,
1426:         7 H(LRMAC) , H(LRMACR) , H(LXMAC) ,
1427:         8 H(LDMAC) , H(LRSTR) , H(LRSCL) , H(LSRSTR) , H(LRSRCL) ,
1428:         8 A(LLEMIC) , A(LLEMAC) , H(LTFLH) , A(LTFLHS) ,
1429:         T H(LLETAL) , NETALY , A(LIDTAL) , NDTALY , H(LLETAL) , NLETAL ,
1430:         T H(LDTALY) , NLDTAL , NETRV , METRV , A(LIETRV) , H(LETRV) ,

```

src/mvp/action.f

```

1431:      T H(LDNFLXSM),H(LRMICSM),H(LSRMICSM),H(LXMICSM),H(LRMACSM),
1432:      T H(LXMACSM), KY, KYPOS, NZONE, A(LKZREG), A(LDNZON), NGP(KPNEUT),
1433:      T H(LRMIMU), H(LSMIMU), H(LWMIMU), H(LRMAMU),
1434:      T NSUZON, NSPACE, A(LKCELL), A(LISUSP), NINPZ, A(LKMAT), A(LKREG),
1435:      W H(P8(1)), H(P8(2)), H(P8(3)), H(P8(4)), H(P8(5)), H(P8(6)),
1436:      W H(P8(7)), H(P8(8)), H(P8(9)), H(P8(10)), H(P8(11)), H(P8(12)),
1437:      W H(P8(13)), H(P8(14)), H(P8(15)), KJSCTM,
1438: c+beffl
1439:      & NEBEF, H(LWCBEF), H(LXBEF) )
1440: c-beffl
1441:      if ( JPERT.ne.0 ) then
1442:          call TALSP( IOW, A, H, CHA, NBATCH, NSKIP, NBPINT, JBPRNT,
1443:          & NGROUP, NGP1, NGP2, NPKIND, NREG, NRESF, NGENEO, NUC, NEMIC, NEMAC,
1444:          & NSTAL, NFISB, NEVENT, NREG,
1445:          & H(LXSOC), H(LXKEFP), H(LWCTRF),
1446:          & NUOPT, NTFT )
1447:      end if
1448: C
1449: C ..... OPTIMIZATION OF KMEMO ARRAY .....
1450: C
1451:      if( NBATCH.le.NMEMOP ) then
1452:          call MEMMEM(
1453:          & IOW, H(LKMEMO), H(LMEMC), H(LMEMZ), NZONE, NMEMO, NBATCH)
1454:      end if
1455: C
1456: C ..... Restart file output after specified batch or history .....
1457: C
1458:      if( JRSTF.ne.0.and.NTASK.eq.1.and.
1459:      & NBATCH.gt.0.and.NRSINT.ne.0 ) then
1460:          if( NRSINT.gt.0.and. mod(NBATCH, NRSINT).eq.0
1461:          & .or.(NRSINT.lt.0.and.mod(NTHIST,abs(NRSINT)).lt.NHIST) )
1462:          & then
1463:              write(IOW,7040) NBATCH, NTHIST
1464: ccccccccc call RWIND( IROT )
1465:          call REST00(IOW, H, 0,TITLE, NTHIST, NBATCH)
1466:          call RESTOT(IOW, H, 0,IDTASK,TITLE)
1467:          call RWIND( IROT )
1468:      end if
1469:      end if
1470: 7040 format(1x,' === RESTART FILE OUTPUT AFTER BATCH ',i6,
1471:      & ', HISTORY',i12,' ===')
1472: C
1473: C ..... Check elapsed time and CPU time
1474: C
1475: C
1476:      call TOKEI(TE1,0)
1477: C/#IF PARA(SX*)
1478: *      call PTCLOCK(DT)
1479: *      t1 = dt
1480: C/#ELSEIF PARA(CRAY*)
1481: *      call TSECND(T1)
1482: C/#ELSE
1483:      call CPUTM(T1)
1484: C/#ENDIF
1485:      NBT1 = NBATCH+1
1486: C
1487: C/#IF PARA(SX* CRAY*)
1488: *      call MVPSYNC_LOCK(2)
1489: C/#ENDIF
1490: C
1491: C      ... print message for starting batch ...
1492: C
1493:      if ( JBPRNT.ne.0 ) then
1494:          write(IOW,7060) IDTASK, NBT1, T1, TE1
1495:      end if

```

```

1496: 7060      format(/' - TASK ',i5,' BATCH ',i8,1p,
1497:      & ' --- CPU ',e11.4,' (SEC) ELAPSED ',e11.4,' (SEC) ---- ')
1498:      if(JTLLST.ne.0) write(IOTL) 'BATCH ', NBT1 ! time-list
1499: C/#IF PARA(SX* CRAY*)
1500: *      call MVPSYNC_UNLOCK(2)
1501: C/#ENDIF
1502:      if( TCPU.ne.0.0.and.T1.gt.TCPU*60.0 ) then
1503:          write(IOW,7080) NPART,T1,TCPU
1504:          go to 9999
1505:      end if
1506: 7080      format(/2x,8('%%%%%%%%%')/
1507:      & 3x,'Run terminated before running ',i10,
1508:      & ' histories, because'/
1509:      & 2x,'used cpu time (' ,1p,e11.4,' sec) ',
1510:      & 'exceeded program limit (' ,e11.4,' min.).'/
1511:      & 2x,8('%%%%%%%%%')/
1512:      & )
1513: C
1514: C .... Check status of usr signal-1 and terminate calculation
1515: C      if SIGUSR1 was caught .....
1516: C
1517: C/#IF .NOT.PARA.AND.UNIX
1518:      ISUSR1 = 0
1519:      call FGETUSRSIG( 1, ISUSR1 )
1520:      if ( ISUSR1.ne.0 ) then
1521:          write(IOW,7100) NTHIST
1522:          goto 9999
1523: 7100      format(/2x,8('%%%%%%%%%')/
1524:      & 3x,'Run terminated after running ',i10,
1525:      & ' histories, because'/
1526:      & 3x,'a signal to terminate calculation is caught.'/
1527:      & 2x,8('%%%%%%%%%')/
1528:      & )
1529:      end if
1530: C/#ENDIF
1531: C
1532: C ... update remaining history number and finished history number
1533: C      for next batch
1534: C
1535: C      NGBAT = MIN( NPART-NTHIST, NHIST )
1536: C      NHIST1 = NGBAT
1537: C      NGENEO = 0
1538: C      WSUMB = 0.0D0
1539: C
1540: C ... select fission source of next batch for eigenvalue problem ...
1541: C
1542: C      if ( JEIGN.ne.0 ) then
1543:          call FISSET( IOW, H(LIFISB), WFFACT,
1544:          & IRAND, NFISB, NHIST, NHIST1, NHSUB, NBANK,
1545:          & A(LWGTf), A(LKZREG),
1546:          & H(LXSOC), NLENG,
1547:          & H(LXXXF), H(LYYYF), H(LZZZF), H(LLEEF), H(LINUF), H(LIZZF),
1548:          & H(LLEVLf), H(LLZZF), H(LLPOSF), H(LLCRSF),
1549:          & H(LIBRGF), H(LIBSPF), H(LKLSFF),
1550:          & H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK,
1551:          & H(LXXXF2), H(LYYYF2), H(LZZZF2), H(LLEEF2),
1552:          & H(LINUF2), H(LIZZF2),
1553:          & H(LLEVLf2), H(LLZZF2), H(LLPOSF2), H(LLCRSF2),
1554:          & H(LIBRGF2), H(LIBSPF2), H(LKLSFF2),
1555:          & H(LDFBK2), KDFBK2, MDFBK2,
1556:          & H(LIFBK2), KIFBK2, MIFBK2,
1557:          & H(P10(1)) )
1558:      end if
1559: C
1560: C

```

src/mvp/action.f

```

1561: C
1562: C-----
1563: C All histories are finished
1564: C-----
1565: C
1566: C      else if( MACT.eq.99 ) then
1567: C
1568: C ..... TAKE SUMMATION OF TALLY ARRAYS EXCEPT FOR INITIAL BATCH
1569: C
1570: C      call TALSUM( IOW, A, H, CHA, NBATCH, NSKIP, NTMINT, NBPINT,
1571: C      2 JBERNT,NGROUP,NPKIND,NREG,NRESP,NGENE0,WSUMB,NUC,NEMIC,NEMAC,
1572: C      4 NSTAL , NFISB , NEVENT, NTREG, A(LPTRG), A(LLPTRG),
1573: C      5 H(LWSUM),H(LWLEK) , H(LFLTR),H(LFLCL),
1574: C      6 H(LSFLTR),H(LSFLCL),A(LRESP),H(LSRETR),H(LSRECL),
1575: C      6 H(LXSOC),H(LXKEF) , H(LNCNTR),H(LWCNTR),H(LWCXTY),
1576: C      7 H(LRMIC),H(LSRMIC),H(LRMICR),H(LXMIC),H(LDNFLX),
1577: C      7 H(LRMAC),H(LRMACR),H(LMAC),
1578: C      8 H(LDMAC),H(LRSTR),H(LRSCL),H(LSRSTR),H(LSRSC),
1579: C      8 A(LLEMIC),A(LLEMAC),H(LTFLH),A(LTFLHS),
1580: C      T H(LIETAL),NETAL,A(LIDTAL),NDTAL,H(LETAL),NLETAL,
1581: C      T H(LDTAL),NLDAL,NETRV,METRV,A(LIETRV),H(LETRV),
1582: C      T H(LDNFLXSM),H(LRMICSM),H(LSRMICSM),H(LXMICSM),H(LRMACSM),
1583: C      T H(LXMACSM),KY,KYPOS,NZONE,A(LKZREG),A(LBNZON),NGP(KPNEUT),
1584: C      T H(LRMIMU),H(LSMIMU),H(LWMIMU),H(LRMAMU),
1585: C      T NSUZON,NSPACE,A(LKCELL),A(LISUSP),NINP2,A(LKMAT),A(LKREG),
1586: C      W H(P8(1)),H(P8(2)),H(P8(3)),H(P8(4)),H(P8(5)),H(P8(6)),
1587: C      W H(P8(7)),H(P8(8)),H(P8(9)),H(P8(10)),H(P8(11)),H(P8(12)),
1588: C      W H(P8(13)),H(P8(14)),H(P8(15)),KJSCTM,
1589: c+beffl
1590: C      & NEBEF,H(LWCBEF),H(LXBEF) )
1591: c-beffl
1592: C      if ( JPERT.ne.0 ) then
1593: C      call TALSP( IOW, A, H, CHA, NBATCH, NSKIP, NBPINT, JBERNT,
1594: C      & NGROUP,NGP1,NGP2,NPKIND,NREG,NRESP,NGENE0, NUC, NEMIC, NEMAC,
1595: C      & NSTAL , NFISB , NEVENT, NTREG,
1596: C      & H(LXSOC),H(LXKEFP),H(LWCTRP),
1597: C      & NUCPT,NTPT )
1598: C      end if
1599: C
1600: C ..... OPTIMIZATION OF KMEMO ARRAY .....
1601: C
1602: C      if( NBATCH.le. NMEMOP ) then
1603: C      call MEMMEM(
1604: C      & IOW,H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
1605: C      end if
1606: C      goto 9999
1607: C      end if
1608: C
1609: C      goto 2222
1610: C
1611: C-----
1612: C ..... END OF RANDOM WALK .....
1613: C-----
1614: C
1615: C 9999 continue
1616: C
1617: C      call TOKEI(TE1,0)
1618: C
1619: C/#IF PARA( SX* )
1620: C      call PTCLOCK( DT )
1621: C      T1 = DT
1622: C/#ELSEIF PARA( CRAY* )
1623: C      call TSECND ( T1 )
1624: C/#ELSE
1625: C      call CPUTM( T1 )

```

```

1626: C/#ENDIF
1627: C
1628: C-----
1629: C ... timing infomation report for each task
1630: C-----
1631: C
1632: C/#IF PARA(SX* CRAY*)
1633: C      call MVPSYNC_LOCK(2)
1634: C      call REPORT(IDTASK,NTGEN, T1-T0, TE1-TE0, H(LNLOST) )
1635: C      call MVPSYNC_UNLOCK(2)
1636: C/#ENDIF
1637: C
1638: C/#IF PARA(PVM MPI)
1639: C/#IF UNIX
1640: C      call FLUSHSTD()
1641: C/#ENDIF
1642: C      call MVPCOM_BARRIER(INFO)
1643: C      IT0 = 1
1644: C      if( IDTASK.eq.IT0 ) then
1645: C      call REPORT(IDTASK,NTGEN, T1-T0, TE1-TE0, H(LNLOST) )
1646: C      TTCPU = T1-T0
1647: C      do 349 I=2,NTASK
1648: C      call MVPCOM_RECV_R4(I,100,TDCPU, 1,info)
1649: C      call MVPCOM_RECV_R4(I,101,TDELP, 1,info)
1650: C      call MVPCOM_RECV_I4(I,102,NTGENI, 1,info)
1651: C      call MVPCOM_RECV_I4(I,103,NLOSTI, 1,info)
1652: C
1653: C      call REPORT(I,NTGENI, TDCPU, TDELP, NLOSTI )
1654: C      TTCPU = TTCPU + TDCPU
1655: C      349 continue
1656: C      write(IPR,'(1X,A,E12.5,A)')
1657: C      & ' CPU TIME SUMMED OVER ALL TASKS: ',TTCPU,' SEC'
1658: C      else
1659: C      it0 = 1
1660: C      call MVPCOM_SEND_R4(IT0,100,T1-T0, 1,info)
1661: C      call MVPCOM_SEND_R4(IT0,101,TE1-TE0, 1,info)
1662: C      call MVPCOM_SEND_I4(IT0,102,NTGEN, 1,info)
1663: C      call MVPCOM_SEND_I4(IT0,103,H(LNLOST), 1,info)
1664: C      endif
1665: C/#ENDIF
1666: C
1667: C      return
1668: C      end

```

src/mvp/actmpp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTMPP( TITLE, A, H, CHA )
3:   C
4:   C/#IF PARA(PVM MPI)
5:   C
6:   C=<MVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C           ( For message passing multi task mode )
9:   C CALLED IN: CENTER
10:  C CALLS:  ACTION,HEADER,VMNTRF
11:  C=====
12:  character TITLE(2)*72
13:  real A(*)
14:  real H(*)
15:  character*4 CHA(*)
16:  C
17:  include 'INC/_KPID$'
18:  include 'INC/_NGPS'
19:  C
20:  include 'INC/_FLAGS'
21:  include ' ../shared/INC/_SIZES'
22:  include 'INC/_SIZES2'
23:  include 'INC/_XBANK'
24:  include 'INC/_SBANK'
25:  include 'INC/_STACK'
26:  include 'INC/_XWORK'
27:  include ' ../shared/INC/_PGEOM'
28:  include 'INC/_CXSEC'
29:  include 'INC/_PXSEC'
30:  include ' ../shared/INC/_PVRED'
31:  include 'INC/_PSOUR'
32:  include ' ../shared/INC/_PTALY0'
33:  include 'INC/_PTALY'
34:  include 'INC/_PTALY2'
35:  include ' ../shared/INC/_PTLSP'
36:  include ' ../shared/INC/_STALY'
37:  C
38:  Ccc include ' ../shared/INC/_COUNTS'
39:  C
40:  C/#IF PARA(PVM)
41:  include ' ../shared/INC/_PVM PARA'
42:  C/#ELSEIF PARA(MPI)
43:  * include 'mpif.h'
44:  C/#ENDIF
45:  include ' ../shared/INC/_TASKDT'
46:  C
47:  include ' ../shared/INC/_IOUNIT'
48:  include 'INC/_IOUNIT2'
49:  C
50:  C ... working array pointers PCA(*) for CEL2AB
51:  include 'INC/_WKC2A'
52:  C
53:  C ... local variables ...
54:  C
55:  C/#IF INTEGER8
56:  * integer*8 KTHIST, IDUM8
57:  C/#ELSE
58:  integer KTHIST
59:  C/#ENDIF
60:  C
61:  C-----
62:  C
63:  C
64:  C-----
65:  C ..... Input Body of restart file

```

```

66:  C           (header is already input in INTRO2 routine)
67:  C-----
68:  C
69:  if ( JREST.ne.0 ) then
70:  C
71:  C ... task 1 inputs data for himself and pass data to other task.
72:  C
73:  if ( IDTASK.eq.1 ) then
74:  call RESTIN( IOW, H, IDTASK, TITLE )
75:  do 100 ITSK = 2, NTASK
76:  call RESTIN( IOW, H, ITSK, TITLE )
77:  100 continue
78:  else
79:  call RESTIN( IOW, H, IDTASK, TITLE )
80:  end if
81:  C
82:  end if
83:  C
84:  C-----
85:  C ..... Monte Carlo run
86:  C-----
87:  C
88:  if( JNRUN.eq.0 ) then
89:  call ACTION( TITLE, A, H, CHA )
90:  end if
91:  C
92:  C-----
93:  C stop task controller process if necessary
94:  C-----
95:  C
96:  C/#IF PARA(PVM)
97:  if ( JREPR.eq.0 .and. JTSKR.ne.0 ) then
98:  if ( IDTASK.eq.1.and.NTASK.gt.1 ) then
99:  call PVMFKILL( ICTASK, INFO )
100:  if ( INFO.ge.0 ) then
101:  write(IPR,7000)
102:  7000 format(// ' TASK CONTROLLER PROCESS TERMINATED',
103:  & ' SUCCESSFULLY.//')
104:  else
105:  call PVMFPERROR( 'TERMINATION OF TASK CONTROLLER.', INFO
106:  & )
107:  end if
108:  end if
109:  end if
110:  C/#ENDIF
111:  C
112:  C-----
113:  C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
114:  C-----
115:  C
116:  if ( JVMNT.ne.0 ) then
117:  call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
118:  call VMNTRF( IOW )
119:  end if
120:  C
121:  call MVPCOM_BARRIER( INFO )
122:  C/#IF UNIX
123:  call FLUSHSTD()
124:  C/#ENDIF
125:  C
126:  C-----
127:  C ..... Output Body of restart file
128:  C-----
129:  if( JRSTF .ne. 0 ) then
130:  C

```


src/mvp/actmpp.f

```

131: C      ... take sum of NTHIST tempoary on KTHIST1
132: C
133:         KTHIST = NTHIST
134: C/#IF INTEGER8
135: *      call TSKSMI( 'NTHIST', IDUM8, KTHIST, 1 )
136: C/#ELSE
137:         call TSKSMI( 'NTHIST', IDUMMY, KTHIST, 1 )
138: C/#ENDIF
139: C
140: C      ... check number of batches
141: C
142:         if ( JEIGN.ne.0 ) then
143:             KBATCH = NBATCH
144:         else
145:             KBATCH = NBATCH
146:             call TSKSMI( 'NBATCH', IDUMMY, KBATCH, 1 )
147:         end if
148: C
149: C      ... output header records ...
150: C
151:         if ( IDTASK.eq.1 ) then
152:             call RWIND( IROT )
153:             call RESTOT( IOW, H, 1, TITLE, KTHIST, KBATCH )
154:         end if
155: C
156: C
157: C      ... output data body ...
158: C
159: C      ... task 1 outputs data for himself and get data from other task.
160: C
161:         if ( IDTASK.eq.1 ) then
162:             call RESTOT( IOW, H, 1, IDTASK, TITLE )
163:             do 110 ITSK = 2, NTASK
164:                 call RESTOT( IOW, H, 1, ITSK, TITLE )
165: 110         continue
166:             call RWIND( IROT )
167:         else
168:             call RESTOT( IOW, H, 1, IDTASK, TITLE )
169:         end if
170:     end if
171: C
172: C-----
173: C ..... Take sum of data on each task
174: C-----
175: C
176:         call MTSUMS( TITLE, H )
177: C
178:         call MVPCOM_BARRIER( INFO )
179: C/#IF UNIX
180:         call FLUSHSTD()
181: C/#ENDIF
182: C
183: C-----
184: C ..... Output fission source file
185: C-----
186: C
187:         if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
188: C
189: C      ... convert in-cell coordinates into absolute coordinates ...
190: C      (XXXF,YYYF,ZZZF)
191: C
192:         if ( JLATT.ne.0 ) then
193:             call CEL2AB( IOW, NFISB, NFBNK0,
194: &                 A, JDEBG, H(LXXXF), H(LYYYF),
195: &                 H(LZZZF), H(LLEVLf), H(LLZZF), H(LLPOSF), H(LLCRSF),

```

```

196: &                 H(LDFBK),KDFBK,MDFBK,
197: &                 H(PCA(1)), H(PCA(2)), H(PCA(3)) )
198:         end if
199: C
200: C      ... gather fission neutron data on task #1
201: C      value of NFISB is changed to the sum of NFISB over tasks.
202: C
203:         if ( NTASK.gt.1 ) then
204:             call MPPFBK( NTASK, IDTASK, NFISB, NFBNK0, NEST, JTLLT,
205: &                 H(LXXXF), H(LYYYF), H(LZZZF), H(LEEF), H(LIZZF),
206: &                 H(LINUF), H(LIBRGF), H(LIBSPF),
207: &                 H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK )
208:         end if
209: C
210:         NUSE = NFISB
211: C
212:         if ( IDTASK.eq.1 ) then
213:             call SCFOUT( 'MVP', TITLE, NUSE, NFBNK0, NEST, NREG,
214: &                 A(LWGTF), NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF),
215: &                 H(LYYYF), H(LZZZF), H(LEEF), H(LIZZF), H(LINUF),
216: &                 H(LIBRGF), H(LIBSPF), CHA(LNUCID), NUC, IERR )
217:         end if
218: C
219:         end if
220: C
221: C ... close time list I/O unit ...
222:         if(JTLST.ne.0) call CLOSE_TLF(IDTASK) ! time-list
223: C
224: C-----
225: C ... tasks other than main task exits here ...
226: C-----
227: C
228:         call MVPCOM_BARRIER( INFO )
229: C/#IF UNIX
230:         call FLUSHSTD()
231: C/#ENDIF
232: C
233: C/#IF .NOT.SYSTEM( FACOMAP3K )
234:         IT0 = 1
235:         if ( IDTASK.ne.IT0 ) then
236:             write(IMG,*) '=== Task ', IDTASK, ' Stopped.'
237:         end if
238: C
239:         call MVPCOM_EXIT( INFO )
240: C
241:         if ( IDTASK.ne.IT0 ) then
242:             stop
243:         end if
244: C/#ENDIF
245: C
246: C/#ENDIF
247: C
248:         return
249:         end
250: C
251: C-----
252: C
253: C=====
254: C      Routines to gather array data on the first task
255: C      history : programmed by M.Sasaki (Jul 1997)
256: C=====
257: C
258: C/#IF PARA(PVM MPI)
259:         subroutine MPPFBK( NTASK, IDTASK,NFISB, NFBNK0,NEST, JTLLT, XXXF,
260: &                 YYYF, ZZZF, EEEF, IZZF, INUF,

```

src/mvp/actmpp.f

```

261:      &          IBRGF, IBSPF,
262:      &          DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
263: C=====
264: C purpose: gather neutron bank data on task ITSK in
265: C   parallel mode on PVM, MPI
266: C history: programmed by M.Sasaki ( 18 Jul 1997 )
267: C 10 Feb 1999: gather optional fission bank data
268: C=====
269:
270:      real*8 XXXF(NFBNK0)
271:      real*8 YYYY(NFBNK0)
272:      real*8 ZZZF(NFBNK0)
273:      real EEEF(NFBNK0)
274:      integer INUF(NFBNK0)
275:      integer IZZF(NFBNK0)
276:      integer IBRGF(NFBNK0)
277:      integer IBSPF(NFBNK0,NEST)
278: C
279:      real*8 DFBK(NFBNK0,*)
280:      integer KDFBK(0:MDFBK)
281:      integer IFBK(NFBNK0,*)
282:      integer KIFBK(0:MIFBK)
283: C -----
284:      call MPPGR8( XXXF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
285:      call MPPGR8( YYYY, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
286:      call MPPGR8( ZZZF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
287:      call MPPGR4( EEEF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
288:      call MPPGI4( INUF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
289:      call MPPGI4( IZZF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
290:      if ( JTLT.ne.0 ) then
291:        call MPPGI4( IBRGF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
292:        do 100 J = 1, NEST
293:          call MPPGI4( IBSPF(1,J), NFBNK0, NFISB, NF, NTASK, IDTASK,
294:            &          IERR )
295:        100 continue
296:      end if
297:
298:      if ( KDFBK(0).gt.0 ) then
299:        do 130 K = 1, KDFBK(0)
300:          call MPPGR8( DFBK(1,K), NFBNK0, NFISB,
301:            &          NF, NTASK, IDTASK, IERR )
302:        130 continue
303:      end if
304:
305:      if ( KIFBK(0).gt.0 ) then
306:        do 150 K = 1, KIFBK(0)
307:          call MPPGI4( IFBK(1,K), NFBNK0, NFISB,
308:            &          NF, NTASK, IDTASK, IERR )
309:        150 continue
310:      end if
311: C
312:      NFISB = NF
313: C
314:      return
315:    end
316: C/#ENDIF
317:
318: C=====
319: C   gather data on an array on task #1.
320: C   Data size may differ in each task.
321: C-----
322: C arguments ( i=input, o=output, w=work )
323: C io IA(na),RA(na),Da(na) : integer,real*4,real*8 arrays
324: C i na : array dimension size
325: C i nd : data size on calling task.

```

```

326: C o ndd : returned as sum of data size on all tasks.
327: C i ntask : number of tasks
328: C i itsk : calling task #
329: C=====
330:      subroutine MPPGI4( IA, NA, ND, NDD, NTASK, ITSK, IERR
331:        &
332:      )
333: C/#IF PARA( PVM MPI)
334:      integer IA(NA)
335:      parameter( MSGTG1 = 987 )
336:      parameter( MSGTG2 = 988 )
337:      parameter( MSGTG3 = 989 )
338: C
339:      include '../shared/INC/_IOUNIT'
340: C
341:      IERR = 0
342: C
343:      if ( ITSK.eq.1 ) then
344:        ND0 = ND
345:        do 100 IT = 2, NTASK
346:          call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
347:
348:          if ( ND0+NN.gt.NA ) then
349:            write(IPR,*)
350:              &          'XXX(MPPGI4) Too many data are being gathered.',
351:              &          ' from task ', IT, ' size ', ND,
352:              &          ' gathered data so far ', ND0, ' limit : ', NA
353:            IERR = 1
354:          end if
355:
356:          call MVPCOM_RECV_I4( IT, MSGTG2, IA(ND0+1), NN, INFO )
357:          ND0 = ND0 + NN
358:        100 continue
359:        NDD = ND0
360:        call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
361:      else
362:        call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
363:        call MVPCOM_SEND_I4( 1, MSGTG2, IA, ND, INFO )
364:        call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
365:        if ( IERR.ne.0 ) then
366:          write(IPR,*) 'XXX(MPPGI4) Error on Task #1.'
367:        end if
368:      end if
369: C/#ENDIF
370:      return
371:    end
372: C
373: C-----
374: C
375:      subroutine MPPGR4( RA, NA, ND, NDD, NTASK, ITSK, IERR
376:        &
377:      )
378: C/#IF PARA( PVM MPI)
379:      real RA(NA)
380:      parameter( MSGTG1 = 987 )
381:      parameter( MSGTG2 = 988 )
382:      parameter( MSGTG3 = 989 )
383: C
384:      include '../shared/INC/_IOUNIT'
385: C
386:      IERR = 0
387: C
388:      if ( ITSK.eq.1 ) then
389:        ND0 = ND
390:        do 100 IT = 2, NTASK

```

src/mvp/actmpp.f

```

391:      call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
392:
393:      if ( ND0+NN.gt.NA ) then
394:        write(IPR,*)
395:        &      'XXX(MPPGR4) Too many data are being gathered.',
396:        &      ' from task ', IT, ' size ', ND,
397:        &      ' gatherd data so far ', ND0, ' limit : ', NA
398:        IERR = 1
399:      end if
400:
401:      call MVPCOM_RECV_R4( IT, MSGTG2, RA(ND0+1), NN, INFO )
402:      ND0 = ND0 + NN
403: 100  continue
404:      NDD = ND0
405:      call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
406:    else
407:      call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
408:      call MVPCOM_SEND_R4( 1, MSGTG2, RA, ND, INFO )
409:      call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
410:      if ( IERR.ne.0 ) then
411:        write(IPR,*) 'XXX(MPPGR4) Error on Task #1.'
412:      end if
413:    end if
414:  C/#ENDIF
415:    return
416:  end
417: C
418: C-----
419: C
420:      subroutine MPPGR8( DA,      NA,      ND,      NDD,      NTASK, ITSK, IERR
421:      &
422:      )
423:  C/#IF PARA( PVM MPI)
424:    real*8 DA(NA)
425:    parameter( MSGTG1 = 987 )
426:    parameter( MSGTG2 = 988 )
427:    parameter( MSGTG3 = 989 )
428:  C
429:    include '../shared/INC/_IOUNIT'
430:  C
431:    IERR = 0
432:  C
433:    if ( ITSK.eq.1 ) then
434:      ND0 = ND
435:      do 100 IT = 2, NTASK
436:        call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
437:
438:        if ( ND0+NN.gt.NA ) then
439:          write(IPR,*)
440:          &      'XXX(MPPGR8) Too many data are being gathered.',
441:          &      ' from task ', IT, ' size ', ND,
442:          &      ' gatherd data so far ', ND0, ' limit : ', NA
443:          IERR = 1
444:        end if
445:
446:        call MVPCOM_RECV_R8( IT, MSGTG2, DA(ND0+1), NN, INFO )
447:        ND0 = ND0 + NN
448: 100  continue
449:        NDD = ND0
450:        call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
451:      else
452:        call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
453:        call MVPCOM_SEND_R8( 1, MSGTG2, DA, ND, INFO )
454:        call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
455:        if ( IERR.ne.0 ) then
456:          write(IPR,*) 'XXX(MPPGR8) Error on Task #1.'
457:        end if
458:      end if
459:    C/#ENDIF
460:    return
461:  end

```

src/mvp/actsgl.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTSGL(TITLE, A, H, CHA)
3:   C
4:   C/#IF .NOT.PARA
5:   C
6:   C=<MVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C CALLED IN: CENTER
9:   C CALLS:   ACTION,HEADER,VMNTRF
10:  C=====
11:  character TITLE(2)*72
12:  CCCCC include '../shared/INC/_ARRAY'
13:  real I(*)
14:  real H(*)
15:  character*4 CHA(*)
16:  C
17:  include 'INC/_KPID$'
18:  include 'INC/_NGPS'
19:  C
20:  include 'INC/_FLAGS'
21:  include '../shared/INC/_SIZES'
22:  include 'INC/_SIZES2'
23:  include 'INC/_XBANK'
24:  include 'INC/_SBANK'
25:  include 'INC/_STACK'
26:  include 'INC/_XWORK'
27:  include '../shared/INC/_PGEOM'
28:  include 'INC/_CXSEC'
29:  include 'INC/_PXSEC'
30:  include '../shared/INC/_PVRED'
31:  include 'INC/_PSOUR'
32:  include '../shared/INC/_PTALY0'
33:  include 'INC/_PTALY'
34:  include 'INC/_PTALY2'
35:  include '../shared/INC/_PTLSP'
36:  include '../shared/INC/_STALY'
37:  C
38:  ccc include '../shared/INC/_COUNTS'
39:
40:  CC/#ELSEIF PARA(PVM)
41:  ccc include '../shared/INC/_PVMPARA'
42:  CC/#ELSEIF PARA(VPP)
43:  ccc include '../shared/INC/_VPPPARA'
44:  CC/#ENDIF
45:  include '../shared/INC/_TASKDT'
46:  C
47:  include '../shared/INC/_IOUNIT'
48:  include 'INC/_IOUNIT2'
49:  C
50:  C ... working array pointers PCA(*) for CEL2AB
51:  include 'INC/_WKC2A'
52:  C
53:  C ... local variables ...
54:  C
55:  C/#IF INTEGER8
56:  * integer*8 KTHIST
57:  C/#ELSE
58:  integer KTHIST
59:  C/#ENDIF
60:  C
61:  C-----
62:  C
63:  IDTASK = 1
64:  ITASK0 = 1
65:  C

```

```

66:  C-----
67:  C ..... Input Body of restart file
68:  C (header is already input in INTRO2 routine)
69:  C-----
70:  C
71:  if( JREST.ne.0 ) then
72:    call RESTIN(IOW,H,IDTASK,TITLE)
73:  end if
74:  C
75:  C-----
76:  C ..... Monte Carlo run
77:  C-----
78:  C
79:  if( JNRUN.eq.0 ) then
80:    call ACTION( TITLE, A, H, CHA )
81:  end if
82:  C
83:  C-----
84:  C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
85:  C-----
86:  C
87:  if( JVMNT .ne. 0 ) then
88:    call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
89:    call VMNTRF( IOW )
90:  end if
91:  C
92:  C-----
93:  C ..... Output restart file
94:  C-----
95:  if( JRSTF .ne. 0 ) then
96:  C
97:  C ... output header records ...
98:  C
99:  KTHIST = NTHIST
100:  KBATCH = NBATCH
101:  C
102:  call RWIND( IROT )
103:  call RESTO0(IOW, H, 1, TITLE, KTHIST, KBATCH)
104:  C
105:  C ... output data body ...
106:  C
107:  call RESTOT(IOW, H, 1, IDTASK,TITLE)
108:  call RWIND( IROT )
109:  end if
110:  C
111:  C-----
112:  C ..... Output fission source file
113:  C-----
114:  C
115:  if( JEIGN.ne.0 .and. JOSRC.ne.0 ) then
116:  C
117:  C ... convert in-cell coordinates into absolute coordinates ...
118:  C (XXXF,YYYF,ZZZF)
119:  C
120:  if( JLATT.ne.0 ) then
121:    call CEL2AB( IOW, NFISB, NFBNK0,
122:  & A, JDEBG,
123:  & H(LXXXF), H(LYYYF), H(LZZZF),
124:  & H(LLEVL), H(LLZZF), H(LLPOSF),H(LLCRSF),
125:  & H(LDFBK),KDFBK,MDFBK,
126:  & H(PCA(1)),H(PCA(2)),H(PCA(3)))
127:  end if
128:  C
129:  NUSE = NFISB
130:  call SCFOUT( 'MVP', TITLE, NUSE, NFBNK0, NEST, NREG,

```

src/mvp/actsgl.f

```
131:      &      A(LWGTF), NZONE, A(LKZREG), JDEBG, JTLLT,
132:      &      H(LXXXF), H(LYYF), H(LZZZF),
133:      &      H(LLEEF), H(LIZZF), H(LINUF),
134:      &      H(LIBRGF),H(LIBSPF),
135:      &      CHA(LNUCID), NUC, IERR )
136: C
137:      end if
138: C/#ENDIF
139: C
140:      return
141:      end
```

SAFE

src/mvp/actsmp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTSMP( TITLE,
3:     & IDTASK1,IAINFL1,LIMITL1,
4:     & IRANT, NRSTEP,NBATCH1,      NPART1,
5:     & NHIST1, NHSUB1, NBANK1, IMPMAX1, NTHIST1,
6:     & NFBANK1,      NFBNK01,      SRCSP, WSUM, WLEK,
7:     & XAVT, AAVT, EAVT, WCNTR, NCNTR, NLOST,
8:     & SFLTR, SFLCL, SRETR, SRECL, XSOC, XSXV, XKEF,
9:     & SRMIC, RMICR, WCXTY, XMIC,  RMAC,  RMACR, XMAC,
10:    & SRSTR, SRSCl, IETAL, ETALY, TFLHS, ETRV,
11:    & NCLSN, WCLSN, NLEAK, WLEAK, NABSB, WABSB,
12:    & NECUT, WECUT, NTCUT, WTCUT, NKILD, WKILD,
13:    & NSURV, WSURV, NSPLT, WSPLT, IRSUT, IRSMT,
14:    & IRSLT,
15:    & XKEFP,
16:    & SRMICSM, RMACSM,
17:    & WCBEF, XBEF )
18: C
19: C/#IF PARA(SX* CRAY*)
20: C
21: C=<MVP>=====
22: C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
23: C           ( For shared memory multi task mode )
24: C CALLED IN: CENTER
25: C CALLS:  ACTION,HEADER,VMNTRF
26: C=====
27:   character TITLE(2)*72
28:   include '../shared/INC/_ARRAY'
29:   integer IAINFL(3), LIMITL
30:   real A(*)
31:   real H(*)
32:   character*4 CHA(*)
33: C
34:   include 'INC/_KPIDS'
35:   include 'INC/_NGPS'
36: CCC
37:   include '../shared/INC/_TASKDT'
38:   include 'INC/_FLAGS'
39:   include '../shared/INC/_SIZES'
40:   include 'INC/_SIZES2'
41:   include 'INC/_XBANK'
42:   include 'INC/_SBANK'
43:   include 'INC/_STACK'
44:   include 'INC/_XWORK'
45:   include '../shared/INC/_PGEOM'
46:   include 'INC/_CXSEC'
47:   include 'INC/_PXSEC'
48:   include '../shared/INC/_PVRED'
49:   include 'INC/_PSOUR'
50:   include '../shared/INC/_PTALY0'
51:   include 'INC/_PTALY'
52:   include 'INC/_PTALY2'
53:   include '../shared/INC/_PTLSP'
54:   include '../shared/INC/_STALY'
55: C
56: CCC include '../shared/INC/_COUNTS'
57:
58:   include '../shared/INC/_IOUNIT'
59:   include 'INC/_IOUNIT2'
60: C
61: C ... working array pointers PCA(*) for CEL2AB
62:   include 'INC/_WKC2A'
63:
64:   include '../shared/INC/_RAND'
65:
66:   integer SRCSP(NSRCSP)
67:   real*8 WSUM, WLEK
68:   real XAVT(3), AAVT(3), EAVT
69:   integer IAINFL1(3)
70: C/#IF INTEGER8
71: *   integer*8 NPART1, NTHIST1
72: C/#ELSE
73:   integer NPART1, NTHIST1
74: C/#ENDIF
75: C
76: C ... local variables ...
77: C
78:   real*8 DT
79: C/#IF INTEGER8
80: *   integer*8 KKHIST, KTHIST
81: C/#ELSE
82:   integer KKHIST, KTHIST
83: C/#ENDIF
84: C-----
85: C
86: C ... IDTASK : task # passed as user defined task value
87: C ... tcp0 : used task CPU time at startup
88: C
89:   IDTASK = IDTASK1
90: C
91:   call RNINIT( NRSTEP )
92: C
93:   if ( IDTASK.ne.ITASK0 ) then
94:     NPART = NPART1
95:     NHIST = NHIST1
96:     NHSUB = NHSUB1
97:     NBANK = NBANK1
98:     IMPMAX = IMPMAX1
99:     NFBANK = NFBANK1
100:    NFBNK0 = NFBNK01
101:    do 100 I = 1, 3
102:      IAINFL(I) = IAINFL1(I)
103: 100 continue
104:    LIMITL = LIMITL1
105: C
106: C ... clear task local data & set random number seed ...
107: C
108:   call PUTVI( H(IAINFL(1)), LIMITL-IAINFL(1)+1, 0 )
109:   call ATRANS( SRCSP, H(LSRCSP), NSRCSP )
110: C
111: C ... copy source specification data ...
112: C
113:   call ATRANS( SRCSP, H(LSRCSP), NSRCSP )
114: C
115: C ... copy special tally information data specification data ...
116: C
117:   call ATRANS( IETAL, H(LIETAL), NETALY )
118: end if
119: C
120: call PUTVD( H(LWSUM), 1, 0.0D0 )
121: call PUTVD( H(LWLEK), 1, 0.0D0 )
122: call PUTV( H(LXAVT), 3, 0.0E0 )
123: call PUTV( H(LAAVT), 3, 0.0E0 )
124: call PUTV( H(LEAVT), 1, 0.0E0 )
125: C
126: if ( JREST.eq.0 ) then
127:   NBATCH = 0
128:   NTHIST = 0
129:   IRAND = IRANT
130:   IRNSU = IRSUT

```

src/mvp/actsmp.f

```

131:      IRNSM   = IRSMT
132:      IRNSL   = IRSLT
133:      end if
134: C
135:      NCOLS   = 0
136:      NDEAD   = 0
137:      IDEFER   = 0
138:      NFISB   = 0
139:      NIMPT   = 0
140:      IMDEFR   = 0
141: C
142: C-----
143: C
144: C
145: C-----
146: C ..... Input Body of restart file
147: C      (header is already input in INTRO2 routine)
148: C-----
149:
150:      if ( JREST.ne.0 ) then
151:
152:          if ( IDTASK.gt.1 ) then
153:              call MVPSYNC_WAIT( IDTASK-1 )
154:          end if
155: C
156:          call MVPSYNC_LOCK( 1 )
157:
158:          call RESTIN( IOW, H, IDTASK, TITLE )
159:
160:          call MVPSYNC_UNLOCK( 1 )
161: C
162:          if ( NTASK.gt.1.and.IDTASK.lt.NTASK ) then
163:              call MVPSYNC_POST( IDTASK )
164:          end if
165:
166:          call MVPSYNC_BARRIER( 1 )
167:
168:      end if
169:
170: C-----
171: C ..... Monte Carlo run
172: C-----
173:
174:      if( JNRUN.eq.0 ) then
175:          call ACTION( TITLE, A, H, CHA )
176:      end if
177:
178: C-----
179: C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
180: C-----
181:
182:      if ( JVMNT.ne.0 ) then
183:          call MVPSYNC_LOCK( 1 )
184:
185:          call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
186:          call VMNTRF( IOW )
187:
188:          call MVPSYNC_UNLOCK( 1 )
189:      end if
190:
191: C-----
192: C ..... Output Body of restart file
193: C-----
194:      if( JRSTF.ne.0 ) then
195: C

```

```

196: C      ... output header records ...
197: C
198: C
199: C      ... take sum of NTHIST tempoary on NTHIST1 and restore value
200: C      of NTHIST1.
201: C
202:      KKHIST = NTHIST1
203:
204:      call MVPSYNC_BARRIER( 1 )
205:      call MVPSYNC_LOCK( 1 )
206:
207:      if( IDTASK.ne.IDTASK0 ) then
208: C/#IF INTEGER8
209: *          call TSKSMI8( 'NTHIST', NTHIST1, NTHIST, 1 )
210: C/#ELSE
211:          call TSKSMI( 'NTHIST', NTHIST1, NTHIST, 1 )
212: C/#ENDIF
213:      end if
214:
215:      call MVPSYNC_UNLOCK( 1 )
216:      call MVPSYNC_BARRIER( 1 )
217: C
218:      KTHIST = NTHIST1
219:      NTHIST1 = KKHIST
220: C
221: C      ... check number of batches
222: C
223:      if ( JEIGN.ne.0 ) then
224:          KBATCH = NBATCH
225:      else
226:          KK      = NBATCH1
227:
228:          call MVPSYNC_BARRIER( 1 )
229:          call MVPSYNC_LOCK( 1 )
230:
231:          if( IDTASK.ne.IDTASK0 ) then
232:              call TSKSMI( 'NBATCH', NBATCH1, NBATCH, 1 )
233:          end if
234:
235:          call MVPSYNC_UNLOCK( 1 )
236:          call MVPSYNC_BARRIER( 1 )
237:
238:          KBATCH = NBATCH1
239:          NBATCH1 = KK
240:      end if
241: C
242:      if ( IDTASK.eq.1 ) then
243:          call RWIND( IROT )
244:          call RESTO0( IOW, H, 1, TITLE, KTHIST, KBATCH )
245:      end if
246: C
247: C      ... output data body ...
248: C
249:      if ( IDTASK.gt.1 ) then
250:          call MVPSYNC_WAIT( IDTASK-1 )
251:      end if
252: C
253:      call MVPSYNC_LOCK( 1 )
254:
255:      call RESTOT( IOW, H, 1, IDTASK, TITLE )
256:
257:      call MVPSYNC_UNLOCK( 1 )
258: C
259:      if ( NTASK.gt.1.and.IDTASK.lt.NTASK ) then
260:          call MVPSYNC_POST( IDTASK )

```

src/mvp/actsmp.f

```

261:         end if
262:
263:         call MVPSYNC_BARRIER( 1 )
264:
265:         if ( IDTASK.eq.1 ) then
266:             call RWIND( IROT )
267:         end if
268:     end if
269: C-----
270: C ..... Take summation of data on each task
271: C-----
272:
273:     call MTSUMS( TITLE, H, IDTASK1, IAINFL1, LIMITL1, IRANT, NRSTEP,
274: &              NBATCH1, NPART1, NHIST1, NBANK1, IMPMAX1, NTHIST1,
275: &              NFBANK1, NFBNK01, SRCSP, WSUM, WLEK, XAVT, AAVT, EAVT,
276: &              NCNTR, NCNTR, NLOST, SFLTR, SFLCL, SRETR, SRECL, XSOG,
277: &              XSXV, XKEF, SRMIC, RMICR, WCXTY, XMIC, RMAC, RMACR, XMAC,
278: &              SRSTR, SRSC1, IETAL, ETALY, TFLHS,
279: &              NETRV, NSKIP, METRV, ETRV,
280: &              NCLSN, WCLSN, NLEAK,
281: &              WLEAK, NABSB, WABSB, NECUT, WECUT, NTCUT, WTCUT, NKILD,
282: &              WKILD, NSURV, WSURV, NSPLT, WSPLT,
283: &              XKEFP,
284: &              SRMCSM, RMACSM,
285: &              WCBEF, XBEF )
286:
287: C
288: C-----
289: C ..... Output fission source file
290: C-----
291: C
292:     if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
293: C
294: C ... convert in-cell coordinates into absolute coordinates ...
295: C (XXXF,YYYF,ZZZF)
296: C
297:         if ( JLATT.ne.0 ) then
298:             call CEL2AB( IOW, NFISB, NFBNK0,
299: &              A, JDEBG, H(LXXXF), H(LYYYF),
300: &              H(LZZZF), H(LLEVL), H(LLZZF), H(LLPOSE), H(LLCRSF),
301: &              H(LDFBK), KDFBK, MDFBK,
302: &              H(PCA(1)), H(PCA(2)), H(PCA(3)) )
303:         end if
304: C
305: C ... use I/O unit IUB to collect fission neutron data
306: C
307:         if ( IDTASK.eq.1 ) call RWIND( IUB )
308: C
309:         call MVPSYNC_BARRIER( 1 )
310: C
311:         if ( IDTASK.gt.1 ) then
312: C
313:             call MVPSYNC_LOCK( 1 )
314: C
315:             call SMPFOT( IUB, IDTASK, NFISB, NFBNK0, NEST, JTLLT,
316: &              H(LXXXF), H(LYYYF), H(LZZZF), H(LLEEF), H(LIZZF),
317: &              H(LINUF), H(LIBRGF), H(LIBSPF),
318: &              H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
319: C
320:             call MVPSYNC_UNLOCK( 1 )
321: C
322:         end if
323: C
324:         call MVPSYNC_BARRIER( 1 )
325: C

```

```

326:         if ( IDTASK.eq.1 ) then
327:             call RWIND( IUB )
328:             call SMPFIN( IUB, NTASK, NFISB, NFBNK0, NEST, JTLLT,
329: &              H(LXXXF), H(LYYYF), H(LZZZF), H(LLEEF), H(LIZZF),
330: &              H(LINUF), H(LIBRGF), H(LIBSPF),
331: &              H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
332:
333:             NUSE = NFISB
334:             call SCFOUT( 'MVP', TITLE, NUSE, NFBNK0, NEST, NREG,
335: &              A(LWGTF), NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF),
336: &              H(LYYYF), H(LZZZF), H(LLEEF), H(LIZZF), H(LINUF),
337: &              H(LIBRGF), H(LIBSPF), CHA(LNUCID), NUC, IERR )
338:         end if
339:     end if
340: C
341: C/#ENDIF
342: C
343:     return
344: end
345: C
346: C
347: C
348:     subroutine SMPFOT( IUB, ITSK, NFISB, NFBNK0, NEST, JTLLT, XXXF,
349: &              YYYF, ZZZF, EEFF, IZZF, INUF,
350: &              IBRGF, IBSPF,
351: &              DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK )
352: C=====
353: C purpose: output fission neutron bank data on task ITSK in
354: C SMP Parallel mode (SX, CRAY ...)
355: C
356: C Task #1 does not call this routine.
357: C
358: C history: programmed by M.Sasaki ( 19 Jul 1997 )
359: C 10 Feb 1999: output optional fission bank data
360: C=====
361: C/#IF PARA(SX* CRAY*)
362:
363:     real*8 XXXF(NFBNK0)
364:     real*8 YYYF(NFBNK0)
365:     real*8 ZZZF(NFBNK0)
366:     real EEFF(NFBNK0)
367:     integer INUF(NFBNK0)
368:     integer IZZF(NFBNK0)
369:     integer IBRGF(NFBNK0)
370:     integer IBSPF(NFBNK0,NEST)
371: C
372:     real*8 DFBK(NFBNK0,*)
373:     integer KDFBK(0:MDFBK)
374:     integer IFBK(NFBNK0,*)
375:     integer KIFBK(0:MIFBK)
376: C-----
377:     write(IUB) ITSK, NFISB
378:     write(IUB) (XXXF(I),I=1,NFISB)
379:     write(IUB) (YYYF(I),I=1,NFISB)
380:     write(IUB) (ZZZF(I),I=1,NFISB)
381:     write(IUB) (EEFF(I),I=1,NFISB)
382:     write(IUB) (INUF(I),I=1,NFISB)
383:     write(IUB) (IZZF(I),I=1,NFISB)
384:
385:     if ( JTLLT.ne.0 ) then
386:         write(IUB) (IBRGF(I),I=1,NFISB)
387:         write(IUB) ((IBSPF(I,J),I=1,NFISB),J=1,NEST)
388:     end if
389:
390:     if ( KDFBK(0).gt.0 ) then

```


src/mvp/actsmp.f

```
391:      do 100 K=1,KDFBK(0)
392:        write(IUB) (DFBK(I,K),I=1,NFISB)
393:      100 continue
394:      end if
395:
396:      if ( KIFBK(0).gt.0 ) then
397:        do 110 K=1,KIFBK(0)
398:          write(IUB) (IFBK(I,K),I=1,NFISB)
399:        110 continue
400:      end if
401: C
402: C/#ENDIF
403:      return
404:      end
405: C
406: C
407:      subroutine SMPFIN( IUB,  NTASK, NFISB, NFBNK0,NEST,  JTLLT, XXXF,
408:      &                YYYYF, ZZZF,  EEEF, IZZF,  INUF,
409:      &                IBRGF, IBSPF,
410:      &                DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
411: C-----
412: C purpose: input fission neutron bank data on scratch file in
413: C   SMP parallel mode (SX, CRAY ...)
414: C
415: C   Only task 1 calls this routine.
416: C
417: C history: programmed by M.Sasaki (Jul 1997)
418: C 10 Feb 1999: input optional fission bank data
419: C-----
420: C/#IF PARA(SX* CRAY*)
421:
422:      real*8 XXXF(NFBNK0)
423:      real*8 YYYYF(NFBNK0)
424:      real*8 ZZZF(NFBNK0)
425:      real  EEEF(NFBNK0)
426:      integer INUF(NFBNK0)
427:      integer IZZF(NFBNK0)
428:      integer IBRGF(NFBNK0)
429:      integer IBSPF(NFBNK0,NEST)
430: C
431:      real*8 DFBK(NFBNK0,*)
432:      integer KDFBK(0:MDFBK)
433:      integer IFBK(NFBNK0,*)
434:      integer KIFBK(0:MIFBK)
435: C-----
436:      do 120 N = 2, NTASK
437:        read(IUB) ITSK, NFISB0
438:        K1      = NFISB + 1
439:        NFISB   = NFISB + NFISB0
440:
441:        read(IUB) (XXXF(I),I=K1,NFISB)
442:        read(IUB) (YYYYF(I),I=K1,NFISB)
443:        read(IUB) (ZZZF(I),I=K1,NFISB)
444:        read(IUB) (EEEF(I),I=K1,NFISB)
445:        read(IUB) (INUF(I),I=K1,NFISB)
446:        read(IUB) (IZZF(I),I=K1,NFISB)
447:
448:        if ( JTLLT.ne.0 ) then
449:          read(IUB) (IBRGF(I),I=K1,NFISB)
450:          read(IUB) ((IBSPF(I,J),I=K1,NFISB),J=1,NEST)
451:        end if
452:
453:        if ( KDFBK(0).gt.0 ) then
454:          do 100 K=1,KDFBK(0)
455:            read(IUB) (DFBK(I,K),I=K1,NFISB)
456:          100 continue
457:        end if
458:
459:        if ( KIFBK(0).gt.0 ) then
460:          do 110 K=1,KIFBK(0)
461:            read(IUB) (IFBK(I,K),I=K1,NFISB)
462:          110 continue
463:        end if
464:
465:        120 continue
466: C
467: C/#ENDIF
468:      return
469:      end
```

src/mvp/actvpp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTVPP( TITLE, A, H, CHA )
3:   C
4:   C/#IF PARA(VPP)
5:   C
6:   C=<MVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C           ( For VPP Fortran multi task mode )
9:   C CALLED IN: CENTER
10:  C CALLS:  ACTION,HEADER,VMNTRF
11:  C=====
12:  character TITLE(2)*72
13:  CCC   include '../shared/INC/_ARRAY'
14:  real A(*)
15:  real H(*)
16:  character*4 CHA(*)
17:  C
18:  include 'INC/_KPIDS'
19:  include 'INC/_NGPS'
20:  CCC
21:  include '../shared/INC/_TASKDT'
22:  include 'INC/_FLAGS'
23:  include '../shared/INC/_SIZES'
24:  include 'INC/_SIZES2'
25:  include 'INC/_XBANK'
26:  include 'INC/_SBANK'
27:  include 'INC/_STACK'
28:  include 'INC/_XWORK'
29:  include '../shared/INC/_PGEOM'
30:  include 'INC/_CXSEC'
31:  include 'INC/_PXSEC'
32:  include '../shared/INC/_PVRED'
33:  include 'INC/_PSOUR'
34:  include '../shared/INC/_PTALY0'
35:  include 'INC/_PTALY'
36:  include 'INC/_PTALY2'
37:  include '../shared/INC/_PTLSP'
38:  include '../shared/INC/_STALY'
39:  C
40:  Ccc   include '../shared/INC/_COUNTS'
41:
42:  include '../shared/INC/_VPPPARA'
43:
44:  include '../shared/INC/_IOUNIT'
45:  include 'INC/_IOUNIT2'
46:  C
47:  C ... working array pointers PCA(*) for CEL2AB
48:  include 'INC/_WKC2A'
49:  C
50:  C ... local variables ...
51:  C
52:  C/#IF INTEGER8
53:  *   integer*8 KTHIST
54:  C/#ELSE
55:  integer KTHIST
56:  C/#ENDIF
57:  real*8 TT0, TT1, TT2, TT3
58:  character*137 BUFF
59:  character*4 CWRK
60:
61:  !xocl processor p( mpe )
62:  C
63:  C-----
64:  C
65:  C=====

```

```

66:  C In current implimentation, each process uses its own message
67:  C output I/O unit (IOW).
68:  C=====
69:  C
70:  write(IPR,*) ' VPP multi task : ntask ', NTASK, ' MPE ', MPE
71:  C
72:  do 100 ITSK = 1, NTASK
73:    if ( JVPPS(1).ne.0 ) then
74:      IVPPPR(ITSK) = IPR
75:    else
76:      IVPPPR(ITSK) = 75 + ITSK
77:    end if
78:  C
79:    write(IPR,*) ' VPP multi task : task ', ITSK,
80:    & ' opened I/O unit ', IVPPPR(ITSK)
81:  C
82:  C ... unnamed file
83:  C
84:    if ( VPPPRT.eq.' ' ) then
85:      open( IVPPPR(ITSK), form ='FORMATTED', status ='SCRATCH' )
86:  C
87:  C ... named file
88:  C
89:    else
90:      CWRK = ' '
91:      write(CWRK,'(i2)') IVPPPR(ITSK)
92:      call CCOMP( CWRK, LEN(CWRK), CWRK, II )
93:      BUFF = VPPPRT(:ICLEN2(VPPPRT)) //'.'/CWRK(:ICLEN(CWRK))
94:      open( IVPPPR(ITSK), form ='FORMATTED', file =
95:      & BUFF(:ICLEN(BUFF)), status ='UNKNOWN' )
96:      write(IPR,*) ' file <', BUFF(:ICLEN(BUFF)), '>'
97:    end if
98:  100 continue
99:
100:  LOCK1 = 1
101:  LOCK2 = 2
102:  C
103:  C ... I/O unit iub is used to collect fission source data if necessary
104:  C
105:  call RWIND( IUB )
106:  C
107:  C==== Parallel region starts here =====
108:  C
109:  call GETTOD( TT2 )
110:  C
111:  !xocl parallel region
112:  C
113:  IDTASK = IDVPROC()
114:  C
115:  C-----
116:  C ... setting of printout I/O unit (IOW) and random number.
117:  C-----
118:  C
119:  call PRPARA( NTASK, A(LIRANT), A(LIRSUT), A(LIRSMT), A(LIRSLT) )
120:  C
121:  C-----
122:  C ..... Input Body of restart file
123:  C (header is already input in INTRO2 routine)
124:  C-----
125:  call GETTOD( TT0 )
126:  C
127:  if ( JREST.ne.0 ) then
128:  !xocl spread do
129:    do 110 ITSK = 1, NTASK
130:

```

src/mvp/actvpp.f

```

131: !xocl lockon LOCK1
132:       call RESTIN( IOW, H, ITSK, TITLE )
133: !xocl end lockon
134:
135:       110       continue
136: !xocl end spread
137:       end if
138:       call GETTOD( TT1 )
139:       TMINPP(IDVPROC()) = TT1 - TT0
140:
141:       if ( JSTOP.eq.0 .and. JNRUN.eq.0 ) then
142:
143: !xocl spread do
144:       do 120 ITSK = 1, NTASK
145:
146:               call GETTOD( TT0 )
147:               write(IOW,*) '##### PE      : ', IDVPROC()
148:               write(IOW,*) '##### IRAND : ', IRAND
149: Ccc
150: C-----
151: C ..... Monte Carlo run
152: C-----
153:               call ACTION( TITLE, A, H, CHA )
154: C-----
155: C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED ....
156: C-----
157:               if ( JVMNT.ne.0 ) then
158:                       call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
159:                       call VMNTRF( IOW )
160:               end if
161: C
162:               call GETTOD( TT1 )
163:               TMACT(IDVPROC()) = TT1 - TT0
164:       120       continue
165:
166: !xocl end spread
167:
168:       end if
169: C
170:       call GETTOD( TT0 )
171: C
172: C-----
173: C ..... Output Body of restart file
174: C-----
175:       if( JRSTF.ne.0 ) then
176: C
177: C       ... take sum of NTHIST tempoary on KTHIST
178: C
179:               KTHIST = 0
180: !xocl spread do
181:               do 130 I = 1, NTASK
182:                       KTHIST = KTHIST + NTHIST
183:       130       continue
184: !xocl end spread sum(KTHIST)
185: C
186: C
187:               if ( JEIGN.ne.0 ) then
188:                       KBATCH = NBATCH
189:               else
190: !xocl spread do
191:               do 140 I = 1, NTASK
192:                       KBATCH = KBATCH + NBATCH
193:       140       continue
194: !xocl end spread sum(KBATCH)
195:       end if

```

```

196: C
197: C       ... output header records ...
198: C
199: Check #####
200: C!xocl spread do
201: C       do i=1,ntask
202: C               write(iow,* ) ' %%% before resto0 : idtask ',idtask
203: C       end do
204: C!xocl end spread
205: C #####
206: C
207: C       .... Using spread do only to use barrier ...
208: C
209: !xocl spread do
210:       do 150 ITSK = 1, NTASK
211:               if ( IDTASK.eq.1 ) then
212:                       call RWIND( IROT )
213:                       call REST00( IOW, H, 1, TITLE, KTHIST, KBATCH )
214:               end if
215:       150       continue
216: !xocl end spread
217: C
218: C ... using barrier may cause error , but why ?
219: C
220: C!xocl barrier
221: C
222: C       ... output data body ...
223: C
224: !xocl spread do
225:       do 160 ITSK = 1, NTASK
226:
227: !xocl lockon LOCK2
228:               call RESTOT( IOW, H, 1, ITSK, TITLE )
229: !xocl end lockon
230:
231:       160       continue
232: !xocl end spread
233: C
234: !xocl spread do
235:       do 170 I = 1, NTASK
236:               if ( IDTASK.eq.1 ) then
237:                       call RWIND( IROT )
238:               end if
239:       170       continue
240: !xocl end spread
241:       end if
242:       call GETTOD( TT1 )
243:       TMOUTP(IDVPROC()) = TT1 - TT0
244: C
245: C-----
246: C ..... Take sum of data on each task
247: C-----
248:
249:       call GETTOD( TT0 )
250:
251:       call MTSUMS( TITLE, H )
252:
253:       call GETTOD( TT1 )
254:       TMSUM(IDVPROC()) = TT1 - TT0
255: C
256: C
257: C-----
258: C ... convert in-cell coordinates into absolute coordinates
259: C       for fission file output ...
260: C       (XXXF,YYYF,ZZZF)

```

src/mvp/actvpp.f

```

261: C-----
262:       call GETTOD( TT0 )
263:       if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
264: C
265:         if ( JLATT.ne.0 ) then
266:           call CEL2AB( IOW, NFISB, NFBNK0,
267:             &          A, JDEBG, H(LXXXF), H(LYYYF),
268:             &          H(LZZZF), H(LLEVL), H(LLZZF), H(LLPOSF), H(LLCRSF),
269:             &          H(LDFBK),KDFBK,MDFBK,
270:             &          H(PCA(1)), H(PCA(2)), H(PCA(3)) )
271:         end if
272: C
273: C ... XXXF,YYYF,ZZZF,INUF,EEEF & IBRGF on file
274: C
275: !xocl spread do
276:   do 180 ITSK = 1, NTASK
277:     if ( ITSK.gt.1 ) then
278: !xocl lockon LOCK2
279:       call VPPFOT( IUB, ITSK, NFISB, NFBNK0, NEST, JTLLT,
280:         &          H(LXXXF), H(LYYYF), H(LZZZF), H(LLEEF), H(LIZZF),
281:         &          H(LINUF), H(LIBRGF), H(LIBSPF),
282:         &          H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK )
283: !xocl end lockon
284:       end if
285:       180 continue
286: !xocl end spread
287: C
288:     end if
289:
290:     call GETTOD( TT1 )
291:     TMOUTP(IDVPROC()) = TMOUTP(IDVPROC()) + TT1 - TT0
292: C
293:     call PESUM8( TMINPP(1), NTASK )
294:     call PESUM8( TMACT(1), NTASK )
295:     call PESUM8( TMOUTP(1), NTASK )
296:     call PESUM8( TMSUM(1), NTASK )
297: C
298:     if ( IOW.ne.6 ) call RWIND( IOW )
299:     IOW = 6
300: C
301: !xocl end parallel
302: C
303:       call GETTOD( TT3 )
304:       TMPARA = TT3 - TT2
305: C
306: C-----
307: C ..... Output fission source file (outside of parallel region )
308: C-----
309: C
310:       if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
311: C
312:         ... gather fission neutron bank data of tsak 2 - Ntask
313: C         output on I/O unit IUB
314: C
315:         call RWIND( IUB )
316:         call VPPFIN( IUB, NTASK, NFISB, NFBNK0, NEST, JTLLT, H(LXXXF),
317:           &          H(LYYYF), H(LZZZF), H(LLEEF), H(LIZZF), H(LINUF),
318:           &          H(LIBRGF), H(LIBSPF),
319:           &          H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK )
320:
321:
322:         NUSE = NFISB
323:         call SCFOUT( 'MVP', TITLE, NUSE, NFBNK0, NEST, NREG, A(LWGTF),
324:           &          NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF), H(LYYYF),
325:           &          H(LZZZF), H(LLEEF), H(LIZZF), H(LINUF), H(LIBRGF),

```

```

326:       &          H(LIBSPF), CHA(LNUCID), NUC, IERR )
327:       end if
328: C
329: C-----
330: C ... collect printout on each task
331: C-----
332: C
333:       call GETTOD( TT0 )
334:       if ( JVPPS(1).eq.0 ) then
335:         do 210 ITSK = 1, MPE
336:
337:           call RWIND( IVPPPR(ITSK) )
338:
339:           write(IOW,7000) ITSK
340:           format(/IX,'@@@ OUPUT FROM TASK #',I3,' ON VPP-FORTRAN'//)
341:           190 BUFF = ' '
342:           read(IVPPPR(ITSK),'(A)',end =200) BUFF
343:           write(IOW,*) BUFF(:ICLEN2(BUFF))
344:           go to 190
345:
346:           200 close( IVPPPR(ITSK) )
347:           210 continue
348:         end if
349:         call GETTOD( TT1 )
350:         TMPRINT = TT1 - TT0
351:
352: C/#ENDIF
353: C
354:       return
355:     end
356: C
357: C
358: C
359:       subroutine VPPFOT( IUB, ITSK, NFISB, NFBNK0,NEST, JTLLT, XXXF,
360:         &          YYYF, ZZZF, EEEF, IZZF, INUF,
361:         &          IBRGF, IBSPF,
362:         &          DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
363: C=====
364: C purpose: output fission neutron bank data on task ITSK in
365: C   Parallel mode on VPP (VPP-Fortran)
366: C
367: C   Task #1 does not call this routine.
368: C
369: C history: programmed by M.Sasaki ( 18 Jul 1997 )
370: C 10 Feb 1999: output optional fission bank data
371: C=====
372: C/#IF PARA(VPP)
373: C
374:       real*8 XXXF(NFBNK0)
375:       real*8 YYYF(NFBNK0)
376:       real*8 ZZZF(NFBNK0)
377:       real EEEF(NFBNK0)
378:       integer INUF(NFBNK0)
379:       integer IZZF(NFBNK0)
380:       integer IBRGF(NFBNK0)
381:       integer IBSPF(NFBNK0,NEST)
382: C
383:       real*8 DFBK(NFBNK0,*)
384:       integer KDFBK(0:MDFBK)
385:       integer IFBK(NFBNK0,*)
386:       integer KIFBK(0:MIFBK)
387: C-----
388:       write(IUB) ITSK, NFISB
389:       write(IUB) (XXXF(I),I=1,NFISB)
390:       write(IUB) (YYYF(I),I=1,NFISB)

```

src/mvp/actvpp.f

```
391:      write(IUB) (ZZZF(I),I=1,NFISB)
392:      write(IUB) (EEEE(I),I=1,NFISB)
393:      write(IUB) (INUF(I),I=1,NFISB)
394:      write(IUB) (IZZF(I),I=1,NFISB)
395:
396:      if ( JTLLT.ne.0 ) then
397:        write(IUB) (IBRGF(I),I=1,NFISB)
398:        write(IUB) ((IBSPF(I,J),I=1,NFISB),J=1,NEST)
399:      end if
400:
401:      if ( KDFBK(0).gt.0 ) then
402:        do 100 K=1,KDFBK(0)
403:          write(IUB) (DFBK(I,K),I=1,NFISB)
404:        100 continue
405:      end if
406:
407:      if ( KIFBK(0).gt.0 ) then
408:        do 110 K=1,KIFBK(0)
409:          write(IUB) (IFBK(I,K),I=1,NFISB)
410:        110 continue
411:      end if
412: C
413: C/#ENDIF
414:      return
415:    end
416: C
417: C
418:      subroutine VPPFIN( IUB, NTASK, NFISB, NFBNK0,NEST, JTLLT, XXXF,
419: &      YYYY, ZZZF, EEEF, IZZF, INUF,
420: &      IBRGF, IBSPF,
421: &      DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
422: C=====
423: C purpose: input fission neutron bank data on scratch file in
424: C   Parallel mode on VPP (VPP-Fortran)
425: C
426: C   Only task 1 calls this routine.
427: C
428: C history: programmed by M.Sasaki ( 18 Jul 1997 )
429: C 10 Feb 1999: input optional fission bank data
430: C=====
431: C/#IF PARA(VPP)
432:
433:      real*8 XXXF(NFBNK0)
434:      real*8 YYYY(NFBNK0)
435:      real*8 ZZZF(NFBNK0)
436:      real EEEF(NFBNK0)
437:      integer INUF(NFBNK0)
438:      integer IZZF(NFBNK0)
439:      integer IBRGF(NFBNK0)
440:      integer IBSPF(NFBNK0,NEST)
441: C
442:      real*8 DFBK(NFBNK0,*)
443:      integer KDFBK(0:MDFBK)
444:      integer IFBK(NFBNK0,*)
445:      integer KIFBK(0:MIFBK)
446: C-----
447:      do 120 N = 2, NTASK
448:        read(IUB) ITSK, NFISB0
449:        K1      = NFISB + 1
450:        NFISB    = NFISB + NFISB0
451:
452:        read(IUB) (XXXF(I),I=K1,NFISB)
453:        read(IUB) (YYYY(I),I=K1,NFISB)
454:        read(IUB) (ZZZF(I),I=K1,NFISB)
455:        read(IUB) (EEEE(I),I=K1,NFISB)
456:
457:        read(IUB) (INUF(I),I=K1,NFISB)
458:
459:        if ( JTLLT.ne.0 ) then
460:          read(IUB) (IBRGF(I),I=K1,NFISB)
461:          read(IUB) ((IBSPF(I,J),I=K1,NFISB),J=1,NEST)
462:        end if
463:
464:        if ( KDFBK(0).gt.0 ) then
465:          do 100 K=1,KDFBK(0)
466:            read(IUB) (DFBK(I,K),I=K1,NFISB)
467:          100 continue
468:        end if
469:
470:        if ( KIFBK(0).gt.0 ) then
471:          do 110 K=1,KIFBK(0)
472:            read(IUB) (IFBK(I,K),I=K1,NFISB)
473:          110 continue
474:        end if
475:
476:        120 continue
477: C
478: C/#ENDIF
479:      return
480:    end
```

src/mvp/artlib.f

```

1:      subroutine ARTLIB( TTEMP, NCIDI, NUCID, IXSF, IXS,  NTDATA3,
2:      &                  CX, LST, KLIMCX, JVPOP, IPR,  IXLOCK,
3:      &                  JXPOOL, ATPOOL, JDEBG, IERR )
4: C=====
5: C purpose : interface to doppler broadning module for temperature free
6: C          neutron cross section library
7: C
8: C          Records 1 & 2 are stored on I/O unit IXS, and record 3 is
9: C          stored on array CX.
10: C
11: C-----
12: C arguments ( i=input, o=output, w=work )
13: C
14: C i TTEMP : nuclide temperature (Kelvin)
15: C i NCIDI : nuclide ID as input
16: C i NUCID : nuclide ID + temperature string
17: C i IXSF : input I/O unit of temperature free library (already opened)
18: C i IXS : output I/O unit of record 1 & 2
19: C o NTDATA3 : length of record 3
20: C ow CX : an array used as working are and on which processed record 3
21: C          is stored
22: C i LST : this routine can use memory from CX(LST)
23: C i KLIMCX : size limit of array CX.
24: C i JVPOP : use vectorized broadning module if non-zero, else use
25: C          non vector module.
26: C i IPR : message printout I/O unit.
27: C i IXLOCK : I/O unit for "lock file" used to save processed file
28: C          in pool directory.
29: C i JXPOOL : use art library pool if possible.
30: C i ATPOOL : diretory path to art library pool.
31: C o IERR : error code (non zero if error occcured)
32: C=====
33:      real TTEMP
34:      real CX(*)
35:      character*(*) NCIDI
36:      character*(*) NUCID
37:      integer JDEBG(*)
38:      character*(*) ATPOOL
39:
40: CCC      parameter( MTMAX = 120, NKMAX = 90 )
41:      include '../artcore/INC/_PARAM'
42:
43: Ccc      parameter( MXNUC = 1500 )
44: C
45: C ***** COMMON data area *****
46:      integer OTAPE
47: C
48:      common /UNITS/ INP, OUP, ITAPE, OTAPE, SCR
49:      common /HEADR/ C1H, C2H, L1H, L2H, N1H, N2H,
50:      & MATH, MFH, MTH, NOSEQ
51: C
52:      common /MAINIO/ NSYSI, NSYSO, NSYSO2
53: C
54:      common /CONTRL/ MATD, ZA, ELUNR, EHUNR, CRSMIN, JEMORY,
55:      & IUNRES, INTUNR, IDUMP, IDBG
56:      common /THMCNT/ METHDT, NATOM, MODF4T
57: C
58:      common /HEAD1/ NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA,
59:      & NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU,
60:      & IOTHERM, ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1,
61:      & LNUD, NNUD, NNF, LASYM, NOTDOP,
62:      & KDUMMY(68),
63:      & ATW, TLAB, ETHERM, ESAB, ELOW,
64:      & EHI, TSTAR, RDUMMY(43)
65: C

```

```

66:      common /HEAD2/ MTINFO(MTMAX), MTPAR(MTMAX), ISTMT(MTMAX),
67:      & IENDMT(MTMAX), NEUMT(MTMAX), LCTMT(MTMAX),
68:      & NEANG(MTMAX), NKFS(MTMAX), NEF5(MTMAX),
69:      & MTTHR(MTMAX), LSTF3(MTMAX), LSTF4(MTMAX),
70:      & LSTF5(MTMAX), NEF5S(NKMAX,MTMAX),
71:      & LFF5S(NKMAX,MTMAX), LSTF5S(NKMAX,MTMAX),
72:      & QVAL(MTMAX), QVAL2(MTMAX), IORDMT(MTMAX)
73: C
74:      common /HEAD3/ NOGAM, NOGAM3, MTGAM(MTMAX), NGTYPE(MTMAX),
75:      & NKGAM(MTMAX), N14GAM(MTMAX), N15GAM(MTMAX),
76:      & LSTGAM(MTMAX), MTTOG(MTMAX), NEGYLD(MTMAX),
77:      & NEF5G(MTMAX), NKFSG(MTMAX), LCTMTG(MTMAX), NCAP,
78:      & NCAPG, MTCAP(MTMAX), MTCAPG(MTMAX), EGI(L(MTMAX),
79:      & EG2U(MTMAX)
80: C
81:      common /F6CONT/ LF6, LSTF6, MTTOF6(MTMAX), EF61L(MTMAX),
82:      & EF62U(MTMAX)
83: C *****
84: C
85: C ... local data
86: C
87:      character*8 HDATE
88:      character*8 MATT
89:      character*120 PTITLE
90:      character*16 NUCOUT
91: C
92:      character*300 FILE, LKFILE
93: C
94:      logical OPD
95: C
96: C-----
97: C
98: C-----initial set
99: C
100:      NSYSO = IPR
101:      NSYSO2 = 0
102: CM      if ( JDEBG(1).ne.0 ) NSYSO2 = IPR
103:      if ( JDEBG(1).gt.2 ) NSYSO2 = IPR
104: C
105:      NIN = IXSF
106:      NOUT = IXS
107: C
108:      OTAPE = 0
109: C
110:      NUCOUT = NCIDI
111: C
112:      NATOM = 0
113:      MODF4T = 0
114: C
115:      call SETCON
116:      call READIN
117: C
118: C ... IVPOP : flag to use vector
119: C
120:      IVPOP = 0
121:      if ( JVPOP.ne.0 ) IVPOP = 1
122: C
123: C
124:      IDBG = 0
125: CM      if ( JDEBG(1).ne.0 ) IDBG = 1
126:      if ( JDEBG(1).ne.0 ) IDBG = JDEBG(1)
127: C
128:      ESAB = 0
129:      TSTAR = 0.0
130:      IRC = 0

```

src/mvp/artlib.f

```

131:      NPTS      = 0
132:      NTDATA     = 0
133:      NUNR       = 0
134:      NUNR2      = 0
135:      NLEV       = 0
136:      NBINA      = 0
137:      NBINE      = 0
138:      NNU        = 0
139:      NMT        = MTMAX
140:      NNK        = NKMAX
141:      IGFLAG     = 0
142:      LFI        = 0
143:      LNU        = 0
144:      LITHEM     = 0
145:      LSTU       = 0
146:      LENDU      = 0
147:      LSNR       = 0
148:      LSN        = 0
149:      NBINAI     = 0
150:      NBINE1     = 0
151:      ATW        = 0.0
152:      TLAB       = 0.0
153:      ETHERM     = 0.0
154:      ESAB       = 0.0
155:      ELOW       = 0.0
156:      EHI        = 0.0
157: Cmod
158:      LSNUD      = 0
159:      NNUD       = 0
160:      LNUD       = 0
161:      NNF        = 0
162:      NOGAM      = 0
163:      NOGAM3     = 0
164:      NCAP       = 0
165:      NCAPG      = 0
166:      LF6        = 0
167:      LSTF6      = 0
168: Cend
169:      call CLEA( RDUMMY, 13, 0.0 )
170:      call ICLEA( KDUMMY, 10, 0 )
171:      LENG      = MTMAX*(13+NKMAX*3)
172:      call ICLEA( MTINFO, LENG, 0 )
173:      call CLEA( QVAL, MTMAX, 0.0 )
174:      call CLEA( QVAL2, MTMAX, 0.0 )
175:      LENG      = MTMAX*10
176:      call ICLEA( MTGAM, LENG, 0 )
177:      call ICLEA( LCTMTG, MTMAX, 1 )
178:      call ICLEA( MTCAP, MTMAX, 0 )
179:      call ICLEA( MTCAPG, MTMAX, 0 )
180:      call CLEA( EG1L, MTMAX, 0.0 )
181:      call CLEA( EG2U, MTMAX, 0.0 )
182:      call ICLEA( MTTOF6, MTMAX*3, 0 )
183: C
184:      call RWIND( IXSF )
185:      read(IXSF) MATT, HDATE, PTITLE, NPTS, NTDATA
186:      call RWIND( IXSF )
187: C
188:      if ( PTITLE(1:9).eq.'VERSION 3' ) then
189:         IVERSN = 3
190:      else if ( INDEX(PTITLE(1:120),'VERSION 2').ne.0 ) then
191:         IVERSN = 2
192:      end if
193: C
194: C      write(NSYSO,*) ' *** npts = ', NPTS
195: C

```

```

196:      IDUMP      = 0
197: C
198: C      ... Place RECORD3 on top of CX ....
199: C
200: C      LREC3     = 1
201: C      LREC3     = LST
202: CM      LOC1     = LREC3 + NTDATA
203: C      LOC1     = LREC3 + NTDATA + NPTS ! extend for 0K elastic XS
204: C      LOC2     = LOC1 + NPTS
205: C      LOC3     = LOC2 + NPTS
206: C      LOC4     = LOC3 + NPTS
207: C      LOC5     = LOC4 + NPTS
208: C      LOC6     = LOC5
209: C      if ( MOD(LOC6,2).eq.0 ) LOC6      = LOC6 + 1
210: C
211: CCC      write(NSYSO,*) ' ** loc6 is ', LOC6
212: C
213: C      LEMORY    = KLIMCX - LOC6 + 1
214: C      LEMORY    = KLIMCX - (LOC6-LST)
215: C      call DIMCHK( 'ARTLIB', LOC5, KLIMCX )
216: C      call DIMCHK( 'ARTLIB', LOC5-LST, KLIMCX )
217: C
218: C -----
219: C      ART module (NTDATA may be modified)
220: C -----
221: C
222: C      call ARTPRC( CX(LOC1), CX(LOC2), CX(LOC3), CX(LOC4), CX(LREC3),
223: C      &           CX(LREC3), CX(LOC6), CX(LOC6), LEMORY, PTITLE, IVPOP,
224: C      &           TTEMP, IXSF , IVERSN)
225: C
226: C      NTDATA3 = NTDATA
227: C
228: C      close( IXSF )
229: C
230: C      if ( IDBG.ge.1 ) call FEND
231: C      if ( IDBG.ge.1 ) call MEND
232: C
233: C      JREC3 = 0 : do not output record3 on file ...
234: C
235: C      JREC3      = 0
236: C
237: C      ... IXS is opened on a named file under directory ATPool
238: C      if possible.
239: C
240: C      JJPOOL    = 0
241: C      if ( ATPool.ne.' '.and.JXPOOL.ne.0 ) then
242: C
243: C          FILE   = ATPool(:ICLEN2(ATPOOL)) //NUCID(:ICLEN2(NUCID))
244: C          inquire(file =FILE(:ICLEN2(FILE)),exist =OPD)
245: C          if ( OPD ) then
246: C              write(IPR,*) '<<MESSAGE>> (ARTLIB) file <',
247: C              &           FILE(:ICLEN2(FILE)),
248: C              &           '> exist. do not write processed data in pool.'
249: C              call CNTERR( 'MESSAGE' )
250: C              go to 100
251: C          end if
252: C
253: C
254: C      ... open lock file
255: C
256: C      LKFILE    = FILE(:ICLEN2(FILE)) //''.LOC'
257: C
258: C      inquire(IXLOCK,opened =OPD)
259: C      if ( OPD ) close( IXLOCK )
260: C

```

src/mvp/artlib.f

```
261:      open( IXLOCK, file =LKFILE(:ICLEN2(LKFILE)), status = 'NEW',
262:      &      form = 'FORMATTED', iostat =IOS )
263:
264:      if ( IOS.ne.0 ) then
265:        write(IPR,*) '!!!(ARTLIB) Cannot open a "lock" file <',
266:      &      LKFILE(:ICLEN2(LKFILE)),
267:      &      '>, so don't save processed', ' library on file <',
268:      &      FILE(:ICLEN2(FILE)), '>.'
269:        call CNTERR( 'WARNING' )
270:        go to 100
271:      end if
272:
273:      open( IXS, file =FILE(:ICLEN2(FILE)), status = 'NEW', form =
274:      &      'UNFORMATTED', iostat =IOS2 )
275:      if ( IOS2.ne.0 ) then
276:        write(IPR,*) '!!!(ARTLIB) Cannot open a file <',
277:      &      LKFILE(:ICLEN2(LKFILE)), '> as a new file in pool.'
278:        call CNTERR( 'WARNING' )
279:
280:        close( IXLOCK, status = 'DELETE' )
281:        go to 100
282:      end if
283:
284:      JJPOOL = 1
285:      JREC3 = 1
286:      write(IPR,*) '>>> Processed data in ARTLIB is saved on file <'
287:      &      FILE(:ICLEN2(FILE)), '>.'
288: C
289:      end if
290: C
291: 100 continue
292: C
293: C      ... open unit IXS as scratch file ...
294: C
295:      if ( JJPOOL.eq.0 ) then
296:        inquire(IXS,opened =OPD)
297:        if ( OPD ) close( IXS )
298:        call OPENF( IXS, ' ', 'UNFORMATTED', 'SCRATCH', 'WRITE', IERR )
299:      end if
300: C
301:      call LIBOUT( CX(LREC3), CX(LREC3), IXS, JREC3, PTTITLE, NUCOUT,
302:      &      IVERS )
303: C
304: C
305:      if ( JJPOOL.ne.0 ) then
306:        close( IXLOCK, status = 'DELETE' )
307:      end if
308: C
309:      return
310:      end
```


src/mvp/beff.f

```

1:      subroutine BEFF( JPRTS, NBATCH, NSKIP, NEBEF,
2:      &                WCNTR, XSOC, XBEF, XSOCB, XKB, XBB,
3:      &                XBSD, COVXB, XBWKC, XBWKB, CORRB, XBMLE, XBERR
4:      &                )
5: C=====
6: C  PURPOSE: Statistical analysis of beta-effective
7: C          Write the results on file (I/O unit 30)
8: C  CALLED IN: CADENZ
9: C  CALLS      :
10: C=====
11:
12:      implicit real*8(A-H,O-Z)
13:
14:      include '../shared/INC/_IOUNIT'
15:      include 'INC/_IOUNIT2'
16:
17:      parameter (MAXXK = 10)
18:
19:      integer JPRTS(*)
20:      real*8 WCNTR(*)
21:      real*8 XSOC(NBATCH)
22:      real*8 XBEF(NEBEF,NBATCH)
23:
24:      real*8 XSOCB(NBATCH)
25:      real*8 XKB(MAXXK,NBATCH)
26:      real*8 XBB(9,NBATCH)
27:      real*8 XBSD(6,NBATCH)
28:      real*8 COVXB(3,3,NBATCH)
29:      real*8 CORRB(3,3,*)
30:      real*8 XBMLE(3,NBATCH)
31:      real*8 XBERR(3,NBATCH)
32:
33:      real*8 XBWKC(3,NBATCH)
34:      real*8 XBWKB(3,NBATCH)
35:
36: C
37: C ... criterion to reject too much correlated combination of
38: C estimation in maximum likelihood estimation.
39: C
40:      parameter( DECORR = 1.0D-5 )
41: C
42: C ... local data ...
43: C
44:      integer JJCC(3)
45:      integer JJCC0(3)
46:      real*8 WTALL(3), COVINV(3,3)
47:
48: C-----
49: C*** STATISTICAL ANALYSIS OF BETA-EFFECTIVE
50: C-----
51: C
52:      call HEADER( IOT, ' RESULT OF BETA-EFFECTIVE CALCULATION ' )
53:
54:      write(IOT,'(/lx,'' << BETA-EFFECTIVE >> ''')')
55:
56:      XBB(1,1) = XBEF(1,1)/XSOCB(1)/XKB(1,1)
57:      XBB(2,1) = XBEF(2,1)/XSOCB(1)/XKB(2,1)
58:      XBB(3,1) = XBEF(3,1)/XSOCB(1)/XKB(3,1)
59:      XBB(4,1) = XBEF(4,1)/XSOCB(1)/XKB(1,1)
60:      XBB(5,1) = XBEF(5,1)/XSOCB(1)/XKB(2,1)
61:      XBB(6,1) = XBEF(6,1)/XSOCB(1)/XKB(3,1)
62:      XBB(7,1) = XBEF(7,1)/XBEF(10,1)
63:      XBB(8,1) = XBEF(8,1)/XBEF(11,1)
64:      XBB(9,1) = XBEF(9,1)/XBEF(12,1)
65:

```

```

66:      do 110 I = 2, NBATCH
67:          XBB(1,I) = (XBEF(1,I)-XBEF(1,I-1))/XSOCB(I)/XKB(1,I)
68:          XBB(2,I) = (XBEF(2,I)-XBEF(2,I-1))/XSOCB(I)/XKB(2,I)
69:          XBB(3,I) = (XBEF(3,I)-XBEF(3,I-1))/XSOCB(I)/XKB(3,I)
70:          XBB(4,I) = (XBEF(4,I)-XBEF(4,I-1))/XSOCB(I)/XKB(1,I)
71:          XBB(5,I) = (XBEF(5,I)-XBEF(5,I-1))/XSOCB(I)/XKB(2,I)
72:          XBB(6,I) = (XBEF(6,I)-XBEF(6,I-1))/XSOCB(I)/XKB(3,I)
73:          XBB(7,I) = (XBEF(7,I)-XBEF(7,I-1))/(XBEF(10,I)-XBEF(10,I-1))
74:          XBB(8,I) = (XBEF(8,I)-XBEF(8,I-1))/(XBEF(11,I)-XBEF(11,I-1))
75:          XBB(9,I) = (XBEF(9,I)-XBEF(9,I-1))/(XBEF(12,I)-XBEF(12,I-1))
76:      110 continue
77:
78:      write(IOT,7020) '=== BETA-EFFECTIVE ( EACH BATCH ) ==='
79: 7020 format(/lx,18x,a//lx,' CONSTANT WEIGHT ',
80:      &      ' IMPORTANCE FUNCTION WEIGHT '//lx,
81:      &      ' BATCH TRACK L. COLLISION ANALOG ',
82:      &      ' TRACK L. COLLISION ANALOG'/lx,
83:      &      '-----',
84:      &      '-----')
85:
86:      write(IOT,7040) (I,(XBB(J,I),J=1,6),I=1,NBATCH)
87: 7040 format(1X,I5,2X,1P,6E12.5)
88:
89:      write(IOT,'(/)')
90:      write(IOT,'(a)')
91:      &      '=== MEULEKAMP'S BETA-EFFECTIVE ( EACH BATCH ) ==='
92:      write(IOT,'(1x,i5,2x,1p,3e12.5)') (I,(XBB(J,I),J=7,9),I=1,NBATCH)
93:
94: c ... Cumulative after this batch
95:
96:      X1 = 0.d0
97:      X2 = 0.d0
98:      X3 = 0.d0
99:      X4 = 0.d0
100:      X5 = 0.d0
101:      X6 = 0.d0
102:      X7 = 0.d0
103:      X8 = 0.d0
104:      X9 = 0.d0
105:
106:      do J = NBATCH,1,-1
107:          X1 = X1 + XBB(1,J)
108:          X2 = X2 + XBB(2,J)
109:          X3 = X3 + XBB(3,J)
110:          X4 = X4 + XBB(4,J)
111:          X5 = X5 + XBB(5,J)
112:          X6 = X6 + XBB(6,J)
113:          X7 = X7 + XBB(7,J)
114:          X8 = X8 + XBB(8,J)
115:          X9 = X9 + XBB(9,J)
116:          XBEF(1,J) = X1/dbble(NBATCH-J+1)
117:          XBEF(2,J) = X2/dbble(NBATCH-J+1)
118:          XBEF(3,J) = X3/dbble(NBATCH-J+1)
119:          XBEF(4,J) = X4/dbble(NBATCH-J+1)
120:          XBEF(5,J) = X5/dbble(NBATCH-J+1)
121:          XBEF(6,J) = X6/dbble(NBATCH-J+1)
122:          XBEF(7,J) = X7/dbble(NBATCH-J+1)
123:          XBEF(8,J) = X8/dbble(NBATCH-J+1)
124:          XBEF(9,J) = X9/dbble(NBATCH-J+1)
125:      end do
126:
127: C ... Calculation of covariance matrix ...
128:
129:      do IW = 1, 3
130:          JINC = 3*(IW-1)

```

src/mvp/beff.f

```

131:
132:   do J = 1, 3
133:     do I = 1, NBATCH
134:       XBWKC(J,I) = XBEF(J+JINC,I)
135:       XBWKB(J,I) = XBB(J+JINC,I)
136:     end do
137:   end do
138:
139:   if( IW.eq.1 ) then
140:     call BEFCOV( NBATCH, XBWKC, XBWKB, COVXB, 1 )
141:   else if( IW.eq.2 .or. IW.eq.3 ) then
142:     call BEFCOV( NBATCH, XBWKC, XBWKB, COVXB, 2 )
143:   end if
144:
145:   do I = 1, NBATCH - 1
146:     XX = 1.d0/(NBATCH-I+1)
147:     do K = 1, 3
148:       do L = 1, 3
149:         COVXB(K,L,I) = COVXB(K,L,I)*XX
150:       end do
151:     end do
152:   end do
153: C
154: C..... KEEP CORRELATION MATRIX FOR ESTIMATION AFTER (NSKIP+1)TH BATCH
155: C
156:   NSKIP1 = NSKIP + 1
157:
158:   do 240 K = 1, 3
159:     do 230 L = 1, 3
160:       CORRB(K,L,1) = COVXB(K,L,NSKIP1) /
161: &          Sqrt(COVXB(K,K,NSKIP1)*COVXB(L,L,NSKIP1))
162: 230   continue
163: 240   continue
164:
165: C ... Fractional error for each estimator ...
166:
167:   do 260 J = 1, 3
168:     do 250 I = 1, NBATCH - 1
169:       XBSD(J+JINC,I) = Sqrt(ABS(COVXB(J,J,I)))
170: &          /XBEF(J+JINC,I)*100.d0
171: 250   continue
172: 260   continue
173:
174: C ... Print out beta-effective by each estimator ...
175:
176:   if( IW.eq.1 ) then
177:     write(IOT,7060) 'CONSTANT WEIGHT'
178:   else if( IW.eq.2 ) then
179:     write(IOT,7060) 'IMPORTANCE FUNCTION WEIGHT'
180:   else if( IW.eq.3 ) then
181:     write(IOT,7060) 'MEULEKAMP'S METHOD'
182:   end if
183: 7060 format('1',' << BETA-EFFECTIVE BY EACH ESTIMATOR ',
184: &          '( CUMULATIVE AFTER THIS BATCH ) >>'//1X,
185: &          ' === ',a,' === '
186: &          //1X,' BATCH TRACK LENGTH EST. COLLISION EST.',
187: &          ' ANALOG EST. ' //1X,
188: &          ' -----',
189: &          ' -----')
190:
191:   if( IW.eq.1 ) then
192:     do I = 1, NBATCH - 1
193:       write(IOT,7080) I, (XBEF(J+JINC,I),XBSD(J+JINC,I),J=1,3)
194:     end do
195:   else if( IW.eq.2 .or. IW.eq.3 ) then

```

```

196:     do I = 2, NBATCH - 1
197:       write(IOT,7080) I, (XBEF(J+JINC,I),XBSD(J+JINC,I),J=1,3)
198:     end do
199:   end if
200:
201: 7080 format(1X,I5,3(1P,E13.5,'(',OP,F6.2,'%)'))
202:
203: C
204: C... GET MAXIMUM LIKELYHOOD ESTIMATION OF 3 ESTIMATORS
205: C
206:   JCORR1 = 0
207:
208:   J = 1
209:
210:   do 340 I = 1, NBATCH - 2
211:     XBERR(J,I) = 0.0
212:     XBMLE(J,I) = 0.0
213: 340   continue
214:
215:   do 300 I = 1, NBATCH - 2
216: C
217: C ... omit one estimator from too strongly correlated pair ...
218: C ( non-diagonal elements of COVXB for omitted estimator is
219: C cleared to 0.0 )
220: C
221:     do 380 L = 3, 1, -1
222:       JJCC(L) = 0
223:       do 360 K = 1, L - 1
224:
225:         CORR1 = COVXB(K,L,I) /Sqrt(COVXB(K,K,I)*COVXB(L,L,I))
226:
227:         if ( ABS(CORR1-1.0D0).lt.DECORR ) then
228:           JJCC(L) = 1
229:           COVXB(L,L,I) = 1.0D0
230:           do 350 KK = 1, 3
231:             if ( KK.ne.L ) then
232:               COVXB(KK,L,I) = 0.0D0
233:               COVXB(L,KK,I) = 0.0D0
234:             end if
235: 350           continue
236:           go to 370
237:         end if
238: 360       continue
239:
240:       continue
241: 380     continue
242:     if ( I.eq.NSKIP1 ) then
243:       do 390 L = 1, 3
244:         JJCC0(L) = JJCC(L)
245:         WTALL(L) = 0.0D0
246: 390       continue
247:     end if
248: C
249: C ... invert covariance matrix ...
250: C
251:     V11V22 = COVXB(1,1,I)*COVXB(2,2,I)
252:     V22V33 = COVXB(2,2,I)*COVXB(3,3,I)
253:     V33V11 = COVXB(3,3,I)*COVXB(1,1,I)
254:     V12V12 = COVXB(1,2,I)**2
255:     V23V23 = COVXB(2,3,I)**2
256:     V31V31 = COVXB(3,1,I)**2
257:     V12V33 = COVXB(1,2,I)*COVXB(3,3,I)
258:     V23V11 = COVXB(2,3,I)*COVXB(1,1,I)
259:     V31V22 = COVXB(3,1,I)*COVXB(2,2,I)
260:     V12V23 = COVXB(1,2,I)*COVXB(2,3,I)

```

src/mvp/beff.f

```

261:      V23V31 = COVXB(2,3,I)*COVXB(3,1,I)
262:      V31V12 = COVXB(3,1,I)*COVXB(1,2,I)
263:      A      = V11V22*COVXB(3,3,I) + 2.*V12V23*COVXB(1,3,I)
264:      &      - V23V23*COVXB(1,1,I) - V31V31*COVXB(2,2,I)
265:      &      - V12V12*COVXB(3,3,I)
266:      T11    = V22V23 - V23V23
267:      T12    = V23V31 - V12V33
268:      T13    = V12V23 - V31V22
269:      T22    = V33V11 - V31V31
270:      T23    = V31V12 - V23V11
271:      T33    = V11V22 - V12V12
272:      U1     = T11 + T12 + T13
273:      U2     = T12 + T22 + T23
274:      U3     = T13 + T23 + T33
275:      USUM   = U1 + U2 + U3
276:
277:      if ( A.eq.0.0 ) then
278:        write(IOT,'(1x,i4,a)')
279:        &      '!!! CANNOT INVERT COVARIANCE MATRIX (BATCH ', I,
280:        &      ' ) --> MAX.LIKELYHODD ESTIMATOR PRINTED AS -1'
281:        XBMLE(J,I) = -1.0D0
282:        XBERR(J,I) = 0.0
283:
284:      else if ( USUM.le.0.0D0 ) then
285:        if ( JCORR1.eq.0 ) then
286:          JCORR1 = 1
287:          write(IOT,'(1x,i4,a,a,a)')
288:          &      '!!! Some M.L.E. triplets cannot be evaluated ',
289:          &      'correctly because of strong correlation or ',
290:          &      'some other reason.'
291:          write(IOT,'(/a/)')
292:          &      ' Dummy values -1.0 will be output.'
293:        end if
294:        write(IOT,'(1x,a,i4/4x,a,3E14.5)')
295:        &      'Cumulative after batch ', I,
296:        &      'weights : ', U1, U2, U3
297:
298:        XBMLE(J,I) = -1.0D0
299:        XBERR(J,I) = 0.0
300:      else
301:        USUM = 1.0D0/USUM
302:        XBMLE(J,I) = (U1*XBEF(1+JINC,I)+U2*XBEF(2+JINC,I)
303:        &      +U3*XBEF(3+JINC,I))*USUM
304:        XBERR(J,I) = USUM*A
305:        XBERR(J,I) = SQRT(ABS(XBERR(J,I))) /XBMLE(J,I)*100.
306:
307:        COVINV(1,1) = T11
308:        COVINV(1,2) = T12
309:        COVINV(1,3) = T13
310:        COVINV(2,1) = T12
311:        COVINV(2,2) = T22
312:        COVINV(2,3) = T23
313:        COVINV(3,1) = T13
314:        COVINV(3,2) = T23
315:        COVINV(3,3) = T33
316:
317:        do 410 K = 1, 3
318:          do 400 L = 1, 3
319:            if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
320:              XBERR(J,I) = XBERR(J,I) + COVINV(K,L)
321:              XBMLE(J,I) = XBMLE(J,I) + COVINV(K,L)
322:              &      *XBEF(L+JINC,I)
323:            &      if ( I.eq.NSKIP1 ) then
324:              WTALL(L) = WTALL(L) + COVINV(K,L)
325:            end if

```

```

326:          end if
327:        400      continue
328:        410      continue
329:
330:        if ( I.eq.NSKIP1 ) then
331:          do 420 L = 1, 3
332:            if ( JJCC(L).eq.0 ) then
333:              WTALL(L) = WTALL(L) /XBERR(J,I)
334:            end if
335:          420      continue
336:        end if
337:
338:        XBERR(J,I) = 1./XBERR(J,I)
339:        XBMLE(J,I) = XBMLE(J,I)*XBERR(J,I)
340:        XBERR(J,I) = SQRT(ABS(XBERR(J,I))) /XBMLE(J,I)*100.
341:        XBERR(J,I) = SQRT(ABS(XBERR(J,I)*A)) /XBMLE(J,I)*100.
342:
343:      end if
344:    300      continue
345:
346:      write(IOT,7100)
347: 7100 format('1',' << BETA-EFFECTIVE BY THE PRINCIPLE OF',
348:      &      ' MAXIMUM LIKELIHOOD',
349:      &      ' ( CUMULATIVE AFTER THIS BATCH ) >>'//1X,
350:      &      ' BATCH ALL ESTIMATORS'/1X,
351:      &      '-----',
352:      &      '-----')
353:
354:      if( IW.eq.1 ) then
355:        do I = 1, NBATCH - 2
356:          write(IOT,7080) I, (XBMLE(J,I),XBERR(J,I),J=1,1)
357:        end do
358:      else if( IW.eq.2 .or. IW.eq.3 ) then
359:        do I = 2, NBATCH - 2
360:          write(IOT,7080) I, (XBMLE(J,I),XBERR(J,I),J=1,1)
361:        end do
362:      end if
363:
364:      write(IOT,7140) NSKIP1, NBATCH
365: 7140 format('1',' << SUMMARY OF BETA-EFFECTIVE >>',
366:      &      ' === RESULT OF BATCHES FROM ',I2,' TO ',I4,' ==='//
367:      &      ' 1 : TRACK LENGTH ESTIMATOR '/'
368:      &      ' 2 : COLLISION ESTIMATOR '/'
369:      &      ' 3 : ANALOG ESTIMATOR '///)
370: C
371: C ... Print correlation table ...
372: C
373:      write(IOT,'(3x,57(''-'))')
374:      write(IOT,'(1x,a)')
375:      &      ' BEFF ERROR(%) CORRELATION '
376:      write(IOT,'(3x,57(''-'))')
377:
378:      do 460 J = 1, 3
379:        write(IOT,7160) J, XBEF(J+JINC,NSKIP1), XBSD(J+JINC,NSKIP1),
380:        &      (CORRB(K,J,1),K=1,J)
381:      460      continue
382:
383: 7160 format(1X,I5,1P,E15.5,0P,F10.4,0P,6F9.3)
384:
385:      write(IOT,'(3X,57(''-'))')
386:
387:      JJCC1 = JJCC0(1) + JJCC0(2) + JJCC0(3)
388:
389:      if ( XBMLE(J,NSKIP1).eq.-1.0 ) then
390:

```

src/mvp/beff.f

```

391:         write(IOT,'((4X,A/4X,A,A,A))')
392:         &         ' !!! The "ALL" estimator above is not valid !!!',
393:         &         '         Probably "Some estimations are too strongly ',
394:         &         'correlated each other to get a ',
395:         &         'maximum likelihood estimation.'
396:
397:         else if ( JJCC1.gt.0 ) then
398:
399:         write(IOT,'((4X,A/4X,A,A))')
400:         &         ' !!! The "ALL" estimator above is not a whole combination',
401:         &         ' of the 3 estimators.',
402:         &         ' Because some estimations are too strongly correlated.'
403:         write(IOT,'(/4X,A/4X,6E12.5)')
404:         &         ' Weight of each estimator : ',(WTALL(L),L=1,3)
405:
406:         end if
407:
408:         write(IOT,7200) (XBMLE(J,NSKIP1),XBERR(J,NSKIP1),J=1,1)
409: 7200 format(/1X,4X,
410:         &         RESULTS BY THE PRINCIPLE OF MAXIMUM LIKELIHOOD'//
411:         &         4X,
412:         &         '+-----'//
413:         &         4X,' | ALL         --> BEFF=',1P,E12.5,
414:         &         ' (' ,0P,F7.4,'%) | ' ,
415:         &         4X,
416:         &         '+-----'//
417:
418:         do 470 M = 1, 3
419:         write(IOT,7220) M, (XBMLE(J,NSKIP1)*
420:         &         (1.0-M*XBERR(J,NSKIP1)/100.0),XBMLE(J,NSKIP1)*
421:         &         (1.0+M*XBERR(J,NSKIP1)/100.0),J=1,1)
422: 470 continue
423:
424: 7220 format(/1X,4X,'         RANGE IN ',11,' STANDARD DEVIATION'//4X,
425:         &         '+-----+'//
426:         &         4X,
427:         &         ' | ALL         --> ',F12.9,' < BEFF < ',F12.9,' | '/4X,
428:         &         '+-----+'//
429:
430:         end do
431:
432:         return
433:         end
434:
435:         subroutine BEFCOV( NBATCH, XBWKC, XWBKB, COVXB, IST )
436:
437: c i IST: The starting batch number for covariance calculation
438: c         1 = constant weight beta-effective
439: c         2 = importance weight beta-effective
440:
441:         implicit real*8(A-H,O-Z)
442:
443:         real*8 XBWKC(3,NBATCH)
444:         real*8 XWBKB(3,NBATCH)
445:         real*8 COVXB(3,3,NBATCH)
446:
447:         do K = 1, NBATCH*3*3
448:         COVXB(K,1,1) = 0.d0
449:         end do
450:
451:         do I = IST, NBATCH - 1
452:         XSS = dble(NBATCH-I+1)
453:         do K = 1, 3
454:         do L = 1, 3
455:         if ( L.ge.K ) then
456:
457:         do J = I, NBATCH
458:         X1 = 1.d0/(XSS-1.d0)
459:         COVXB(K,L,I) = COVXB(K,L,I)
460:         &         + X1*(XWBKB(K,J)-XBWKC(K,I))*
461:         &         (XWBKB(L,J)-XBWKC(L,I))
462:         end do
463:         else
464:         COVXB(K,L,I) = COVXB(L,K,I)
465:         end if
466:         end do
467:         end do
468:
469:         return
470:         end

```

src/mvp/blkiso.f

```

1:      subroutine BLKISO
2:      c=<MVP>=====
3:      c purpose : store the number of isotopes and its abundance and mass.
4:      c called in: main
5:      c calls : (none)
6:      c=====
7:      implicit real (a-h,o-z)
8:      include 'INC/_ISOTOP'
9:      c
10:     dimension TISOTP0(3,10,MATOM),ISOTOP0(MATOM)
11:     c
12:     c      mass and abundance of stable isotope
13:     data ISOTOP0
14:     & 2, 2, 2, 1, 2, 2, 2, 3, 1, 3,
15:     & 1, 3, 1, 3, 1, 4, 2, 3, 3, 6,
16:     & 1, 5, 2, 4, 1, 4, 1, 5, 2, 5,
17:     & 2, 5, 1, 6, 2, 6, 2, 4, 1, 5,
18:     & 1, 7, 0, 7, 1, 6, 2, 8, 2, 10,
19:     & 2, 8, 1, 9, 1, 7, 2, 4, 1, 7,
20:     & 0, 2, 2, 7, 1, 7, 1, 6, 1, 7,
21:     & 2, 6, 2, 5, 2, 7, 2, 6, 1, 7,
22:     & 2, 4, 1, 0, 0, 0, 0, 0, 0, 1,
23:     & 0, 3, 0, 0, 0, 0, 0, 0, 0, 0,
24:     & 0, 0, 0 /
25:     c
26:     data (((TISOTP0(i,j,k),i=1,3),j=1,10),k=1,20)
27:     &/ 1., 1.00782504, 0.99985,
28:     & 2., 2.0141018, 0.00015, 24*0.0,
29:     & 3., 3.0160293, 0.00000137,
30:     & 4., 4.0026032, 0.99999863, 24*0.0,
31:     & 6., 6.015121, 0.075,
32:     & 7., 7.016003, 0.925, 24*0.0,
33:     & 9., 9.012182, 1.0, 27*0.0,
34:     & 10., 10.012937, 0.199,
35:     & 11., 11.009305, 0.801, 24*0.0,
36:     & 12., 12.0, 0.9890,
37:     & 13., 13.0033548, 0.0110, 24*0.0,
38:     & 14., 14.0030740, 0.99634,
39:     & 15., 15.0001090, 0.00366, 24*0.0,
40:     & 16., 15.9949146, 0.99762,
41:     & 17., 16.999131, 0.00038,
42:     & 18., 17.999160, 0.00200, 21*0.0,
43:     & 19., 18.9984032, 1.0, 27*0.0,
44:     & 20., 19.99244, 0.9048,
45:     & 21., 20.99384, 0.0027,
46:     & 22., 21.99138, 0.0925, 21*0.0,
47:     & 23., 22.989768, 1.0, 27*0.0,
48:     & 24., 23.985042, 0.7899,
49:     & 25., 24.985837, 0.1000,
50:     & 26., 25.982594, 0.1101, 21*0.0,
51:     & 27., 26.981539, 1.0, 27*0.0,
52:     & 28., 27.976927, 0.9223,
53:     & 29., 28.976495, 0.0467,
54:     & 30., 29.973771, 0.0310, 21*0.0,
55:     & 31., 30.973762, 1.0, 27*0.0,
56:     & 32., 31.972071, 0.9502,
57:     & 33., 32.971458, 0.0075,
58:     & 34., 33.967867, 0.0421,
59:     & 36., 35.967081, 0.0002, 18*0.0,
60:     & 35., 34.9688527, 0.7577,
61:     & 37., 36.9659026, 0.2423, 24*0.0,
62:     & 36., 35.967546, 0.003365,
63:     & 38., 37.962733, 0.000632,
64:     & 40., 39.962384, 0.996003, 21*0.0,
65:     & 39., 38.963707, 0.932581,
66:     & 40., 39.963999, 0.000117,
67:     & 41., 40.961825, 0.067302, 21*0.0,
68:     & 40., 39.962591, 0.96941,
69:     & 42., 41.958618, 0.00647,
70:     & 43., 42.958766, 0.00135,
71:     & 44., 43.955481, 0.02086,
72:     & 46., 45.95369, 0.00004,
73:     & 48., 47.95253, 0.00187, 12*0.0 /
74:     data (((TISOTP0(i,j,k),i=1,3),j=1,10),k=21,40)
75:     &/ 45., 44.955910, 1.0, 27*0.0,
76:     & 46., 45.952629, 0.080,
77:     & 47., 46.951764, 0.073,
78:     & 48., 47.947947, 0.738,
79:     & 49., 48.947871, 0.055,
80:     & 50., 49.944792, 0.054, 15*0.0,
81:     & 50., 49.94716, 0.00250,
82:     & 51., 50.94396, 0.99750, 24*0.0,
83:     & 50., 49.94605, 0.04345,
84:     & 52., 51.94051, 0.83789,
85:     & 53., 52.94065, 0.09501,
86:     & 54., 53.93888, 0.02365, 18*0.0,
87:     & 55., 54.93805, 1.0, 27*0.0,
88:     & 54., 53.93961, 0.058,
89:     & 56., 55.93494, 0.9172,
90:     & 57., 56.93540, 0.022,
91:     & 58., 57.93328, 0.0028, 18*0.0,
92:     & 59., 58.93320, 1.0, 27*0.0,
93:     & 58., 57.93535, 0.68077,
94:     & 60., 59.93079, 0.26223,
95:     & 61., 60.93106, 0.01140,
96:     & 62., 61.92835, 0.03634,
97:     & 64., 63.92797, 0.00926, 15*0.0,
98:     & 63., 62.92960, 0.6917,
99:     & 65., 64.92779, 0.3083, 24*0.0,
100:    & 64., 63.92914, 0.486,
101:    & 66., 65.92603, 0.279,
102:    & 67., 66.92713, 0.041,
103:    & 68., 67.92485, 0.188,
104:    & 70., 69.92533, 0.006, 15*0.0,
105:    & 69., 68.92558, 0.60108,
106:    & 71., 70.92470, 0.39892, 24*0.0,
107:    & 70., 69.92425, 0.2123,
108:    & 72., 71.92208, 0.2766,
109:    & 73., 72.92346, 0.0773,
110:    & 74., 73.92118, 0.3594,
111:    & 76., 75.92140, 0.0744, 15*0.0,
112:    & 75., 74.92159, 1.0, 27*0.0,
113:    & 74., 73.92247, 0.0089,
114:    & 76., 75.91921, 0.0936,
115:    & 77., 76.91991, 0.0763,
116:    & 78., 77.91731, 0.2378,
117:    & 80., 79.91652, 0.4961,
118:    & 82., 81.91670, 0.0873, 12*0.0,
119:    & 79., 78.91834, 0.5069,
120:    & 81., 80.91629, 0.4931, 24*0.0,
121:    & 78., 77.92040, 0.0035,
122:    & 80., 79.91638, 0.0225,
123:    & 82., 81.91348, 0.116,
124:    & 83., 82.91414, 0.115,
125:    & 84., 83.91151, 0.570,
126:    & 86., 85.91062, 0.173, 12*0.0,
127:    & 85., 84.91179, 0.72165,
128:    & 87., 86.90919, 0.27835, 24*0.0,
129:    & 84., 83.91343, 0.0056,
130:    & 86., 85.90927, 0.0986,

```

src/mvp/blkiso.f

```
131: & 87., 86.90888, 0.0700, 196: & 124., 123.90589, 0.0010,
132: & 88., 87.90562, 0.8258, 18*0.0, 197: & 126., 125.90428, 0.0009,
133: & 89., 88.90585, 1.0, 27*0.0, 198: & 128., 127.90353, 0.0191,
134: & 90., 89.90470, 0.5145, 199: & 129., 128.90478, 0.264,
135: & 91., 90.90564, 0.1122, 200: & 130., 129.90351, 0.041,
136: & 92., 91.90504, 0.1715, 201: & 131., 130.90507, 0.212,
137: & 94., 93.90631, 0.1738, 202: & 132., 131.90414, 0.269,
138: & 96., 95.90828, 0.0280, 15*0.0 / 203: & 134., 133.90540, 0.104,
139: data (( (TISOTP0(i,j,k),i=1,3),j=1,10),k=41,60) 204: & 136., 135.90721, 0.089, 3*0.0,
140: &/ 93., 92.90638, 1.0, 27*0.0, 205: & 133., 132.90543, 1.0, 27*0.0,
141: & 92., 91.90681, 0.1484, 206: & 130., 129.90628, 0.00106,
142: & 94., 93.90509, 0.0925, 207: & 132., 131.90504, 0.00101,
143: & 95., 94.90584, 0.1592, 208: & 134., 133.90449, 0.02417,
144: & 96., 95.90468, 0.1668, 209: & 135., 134.90567, 0.06592,
145: & 97., 96.90602, 0.0955, 210: & 136., 135.90455, 0.07854,
146: & 98., 97.90541, 0.2413, 211: & 137., 136.90581, 0.1123,
147: & 100., 99.90748, 0.0963, 9*0.0, 212: & 138., 137.90523, 0.7170, 9*0.0,
148: & 30*0.0, 213: & 138., 137.90711, 0.000902,
149: & 96., 95.90760, 0.0552, 214: & 139., 138.90635, 0.999098, 24*0.0,
150: & 98., 97.90529, 0.0188, 215: & 136., 135.9071, 0.0019,
151: & 99., 98.90594, 0.127, 216: & 138., 137.90599, 0.0025,
152: & 100., 99.90422, 0.126, 217: & 140., 139.90543, 0.8848,
153: & 101., 100.90558, 0.170, 218: & 142., 141.90924, 0.1108, 18*0.0,
154: & 102., 101.90435, 0.316, 219: & 141., 140.90765, 1.0, 27*0.0,
155: & 104., 103.90542, 0.187, 9*0.0, 220: & 142., 141.90772, 0.2713,
156: & 103., 102.90550, 1.0, 27*0.0, 221: & 143., 142.90981, 0.1218,
157: & 102., 101.90563, 0.0102, 222: & 144., 143.91008, 0.2380,
158: & 104., 103.90403, 0.1114, 223: & 145., 144.91257, 0.0830,
159: & 105., 104.90508, 0.2233, 224: & 146., 145.91311, 0.1719,
160: & 106., 105.90348, 0.2733, 225: & 148., 147.91689, 0.0576,
161: & 108., 107.90390, 0.2646, 226: & 150., 149.92089, 0.0564, 9*0.0 /
162: & 110., 109.9052, 0.1172, 12*0.0, 227: data (( (TISOTP0(i,j,k),i=1,3),j=1,10),k=61,80)
163: & 107., 106.90509, 0.51839, 30*0.0,
164: & 109., 108.90476, 0.48161, 24*0.0, 228: &/
165: & 106., 105.90646, 0.0125, 229: & 144., 143.91200, 0.031,
166: & 108., 107.90418, 0.0089, 230: & 147., 146.91489, 0.150,
167: & 110., 109.90301, 0.1249, 231: & 148., 147.91482, 0.113,
168: & 111., 110.90418, 0.1280, 232: & 149., 148.91718, 0.138,
169: & 112., 111.90276, 0.2413, 233: & 150., 149.91727, 0.074,
170: & 113., 112.90440, 0.1222, 234: & 152., 151.91973, 0.267,
171: & 114., 113.90336, 0.2873, 235: & 154., 153.92221, 0.227, 9*0.0,
172: & 116., 115.90476, 0.0749, 6*0.0, 236: & 151., 150.91970, 0.478,
173: & 113., 112.90406, 0.043, 24*0.0, 237: & 153., 152.92123, 0.522,
174: & 115., 114.90388, 0.957, 238: & 152., 151.91979, 0.0020,
175: & 112., 111.90483, 0.0097, 239: & 154., 153.92086, 0.0218,
176: & 114., 113.90278, 0.0065, 240: & 155., 154.92262, 0.1480,
177: & 115., 114.90335, 0.0034, 241: & 156., 155.92212, 0.2047,
178: & 116., 115.90175, 0.1453, 242: & 157., 156.92396, 0.1565,
179: & 117., 116.90296, 0.0768, 243: & 158., 157.92401, 0.2484,
180: & 118., 117.90161, 0.2423, 244: & 160., 159.92705, 0.2186, 9*0.0,
181: & 119., 118.90331, 0.0859, 245: & 159., 158.92534, 1.0, 27*0.0,
182: & 120., 119.90220, 0.3259, 246: & 156., 155.92428, 0.0006,
183: & 122., 121.90344, 0.0463, 247: & 158., 157.92440, 0.0010,
184: & 124., 123.90527, 0.0579, 248: & 160., 159.92519, 0.0234,
185: & 121., 120.90382, 0.5736, 249: & 161., 160.92693, 0.189,
186: & 123., 122.90422, 0.4264, 250: & 162., 161.92680, 0.255,
187: & 120., 119.9040, 0.00096, 251: & 163., 162.92873, 0.249,
188: & 122., 121.90305, 0.02603, 252: & 164., 163.92917, 0.282, 9*0.0,
189: & 123., 122.90427, 0.00908, 253: & 165., 164.93032, 1.0, 27*0.0,
190: & 124., 123.90282, 0.04816, 254: & 162., 161.92878, 0.0014,
191: & 125., 124.90443, 0.07139, 255: & 164., 163.92920, 0.0161,
192: & 126., 125.90331, 0.18952, 256: & 166., 165.93029, 0.336,
193: & 128., 127.90446, 0.31687, 257: & 167., 166.93205, 0.2295,
194: & 130., 129.90623, 0.33799, 6*0.0, 258: & 168., 167.93237, 0.268,
195: & 127., 126.90447, 1.0, 27*0.0, 259: & 170., 169.93546, 0.149, 12*0.0,
260: & 169., 168.93421, 1.0, 27*0.0,
```

src/mvp/blkiso.f

```

261:      & 168., 167.93389, 0.0013,
262:      & 170., 169.93476, 0.0305,
263:      & 171., 170.93632, 0.143,
264:      & 172., 171.93638, 0.219,
265:      & 173., 172.93821, 0.1612,
266:      & 174., 173.93886, 0.318,
267:      & 176., 175.94256, 0.127, 9*0.0,
268:      & 175., 174.94077, 0.9741,
269:      & 176., 175.94268, 0.0259, 24*0.0,
270:      & 174., 173.94004, 0.00162,
271:      & 176., 175.94141, 0.05206,
272:      & 177., 176.94322, 0.18606,
273:      & 178., 177.94370, 0.27297,
274:      & 179., 178.94581, 0.13629,
275:      & 180., 179.94655, 0.35100, 12*0.0,
276:      & 180., 179.94746, 0.00012,
277:      & 181., 180.94799, 0.99988, 24*0.0,
278:      & 180., 179.94670, 0.0013,
279:      & 182., 181.94820, 0.263,
280:      & 183., 182.95022, 0.143,
281:      & 184., 183.95093, 0.3067,
282:      & 186., 185.95436, 0.286, 15*0.0,
283:      & 185., 184.95295, 0.3740,
284:      & 187., 186.95574, 0.8260, 24*0.0,
285:      & 184., 183.95249, 0.0002,
286:      & 186., 185.95383, 0.0158,
287:      & 187., 186.95574, 0.016,
288:      & 188., 187.95583, 0.133,
289:      & 189., 188.95814, 0.161,
290:      & 190., 189.95844, 0.264,
291:      & 192., 191.96147, 0.410, 9*0.0,
292:      & 191., 190.96058, 0.373,
293:      & 193., 192.96292, 0.627, 24*0.0,
294:      & 190., 189.95992, 0.0001,
295:      & 192., 191.96102, 0.0079,
296:      & 194., 193.96266, 0.329,
297:      & 195., 194.96477, 0.338,
298:      & 196., 195.96493, 0.253,
299:      & 198., 197.96787, 0.072, 12*0.0,
300:      & 197., 196.96654, 1.0, 27*0.0,
301:      & 196., 195.96581, 0.0015,
302:      & 198., 197.96674, 0.0997,
303:      & 199., 198.96825, 0.1687,
304:      & 200., 199.96830, 0.2310,
305:      & 201., 200.97028, 0.1318,
306:      & 202., 201.97062, 0.2986,
307:      & 204., 203.97347, 0.0687, 9*0.0 /
308:      data (( (TISOTP0(i,j,k),i=1,3),j=1,10),k=81,103)
309:      &/ 203., 202.97232, 0.29524,
310:      & 205., 204.97440, 0.70476, 24*0.0,
311:      & 204., 203.97302, 0.014,
312:      & 206., 205.97444, 0.241,
313:      & 207., 206.97587, 0.221,
314:      & 208., 207.97663, 0.524, 18*0.0,
315:      & 209., 208.98037, 1.0, 27*0.0,
316:      & , 30*0.0,
317:      & , 30*0.0,
318:      & , 30*0.0,
319:      & , 30*0.0,
320:      & , 30*0.0,
321:      & , 30*0.0,
322:      & 232., 232.03805, 1.0, 27*0.0,
323:      & , 30*0.0,
324:      & 234., 234.04095, 0.000055,
325:      & 235., 235.04392, 0.007200,

```

```

326:      & 238., 238.05078, 0.992745, 21*0.0,
327:      & , 30*0.0,
328:      & , 30*0.0,
329:      & , 30*0.0,
330:      & , 30*0.0,
331:      & , 30*0.0,
332:      & , 30*0.0,
333:      & , 30*0.0,
334:      & , 30*0.0,
335:      & , 30*0.0,
336:      & , 30*0.0,
337:      & , 30*0.0 /
338: c -----
339: c
340: c
341:      do I = 1, MATOM
342:          ISOTOP(I) = ISOTOP0(I)
343: c
344:          do J = 1, 10
345:              TISOTP(1,J,I) = TISOTP0(1,J,I)
346:              TISOTP(2,J,I) = TISOTP0(2,J,I)
347:              TISOTP(3,J,I) = TISOTP0(3,J,I)
348:          end do
349:      end do
350:      return
351:      end

```

src/mvp/bprogv.f

```
1:      subroutine BPROGV
2: C=<MVP>=====
3: C purpose: initialization of "PROGV" strings (program version
4: C      description)
5: C=====
6: C*
7:      include 'INC/_PROGV'
8: C
9:      character*64 PROGVO(40)
10: C
11: C 8' TRACKING PROCEDURE : Event selection or zone selection. ',
12: C
13: C data (PROGVO(I),I=1,15)/
14: C 1' MVP : JAEA Monte Carlo transport code
15: C 2' with continuous energy cross section library
16: C 3'
17: C 4'
18: C *** Main capability of MVP ***
19: C 6'
20: C 7' PROBLEM TYPE : Fixed source & eigenvalue.
21: C 7' Neutron & photon calculation.
22: C 9' CROSS SECTION : Continuous energy cross section
23: C 9' Arbitrary temperature neutron lib.
24: C A' GEOMETRY : Combinatorial Geometry with
25: C 1' rectangular & hexagonal lattice.
26: C 2' GEOMETRY CHECK : 2-D slice image with characters.
27: C 3' FIXED SOURCE : Programmable & flexible input form
28: C 4' with many type builtin sampling method.
29: C DATA (PROGVO(I),I=16,40)/
30: C 5' ESTIMATORS : Collision & track length estimator.
31: C 6' BOUNDARY CONDITION : Vacuum, perfect reflection &
32: C 7' white reflection & periodic.
33: C 8'
34: C 9' MATERIAL NUMBER ASSIGNMENT :
35: C A'
36: C 1' .GE. 1 : REAL MEDIUM.
37: C 2' 0 : INNER VOID.
38: C 3' -1 TO -998 : LATTICE.
39: C 4' -1000 : OUTER VOID.
40: C 5' -1001 TO -1999 : ALBEDO MATERIAL (NOT IN USE)
41: C 6' -2000 : PERFECT REFLECTOR (MIRROR)
42: C 7' -3000 : WHITE REFLECTOR.
43: C 8' -4000 : PERIODIC BOUNDARY.
44: C 9'
45: C A' FILES : RESTART DATA READ FROM I/O UNIT 10
46: C 1' WRITTEN ON I/O UNIT 20
47: C 2' BINARY OUTPUT WRITTEN ON I/O UNIT 30
48: C E 6* ' ',
49: C F'RESTART FILE TYPE 3.5 (JUL 2005)
50: C F'RESTART FILE TYPE 3.4 (MAR 2003)
51: C F'RESTART FILE TYPE 3.3 (AUG 2003)
52: C F'RESTART FILE TYPE 3.2 (FEB 1999)
53: C F'RESTART FILE TYPE 3.1 (OCT 1998)
54: C F'RESTART FILE TYPE 3.0 (MAY 1997)
55: C Cccc F'RESTART FILE TYPE 2.1 (MAY 1994)
56: C CCCCCC 40'TH LINE OF PROGV IS USED AS RESTART FILE INFORMATION
57: C
58: C NLines = 40
59: C do 100 I=1,NLines
60: C PROGVO(I) = PROGVO(I)
61: C 100 continue
62: C
63: C return
64: C end
```


src/mvp/cadenz.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine CADENZ( TITLE, A, H, CHA, LIMIT, LIMITL, LIMITC )
3: C=<MVP>=====
4: C  PURPOSE:  PROCESSING AFTER MONTE CARLO RUN
5: C  CALLED IN: CENTER
6: C  CALLS:  KEPV,TALLYO,PRMNTR
7: C=====
8: C
9:   real A(*)
10:  real H(*)
11:  character*4 CHA(*)
12: C
13:   include '../shared/INC/_LISTOFF'
14:   include '../shared/INC/_TASKDT'
15: C
16:   include 'INC/_KPIDS'
17:   include 'INC/_NGPS'
18: C
19:   include 'INC/_FLAGS'
20:   include '../shared/INC/_SIZES'
21:   include 'INC/_SIZES2'
22:   include '../shared/INC/_PGEOM'
23:   include 'INC/_XWORK'
24:   include 'INC/_CXSEC'
25:   include 'INC/_PXSEC' ! photo-nuc
26:   include '../shared/INC/_PTALY0'
27:   include 'INC/_PTALY'
28:   include 'INC/_PTALY2'
29:   include '../shared/INC/_STALY'
30:   include '../shared/INC/_PTLSP'
31:   include 'INC/_XBANK'
32:   include 'INC/_STACK'
33:   include 'INC/_SBANK'
34: C
35:   include 'INC/_WKTLO'
36:   include '../shared/INC/_LISTON'
37: C
38:   character TITLE(2)*72
39:   include '../shared/INC/_WORDL'
40:
41:   include 'INC/_PERT' ! pert
42: C
43: C .....IOT:  PRINTOUT UNIT OF CADENZ STEP ...
44:   IOT      = 6
45: C
46: C ... do tally output etc. in customized version.
47: C
48:   call ZZTALO( H(LWSUM), IOT, A, A, H, H, CHA )
49: C
50: C ..... PRINT OUT OF TALLY DATA .....
51: C
52:   LW      = LWORK
53: C
54:   call TALLYO( TITLE, A, CHA,
55: &      NTHIST, NGROUP, NGP1, NGP2, NPKIND, NREG,
56: &      NRESP, NBATCH, NUC, NPATOM, NEMIC, NEMAC, NTREG, NSTAL,
57: &      NSKIP, NHIST, NTASK, NEVENT, H(LWSUM), H(LWLEK),
58: &      H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL), A(LENGYB),
59: &      A(LENGPB), A(LRVOL), A(LTRVOL), H(LWCNTR), H(LXSXV),
60: &      A(LDNREG), H(LWCXTY), H(LSRMIC), H(LRMICR), H(LXMIC),
61: &      H(LRMAC), H(LRMACR), H(LXMAC), A(LLEMIC), A(LLEMAC),
62: &      CHA(LNUCID), A(LTEMPN),
63: &      H(LSRSTR), H(LSRACL), H(LTFLHS), A(LITRNM),
64: &      CHA(LTNAMS), NNAME, NSTALY, A(LTIMEB), NTIME,
65: &      H(LSRMICSM),H(LXMICSM),H(LRMACSM),H(LXMACSM),

```

```

66: &      H(LSMIMU), H(LRMAMU),
67: &      H(P9(9)), H(P9(10)) )
68: C
69:   if ( JEIGN.ne.0 ) then
70:     call KEFF( TITLE, JPRTS, NBATCH, NSKIP, NMXLG, NEITER,
71: &      H(LWLEK), H(LWCNTR),
72: &      H(LXSOC), H(LXKEF), H(P9(1)), H(P9(2)), H(P9(3)),
73: &      H(P9(4)), H(P9(5)), H(P9(6)), H(P9(7)),
74: &      H(P9(13)),H(P9(14)),H(P9(15)),H(P9(16)),
75: &      JPHNU )
76:
77:   if ( JPRT.ne.0 ) then
78:     call KEFFPT(TITLE, JPRTS, NBATCH, NSKIP, NPTDS, NPTCS, MAXDA,
79: &      H(LXSOC), H(LXKEFP), H(P9(1)), H(P9(2)), H(P9(21)),
80: &      H(P9(22)), H(P9(23)), A(LDELA), NUCPT, NUC, A(LJPTNU),
81: &      CHA(LNUCID), JPRT, JPDEN, NTPT, NGSP, NOCS, NORDDS, NPTBE,
82: &      MXPTOP, A(LJPTOP))
83:   end if
84:
85:   if( JBEFF.ne.0 ) then
86:     call BEFF(JPRTS, NBATCH, NSKIP, NEBEF,
87: &      H(LWCNTR), H(LXSOC), H(LXBEF), H(P9(1)), H(P9(2)),
88: &      H(P9(24)), H(P9(25)), H(P9(26)), H(P9(27)), H(P9(28)),
89: &      H(P9(29)), H(P9(30)), H(P9(31)))
90:   end if
91: end if
92: C
93:   call TALPRT( TITLE, A, CHA,
94: &      NTHIST, NGROUP, NPKIND, NREG,
95: &      NRESP, NBATCH, NUC, NPATOM, NEMIC, NEMAC, NTREG, NSTAL,
96: &      NSKIP, NHIST, NTASK, NEVENT, H(LWSUM), H(LWLEK),
97: &      H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL), A(LENGYB),
98: &      A(LRVOL), A(LTRVOL), H(LWCNTR), H(LSRMIC),
99: &      H(LRMICR), H(LXMIC), H(LRMAC), H(LRMACR), H(LXMAC),
100: &      A(LLEMIC), A(LLEMAC), CHA(LNUCID), H(LSRSTR), H(LSRACL),
101: &      H(LTFLHS), A(LITRNM), CHA(LTNAMS), NNAME, H(P9(8)),
102: &      NGP(KPNEUT),H(LSRMICSM),H(LXMICSM),H(LRMACSM),H(LXMACSM),
103: &      H(LSMIMU), H(LRMAMU),
104: c##<2007/03/14:PN3:
105: &      H(P9(9)) )
106: c##>
107: C
108: C ..... Output of special tally data .....
109: C
110:   if ( NSTALY.gt.0 ) then
111:     call STLOUT( H(LIETAL),NIETAL,NETALY,H(LETALY),NLETAL,
112: &      JEIGN, NBATCH, NSKIP, H(LWSUM),
113: &      A(LENGYB), KENGP, KPLIM, A(LTIMEB), A(LANGLB),
114: &      NETRV, METRV, NBETRV,A(LIETRV), H(LETRV), NMXLG,NEITER,
115: &      H(P9(11)), CHA(LRNM),NBINMX,NEDTMX,MXRGBN,
116: c##<2007/03/14:PN3:
117: &      H(P9(15)),H(P9(16)))
118: &      H(P9(15)),H(P9(16)),
119: &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, CHA(LNUCID),
120: &      A(LNATMT), CHA(LNCIDPN), KPNPRD )
121: c##>
122:   end if
123: C
124: C ..... PRINT OUT OF MONITORING DATA .....
125: C
126:   call PRMNTR( IOT,A,CHA,JMNTR,NGROUP,NREG, JTIME, JNEUT, JPHOT,
127: &      JDLYN, NEVENT, H(LNCLSN), H(LWCLSN), H(LNLEAK), H(LWLEAK),
128: &      H(LNABSB), H(LWABSB), H(LNECUT), H(LWECUT), H(LNLCUT),
129: &      H(LWTCUT), H(LNKILD), H(LWKILD), H(LNSURV), H(LWSURV),
130: c##<2007/03/14:PN3:

```

src/mvp/cadenz.f

```
131: c## &      H(LNCNTR), H(LWCNTR) )
132:      &      H(LNCNTR), H(LWCNTR),
133:      &      JPHNU, NMONPNW, A(LMONPNW), H(LNMPNWGT), H(LWMPNWGT) )
134: c##>
135: C
136: C
137:      return
138:      end
```

SAE

```

center.f
C
TOTAL,SCALAR
C
PARAMETER(NTASK0=1,NTASK1=1,NTASK2=1,NTASK3=1,NTASK4=1,NTASK5=1,NTASK6=1,NTASK7=1,NTASK8=1,NTASK9=1,NTASK10=1,NTASK11=1,NTASK12=1,NTASK13=1,NTASK14=1,NTASK15=1,NTASK16=1,NTASK17=1,NTASK18=1,NTASK19=1,NTASK20=1,NTASK21=1,NTASK22=1,NTASK23=1,NTASK24=1,NTASK25=1,NTASK26=1,NTASK27=1,NTASK28=1,NTASK29=1,NTASK30=1,NTASK31=1,NTASK32=1,NTASK33=1,NTASK34=1,NTASK35=1,NTASK36=1,NTASK37=1,NTASK38=1,NTASK39=1,NTASK40=1,NTASK41=1,NTASK42=1,NTASK43=1,NTASK44=1,NTASK45=1,NTASK46=1,NTASK47=1,NTASK48=1,NTASK49=1,NTASK50=1,NTASK51=1,NTASK52=1,NTASK53=1,NTASK54=1,NTASK55=1,NTASK56=1,NTASK57=1,NTASK58=1,NTASK59=1,NTASK60=1,NTASK61=1,NTASK62=1,NTASK63=1,NTASK64=1,NTASK65=1,NTASK66=1,NTASK67=1,NTASK68=1,NTASK69=1,NTASK70=1,NTASK71=1,NTASK72=1,NTASK73=1,NTASK74=1,NTASK75=1,NTASK76=1,NTASK77=1,NTASK78=1,NTASK79=1,NTASK80=1,NTASK81=1,NTASK82=1,NTASK83=1,NTASK84=1,NTASK85=1,NTASK86=1,NTASK87=1,NTASK88=1,NTASK89=1,NTASK90=1,NTASK91=1,NTASK92=1,NTASK93=1,NTASK94=1,NTASK95=1,NTASK96=1,NTASK97=1,NTASK98=1,NTASK99=1,NTASK100=1,NTASK101=1,NTASK102=1,NTASK103=1,NTASK104=1,NTASK105=1,NTASK106=1,NTASK107=1,NTASK108=1,NTASK109=1,NTASK110=1,NTASK111=1,NTASK112=1,NTASK113=1,NTASK114=1,NTASK115=1,NTASK116=1,NTASK117=1,NTASK118=1,NTASK119=1,NTASK120=1,NTASK121=1,NTASK122=1,NTASK123=1,NTASK124=1,NTASK125=1,NTASK126=1,NTASK127=1,NTASK128=1,NTASK129=1,NTASK130=1,NTASK131=1,NTASK132=1,NTASK133=1,NTASK134=1,NTASK135=1,NTASK136=1,NTASK137=1,NTASK138=1,NTASK139=1,NTASK140=1,NTASK141=1,NTASK142=1,NTASK143=1,NTASK144=1,NTASK145=1,NTASK146=1,NTASK147=1,NTASK148=1,NTASK149=1,NTASK150=1,NTASK151=1,NTASK152=1,NTASK153=1,NTASK154=1,NTASK155=1,NTASK156=1,NTASK157=1,NTASK158=1,NTASK159=1,NTASK160=1,NTASK161=1,NTASK162=1,NTASK163=1,NTASK164=1,NTASK165=1,NTASK166=1,NTASK167=1,NTASK168=1,NTASK169=1,NTASK170=1,NTASK171=1,NTASK172=1,NTASK173=1,NTASK174=1,NTASK175=1,NTASK176=1,NTASK177=1,NTASK178=1,NTASK179=1,NTASK180=1,NTASK181=1,NTASK182=1,NTASK183=1,NTASK184=1,NTASK185=1,NTASK186=1,NTASK187=1,NTASK188=1,NTASK189=1,NTASK190=1,NTASK191=1,NTASK192=1,NTASK193=1,NTASK194=1,NTASK195=1,NTASK196=1,NTASK197=1,NTASK198=1,NTASK199=1,NTASK200=1,NTASK201=1,NTASK202=1,NTASK203=1,NTASK204=1,NTASK205=1,NTASK206=1,NTASK207=1,NTASK208=1,NTASK209=1,NTASK210=1,NTASK211=1,NTASK212=1,NTASK213=1,NTASK214=1,NTASK215=1,NTASK216=1,NTASK217=1,NTASK218=1,NTASK219=1,NTASK220=1,NTASK221=1,NTASK222=1,NTASK223=1,NTASK224=1,NTASK225=1,NTASK226=1,NTASK227=1,NTASK228=1,NTASK229=1,NTASK230=1,NTASK231=1,NTASK232=1,NTASK233=1,NTASK234=1,NTASK235=1,NTASK236=1,NTASK237=1,NTASK238=1,NTASK239=1,NTASK240=1,NTASK241=1,NTASK242=1,NTASK243=1,NTASK244=1,NTASK245=1,NTASK246=1,NTASK247=1,NTASK248=1,NTASK249=1,NTASK250=1,NTASK251=1,NTASK252=1,NTASK253=1,NTASK254=1,NTASK255=1,NTASK256=1,NTASK257=1,NTASK258=1,NTASK259=1,NTASK260=1,NTASK261=1,NTASK262=1,NTASK263=1,NTASK264=1,NTASK265=1,NTASK266=1,NTASK267=1,NTASK268=1,NTASK269=1,NTASK270=1,NTASK271=1,NTASK272=1,NTASK273=1,NTASK274=1,NTASK275=1,NTASK276=1,NTASK277=1,NTASK278=1,NTASK279=1,NTASK280=1,NTASK281=1,NTASK282=1,NTASK283=1,NTASK284=1,NTASK285=1,NTASK286=1,NTASK287=1,NTASK288=1,NTASK289=1,NTASK290=1,NTASK291=1,NTASK292=1,NTASK293=1,NTASK294=1,NTASK295=1,NTASK296=1,NTASK297=1,NTASK298=1,NTASK299=1,NTASK300=1,NTASK301=1,NTASK302=1,NTASK303=1,NTASK304=1,NTASK305=1,NTASK306=1,NTASK307=1,NTASK308=1,NTASK309=1,NTASK310=1,NTASK311=1,NTASK312=1,NTASK313=1,NTASK314=1,NTASK315=1,NTASK316=1,NTASK317=1,NTASK318=1,NTASK319=1,NTASK320=1,NTASK321=1,NTASK322=1,NTASK323=1,NTASK324=1,NTASK325=1,NTASK326=1,NTASK327=1,NTASK328=1,NTASK329=1,NTASK330=1,NTASK331=1,NTASK332=1,NTASK333=1,NTASK334=1,NTASK335=1,NTASK336=1,NTASK337=1,NTASK338=1,NTASK339=1,NTASK340=1,NTASK341=1,NTASK342=1,NTASK343=1,NTASK344=1,NTASK345=1,NTASK346=1,NTASK347=1,NTASK348=1,NTASK349=1,NTASK350=1,NTASK351=1,NTASK352=1,NTASK353=1,NTASK354=1,NTASK355=1,NTASK356=1,NTASK357=1,NTASK358=1,NTASK359=1,NTASK360=1,NTASK361=1,NTASK362=1,NTASK363=1,NTASK364=1,NTASK365=1,NTASK366=1,NTASK367=1,NTASK368=1,NTASK369=1,NTASK370=1,NTASK371=1,NTASK372=1,NTASK373=1,NTASK374=1,NTASK375=1,NTASK376=1,NTASK377=1,NTASK378=1,NTASK379=1,NTASK380=1,NTASK381=1,NTASK382=1,NTASK383=1,NTASK384=1,NTASK385=1,NTASK386=1,NTASK387=1,NTASK388=1,NTASK389=1,NTASK390=1,NTASK391=1,NTASK392=1,NTASK393=1,NTASK394=1,NTASK395=1,NTASK396=1,NTASK397=1,NTASK398=1,NTASK399=1,NTASK400=1,NTASK401=1,NTASK402=1,NTASK403=1,NTASK404=1,NTASK405=1,NTASK406=1,NTASK407=1,NTASK408=1,NTASK409=1,NTASK410=1,NTASK411=1,NTASK412=1,NTASK413=1,NTASK414=1,NTASK415=1,NTASK416=1,NTASK417=1,NTASK418=1,NTASK419=1,NTASK420=1,NTASK421=1,NTASK422=1,NTASK423=1,NTASK424=1,NTASK425=1,NTASK426=1,NTASK427=1,NTASK428=1,NTASK429=1,NTASK430=1,NTASK431=1,NTASK432=1,NTASK433=1,NTASK434=1,NTASK435=1,NTASK436=1,NTASK437=1,NTASK438=1,NTASK439=1,NTASK440=1,NTASK441=1,NTASK442=1,NTASK443=1,NTASK444=1,NTASK445=1,NTASK446=1,NTASK447=1,NTASK448=1,NTASK449=1,NTASK450=1,NTASK451=1,NTASK452=1,NTASK453=1,NTASK454=1,NTASK455=1,NTASK456=1,NTASK457=1,NTASK458=1,NTASK459=1,NTASK460=1,NTASK461=1,NTASK462=1,NTASK463=1,NTASK464=1,NTASK465=1,NTASK466=1,NTASK467=1,NTASK468=1,NTASK469=1,NTASK470=1,NTASK471=1,NTASK472=1,NTASK473=1,NTASK474=1,NTASK475=1,NTASK476=1,NTASK477=1,NTASK478=1,NTASK479=1,NTASK480=1,NTASK481=1,NTASK482=1,NTASK483=1,NTASK484=1,NTASK485=1,NTASK486=1,NTASK487=1,NTASK488=1,NTASK489=1,NTASK490=1,NTASK491=1,NTASK492=1,NTASK493=1,NTASK494=1,NTASK495=1,NTASK496=1,NTASK497=1,NTASK498=1,NTASK499=1,NTASK500=1,NTASK501=1,NTASK502=1,NTASK503=1,NTASK504=1,NTASK505=1,NTASK506=1,NTASK507=1,NTASK508=1,NTASK509=1,NTASK510=1,NTASK511=1,NTASK512=1,NTASK513=1,NTASK514=1,NTASK515=1,NTASK516=1,NTASK517=1,NTASK518=1,NTASK519=1,NTASK520=1,NTASK521=1,NTASK522=1,NTASK52
```

ng ...

```

66:      &' _/      _/      _/      _/      '/'
67:
68: C
69: C ... initiate multitasking mode ( PVM/MPI/SX/CRAY/VPP )
70: C
71: C/#IF PARA( PVM )
72: *      call TSKPVM( NTASK0, TSKINF )
73: C/#ELSEIF PARA( MPI )
74: *      call TSKMPI( NTASK0, TSKINF )
75: C/#ELSEIF PARA( SX* CRAY* )
76: *      ITASK0 = 1
77: *      if( NTASK0 .eq. 0 ) NTASK0 = 1
78: *      IDTASK = 1
79: C/#ELSEIF PARA( VPP )
80: *      ITASK0 = 1
81: *      NTASK0 = MPE
82: *      IDTASK = IDVPROC()
83: C/#ELSE
84: *      ITASK0 = 1
85: *      NTASK0 = 1
86: *      IDTASK = 1
87: C/#ENDIF
88: C
89: C ..... PRINT DATE, TIME & PROGRAM DESCRIPTION .....
90: C
91:      call HIZUKE( XDATE )
92:      call JIKAN( TIME8 )
93:      call TOKEI( T00, 1 )
94:      call CPUTM( TC0 )
95: C
96:      call VERSTR
97:      call GETVER( VER, RDATE )
98: C
99:      BLANK = ' '
100:      write(IPR,'((8X,A,A))') (BLANK(:8-I),MVP(i),I=1,7)
101: C
102:      write(IPR,7000) VER(:ICLEN2(VER)), RDATE(:ICLEN2(RDATE)),
103:      &      XDATE,TIME8
104: 7000 format(/8X,' VERSION : ',A, '      RELEASE DATE : ',A/
105:      &      8X,' EXECUTION DATE : ',A,'      TIME : ',A/)
106: C
107: C === PRINT COMMENTS TO PROGRAM ( VERSION, OPTIONS ETC. )
108: C      (main task only or if SUBTASK-PRINT(1)==2)
109: C
110: C << STYLE >>
111: C
112: C *****
113: C * * * * *
114: C *      PROGV(1) *
115: C *      PROGV(2) *
116: C *      .... *
117: C * * * * *
118: C *****
119: C
120: C if ( JSTPRN(1).ge.2.or.IDTASK.eq.1 ) then
121: C      write(IPR,7010)
122: 7010 C      format(8X,'*',7('*****'),'*')
123: C
124: C      write(IPR, '(8X,'''',5X,A60,5X,'''')')
125: C      &      ' '
126: C      &      (PROGV(I),I=1,40),
127: C      &      ' '
128: C
129: C      write(IPR,7010)
130: C      end if

```

src/mvp/center.f

```

131: C
132: C ==== OUTPUT RUNNIG SYSTEM & MODE etc. =====
133: C
134:      call STAMP
135: C
136: C/#IF PARA(VPP)
137: *      write(IPR,'(30X,' 'THE NUMBER OF PROCESSORS:' ',I3)' ) NTASK0
138: *      call GETTOD(TT0)
139: C/#ENDIF
140: C
141: C      ... printout I/O unit management for sub tasks
142: C
143:      call STPRN1('ON',JSTPRN,IDTASK)
144: C
145: C -----
146: C ..... PREPARATION FOR MONTE CARLO RUN
147: C -----
148: C
149:      call INTRO( TITLE, NTASK0, MLIMIT )
150: C
151: C      ... disable printout I/O unit management for sub tasks
152: C
153:      call STPRN1('OFF',JSTPRN,IDTASK)
154: C
155: C
156: C/#IF PARA(VPP)
157: *      call GETTOD(TT1)
158: *      TMINTR = TT1 - TT0
159: C/#ENDIF
160: C
161: C/#IF PARA(MPI PVM)
162:      call MVPCOM_BARRIER(IERR)
163: C/#ENDIF
164: C/#IF UNIX
165:      call FLUSHSTD( )
166: C/#ENDIF
167: C
168: C -----
169: C ..... MONTE CARLO RUN
170: C -----
171: C
172: C ----- For Shared Memory Multi Task run -----
173: C
174: C/#IF PARA( CRAY* SX* )
175: *
176: *      if( JSTOP.eq.0 ) then
177: *          call MTTASK( TITLE, A, H, CHA, IAINFL, LIMITL,
178: *              & A(LITASK),NTASK,MNTASK,ITASK0,
179: *              & A(LIRANT),ILOCK, IBARR, A(LIRSUT), A(LIRSLT) )
180: *      end if
181: *
182: C
183: C ----- For VPP Fortran -----
184: C
185: C/#ELSEIF PARA(VPP)
186: C
187:      call ACTVPP( TITLE, A, H, CHA )
188: C
189: C/#ELSEIF PARA(PVM MPI)
190: C
191: C ----- For Distributed Memory MultiTask -----
192: C
193:      if( JSTOP.eq.0 ) then
194:          call ACTMPP( TITLE, A, H, CHA )
195:      end if

```

```

196: C
197: C/#ELSE
198: C
199: C ----- For Single Task -----
200: C
201:      if( JSTOP.eq.0 ) then
202:          call ACTSGL( TITLE, A, H, CHA )
203:      end if
204: C
205: C/#ENDIF
206: C
207: C -----
208: C ..... TALLY DATA ANALYSIS & OUTPUT .....
209: C -----
210: C
211: C
212:      if( JSTOP.eq.0 ) then
213:
214: C/#IF PARA(VPP)
215:          call GETTOD(TT0)
216: C/#ENDIF
217:
218:          call CADENZ( TITLE, A, H, CHA, LIMIT, LIMITL, LIMITC )
219:
220: C/#IF PARA(VPP)
221:          call GETTOD(TT1)
222:          TMCDNZ = TT1 - TT0
223: C/#ENDIF
224:
225:      end if
226: C
227:      call TOKEI( TT1, 0 )
228:      call CPUTM( TC1 )
229:      TEL1 = TT1-TT0
230:      TCP1 = TC1-TC0
231:
232:      write(IPR,7200) TEL1,
233:      & int(TEL1/3600.0),int(mod(TEL1,3600.0)/60.0),int(mod(TEL1,60.0)),
234:      & TCP1,
235:      & int(TCP1/3600.0),int(mod(TCP1,3600.0)/60.0),int(mod(TCP1,60.0))
236:
237: 7200 format(/5x,' === Elapsed time : ',1p,e12.5,' sec. ',
238:      & '(',i4,' hours ',i2,' min ',i2,' sec ) ==='
239:      & //5x,' === CPU time : ',1p,e12.5,' sec. ',
240:      & '(',i4,' hours ',i2,' min ',i2,' sec ) ===')
241:
242:      if( TEL1.ne.0.0 ) then
243:          write(IPR,7210) TCP1/TEL1*100.0
244:          format( 5x,'                                (' ,f10.4,'%')' )
245:      end if
246: C
247: C/#IF SYSTEM( FACOMAP3K )
248: *      call MVPCOM_EXIT( INFO )
249: C/#ENDIF
250:      return
251:      end

```

src/mvp/check.f

```

1:      SUBROUTINE CHECK( COMENT )
2: C=====
3: C PURPOSE : CHECK WRITE OF ALL COMMON PARAMETERS
4: C CALLED IN :  VARIOUS ROUTINES
5: C=====
6:      include '../shared/INC/_LISTOFF'
7:      include '../shared/INC/_IOUNIT'
8:      include '../shared/INC/_SIZES'
9:      include 'INC/_SIZES2'
10: C
11:      include 'INC/_KPIDS'
12:      include 'INC/_NGPS'
13: C
14:      include 'INC/_FLAGS'
15:      include '../shared/INC/_PGEOM'
16:      include 'INC/_PMNTR'
17:      include 'INC/_PSOUR'
18:      include '../shared/INC/_PTALY0'
19:      include 'INC/_PTALY'
20:      include 'INC/_PTALY2'
21:      include '../shared/INC/_PVRED'
22:      include 'INC/_PXSEC'
23:      include 'INC/_CXSEC'
24:      include 'INC/_STACK'
25:      include 'INC/_XBANK'
26:      include 'INC/_SBANK'
27:      include 'INC/_XWORK'
28:      include '../shared/INC/_LISON'
29: CHARACTER*(*) COMENT
30: WRITE(IPR, '(/1x, ''+++++ CHECK WRITE !! +++++''/1x ,A/)' ) COMENT
31: C-----
32: C .... SIZE & LIMIT PARAMETERS
33: C-----
34:      WRITE(IPR,*) ' ==== COMMON /SIZES/ ===== '
35: cccc WRITE(IPR,7000)
36:      WRITE(IPR,*)
37:      & 'NPART ',NPART, 'NHIST ',NHIST, 'NBANK ',NBANK,
38:      & 'NGROUP',NGROUP, 'NZONE ',NZONE, 'NINPZ ',NINPZ,
39:      & 'NSURF ',NSURF, 'NSDA ',NSDA, 'NZDA ',NZDA,
40:      & 'NMEMO ',NMEMO, 'NMAT ',NMAT, 'NREG ',NREG,
41:      & 'NTIME ',NTIME, 'NSOUR ',NSOUR,
42:      & 'NRESP ',NRESP, 'TCPU ',TCPU, 'IRAND ',IRAND,
43:      & 'NTHIST',NTHIST, 'ECUT ',ECUT, 'TCUT ',TCUT,
44:      & 'SUPPLY',SUPPLY, 'NFBANK',NFBANK, 'NSKIP ',NSKIP,
45:      & 'NBATCH',NBATCH, 'NPICT ',NPICT, 'NREFS ',NREFS,
46:      & 'NMEMS ',NMEMS, 'NMEMOP',NMEMOP, 'NLATT ',NLATT,
47:      & 'NLLEV ',NLLEV, 'NCELL ',NCELL, 'NEST ',NEST,
48:      & 'NLBZ ',NLBZ, 'NRSKIP',NRSKIP
49: c7000 FORMAT(1x ,5(1x,A6,1x,G9.3))
50: C-----
51: C .... OPTION FLAGS
52: C-----
53:      WRITE(IPR,*) ' ===== COMMON /FLAGS / ===== '
54:      WRITE(IPR,7100)
55:      & 'JREST',JREST, 'JNEUT',JNEUT, 'JPHOT',JPHOT, 'JIMAG',JIMAG,
56:      & 'JTIME',JTIME, 'JDELT',JDELT, 'JFISS',JFISS, 'JEIGN',JEIGN,
57:      & 'JFIXD',JFIXD, 'JADJM',JADJM, 'JDDX ',JDDX, 'JRRLT',JRRLT,
58:      & 'JIMPT',JIMPT, 'JWWND',JWWND, 'JPSTR',JPSTR, 'JFCOL',JFCOL,
59:      & 'JBias',JBias, 'JRESP',JRESP, 'JMNTR',JMNTR, 'JVMNT',JVMNT,
60:      & 'JALLZ',JALLZ, 'JONEZ',JONEZ, 'JSIMP',JSIMP,
61:      & 'JPIC',JPIC, 'JREFL',JREFL, 'JLATT',JLATT, 'JHLAT',JHLAT,
62:      & 'JFPR',JFPR, 'JRTTR',JRTTR, 'JRTCL',JRTCL
63:      7100 FORMAT(1x ,9(A5,I2,1x))
64:      write(IPR,*) 'JDEBG',JDEBG
65: C-----

```

```

66: C .... GEOMETRY ARRAY POINTERS .....
67: C-----
68:      WRITE(IPR,*) ' === COMMON /PGEOM/ === '
69:      WRITE(IPR,7200)
70:      & 'LSDA ',LSDA, 'LKMAT ',LKMAT, 'LKREG ',LKREG, 'LKZMAT',LKZMAT,
71:      & 'LKZREG',LKZREG, 'LKINPZ',LKINPZ, 'LKZDA ',LKZDA, 'LKZAA ',LKZAA,
72:      & 'LKMEMO',LKMEMO, 'LVOL ',LVOL, 'LPAPER',LPAPER, 'LKSREF',LKSREF,
73:      & 'LKKREF',LKKREF, 'LMEMC',LMEMC, 'LMEMZ ',LMEMZ, 'LKCELL',LKCELL,
74:      & 'LIDCEL',LIDCEL, 'LICTYP',LICTYP, 'LIPCEL',LIPCEL, 'LCELSZ',LCELSZ,
75:      & 'LIDLAT',LIDLAT, 'LLTYP ',LLTYP, 'LNVLAT',LNVLAT, 'LSZLAT',LSZLAT,
76:      & 'LIPLAT',LIPLAT, 'LKLATT',LKLATT, 'LKS LAT',LKS LAT, 'LKZLBZ',LKZLBZ,
77:      & 'LMLBZZ',LMLBZZ, 'LNKZAA',LNKZAA, 'LDALT ',LDALT, 'LKDALT',LKDALT,
78:      & 'LKHLAT',LKHLAT
79:      7200 FORMAT(1x ,5(1x,A6,I8) )
80: CC
81:      WRITE(IPR,*) ' == COMMON /PGEOM2/ DEPS ',DEPS, ' DINF ',DINF
82: C .... CPU & VECTOR LENGTH MONITOR .....
83: C      COMMON /PMNTR/
84: C      & CPUSOR,CPUSEL,CPUFly,CPUSCH,CPUCOL,
85: C      & NEXSOR,NVSOR,VLSOR,VL2SOR,MIVSOR,MAVSOR,
86: C      & NEXSEL,NVSEL,VLSEL,VL2SEL,MIVSEL,MAVSEL,
87: C      & NEXFLY,NVFLY,VLFLY,VL2FLY,MIVFLY,MAVFLY,
88: C      & NEXSCH,NVSCH,VLSCH,VL2SCH,MIVSCH,MAVSCH,
89: C      & NEXCOL,NVCOL,VLCOL,VL2COL,MIVCOL,MAVCOL
90: C
91: C CPU??? R4 CPU TIME OF SUBROUTINE
92: C NEX??? I4 NUMBER OF EXECUTION OF SUBROUTINE
93: C NV??? I4 TOTAL NUMBER OF VECTOR EVENT
94: C VL??? R4 SUM OF VECTOR LENGTH
95: C VL2??? R4 SQUARE SUM OF VECTOR LENGTH
96: C MI??? I4 MINIMUM VECTOR LENGTH
97: C MA??? I4 MAXIMUM VECTOR LENGTH
98: C
99: C '???' IS THE SYMBOL FOR EACH SUBROUTINE.
100: C SOR = SOURCE, SEL = SELECT, FLY = FLIGHT, SCH = SEARCH, COL = COLISN
101: C
102: C-----
103: C .... SOURCE DATA & ARRAY POINTERS
104: C-----
105:      WRITE(IPR,*) ' ===== COMMON /PSOUR/ ===== '
106:      WRITE(IPR,7200)
107:      $ 'LKSOUR',LKSOUR, 'LISZON',LISZON, 'LSOUR ',LSOUR, 'LPSPAC',LPSPAC,
108:      $ 'LPENRG',LPENRG, 'LKENRG',LKENRG, 'LNSTIM',LNSTIM, 'LSTIM ',LSTIM,
109:      $ 'LPSTIM',LPSTIM, 'LISTIM',LISTIM, 'LNSANG',LNSANG, 'LSANG ',LSANG,
110:      $ 'LSAXIS',LSAXIS, 'LPSANG',LPSANG, 'LISANG',LISANG, 'LIFISM',LIFISM,
111:      $ 'LMEZMN',LMEZMN
112: C-----
113: C .... TALLY ARRAY POINTERS & PARAMETERS .....
114: C-----
115:      WRITE(IPR,*) ' ===== COMMON /PTALY/ === '
116: cccc WRITE(IPR,7000)
117: cccc WRITE(IPR,*)
118: cccc $ 'WSUM ',WSUM
119:      WRITE(IPR,*)
120:      & 'LWSUM ',LWSUM, 'LWLEK ',LWLEK
121:      WRITE(IPR,7200) 'NLOST ',NLOST, 'LRESP ',LRESP, 'LFLTR ',LFLTR,
122:      $ 'LFLCL ',LFLCL, 'LRETR ',LRETR, 'LRECL ',LRECL, 'LXSOC ',LXSOC,
123:      $ 'LXSXV ',LXSXV,
124:      $ 'LXKEF ',LXKEF, 'LNCLSN',LNCLSN, 'LWCLSN',LWCLSN, 'LNLEAK',LNLEAK,
125:      $ 'LWLEAK',LWLEAK, 'LNABSB',LNABSB, 'LWABSB',LWABSB, 'LNECUT',LNECUT,
126:      $ 'LWECUT',LWECUT, 'LNTCUT',LNTCUT, 'LWTCUT',LWTCUT, 'LNKILD',LNKILD,
127:      $ 'LWKILD',LWKILD, 'LNSURV',LNSURV, 'LWSURV',LWSURV, 'LNSPLT',LNSPLT,
128:      $ 'LWSPLT',LWSPLT, 'LENGYB',LENGYB, 'LTIMEB',LTIMEB, 'LRVOL ',LRVOL,
129:      $ 'LNCNTR',LNCNTR, 'LWCNTR',LWCNTR
130: C-----

```

src/mvp/check.f

```
131: C ..... TALLY ARRAY POINTERS (CONTINUOUS ENERGY) .....
132: C-----
133:       WRITE(IPR,*) ' ==== COMMON /PTALY2/ ==== '
134:       WRITE(IPR,7200)
135:       & 'LRMIC ',LRMIC, 'LRMICR',LRMICR,'LXMIC ',LXMIC,'LRMAC ',LRMAC,
136:       & 'LRMACR',LRMACR,'LXMAC ',LXMAC, 'LRSTR ',LRSTR,'LRSCl ',LRSCl
137: C-----
138: C .... VARIANCE REDUCTION ARRAY POINTERS
139: C-----
140:       WRITE(IPR,*) ' === COMMON /PVRED/ === '
141:       WRITE(IPR,7200)
142:       & 'LXIMP ',LXIMP,'LWKIL ',LWKIL,'LWSRV ',LWSRV
143: C-----
144: C..... POINTERS TO MATERIAL & NUCILDE DATA (CROSS SECTION).
145: C..... THRESHOLD PARAMETERS .....
146: C-----
147:       WRITE(IPR,*) ' ===== COMMON /CXSEC/ ===== '
148: cccc WRITE(IPR,7000)
149:       WRITE(IPR,*)
150:       & 'ETOP ',ETOP, 'EBOT ',EBOT, 'ETHMAX',ETHMAX,'AMLIM ',AMLIM,
151:       & 'EWCUT ',EWCUT, 'ETOPL ',ETOPL,'EBOTL ',EBOTL, 'ESABL ',ESABL
152:       WRITE(IPR,7200)
153:       & 'LIDMAT',LIDMAT,'LMNUC ',LMNUC,'LINUCT',LINUCT,'LDENST',LDENST,
154:       & 'LLPDEN',LLPDEN,'NUC ',NUC,'LNUCID',LNUCID,'LMATT ',LMATT,
155:       & 'LKLIB1',LKLIB1,'LXLIB1',LXLIB1,'LKLIB2',LKLIB2,'LKLB2S',LKLB2S,
156:       & 'LXLIB2',LXLIB2,'LCX ',LCX
157: C=====
158: C-----
159: C .... CROSS SECTION ARRAY POINTERS
160: C-----
161:       WRITE(IPR,*) ' === COMMON /PXSEC/ ===== '
162:       WRITE(IPR,7200)
163:       & 'LWGTf ',LWGTf,'LWGTfI',LWGTfI
164: C-----
165: C .... PARTICLE STACK POINTERS & LENGTHES
166: C-----
167:       WRITE(IPR,*) ' ===== COMMON /STACK/ ===== '
168:       WRITE(IPR,7200)
169:       & 'LLSFFL',LLSFFL,'LNFFL ',LNFFL,'LIZFFL',LIZFFL,'LLSCOL',LLSCOL,
170:       & 'NCOLS ',NCOLS,'LLSSRC',LLSSRC,'LNNXT ',LNNXT,'LIZNXT',LIZNXT,
171:       & 'IDEFER',IDEFER,'LLSDED',LLSDED,'NDEAD ',NDEAD,'NFI5B ',NFI5B,
172:       & 'LLSREF',LLSREF,'LIZREF',LIZREF,'LISREF',LISREF,'LNBREF',LNBREF,
173:       & 'LLSLAT',LLSLAT,'LIZLAT',LIZLAT,'LNXLt ',LNXLt
174: C-----
175: C .... PARTICLE BANK POINTERS
176: C-----
177:       WRITE(IPR,*) ' ===== COMMON /XBANK/ ===== '
178:       WRITE(IPR,7200)
179:       & 'LXXX ',LXXX,'LYYY ',LYYY,'LZZZ ',LZZZ,'LAAA ',LAAA,
180:       & 'LBBB ',LBBB,'LCCC ',LCCC,'LWWW ',LWWW,'LEEE ',LEEE,
181:       & 'LIZZ ',LIZZ,
182:       & 'LIGG ',LIGG,'LTTT ',LTTT,'LXXXf',LXXXf,'LYYYf',LYYYf,
183:       & 'LZZZF',LZZZF,'LIZZF ',LIZZF,'LLEVL',LLEVL,'LLZZ ',LLZZ,
184:       & 'LLPOS',LLPOS,'LXIM ',LXIM,'LLEVLF',LLEVLF,'LLZZf',LLZZf,
185:       & 'LLPOSf',LLPOSf,'LLCRS ',LLCRS,'LLCRSF',LLCRSF
186: C-----
187: C .... BANK POINTERS FOR CROSS SECTION AND OTHER DATA RELATED TO
188: C CONTINUOUS ENERGY MONTE CARLO CALCULATION.
189: C-----
190:       WRITE(IPR,*) ' ===== COMMON /SBANK/ ===== '
191:       WRITE(IPR,7200)
192:       & 'MB ',MB,'NSMAC ',NSMAC,'NSMIC ',NSMIC,
193:       & 'LSMAC ',LSMAC,'LKMAC ',LKMAC,'LMMAC ',LMMAC,'LSMIC ',LSMIC,
194:       & 'LKSPI ',LKSPI,'LSGTAL',LSGTAL
195: C-----
```

```
196: C ..... VECTOR WORK AREA POINTERS .....
197: C-----
198:       WRITE(IPR,*) ' ===== COMMON /XWORK/ ===== '
199:       WRITE(IPR,7200)
200:       & 'NWORK ',NWORK,'LWORK ',LWORK,'NWORKB',NWORKB,'LWORKB',LWORKB
201: C
202:       RETURN
203:       END
```

src/mvp/chkenv.f

```
1:      subroutine CHKENV( PROG )
2:
3: C/#IF .NOT.NOGETENV
4:
5: C=<MVP>=====
6: C Purpose: Check enviromnet variables
7: C=====
8: C argument :
9: C
10: C i prog : code name ( "MVP" or "GMVP" )
11: C
12: C=====
13: C/#IF CUTIL.AND.MS_VISUAL
14: C
15: C ... This interface block is for CUTIL & MS-Visual tools. ...
16: C
17: * interface
18: *      subroutine FUNBUF()
19: *CDEC$      ATTRIBUTES C :: FUNBUF
20: *      end subroutine
21: *      end interface
22: C/#ENDIF
23: C
24:      include '../shared/INC/_IOUNIT'
25: C
26: C ... ATPOOL may be set.
27: C
28:      include 'INC/_ATPOOL'
29: C
30: C/#IF PARA(VPP)
31:      include '../shared/INC/_VPPPARA'
32:      include '../shared/INC/_TASKDT'
33: C/#ENDIF
34: C
35: C ... external data ...
36: C
37:      character*(*) PROG
38: C
39: C ... local data ...
40: C
41:      character*64 VAR
42:      character*128 STRING
43: C
44: C-----
45: C
46: C ==== check MVPUNBUF : set standard output unbufferd
47: C ==== check MVP_UNBUF : set standard output unbufferd
48: C ("line buffered" saying exactly )
49: C Other than null or blank should be defined for the variable
50: C to activate it.
51: C
52:      STRING = ' '
53: C
54:      call ENVGET( 'MVPUNBUF', STRING )
55:      if( STRING.eq.' ' ) then
56:          call ENVGET( 'MVP_UNBUF', STRING )
57:      end if
58: C/#IF CUTIL
59:      if( STRING.ne.' ' ) then
60:          call FUNBUF()
61:          write(IPR,*) '==(CHKENV) Standard output is line buffered.'
62:      end if
63: C/#ENDIF
64: C
65: C/#IF PARA(VPP)
```

```
66: C
67: C ==== check MVPVPPRT : set standard output unbufferd
68: C ==== check MVP_VPP_PRT : set standard output unbufferd
69: C
70: C      VPPRT//<I/O unit #> is a file name of printout
71: C      from each task.
72: C
73:      VPPRT = ' '
74:      STRING = ' '
75:      call ENVGET( 'MVPVPPRT', STRING )
76:      if( STRING.eq.' ' ) then
77:          call ENVGET( 'MVP_VPP_PRT', STRING )
78:      end if
79:      if( STRING.ne.' ' ) then
80:          VPPRT = STRING
81:          write(IPR,*) '==(CHKENV) VPP printout file name base <',
82:          & VPPRT(:iclen2(VPPRT)), '>'
83:      end if
84: C/#ENDIF
85: C
86: C ----- only for MVP -----
87: C
88: C ==== check MVP_ARTPOOL :
89: C      Cross section data processed by ART module is stored in
90: C      this directory.
91: C
92:      STRING = ' '
93: C
94:      call ENVGET( 'MVP_ARTPOOL', STRING )
95: C
96:      if( STRING.ne.' ' ) then
97:          ATPOOL = STRING
98:          write(IPR,*) '==(CHKENV) MVP_ARTPOOL <',
99:          & ATPOOL(:ICLEN2(ATPOOL)), '>'
100:      end if
101: C
102: C/#ENDIF
103: C
104:      return
105: end
```

src/mvp/chkstk.f

```
1:      subroutine CHKSTK( COMMT, IOW,   H,      IWRK )
2: C
3: C   DEBUGGING UTILITY :
4: C
5: C=<MVP>=====
6: C   PURPOSE: CHECK VALIDITY OF PARTICLE STACKS
7: C       * APPEARANCE OF IDENTICAL BANK POINTERS
8: C       * OVERLAPPING OF STACKS
9: C
10: C=====
11:      character*(*) COMMT
12:      real H(*)
13: CCC   include '../shared/INC/_ARRAY'
14:      include '../shared/INC/_LISTOFF'
15:      include '../shared/INC/_SIZES'
16:      include 'INC/_SIZES2'
17:      include 'INC/_FLAGS'
18:      include 'INC/_STACK'
19:      include '../shared/INC/_LISTON'
20: C
21:      character*4 IWRK(NBANK)
22: C
23:      write(IOW,*) '== STACK CHECKER: ', COMMT, ' == '
24:      do 100 I = 1, NBANK
25:         IWRK(I) = ' '
26:      100 continue
27: C
28: C .... CHECK APPEARANCE OF IDENTICAL BANK POINTERS
29: C       AND SUM CHECK OF ZONE-WISE PARTICLE NUMBER ARRAY
30: C
31:      N      = 0
32: C
33:      call STKIDT( 'FLIGHT', IOW, H(LLSFFL), H(LNFFL), NZONE+1, IWRK,
34: &      NBANK, N )
35: C
36:      call STKIDT( 'SEARCH', IOW, H(LLSSRC), H(LNNXT), NZONE+1, IWRK,
37: &      NBANK, N )
38: C
39:      if ( JNEUT.ne.0 )
40: &      call STKIDT( 'N-COLLISION', IOW, H(LLSCOL), NCOLS, 1, IWRK,
41: &      NBANK, N )
42: C
43:      if ( JPHOT.ne.0 )
44: &      call STKIDT( 'P-COLLISION', IOW, H(LLSCLP), NCOLP, 1, IWRK,
45: &      NBANK, N )
46: C
47:      if ( JREFL.ne.0 )
48: &      call STKIDT( 'REFL. ', IOW, H(LLSREF), H(LNBREF), NREFS+1,
49: &      IWRK, NBANK, N )
50: C
51:      if ( JLATT.ne.0 )
52: &      call STKIDT( 'LATT. ', IOW, H(LLSLAT), H(LNXLT), NLBZ+1, IWRK,
53: &      NBANK, N )
54: C
55:      call STKIDT( 'DEAD', IOW, H(LLSDED), NDEAD, 1, IWRK, NBANK, N )
56: C
57:      if ( N.ne.NBANK ) write(IOW,*) ' == BANK TOTAL MISMATCH !! ',
58: &      ' NBANK= ', NBANK, ' TOT=', N
59: C
60:      return
61:      end
```


src/mvp/chsymb.f

```
1:      character*2 function CHSYMB(NA)
2: C=<MVP>=====
3: C  PURPOSE :  RETURN CHEMICAL SYMBOL FOR ATOMIC NUMBER
4: C  CALLED IN:  MTDATP, DENTB2
5: C=====
6: C
7: C  ---- CHEMICAL SYMBOLS IN ORDER OF ATOMIC NUMBER ----
8: C
9:      parameter( MATOM      = 103 )
10:     character*2 CSYMBL(MATOM)
11: C
12:     data (CSYMBL(I),I=1,MATOM)
13:     1 'H ',
14:     2 'LI','BE',
15:     3 'NA','MG',
16:     4 'K ', 'CA', 'SC', 'TI', 'V ', 'CR', 'MN', 'FE', 'CO', 'NI', 'CU', 'ZN',
17:     &
18:     5 'RB', 'SR', 'Y ', 'ZR', 'NB', 'MO', 'TC', 'RU', 'RH', 'PD', 'AS', 'CD',
19:     &
20:     6 'CS', 'BA',
21:     L   'LA', 'CE', 'PR', 'ND', 'PM', 'SM', 'EU', 'GD', 'TB', 'DY', 'HO',
22:     L   'ER', 'TM', 'YB', 'LU',
23:     &
24:     &
25:     7 'FR', 'RA',
26:     A   'AC', 'TH', 'PA', 'U ', 'NP', 'PU', 'AM', 'CM', 'BK', 'CF', 'ES',
27:     A   'FM', 'MD', 'NO', 'LR' /
28: C
29:     if ( NA.le.MATOM ) then
30:       CHSYMB = CSYMBL(NA)
31:     else
32:       CHSYMB = ' '
33:     end if
34:     return
35: end
```

src/mvp/ckpool.f

```

1:      subroutine CKPOOL( ATPool,NCID, FILE, IXS,  IPRT, JPOOL )
2: C=====
3: C Purpose: check arbitrary temperature library pool
4: C
5: C   Check existence of processed xsec by ART in directory ATPool.
6: C   If processed one in found under directory ATPool,
7: C   and opened successfully, returns JPOOL = 1.
8: C
9: C   If the file exists but a "lock file" also exists, some other
10: C   process is writing the file , so do not use the file.
11: C
12: C Calls: ICLEN2 (get position of the last non blank letter in a string)
13: C
14: C-----
15: C arguments ( i = input, o=output , w=work )
16: C
17: C i ATPool : directory path to arbitrary temperature library pool.
18: C           Must compose full path by directly connecting with
19: C           nuclide ID.
20: C           eg. on UNIX or compatible OS,
21: C           ATPool = '/path1/path/' is valid, but
22: C           ATPool = '/path1/path' may not be valid.
23: C i NCID : nuclide ID (+temperature string).
24: C           Tries to open FILE = ATPool//NUCID.
25: C
26: C i IXS : I/O unit on which library file is opened (if the file is
27: C         already created.
28: C i IPRT : message printout I/O unit
29: C o JPOOL : returns non zero value when a pooled file is opened
30: C           succesfully.
31: C=====
32:      character*(*) ATPool
33:      character*(*) NCID
34:      character*(*) FILE
35: C
36: C .... Local data
37: C
38:      logical JJJ
39:      character*300 LKFILE
40: C
41: C-----
42: C
43:      JPOOL = 0
44: C
45: C -----
46: C .... check existence of the expected file
47: C -----
48: C
49:      FILE = ATPool(:ICLEN2(ATPool)) //NCID(:ICLEN2(NCID))
50: C
51:      inquire(file =FILE(:ICLEN2(FILE)),exist =JJJ)
52: C
53:      if ( .not.JJJ ) return
54: C
55: C -----
56: C .... check existence of a "lock file" : file//'.LOC'
57: C -----
58: C
59:      LKFILE = FILE(:ICLEN2(FILE)) //''.LOC'
60:      inquire(file =LKFILE(:ICLEN2(LKFILE)),exist =JJJ)
61: C
62:      if ( JJJ ) then
63:         write(IPRT,*) '!!!(CKPOOL) Pooled ART library file <',
64:         & FILE(:ICLEN2(FILE)), '> exists, but a lock file <',
65:         & LKFILE(:ICLEN2(LKFILE)), '> also exists.'
```

```

66:         write(IPRT,*) ' Probably some other process is locking the ',
67:         & 'file for output. Do not read the file.'
68:         call CNTERR( 'WARNING' )
69:         return
70:         end if
71: C
72: C -----
73: C ... try to open the pooled file.
74: C -----
75: C
76:      inquire(IXS,opened =JJJ)
77:      if ( JJJ ) close( IXS )
78:      open( IXS, file =FILE(:ICLEN2(FILE)), status ='OLD', form =
79:      & 'UNFORMATTED', iostat =IOS )
80: C
81:      if ( IOS.eq.0 ) then
82:         JPOOL = 1
83:         write(IPRT,*) ' == Pooled ART library file <',
84:         & FILE(:ICLEN2(FILE)), '> is opened.'
85:      else
86:         write(IPRT,*)
87:         & '!!!(CKPOOL) Failed to open a pooled ART library file <',
88:         & FILE(:ICLEN2(FILE)), '>. Code =', IOS
89:         end if
90: C
91:      return
92:      end
```

src/mvp/clrstk.f

```

1:      subroutine CLRSTK( IOW, NENDL, WENDL, WWW,
2:      &                  NZONE, NREFS, NLBZ, NBANK,
3:      &                  NDEAD,
4:      &                  NFFL, LSFFL,
5:      &                  NNXT, IDEFER, LSSRC,
6:      &                  NCOLS, LSCOL,
7:      &                  NBREF, LSREF,
8:      &                  NXLT, LSLAT,
9:      &                  NCOLP, LSCLP )
10: C=====
11: C purpose: clear all particle stacks.
12: C return number and weight of remaining particles in NENDL and WENDL.
13: C
14: C called int selone
15: C=====
16: C
17:      real*8 WENDL
18: C
19:      include 'INC/_FLAGS'
20: C
21:      integer NFFL(NZONE+1), LSFFL(NBANK)
22:      integer NNXT(NZONE+1), IDEFER, LSSRC(NBANK)
23:      integer NCOLS, LSCOL(NBANK)
24:      integer NBREF(NREFS+1), LSREF(NBANK)
25:      integer NXLT(NLBZ+1), LSLAT(NBANK)
26:      integer NCOLP, LSCLP(NBANK)
27: C
28:      real WWW(NBANK)
29: C
30: C-----
31: C
32:      NDEAD = NBANK
33: C
34:      NENDL = 0
35:      WENDL = 0.0D0
36: C
37: C
38:      NENDL = NENDL + NFFL(NZONE+1)
39:      WENDL0 = 0.0D0
40:      do 100 I = 1, NFFL(NZONE+1)
41:          WENDL0 = WENDL0 + WWW(LSFFL(I))
42: 100 continue
43:      WENDL = WENDL + WENDL0
44:      write(IOW,7000) 'Flight',NFFL(NZONE+1),WENDL0
45:      call PUTVI( NFFL, NZONE+1, 0 )
46: C
47:      NENDL = NENDL + NNXT(NZONE+1)
48:      WENDL0 = 0.0D0
49:      do 110 I = 1, NNXT(NZONE+1)
50:          WENDL0 = WENDL0 + WWW(LSSRC(I))
51: 110 continue
52:      WENDL = WENDL + WENDL0
53:      write(IOW,7000) 'Search',NNXT(NZONE+1),WENDL0
54:      call PUTVI( NNXT, NZONE+1, 0 )
55:      IDEFER = 0
56: C
57:      if ( JREFL.ne.0 ) then
58:          NENDL = NENDL + NBREF(NREFS+1)
59:          WENDL0 = 0.0D0
60:          do 120 I = 1, NBREF(NREFS+1)
61:              WENDL0 = WENDL0 + WWW(LSREF(I))
62: 120 continue
63:          WENDL = WENDL + WENDL0
64:          write(IOW,7000) 'Reflection',NBREF(NREFS+1),WENDL0
65:          call PUTVI( NBREF, NREFS+1, 0 )

```

```

66:      end if
67: C
68:      if ( JLATT.ne.0 ) then
69:          NENDL = NENDL + NXLT(NLBZ+1)
70:          WENDL0 = 0.0D0
71:          do 130 I = 1, NXLT(NLBZ+1)
72:              WENDL0 = WENDL0 + WWW(LSLAT(I))
73: 130 continue
74:          WENDL = WENDL + WENDL0
75:          write(IOW,7000) 'Lattice',NXLT(NLBZ+1),WENDL0
76:          call PUTVI( NXLT, NLBZ+1, 0 )
77:      end if
78: C
79:      if ( JNEUT.ne.0 ) then
80:          NENDL = NENDL + NCOLS
81:          WENDL0 = 0.0D0
82:          do 140 I = 1, NCOLS
83:              WENDL0 = WENDL0 + WWW(LSCOL(I))
84: 140 continue
85:          WENDL = WENDL + WENDL0
86:          write(IOW,7000) 'Collision(N)',NCOLS,WENDL0
87:          NCOLS = 0
88:      end if
89: C
90:      if ( JPHOT.ne.0 ) then
91:          NENDL = NENDL + NCOLP
92:          WENDL0 = 0.0D0
93:          do 150 I = 1, NCOLP
94:              WENDL0 = WENDL0 + WWW(LSCLP(I))
95: 150 continue
96:          WENDL = WENDL + WENDL0
97:          write(IOW,7000) 'Collision(P)',NCOLP,WENDL0
98:          NCOLP = 0
99:      end if
100: C
101:      7000 format(1x,'** ',a,' stack has ',i5,' particles. ',
102:      &          'Weight sum = ',E12.5)
103: C
104:      return
105: end

```

src/mvp/ctxsin.f

```

1:      subroutine CTXSIN( A,      IA,      CHA,      LIMIT,
2:      &                  LIMITC,JXSKIP )
3:      C=<MVP>=====
4:      C PURPOSE : INPUT & PROCESSING OF CONTINUOUS-ENERGY CROSS SETION
5:      C          AND RELATED DATA
6:      C CALLED IN:  INTRO
7:      c##<2007/03/14:PN3:
8:      c##C CALLS   : FREADN,VALUE0,MTDATA,DENTAB,XSFILE,XSEC,XSECP,XSECE,MEMERR
9:      C CALLS      : FREADN,VALUE0,MTDATA,DENTAB,XSFILE,XSEC,XSECP,XSECE,XSECPN,
10:     C              MEMERR
11:     c##>
12:     C-----
13:     implicit real(A-H,O-Z)
14:     real A(LIMIT)
15:     integer IA(LIMIT)
16:     character*4 CHA(LIMITC)
17:     C
18:     include '../shared/INC/_WORDL'
19:     include '../shared/INC/_IOUNIT'
20:     include 'INC/_IOUNIT2'
21:
22:     C/#IF PARA(PVM)
23:     *   include '../shared/INC/_PVMPARA'
24:     C/#ELSEIF PARA(MPI)
25:     *   include 'mpif.h'
26:     C/#ENDIF
27:     include '../shared/INC/_TASKDT'
28:     C
29:     include 'INC/_KPIDS'
30:     include 'INC/_NGPS'
31:     C
32:     include 'INC/_CXSEC'
33:     include 'INC/_ATPOOL'
34:     include '../shared/INC/_SIZES'
35:     include 'INC/_SIZES2'
36:     include 'INC/_FLAGS'
37:     c##<2007/03/14:PN3:
38:     include 'INC/_ISOTOP'
39:     c##>
40:     C
41:     C ... local data ...
42:     C
43:     c##<2007/03/14:PN3:
44:     c## character*16 ITEM
45:     character ITEM*33, ITEM2*13, HERRMES*60
46:     c##>
47:     integer OUFIL
48:     real RWRK(2)
49:     real*8 DWRK(2)
50:     real*8 TPRECS, TPRMIN, TEMPMT, DTEMP, TMPMAX, TMPMIN
51:
52:     integer NDIGIT
53:     character*16 FMT
54:     C-----
55:     C
56:     call HEADER( IPR, 'MATERIAL & CROSS SECTION DATA' )
57:     C
58:     C-----
59:     C INITIALIZATION
60:     C-----
61:     C
62:     NMAT      = 0
63:     NUC        = 0
64:     INFILE     = 0
65:     OUFIL      = 0

```

```

66:     MLEN      = 0
67:     c##<2007/03/14:PN3:
68:     NPATOM    = 0
69:     NUCPN     = 0
70:     c##>
71:     C
72:     C ... default material temperature is negative (used fixed temperature
73:     C      library data).
74:     C
75:     TEMPMT    = -1.0D0
76:     TMPMAX    = TEMPMT
77:     TMPMIN    = TEMPMT
78:     C
79:     C ... default precision of temperature is 1.0 degree
80:     C
81:     C Actual temperature is rounded value as;
82:     C
83:     C      nint( temperature / tprec )*tprec
84:     C
85:     TPRECS    = 1.0D0
86:     TPRMIN    = TPRECS
87:     C
88:     C ... default mode of doppler broadning procedure (vector/non-vector)
89:     C
90:     JVPOP     = 0
91:     C/#IF VECTORIZE
92:     *         JVPOP = 1
93:     C/#ENDIF
94:     C
95:     C ... check pooled library and read/store in $MVP_ARTPOOL if possible.
96:     C
97:     JXPOOL    = 1
98:     c##<2007/03/14:PN3:
99:     C
100:    C ... elimination of charged particle data in photonuclear library
101:    C      (default is no elimination)
102:    C
103:    JPNLCP    = 0
104:    c##>
105:    C
106:    call LABEL( IPR, 'MATERIAL COMPOSITION DATA' )
107:    C
108:    KMNUC     = 0
109:    C
110:    C-----
111:    C create data packet container for temporary storage of material
112:    C composition data etc.
113:    C-----
114:    call REMAIN( 'MPK', NLTEMP, 'I4D', LAST )
115:    call KEEP( 'MPK', LMPK, NLTEMP, 'I4D', LAST, JDEBG )
116:    call PACK00( A(LMPK), 'MATERIAL COMPOSITION', NLTEMP, IRET )
117:    if ( IRET.ne.0 ) then
118:      call CNTERR( 'FATAL' )
119:      return
120:    end if
121:    C-----
122:    C READ ITEM (VARIABLE / ITEM NAME)
123:    C-----
124:    C
125:    100 call FREADS( ITEM, NLEN, IEND )
126:    if ( IEND.ne.0 ) then
127:      c##<2007/03/14:PN3:
128:      c## write(IMG,*)
129:      c## & 'XXX Unexpected end of input data in "$CROSS SECTION" block.'
130:      write(IMG,'(1X,A,A)')

```

src/mvp/ctxsin.f

```

131:      &          'XXX(ctxsin) Unexpected end of input data in "$CROSS',
132:      &          ' SECTION' block.'
133: c##>
134:      call PRSTOP( 1, 'Invalid "$CROSS SECTION' block.' )
135:      stop 888
136:      end if
137: c
138: c ... IGNORE "SETCION" of "$CROSS SECTION" ...
139: c
140:      if ( ITEM(1:NLEN).eq.'SECTION' ) then
141:      go to 100
142: c
143: C-----
144: C INFILE: input of processed cross section if any (unimplemented)
145: C-----
146:      else if ( ITEM(1:NLEN).eq.'INFILE' ) then
147:      call VALUE0( ' ', ICALL, 'I4', INFILE )
148:      if ( INFILE.ne.0 ) then
149:      write(IPR,7000) INFILE
150: 7000      format(/1X,'==== ALREADY PROCESSED CROSS SECTION DATA ',
151:      &          'ARE INPUT FROM UNIT INFILE (=' ,I2,')')
152:      end if
153: C-----
154: C OUTFILE: output of cross section data after processing(unimplemented)
155: C-----
156:      else if ( ITEM(1:NLEN).eq.'OUTFILE' ) then
157:      call VALUE0( ' ', ICALL, 'I4', OUFIL )
158:      if ( OUFIL.ne.0 ) then
159:      write(IPR,7020) OUFIL
160: 7020      format(/1X,'==== PROCESSED CROSS SECTION DATA ',
161:      &          'ARE OUTPUT TO UNIT OUTFILE (=' ,I2,')')
162:      end if
163: C-----
164: C TPRECS : temperature precision
165: C-----
166:      else if ( ITEM(1:NLEN).eq.'TPRECS' ) then
167:      call R8READ( ITEM(:NLEN), TPRECS, NA, -1, IEND )
168:      if ( IEND.ne.0 ) then
169: c##<2007/03/14:PN3:
170: c##      write(IMG,*) 'XXX(CTXSIN) Input error for <', ITEM(:NLEN),
171: c##      &          '>'
172:      write(IMG,'(1X,A,A,A)')
173:      &          'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '>'
174: c##>
175:      call CNTERR( 'FATAL' )
176:      end if
177:      if ( TPRECS.le.0.0 ) then
178: c##<2007/03/14:PN3:
179: c##      write(IMG,*) 'XXX(CTXSIN) Temperature precision <TPRECS>',
180: c##      &          ' must be positive. ', TPRECS
181:      write(IMG,'(1X,A,A,1P,E13.5)')
182:      &          'XXX(CTXSIN) Temperature precision <TPRECS>',
183:      &          ' must be positive. ', TPRECS
184: c##>
185:      call CNTERR( 'FATAL' )
186:      TPRECS = 1.0
187:      else
188:      TPRMIN = MIN(TPRMIN,TPRECS)
189:      write(IPR,7040) TPRECS
190: c##<2007/03/14:PN3:
191: c7040      format(/1X,'>>> Temperature precision changed to ',E13.5/)
192: 7040      format(/' >>> Temperature precision changed to ',1P,E13.5/)
193: c##>
194:      end if
195: C-----

```

```

196: C JVPOP : broadning procedure by vector mode if non zero
197: C-----
198:      else if ( ITEM(1:NLEN).eq.'JVPOP' ) then
199:      call I4READ( ITEM(:NLEN), JVPOP, NA, -1, IEND )
200:      if ( IEND.ne.0 ) then
201: c##<2007/03/14:PN3:
202: c##      write(IMG,*) 'XXX(CTXSIN) Input error for <', ITEM(:NLEN),
203: c##      &          '>'
204:      write(IMG,'(1X,A,A,A)')
205:      &          'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '>'
206: c##>
207:      call CNTERR( 'FATAL' )
208:      end if
209: C-----
210: C data name "JXPOOL" :
211: C JXPOOL : pool processed library by ART in directory $MVP_ARTPOOL
212: C if non-zero, else always process by ART and do not store
213: C in pool.
214: C-----
215:      else if ( ITEM(1:NLEN).eq.'JXPOOL' .or. ITEM(1:NLEN).eq.'ARTPOOL'
216:      &          ) then
217:      call I4READ( ITEM(:NLEN), JXPOOL, NA, -1, IEND )
218:      if ( IEND.ne.0 ) then
219: c##<2007/03/14:PN3:
220: c##      write(IMG,*) 'XXX(CTXSIN) Input error for <', ITEM(:NLEN),
221: c##      &          '>'
222:      write(IMG,'(1X,A,A,A)')
223:      &          'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '>'
224: c##>
225:      call CNTERR( 'FATAL' )
226:      end if
227: c##<2007/03/14:PN3:
228: C-----
229: C JPNLCP : option of elimination for charged particle data in photo-
230: C nuclear library : 0 / 1 = no eliminate / eliminate
231: C-----
232:      else if ( ITEM(1:NLEN).eq.'JPNLCP' ) then
233:      call I4READ( ITEM(:NLEN), JPNLCP, NA, -1, IEND )
234:      if ( IEND.ne.0 ) then
235:      write(IMG,'(1X,A,A,A)')
236:      &          'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '>'
237:      call CNTERR( 'FATAL' )
238:      end if
239: c##>
240: C-----
241: C '&' : STARTING POINT MARKER OF DATA FOR A MATERIAL.
242: C
243: C =====
244: C DATA PACKETS for each material in a(lmpk)
245: C =====
246: C
247: C label:      '&'+material# (I3)
248: C label:      'IDMAT'
249: C integer:    material ID
250: C
251: C character:  nuclide ID
252: C real:       nuclide number density
253: C real:       temperature
254: C
255: C ... repeat ID & density ...
256: C
257: C label:      'MNUC'
258: C integer:    number of nuclides
259: C
260: C-----

```

src/mvp/ctxsin.f

```

261:      else if ( ITEM(1:NLEN).eq.'&' ) then
262:          NMAT      = NMAT + 1
263:      C
264:      C .... STORE THE NUMBER OF NUCLIDES FOR THE LAST MATERIAL HERE ! ....
265:      C
266:          if ( NMAT.gt.1 ) then
267:              write(IPR,7100) KMNUC
268:              call PACKLB( A(LMPK), 'NUCLIDE NUMBER', 'MNUC', IRET1 )
269:              call PACKND( A(LMPK), 'NUCLIDE NUMBER', 'I4', KMNUC, 1,
270:                  &          IRET2 )
271:              if ( IRET1.ne.0 .or. IRET2.ne.0 ) then
272:                  call CNTERR( 'FATAL' )
273:                  return
274:              end if
275:      C
276:          MLEN      = MLEN + KMNUC
277:          KMNUC      = 0
278:      end if
279:      C
280:      C .... put label in data packet ...
281:      C
282:      C      if ( NMAT.ge.10000 ) then
283:      C          write(IMG, '( /lx,a,a/ )' )
284:      C      &          'XXX(CTXSIN) Number of materials must be less than ',
285:      C      &          '10000.'
286:      C          call CNTERR( 'FATAL' )
287:      C          return
288:      C      end if
289:      C
290:      NDIGIT = int(log10(real(NMAT)))+1
291:      write(FMT, '(a2,i1,a1)') '(I', NDIGIT, ')'
292:      write(ITEM(2:), FMT) NMAT
293:      call PACKLB(A(LMPK), 'MATERIAL HEADER', ITEM(1:NDIGIT+1), IREG)
294:      C
295:      if ( IRET.ne.0 ) then
296:          call CNTERR( 'FATAL' )
297:          return
298:      end if
299:      C
300:      C ... reset material temperature ...
301:      C
302:      TEMPMT = -1.D0
303:      C-----
304:      C      IDMAT : MATERIAL ID NUMBER ( FROM 1 TO 998 )
305:      C      (THIS MUST BE THE FIRST INPUT ITEM FOR EACH MATERIAL ! )
306:      C-----
307:      else if ( ITEM(1:NLEN).eq.'IDMAT' ) then
308:          call VALUE0( ' ', ICALL, 'I4', IDM )
309:      c##<2007/03/14:PN3:
310:          if ( NMAT.le.1 ) write(IPR, '(1h)')
311:      c##>
312:          write(IPR,7060) NMAT, IDM
313:      7060      format(6X, '*****')
314:      &          /6X, ' MATERIAL NO. ', I8, ' : ID = ', I8
315:      &          /6X, '*****')
316:      C
317:          if ( IDM.lt.0 ) then
318:              write(IMG, '( /lx,a,i7/ )' )
319:              &          'XXX(CTXSIN) Negative material ID is not allowed:',
320:              &          IDM
321:              call CNTERR( 'FATAL' )
322:          else if ( IDM.eq.0 ) then
323:      c##<2007/03/14:PN3:
324:      c##          write(IMG, '( /lx,a/ )' )
325:      c## &          '!!! Mat. ID is zero. This material is treated as inner void.'

```

```

326:          write(IMG, '( /lx,a,a/ )' )
327:          &          '!!! (ctxsin) Material ID is zero. This material',
328:          &          ' is treated as inner void.'
329:      c##>
330:          call CNTERR( 'WARNING' )
331:      end if
332:      C
333:      c##<2007/03/14:PN3:
334:      c##          write(IPR,7080)
335:      c7080      format(5X, ' ',
336:      c## &          ' NUCLIDE ', ' ', ' ', ' DENSITY ', ' ', ' TEMPERATURE' /
337:      c## &          5X, ' ', ' ', '-----', 2( ' ', ' -----' ) )
338:          if ( JPHNU.eq.0 ) then
339:              write(IPR,7080)
340:          else
341:              write(IPR,7081)
342:          end if
343:      7080      format(5X, ' ', ' ', ' NUCLIDE ', ' ', ' ', ' DENSITY ', ' ', ' TEMPERATURE' /
344:      &          5X, ' ', ' ', '-----', 2( ' ', ' -----' ) )
345:      7081      format(5X, ' ', ' ', ' NUCLIDE ', ' ', ' ', ' DENSITY ', ' ', ' TEMPERATURE',
346:      &          ' PHOTO-NUC'
347:      &          /5X, ' ', ' ', '-----', 2( ' ', ' -----' ), ' -----' )
348:      c##>
349:      C
350:          call PACKLB( A(LMPK), 'MATERIAL ID', 'IDMAT', IRET1 )
351:          call PACKND( A(LMPK), 'MATERIAL ID', 'I4', IDM, 1, IRET2 )
352:          if ( IRET1.ne.0 .or. IRET2.ne.0 ) then
353:              call CNTERR( 'FATAL' )
354:              return
355:          end if
356:      C-----
357:      C      TEMPMT : temperature of the material.
358:      C      When this data is specified as positive, all nuclide libraries
359:      C      following after this data in this material are created from
360:      C      temperature free library by ART module.
361:      C-----
362:          else if ( ITEM(1:NLEN).eq.'TEMPMT' ) then
363:              call R8READ( ITEM(:NLEN), TEMPMT, NA, -1, IEND )
364:              if ( IEND.ne.0 ) then
365:      c##<2007/03/14:PN3:
366:      c##          write(IMG, *) 'XXX(CTXSIN) Input error for <', ITEM(:NLEN),
367:      c## &          '>'
368:              write(IMG, '(1X,A,A,A)')
369:              &          'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '>'
370:      c##>
371:              call CNTERR( 'FATAL' )
372:          end if
373:          if ( TEMPMT.lt.0.0 ) then
374:      c##<2007/03/14:PN3:
375:      c##          write(IMG, *)
376:      c##          write(IMG, '(1X,A,1P,E13.5)')
377:      c##>
378:      &          '!!! (CTXSIN) Negative <TEMPMT> data is ignored. ',
379:      &          TEMPMT
380:          call CNTERR( 'WARNING' )
381:          TEMPMT = -1.0D0
382:          end if
383:      C-----
384:      C      '$END' : END OF MATERIAL DATA INPUT
385:      C-----
386:          else if ( ITEM(1:NLEN).eq.'END' .or. ITEM(1:NLEN).eq.'$END' ) then
387:      c##<2007/03/14:PN3:
388:      c##          write(IPR,7100) KMNUC
389:      c7100      format(5X, ' ', ' ', '-----', 2( ' ', ' -----' ) /5X, ' ',
390:      c## &          ' NUMBER OF NUCLIDES = ', I3 / )

```

src/mvp/ctxsin.f

```

391:      if ( JPHNU.eq.0 ) then
392:        write(IPR,7100) KMNUC
393:      else
394:        write(IPR,7101) KMNUC
395:      end if
396: 7100 format(5X,' ', '-----',2(' ', '-----')
397:      &      /5X,'      NUMBER OF NUCLIDES = ',I3/)
398: 7101 format(5X,' ', '-----',2(' ', '-----'), ' -----'
399:      &      /5X,'      NUMBER OF NUCLIDES = ',I3/)
400: c##>
401: C
402:      call PACKLB( A(LMPK), 'nuclide number', 'MNUC', IRET )
403:      call PACKND( A(LMPK), 'nuclide number', 'I4', KMNUC, 1, IRET )
404:      if ( IRET.ne.0 ) then
405:        call CNTERR( 'FATAL' )
406:        write(IMG,*) 'XXX(CTXSIN) DATA PACKET ERROR ', IRET1, IRET2
407:        return
408:      end if
409: C
410:      MLEN = MLEN + KMNUC
411: C
412:      go to 110
413: C-----
414: C  NUMBER DENSITIES (and temperature if any) OF EACH NUCLIDE
415: C-----
416:      else
417:        NB = -2
418:        ccccccc call R8READ( ITEM(:NLEN), DWRK, NA, NB, IEND )
419:        call R8READ( ' ', DWRK, NA, NB, IEND )
420:        if ( IEND.ne.0 ) then
421: c##<2007/03/14:PN3:
422: c##      write(IMG,*) 'XXX(CTXSIN) Input error for <', ITEM(:NLEN),
423: c## &      '> ',
424: c## &      ' ( a NUCLIDE-ID(density temperature) is expected.)'
425:      write(IMG,'(1X,A,A,A,A)')
426:      &      'XXX(CTXSIN) Input error for <', ITEM(:NLEN), '> ',
427:      &      ' ( a NUCLIDE-ID(density temperature) is expected.)'
428: c##>
429:      call CNTERR( 'FATAL' )
430:      end if
431: C
432:      if ( NA.lt.1 .or. NA.gt.2 ) then
433: c##<2007/03/14:PN3:
434: c##      write(IMG,*) 'XXX(CTXSIN) Number of data for <',
435: c## &      ITEM(:NLEN), '> ', ' must be 1 or 2, but ', NA,
436: c## &      ' data are given.'
437: c##      write(IMG,*)
438: c## &      ' ( a NUCLIDE-ID(density temperature) is expected.)'
439:      write(IMG,'(1X,A,A,A,I6,A)')
440:      &      'XXX(CTXSIN) Number of data for <', ITEM(:NLEN),
441:      &      '> must be 1 or 2, but ', NA, ' data are given.'
442:      write(IMG,'(1X,A)')
443:      &      ' ( a NUCLIDE-ID(density temperature) is expected.)'
444: c##>
445:      call CNTERR( 'FATAL' )
446:      end if
447: C
448:      RDEN = DWRK(1)
449:      if ( JNEUT.ne.0 ) then
450:        if ( NA.eq.2 ) then
451:          DTEMP = DWRK(2)
452:        else
453:          DTEMP = TEMPMT
454:        end if
455:      else

```

```

456: c
457: c      ... nuclide temperature has meanings only in neutron problem.
458: c
459:      DTEMP = -1.0D0
460:      end if
461: C
462:      if ( DTEMP.ge.0.0 ) then
463:        TMPMAX = MAX(TMPMAX,DTEMP)
464:        TMPMIN = MIN(TMPMIN,DTEMP)
465:      end if
466: C
467:      KMNUC = KMNUC + 1
468: C
469: C      ... pack nuclide ID, density and temperature ...
470: C
471: c##<2007/03/14:PN3:
472: c##      call PACKCS( A(LMPK), 'NUCLIDE ID', ITEM(:NLEN), IRET1 )
473:      NHYP = 0
474:      ISH1 = NLEN
475:      ISH2 = 1
476:      ISH3 = NLEN
477:      do I=1,NLEN
478:        if ( ITEM(I:I).eq.'-' ) then
479:          NHYP = NHYP + 1
480:          ISH1 = I - 1
481:          ISH2 = I + 1
482:        end if
483:      end do
484: C
485:      call PACKCS( A(LMPK), 'NUCLIDE ID', ITEM(:ISH1), IRET1 )
486:      call PACKCS( A(LMPK), 'PN-NUCLIDE ID', ITEM(ISH2:ISH3), IRET5 )
487: c##>
488:      call PACKND( A(LMPK), 'DENSITY', 'R4', RDEN, 1, IRET2 )
489:      call PACKND( A(LMPK), 'TEMPERATURE', 'R8', DTEMP, 1, IRET3 )
490:      call PACKND( A(LMPK), 'TPRECS', 'R8', TPRECS, 1, IRET4 )
491: c##<2007/03/14:PN3:
492: c##      if ( IRET1.ne.0 .or. IRET2.ne.0 .or. IRET3.ne.0 .or. IRET4.ne.0
493: c## &      ) then
494: c## &      if ( IRET1.ne.0 .or. IRET2.ne.0 .or. IRET3.ne.0 .or.
495: c## &      IRET4.ne.0 .or. IRET5.ne.0 ) then
496: c##>
497:      call CNTERR( 'FATAL' )
498: c##<2007/03/14:PN3:
499: c##      go to 150
500:      go to 910
501: c##>
502:      end if
503: C
504: c##<2007/03/14:PN3:
505: c##      if ( DTEMP.ge.0 ) then
506: c##        write(IPR,7120) ITEM(:NLEN), RDEN, DTEMP
507: c7120 format(5X,' ',A,T21,1P,E13.5,1X,E13.5)
508: c##      else if ( JNEUT.ne.0 ) then
509: c##        write(IPR,7140) ITEM(:NLEN), RDEN, ' FROM LIBRARY'
510: c7140 format(5X,' ',A,T21,1P,E13.5,1X,A)
511: c##      else
512: c##        write(IPR,7140) ITEM(:NLEN), RDEN, ' NO MEANING'
513: c##      end if
514:      if ( DTEMP.ge.0 ) then
515:        write(ITEM2,'(1P,E13.5)') DTEMP
516:      else if ( JNEUT.ne.0 ) then
517:        ITEM2 = ' FROM LIBRARY'
518:      else
519:        ITEM2 = ' NO MEANING '
520:      end if

```

src/mvp/ctxsin.f

```

521:      if ( JPHNU.eq.0 ) then
522:        write(IPR,7120) ITEM(:ISH1), RDEN, ITEM2
523:      else
524:        write(IPR,7120) ITEM(:ISH1), RDEN, ITEM2, ITEM(ISH2:ISH3)
525:      endif
526: 7120 format(5X,' ',A,T18,1P,E13.5,1X,A:2X,A)
527: c##>
528: c
529:      if ( RDEN.lt.0.0 ) then
530: c##<2007/03/14:PN3:
531: c##      write(IMG,'(/lx,a,e13.5/)')
532: c## &      'XXX Negative density is input: ', RDEN
533: c##      write(IMG,'(/lx,a,e13.5/)')
534: c## &      'XXX(ctxsin) Negative density is input: ', RDEN
535: c##>
536:      call CNTERR( 'FATAL' )
537:    end if
538: C
539: C-----
540: C REPEAT INPUT
541: C-----
542:    end if
543:    go to 100
544: C-----
545: C END OF MATERIAL DATA INPUT
546: C-----
547: 110 continue
548: c
549: c ... reset input data I/O record
550: c
551:    call RESET
552: C
553: C .... close packet container & release memory ...
554: C
555:    call PCTCLS( A(LMPK), 'MATERIAL DATA', NLTEMP, NLP, IRET )
556: C
557:    call RESIZE( 'MPK', LMPK, NLP, 'I4D', LAST )
558: C-----
559: C INPUT PROCESSED CROSS SECTION DATA FROM UNIT INFILE (INFILE >0)
560: C (unimplemented)
561: C-----
562:    if ( INFILE.gt.0 ) then
563:      call XSFILE
564:      return
565:    end if
566: C
567: C-----
568: C MAKE DATA REGIONS FOR IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1) etc.
569: C-----
570:    call KEPV(A(1), 'IDMAT', LIDMAT, NMAT, 'I4', LAST, 0, JDEBG )
571:    call KEPV(A(1), 'MNUC', LMNUC, NMAT, 'I4', LAST, 0, JDEBG )
572:    call KEPV(A(1), 'LPDEN', LLPDEN, NMAT+1, 'I4', LAST, 0, JDEBG )
573:    call KEPV(A(1), 'INUCT', LINUCT, MLEN, 'I4', LAST, 0, JDEBG )
574:    call KEPV(A(1), 'DENST', LDENST, MLEN, 'R4', LAST, 0, JDEBG )
575:    call KEPV(A(1), 'DSMAT', LDSMAT, NMAT, 'R4', LAST, 0, JDEBG )
576: C-----
577: C PUT DATA TO IDMAT, MNUC, LPDEN & COUNT TOTAL NUCLIDE NUMBER
578: C-----
579: C
580: C ... give all remaining memory to 'NUCID' temporaly.
581: C
582:    call REMAINC( 'NUCID', MAXNC, 'C16', LASTC )
583:    if ( MAXNC.le.0 ) then
584: c##<2007/03/14:PN3:
585: c##      write(IMG,*) 'XXX(CTXSIN) Character memory is insufficient ',

```

```

586:      write(IMG,'(1X,A,A)')
587: &      'XXX(CTXSIN) Character memory is insufficient',
588: c##>
589: &      ' to store nuclide ID table in subroutine.'
590:    call CNTERR( 'FATAL' )
591:    call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
592:    stop 999
593:  end if
594:
595:    call KEEPC( 'NUCID', LNUCID, MAXNC, 'C16', LASTC, JDEBG )
596:    call RWIND( IUB )
597:
598:    call MTDATA( A(LMPK), IA(LIDMAT), IA(LMNUC), IA(LLPDEN), NMAT,
599: &      A(LINUCT), A(LDENST), TPRMIN, TMPMAX, TMPMIN, A(LDSMAT),
600: c##<2007/03/14:PN3:
601: c## &      CHA(LNUCID), NUC, MAXNC, MLEN, LAST, LIMIT, LIMITA )
602: &      CHA(LNUCID), NUC, MAXNC, MLEN, LAST, LIMIT, LIMITA,
603: &      JPHNU )
604: c##>
605: C
606:    call RESIZEC( 'NUCID', LNUCID, NUC, 'C16', LASTC )
607: C
608: C ... nuclide ID's as input (used to search index file)
609: C
610: c##<2007/03/14:PN3:
611:    call KEEPC( 'NCIDP', LNCIDP, NUC, 'C16', LASTC, JDEBG )
612: c##>
613:    call KEEPC( 'NCIDI', LNCIDI, NUC, 'C16', LASTC, JDEBG )
614:    if ( LASTC.gt.LIMITC+1 ) then
615: c##<2007/03/14:PN3:
616: c##      write(IMG,*) 'XXX(CTXSIN) Character memory is insufficient ',
617: c##      write(IMG,'(1X,A,A)')
618: c## &      'XXX(CTXSIN) Character memory is insufficient',
619: c##>
620: &      ' to store neutron library ID string <NCIDI>.'
621:    call CNTERR( 'FATAL' )
622:    call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
623:    stop 999
624:  end if
625: C
626: C ... set nuclide temperature (from temporary file IUB) ...
627: C
628:    call KEPV(A(1), 'TEMPN', LTEMPN, NUC, 'R8', LAST, 0.0D0, JDEBG )
629: C
630: C
631:    call RWIND( IUB )
632: c##<2007/03/14:PN3:
633: c##      call INTMPN( IUB, CHA(LNCIDI), A(LTEMPN), NUC, IMG )
634: c##      call INTMPN( IUB, CHA(LNCIDI), A(LTEMPN), NUC, IMG, CHA(LNCIDP) )
635: c##>
636: C
637: C-----
638: C CHECK NUMBER DENSITY TABLE FOR PHOTON PROBLEM.
639: C-----
640: C
641:    if ( JPHOT.ne.0 ) then
642: C
643:      call KEPV(A(1), 'MPATM', LMPATM, NMAT, 'I4', LAST, 0, JDEBG )
644:      call KEPV(A(1), 'LPDNP', LLPDNP, NMAT+1, 'I4', LAST, 0, JDEBG )
645:      call KEPV(A(1), 'NATMT', LNATMT, NUC, 'I4', LAST, 0, JDEBG )
646:      call KEPV(A(1), 'IATMT', LIATMT, MLEN, 'I4', LAST, 0, JDEBG )
647:      call KEPV(A(1), 'DNSTP', LDNSTP, MLEN, 'R4', LAST, 0, JDEBG )
648:      if ( LSIZ(LAST).gt.LIMIT ) then
649:        call MEMERR( 'CTXSIN',
650: &      'MATERIAL COMPOSITION HANDLING FOR PHOTON',

```


src/mvp/ctxsin.f

```

651:      &          LSIZ(LAST), LIMIT )
652:      call PRSTOP( 1, 'MEMORY OVER.' )
653:      stop 999
654:      end if
655: C
656:      call MTDATP( A(LIDMAT), A(LMNUC), A(LLPDEN), NMAT, CHA(LNUCID),
657:      &          NUC, A(LINUCT), A(LDENST), MLEN, NPATOM, A(LNATMT),
658:      &          A(LMPATM), A(LLPDNP), A(LIATMT), A(LDNSTP) )
659: C
660:      end if
661: C
662:      if ( JXSKIP.eq.0 ) then
663: C
664: C=====
665: C INPUT & PROCESSING OF CROSS SECTION DATA
666: C=====
667: C
668:      call LABEL( IPR, 'CROSS SECTION LIBRARY INPUT' )
669: C
670: C
671: C-----
672: C KEEP MEMORY FOR CROSS SECTION ARRAY (NEUTRON)
673: C-----
674: C
675:      if ( JNEUT.ne.0 ) then
676:          NMT = 120
677:          NNK = 10
678: C
679:          call KEEPC( 'MATT', LMATT, NUC, 'C136', LASTC, JDEBG )
680:          if ( LASTC.gt.LIMITC+1 ) then
681: c##<2007/03/14:PN3:
682: c##          write(IMG,*) 'XXX Character memory is insufficient',
683:          write(IMG, '(1X,A,A,A)')
684:      &          'XXX(ctxsin) Character memory is insufficient',
685: c##>
686:      &          ' to store neutron library description string',
687:      &          ' in subroutine <CTXSIN>.'
688:          call CNTERR( 'FATAL' )
689:          call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
690:          stop 999
691:      end if
692: C
693:      call KEVP(A(1), 'KLIB1', LKLIB1, NUC*31, 'I4', LAST, 0, JDEBG )
694:      call KEVP(A(1), 'XLIB1', LXLIB1, NUC*11, 'R4', LAST, 0, JDEBG )
695:      call KEVP(A(1), 'KLIB2', LKLIB2, NUC*NMT*21, 'I4', LAST, 0, JDEBG )
696:      &
697:      call KEVP(A(1), 'XLIB2', LXLIB2, NUC*NMT*4, 'R4', LAST, 0, JDEBG )
698:      &
699: C
700: C ... keep CX to start on doubleword boundary.
701: C It is for working area in ARTLIB process.
702: C
703:      call REMAIN( 'CX', MMCX, 'R4D', LAST )
704:      call KEEP( 'CX', LCX, MMCX, 'R4D', LAST, JDEBG )
705: C
706: C
707: C/#IF PARA( PVM MPI )
708:      if ( IDTASK.eq.1 ) then
709: C/#ENDIF
710: C
711:      call XSEC( A(LCX), IA(LCX), MMCX, LCX, MCX, LAST, LIMIT,
712:      &          NMT, CHA(LMATT), IA(LKLIB1), A(LXLIB1),
713:      &          IA(LKLIB2), A(LXLIB2), ETOPL, EBOTL, ESABL, NUC,
714:      &          CHA(LNUCID), CHA(LNCIDI), A(LTEMPN), JVPOP,
715:      &          JXPOOL, ATPool, JDEBG )

```

```

716:
717: C/#IF PARA( PVM MPI )
718:      end if
719: C
720:      if ( NTASK.gt.1 ) then
721:          call TOKEI( TX0, 0 )
722:          IT0 = 1
723:          call MVPCOM_BCAST_I4( IT0, 55, A(LKLIB1), NUC*31, INFO )
724:          call MVPCOM_BCAST_R4( IT0, 55, A(LXLIB1), NUC*11, INFO )
725:          call MVPCOM_BCAST_I4( IT0, 55, A(LKLIB2), NUC*NMT*21,
726:          &          INFO )
727:          call MVPCOM_BCAST_R4( IT0, 55, A(LXLIB2), NUC*NMT*4, INFO )
728:          &
729:          call MVPCOM_BCAST_R4( IT0, 55, ETOPL, 1, INFO )
730:          call MVPCOM_BCAST_R4( IT0, 55, EBOTL, 1, INFO )
731:          call MVPCOM_BCAST_R4( IT0, 55, ESABL, 1, INFO )
732:          call MVPCOM_BCAST_I4( IT0, 55, MCX, 1, INFO )
733:          MBLOCK = 2**15
734:          do 120 K = 1, MCX, MBLOCK
735:              K2 = MIN(MCX,K+MBLOCK-1)
736:              NN = K2 - K + 1
737:              LLL = LCX + K - 1
738:              call MVPCOM_BCAST_I4( IT0, K, A(LLL), NN, INFO )
739:          continue
740:          call TOKEI( TX1, 0 )
741:          write(IPR, '(1x,a,a,i9,a,1p,e12.5)')
742:          &          '=== Elapsed time to broadcast cross section ',
743:          &          ' to tasks (neutron: ', MCX, ' words) : ', TX1
744:          &          - TX0
745:      end if
746: C
747: C/#ENDIF
748:      call RESIZE( 'CX', LCX, MCX, 'R4D', LAST )
749:      end if
750: C
751: C
752: C-----
753: C KEEP MEMORY FOR CROSS SECTION ARRAY (PHOTON)
754: C-----
755: C
756:      if ( JPHOT.ne.0 ) then
757:          MM = 136/8*MWORD
758:          NMTP = 60
759: C
760:      call KEVP( 'MATT', LMATT, NPATOM, 'C136', LAST, 0, JDEBG )
761:      call KEEPC( 'MATT', LMATT, NPATOM, 'C136', LASTC, JDEBG )
762:      if ( LASTC.gt.LIMITC+1 ) then
763: c##<2007/03/14:PN3:
764: c##          write(IMG,*) 'XXX Character memory is insufficient',
765:          write(IMG, '(1X,A,A,A)')
766:      &          'XXX(ctxsin) Character memory is insufficient',
767: c##>
768:      &          ' to store photon library description string',
769:      &          ' in subroutine <CTXSIN>.'
770:          call CNTERR( 'FATAL' )
771:          call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
772:          stop 999
773:      end if
774: C
775:      call KEVP(A(1), 'KLBP1', LKLBP1, NPATOM*20, 'I4', LAST, 0, JDEBG )
776:      &
777:      call KEVP(A(1), 'XLBP1', LXLB1, NPATOM*10, 'R4', LAST, 0, JDEBG )
778:      &
779:      call KEVP(A(1), 'KLBP2', LKLBP2, NPATOM*NMTP*6, 'I4', LAST, 0,
780:      &          JDEBG )

```

src/mvp/ctxsin.f

```

781:      call KEPV(A(1),'XLBP2',LXLBP2,NPATOM*NMTP*2,'R4', LAST, 0,
782:      &          JDEBG )
783: C
784:      call REMAIN( 'CXP', MMCXP, 'R4', LAST )
785:      call KEEP( 'CXP', LCXP, MMCXP, 'R4', LAST, JDEBG )
786: C
787: C/#IF PARA( PVM MPI )
788:      if ( IDTASK.eq.1 ) then
789: C/#ENDIF
790:
791:      call XSECP( A(LCXP), A(LCXP), MMCXP, LCXP, MCXP, LAST,
792:      &          LIMIT, NMTP, CHA(LMATTP), A(LKLB1), A(LXLBP1),
793:      &          A(LKLB2), A(LXLBP2), ETOPLP, EBOTLP, NPATOM,
794:      &          A(LNATMT), JDEBG )
795: C
796: C/#IF PARA( PVM MPI )
797:      end if
798: C
799:      if ( NTASK.gt.1 ) then
800:      call TOKEI( TX0, 0 )
801:      IT0 = 1
802:      call MVPCOM_BCAST_I4( IT0, 56, A(LKLB1), NPATOM*20, INFO
803:      &          )
804:      call MVPCOM_BCAST_R4( IT0, 56, A(LXLBE1), NPATOM*10, INFO
805:      &          )
806:      call MVPCOM_BCAST_I4( IT0, 56, A(LKLB2), NPATOM*NMTP*6,
807:      &          INFO )
808:      call MVPCOM_BCAST_R4( IT0, 56, A(LXLBP2), NPATOM*NMTP*2,
809:      &          INFO )
810:      call MVPCOM_BCAST_R4( IT0, 56, ETOPLP, 1, INFO )
811:      call MVPCOM_BCAST_R4( IT0, 56, EBOTLP, 1, INFO )
812:      call MVPCOM_BCAST_I4( IT0, 56, MCXP, 1, INFO )
813: C
814:      MBLOCK = 2**15
815:      do 130 K = 1, MCXP, MBLOCK
816:      K2 = MIN(MCXP,K+MBLOCK-1)
817:      NN = K2 - K + 1
818:      LLL = LCXP + K - 1
819:      call MVPCOM_BCAST_I4( IT0, K, A(LLL), NN, INFO )
820: 130 continue
821:      call TOKEI( TX1, 0 )
822:      write(IPR,'(1x,a,a,i9,a,1p,e12.5)')
823:      &          '=== Elapsed time to broadcast cross section ',
824:      &          ' to tasks (photon: ', MCXP, ' words) : ', TX1
825:      &          - TX0
826:      end if
827: C
828: C/#ENDIF
829:      call RESIZE( 'CXP', LCXP, MCXP, 'R4', LAST )
830: C
831:      end if
832: C
833: C-----
834: C      KEEP MEMORY FOR CROSS SECTION ARRAY (ELECTRON)
835: C-----
836: C
837:      if ( JBREM.ne.0 ) then
838: C
839:      NEE = 134
840:      MTOP = 49
841:      NEMTP = NEE*MTOP
842:      NEMTP1 = NEE*(MTOP-1)
843: C
844:      call KEPV(A(1),'KLBE1',LKLBE1,NPATOM*20,'I4',LAST,0, JDEBG
845:      &          )

```

```

846:      call KEPV(A(1),'XLBE1',LXLBE1,NPATOM*6,'R4', LAST,0, JDEBG )
847:      call KEPV(A(1),'EEL ',LEEL,NEE, 'R4', LAST, 0, JDEBG )
848:      call KEPV(A(1),'RKT ',LRKT,MTOP, 'R4', LAST, 0, JDEBG )
849:      call KEPV(A(1),'PBR ',LPBR,NEE*NMAT, 'R4', LAST, 0, JDEBG )
850:      call KEPV(A(1),'EBA ',LEBA,NEMTP*NMAT,'R4',LAST, 0, JDEBG )
851:      call KEPV(A(1),'PBT ',LPBT,NEE*NMAT,'R4',LAST, 0, JDEBG )
852:      call KEPV(A(1),'EBT ',LEBT,NEMTP*NMAT,'R4',LAST, 0, JDEBG )
853:      call KEPV(A(1),'RNG ',LRNG,NEE*NMAT,'R4',LAST, 0, JDEBG )
854:      call KEPV(A(1),'EBTP ',LEBTP,NEMTP1*NMAT,'R4',LAST,0, JDEBG
855:      &          )
856:      call KEPV(A(1),'IEBTP',LIEBTP, NEMTP1*NMAT*2, 'I4', LAST, 0,
857:      &          JDEBG )
858:      call KEPV(A(1), 'FME ', LFME, MLEN, 'R4', LAST, 0, JDEBG )
859:      call KEPV(A(1), 'WWAG ', LWWAG, NMAT, 'R4', LAST, 0, JDEBG )
860:      call KEPV(A(1), 'EEDG ', LEEDG, NMAT, 'R4', LAST, 0, JDEBG )
861:      call KEPV(A(1), 'EEEK ', LEEEEK, NMAT, 'R4', LAST, 0, JDEBG )
862: C
863: C
864:      call REMAIN( 'CXE', MMCXE, 'R4', LAST )
865:      call KEEP( 'CXE', LCXE, MMCXE, 'R4', LAST, JDEBG )
866: C
867: C/#IF PARA( PVM MPI )
868:      if ( IDTASK.eq.1 ) then
869: C/#ENDIF
870: C
871:      call XSECE( A(LCXE), MMCXE, LCXE, MCXE, LAST, LIMIT, NEE,
872:      &          MTOP, A(LKLBE1), A(LXLBE1), A(LEEL), A(LRKT),
873:      &          A(LPBR), A(LEBA), A(LPBT), A(LEBT), A(LRNG),
874:      &          A(LEBTP), A(LIEBTP), ETOPLP, EBOTLP, MLEN,
875:      &          NPATOM, NMAT, A(LNATMT), A(LMPATM), A(LLPDNP),
876:      &          A(LIATMT), A(LDNSTP), A(LFME), A(LWWAG),
877:      &          A(LEEDG), A(LEEEK), JDEBG )
878: C
879: C/#IF PARA( PVM MPI )
880:      end if
881: C
882:      if ( NTASK.gt.1 ) then
883:      call TOKEI( TX0, 0 )
884:      IT0 = 1
885:      call MVPCOM_BCAST_I4( IT0, 57, A(LKLB1), NPATOM*20, INFO
886:      &          )
887:      call MVPCOM_BCAST_R4( IT0, 57, A(LXLBE1), NPATOM*6, INFO
888:      &          )
889:      call MVPCOM_BCAST_R4( IT0, 57, A(LEEL), NEE, INFO )
890:      call MVPCOM_BCAST_R4( IT0, 57, A(LRKT), MTOP, INFO )
891:      call MVPCOM_BCAST_R4( IT0, 57, A(LPBR), NEE*NMAT, INFO )
892:      call MVPCOM_BCAST_R4( IT0, 57, A(LEBA), NEMTP*NMAT, INFO
893:      &          )
894:      call MVPCOM_BCAST_R4( IT0, 57, A(LPBT), NEE*NMAT, INFO )
895:      call MVPCOM_BCAST_R4( IT0, 57, A(LEBT), NEMTP*NMAT, INFO
896:      &          )
897:      call MVPCOM_BCAST_R4( IT0, 57, A(LRNG), NEE*NMAT, INFO )
898:      call MVPCOM_BCAST_R4( IT0, 57, A(LEBTP), NEMTP1*NMAT,
899:      &          INFO )
900:      call MVPCOM_BCAST_R4( IT0, 57, A(LIEBTP), NEMTP1*NMAT*2,
901:      &          INFO )
902:      call MVPCOM_BCAST_R4( IT0, 57, A(LFME), MLEN, INFO )
903:      call MVPCOM_BCAST_R4( IT0, 57, A(LWWAG), NMAT, INFO )
904:      call MVPCOM_BCAST_R4( IT0, 57, A(LEEDG), NMAT, INFO )
905:      call MVPCOM_BCAST_R4( IT0, 57, A(LEEEK), NMAT, INFO )
906:      call MVPCOM_BCAST_I4( IT0, 57, MCXE, 1, INFO )
907: C
908:      MBLOCK = 2**15
909:      do 140 K = 1, MCXE, MBLOCK
910:      K2 = MIN(MCXE,K+MBLOCK-1)

```

src/mvp/ctxsin.f

```

911:      NN      = K2 - K + 1
912:      LLL     = LCXE + K - 1
913:      call MVPCOM_BCAST_I4( IT0, K, A(LLI), NN, INFO )
914: 140      continue
915:      call TOKEI( TX1, 0 )
916:      write(IPR,'(1x,a,a,i9,a,1p,e12.5)')
917:      &      '=== Elapsed time to broadcast cross section ',
918:      &      ' to tasks (electron: ', MCXE, ' words) : ', TX1
919:      &      - TX0
920:      end if
921: C/#ENDIF
922:
923:      call RESIZE( 'CXE', LCXE, MCXE, 'R4', LAST )
924: C
925:      end if
926: c##<2007/03/14:PN3:
927: C
928: C-----
929: C      KEEP MEMORY FOR CROSS SECTION ARRAY (PHOTO-NUCLEAR)
930: C-----
931: C
932:      if ( JPHNU.ne.0 ) then
933:      NMTPN = 135
934:      NUCPNI = 0
935:      do I = 1, NPATOM
936:      NA = IA(LNATMT+I-1)
937:      NUCPNI = NUCPNI + ISCTOP(NA)
938:      end do
939:      if ( NUCPNI.le.0 ) NUCPNI = NUC
940:      if ( NUCPNI.le.30 ) then      ! space for duplicated nuclide
941:      NUCPNI = NUCPNI + 10
942:      else
943:      NUCPNI = NUCPNI + 20
944:      end if
945:      call KEEP( 'MATTPN', LMATTPN, NUCPNI, 'C136', LASTC, JDEBG )
946:      if ( LASTC.gt.LIMITC+1 ) then
947:      HERRMES = 'photo-nuclear library description string.'
948:      go to 930
949:      end if
950:      call KEEP( 'NCIDPN', LNCIDPN, NUCPNI, 'C16', LASTC, JDEBG )
951:      if ( LASTC.gt.LIMITC+1 ) then
952:      HERRMES = 'photo-nuclear library ID string <NCIDPN>'
953:      go to 930
954:      end if
955:      call KEPV(A(1),'KLBN1',LKLBN1,NUCPNI*16,'I4',LAST,0,JDEBG)
956:      call KEPV(A(1),'XLBN1',LXLBN1,NUCPNI*6, 'R4',LAST,0,JDEBG)
957:      call KEPV(A(1),'KLBN2',LKLBN2,NUCPNI*NMTPN*13,
958:      &      'I4',LAST,0,JDEBG)
959:      call KEPV(A(1),'XLBN2',LXLBN2,NUCPNI*NMTPN*2,
960:      &      'R4',LAST,0,JDEBG)
961:      call REMAIN( 'CXPN', MMCXPN, 'R4', LAST )
962:      call KEEP( 'CXPN', LCXPN, MMCXPN, 'R4', LAST, JDEBG )
963: C
964: C/#IF PARA( PVM MPI )
965:      if ( IDTASK.eq.1 ) then
966: C/#ENDIF
967:      call XSECPN( A(LCXPN), IA(LCXPN), MMCXPN, LCXPN, MCXPN,
968:      &      LAST, LIMIT, NMTPN, CHA(LMATTPN), IA(LKLBN1),
969:      &      A(LXLBN1), IA(LKLBN2), A(LXLBN2), ETOTLPN,
970:      &      EBOTLPN, NUC, NUCPNI, CHA(LNCIDP), CHA(LNCIDPN),
971:      &      NMAT, MLEN, IA(LMNUC), IA(LLPDEN), IA(LINUCT),
972:      &      NUCPNA, NUCPNB, NUCPN, JDEBG, JPNLCP )
973: C/#IF PARA( PVM MPI )
974:      end if
975: C

```

```

976:      if ( NTASK.gt.1 ) then
977:      call TOKEI( TX0, 0 )
978:      IT0 = 1
979:      call MVPCOM_BCAST_I4(IT0,58,A(LKLBN1),NUCPNI*16,INFO)
980:      call MVPCOM_BCAST_R4(IT0,58,A(LXLBN1),NUCPNI*6, INFO)
981:      call MVPCOM_BCAST_I4(IT0,58,A(LKLBN2),NUCPNI*NMTPN*13,
982:      &      INFO)
983:      call MVPCOM_BCAST_R4(IT0,58,A(LXLBN2),NUCPNI*NMTPN*2,
984:      &      INFO)
985:      call MVPCOM_BCAST_R4(IT0,58,ETOTLPN, 1, INFO)
986:      call MVPCOM_BCAST_R4(IT0,58,EBOTLPN, 1, INFO)
987:      call MVPCOM_BCAST_I4(IT0,58,MCXPN, 1, INFO)
988:      MBLOCK = 2**15
989:      do K = 1, MCXPN, MBLOCK
990:      K2 = min( MCXPN, K+MBLOCK-1 )
991:      NN = K2 - K + 1
992:      LLL = LCXPN + K - 1
993:      call MVPCOM_BCAST_I4(IT0,K,A(LLI),NN,INFO)
994:      end do
995:      call TOKEI( TX1, 0 )
996:      write(IPR,'(1x,a,a,i9,a,1p,e12.5)')
997:      &      '=== Elapsed time to broadcast cross section ',
998:      &      ' to tasks (photo-nuclear: ', MCXPN, ' words) : ',
999:      &      TX1-TX0
1000:      end if
1001: C/#ENDIF
1002:      call KEPV(A(1),'MNUCPN',LMNUCPN,NMAT+1, 'I4',LAST,0,JDEBG)
1003:      call KEPV(A(1),'NNUCPN',LNUCPN,NUCPNB*NMAT,
1004:      &      'I4',LAST,0,JDEBG)
1005:      call KEPV(A(1),'LPDNP',LLPDNP,NUCPNB*NMAT,
1006:      &      'I4',LAST,0,JDEBG)
1007:      call KEPV(A(1),'DNSTPN',LDNSTPN,NUCPNA, 'R4',LAST,0,JDEBG)
1008:      call KEPV(A(1),'LPIDPN',LLPIDPN,NUCPNA, 'I4',LAST,0,JDEBG)
1009:      call KEPV(A(1),'LIZBPN',LIZBPN,NUCPNA, 'I4',LAST,0,JDEBG)
1010:      call KEPV(A(1),'MNEUTPN',LMNEUTPN,NUCPNA, 'I4',LAST,0,JDEBG)
1011: C
1012:      call MTDATPN(NUCPNA, NUCPNB, NUCPN, NMAT, MLEN, IA(LMNUC),
1013:      &      IA(LLPDEN), IA(LINUCT), A(LDENST), NUC, NUCPNI,
1014:      &      CHA(LNCIDP), CHA(LNCIDPN), IA(LMNUCPN),
1015:      &      IA(LNUCPN), IA(LLPDNP), A(LDNSTPN),
1016:      &      IA(LLPIDPN), IA(LIZBPN), IA(LMNEUTPN))
1017:      end if
1018: c##>
1019: C
1020: C-----
1021: C      KEEP MEMORY FOR DOPPLER SCATTERING
1022: C-----
1023: C
1024:      if ( JEXDP.ne.0 ) then
1025:      call KEPV(A(1), 'INTDS', LINTDS, NUC, 'I4',LAST,0,JDEBG)
1026:      call MKNTDS(NUC, NMT, A(LKLBN2), A(LINTDS))
1027:      end if
1028:
1029:      else
1030:      write(IPR,'(1x,a)') '=== Cross section input is skipped.'
1031:      end if
1032: C
1033:      return
1034: C
1035: c##<2007/03/14:PN3:
1036: c#150 write(IMG,'(1x,a,i10,a)')
1037: c## &      'XXX MEMORY OVER IN CTXSIN !!! (LIMIT = ', LIMIT, ' STOP '
1038:      910 write(IMG,'(1x,a,i10,a)')
1039:      &      'XXX(CTXSIN) MEMORY OVER !!! (LIMIT = ', LIMIT, ' ) STOP '
1040:      call CNTERR( 'FATAL' )

```

src/mvp/ctxsin.f

```
1041: c##>
1042:     call PRSTOP( 1, 'MEMORY OVER.' )
1043:     stop 999
1044: c##<2007/03/14:PN3:
1045: 930 write(IMG,'(lx,a,a,a)')
1046:     &      'XXX(CTXSIN) Character memory is insufficient',
1047:     &      ' to store ',HERMES
1048:     call CNTERR( 'FATAL' )
1049:     call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
1050: 999 stop 999
1051: c##>
1052:
1053: end
```

FAIL

src/mvp/cxsint.f

```
1:      SUBROUTINE CXSINT(NN,IBP, N,MT,
2:      X          SMIC, CX, MCX, KLIB2,NUC,NMT,
3:      W          IEP ,DE, DE1, NBANK)
4: C=====
5: C  PURPOSE: GET CROSS SECTIONS FROM CONTINUOUS LIBRARY BY INTERPOLATION
6: C          ( obsolete in current version )
7: C  CALLED IN:  none ( *GETMIC* )
8: C=====
9: C
10: C   N : NUCLIDE #      MT : REACTION NUMBER IN LIBRARAY
11: C   N2  : = LNU*NNU
12: C   NPTS : TOTAL NUMBER OF ENERGY POINTS
13: C
14: C   .... NN: NUMBER OF PARTICLES  IBP: POINTER  E: ENERGY .....
15: C   INTEGER      IBP(NN)
16: C
17: C   .... BANK .....
18: C   REAL          SMIC(NBANK)
19: C
20: C   .... CROSS SECTION & POINTERS .....
21: C   REAL          CX(MCX)
22: C##<2007/03/14:PN3:
23: C##  INTEGER      KLIB2(NUC,NMT,13)
24: C##  INTEGER      KLIB2(NUC,NMT,21)
25: C##>
26: C
27: C   .... ENERGY MESH POINT # .....
28: C   INTEGER      IEP(NBANK)
29: C   REAL          DE(NBANK),DE1(NBANK)
30: C-----
31: C
32: C   L0          = KLIB2(N,MT,11) - KLIB2(N,MT,3)
33: C *VOCL LOOP,NOVREC
34: C   DO 100 I=1,NN
35: C     IF(IEP(I).GE.KLIB2(N,MT,3).AND. IEP(I).LT.KLIB2(N,MT,4)) THEN
36: C       LL      = L0 + IEP(I)
37: C       SMIC(IBP(I)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
38: C     ELSE
39: C       SMIC(IBP(I)) = 0.0
40: C     ENDIF
41: C   100 CONTINUE
42: C   RETURN
43: C   END
```

src/mvp/delayed.f

```

1:      subroutine DELAYPN( IOW, JTIME, NBANK, IRAND, KDDD, KN, EIND, XMU,
2:      D TOUT,
3:      X MCXPN, CXPN, ICXPN, KLBPN1, KLBPN2, NUCPNI, NMTPN,
4:      W R, IWK2, IWK8, IWK9, IWK10 )
5: C=<MVP>=====
6: C purpose: sample the energy and direction for delayed neutron, in
7: C photo-nuclear fission.
8: C called in: PHOTNUC
9: C calls: RANU2, BSVDEC, BSDEC3
10: C=====
11: C arguments (variable) :
12: C KDDD .... number of delayed neutrons producing in here.
13: C KN ..... sampled fissile nuclide's number.
14: C EIND .... incident energy (input) and outgoing energy (output)
15: C XMU ..... scattering angle (cosine); output
16: C TOUT .... delayed time (output), if time-dependent.
17: C=====
18: C
19: C implicit real*8 (A-H,O-Z)
20: C
21: C parameter (SMALL = 1.0D-35)
22: C parameter (MTD = 98) ! MT for delayed neutron data
23: C parameter (MTF = 18) ! MT for fission
24: C
25: C ..... cross section data
26: C real CXPN(MCXPN)
27: C integer ICXPN(MCXPN), KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
28: C
29: C ..... temporary data area
30: C real EIND(*) ! incident and outgoing energy
31: C real XMU(*) ! scattering angle
32: C real TOUT(*) ! delayed time
33: C
34: C ..... working area
35: C real R(4*NBANK)
36: C integer IWK2(NBANK), IWK8(NBANK), IWK9(NBANK), IWK10(NBANK)
37: C
38: C ..... local data
39: C data N30 / 30 /
40: C
41: C=====
42: C
43: C
44: C NNF = KLBPN1(KN,13)
45: C LSNUD = KLBPN1(KN,14)
46: C LST2G = KLBPN2(KN,MTD,12)
47: C NE2G = ICXPN(LST2G)
48: C NK2GMT = ICXPN(LST2G+1)
49: C IEPROB = LST2G + 2 + 2*N30 + 5*NK2GMT
50: C if ( NK2GMT.le.1.or.NK2GMT.ne.NNF ) go to 901
51: C
52: C-----
53: C**** select the sampling subsection of delayed neutron.
54: C-----
55: C
56: C call BSVDEC( CXPN(IEPROB), NE2G, EIND, IWK2, KDDD )
57: C call RANU2( IRAND, R, KDDD*3, ICON )
58: C IM2 = 2 * KDDD
59: C do I = 1, KDDD
60: C if ( IWK2(I).le.0 ) IWK2(I) = 1
61: C end do
62: C *VOCL LOOP,NOVREC
63: C do I = 1, KDDD
64: C L2 = IEPROB + IWK2(I)
65: C L1 = L2 - 1

```

```

66:      X1 = (CXPN(L1)-EIND(I)) / (CXPN(L1)-CXPN(L2))
67:      X1 = min(1.0D0, max(0.0D0, X1))
68:      IT = min(1, int(R(I)+X1)) + IWK2(I)
69:      L1 = IEPROB + 2*NE2G + 3*(IT-1)*NK2GMT - 1
70:      L2 = min(int(NK2GMT*R(I+KDDD))+1, NK2GMT)
71:      L3 = L1 + L2
72:      L4 = min(2*L2, int(2*L2+R(I+IM2)-CXPN(L3))) + L1 + NK2GMT
73:      IWK2(I) = ICXPN(L4) ! selected subsection number
74:      end do
75: C
76: C-----
77: C**** outgoing angle (cosine) of delayed neutron.
78: C-----
79: C
80: C MT = MTD
81: C NEANG = KLBPN2(KN,MT,6)
82: C if ( NEANG.eq.0.and.KLBPN2(KN,MTF,6).gt.0 ) then
83: C MT = MTF
84: C NEANG = KLBPN2(KN,MT,6)
85: C end if
86: C if ( NEANG.eq.0 ) then ! no data ---> isotropic
87: C call RANU2( IRAND, R, KDDD, ICON )
88: C do I = 1, KDDD
89: C XMU(I) = 1 - 2*R(I)
90: C end do
91: C else ! equal-probability bins
92: C NBINA = KLBPN1(KN,5)
93: C LSTF4 = KLBPN2(KN,MT,11)
94: C do I = 1, KDDD
95: C IWK8(I) = LSTF4
96: C IWK9(I) = NEANG
97: C end do
98: C call BSDEC3( CXPN, IWK9, R, IWK8, EIND, IWK10, KDDD )
99: C call RANU2( IRAND, R, KDDD*3, ICON )
100: C do I = 1, KDDD
101: C if ( IWK10(I).eq.0 ) IWK10(I) = 1
102: C L2 = IWK8(I) + IWK10(I)
103: C L1 = L2 - 1
104: C INTE = mod(ICXPN(L2+IWK9(I)), 10)
105: C if ( INTE.eq.5.or.INTE.eq.3 ) then
106: C X1 = log(CXPN(L1)/EIND(I))/log(CXPN(L1)/CXPN(L2))
107: C else
108: C X1 = (CXPN(L1)-EIND(I))/(CXPN(L1)-CXPN(L2))
109: C end if
110: C X1 = min(1.0D0, max(0.0D0, X1))
111: C IT = min(1, int(R(I)+X1)) + IWK10(I)
112: C L3 = min(int(NBINA*R(I+KDDD))+1, NBINA) + IWK8(I) +
113: C & 2*IWK9(I) + (NBINA+1)*(IT-1)
114: C XMU(I) = (CXPN(L3)-CXPN(L3-1))*R(I+KDDD*2) + CXPN(L3-1)
115: C end do
116: C end if
117: C
118: C-----
119: C**** outgoing energy of delayed neutron.
120: C-----
121: C
122: C do I = 1, KDDD
123: C LST2GS = ICXPN(LST2G+1+2*N30+IWK2(I))
124: C LF = ICXPN(LST2GS)
125: C IWK8(I) = LST2GS + 2 ! starting position of EE
126: C
127: C IWK9(I) = ICXPN(LST2GS+1) ! NEF5S
128: C if ( LF.ne.1.and.LF.ne.5 ) go to 903
129: C end do
130: C call BSDEC3( CXPN, IWK9, R, IWK8, EIND, IWK10, KDDD )

```

src/mvp/delayed.f

```

130:      N7RR = 6 * KDDD
131:      if ( LF.eq.1 ) then
132:        call RANU2( IRAND, R, KDDD*4, ICON )
133:      elseif ( LF.eq.5 ) then
134:        call RANU2( IRAND, R, KDDD*6, ICON )
135:        N7R = 3 * KDDD
136:      end if
137:      do I = 1, KDDD
138:        if ( IWK10(I).eq.0 ) IWK10(I) = 1
139:        L2 = IWK8(I) + IWK10(I)
140:        L1 = L2 - 1
141:        INTE0 = ICXPN(L2+IWK9(I))
142:        INTE1 = INTE0 / 10
143:        INTE = INTE0 - 10 * INTE1
144:        if ( INTE.eq.5.or.INTE.eq.3 ) then
145:          X1 = log(CXPN(L1)/EIND(I))/log(CXPN(L1)/CXPN(L2))
146:        else
147:          X1 = (CXPN(L1)-EIND(I))/(CXPN(L1)-CXPN(L2))
148:        end if
149:        X1 = min(1.0D0, max(0.0D0, X1)) ! fraction in incident en
150:        L1 = IWK8(I) + 2 * IWK9(I) ! starting position of LS
151:        EE0 = 0
152:        if ( LF.ne.1 ) go to 110
153: C ..... LF=1 .... arbitrary tabulated function
154:        IT = min(1, int(R(I)+X1)/ + IWK10(I))
155:        LSTF5E = ICXPN(L1+IT-1)
156:        NEP = ICXPN(LSTF5E)
157:        LL2 = min(int(NEP*R(I+KDDD))+1, NEP)
158:        LL3 = LSTF5E + LL2
159:        LL4 = min(2*LL2,int(2*LL2+R(I+KDDD*2)-CXPN(LL3)))+LSTF5E+NEP
160:        LLL = LSTF5E + 3 * NEP
161:        L3 = LLL + ICXPN(LL4)
162:        L4 = L3 + NEP + 1
163:        LLE = LLL + 1
164:        if ( IT.eq.IWK10(I) ) then
165:          NEP1 = NEP
166:          LLE1 = LLE
167:          LSTF5T = ICXPN(L1+IT)
168:          NEP2 = ICXPN(LSTF5T)
169:          LLE2 = LSTF5T + 3*NEP2 + 1
170:        else
171:          NEP2 = NEP
172:          LLE2 = LLE
173:          LSTF5T = ICXPN(L1+IT-2)
174:          NEP1 = ICXPN(LSTF5T)
175:          LLE1 = LSTF5T + 3*NEP1 + 1
176:        end if
177:        E0 = CXPN(L3)
178:        E1 = CXPN(L3+1)
179:        P0 = CXPN(L4)
180:        P1 = CXPN(L4+1)
181:        RRR = (P0 + P1) * R(I+KDDD*3)
182:        EE = E1 + (E0-E1)*RRR / (sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
183:        if ( INTE1.eq.2 ) then ! linear interpola
184:          EMAX = CXPN(LLE1) + X1 * (CXPN(LLE2)-CXPN(LLE1))
185:          EE0 = EE * EMAX / CXPN(LLE)
186:        else if ( INTE1.eq.1 ) then ! linear interpola
187:          EMAX = CXPN(LLE1) + X1 * (CXPN(LLE2)-CXPN(LLE1))
188:          EMIN = CXPN(LLE1+NEP1)+X1*(CXPN(LLE2+NEP2)-CXPN(LLE1+NEP1))
189:          EE0 = EMIN + (EMAX-EMIN) * (EE-CXPN(LLE+NEP)) /
190:            & (CXPN(LLE)-CXPN(LLE+NEP))

```

```

191:      end if
192:      go to 120
193: C ..... LF=5 .... general evaporation spectrum
194: 110 if ( LF.ne.5 ) go to 120
195:      L5 = L1 + IWK10(I)
196:      THETA5 = CXPN(L5)*X1 + CXPN(L5-1)*(1-X1)
197:      LL = L1 + IWK9(I)
198:      NEP = ICXPN(LL)
199:      U5 = CXPN(LL+5*NEP+3)
200:      EHI5 = EIND(I) - U5
201: 115 LL5 = min(int(NEP*R(I))+1, NEP)
202:      LL6 = LL + LL5
203:      LL7 = 2 * LL5
204:      LL7 = min(LL7, int(LL7+R(I+KDDD)-CXPN(LL6))) + LL + NEP
205:      L6 = LL + 3 * NEP + ICXPN(LL7)
206:      L7 = L6 + NEP + 1
207:      E0 = CXPN(L6)
208:      E1 = CXPN(L6+1)
209:      P0 = CXPN(L7)
210:      P1 = CXPN(L7+1)
211:      RRR = (P0 + P1) * R(I+KDDD*2)
212:      X2 = E1 + (E0-E1)*RRR / (sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
213:      EE0 = X2 * THETA5
214:      if ( EE0.gt.EHI5 ) then
215:        R(I) = R(N7R+1)
216:        R(I+KDDD) = R(N7R+2)
217:        R(I+KDDD*2) = R(N7R+3)
218:        N7R = N7R + 3
219:        if ( N7R+3.gt.N7RR ) then
220:          N7R = 3 * KDDD
221:          call RANU2( IRAND, R(N7R+1), N7R, ICON )
222:        end if
223:        go to 115
224:      end if
225: C
226: 120 continue
227:      EIND(I) = EE0 ! outgoing energy
228:    end do
229: C
230: C-----
231: C**** delayed time by decay constant.
232: C-----
233: C
234:      if ( JTIME.ne.0 ) then
235:        LSRAMDA = LSNUD + KLBPN1(KN,11) * KLBPN1(KN,12) - 1
236:        call RANU2( IRAND, R, KDDD, ICON )
237:        do I = 1, KDDD
238:          RAMDA = CXPN(LSRAMDA+IWK2(I))
239:          TOUT(I) = - log(1-R(I)) / RAMDA
240:        end do
241:      else
242:        do I = 1, KDDD
243:          TOUT(I) = 0
244:        end do
245:      end if
246: C
247:      return
248: C
249: C ..... error process
250:      901 write(IOW,902) NK2GMT, NE2G, LST2G, NNF
251:      902 format(/' XXX(delaypn) The number of subsections in energy',
252:        & ' distribution of delayed neutron is not allowed.'
253:        &/' NK2GMT=',i8,' NE2G=',i8,' LST2G=',i9,' NNF=',i8)
254:      go to 999
255:      903 write(IOW,904) LF, LST2GS, LST2G, KN, KDDD, EIND(I), I, IWK2(I)

```

src/mvp/delayed.f

```
256: 904 format('// XXX(delaypn) LF is not 1 (arbitrary tabulated',  
257: & ' function) or 5 (general evaporation sepctrum).'  
258: & '// LF=',i4,' LST2GS=',i9,' LST2G=',i9,' KN=',i5,' KDDD=',  
259: & i6,' incident energy=',lpe12.5,' I=',i6,' subsection=',i4)  
260: 999 stop 666  
261: end
```

SAE

src/mvp/dentb2.f

```
1:      subroutine DENTB2( JNEUT, JPHOT, DNZON, NUC,   NZONE, KZMAT,
2:      &                  INUCT, DENST, LPDEN, NMAT,   NUCID, NATMT,
3:      &                  NPATOM,IATMT, LPDNP, DNSTP, DNREG, RVOL,   VOL,
4:      &                  NREG,  NINPZ, KREG,  KMAT )
5: C=<MVP>=====
6: C  MAKE NUMBER DENSITY TABLE FOR EACH ZONE & EACH REGION
7: C  CALLED IN: INTRO2
8: C
9: C=====
10:      real    DNZON(NUC,NZONE,2), DENST(*), DNSTP(*)
11:      real    DNREG(NUC,NREG,2),  RVOL(NREG), VOL(NINPZ)
12:      integer LPDEN(NMAT+1), INUCT(*), KZMAT(NZONE,2), KREG(NINPZ),
13:      &        KMAT(NINPZ)
14:      integer NATMT(NPATOM), LPDNP(NMAT+1), IATMT(*)
15:      character*16 NUCID(NUC)
16: C
17:      external CHSYMB
18:      character*2 CHSYMB
19: C
20:      include '../shared/INC/_PMLATT'
21:      include '../shared/INC/_SFLATT'
22: C
23: C-----
24: C  NUMBER DENSITY OF EACH ZONE
25: C-----
26: C
27:      do 120 N = 1, NZONE
28:          MAT = KZMAT(N,1)
29: CCCCC  if ( MAT.le.-1.and.MAT.ge.-998 ) MAT = KZMAT(N,2)
30:          if ( ISLAT(MAT) ) MAT = KZMAT(N,2)
31:          if ( MAT.ge.1.and.MAT.le.NMAT ) then
32:              if ( JNEUT.ne.0 ) then
33:                  do 100 M = LPDEN(MAT), LPDEN(MAT+1) - 1
34: C##<2007/03/14:PN3:
35: C##          DNZON( INUCT(M),N,1) = DENST(M)
36: C##          DNZON( INUCT(M),N,1) = DNZON( INUCT(M),N,1) + DENST(M)
37: C##>
38:              100      continue
39:              end if
40:              if ( JPHOT.ne.0 ) then
41:                  do 110 M = LPDNP(MAT), LPDNP(MAT+1) - 1
42:                      DNZON(IATMT(M),N,2) = DNSTP(M)
43:              110      continue
44:              end if
45:              end if
46:          120 continue
47: C
48:      return
49:      end
```

src/mvp/dumpbk.f

```
1:      subroutine DUMPBK( N1,N2, IPRT,  CHAR, A, IA, H, IH )
2: C=====
3: C  PURPOSE : PRINTOUT OF PARTICLE BANK. (FOR DEBUG ONLY)
4: C      (BANK #  N1 TO N2 )
5: C      WHEN N1 = 0 : ALL SURVIVING PARTICLE.
6: C      N1 = -1 : ALL PARTICLES IN BANK.
7: C CALLED IN : VARIOUS ROUTINES ( or never be called I hope :- )
8: C=====
9: CC      include '../shared/INC/_ARRAY'
10:      real A(*)
11:      integer IA(*)
12:      real H(*)
13:      integer IH(*)
14: C
15:      character*(*) CHAR
16:      include 'INC/_XBANK'
17:      include 'INC/_FLAGS'
18:      include '../shared/INC/_SIZES'
19:      include 'INC/_SIZES2'
20:      include 'INC/_STACK'
21: C
22: C ..... COMMENT .....
23:      if ( CHAR.ne.' ' ) write(IPRT,'(1x,A)' ) CHAR
24: C
25: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP,
26: C
27:      call DUMPXX( N1,N2,IPRT,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),
28:      &      H(LCCC), H(LWWW), H(LEEE), H(LIZZ), H(LIGG) )
29: C
30: C === LATTICE GEOMETRY ===
31: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP, LEVEL, POS, CRS
32: C
33:      if ( JLATT.ne.0 ) then
34:      ML      = 25
35:      do 110 K = 1, NEST
36:      write(IPRT,7000) K
37:      do 100 L = N1, N2, ML
38:      L2      = MIN(L+ML-1,N2)
39:      K1      = NBANK*(K-1) + L - 1
40:      K2      = NBANK*(K-1) + L2 - 1
41:      write(IPRT,7020) ' --- ', (I,I=L,L2)
42:      write(IPRT,7020) 'LEVL ', (H(LLEVL+I),I=K1,K2)
43:      write(IPRT,7020) 'LZZ ', (H(LLZZ+I),I=K1,K2)
44:      write(IPRT,7020) 'LPOS ', (H(LLPOS+I),I=K1,K2)
45:      if ( JHLAT.ne.0 ) then
46:      write(IPRT,7020) 'LCRS ', (H(LLCRS+I),I=K1,K2)
47:      end if
48:      100      continue
49:      110      continue
50:      end if
51: C
52: C 7000 format(('      LEVEL ',I2,'          ',10X:))
53: C 7020 format(1X,A5,25I5)
54: C
55:      return
56:      end
```

src/mvp/dumpst.f

```

1:      subroutine DUMPST( CHAR, IPRT,  IH )
2: C=====
3: C  PURPOSE : PRINTOUT OF STACK DATA. (FOR DEBUG ONLY)
4: C  CALLED IN : VARIOUS ROUTINES (or never be called I hope :-) )
5: C=====
6: CC      include '../shared/INC/_ARRAY'
7:      integer IH(*)
8: C
9:      character*(*) CHAR
10:     include 'INC/_FLAGS'
11:     include 'INC/_STACK'
12:     include '../shared/INC/_SIZES'
13:     include 'INC/_SIZES2'
14:     include 'INC/_XBANK'
15: C
16: C ..... COMMENT .....
17: C
18:     write(IPRT, '(A)') CHAR
19: C
20: C ..... FLIGHT STACK ....
21: C
22:     ML      = 20
23:     write(IPRT,7000) 'NFFL ', (IH(LNFFL+I),I=0,NZONE)
24:     MM      = IH(LNFFL+NZONE)
25: C
26:     do 100 K1 = 1, MM, ML
27:         K2      = MIN(K1+ML-1,MM) - 1
28:         write(IPRT,7020) 'LSFFL', (IH(LLSFFL+I),I=K1-1,K2)
29:         write(IPRT,7020) 'IZFFL', (IH(LIZFFL+I),I=K1-1,K2)
30:         write(IPRT,7020) 'IZZ ', (IH(LIZZ+IH(LLSFFL+I)-1),I=K1-1,K2)
31:     100 continue
32: C
33: C ..... SEARCH STACK ....
34: C
35:     write(IPRT,7000) 'NNXT ', (IH(LNNXT+I),I=0,NZONE)
36:     MM      = IH(LNNXT+NZONE)
37:     do 110 K1 = 1, MM, ML
38:         K2      = MIN(K1+ML-1,MM) - 1
39:         write(IPRT,7020) 'LSSRC', (IH(LLSSRC+I),I=K1-1,K2)
40:         write(IPRT,7020) 'IZNXT', (IH(LIZNXT+I),I=K1-1,K2)
41:         write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSSRC+I)-1),I=K1-1,K2)
42:     110 continue
43: C
44: C ..... COLLISION STACK ....
45: C
46:     write(IPRT,7000) 'NCOLS', NCOLS
47:     MM      = NCOLS
48:     do 120 K1 = 1, MM, ML
49:         K2      = MIN(K1+ML-1,MM) - 1
50:         write(IPRT,7020) 'LSCOL', (IH(LLSCOL+I),I=K1-1,K2)
51:         write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSCOL+I)-1),I=K1-1,K2)
52:     120 continue
53: C
54:     write(IPRT,7000) 'NCOLP', NCOLP
55:     MM      = NCOLP
56:     do 130 K1 = 1, MM, ML
57:         K2      = MIN(K1+ML-1,MM) - 1
58:         write(IPRT,7020) 'LSCLP', (IH(LLSCLP+I),I=K1-1,K2)
59:         write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSCLP+I)-1),I=K1-1,K2)
60:     130 continue
61: C
62: C ..... DEAD PARTICLE STACK ....
63: C
64:     write(IPRT,7000) 'NDEAD', NDEAD
65: C      MM = NDEAD

```

```

66: C      DO 260 K1=1,MM,ML
67: C          K2 = MIN(K1+ML-1,MM) - 1
68: C          WRITE(IPRT,7100) 'LSDDED', (IH(LLSDED+I),I=K1-1,K2)
69: C 260 CONTINUE
70: C
71: C ..... REFLECTION STACK
72: C
73:     if ( JREFL.ne.0 ) then
74:         write(IPRT,7000) 'NBREF', (IH(LNBREF+I),I=0,NREFS)
75:         MM      = IH(LNBREF+NREFS)
76:         do 140 K1 = 1, MM, ML
77:             K2      = MIN(K1+ML-1,MM) - 1
78:             write(IPRT,7020) 'LSREF', (IH(LLSREF+I),I=K1-1,K2)
79:             write(IPRT,7020) 'IZREF', (IH(LIZREF+I),I=K1-1,K2)
80:             write(IPRT,7040) 'ISREF', (IH(LISREF+I),I=K1-1,K2)
81:         140 continue
82:     end if
83: C
84: C ..... LATTICE SEARCH STACK .....
85: C
86:     if ( JLATT.ne.0 ) then
87:         write(IPRT,7000) 'NXLT ', (IH(LNXLT+I),I=0,NLBZ)
88:         MM      = IH(LNXLT+NLBZ)
89:         do 150 K1 = 1, MM, ML
90:             K2      = MIN(K1+ML-1,MM) - 1
91:             write(IPRT,7020) 'LSLAT', (IH(LLSLAT+I),I=K1-1,K2)
92:             write(IPRT,7040) 'IZLAT', (IH(LIZLAT+I),I=K1-1,K2)
93:         150 continue
94:     end if
95: C
96: C
97:     7000 format(1X,'* ',A6,': ',20I4/(1X,9X,20I4))
98:     7020 format(1X,A5,20I5)
99:     7040 format(1X,A5,20I5/)
100:     return
101: end

```

src/mvp/dumpxx.f

```
1:      subroutine DUMPXX( N1,N2,IPRT, XXX,   YY,   ZZ,   AA,   BB,  
2:      &                CCC,   WW,   EE,   IZ,   IG )  
3: C=====   
4: C purpose: dump particle bank parameters   
5: C=====   
6:      real*8 XXX(*), YY(*), ZZ(*), AA(*), BB(*), CC(*)  
7:      real WW(*), EE(*)  
8:      integer IZ(*), IG(*)  
9: C   
10:     write(IPRT,7000) (I,XXX(I),YY(I),ZZ(I),  
11: &                AAA(I),BBB(I),CCC(I),WWW(I),  
12: &                EEE(I),IZZ(I),IGG(I),I=N1,N2)  
13: 7000 format(/1X,' # ', ' XXX ', ' YY ', ' ZZ ',  
14: &                ' AAA ', ' BBB ', ' CCC ', ' WWW ',  
15: &                ' EEE ', ' IZZ ', ' IGG '//  
16: &                (1X,I5,1X,1P,6E11.4,2E11.4,2I5))  
17:     return  
18:     end
```

src/mvp/editfl.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine EDITFL( JEIGN, JMICE, JMACE, LMIC, LMAC, NSMIC,
3:     & NSMAC, LEMIC, LEMAC, NEMIC, NEMAC, JRESP,
4:     & IETAL, KCRES, NIETAL,NETALY,NSTAL,
5:     & JPTDT, LMICI, LMACI, NSMICI, NSMACI, KCRESI,
6:     & JPHNU, NSMICPN, ! ph-nuc-3
7:     & JBEFF ) ! beff
8: C=====
9: C PURPOSE: SET EDIT FLAGS & REACTION POINTER OF SIGMA BANK & TALLY
10: C ARRAYS
11: C CALLED IN: INTRO2
12: C CALLS:
13: C=====
14:   integer JMICE(8), JMACE(8)
15:   integer LMAC(8), LMIC(8), LEMAC(16), LEMIC(16)
16:   integer LMACI(8), LMICI(8)
17:   integer IETAL(*), KCRES(NSTAL,4), KCRESI(NSTAL,4)
18: C
19: C ... local data ...
20: C
21:   integer INFO(10)
22:   character*6 TAG
23:   character PLAB*32
24: C
25: C JRTTR I4 EDIT-BY-TRACK-LENGTH YES
26: C JRTCL I4 EDIT-BY-COLLISION NO
27: C
28: C***** EDIT FLAGS *****
29: C K = AN 8-BIT INTEGER WHOSE DIGITS REPRESENT TREATMENTS FOR
30: C REACTIONS.
31: C
32: C 0 : DEFALUT ==> SEE SUBROUTINE INTRO.
33: C 1 : SAVE CROSS SECTION & RATE IN FILE (FT30F001) & PRINT THEM.
34: C 2 : SAVE CROSS SECTION & RATE IN FILE (FT30F001) BUT NOT PRINT THEM.
35: C 3 : SAVE CROSS SECTION & RATE IN FILE (FT30F001) & PRINT ONLY RATES.
36: C 3 : SAVE CROSS SECTION & RATE IN FILE (FT30F001) & PRINT ONLY XSEC.
37: C
38: C JMICE I4 EDIT-MICROSCOPIC-DATA(K) YES
39: C JMACE I4 EDIT-MACROSCOPIC-DATA(K) YES
40: C
41:   if ( JEIGN.ne.0 ) then
42:     LMIC(1) = 1
43:     LMIC(2) = 2
44:     LMIC(3) = 3
45:     LMIC(4) = 5
46:     LMIC(5) = 6
47:     LMIC(8) = 4
48:     KK = 6
49:     if ( JMICE(6).ne.0 .or. JMACE(6).ne.0 ) then
50:       KK = KK + 1
51:       LMIC(6) = KK
52:     end if
53:     if ( JMICE(7).ne.0 .or. JMACE(7).ne.0 ) then
54:       KK = KK + 1
55:       LMIC(7) = KK
56:     end if
57:     NSMIC = KK
58: C
59:   LMAC(1) = 1
60:   LMAC(2) = 2
61:   if( JBEFF.eq.0 ) then ! beff >>
62:     LMAC(8) = 3
63:     NSMAC = 3
64:   else
65:     LMAC(3) = 3

```

```

66:     LMAC(8) = 4
67:     NSMAC = 4
68:   end if ! beff <<
69: else
70:   LMIC(1) = 1
71:   LMIC(2) = 2
72:   LMIC(3) = 3
73:   LMIC(4) = 4
74:   LMIC(5) = 5
75:   KK = 5
76:   do 100 I = 6, 8
77:     if ( JMICE(I).ne.0 .or. JMACE(I).ne.0 ) then
78:       KK = KK + 1
79:       LMIC(I) = KK
80:     end if
81:   100 continue
82:   NSMIC = KK
83: C
84:   LMAC(1) = 1
85:   if ( JPHNU.ne.0 ) then ! ph-nuc-3 >>
86:     LMAC(2) = 2
87:     NSMAC = 2
88:   else
89:     NSMAC = 1
90:   end if ! ph-nuc-3 <<
91: end if
92: C ..... photonuclear ! ph-nuc-3 >>
93:   if ( JPHNU.ne.0 ) then
94:     NSMICPN = 3 ! for nonelastic, nu*fission, micro-response
95:   else
96:     NSMICPN = 0
97:   end if ! ph-nuc-3 <<
98: C
99:   NEMIC = 0
100:   NEMAC = 0
101:   do 110 I = 1, 8
102:     if ( JMACE(I).ne.0 ) then
103:       NEMAC = NEMAC + 1
104:       LEMAC(I) = NEMAC
105:       LEMAC(NEMAC+8) = I
106:       if ( JMICE(I).eq.0 ) LEMIC(I) = 1
107:     end if
108:     if ( JMICE(I).ne.0 .or. LEMIC(I).eq.1 ) then
109:       NEMIC = NEMIC + 1
110:       LEMIC(I) = NEMIC
111:       LEMIC(NEMIC+8) = I
112:     end if
113:   110 continue
114: C
115: C ... NSMIC, NSMAC, LMIC, LMAC, KCRES are modified for special tallies
116: C
117:   if ( JRESP.ne.0 .and. NSTAL.gt.0 ) then
118:     do 120 I = 1, NSTAL
119:       KCRES(I,4) = 1
120:     continue
121:   end if
122: C
123:   do 130 N = 1, NETALY
124:     TAG = ' '
125:     write(TAG,'(','.',i5)') N
126:     call CCOMP( TAG, LEN(TAG), TAG, IIF )
127:     LTAG = ICLEN(TAG)
128:     PLAB = 'TALLY'//TAG(:LTAG)
129:     call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
130:     call UNPKND( IETAL, 'INFO', 'I4', INFO, 10, NA, IRET )

```

src/mvp/editfl.f

```
131: C
132:     if ( INFO(7).ge.1001 .and. INFO(7).le.1002 ) then
133:         INUC = INFO(8)/1024
134:         IMIC = INFO(8) - INUC*1024
135:         if ( INFO(6).ne.1 ) then
136:             if ( LMIC(IMIC).eq.0 ) then
137:                 NSMIC = NSMIC + 1
138:                 LMIC(IMIC) = NSMIC
139:             end if
140:         else
141:             if ( LMICI(IMIC).eq.0 ) then
142:                 NSMICI = NSMICI + 1
143:                 LMICI(IMIC) = NSMICI
144:             end if
145:         end if
146:     else if ( INFO(7).eq.2000 ) then
147:         if ( IRESP.eq.0.and.INFO(6).ne.1 ) KCRES(INFO(8),4) = 1
148:         if ( INFO(6).eq.1 ) KCRESI(INFO(8),4) = 1
149:     else if ( INFO(7).ge.3000 .and. INFO(7).lt.4000 ) then
150:         if ( LMIC(INFO(8)).eq.0 ) then
151:             NSMIC = NSMIC + 1
152:             LMIC(INFO(8)) = NSMIC
153:         end if
154:         if ( LMAC(INFO(8)).eq.0 ) then
155:             NSMAC = NSMAC + 1
156:             LMAC(INFO(8)) = NSMAC
157:         end if
158:     end if
159: 130 continue
160:
161:     if ( JPTDT.ne.0 ) then
162:         NSMICI = NSMIC
163:         NSMACI = NSMAC
164:         do I = 1, 8
165:             LMICI(I) = LMIC(I)
166:             LMACI(I) = LMAC(I)
167:         end do
168:     end if
169: C
170:     return
171: end
172: C
173:
174:     subroutine STKCRS( KCRES, KCRESI, NSTAL )
175: C=====
176: C CALLED IN INTRO2
177: C=====
178:     integer KCRES(NSTAL,4), KCRESI(NSTAL,4)
179: C
180:     do 100 I = 1, NSTAL
181:         KCRESI(I,1) = KCRES(I,1)
182:         KCRESI(I,2) = KCRES(I,2)
183:         KCRESI(I,3) = KCRES(I,3)
184: 100 continue
185: C
186:     return
187: end
```

src/mvp/egychk.f

```

1:      subroutine EGYCHK( A, LIMIT,
2:      &                  JKPAR, ETOPX, EBOTX,
3:      &                  JNEUT, JPHOT, JEIGN, JDEBG, NGP1, NGP2,
4:      &                  NGROUP,ETOP,  ETOPL, EBOT,  EBOTL, ETHMAX,
5:      &                  ESABL, ETOPP, ETOPLP,EBOTP, EBOTLP,LENGYB,
6:      &                  LENGPB )
7: C=====
8: C purpose: check neutron/photon energy group boundary data and
9: C          modify them if necessary.
10: C=====
11: C
12:      real A(*)
13: CCCC include '../shared/INC/_ARRAY'
14:      include '../shared/INC/_IUNIT'
15: C
16:      include 'INC/_KPIDS'
17:      include 'INC/_KPSYMS'
18:      include 'INC/_NGPS'
19: C
20:      integer JDEBG(*)
21: C
22:      integer JKPAR(KPLIM)
23:      real*8  ETOPX(KPLIM)
24:      real*8  EBOTX(KPLIM)
25: C
26: C-----
27: C
28:
29: CCC  if ( JNEUT.ne.0 ) then
30:      if ( JKPAR(KPNEUT).ne.0 ) then
31: C
32:          if ( ETOP.gt.ETOPL ) then
33: c##<2007/03/14:PN3:
34: c##          write(IMG,'(/lx,a,a,e12.5/)')
35:          write(IMG,'(/lx,a,a,lp,e12.5/)')
36: c##>
37:      &          '!!! "ETOP(.N)" IS GREATER THAN UPPER ENERGY LIMIT',
38:      &          ' IN CROSS SECTION LIBRARY. IT IS SET TO ', ETOPL
39:          call CNTERR( 'WARNING' )
40:          ETOP = ETOPL
41:      end if
42:      if ( EBOT.lt.EBOTL ) then
43: c##<2007/03/14:PN3:
44: c##          write(IMG,'(/lx,a,a,e12.5/)')
45:          write(IMG,'(/lx,a,a,lp,e12.5/)')
46: c##>
47:      &          '!!! "EBOT(.N)" IS SMALLER THAN LOWER ENERGY LIMIT',
48:      &          ' IN CROSS SECTION LIBRARY. IT IS SET TO ', EBOTL
49:          call CNTERR( 'WARNING' )
50:          EBOT = EBOTL
51:      end if
52:      if ( ETHMAX.gt.0. ) then
53:          if ( EBOT.gt.EBOTL ) then
54: c##<2007/03/14:PN3:
55: c##          write(IMG,'(/lx,a,a/4x,a,e12.5/)')
56:          write(IMG,'(/lx,a,a/4x,a,lp,e12.5/)')
57: c##>
58:      &          '!!! THERMAL SCATTERING WILL OCCUR, BUT "EBOT"',
59:      &          ' IS GREATER THAN THE LOWER ENERGY LIMIT IN LIBRARY.',
60:      &          ' IT IS SET TO ', EBOTL
61:          call CNTERR( 'WARNING' )
62:          EBOT = EBOTL
63:      end if
64:      if ( ETHMAX.lt.EBOT ) then
65: c##<2007/03/14:PN3:

```

```

66: c##          write(IMG,'(/lx,a,a/4x,a,e12.5/)')
67:          write(IMG,'(/lx,a,a/4x,a,lp,e12.5/)')
68: c##>
69:      &          '!!! THERMAL SCATTERING WILL OCCUR, BUT "ETHMAX"',
70:      &          ' IS SMALLER THAN LOWER ENERGY LIMIT IN LIBRARY.',
71:      &          ' IT IS SET TO ', ESABL
72:          call CNTERR( 'WARNING' )
73:          ETHMAX = ESABL
74:      end if
75:      end if
76:      if ( JEIGN.ne.0.and.EBOT.gt.EBOTL ) then
77: c##<2007/03/14:PN3:
78: c##          write(IMG,'(/lx,a,a,e12.5/)')
79:          write(IMG,'(/lx,a,a,lp,e12.5/)')
80: c##>
81:      &          '!!! "EBOT" IS GREATER THAN LOWER ENERGY LIMIT',
82:      &          ' IN CROSS SECTION LIBRARY. IT IS SET TO ', EBOTL
83:          call CNTERR( 'WARNING' )
84:          EBOT = EBOTL
85:      end if
86: C
87:      ETOPX(KPNEUT) = ETOP
88:      EBOTX(KPNEUT) = EBOT
89: C
90: C      if ( LENGYB.ne.0 ) then
91: C          if ( A(LENGYB).lt.ETOP ) then
92: C              A(LENGYB) = ETOP
93: C              write(IMG,'(/lx,a,a,e12.5/)')
94: C              &          '!!! UPPERMOST ENERGY BIN BOUNDARY IS SMALLER ',
95: C              &          'THAN "ETOP", IT IS SET TO ', ETOP
96: C              call CNTERR( 'WARNING' )
97: C          end if
98: C          if ( A(LENGYB+NGP1).gt.EBOT ) then
99: C              A(LENGYB+NGP1) = EBOT
100: C              write(IMG,'(/lx,a,a,e12.5/)')
101: C              &          '!!! LOWEST ENERGY BIN BOUNDARY IS GREATER ',
102: C              &          'THAN "EBOT", IT IS SET TO ', EBOT
103: C              call CNTERR( 'WARNING' )
104: C          end if
105: C
106: C          call CKODR4( A(LENGYB), NGP1+1, 1, INVLD )
107: C          if ( INVLD.ne.0 ) then
108: C              write(IMG,*)
109: C              &          'XXX ORDER OF NEUTRON ENERGY BOUNDARY IS INCORRECT.'
110: C              call CNTERR( 'FATAL' )
111: C          end if
112: C      else
113: C          write(IMG,'(/lx,a,a)')
114: C          &          '!!! NO ENERGY BOUNDARY INPUT. ENERGY',
115: C          &          ' STRUCTURE HAVING CONSTANT LETHARGY WIDTH IS ASSUMED.'
116: C          call CNTERR( 'WARNING' )
117: C
118: C          call KEEP( 'ENGYB', LENGYB, NGP1+1, 'R4', LAST, JDEBG )
119: C
120: C          DU = LOG(ETOP/EBOT) /NGP1
121: C          call LTHRGY( A(LENGYB), DU, ETOP, NGP1+1 )
122: C      end if
123: C
124:      end if
125: C
126: CC  if ( JPHOT.ne.0 ) then
127:      if ( JKPAR(KPPHOT).ne.0 ) then
128: C
129:          if ( ETOPP.gt.ETOPLP ) then
130: c##<2007/03/14:PN3:

```

src/mvp/egychk.f

```

131: c##      write(IMG, '(/lx,a,a,e12.5/)')
132:      write(IMG, '(/lx,a,a,lp,e12.5/)')
133: c##>
134:      &          '!!! "ETOP.P" IS GREATER THAN UPPER ENERGY LIMIT',
135:      &          ' IN PHOTON CROSS SECTION LIBRARY. IT IS SET TO ',
136:      &          ETOPLP
137:      call CNTERR( 'WARNING' )
138:      ETOPP = ETOPLP
139:      end if
140:      if ( EBOTP.lt.EBOTLP ) then
141: c##<2007/03/14:PN3:
142: c##      write(IMG, '(/lx,a,a,e12.5/)')
143: c##      write(IMG, '(/lx,a,a,lp,e12.5/)')
144: c##>
145:      &          '!!! "EBOT.P" IS SMALLER THAN LOWER ENERGY LIMIT',
146:      &          ' IN PHOTON CROSS SECTION LIBRARY. IT IS SET TO ',
147:      &          EBOTLP
148:      call CNTERR( 'WARNING' )
149:      EBOTP = EBOTLP
150:      end if
151:      ETOPX(KPPHOT) = ETOPP
152:      EBOTX(KPPHOT) = EBOTP
153: C
154: C      if ( LENGPB.ne.0 ) then
155: C      if ( A(LENGPB).lt.ETOPP ) then
156: C      A(LENGPB) = ETOPP
157: C      write(IMG, '(/lx,a,a,e12.5/)')
158: C      &          '!!! UPPERMOST ENERGY BIN BOUNDARY (PHOTON) IS SMALLER ',
159: C      &          'THAN "ETOP.P", IT IS SET TO ', ETOPP
160: C      call CNTERR( 'WARNING' )
161: C      end if
162: C      if ( A(LENGPB+NGP2).gt.EBOTP ) then
163: C      A(LENGPB+NGP2) = EBOTP
164: C      write(IMG, '(/lx,a,a,e12.5/)')
165: C      &          '!!! LOWEST ENERGY BIN BOUNDARY (PHOTON) IS GREATER ',
166: C      &          'THAN "EBOT.P", IT IS SET TO ', EBOTP
167: C      call CNTERR( 'WARNING' )
168: C      end if
169: C
170: C      call CKODR4( A(LENGPB), NGP2+1, 1, INVLD )
171: C      if ( INVLD.ne.0 ) then
172: C      write(IMG,*)
173: C      & 'XXX ORDER OF NEUTRON/PHOTON ENERGY BIN BOUNDARY IS INCORRECT.'
174: C      call CNTERR( 'FATAL' )
175: C      end if
176: C      else
177: C      write(IMG, '(/lx,a,a)')
178: C      &          '!!! NO ENERGY BOUNDARY IS SPECIFIED (PHOTON). ENERGY',
179: C      &          ' STRUCTURE HAVING CONSTANT LEHARGY WIDTH IS ASSUMED.'
180: C      call CNTERR( 'WARNING' )
181: C
182: C      call KEEP( 'ENGPB', LENGPB, NGP2+1, 'R4', LAST, JDEBG )
183: C
184: C      DU = LOG(ETOPP/EBOTP) /NGP2
185: C      call LTHRGY( A(LENGPB), DU, ETOPP, NGP2+1 )
186: C      end if
187: C
188:      end if
189: C
190: C      .... check energy boundaries ...
191: C
192:      if ( LENGYB.eq.0 ) then
193:      call KEPV( A(1), 'ENGYB', LENGYB, KENGP(KPLIM+1)-1, 'R4',
194:      &          LAST, 0.0, JDEBG )
195:      end if

```

```

196: C
197:      do 100 IK=1,KPLIM
198:      if ( JKPAP(IK).ne.0 ) then
199:      L1 = LENGYB+KENGP(IK)-1
200:      L2 = L1+NGP(IK)
201:      NNZ = 0
202:      do 110 I=L1,L2
203:      if ( A(I).ne.0.0 ) then
204:      NNZ = 1
205: c##<2007/03/14:PN3:
206: c##      goto 120
207:      go to 120
208: c##>
209:      end if
210:      110      continue
211:      120      continue
212: C
213:      if ( NNZ.eq.0 ) then
214: C
215: C      ... set default bin
216: C
217:      DU = LOG(ETOPX(IK)/EBOTX(IK)) /NGP(IK)
218:      call LTHRGY( A(L1), DU, real(ETOPX(IK)), NGP(IK)+1 )
219: c##<2007/03/14:PN3:
220: c##      write(IMG,*)
221: c##      write(IMG, '(/lx,A,A,A,A)')
222: c##>
223:      &          '!!! No energy boundary is specified for <',
224:      &          KPSYM(1,IK)(:ICLEN2(KPSYM(1,IK))), '>',
225:      &          ' Energy structure having constant lethargy width is set.'
226:      call CNTERR( 'WARNING' )
227:      else
228:      if ( A(L1).lt.ETOPX(IK) ) then
229:      A(L1) = ETOPIX(IK)
230: c##<2007/03/14:PN3:
231: c##      write(IMG, '(/lx,a,a,a,e12.5/)')
232: c##      write(IMG, '(/lx,a,a,a,lp,e12.5/)')
233: c##>
234:      &          '!!! Uppermost energy bin boundary of <',
235:      &          KPSYM(1,IK)(:ICLEN2(KPSYM(1,IK))),
236:      &          '> is smaller than "ETOPX", it is set to ',
237:      &          ETOPIX(IK)
238:      call CNTERR( 'WARNING' )
239:      end if
240:      if ( A(L2).gt.EBOTX(IK) ) then
241:      A(L2) = EBOTX(IK)
242: c##<2007/03/14:PN3:
243: c##      write(IMG, '(/lx,a,a,a,e12.5/)')
244: c##      write(IMG, '(/lx,a,a,a,lp,e12.5/)')
245: c##>
246:      &          '!!! Lowest energy bin boundary of <',
247:      &          KPSYM(1,IK)(:ICLEN2(KPSYM(1,IK))),
248:      &          '> is larger than "EBOTX", it is set to ',
249:      &          EBOTX(IK)
250:      call CNTERR( 'WARNING' )
251:      end if
252:      call CKODR4( A(L1), NGP(IK)+1, 1, INVLD )
253:      if ( INVLD.ne.0 ) then
254: c##<2007/03/14:PN3:
255: c##      write(IMG,*)
256: c##      &          'XXX Order of <',
257: c##      write(IMG, '(/lx,A,A,A,A)')
258: c##      &          'XXX(egychk) Order of <',
259: c##>
260:      &          KPSYM(1,IK)(:ICLEN2(KPSYM(1,IK))),

```


src/mvp/egychk.f

```
261:      &                '> bin boundary is incorrect.'  
262:      call CNTERR( 'FATAL' )  
263:      end if  
264:      end if  
265:      end if  
266:      100 continue  
267:      return  
268:      end
```

SAFE

src/mvp/elgen.f

```

1:      subroutine ELGEN ( IOW, JTIME, JIMPT, JLATT, JHLAT, JTLT,
2:      F      IRAND, NBANK, NEST, EBOTE, NELC,
3:      B      XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE,
4:      B      TTT, XIM, KLSF, IZZ, IGG, LEVL, LZZ, LPOS, LCRS, IBREG, IBSPC,
5:      W      R,
6:      W      IMTTE, IMTTE0, IWK20, WK20, WK21, WK22,
7:      W      WK23, WK34,
8:      W      WK30, WK31, WK32, WK33, WK35,
9:      W      WK36,
10:     W      IWK31, IWK32,
11:     W      WK41, WK42, WK43 )
12: C=====
13: C PURPOSE : Make one or two electrons in a photon collision.
14: C CALLED IN: PHOTR
15: C
16: C=====
17: C
18: C IMTTE      I4  NUMBER OF PHOTON COLLISIONS WHICH ARE POSSIBLE TO
19: C             GENERATE ELECTRON.
20: C IMTTE0(4)  I4  NUMBER OF PHOTON COLLISIONS BY COLLISION TYPE
21: C             WHICH IS POSSIBLE TO GENERATE ELECTRON.
22: C             1  NUMBER OF PAIR PRODUCTIONS
23: C             2  NUMBER OF INCOHERENT (COMPTON) SCATTERING
24: C             3  NUMBER OF PHOTOELECTRIC EFFECT
25: C             4  NUMBER OF AUGER ELECTRONS
26: C*
27: C NELC      I4  NUMBER OF REALLY GENERATED ELECTRONS BY PHOTON
28: C             COLLISIONS.
29: C*
30: C IWK20(NBANK)  I4  BANK INDEX OF PARENT PHOTON GENERATING ELECTRON
31: C             (IF NEGATIVE, PARENT PHOTON DIED).
32: C WK20(NBANK)   R4  PHOTON WEIGHTS BEFORE PHOTON INCURED A COLLISION.
33: C WK21(NBANK)   R4  PHOTON ENERGIES BEFORE PHOTON INCURED A
34: C             COLLISION.
35: C WK22(NBANK)   R4  PHOTON ENERGIES AFTER PHOTON INCURED A
36: C             COLLISION, IF PHOTON SURVIVE.
37: C             OR PHOTO-ELECTRON ENERGY
38: C WK23(NBANK)   R4  AUGER ELECTRON ENERGIES AFTER PHOTON INCURED
39: C             A PHOTOELECTRIC COLLISION.
40: C WK34(NBANK)   R4  DIRECTION COSINES AFTER PHOTON INCURED A
41: C*
42: C*
43: C ETOPE      R4  UPPER ENERGY LIMIT SPECIFIED BY USER.
44: C EBOTE      R4  LOWER ENERGY LIMIT SPECIFIED BY USER.
45: C*
46: C R(NBANK*4)  R4  TEMPORARY AREA FOR RANDUM NUMBER.
47: C*
48: C DMOC2      R4  REST ENERGY OF ELECTRON (0.511E+6 EV).
49: C
50: C=====
51: C
52:      implicit real*8(A-H,O-Z)
53: C
54: C
55: C**** CROSS SECTION DATA
56: C
57:      real EBOTE
58: C
59: C**** STACK
60: C
61:      integer NELC
62: C
63: C**** PARTICLE BANK
64: C
65:      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)

```

```

66:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
67:      integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
68:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
69:      real EEE(NBANK), WWW(NBANK), XIM(NBANK)
70:      real*8 TTT(NBANK)
71:      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
72: C
73: C
74: C**** WORKING AREA
75: C
76: C      .... PROVIDED DATA ....
77: C
78:      integer IMTTE, IMTTE0(4), IWK20(NBANK)
79:      real WK20(NBANK), WK21(NBANK), WK22(NBANK), WK23(NBANK)
80: C
81: C      .... CREATING DATA ....
82: C
83:      integer IWK31(2*NBANK)
84:      real WK30(2*NBANK), WK31(2*NBANK), WK32(2*NBANK)
85:      real*8 WK41(2*NBANK), WK42(2*NBANK), WK43(2*NBANK)
86: C
87: C      .... TEMPORARY DATA ....
88: C
89:      integer IWK32(NBANK)
90:      real WK33(NBANK), WK34(2*NBANK), WK35(NBANK), WK36(2*NBANK),
91:      &      R(4*NBANK)
92: C
93:      parameter( DMOC2 = 0.511D+6, ONE = 0.999999D0 )
94: C
95:      parameter ( PI = 3.141592653589793238D0 )
96:      parameter ( PI2 = 2.0D0 * PI )
97:      parameter ( PI23 = 2.0D0 * PI / 3.0D0 )
98: C
99: C=====
100: C
101: C SPECIAL DATA ARRAY FOR EMACKER (GET FROM MCNP-4)
102: C (A) DETERMINATION OF ELECTRON EMISSION ANGLE BY PHOTOELECTRIC
103: C     EE : ENERGY TABLE
104: C     CE : FACTOR AS EQUAL PROBABILITY TO CALCULATE EMISSION ANGLE
105: C (B) DETERMINATION OF ELECTRON ENERGY BY PAIR-PRODUCTION
106: C     EP : ENERGY TABLE
107: C     DP : FACTOR AS EQUAL PROBABILITY TO CALCULATE ENERGY
108: C
109:      real EE(14), EP(9)
110:      real*8 CE(21,13), DP(21,9)
111: C
112:      data EE /1.00E+8, 5.00E+7, 2.00E+7, 1.00E+7, 5.00E+6, 2.00E+6,
113:      &      1.00E+6, 5.00E+5, 2.00E+5, 1.00E+5, 5.00E+4, 2.00E+4,
114:      &      1.00E+4, 0.0/
115: C
116:      data(CE(I,1),I=1,21) /5.0000000E-1, 1.8391734E+2, 3.7051644E+2,
117:      &      5.6409171E+2, 8.5433494E+2, 8.6549644E+2, 1.1960445E+3,
118:      &      1.8133120E+3, 3.3154915E+3, 1.1560527E+4, 7.7358920E+4,
119:      &      7.7358920E+4, 7.7358920E+4, 7.7358920E+4, 7.7358920E+4,
120:      &      7.7358920E+4, 7.7358920E+4, 7.7358920E+4, 7.7358920E+4,
121:      &      7.7358920E+4, 7.7358920E+4/
122:      data(CE(I,2),I=1,21) /5.0001000E-1, 1.5734214E+2, 3.1179562E+2,
123:      &      4.6600627E+2, 6.8708044E+2, 7.2630997E+2, 9.3864458E+2,
124:      &      1.2726402E+3, 1.8585851E+3, 3.1036588E+3, 7.2354569E+3,
125:      &      1.9539631E+4, 1.9539631E+4, 1.9539631E+4, 1.9539631E+4,
126:      &      1.9539631E+4, 1.9539631E+4, 1.9539631E+4, 1.9539631E+4,
127:      &      1.9539631E+4, 1.9539631E+4/
128:      data(CE(I,3),I=1,21) /5.0008000E-1, 7.3991679E+1, 1.4697244E+2,
129:      &      2.2059103E+2, 2.9650433E+2, 3.6916714E+2, 4.5806222E+2,
130:      &      5.6159689E+2, 5.9186199E+2, 6.8870573E+2, 8.0858436E+2,

```

src/mvp/elgen.f

```
131:      &      9.5808897E+2, 1.1453783E+3, 1.3799180E+3, 1.6705798E+3,
132:      &      2.0208947E+3, 2.4192528E+3, 2.8236151E+3, 3.1519086E+3,
133:      &      3.2220501E+3, 3.2220501E+3/
134:      data(CE(I,4),I=1,21) /5.0030000E-1, 2.5031580E+1, 4.8446990E+1,
135:      &      7.2297589E+1, 9.6665639E+1, 1.2168910E+2, 1.4755735E+2,
136:      &      1.7420693E+2, 2.0162045E+2, 2.2944535E+2, 2.6029701E+2,
137:      &      2.8916076E+2, 3.2332973E+2, 3.6023191E+2, 3.8754624E+2,
138:      &      4.2971674E+2, 4.8031309E+2, 5.4202665E+2, 6.1881714E+2,
139:      &      7.1677975E+2, 8.4578033E+2/
140:      data(CE(I,5),I=1,21) /5.0108000E-1, 8.5127000E+0, 1.5526810E+1,
141:      &      2.2556960E+1, 2.9680480E+1, 3.6933560E+1, 4.4343450E+1,
142:      &      5.1940279E+1, 5.9746320E+1, 6.7800979E+1, 7.6153979E+1,
143:      &      8.4843859E+1, 9.3893880E+1, 1.0344828E+2, 1.1366273E+2,
144:      &      1.2457488E+2, 1.3645768E+2, 1.4961923E+2, 1.6563791E+2,
145:      &      1.8436704E+2, 2.3213951E+2/
146:      data(CE(I,6),I=1,21) /5.0528999E-1, 2.9538700E+0, 4.7418300E+0,
147:      &      6.4446200E+0, 8.1125700E+0, 9.7666099E+0, 1.1419290E+1,
148:      &      1.3080480E+1, 1.4758300E+1, 1.6461250E+1, 1.8198290E+1,
149:      &      1.9979720E+1, 2.1817860E+1, 2.3728780E+1, 2.5733030E+1,
150:      &      2.7860760E+1, 3.0156810E+1, 3.2697300E+1, 3.5628950E+1,
151:      &      3.9331470E+1, 4.7791030E+1/
152:      data(CE(I,7),I=1,21) /5.1518000E-1, 1.7938600E+0, 2.5771400E+0,
153:      &      3.2745600E+0, 3.9273000E+0, 4.5526800E+0, 5.1606300E+0,
154:      &      5.7578800E+0, 6.3496200E+0, 6.9402800E+0, 7.5340199E+0,
155:      &      8.1350700E+0, 8.7480799E+0, 9.3785600E+0, 1.0033510E+1,
156:      &      1.0722500E+1, 1.1459730E+1, 1.2268640E+1, 1.3194470E+1,
157:      &      1.4353420E+1, 1.6972980E+1/
158:      data(CE(I,8),I=1,21) /5.3681000E-1, 1.2748100E+0, 1.6511800E+0,
159:      &      1.9657000E+0, 2.2487300E+0, 2.5125400E+0, 2.7637900E+0,
160:      &      3.0066900E+0, 3.2442600E+0, 3.4788700E+0, 3.7125900E+0,
161:      &      3.9473700E+0, 4.1852200E+0, 4.4284100E+0, 4.6797000E+0,
162:      &      4.9428000E+0, 5.2230800E+0, 5.5293500E+0, 5.8785000E+0,
163:      &      6.3138799E+0, 7.2922699E+0/
164:      data(CE(I,9),I=1,21) /5.8986000E-1, 9.4238999E-1, 1.1027300E+0,
165:      &      1.2324200E+0, 1.3469100E+0, 1.4522500E+0, 1.5516100E+0,
166:      &      1.6469600E+0, 1.7396700E+0, 1.8307700E+0, 1.9211600E+0,
167:      &      2.0116400E+0, 2.1030300E+0, 2.1962100E+0, 2.2923600E+0,
168:      &      2.3926000E+0, 2.4992800E+0, 2.6156400E+0, 2.7480400E+0,
169:      &      2.9128300E+0, 3.2821700E+0/
170:      data(CE(I,10),I=1,21) /6.4590000E-1, 8.5676000E-1, 9.5119999E-1,
171:      &      1.0272100E+0, 1.0941100E+0, 1.1555200E+0, 1.2133500E+0,
172:      &      1.2687700E+0, 1.3225900E+0, 1.3754400E+0, 1.4278200E+0,
173:      &      1.4802100E+0, 1.5330900E+0, 1.5869800E+0, 1.6425100E+0,
174:      &      1.7004800E+0, 1.7620900E+0, 1.8292500E+0, 1.9056400E+0,
175:      &      2.0006700E+0, 2.2135200E+0/
176:      data(CE(I,11),I=1,21) /7.0787000E-1, 8.3725999E-1, 8.9556000E-1,
177:      &      9.4266000E-1, 9.8425999E-1, 1.0225600E+0, 1.0587400E+0,
178:      &      1.0935100E+0, 1.1273700E+0, 1.1607100E+0, 1.1938500E+0,
179:      &      1.2270900E+0, 1.2607400E+0, 1.2951300E+0, 1.3306700E+0,
180:      &      1.3679000E+0, 1.4075900E+0, 1.4510100E+0, 1.5006000E+0,
181:      &      1.5626000E+0, 1.7026900E+0/
182:      data(CE(I,12),I=1,21) /7.8624000E-1, 8.6494000E-1, 9.0017000E-1,
183:      &      9.2852999E-1, 9.5349000E-1, 9.7640999E-1, 9.9800000E-1,
184:      &      1.0187000E+0, 1.0388100E+0, 1.0585700E+0, 1.0781600E+0,
185:      &      1.0977700E+0, 1.1175700E+0, 1.1377600E+0, 1.1585700E+0,
186:      &      1.1803100E+0, 1.2034400E+0, 1.2286600E+0, 1.2573800E+0,
187:      &      1.2931400E+0, 1.3733800E+0/
188:      data(CE(I,13),I=1,21) /8.3683000E-1, 8.9150999E-1, 9.1594999E-1,
189:      &      9.3561000E-1, 9.5291000E-1, 9.6878000E-1, 9.8372000E-1,
190:      &      9.9804000E-1, 1.0119400E+0, 1.0255900E+0, 1.0391300E+0,
191:      &      1.0526600E+0, 1.0663200E+0, 1.0802400E+0, 1.0945900E+0,
192:      &      1.1095700E+0, 1.1254900E+0, 1.1428500E+0, 1.1625900E+0,
193:      &      1.1871600E+0, 1.2422200E+0/
194: C
195:      data EP /200.0, 80.0, 40.0, 20.0, 10.0, 6.0, 4.0, 3.0, 2.0/
```

```
196: C
197:      data(DP(I,1),I=1,21) /.50000, .47443, .44894, .42358, .39836,
198:      &      .37338, .34863, .32410, .29982, .27581, .25210, .22872,
199:      &      .20568, .18287, .16023, .13762, .11495, .09196, .06816,
200:      &      .04245, .00000/
201:      data(DP(I,2),I=1,21) /.50000, .47577, .45159, .42753, .40359,
202:      &      .37982, .35621, .33274, .30944, .28629, .26323, .24027,
203:      &      .21735, .19441, .17135, .14807, .12443, .10011, .07446,
204:      &      .04614, .00000/
205:      data(DP(I,3),I=1,21) /.50000, .47705, .45413, .43123, .40838,
206:      &      .38560, .36288, .34020, .31753, .29485, .27218, .24948,
207:      &      .22670, .20379, .18069, .15727, .13336, .10857, .08217,
208:      &      .05237, .00000/
209:      data(DP(I,4),I=1,21) /.50000, .47861, .45720, .43578, .41431,
210:      &      .39280, .37125, .34962, .32793, .30614, .28417, .26204,
211:      &      .23967, .21699, .19381, .16996, .14517, .11897, .09064,
212:      &      .05895, .00000/
213:      data(DP(I,5),I=1,21) /.50000, .47973, .45943, .43907, .41864,
214:      &      .39813, .37751, .35676, .33583, .31465, .29318, .27137,
215:      &      .24920, .22657, .20330, .17918, .15390, .12696, .09752,
216:      &      .06325, .00000/
217:      data(DP(I,6),I=1,21) /.50000, .47973, .45946, .43919, .41892,
218:      &      .39864, .37830, .35772, .33692, .31587, .29454, .27275,
219:      &      .25049, .22764, .20390, .17930, .15363, .12622, .09608,
220:      &      .06173, .00000/
221:      data(DP(I,7),I=1,21) /.50000, .47962, .45925, .43887, .41850,
222:      &      .39812, .37777, .35718, .33635, .31545, .29451, .27329,
223:      &      .25155, .22932, .20625, .18221, .15717, .13011, .10020,
224:      &      .06496, .00000/
225:      data(DP(I,8),I=1,21) /.50000, .47915, .45830, .43744, .41659,
226:      &      .39574, .37489, .35403, .33318, .31248, .29120, .26904,
227:      &      .24600, .22301, .19878, .17326, .14770, .12050, .09134,
228:      &      .05931, .00000/
229:      data(DP(I,9),I=1,21) /.50000, .47500, .45000, .42500, .40000,
230:      &      .37500, .35000, .32500, .30000, .27500, .25000, .22500,
231:      &      .20000, .17500, .15000, .12500, .10000, .07500, .05000,
232:      &      .02500, .00000/
233: C
234: C-----
235: C
236:      NPHE      = 0
237:      NAUG      = 0
238:      NPPR      = 0
239:      NINC      = 0
240: C
241:      LTTT      = 0
242: C
243: C
244: C-----
245: C MAKE A PHOTOELECTRON.
246: C-----
247: C
248: C      IWK32      IWK31 : INDEX VECTOR ON EE TABLE FOR COLLIDING
249: C                      PHOTON ENERGY.
250: C      IWK31      : BANK POINTER OF PARENT PHOTON.
251: C      WK30      : EMITTED ELECTRON WEIGHT.
252: C      WK31      : EMITTED ELECTRON ENERGY.
253: C      WK32      : DIRECTION COSINE OF EMITTED ELECTRON.
254: C
255:      if ( IMTTE0(3).gt.0 ) then
256: C
257:      NTTT      = IMTTE0(1) + IMTTE0(2)
258: C
259:      *VOCL LOOP,NOVREC
260:      do 100 I = 1, IMTTE0(3)
```

src/mvp/elgen.f

```

261:      if ( WK22(NTTT+I).gt.EBOTE ) then
262:        NPHE = NPHE + 1
263:        IWK31(NPHE) = IWK20(NTTT+I)
264:        WK30(NPHE) = WK20(NTTT+I)
265:        WK31(NPHE) = WK22(NTTT+I)
266:        WK21(NTTT+NPHE) = WK21(NTTT+I)
267:      end if
268: 100 continue
269: C
270:   N = 14
271:   call BSVDEC( EE, N, WK31, IWK32, NPHE )
272: C
273:   call RANU2( IRAND, R, NPHE, ICON )
274: C
275:   do 110 I = 1, NPHE
276: C
277:     if ( IWK32(I).eq.13 ) then
278:       CCCCCC AL = ACOS(1.0-2.0*R(I)) /3.0
279:       AL = ACOS(1.0D0-2.0D0*R(I)) /3.0D0
280:       AM = 2.0D0*COS(PI23-AL)
281:     else
282:       EK = EE(IWK32(I)) - WK31(I)
283:       E1 = WK31(I) - EE(IWK32(I)+1)
284:       RR = R(I)*20.0D0 + 1.0D0
285: C/#IF ROUNDOFF(NEAREST)
286: *      RR = MIN( RR , 20.0D0)
287: C/#ENDIF
288:       L = RR
289:       P = RR - L
290:       AM1 = (1.0-P)*E1*CE(L,IWK32(I)) + P*E1*CE(L+1,IWK32(I))
291:       &      + (1.0-P)*EK*CE(L,IWK32(I)+1)
292:       &      + P*EK*CE(L+1,IWK32(I)+1)
293:       EW = WK21(NTTT+I)
294: csasaki AM2 = (EW + DM0C2) / SQRT( EW * (EW + DM0C2))
295:       AM2 = (EW+DM0C2) /SQRT(EW*(EW+2*DM0C2))
296:       AM3 = (1.0-(EE(IWK32(I))-EE(IWK32(I)+1))/AM1)*AM2
297:       AM = MIN(MAX(AM3,-1.0D0),1.0D0)
298:     end if
299: C
300:       WK32(I) = AM
301: 110 continue
302: C
303:       LTTT = LTTT + NPHE
304: C
305: C-----
306: C MAKE AN AUGER ELECTRON.
307: C-----
308: C
309: C       WK31 : EMITTED ELECTRON ENERGY.
310: C       WK32 : DIRECTION COSINE OF EMITTED ELECTRON.
311: C
312:   if ( IMTTE0(4).gt.0 ) then
313: C
314:     call RANU2( IRAND, R, IMTTE0(4), ICON )
315: C
316:     do 120 I = 1, IMTTE0(4)
317:       if ( WK23(NTTT+I).gt.EBOTE ) then
318:         LTTT = LTTT + 1
319:         IWK31(LTTT) = IWK20(NTTT+I)
320:         WK30(LTTT) = WK20(NTTT+I)
321:         WK31(LTTT) = WK23(NTTT+I)
322:         WK32(LTTT) = R(I)*2.0D0 - 1.0D0
323: CCCC      WK32(LTTT) = R(I)*2.0 - 1.0
324:       end if
325: 120 continue

```

```

326:       NAUG = LTTT - NPHE
327:     end if
328:   end if
329: C
330: C
331: C-----
332: C MAKE AN ELECTRON-POSITRON PAIR.
333: C-----
334: C
335: C       LTTT : STARTING LOCATION TO STORE ENERGIES AND
336: C       DIRECTION COSINES.
337: C       NTTT : NUMBER OF PHOTON COLLISIONS PROCESSED
338: C       PREVIOUS TO CURRENT REACTION.
339: C
340:   if ( IMTTE0(1).le.0 ) go to 190
341:   NPPR0 = IMTTE0(1)
342:   NTTT = 0
343: C
344: C       WK33 : COLLIDING PHOTON ENERGY IN UNIT OF ELECTRON
345: C       MASS, BUT MAXIMUM ENERGY IS 200 MEV.
346: C       IWK32 : INDEX VECTOR ON EP TABLE FOR COLLIDING
347: C       PHOTON ENERGY.
348: C
349:   DM0C22 = DM0C2*2.0
350: C
351:   do 140 I = 1, NPPR0
352:     ALP = WK21(I) /DM0C2
353:     WK33(I) = MIN(ALP,DBLE(EP(1)))
354: 140 continue
355: C
356:   N = 9
357:   call BSVDEC( EP, N, WK33, IWK32, NPPR0 )
358: C
359: C       B : PHOTON KINETIC ENERGY IN CM-SYSTEM.
360: C       WK32 : DIRECTION COSINE OF EMITTED PAIRS(TEMPORARY)
361: C       WK31 : EMITTED PAIR ENERGY BY PAIR-PRODUCTION.
362: C
363:   call RANU2( IRAND, R, NPPR0*4, ICON )
364:   NPPR2 = NPPR0*2
365:   NPPR3 = NPPR0*3
366: C
367: C
368:   do 150 I = 1, NPPR0
369:     J = LTTT + I
370:     RP1 = EP(IWK32(I)) - WK33(I)
371:     RP2 = WK33(I) - EP(IWK32(I)+1)
372: C
373:     RRRR = R(I)*20.0D0 + 1.0D0
374: C/#IF ROUNDOFF(NEAREST)
375: *     RRRR = MIN(RRRR,20.0D0)
376: C/#ENDIF
377:     LSR = RRRR
378:     RP3 = RRRR - LSR
379:     EM = WK21(I) - DM0C22
380: C .... ELECTRON ENERGY AND DIRECTION COSINE
381:     WK31(J) = EM*((1.0-RP3)*RP2*DP(LSR,IWK32(I))
382:     &      +RP3*RP2*DP(LSR+1,IWK32(I))+(1.0-RP3)*RP1*
383:     &      DP(LSR,IWK32(I)+1)+RP3*RP1*DP(LSR+1,IWK32(I)+1)) /
384:     &      (EP(IWK32(I))-EP(IWK32(I)+1))
385:     if ( R(I+NPPR0).gt.0.5 ) WK31(J) = EM - WK31(J)
386: C
387:     B = SQRT(WK21(I)*(WK21(I)+DM0C22)) / (WK21(I)+DM0C2)
388: CCCCC    RRR3 = R(I+NPPR2)*2.0
389:     RRR3 = R(I+NPPR2)*2.0D0
390:     AM = (RRR3-1.0-B) / (RRR3*B-1.0-B)

```

src/mvp/elgen.f

```

391:          WK32(J) = MIN(MAX(-ONE,AM),ONE)
392: C ..... POSITRON ENERGY AND DIRECTION COSINE
393:          WK31(NPPR0+J) = EM - WK31(J)
394: CCCCC   RRR4 = R(I+NPPR3)*2.0
395:          RRR4 = R(I+NPPR3)*2.0D0
396:          AM = (RRR4-1.0-B)/(RRR4*B-1.0-B)
397:          WK32(NPPR0+J) = MIN(MAX(-ONE,AM),ONE)
398:   150 continue
399: C
400:       N = LTTT
401: *VOCL LOOP,NOVREC
402:   do 160 I = 1, NPPR0
403:       LTTTI = LTTT + I
404:       if ( WK31(LTTTI).gt.EBOTE .or. WK31(LTTTI+NPPR0).gt.EBOTE )
405:       & then
406:           N = N + 1
407:           IWK31(N) = IWK20(I)
408:           WK30(N) = WK20(I)
409:           WK31(N) = WK31(LTTTI)
410:           IWK32(N-LTTT) = LTTTI
411:       end if
412:   160 continue
413: C
414:       NPPR = N - LTTT
415: C
416: *VOCL LOOP,NOVREC
417:   do 170 I = 1, NPPR
418:       J = IWK32(I) + NPPR0
419:       N1 = LTTT + I
420:       N2 = NPPR + N1
421:       IWK31(N2) = IWK31(N1)
422:       WK30(N2) = WK30(N1)
423:       WK31(N2) = WK31(J)
424:       if ( WK31(N1).le.EBOTE ) then
425:           WK31(N1) = EBOTE
426:           WK30(N1) = 0.0
427:       end if
428:       if ( WK31(N2).le.EBOTE ) then
429:           WK31(N2) = EBOTE
430:           WK30(N2) = 0.0
431:       end if
432:   170 continue
433: C
434: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
435: C
436: C           WK41 : DIRECTION ON X-AXIS
437: C           WK42 : DIRECTION ON Y-AXIS
438: C           WK43 : DIRECTION ON Z-AXIS
439: C
440:       call RANU2( IRAND, R, NPPR*2, ICON )
441: C
442: C
443: C
444: *VOCL LOOP,NOVREC
445:   do 180 I = 1, NPPR
446:       IP = ABS(IWK31(LTTT+I))
447:       J = IWK32(I)
448: C
449: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
450: C
451:       DA0 = AAA(IP)
452:       DB0 = BBB(IP)
453:       DC0 = CCC(IP)
454:       ETA = PI2*R(I)
455:       SINETA = SIN(ETA)

```

```

456:       COSETA = COS(ETA)
457:       STHETA = 1.0 - DA0*DA0
458: *VOCL STMT,IF(100)
459:       if ( STHETA.gt.0 ) then
460:           STHETA = SQRT(STHETA)
461:           COSPHI = DB0/STHETA
462:           SINPHI = DC0/STHETA
463:       else
464:           STHETA = 0.0
465:           COSPHI = 1.0
466:           SINPHI = 0.0
467:       end if
468:       XMU = WK32(J)
469: c950717-- SINPSI = SQRT( 1.0 - XMU * XMU )
470:       SINPSI = SQRT(ABS(1.0-XMU*XMU))
471:       DB1 = DB0*XMU + (DA0*COSPHI*COSETA-SINPHI*SINETA)*SINPSI
472:       DC1 = DC0*XMU + (DA0*SINPHI*COSETA+COSPHI*SINETA)*SINPSI
473:       DA1 = DA0*XMU - COSETA*SINPSI*STHETA
474:       S = 1.0/SQRT(DA1*DA1+DB1*DB1+DC1*DC1)
475:       WK41(I) = DA1*S
476:       WK42(I) = DB1*S
477:       WK43(I) = DC1*S
478: C
479: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF POSITRON .....
480: C
481:       XMUP = WK32(NPPR0+J)
482: c950717 B = SQRT( (1.0 - XMUP * XMUP) /
483: c & (1.0 - XMU * XMU) )
484:       B = SQRT(ABS((1.0D0-XMUP*XMUP)/(1.0D0-XMU*XMU)))
485: C
486:       A = XMUP - B*XMU
487:       DA1 = A*DA0 + B*WK41(I)
488:       DB1 = A*DB0 + B*WK42(I)
489:       DC1 = A*DC0 + B*WK43(I)
490:       S = 1.0/SQRT(DA1*DA1+DB1*DB1+DC1*DC1)
491:       WK41(NPPR+I) = DA1*S
492:       WK42(NPPR+I) = DB1*S
493:       WK43(NPPR+I) = DC1*S
494:   180 continue
495: C
496:       LTTT = LTTT + NPPR*2
497: C
498:   190 continue
499: C
500: C-----
501: C MAKE A COMPTON RECOIL ELECTRON BY INCOHERENT SCATTERING.
502: C-----
503: C
504: C           WK31 : EMITTED ELECTRON ENERGY.
505: C           WK32 : DIRECTION COSINE OF EMITTED ELECTRON
506: C           WK41 : COEFFICIENT OF DIRECTION FOR COMPTON.
507: C           WK42 : COEFFICIENT OF DIRECTION FOR COMPTON.
508: C           WK43 : DIRECTION COSINE OF EMITTED PHOTON.
509: C           COPIED FROM WK34
510: C
511:       if ( IMTTE0(2).le.0 ) go to 210
512:       NTTT = IMTTE0(1)
513:       N = NPPR*2
514: C
515: *VOCL LOOP,NOVREC
516:   do 200 I = 1, IMTTE0(2)
517:       EINC = WK21(NTTT+I) - WK22(NTTT+I)
518:       if ( EINC.gt.EBOTE ) then
519:           LTTT = LTTT + 1
520:           N = N + 1

```

src/mvp/elgen.f

```

521:      XMU      = WK34(NTTT+I)
522:      BT       = WK21(NTTT+I) /WK22(NTTT+I)
523:      AM       = (BT-XMU) /SQRT(ABS(BT-2.0*XMU)+1.0))
524: C950717
525: c   &          SQRT( BT* (BT - 2.0 * XMU) + 1.0 )
526:      WK32(LTTT) = MIN(AM,1.0D0)
527: C
528:      if ( XMU.gt.-1.0 ) then
529: c950717      WK42(N) = - SQRT( (1.0 - WK32(LTTT) * WK32(LTTT)) /
530: c   &          (1.0 - XMU * XMU) )
531:
532:      WK42(N) = - SQRT(
533: c   &          ABS((1.0D0-BELE(WK32(LTTT))*2)/(1.0D0-XMU*XMU)))
534: CCCC &          ABS((1.0-WK32(LTTT)*WK32(LTTT))/(1.0-XMU*XMU)))
535:      else
536:      WK42(N) = 0.0
537:      end if
538:
539:      WK41(N) = WK32(LTTT) - WK42(N)*XMU
540:      WK43(N) = XMU
541:      IWK31(LTTT) = IWK20(NTTT+I)
542:      WK30(LTTT) = WK20(NTTT+I)
543:      WK31(LTTT) = EINC
544:      end if
545: 200 continue
546: C
547:      NINC      = N - NPPR*2
548: C
549: C
550: 210 continue
551: C      NELC      : TOTAL NUMBER OF FINALLY GENERATED ELECTRON.
552: C      IMTTE0(1): PHOTOLECTRIC + AUGER ELECTRONS
553: C      IMTTE0(2):      + PAIR PRODUCTION ELCTRONS
554: C      IMTTE0(3):      + PAIR PRODUCTION POSITRONS
555: C      IMTTE0(4):      + INCOHERENT SCATTERED ELECTRONS
556: C
557:      IMTTE0(1) = NPHE + NAUG
558:      IMTTE0(2) = IMTTE0(1) + NPPR
559:      IMTTE0(3) = IMTTE0(2) + NPPR
560:      IMTTE0(4) = IMTTE0(3) + NINC
561: C
562:      NELC      = IMTTE0(4)
563: C
564:      return
565:      end
566: C
567: C=====
568:      subroutine ELGEN2 ( IOW, ITYP, JSTK , IRAND,
569:      S      ISEL , NELC , NBANK,
570:      B      A0 , B0 , C0 , AAE , BBE , CCE ,
571:      B      EIN , EELE , WE ,
572:      B      COSL , EOUT , AA , BB , CC ,
573:      W      R , COSLE, IWK1 , WK1 )
574: C=====
575: C PURPOSE : Make one electron in a photon collision for next event
576: C      estimator
577: C      no energy cut off
578: C CALLED IN: NXPHR3, NXPHR4
579: C Update:
580: C
581: C=====
582: C
583: C ITYP      i electron generating reaction
584: C      1 : photo electron
585: C      2 : auger electron

```

```

586: C      3 : pair creation
587: C      4 : Compton recoil electron
588: C      JSTK      i flag to indicate usage of stack tor bank data access
589: C      ISEL      i bank pointer
590: C      NELC      i number of photons generating electrons
591: C      (A0,B0,C0) i directional cosine of parent photon
592: C      (AAE,BBE,CCE) o directional cosine of electron
593: C      EIN      i photon energy generating electron
594: C      EELE      io generated electron energy
595: C
596: C      COSL      i scattering angle of photon by incoherent scattering
597: C      ( not bank, but sequential data )
598: C      EOUT      i scattered photon energy by incoherent scattering
599: C      ( not bank, but sequential data )
600: C      (AA,BB,CC) i directional cosine of scattered photon
601: C      ( not bank, but sequential data )
602: C
603: C=====
604: C
605: C      implicit real*8(A-H,O-Z)
606: C
607: C**** STACK
608: C
609: C      integer ISEL(NBANK)
610: C
611: C**** PARTICLE BANK
612: C
613: C      real*8 A0(NBANK), B0(NBANK), C0(NBANK)
614: C      real*8 AAE(NBANK), BBE(NBANK), CCE(NBANK)
615: C      real EIN(NBANK), WE(NBANK), EELE(NBANK)
616: C
617: C      real COSL(NBANK), EOUT(NBANK)
618: C      real*8 AA(NBANK), BB(NBANK), CC(NBANK)
619: C
620: C**** WORKING AREA
621: C
622: C      integer IWK1(NBANK)
623: C      real COSLE(NBANK), WK1(NBANK), R(NBANK*3)
624: C
625: C
626: C      parameter( DM0C2      = 0.511D+6, ONE = 0.999999D0 )
627: C
628: C      parameter ( PI      = 3.141592653589793238D0 )
629: C      parameter ( PI2     = 2.0D0 * PI )
630: C      parameter ( PI23    = 2.0D0 * PI / 3.0D0 )
631: C
632: C=====
633: C
634: C      real EE(14), EP(9)
635: C      real*8 CE(21,13), DP(21,9)
636: C
637: C      data EE /1.00E+8, 5.00E+7, 2.00E+7, 1.00E+7, 5.00E+6, 2.00E+6,
638: c   &      1.00E+6, 5.00E+5, 2.00E+5, 1.00E+5, 5.00E+4, 2.00E+4,
639: c   &      1.00E+4, 0.0/
640: C
641: C      data(CE(I,1),I=1,21) /5.0000000E-1, 1.8391734E+2, 3.7051644E+2,
642: c   &      5.6409171E+2, 8.5433494E+2, 8.6549644E+2, 1.1960445E+3,
643: c   &      1.8133120E+3, 3.3154915E+3, 1.1560527E+4, 7.7358920E+4,
644: c   &      7.7358920E+4, 7.7358920E+4, 7.7358920E+4, 7.7358920E+4,
645: c   &      7.7358920E+4, 7.7358920E+4, 7.7358920E+4, 7.7358920E+4,
646: c   &      7.7358920E+4, 7.7358920E+4/
647: C      data(CE(I,2),I=1,21) /5.0001000E-1, 1.5734214E+2, 3.1179562E+2,
648: c   &      4.6600627E+2, 6.8708044E+2, 7.2630997E+2, 9.3864458E+2,
649: c   &      1.2726402E+3, 1.8585851E+3, 3.1036588E+3, 7.2354569E+3,
650: c   &      1.9539631E+4, 1.9539631E+4, 1.9539631E+4, 1.9539631E+4,

```

src/mvp/elgen.f

```

651: &      1.9539631E+4, 1.9539631E+4, 1.9539631E+4, 1.9539631E+4,
652: &      1.9539631E+4, 1.9539631E+4/
653: data(CE(I,3),I=1,21) /5.0008000E-1, 7.3991679E+1, 1.4697244E+2,
654: &      2.2059103E+2, 2.9650433E+2, 3.6916714E+2, 4.5806222E+2,
655: &      5.6159689E+2, 5.9186199E+2, 6.8870573E+2, 8.0858436E+2,
656: &      9.5808897E+2, 1.1453783E+3, 1.3799180E+3, 1.6705798E+3,
657: &      2.0208947E+3, 2.4192528E+3, 2.8236151E+3, 3.1519086E+3,
658: &      3.2220501E+3, 3.2220501E+3/
659: data(CE(I,4),I=1,21) /5.0030000E-1, 2.5031580E+1, 4.8446990E+1,
660: &      7.2297589E+1, 9.6665639E+1, 1.2168910E+2, 1.4755735E+2,
661: &      1.7420693E+2, 2.0162045E+2, 2.2944535E+2, 2.6029701E+2,
662: &      2.8916076E+2, 3.2332973E+2, 3.6023191E+2, 3.8754624E+2,
663: &      4.2971674E+2, 4.8031309E+2, 5.4202665E+2, 6.1881714E+2,
664: &      7.1677975E+2, 8.4578033E+2/
665: data(CE(I,5),I=1,21) /5.0108000E-1, 8.5127000E+0, 1.5526810E+1,
666: &      2.2556960E+1, 2.9680480E+1, 3.6933560E+1, 4.4343450E+1,
667: &      5.1940279E+1, 5.9746320E+1, 6.7800979E+1, 7.6153979E+1,
668: &      8.4843859E+1, 9.3893880E+1, 1.0344828E+2, 1.1366273E+2,
669: &      1.2457488E+2, 1.3645768E+2, 1.4961923E+2, 1.6563791E+2,
670: &      1.8436704E+2, 2.3213951E+2/
671: data(CE(I,6),I=1,21) /5.0528999E-1, 2.9538700E+0, 4.7418300E+0,
672: &      6.4446200E+0, 8.1125700E+0, 9.7666099E+0, 1.1419390E+1,
673: &      1.3080480E+1, 1.4758300E+1, 1.6461250E+1, 1.8198290E+1,
674: &      1.9979720E+1, 2.1817860E+1, 2.3728780E+1, 2.5733030E+1,
675: &      2.7860760E+1, 3.0156810E+1, 3.2697300E+1, 3.5626950E+1,
676: &      3.9331470E+1, 4.7791030E+1/
677: data(CE(I,7),I=1,21) /5.1518000E-1, 1.7938600E+0, 2.5771400E+0,
678: &      3.2745600E+0, 3.9273000E+0, 4.5526800E+0, 5.1686300E+0,
679: &      5.7578800E+0, 6.3496200E+0, 6.9402800E+0, 7.5340199E+0,
680: &      8.1350700E+0, 8.7480799E+0, 9.3785600E+0, 1.0033510E+1,
681: &      1.0722500E+1, 1.1459730E+1, 1.2268640E+1, 1.3194470E+1,
682: &      1.4353420E+1, 1.6972980E+1/
683: data(CE(I,8),I=1,21) /5.3681000E-1, 1.2748100E+0, 1.6511800E+0,
684: &      1.9657000E+0, 2.2487300E+0, 2.5125400E+0, 2.7637900E+0,
685: &      3.0066900E+0, 3.2442600E+0, 3.4788700E+0, 3.7125900E+0,
686: &      3.9473700E+0, 4.1852200E+0, 4.4284100E+0, 4.6797000E+0,
687: &      4.9428000E+0, 5.2230800E+0, 5.5293500E+0, 5.8785000E+0,
688: &      6.3138799E+0, 7.2922699E+0/
689: data(CE(I,9),I=1,21) /5.8986000E-1, 9.4238999E-1, 1.1027300E+0,
690: &      1.2324200E+0, 1.3469100E+0, 1.4522500E+0, 1.5516100E+0,
691: &      1.6469600E+0, 1.7396700E+0, 1.8307700E+0, 1.9211600E+0,
692: &      2.0116400E+0, 2.1030300E+0, 2.1962100E+0, 2.2922600E+0,
693: &      2.3926000E+0, 2.4992800E+0, 2.6156400E+0, 2.7480400E+0,
694: &      2.9128300E+0, 3.2821700E+0/
695: data(CE(I,10),I=1,21) /6.4590000E-1, 8.5676000E-1, 9.5119399E-1,
696: &      1.0272100E+0, 1.0941100E+0, 1.1555200E+0, 1.2133500E+0,
697: &      1.2687700E+0, 1.3225900E+0, 1.3754400E+0, 1.4278200E+0,
698: &      1.4802100E+0, 1.5330900E+0, 1.5869800E+0, 1.6425100E+0,
699: &      1.7004800E+0, 1.7620900E+0, 1.8292500E+0, 1.9056400E+0,
700: &      2.0006700E+0, 2.2135200E+0/
701: data(CE(I,11),I=1,21) /7.0787000E-1, 8.3725999E-1, 8.9556000E-1,
702: &      9.4266000E-1, 9.8425999E-1, 1.0225600E+0, 1.0587400E+0,
703: &      1.0935100E+0, 1.1273700E+0, 1.1607100E+0, 1.1938500E+0,
704: &      1.2270900E+0, 1.2607400E+0, 1.2951300E+0, 1.3306700E+0,
705: &      1.3679000E+0, 1.4075900E+0, 1.4510100E+0, 1.5006000E+0,
706: &      1.5626000E+0, 1.7026900E+0/
707: data(CE(I,12),I=1,21) /7.8624000E-1, 8.6494000E-1, 9.0017000E-1,
708: &      9.2852999E-1, 9.5349000E-1, 9.7640999E-1, 9.9800000E-1,
709: &      1.0187000E+0, 1.0388100E+0, 1.0585700E+0, 1.0781600E+0,
710: &      1.0977700E+0, 1.1175700E+0, 1.1377600E+0, 1.1585700E+0,
711: &      1.1803100E+0, 1.2034400E+0, 1.2286600E+0, 1.2573800E+0,
712: &      1.2931400E+0, 1.3733800E+0/
713: data(CE(I,13),I=1,21) /8.3683000E-1, 8.9150999E-1, 9.1594999E-1,
714: &      9.3561000E-1, 9.5291000E-1, 9.6878000E-1, 9.8372000E-1,
715: &      9.9804000E-1, 1.0119400E+0, 1.0255900E+0, 1.0391300E+0,

```

```

716: &      1.0526600E+0, 1.0663200E+0, 1.0802400E+0, 1.0945900E+0,
717: &      1.1095700E+0, 1.1254900E+0, 1.1428500E+0, 1.1625900E+0,
718: &      1.1871600E+0, 1.2422200E+0/
719: C
720: data EP /200.0, 80.0, 40.0, 20.0, 10.0, 6.0, 4.0, 3.0, 2.0/
721: C
722: data(DP(I,1),I=1,21) /.50000, .47443, .44894, .42358, .39836,
723: &      .37338, .34863, .32410, .29982, .27581, .25210, .22872,
724: &      .20568, .18287, .16023, .13762, .11495, .09196, .06816,
725: &      .04245, .00000/
726: data(DP(I,2),I=1,21) /.50000, .47577, .45159, .42753, .40359,
727: &      .37982, .35621, .33274, .30944, .28629, .26323, .24027,
728: &      .21735, .19441, .17135, .14807, .12443, .10011, .07446,
729: &      .04614, .00000/
730: data(DP(I,3),I=1,21) /.50000, .47705, .45413, .43123, .40838,
731: &      .38560, .36288, .34020, .31753, .29485, .27218, .24948,
732: &      .22670, .20379, .18069, .15727, .13336, .10857, .08217,
733: &      .05237, .00000/
734: data(DP(I,4),I=1,21) /.50000, .47861, .45720, .43578, .41431,
735: &      .39280, .37125, .34962, .32793, .30614, .28417, .26204,
736: &      .23967, .21699, .19381, .16996, .14517, .11897, .09064,
737: &      .05895, .00000/
738: data(DP(I,5),I=1,21) /.50000, .47973, .45943, .43907, .41864,
739: &      .39813, .37751, .35676, .33583, .31465, .29318, .27137,
740: &      .24920, .22657, .20330, .17918, .15390, .12696, .09752,
741: &      .06325, .00000/
742: data(DP(I,6),I=1,21) /.50000, .47973, .45946, .43919, .41892,
743: &      .39864, .37830, .35772, .33692, .31587, .29454, .27275,
744: &      .25049, .22764, .20390, .17930, .15363, .12622, .09608,
745: &      .06173, .00000/
746: data(DP(I,7),I=1,21) /.50000, .47962, .45925, .43887, .41850,
747: &      .39812, .37777, .35718, .33635, .31545, .29451, .27329,
748: &      .25155, .22932, .20625, .18221, .15717, .13011, .10020,
749: &      .06496, .00000/
750: data(DP(I,8),I=1,21) /.50000, .47915, .45830, .43744, .41659,
751: &      .39574, .37489, .35403, .33318, .31248, .29120, .26904,
752: &      .24600, .22301, .19878, .17326, .14770, .12050, .09134,
753: &      .05931, .00000/
754: data(DP(I,9),I=1,21) /.50000, .47500, .45000, .42500, .40000,
755: &      .37500, .35000, .32500, .30000, .27500, .25000, .22500,
756: &      .20000, .17500, .15000, .12500, .10000, .07500, .05000,
757: &      .02500, .00000/
758: C
759: C
760: C
761: if ( JSTK.eq.1 ) then
762: C
763: C-----
764: C MAKE A PHOTOELECTRON.
765: C-----
766: C
767: if ( ITYP.eq.1 ) then
768: C
769: do 100 I = 1, NELC
770: WK1(I) = EELE(ISEL(I))
771: 100 continue
772: C
773: N = 14
774: call BSVDEC( EE, N, WK1, IWK1, NELC )
775: C
776: call RANU2( IRAND, R, NELC, ICON )
777: C
778: do 110 I = 1, NELC
779: if ( IWK1(I).eq.13 ) then
780: AL = ACOS(1.0D0-2.0D0*R(I)) /3.0D0

```

src/mvp/elgen.f

```

781:          AM      = 2.0D0*COS(PI23-AL)
782:          else
783:            EK      = EE(IWK1(I)) - EELE(ISEL(I))
784:            E1      = EELE(ISEL(I)) - EE(IWK1(I)+1)
785:            RR      = R(I)*20.0D0 + 1.0D0
786: C/#IF ROUNDOFF(NEAREST)
787: *            RR      = MIN( RR , 20.0D0)
788: C/#ENDIF
789:            L      = RR
790:            P      = RR - L
791:            AM1     = (1.0-P)*E1*CE(L,IWK1(I))
792:            &      + P*E1*CE(L+1,IWK1(I))
793:            &      + (1.0-P)*EK*CE(L,IWK1(I)+1)
794:            &      + P*EK*CE(L+1,IWK1(I)+1)
795:            EW      = E1N(I)
796:            AM2     = (EW+DM0C2) /SQRT(EW*(EW+2*DM0C2))
797:            AM3     = (1.0-(EE(IWK1(I))-EE(IWK1(I)+1))/AM1)*AM2
798:            AM      = MIN(MAX(AM3,-1.0D0),1.0D0)
799:          end if
800: C
801:          COSLE(I) = AM
802: 110      continue
803:        end if
804: C
805: C-----
806: C MAKE AN AUGER ELECTRON.
807: C-----
808: C
809:        if ( ITYP.eq.2 ) then
810: C
811:          call RANU2( IRAND, R, NELC ICON )
812: C
813:          do 120 I = 1, NELC
814:            COSLE(I) = R(I)*2.0D0 - 1.0D0
815: 120      continue
816:          end if
817: C
818: C-----
819: C MAKE AN ELECTRON-POSITRON PAIR.
820: C-----
821: C
822:        if ( ITYP.eq.3 ) then
823: C
824:          DM0C22 = DM0C2*2.0
825: C
826:          do 130 I = 1, NELC
827:            ALP    = E1N(ISEL(I)) /DM0C2
828:            WK1(I) = MIN(ALP,DBLE(EP(1)))
829: 130      continue
830: C
831:            N      = 9
832:            call BSVDEC( EP, N, WK1, IWK1, NELC )
833: C
834:            call RANU2( IRAND, R, NELC*3, ICON )
835:            NELC2   = NELC*2
836:            NELC3   = NELC*3
837: C
838: *VOCL LOOP,NOVREC
839:          do 140 I = 1, NELC
840:            RP1     = EP(IWK1(I)) - WK1(I)
841:            RP2     = WK1(I) - EP(IWK1(I)+1)
842: C
843:            RRRR    = R(I)*20.0D0 + 1.0D0
844: C/#IF ROUNDOFF(NEAREST)
845: *            RRRR    = MIN(RRRR,20.0D0)

```

```

846: C/#ENDIF
847:          LSR      = RRRR
848:          RP3      = RRRR - LSR
849:          EM       = E1N(ISEL(I)) - DM0C22
850: C .... ELECTRON ENERGY AND DIRECTION COSINE
851:          EELE(ISEL(I)) = EM*((1.0-RP3)*RP2*DP(LSR,IWK1(I))
852:            &      +RP3*RP2*DP(LSR+1,IWK1(I))+(1.0-RP3)*RP1*
853:            &      DP(LSR,IWK1(I)+1)+RP3*RP1*DP(LSR+1,IWK1(I)+1)) /
854:            &      (EP(IWK1(I))-EP(IWK1(I)+1))
855:          if ( R(I+NELC).gt.0.5 ) then
856:            EELE(ISEL(I)) = EM - EELE(ISEL(I))
857:          end if
858: C
859:          B        = SQRT(E1N(ISEL(I))*(E1N(ISEL(I))+DM0C22))
860:            &      /(E1N(ISEL(I))+DM0C2)
861:          RRR3     = R(I+NELC2)*2.0D0
862:          AM       = (RRR3-1.0-B) / (RRR3*B-1.0-B)
863:          COSLE(I) = MIN(MAX(-ONE,AM),ONE)
864: C ..... POSITRON ENERGY AND DIRECTION COSINE
865: C          WK2(I) = EM - EELE(ISEL(I))
866: C          RRR4   = R(I+NELC3)*2.0D0
867: C          AM     = (RRR4-1.0-B) / (RRR4*B-1.0-B)
868: C          WK3(I) = MIN(MAX(-ONE,AM),ONE)
869: C
870:          WE(ISEL(I)) = WE(ISEL(I)) * 2.0
871: 140      continue
872:        end if
873: C
874: C-----
875: C MAKE A COMPTON RECOIL ELECTRON BY INCOHERENT SCATTERING.
876: C-----
877: C
878:        if ( ITYP.eq.4 ) then
879: C
880: *VOCL LOOP,NOVREC
881:          do 200 I = 1, NELC
882:            IP      = ISEL(I)
883:            EELE(IP) = E1N(IP) - EOUT(I)
884:            XMU     = COSL(I)
885:            BT      = E1N(IP) /EOUT(I)
886:            TT      = BT*(BT-2.0*XMU)+1.0D0
887:            if ( TT.gt.0.0 ) then
888:              AM    = (BT-XMU) /SQRT(TT)
889:              AM    = MIN(AM,1.0D0)
890:            else
891:              AM    = 0.0
892:            end if
893: C
894:            if ( XMU.gt.-1.0.and.XMU.lt.1.0 ) then
895:              B      = - SQRT(
896:            &      ABS((1.0D0-AM**2)/(1.0D0-XMU*XMU)))
897:            else
898:              B      = 0.0
899:            end if
900:            A        = AM - B*XMU
901: C
902:            AAE(IP) = A*A0(IP) + B*AA(I)
903:            BBE(IP) = A*B0(IP) + B*BB(I)
904:            CCE(IP) = A*C0(IP) + B*CC(I)
905: 200      continue
906:          end if
907: C
908: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
909: C
910:          if( ITYP.ne.4 ) then

```


src/mvp/elgen.f

```

911:      call RANU2( IRAND, R, NELC*2, ICON )
912: C
913: *VOCL LOOP,NOVREC
914:      do 300 I = 1, NELC
915:          IP      = ISEL(I)
916: C
917: C      .... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
918: C
919:          DA0      = A0(IP)
920:          DB0      = B0(IP)
921:          DC0      = C0(IP)
922:          ETA      = PI2*R(I)
923:          SINETA   = SIN(ETA)
924:          COSETA   = COS(ETA)
925:          STHETA   = 1.0 - DA0*DA0
926: *VOCL STMT,IF(100)
927:          if ( STHETA.gt.0 ) then
928:              STHETA = SQRT(STHETA)
929:              COSPHI = DB0/STHETA
930:              SINPHI = DC0/STHETA
931:          else
932:              STHETA = 0.0
933:              COSPHI = 1.0
934:              SINPHI = 0.0
935:          end if
936:          XMU      = COSLE(I)
937:          SINPSI   = SQRT(ABS(1.0-XMU*XMU))
938:          DB1      = DB0*XMU +
939:          &        (DA0*COSPHI*COSETA-SINPHI*SINETA)*SINPSI
940:          DC1      = DC0*XMU +
941:          &        (DA0*SINPHI*COSETA+COSPHI*SINETA)*SINPSI
942:          DA1      = DA0*XMU - COSETA*SINPSI*STHETA
943:          S        = 1.0/SQRT(DA1*DA1+DB1*DB1+DC1*DC1)
944:          AAE(IP)  = DA1*S
945:          BBE(IP)  = DB1*S
946:          CCE(IP)  = DC1*S
947:      300      continue
948:      end if
949: C
950: C=====
951: C      sequential processing og bank and other data
952: C=====
953: C
954:      else
955: C-----
956: C
957: C      MAKE A PHOTOELECTRON.
958: C-----
959: C
960:      if ( ITYP.eq.1 ) then
961: C
962:          N      = 14
963:          call BSVDEC( EE, N, EELE, IWK1, NELC )
964: C
965:          call RANU2( IRAND, R, NELC, ICON )
966: C
967:          do 410 I = 1, NELC
968:              if ( IWK1(I).eq.13 ) then
969:                  AL      = ACOS(1.0D0-2.0D0*R(I)) /3.0D0
970:                  AM      = 2.0D0*COS(PI23-AL)
971:              else
972:                  EK      = EE(IWK1(I)) - EELE(I)
973:                  EI      = EELE(I) - EE(IWK1(I)+1)
974:                  RR      = R(I)*20.0D0 + 1.0D0
975: C/#IF ROUND OFF(NEAREST)

```

```

976: *          RR      = MIN( RR , 20.0D0)
977: C/#ENDIF
978:          L      = RR
979:          P      = RR - L
980:          AM1     = (1.0-P)*E1*CE(L,IWK1(I))
981:          &      + P*E1*CE(L+1,IWK1(I))
982:          &      + (1.0-P)*EK*CE(L,IWK1(I)+1)
983:          &      + P*EK*CE(L+1,IWK1(I)+1)
984:          EW      = EIN(I)
985:          AM2     = (EW+DMOC2) /SQRT(EW*(EW+2*DMOC2))
986:          AM3     = (1.0-(EE(IWK1(I))-EE(IWK1(I)+1))/AM1)*AM2
987:          AM      = MIN(MAX(AM3,-1.0D0),1.0D0)
988:      end if
989: C
990:          COSLE(I) = AM
991:      410      continue
992:      end if
993: C
994: C-----
995: C      MAKE AN AUGER ELECTRON.
996: C-----
997: C
998:      if ( ITYP.eq.2 ) then
999: C
1000:          call RANU2( IRAND, R, NELC ICON )
1001: C
1002:          do 420 I = 1, NELC
1003:              COSLE(I) = R(I)*2.0D0 - 1.0D0
1004:          420      continue
1005:          end if
1006: C
1007: C-----
1008: C      MAKE AN ELECTRON-POSITRON PAIR.
1009: C-----
1010: C
1011:      if ( ITYP.eq.3 ) then
1012: C
1013:          DMOC22   = DMOC2*2.0
1014: C
1015:          do 430 I = 1, NELC
1016:              ALP   = EIN(I) /DMOC2
1017:              WK1(I) = MIN(ALP,DBLE(EP(1)))
1018:          430      continue
1019: C
1020:          N        = 9
1021:          call BSVDEC( EP, N, WK1, IWK1, NELC )
1022: C
1023:          call RANU2( IRAND, R, NELC*3, ICON )
1024:          NELC2    = NELC*2
1025:          NELC3    = NELC*3
1026: C
1027: *VOCL LOOP,NOVREC
1028:          do 450 I = 1, NELC
1029:              RP1   = EP(IWK1(I)) - WK1(I)
1030:              RP2   = WK1(I) - EP(IWK1(I)+1)
1031: C
1032:              RRRR  = R(I)*20.0D0 + 1.0D0
1033: C/#IF ROUND OFF(NEAREST)
1034: *          RRRR  = MIN(RRRR,20.0D0)
1035: C/#ENDIF
1036:              LSR   = RRRR
1037:              RP3   = RRRR - LSR
1038:              EM     = EIN(I) - DMOC22
1039: C      .... ELECTRON ENERGY AND DIRECTION COSINE
1040:              EELE(I) = EM*((1.0-RP3)*RP2*DP(LSR,IWK1(I))

```

src/mvp/elgen.f

```

1041:      &      +RP3*RP2*DP(LSR+1,IWK1(I))+(1.0-RP3)*RP1*
1042:      &      DP(LSR,IWK1(I)+1)+RP3*RP1*DP(LSR+1,IWK1(I)+1)) /
1043:      &      (EP(IWK1(I))-EP(IWK1(I)+1))
1044:      if ( R(I+NELC).gt.0.5 ) then
1045:      EELE(I) = EM - EELE(I)
1046:      end if
1047: C
1048:      B      = SQRT(EIN(I)*(EIN(I)+DMOC22))
1049:      &      /(EIN(I)+DMOC2)
1050:      RRR3    = R(I+NELC2)*2.0D0
1051:      AM      = (RRR3-1.0-B)/(RRR3*B-1.0-B)
1052:      COSLE(I) = MIN(MAX(-ONE,AM),ONE)
1053: C ..... POSITRON ENERGY AND DIRECTION COSINE
1054: C      WK2(I) = EM - EELE(I)
1055: C      RRR4    = R(I+NELC3)*2.0D0
1056: C      AM      = (RRR4-1.0-B)/(RRR4*B-1.0-B)
1057: C      WK3(I)  = MIN(MAX(-ONE,AM),ONE)
1058: C
1059:      WE(I) = WE(I) + 2.0
1060: 450      continue
1061:      end if
1062: C
1063: C-----
1064: C MAKE A COMPTON RECOIL ELECTRON BY INCOHERENT SCATTERING.
1065: C-----
1066: C
1067:      if ( ITYP.eq.4 ) then
1068: C
1069:      *VOCL LOOP,NOVREC
1070:      do 500 I = 1, NELC
1071:      EELE(I) = EIN(I) - EOUT(I)
1072:      XMU     = COSL(I)
1073:      BT      = EIN(I) /EOUT(I)
1074:      TT      = BT*(BT-2.0*XMU)+1.0D0
1075:      if ( TT.gt.0.0 ) then
1076:      AM      = (BT-XMU) /SQRT(TT)
1077:      AM      = MIN(AM,1.0D0)
1078:      else
1079:      AM      = 0.0
1080:      end if
1081: C
1082:      if ( XMU.gt.-1.0.and.XMU.lt.1.0 ) then
1083:      B      = - SQRT(
1084:      &      ABS((1.0D0-AM**2)/(1.0D0-XMU*XMU)))
1085:      else
1086:      B      = 0.0
1087:      end if
1088:      A      = AM - B*XMU
1089: C
1090:      AAE(I) = A*A0(I) + B*AA(I)
1091:      BBE(I) = A*B0(I) + B*BB(I)
1092:      CCE(I) = A*C0(I) + B*CC(I)
1093: 500      continue
1094:      end if
1095: C
1096: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
1097: C
1098:      if( ITYP.ne.4 ) then
1099:      call RANU2( IRAND, R, NELC*2, ICON )
1100: C
1101:      *VOCL LOOP,NOVREC
1102:      do 600 I = 1, NELC
1103: C
1104: C ..... DETERMINE THE EMISSION ANGLE VECTOR OF ELECTRON AND POSITRON
1105: C

```

```

1106:      DA0     = A0(I)
1107:      DB0     = B0(I)
1108:      DC0     = C0(I)
1109:      ETA     = PI2*R(I)
1110:      SINETA  = SIN(ETA)
1111:      COSETA  = COS(ETA)
1112:      STHETA  = 1.0 - DA0*DA0
1113:      *VOCL STMT,IF(100)
1114:      if ( STHETA.gt.0 ) then
1115:      STHETA  = SQRT(STHETA)
1116:      COSPHI  = DB0/STHETA
1117:      SINPHI  = DC0/STHETA
1118:      else
1119:      STHETA  = 0.0
1120:      COSPHI  = 1.0
1121:      SINPHI  = 0.0
1122:      end if
1123:      XMU     = COSLE(I)
1124:      SINPSI  = SQRT(ABS(1.0-XMU*XMU))
1125:      DB1     = DB0*XMU +
1126:      &      (DA0*COSPHI*COSETA-SINPHI*SINETA)*SINPSI
1127:      DC1     = DC0*XMU +
1128:      &      (DA0*SINPHI*COSETA+COSPHI*SINETA)*SINPSI
1129:      DA1     = DA0*XMU - COSETA*SINPSI*STHETA
1130:      S       = 1.0/SQRT(DA1*DA1+DB1*DB1+DC1*DC1)
1131:      AAE(I)  = DA1*S
1132:      BBE(I)  = DB1*S
1133:      CCE(I)  = DC1*S
1134: 600      continue
1135:      end if
1136: C
1137:      end if
1138: C
1139:      return
1140:      end

```

src/mvp/elmntr.f

```

1:      subroutine ELMNTR( IOW, NLOOP, MACT, MZONE, MMAX,
2:      &                  NZONE,NREFS, NLBZ, NBANK,
3:      &                  NFFL,  LSFFL, IZFFL,
4:      &                  NNXT,  IDEFER, LSSRC, IZNXT,
5:      &                  NCOLS, LSCOL,
6:      &                  NBREF, LSREF,
7:      &                  NXLT,  LSLAT,
8:      &                  NCOLP, LSCLP,
9:      &                  IZZ, XXX, YYY, ZZZ, AAA, BBB, CCC,
10:     &                  EEE, TTT, WWW )
11: C=====
12: C purpose: Monitoring output for particles suspicious to be in
13: C         endless loop.
14: C=====
15: C
16:     include 'INC/_FLAGS'
17: C
18:     integer NFFL(NZONE+1), LSFFL(NBANK), IZFFL(NBANK)
19:     integer NNXT(NZONE+1), IDEFER, LSSRC(NBANK), IZNXT(NBANK)
20:     integer NCOLS, LSCOL(NBANK)
21:     integer NBREF(NREFS+1), LSREF(NBANK)
22:     integer NXLT(NLBZ+1), LSLAT(NBANK)
23:     integer NCOLP, LSCLP(NBANK)
24: C
25:     integer IZZ(NBANK)
26:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
27:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
28:     real EEE(NBANK)
29:     real*8 TTT(NBANK)
30:     real WWW(NBANK)
31: C
32: C-----
33: C
34: C         ... select particle whose coordinates etc. are printed
35: C
36:     IP      = 0
37:     if ( MACT.eq.1 ) then
38:       do 100 I = 1, NFFL(NZONE+1)
39:         if ( IZFFL(I).eq.MZONE.and.LSFFL(I).gt.IP ) then
40:           IP      = LSFFL(I)
41:         end if
42:       continue
43:     else if ( MACT.eq.2.and.MZONE.ne.0 ) then
44:       do 110 I = 1, NNXT(NZONE+1)
45:         if ( IZNXT(I).eq.MZONE.and.LSSRC(I).gt.IP ) then
46:           IP      = LSSRC(I)
47:         end if
48:       continue
49:     else if ( MACT.eq.2.and.MZONE.eq.0 ) then
50:       do 120 I = 1, NNXT(NZONE+1)
51:         if ( IZNXT(I).lt.0.and.LSSRC(I).gt.IP ) then
52:           IP      = LSSRC(I)
53:         end if
54:       continue
55:     else if ( MACT.eq.3 ) then
56:       do 130 I = 1, NCOLS
57:         if ( LSCOL(I).gt.IP ) then
58:           IP      = LSCOL(I)
59:         end if
60:       continue
61: C
62: C         ... MACT 4 is MACT=2 and MZONE=0
63: C
64:     else if ( MACT.eq.5 ) then
65:       do 140 I = 1, NBREF(NREFS+1)

```

```

66:         if ( LSREF(I).gt.IP ) then
67:           IP      = LSREF(I)
68:         end if
69:       140 continue
70:     else if ( MACT.eq.6 ) then
71:       do 150 I = 1, NXLT(NLBZ+1)
72:         if ( LSLAT(I).gt.IP ) then
73:           IP      = LSLAT(I)
74:         end if
75:       150 continue
76:     else if ( MACT.eq.7 ) then
77:       do 160 I = 1, NCOLP
78:         if ( LSCLP(I).gt.IP ) then
79:           IP      = LSCLP(I)
80:         end if
81:       160 continue
82:     end if
83: C
84:     if ( NLOOP.eq.1 ) write(IOW,7020)
85: C
86:     write(IOW,7000) MACT, MZONE, MMAX
87: 7000 format(1X,'** EVENT: ',I2,' ZONE ',I5,' PARTICLES ',I5)
88: C
89:     if ( IP.gt.0 ) then
90:       if ( JTIME.eq.0 ) then
91:         write(IOW,7040) IP,IZZ(IP), XXX(IP), YYY(IP), ZZZ(IP),
92:         &                AAA(IP),BBB(IP), CCC(IP), EEE(IP), 0.0d0, WWW(IP)
93:       else
94:         write(IOW,7040) IP,IZZ(IP), XXX(IP), YYY(IP), ZZZ(IP),
95:         &                AAA(IP),BBB(IP), CCC(IP), EEE(IP), TTT(IP), WWW(IP)
96:       end if
97:     end if
98: C
99: 7020 format(1X,6x,' ZONE',
100:    & '      X      ',Y      ',Z      ',
101:    & '      MU      ',ETA     ',XI     ',
102:    & ' ENERGY ',TIME      ',WEIGHT')
103: 7040 format(1X,I6,I5,1P,3E13.5,3E13.5,E12.4,E12.4,E12.4)
104: C
105:     return
106: end

```

src/mvp/faloe.f

```

1:      subroutine FALOE( IXEE,  CHSYM, FILE,  IERR )
2:      C
3:      C=<MVP>=====
4:      C PURPOSE : GET FILE-NAME OF CROSS SECTION DATA OF NUCLIDE 'CHSYM'
5:      C           FOR ELECTRON
6:      C CALLED IN: XSECE      CALLS: INPIDX
7:      C=====
8:      C
9:      C < FORM OF INDEX FILE >
10:     C
11:     C * UPTO 72 COLUMNED TEST FILE.
12:     C * DATA ASSUMMED:
13:     C
14:     C REPEAT THE FOLLOWING PAIRS OF DATA FOR EACH NUCLIDE.
15:     C
16:     C LIBRARY-ID      FILENAME
17:     C <BLANK LINE FOR FUTURE USE>
18:     C
19:     C NUCLIDE-ID & FILENAME MUST BE SEPARATED BY MORE THAN ONE BLNARS.
20:     C '*' ON THE FIRST COLUMN MEANS COMMENT LINE AND IGNORE IT.
21:     C (DO NOT INSERT ANY COMMENT LINES BETWEEN THE DATA PAIRS]])
22:     C
23:     C=====
24:     C
25:     C parameter( NCMAX      = 100 )
26:     C character CHSYM*2, FILE*(*)
27:     C
28:     C character IDS(NCMAX)*16, FILES(NCMAX)*128
29:     C
30:     C include '../shared/INC/_IOUNIT'
31:     C include 'INC/_IOUNIT2'
32:     C
33:     C character PATH*128
34:     C logical OPEND
35:     C
36:     C data LPATH /0/
37:     C data PATH  '/' '/'
38:     C data NC /0/
39:     C data IDS  /NCMAX*' '/'
40:     C data FILES /NCMAX*' '/'
41:     C
42:     C-----
43:     C GET ID & FILE-NAMES FROM UNIT FT27 FOR ELECTRON
44:     C-----
45:     C if ( NC.eq.0 ) then
46:     C
47:     C 100      NC      = NC + 1
48:     C
49:     C if ( NC.gt.NCMAX ) then
50:     C write(IMG,*)
51:     C & 'XXX(FALOE) TOO MANY DATA IN FILE-NAME INDEX. ',
52:     C & ' NCMAX = ', NCMAX, ' IN SUBROUTINE FALOE '
53:     C stop 999
54:     C end if
55:     C
56:     C call INPIDX( IDXE, IDS(NC), FILES(NC), PATH, LPATH,IMG,IEND )
57:     C
58:     C if ( IEND.eq.1 ) then
59:     C NC      = NC - 1
60:     C go to 110
61:     C end if
62:     C go to 100
63:     C
64:     C 110      continue
65:     C-----

```

```

66:     C if no element found in index file (may be file not allocated)
67:     C-----
68:     C if ( NC.le.0 ) then
69:     C write(IMG,*) 'XXX NO DATA IN ELECTRON LIBRARY INDEX FILE'
70:     C write(IMG,*) ' (PROBABLY NO FILE ASSIGNMENT ON UNIT',
71:     C & ' IDXE, ' )'
72:     C call PRSTOP( 1,
73:     C & 'ERROR IN INPUT OF ELECTRON XSEC LIBRARY INDEX FILE.'
74:     C & )
75:     C stop 888
76:     C end if
77:     C end if
78:     C-----
79:     C SEARCH ID & FILE-NAME TABLE
80:     C-----
81:     C IERR      = 0
82:     C
83:     C NZA      = NATOMZ(CHSYM)
84:     C do 120 N = 1, NC
85:     C if ( NZA.eq.NATOMZ(IDS(N)) ) then
86:     C
87:     C inquire(IXEE,opened =OPEND)
88:     C if ( OPEND ) close( IXEE )
89:     C
90:     C call OPENF( IXEE, FILES(N), 'UNFORMATTED', 'OLD', 'READ',
91:     C & JERR )
92:     C
93:     C if ( JERR.ne.0 ) then
94:     C write(IMG,*) 'XXX(FALOE) FAILED TO OPEN FILE : ',
95:     C & FILES(N)(:ICLEN2(FILES(N)))
96:     C write(IMG,*) ' CODE :', JERR
97:     C go to 9000
98:     C end if
99:     C FILE      = FILES(N)
100:    C
101:    C return
102:    C end if
103:    C 120 continue
104:    C
105:    C
106:    C 9000 IERR      = 1
107:    C write(IMG,7000) CHSYM
108:    C 7000 format(/
109:    C & 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'/
110:    C & ' XXX NO ELECTRON CROSS SECTION LIBRARY FOR <' ,A,> !!!' /
111:    C & 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'/
112:    C & )
113:    C return
114:    C end

```

src/mvp/falocf.f

```

1:      subroutine FALOCF( IXSS, NUCID, FILE, IERR )
2: C=<MVP>=====
3: C  PURPOSE : Get file-name of cross section data of nuclide 'NUCID'
4: C            from index file for free temperature neutron library.
5: C  CALLED IN: XSEC      CALLS: NOBLNK, LEN(FORTRAN-77 INTRINSIC)
6: C=====
7: C
8: C < FORM OF INDEX FILE >
9: C
10: C * Upto 72 columned text file.
11: C * Data assumed:
12: C
13: C   Repeat the following pairs of data for each nuclide.
14: C
15: C   NUCLIDE-ID   filename
16: C
17: C   NUCLIDE-ID & filename must be separated by one or more blanks.
18: C   '*' on the first column means comment line and ignore it.
19: C
20: C * Index file can include "path-specifier" such as:
21: C
22: C   PATH < directory-name / user name etc. >
23: C
24: C   An actual file name is composed as:
25: C
26: C   path-name + filename
27: C
28: C   A path-name is effective until next path-specifier is input.
29: C   If no-path-name is specified at a time of "filename" input,
30: C   File name is "filename" itself.
31: C
32: C -----
33: C   parameter( NCMAX      = 500 )
34: C   character NUCID*(*), FILE*(*)
35: C
36: C
37: C   character IDS(NCMAX)*16, FILES(NCMAX)*128
38: C   character PATH*128
39: C   logical OPEND
40: C   include '../shared/INC/_IOUNIT'
41: C   include 'INC/_IOUNIT2'
42: C
43: C   .... STATIC DATA:  NC: NUMBER OF DATA MEMORIZED IN IDS & FILES
44: C                       IC: POSITION OF NEWEST DATA IN IDS & FILES
45: C
46: C   data NC /0/, IC /0/
47: C
48: C   .... IDS, FILES & PATH MUST BE STATIC ....
49: C
50: C   data IDS  /NCMAX*' ' /
51: C   data FILES /NCMAX*' ' /
52: C   data PATH  /' ' /
53: C   data LPATH /0/
54: C
55: C   .... IRWND: NUMBER OF REWIND OPERATIONS FOR INDEX FILE ....
56: C   ( IF SEARCHING OPERATION NEEDS REWIND TWICE, IT MEANS
57: C     NO-MATCHING ID IN THE FILE. )
58: C
59: C   IRWND      = 0
60: C
61: C   IERR       = 0
62: C
63: C   LNV        = INDEX(NUCID,' ') - 1
64: C   if ( LNV.lt.0 ) LNV = LEN(NUCID)
65: C

```

```

66: C-----
67: C   SEARCH FILE NAME IN MEMORIZED DATA.
68: C-----
69: C
70: C   if ( NC.gt.0 ) then
71: C     do 100 N = 1, NC
72: C       LDS      = INDEX(IDS(N),' ') - 1
73: C       if ( LDS.lt.0 ) LDS = LEN(IDS(N))
74: C       if ( LNV.eq.LDS.and.NUCID(1:LNV).eq.IDS(N)(1:LDS) ) then
75: C
76: C         inquire(IXSS,opened =OPEND)
77: C         if ( OPEND ) close( IXSS )
78: C
79: C         call OPENF( IXSS, FILES(N), 'UNFORMATTED', 'OLD', 'READ',
80: C                   &
81: C                   JERR )
82: C
83: C         if ( JERR.ne.0 ) then
84: C           write(IMG,*) 'XXX(FALOCF) FAILED TO OPEN FILE : ',
85: C             &
86: C             FILES(N)(:ICLEN2(FILES(N)))
87: C           go to 120
88: C         end if
89: C
90: C         FILE      = FILES(N)
91: C       end if
92: C     100 continue
93: C   end if
94: C-----
95: C   GET ID & FILE-NAMES FROM UNIT FT25
96: C   IC : POSITION OF NEWLY INPUT ITEM.
97: C-----
98: C   110 continue
99: C
100: C   if ( NC.lt.NCMAX ) then
101: C     IC      = NC + 1
102: C   else
103: C     if ( IC.lt.NC ) then
104: C       IC      = IC + 1
105: C     else
106: C       IC      = 1
107: C     end if
108: C   end if
109: C
110: C   call INPIDX( IDXNF, IDS(IC), FILES(IC), PATH, LPATH, IMG, IEND )
111: C
112: C   .... CHECK NEWLY FETCHED NUCLIDE ID ....
113: C
114: C   if ( IEND.eq.0 ) then
115: C     if ( NC.lt.NCMAX ) NC      = NC + 1
116: C
117: C     LDS      = INDEX(IDS(IC),' ') - 1
118: C     if ( LDS.lt.0 ) LDS = LEN(IDS(IC))
119: C
120: C     if ( LNV.eq.LDS.and.NUCID(1:LNV).eq.IDS(IC)(1:LDS) ) then
121: C       inquire(IXSS,opened =OPEND)
122: C       if ( OPEND ) close( IXSS )
123: C
124: C       call OPENF( IXSS, FILES(IC), 'UNFORMATTED', 'OLD', 'READ',
125: C                 &
126: C                 JERR )
127: C
128: C       if ( JERR.ne.0 ) then
129: C         write(IMG,*) 'XXX(FALOCF) FAILED TO OPEN FILE: ',
130: C           &
131: C           FILES(IC)(:ICLEN2(FILES(IC)))
132: C         go to 120

```

src/mvp/falocf.f

```
131:          end if
132: C
133:          FILE      = FILES(IC)
134: C
135:          return
136:        end if
137: C
138: C      ..... END OF INDEX FILE ENCOUNTERED .....
139: C
140:        else
141:          IRWND      = IRWND + 1
142:          if ( IRWND.gt.1 ) go to 120
143: C
144:          rewind IDXNF
145:        end if
146: C
147:        go to 110
148: C
149: C
150:        120 IERR      = 1
151:          write(IMG,7000) NUCID(:LNV)
152:        7000 format(/1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
153:          &          1X,' XXX  NO CROSS SECTION LIBRARY FOR <'A,'> !!!'//
154:          &          1X,' XXX  (Free temperature neutron library)'/
155:          &          1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'//)
156:          return
157:        end
```

src/mvp/faloc.f

```

1:      subroutine FALOC( IXSS, NUCID, FILE, IERR )
2: C=<MVP>=====
3: C  PURPOSE : Get file-name of cross section data of nuclide 'NUCID'
4: C  CALLED IN: XSEC      CALLS: NOBLNK, LEN(FORTRAN-77 INTRINSIC)
5: C
6: C=====
7: C
8: C < FORM OF INDEX FILE >
9: C
10: C * Upto 72 columned text file.
11: C * Data assumed:
12: C
13: C   Repeat the following pairs of data for each nuclide.
14: C
15: C   NUCLIDE-ID      filename
16: C
17: C   NUCLIDE-ID & filename must be separated by one or more blanks.
18: C   '*' on the first column means comment line and ignore it.
19: C
20: C * Index file can include "path-specifier" such as:
21: C
22: C   PATH < directory-name / user name etc. >
23: C
24: C   An actual file name is composed as:
25: C
26: C   path-name + filename
27: C
28: C   A path-name is effective until next path-specifier is input.
29: C   If no-path-name is specified at a time of "filename" input,
30: C   File name is "filename" itself.
31: C
32: C -----
33:      parameter( NCMAX      = 500 )
34:      character NUCID*(*), FILE*(*)
35: C
36: C
37:      character IDS(NCMAX)*16, FILES(NCMAX)*256
38:      character PATH*128
39:      logical OPEND
40:      include '../shared/INC/_IOUNIT'
41:      include 'INC/_IOUNIT2'
42: C
43: C .... STATIC DATA:  NC: NUMBER OF DATA MEMORIZED IN IDS & FILES
44: C                    IC: POSITION OF NEWEST DATA IN IDS & FILES
45: C
46:      data NC /0/, IC /0/
47: C
48: C .... IDS, FILES & PATH MUST BE STATIC ....
49: C
50:      data IDS  /NCMAX*' '/
51:      data FILES /NCMAX*' '/
52:      data PATH  /' '/
53:      data LPATH /0/
54: C
55: C .... IRWND: NUMBER OF REWIND OPERATIONS FOR INDEX FILE ....
56: C   ( IF SEARCHING OPERATION NEEDS REWIND TWICE, IT MEANS
57: C     NO-MATCHING ID IN THE FILE. )
58: C
59:      IRWND      = 0
60: C
61:      IERR        = 0
62: C
63:      LNV          = INDEX(NUCID,' ') - 1
64:      if ( LNV.lt.0 ) LNV = LEN(NUCID)
65: C

```

```

66: C-----
67: C  SEARCH FILE NAME IN MEMORIZED DATA.
68: C-----
69: C
70:      if ( NC.gt.0 ) then
71:      do 100 N = 1, NC
72:          LDS      = INDEX(IDS(N),' ') - 1
73:          if ( LDS.lt.0 ) LDS = LEN(IDS(N))
74:          if ( LNV.eq.LDS.and.NUCID(1:LNV).eq.IDS(N)(1:LDS) ) then
75: C
76:          inquire(IXSS,opened =OPEND)
77:          if ( OPEND ) close( IXSS )
78: C
79:          call OPENF( IXSS, FILES(N), 'UNFORMATTED', 'OLD', 'READ',
80: &                  JERR )
81: C
82:          if ( JERR.ne.0 ) then
83:              write(IMG,*) 'XXX(FALOC) FAILED TO OPEN FILE : ',
84: &                  FILES(N)(:ICLEN2(FILES(N)))
85:              go to 120
86:          end if
87: C
88:          FILE      = FILES(N)
89: C
90:          return
91:          end if
92:      100 continue
93:      end if
94: C-----
95: C  GET ID & FILE-NAMES FROM UNIT FT25
96: C  IC : POSITION OF NEWLY INPUT ITEM.
97: C-----
98:      110 continue
99: C
100:      if ( NC.lt.NCMAX ) then
101:          IC      = NC + 1
102:      else
103:          if ( IC.lt.NC ) then
104:              IC      = IC + 1
105:          else
106:              IC      = 1
107:          end if
108:      end if
109: C
110: C
111:      call INPIDX( IDXN, IDS(IC), FILES(IC), PATH, LPATH, IMG, IEND )
112: C
113: C .... CHECK NEWLY FETCHED NUCLIDE ID ....
114: C
115:      if ( IEND.eq.0 ) then
116:          if ( NC.lt.NCMAX ) NC      = NC + 1
117: C
118:          LDS      = INDEX(IDS(IC),' ') - 1
119:          if ( LDS.lt.0 ) LDS = LEN(IDS(IC))
120: C
121:          if ( LNV.eq.LDS.and.NUCID(1:LNV).eq.IDS(IC)(1:LDS) ) then
122:              inquire(IXSS,opened =OPEND)
123:              if ( OPEND ) close( IXSS )
124: C
125:              call OPENF( IXSS, FILES(IC), 'UNFORMATTED', 'OLD', 'READ',
126: &                  JERR )
127: C
128:              if ( JERR.ne.0 ) then
129:                  write(IMG,*) 'XXX(FALOC) FAILED TO OPEN FILE: ',
130: &                  FILES(IC)(:ICLEN2(FILES(IC)))

```

src/mvp/faloc.f

```
131:          go to 120
132:      end if
133: C
134:          FILE      = FILES(IC)
135: C
136:          return
137:      end if
138: C
139: C      ..... END OF INDEX FILE ENCOUNTERED .....
140: C
141:      else
142:          IRWND      = IRWND + 1
143:          if ( IRWND.gt.1 ) go to 120
144: C
145:          rewind IDXN
146:      end if
147: C
148:      go to 110
149: C
150: C
151:      120 IERR      = 1
152:          write(IMG,7000) NUCID(:LNV)
153:      7000 format(/1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'//
154:          &      1X,' XXX  NO CROSS SECTION LIBRARY FOR <'A,'> !!!'//
155:          &      1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'//)
156:          return
157:      end
```


src/mvp/falocp.f

```

1:      subroutine FALOCP( IXPP, CHSYM, FILE, IERR )
2: C=<MVP>=====
3: C  PURPOSE : GET FILE-NAME OF CROSS SECTION DATA OF NUCLIDE 'CHSYM'
4: C  CALLED IN: XSEC      CALLS: NOBLNK, LEN(FORTRAN-77 INTRINSIC)
5: C=====
6: C
7: C < FORM OF INDEX FILE >
8: C
9: C * UPTO 72 COLUMNED TEST FILE.
10: C * DATA ASSUMMED:
11: C
12: C      REPEAT THE FOLLOWING PAIRS OF DATA FOR EACH NUCLIDE.
13: C
14: C      LIBRARY-ID      FILENAME
15: C      <BLANK LINE FOR FUTURE USE>
16: C
17: C      NUCLIDE-ID & FILENAME MUST BE SEPARATED BY MORE THAN ONE BLANKS.
18: C      '**' ON THE FIRST COLUMN MEANS COMMENT LINE AND IGNORE IT.
19: C      (DO NOT INSERT ANY COMMENT LINES BETWEEN THE DATA PAIRS!!)
20: C
21: C
22: C      parameter( NCMAX      = 500 )
23: C      character CHSYM*2, FILE*(*)
24: C
25: C
26: C      character IDS(NCMAX)*16, FILES(NCMAX)*128
27: C      include '../shared/INC/_IOUNIT'
28: C      include 'INC/_IOUNIT2'
29: C      character PATH*128
30: C      logical OPEND
31: C
32: C      data LPATH      /0/
33: C      data PATH      /' '/
34: C
35: C      data NC         /0/
36: C      data IDS        /NCMAX*' '/
37: C      data FILES      /NCMAX*' '/
38: C
39: C-----
40: C  GET ID & FILE-NAMES FROM UNIT FT26
41: C-----
42: C      if ( NC.eq.0 ) then
43: C
44: C 100      NC      = NC + 1
45: C
46: C      if ( NC.gt.NCMAX ) then
47: C          write(IMG,*)
48: C          &          'XXX(FALOCP) TOO MANY DATA IN FILE-NAME INDEX. ',
49: C          &          ' NCMAX = ', NCMAX, ' IN SUBROUTINE FALOC '
50: C          call PRSTOP( 1, 'TOO MANY DATA IN XSEC LIBRARY INDEX FILE.')
51: C          stop 999
52: C      end if
53: C
54: C      call INPIDX( IDXP, IDS(NC), FILES(NC), PATH, LPATH, IMG,IEND )
55: C      if ( IEND.eq.1 ) then
56: C          NC      = NC - 1
57: C          go to 110
58: C      end if
59: C      go to 100
60: C
61: C 110      continue
62: C-----
63: C  if no element found in index file (may be file not allocated)
64: C-----
65: C      if ( NC.le.0 ) then

```

```

66:          write(IMG,*) 'XXX NO DATA IN PHOTON LIBRARY INDEX FILE'
67:          write(IMG,*) ' (PROBABLY NO FILE ASSIGNMENT ON UNIT',
68: C          &          IDXP, ' )'
69:          call PRSTOP( 1,
70: C          &          'ERROR IN INPUT OF PHOTON XSEC LIBRARY INDEX FILE.'
71: C          &          )
72:          stop 888
73:      end if
74:  end if
75: C-----
76: C  SEARCH ID & FILE-NAME TABLE
77: C-----
78: C      IERR      = 0
79: C
80: C      NZA      = NATOMZ(CHSYM)
81: C      do 120 N = 1, NC
82: C          if ( NZA.eq.NATOMZ(IDS(N)) ) then
83: C
84: C          inquire(IXPP,opened =OPEND)
85: C          if ( OPEND ) close( IXPP )
86: C
87: C          call OPENF( IXPP, FILES(N), 'UNFORMATTED', 'OLD', 'READ',
88: C          &          JERR )
89: C
90: C          if ( JERR.ne.0 ) then
91: C              write(IMG,*) 'XXX(FALOCP) FAILED TO OPEN FILE : ',
92: C          &          FILES(N):(ICLEN2(FILES(N)))
93: C              write(IMG,*) '      CODE :', JERR
94: C              go to 9000
95: C          end if
96: C          FILE      = FILES(N)
97: C
98: C          return
99: C      end if
100: 120 continue
101: C
102: C
103: 9000 IERR      = 1
104:          write(IMG,7000) CHSYM
105: 7000 format(/1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX '/
106: C          &          1X,' XXX CAN'T GET CROSS SECTION DATA FOR <',A,'> !!!'/
107: C          &          1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX '/')
108:          return
109:      end

```

src/mvp/falocpn.f

```

1:      subroutine FALOCPN( IXPPN, NCIDPN, FILE, IERR )
2: C=<MVP>=====
3: C  PURPOSE : Get file-name of photo-nuclear cross section data of
4: C            nuclide 'NCIDPN'
5: C  CALLED IN: XSECPN
6: C  CALLS : OPENF, INPIDX, LEN(FORTRAN-77 INTRINSIC)
7: C
8: C=====
9: C
10: C < FORM OF INDEX FILE >
11: C
12: C * Upto 72 columned text file.
13: C * Data assumed:
14: C
15: C   Repeat the following pairs of data for each nuclide.
16: C
17: C   NUCLIDE-ID      filename
18: C
19: C   NUCLIDE-ID & filename must be separated by one or more blanks.
20: C   '*' on the first column means comment line and ignore it.
21: C
22: C * Index file can include "path-specifier" such as:
23: C
24: C   PATH < directory-name / user name etc. >
25: C
26: C   An actual file name is composed as:
27: C
28: C   path-name + filename
29: C
30: C   A path-name is effective until next path-specifier is input.
31: C   If no-path-name is specified at a time of "filename" input,
32: C   File name is "filename" itself.
33: C
34: C -----
35: C   include '../shared/INC/_IUNIT'
36: C   include 'INC/_IUNIT2'
37: C
38: C   parameter( NCMAX      = 500 )
39: C   character NCIDPN*(*), FILE*(*)
40: C   character IDS(NCMAX)*16, FILES(NCMAX)*128
41: C   character PATH*128
42: C   logical OPEND
43: C
44: C   .... STATIC DATA:  NC: NUMBER OF DATA MEMORIZED IN IDS & FILES
45: C                      IC: POSITION OF NEWEST DATA IN IDS & FILES
46: C   data NC /0/, IC /0/
47: C
48: C   .... IDS, FILES & PATH MUST BE STATIC ....
49: C
50: C   data IDS  /NCMAX*' ' /
51: C   data FILES /NCMAX*' ' /
52: C   data PATH  /' ' /
53: C   data LPATH /0/
54: C   data IRWND /0/
55: C   .... IRWND: NUMBER OF REWIND OPERATIONS FOR INDEX FILE ....
56: C   ( IF SEARCHING OPERATION NEEDS REWIND TWICE, IT MEANS
57: C     NO-MATCHING ID IN THE FILE. )
58: C
59: C -----
60: C
61: C   IERR      = 0
62: C   LNV       = INDEX(NCIDPN,' ') - 1
63: C   if ( LNV.lt.0 ) LNV = LEN(NCIDPN)
64: C
65: C -----

```

```

66: C   SEARCH FILE NAME IN MEMORIZED DATA.
67: C -----
68: C
69: C   if ( NC.gt.0 ) then
70: C     do 100 N = 1, NC
71: C       LDS      = INDEX(IDS(N),' ') - 1
72: C       if ( LDS.lt.0 ) LDS = LEN(IDS(N))
73: C       if ( LNV.le.LDS .and. NCIDPN(1:LNv).eq.IDS(N)(1:LNv) ) then
74: C         inquire(IXPPN,opened=OPEND)
75: C         if ( OPEND ) close( IXPPN )
76: C         call OPENF( IXPPN, FILES(N), 'UNFORMATTED', 'OLD',
77: C                   & 'READ', JERR )
78: C         if ( JERR.ne.0 ) go to 920
79: C         NCIDPN  = IDS(N)
80: C         FILE    = FILES(N)
81: C         return
82: C       end if
83: C     100 continue
84: C   end if
85: C -----
86: C   GET ID & FILE-NAMES FROM UNIT FT28
87: C   IC : POSITION OF NEWLY INPUT ITEM.
88: C -----
89: C   110 continue
90: C
91: C   if ( NC.lt.NCMAX ) then
92: C     IC      = NC + 1
93: C   else
94: C     if ( IC.lt.NC ) then
95: C       IC      = IC + 1
96: C     else
97: C       IC      = 1
98: C     end if
99: C   end if
100: C
101: C   call INPIDX( IDXPIN, IDS(IC), FILES(IC), PATH, LPATH, IMMSG, IEND )
102: C
103: C   .... CHECK NEWLY FETCHED NUCLIDE ID ....
104: C
105: C   if ( IEND.eq.0 ) then
106: C     if ( NC.lt.NCMAX ) NC      = NC + 1
107: C     LDS      = INDEX(IDS(IC),' ') - 1
108: C     if ( LDS.lt.0 ) LDS = LEN(IDS(IC))
109: C     if ( LNV.le.LDS .and. NCIDPN(1:LNv).eq.IDS(IC)(1:LNv) ) then
110: C       inquire(IXPPN,opened=OPEND)
111: C       if ( OPEND ) close( IXPPN )
112: C       call OPENF( IXPPN, FILES(IC), 'UNFORMATTED', 'OLD', 'READ',
113: C                 & JERR )
114: C       if ( JERR.ne.0 ) go to 920
115: C       NCIDPN  = IDS(IC)
116: C       FILE    = FILES(IC)
117: C       return
118: C     end if
119: C
120: C   .... END OF INDEX FILE ENCOUNTERED ....
121: C
122: C   else
123: C     IRWND      = IRWND + 1
124: C     if ( IRWND.gt.1 ) go to 910
125: C     rewind IDXPIN
126: C     NC          = 0
127: C   end if
128: C
129: C   go to 110
130: C

```

src/mvp/falocpn.f

```
131: 910 write(IMG,911) NCIDPN(:LNV)
132: 911 format('// !!!(falocpn) Photonuclear data ID <',a,'> is not',
133: & ' given in the index file. This nuclide is treated without',
134: & ' photonuclear.')
135: go to 999
136: 920 write(IMG,921) NCIDPN(:LNV), FILES(IC)(:ICLEN2(FILES(IC)))
137: 921 format('// !!!(falocpn) Failed to open the file of ID <',a,'> : ',a
138: & '/' This nuclide is treated without photonuclear.')
139: 999 call CNTERR( 'WARNING' )
140: IERR = 1
141: return
142: end
```

SAFE

src/mvp/fisgen.f

```

1:      subroutine FISGEN( IOW, IREG, INUC, IBPF, NGROUP,NZONE, NMAT,
2:      & NREG, NTIME, NBANK, NFBANK,NFBNK0,NEST, NUC,
3:      & NNK, NMT, MB, NSMIC, MMAC, WGTF,
4:      & WGTFI, WTIME, WLLIM, MXPGEN,ETOP, EBOT,
5:      & ETHMAX,CX, MCX, KLIB1, XLIB1, KLIB2,
6:      & KLB2S, XLIB2, SMIC, LMIC, LSFFL, NFFL,
7:      & IZFFL, LSCOL, NCOLS, LSDED, NDEAD, NFISB, XXX,
8:      & YYY, ZZZ, AAA, BBB, CCC, WWW, EEE,
9:      & TTT, ITT, XIM,KKP,IZZ, IGG, LEVL, LZZ,
10:     & LPOS, LCRS, KLSF, IBREG, IBSPC, XXXF, YYYYF,
11:     & ZZZF, EEEF, INUF, IZZF, LEVLF, LZZF,
12:     & LPOSF, LCRSF, IBRGF, IBSPF, DBNK, KDBNK,
13:     & MDBNK, IBNK, KIBNK, MIBNK, DFBK, KDFBK,
14:     & MDFBK, IFBK, KIFBK, MIFBK, NECUT, WECUT,
15:     & NCNTR, WCNTR, IRAND, IWK1, IWK4, IWK6, IWK7,
16:     & IWK8, IWK9, IWK10, IWK11, IWK12, IWK13, WK1,
17:     & WK2, WK3, WK4, WK5, R,
18:     & SMAC, LMAC, NSMAC, SMACP, SMICP, ! part
19:     & NPTCS,WWD ,WDO ,WDOA ,WDOB ,NBATCH , ! part
20:     & NSKIP, JPTTR, JPTNU, KZREG, NPTCS, WWR, WR0, ! part
21:     & WR0A, MAXDA, DELA , ! part
22:     & WWT,WTO,WTOA,WTOB,NUCPT, ! part
23:     & WKPT, WKPD, WKEN, ! part
24:     & NGSP, WSD, WSC, NORDD, NOFDS, ! part
25:     & RNU, IWK14, IWK15, WK6, IMTF, ! an
26:     & NEBEF, WCBEF, WK7, WK8, ! beff
27:     & TCUT, TIMEB, KPNPRD, NUC_MAX, KPNEG, ! photo-
nuc
28:     & WWB, WB0, WSB, NPTBE, ! beff
29:     & WWBD, WBD0, WSBD, WWLD, WLD0, WSLD) ! beff
30: C=====
31: C PURPOSE: Fission neutron generation
32: C CALLED IN: NEUTR
33: C CALLS: RANU2, SEF5N2
34: C-----
35: C Argument: (i=input, o=output, w=work)
36: C
37: C i IREG : region # set in caller routine (NEUTR)
38: C io INUC : coliding nuc # set in caller routine (NEUTR) and modified
39: C in this routine if contents of collision neutron stack
40: C are changed.
41: C wo IBPF : can be used as working area, but finally, bank index of
42: C generated fission neutrons must be stored.
43: C io LSCOL(NCOLS) : collision neutron stack. Contents of this stack
44: C may be changed if parent neutron is replaced by fission
45: C neutron (by lack of particle bank space).
46: C io NCOLS : may be changed by the same reason above.
47: C
48: C-----
49: C=====
50:
51: C****IMPLICIT REAL*8 (A-H,O-Z)
52:     implicit real*8(A-H,O-Q,S-Z)
53:     include 'INC/_FLAGS'
54: C
55:     integer IREG(NBANK), INUC(NBANK), IBPF(NBANK)
56: C
57: C**** CROSS SECTION DATA IN CX ARRAY
58: C
59:     real EBOT, ETOP, ETHMAX
60:     real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
61:     integer KLIB1(NUC,31), KLIB2(NUC,NMT,21), KLB2S(NUC,NNK,NMT,3)
62: C
63: C**** SIGMA BANK
64: C

```

```

65:     real SMIC(NBANK,NUC,NSMIC)
66:     real RNU(NBANK,NUC_MAX,3) ! dn: photo-nuc:NUC->NUC_MAX
67:     real WGTF(NREG,2) ! photo-nuc:(NREG)->(NREG,2)
68:     real WGTFI(NREG), TIMEB(NTIME+1), TCUT
69:     real WTIME(NTIME)
70:     real WLLIM
71:     integer MMAC(NBANK,2), LMIC(8)
72: C
73: C**** STACK
74: C
75:     integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE), LSCOL(NBANK),
76:     & LSDED(NBANK)
77: C
78: C**** PARTICLE BANK
79: C
80:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
81:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
82:     real*8 TTT(NBANK)
83:     real EEE(NBANK), WWW(NBANK), XIM(NBANK)
84:     integer KKP(NBANK)
85:     integer IZZ(NBANK), IGG(NBANK)
86:     integer ITT(NBANK)
87:     integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST)
88:     integer LCRS(NBANK,NEST)
89:     integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
90:     integer KLSF(NBANK)
91: C
92:     real*8 DBNK(NBANK,*)
93:     integer KDBNK(0:MDBNK)
94:     integer IBNK(NBANK,*)
95:     integer KIBNK(0:MIBNK)
96: c##<2007/03/14:PN4:
97:     integer KPNFG(NBANK)
98: c##>
99: C
100: C**** fission neutron bank
101: C
102:     real*8 XXXF(NFBNK0), YYYYF(NFBNK0), ZZZF(NFBNK0)
103:     real EEEF(NFBNK0)
104:     integer IZZF(NFBNK0), INUF(NFBNK0)
105:     integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST)
106:     integer LCRSF(NFBNK0,NEST), LPOSF(NFBNK0,NEST)
107:     integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
108:     integer IMTF(NFBNK0)
109: C
110: C
111:     real*8 DFBK(NFBNK0,*)
112:     integer KDFBK(0:MDFBK)
113:     integer IFBK(NFBNK0,*)
114:     integer KIFBK(0:MIFBK)
115: C
116: C**** TALLY ARRAY
117: C
118:     real*8 WECUT(NREG), WCNTR(*), NCNTR(*)
119:     real*8 NECUT(NREG)
120:     real*8 WCBEF(NEBEF) ! beff
121: C
122: C**** WORKING ARRAYS
123: C
124:     integer IWK1(NBANK), IWK4(NBANK), IWK6(NBANK), IWK7(NBANK),
125:     & IWK8(NBANK), IWK9(NBANK), IWK10(NBANK), IWK11(NBANK),
126:     & IWK12(NBANK), IWK13(NBANK)
127:     integer IWK14(NBANK), IWK15(NBANK) ! dn
128:     real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK), WK5(NBANK),
129:     & WK6(NBANK), R(8*NBANK) ! dn: R(NBANK)->R(8*NBANK)

```

src/mvp/fisgen.f

```

130:      real WK7(NBANK), WK8(NBANK) ! beff
131:
132: C 0...NBANK..2NBANK..3NBANK..4NBANK..5NBANK..6NBANK..7NBANK..8NBANK
133: C IWK1 IWK2 IWK3 IWK4 IWK5 IWK6 IWK7 RP R,IWORK
134: C
135: C
136: C**** PERTURBATION
137: C
138: c ... sigma bank ...
139:      real SMAC(NBANK,MB,NSMAC)
140:      real SMACP(NBANK,MB,NSMAC),SMICP(NBANK,NUC,NSMIC)
141:      integer LMAC(8)
142:
143:      real*8 WWD(NBANK,NPTDS,NORDDS),WWR(NBANK,NPTCS)
144:      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
145:      real*8 WSC(NBANK,0:NGSP,NPTCS)
146:      real*8 WDO(NFBNK0,0:NGSP,NPTDS,NOPSDS)
147:      real*8 WRO(NFBNK0,0:NGSP,NPTCS)
148:      integer JPTTR(NREG,NPTDS+NPTCS),JPTNU(NUC,NPTDS)
149:      integer NREG(NZONE)
150:      real DELA(MAXDA,NPTDS+NPTCS)
151:      real*8 WWT(NBANK,NPTDS),WTO(NFBNK0)
152:      real*8 WTA(NFBNK0),WTOB(NFBNK0)
153:      real*8 WWB(NBANK) ! beff
154:      real*8 WBO(NFBNK0,0:NGSP) ! beff
155:      real*8 WSB(NBANK,0:NGSP) ! beff
156:      real*8 WWBD(NBANK) ! beff
157:      real*8 WBD0(NFBNK0,NGSP) ! beff
158:      real*8 WBSD(NBANK,NGSP) ! beff
159:      real*8 WULD(NBANK) ! beff
160:      real*8 WLD0(NFBNK0,NGSP) ! beff
161:      real*8 WSLD(NBANK,NGSP) ! beff
162: c ... working area ...
163:      real*8 WKPT(NBANK,NPTDS)
164:      real*8 WKPD(NBANK,NPTDS)
165:      real*8 WKPN(NUCPT,NBANK,NPTDS)
166: c ... local variable ...
167:      real*8 SNUFP
168:      real*8 RRNU ! beff
169: C**** END OF PERTURBATION arrays and variables
170:
171: C***** INPUT
172: C IREG : IREG
173: C INUC : COLLIDING NUCLIDE
174: C
175: C***** OUTPUT
176: C
177: C IKW5 : TEMPORARY FISSION NEUTRON STACK (NFISB) SET IN FISGEN
178: C (JEIGN=0)
179: C
180: C
181:      parameter( PI = 3.1415926535897932D0 )
182:      parameter( PI2 = 2.0D0*PI )
183: C
184: C-----
185: C
186:      WMIN = WLLIM
187: C
188: C=====
189: C**** Fission neutron generation
190: C=====
191: C
192: C ... IWK4(I) is number of fission neutrons to be generated,
193: C ( not all of them are launched to random walk (fixed source)
194: C or banked for next batch (eigenvalue) )

```

```

195: C
196:      call RANU2( IRAND, R, 2*NCOLS, ICON ) ! dn:NCOLS->2*NCOLS
197: C
198:      NFISS = 0
199:      NFISD = 0 ! dn
200:      if ( JWTIM.eq.0.and.JRWVR.eq.0 ) then
201:      do 100 I = 1, NCOLS
202:          IP = LSCOL(I)
203:          WK1(I) = WWW(IP) /SMIC(IP,INUC(I),LMIC(1))
204:          JJDLYN = JDLYN
205:          if ( KLIB1(INUC(I),23).eq.0.or.KLIB2(INUC(I),98,8).le.0 )
206:          & JJDLYN = 0
207:          if ( JJDLYN.eq.0.or.
208:          & RNU(IP,INUC(I),2)+RNU(IP,INUC(I),3).le.0.0 ) then
209:              WK2(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(2)) ! total fiss
ion weight
210:              WW = WK2(I) * WGTFI(IREG(I)) ! total fiss
ion yield
211:              WK6(I) = 0. ! no delayed
fission weight
212:              WWDN = 0.0d0 ! no delayed
fission yield
213:          else
214:              WK2(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(3)) * ! prompt fis
sion weight
215:              & RNU(IP,INUC(I),3)
216:              WW = WK2(I) / WGTFI(IREG(I),1) ! prompt fis
sion yield
217:              WK6(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(3)) * ! delayed fi
ssion weight
218:              & RNU(IP,INUC(I),2)
219:              WWDN = WK6(I) / WGTFI(IREG(I),2) ! delayed fi
ssion yield
220:          end if
221:
222:          if( JBEFF.ne.0 ) then ! beff
223:              WK7(I) = WK1(I)*SMIC(IP,INUC(I),LMIC(3)) ! beff
224:          end if ! beff
225: C/#!/IF ROUNDOFF(NEAREST)
226:          if ( WW.ne.0.0d0 ) then
227:              IWK4(I) = MIN(INT(WW)+1,INT(WW+R(I)))
228:          else
229:              IWK4(I) = 0
230:          end if
231:
232:          if ( WWDN.ne.0.0d0 ) then
233:              IWK14(I) = min(int(WWDN)+1, int(WWDN+R(I+NCOLS))) ! delayed fi
ssion integer yield
234:          else
235:              IWK14(I) = 0
236:          end if
237: C/#!/ELSE
238:      * IWK4(I) = WW + R(I)
239:      * IWK14(I) = WWDN + R(I+NCOLS)
240: C/#!/ENDIF
241:      NFISS = NFISS + IWK4(I) + IWK14(I)
242:      NFISD = NFISD + IWK14(I)
243:      100 continue
244: C
245: C ... time dependent weighting factor WTIME is applied.
246: C or use weight relative to that on birth
247: C
248:      else
249:      do 110 I = 1, NCOLS
250:          IP = LSCOL(I)

```

src/mvp/fisgen.f

```

251:      WK1(I) = WWW(IP) /SMIC(IP,INUC(I),LMIC(1))
252:      JJDLYN = JDLYN
253:      if ( KLIB1(INUC(I),23).eq.0.or.KLIB2(INUC(I),98,8).le.0 )
254:      &      JJDLYN = 0
255:      if ( JJDLYN.eq.0.or.
256:      &      RNU(IP,INUC(I),2)+RNU(IP,INUC(I),3).le.0.0 ) then
257:      WK2(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(2))      ! total fiss
ion weight
258:      WW      = WGTF(IREG(I),1)      ! generation
weight of total fission
259:      if ( JWTIM.ne.0 ) WW = WW * WTIME(ITT(IP))
260:      if ( JRWVR.ne.0 ) WW = WW * DBNK(IP,KDBNK(1))
261:      WW      = max(WW, WMIN)
262:      WW      = WK2(I) / WW      ! total fiss
ion yield
263:      WK6(I) = 0.      ! no delayed
fission weight
264:      WWDN      = 0.d0      ! no delayed
fission yield
265:      else
266:      WK2(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(3)) *      ! prompt fis
sion weight
267:      &      RNU(IP,INUC(I),3)
268:      WW      = WGTF(IREG(I),1)      ! generation
weight of prompt fission
269:      if ( JWTIM.ne.0 ) WW = WW * WTIME(ITT(IP))
270:      if ( JRWVR.ne.0 ) WW = WW * DBNK(IP,KDBNK(1))
271:      WW      = max(WW, WMIN)
272:      WW      = WK2(I) / WW      ! prompt fis
sion yield
273:      WK6(I) = WK1(I) * SMIC(IP,INUC(I),LMIC(3)) *      ! delayed fi
ssion weight
274:      &      RNU(IP,INUC(I),2)
275:      WWDN      = WGTF(IREG(I),2)      ! generation
weight of delayed fission
276:      if ( JWTIM.ne.0 ) WWDN = WWDN * WTIME(ITT(IP))
277:      if ( JRWVR.ne.0 ) WWDN = WWDN * DBNK(IP,KDBNK(1))
278:      WWDN      = max(WWDN, WMIN)
279:      WWDN      = WK6(I) / WWDN      ! delayed fi
ssion yield
280:      end if
281:
282:      if( JBEFF.ne.0 ) then      ! beff
283:      WK7(I) = WK1(I)*SMIC(IP,INUC(I),LMIC(3)) ! beff
284:      end if      ! beff
285: C/#IF ROUNDOFF(NEAREST)
286:      if ( WW.ne.0.0d0 ) then
287:      IWK4(I) = MIN(INT(WW)+1,INT(WW+R(I)))
288:      else
289:      IWK4(I) = 0
290:      end if
291:
292:      if ( WWDN.ne.0.0d0 ) then
293:      IWK14(I) = min(int(WWDN)+1, int(WWDN+R(I+NCOLS))) ! delayed fi
ssion integer yield
294:      else
295:      IWK14(I) = 0
296:      end if
297: C/#ELSE
298: *      IWK4(I) = WW + R(I)
299: *      IWK14(I) = WWDN + R(I+NCOLS)
300: C/#ENDIF
301:      NFISS      = NFISS + IWK4(I) + IWK14(I)
302:      NFISSD      = NFISSD + IWK14(I)
303:      110      continue

```

```

304:      end if
305: C
306: C=====
307: C      Fixed source problem
308: C=====
309: C
310:      if ( JEIGN.eq.0 ) then
311: C
312: C      ... check generation cutoff
313: C
314:      if ( NFISS.gt.0.and.MXPGEN.gt.0 ) then
315:      NGCUT      = 0
316:      c%120      continue
317: C
318:      c%130      continue
319:      NCUT      = 0
320:      WWC      = 0
321:      if ( JDLYN.eq.0 ) then
322:      do I = 1, NCOLS
323:      IP      = LSCOL(I)
324:      if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.
325:      &      IWK4(I).gt.0 ) then
326:      NGCUT      = NGCUT + 1
327:      NCUT      = NCUT + IWK4(I)
328:      WWC      = WWC + WK2(I)
329:      IWK4(I) = 0
330:      end if
331:      end do
332:      else
333:      do I = 1, NCOLS
334:      IP      = LSCOL(I)
335:      if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.
336:      &      IWK4(I)+IWK14(I).gt.0 ) then
337:      NGCUT      = NGCUT + 1
338:      NCUT      = NCUT + IWK4(I) + IWK14(I)
339:      WWC      = WWC + WK2(I) + WK6(I)
340:      NFISSD      = NFISSD - IWK14(I)
341:      IWK4(I) = 0
342:      IWK14(I)= 0
343:      end if
344:      end do
345:      end if
346:      if ( NGCUT.gt.0 ) then
347:      NCNTR(27) = NCNTR(27) + NCUT
348:      WCNTR(27) = WCNTR(27) + WWC
349:      NFISS      = NFISS - NCUT
350:      end if
351:      end if
352: C
353: C
354:      if ( NFISS.le.0 ) then
355:      NFISB      = 0
356:      return
357:      end if
358:
359:      do I = 1, NBANK ! dn
360:      IWK15(I) = 0 ! dn
361:      end do      ! dn
362: C
363: C      Fission neutrons are stored in particle bank(XXX,YYY,.....)
364: C
365: C      IBPF : fission neutron stack
366: C
367: C***** All fission neutron can be stored in bank
368: C

```

src/mvp/fisgen.f

```

369:         if ( NFISS.le.NDEAD ) then
370:             MA = 0
371:             c%140 continue
372:             if ( JDLYN.eq.0.or.NFISSD.le.0 ) then
373:                 do I = 1, NCOLS
374:                     MA = max(MA, IWK4(I))
375:                 end do
376:             else
377:                 do I = 1, NCOLS
378:                     MA = max(MA, IWK4(I), IWK14(I))
379:                 end do
380:             end if
381: C
382: C ..... IWK6 : STACK POINTER
383: C
384: N = 0
385: do 160 K = 1, MA
386:     do 150 I = 1, NCOLS
387:         if ( IWK4(I).ge.K ) then
388:             N = N + 1
389:             IWK6(N) = I
390: c##<2007/03/14:PN3:
391: c## WK3(N) = WGTF(IREG(I))
392: WK3(N) = WGTF(IREG(I),1) ! generation we
ight of total or prompt fission
393: c##>
394:         end if
395:         if ( JDLYN.ne.0.and.NFISSD.gt.0 ) then
396:             if ( IWK14(I).ge.K ) then
397:                 N = N + 1
398:                 IWK6(N) = I
399:                 IWK15(N)= 98 ! flag of delay
ed neutron
400: c##<2007/03/14:PN3:
401: c## WK3(N) = WGTF(IREG(I))
402: WK3(N) = WGTF(IREG(I),2) ! generation we
ight of delayed fission
403: c##>
404:                 NCNTR(30) = NCNTR(30) + 1 ! event monitor
: delayed fission
405: c##<2007/03/14:PN3:
406: c## WCNTR(30) = WCNTR(30) + WGTF(IREG(I))
407: WCNTR(30) = WCNTR(30) + WGTF(IREG(I),2)
408: c##>
409:             end if
410:         end if
411:         150 continue
412:         160 continue
413: C
414: C ... apply WTIME (time dependent weighting factor)
415: C
416:         if ( JWTIM.ne.0 ) then
417:             do 170 J = 1, N
418:                 WK3(J) = WK3(J)*WTIME(ITT(LSCOL(IWK6(J))))
419:             170 continue
420:         end if
421: C
422: C ... WGTF is relative to weight on birth
423: C
424:         if ( JRWVR.ne.0 ) then
425:             do 180 J = 1, N
426:                 WK3(J) = WK3(J)*DBNK(LSCOL(IWK6(J)),KDBNK(1))
427:             180 continue
428:         end if
429:         do 190 J = 1, N
430:             WK3(J) = MAX(WK3(J),WLLIM)
431:             190 continue
432: C
433: C***** not all fission neutrons can be stored in bank
434: C
435:             else
436: C
437:                 NFISS = 0
438:                 N = 0
439:                 I1 = 0
440:                 NDLL = 0 ! dn
441:                 do 210 I = 1, NCOLS
442:                     K = IWK4(I)
443:                     if ( JDLYN.ne.0.and.NFISSD.gt.0 ) then
444:                         K1 = IWK14(I)
445:                     else
446:                         K1 = 0
447:                     end if
448:                     NFISS = NFISS + K + K1
449:                     if ( NFISS.gt.NDEAD ) go to 230
450:                     I1 = I
451:                     do 200 II = 1, K
452:                         N = N + 1
453:                         IWK6(N) = I
454:                         WK3(N) = WGTF(IREG(I),1) ! generation we
ight of total or prompt fission
455:                         200 continue
456:                         if ( K1.gt.0 ) then ! dn
457:                             do II = 1, K1 ! dn
458:                                 N = N + 1 ! dn
459:                                 IWK6(N) = I ! dn
460:                                 IWK15(N)= 98 ! flag of delay
ed neutron
461: WK3(N) = WGTF(IREG(I),2) ! generation we
ight of delayed fission
462: NCNTR(30) = NCNTR(30) + 1 ! event monitor
: delayed fission
463: WCNTR(30) = WCNTR(30) + WGTF(IREG(I),2) ! dn
464:         end do ! dn
465:                 NDLL = NDLL + K1 ! dn
466:             end if ! dn
467:             210 continue
468: C
469: C ( program never comes here (because NFISS > NDEAD))
470:             do 220 J = 1, N
471:                 WK3(J) = MAX(WK3(J),WLLIM)
472:             220 continue
473:             go to 290
474: C
475: C .... number of fission neutron is modified here ....
476: C
477:             230 NFISS = N
478:             NFISSD = NDLL ! dn
479: C
480: C ... apply WTIME (time dependent weighting factor)
481: C
482:             if ( JWTIM.ne.0 ) then
483:                 do 240 J = 1, N
484:                     WK3(J) = WK3(J)*WTIME(ITT(LSCOL(IWK6(J))))
485:                 240 continue
486:             end if
487: C
488: C ... WGTF is relative to weight on birth
489: C
490:             if ( JRWVR.ne.0 ) then

```

src/mvp/fisgen.f

```

491:      do 250 J = 1, N
492:        WK3(J) = WK3(J)*DBNK(LSCOL(IWK6(J)),KDBNK(1))
493:      250      continue
494:    end if
495:    do 260 J = 1, N
496:      WK3(J) = MAX(WK3(J),WLLIM)
497:    260      continue
498:  C
499:  C
500:  C .... count fission neutron not launched to random walk ....
501:  C
502:  C%270      continue
503:    if ( JDLYN.eq.0 ) then
504:      do I = I1 + 1, NCOLS
505:        NCNTR(13) = NCNTR(13) + IWK4(I)
506:        WCNTR(13) = WCNTR(13) + WK2(I)
507:      end do
508:    else
509:      do I = I1 + 1, NCOLS
510:        NCNTR(13) = NCNTR(13) + IWK4(I) + IWK14(I)
511:        WCNTR(13) = WCNTR(13) + WK2(I) + WK6(I)
512:      end do
513:    end if
514:  C
515:  C .... Fission neutrons not banked above are replaced with parent
516:  C neutrons with probability Sigma(I)/Sigma(S).
517:  C Replaced parent neutron dies.
518:  C
519:    I2 = I1
520:    I3 = NCOLS - I1
521:    if ( JDLYN.eq.0.or.NFISSD.le.0 ) then
522:      call RANU2( IRAND, R, I3, ICON )
523:    else
524:      call RANU2( IRAND, R, I3*2, ICON )
525:    end if
526:  C
527:  *VOCL LOOP,NOVREC
528:    do 280 I = I1 + 1, NCOLS
529:  C
530:      IP = LSCOL(I)
531:      IWK3I = INUC(I)
532:  C
533:  C
534:  C      X1 =SMIC(IP,IWK3I,LMIC(1))/(SMIC(IP,IWK3I,LMIC(1))
535:  C      +      -SMIC(IP,IWK3I,LMIC(3))-SMIC(IP,IWK3I,LMIC(5)) )
536:  C      WWW(IP) = WWW(IP) + WK2(I)*X1
537:  C
538:  C ..... S1 : FISSION CROSS SECTION
539:  C      ST : TOTAL CROSS SECTION
540:  C
541:  C      S1 = SMIC(IP,IWK3I,LMIC(3)) ! fission cross section
542:  C      ST = SMIC(IP,IWK3I,LMIC(1)) ! total cross section
543:  C
544:  C
545:  C      if ( IWK4(I)+IWK14(I).gt.0.and.ST.gt.0.d0 ) then
546:  C        IR = S1 / ST + R(I-I1)
547:  C        if ( IR.gt.0.and.JDLYN.ne.0.and.IWK14(I).gt.0 ) then
548:  C          if ( IWK4(I).le.0 ) then
549:  C            IR = 2
550:  C          else if ( RNU(IP,IWK3I,3).le.0.0 ) then
551:  C            IR = 2
552:  C          else if ( RNU(IP,IWK3I,2).gt.0.0 ) then
553:  C            R1 = RNU(IP,IWK3I,2) / RNU(IP,IWK3I,1)
554:  C            if ( R1.le.R(I-I1+I3) ) IR = 2
555:  C
556:  C      end if
557:  C    end if
558:  C
559:  C      incident neutron energy : WK4 FROM (NFISS+1)
560:  C      colliding nuclide : IWK1 FROM (NFISS+1)
561:  C      bank pointer : IWK6 FROM (NFISS+1)
562:  C
563:  C      Parent neutron is killed
564:  C
565:  C      ... region # is memorized here
566:  C      if ( IR.eq.-1 ) then ! scattering ... pare
567:  C        nt is survived
568:  C        I2 = I2 + 1
569:  C        LSCOL(I2) = IP
570:  C        INUC(I2) = IWK3I
571:  C      else if ( IR.eq.0 ) then ! scattering ... pare
572:  C        nt is survived
573:  C        I2 = I2 + 1
574:  C        LSCOL(I2) = IP
575:  C        INUC(I2) = IWK3I
576:  C        WWW(IP) = WWW(IP)*ST/(ST-S1)
577:  C      else ! fission ..... pare
578:  C        nt is killed
579:  C        WCNTR(24) = WCNTR(24) + WWW(IP)
580:  C        N = N + 1
581:  C        IWK6(N) = IP ! bank pointer f
582:  C
583:  C      rom NFISS+1 to N
584:  C      ide from NFISS+1 to N
585:  C      IWK1(N) = IWK3I ! colliding nucl
586:  C      IWK13(N)= IREG(I) ! region number
587:  C
588:  C      from NFISS+1 to N
589:  C      WK4(N) = EEE(IP) ! incident neutr
590:  C      on energy from NFISS+1 to N
591:  C      if ( IR.eq.1 ) then
592:  C        WWW(IP) = WK2(I) * ST / S1 ! adjusted weigh
593:  C      else if ( IR.eq.2 ) then
594:  C        WWW(IP) = WK6(I) * ST / S1 ! adjusted weigh
595:  C      t of total or prompt
596:  C      t of delayed
597:  C      IWK15(N)= 98 ! flag of delaye
598:  C      d neutron
599:  C      NCNTR(30) = NCNTR(30) + 1 ! event monitor:
600:  C      delayed fission
601:  C      WCNTR(30) = WCNTR(30) + WWW(IP)
602:  C      end if
603:  C    end if
604:  C
605:  C      continue
606:  C
607:  C      NCNTR(24) = NCNTR(24) + NCOLS - I2
608:  C      NCOLS = I2
609:  C
610:  C      280      continue
611:  C
612:  C      290      continue
613:  C
614:  C      end if
615:  C
616:  C      605:  C
617:  C      606:  C
618:  C      607:  C ..... N = NFISS when all fission neutrons are stored in the bank

```


src/mvp/fisgen.f

```

608: C      N = NFISS (fission n with weight WGTF) +
609: C      fission replacing parents with weight WGTF(*factor)
610: C
611: C
612: C      if ( N.le.0 ) then
613: C          NFISB = 0
614: C          return
615: C      end if
616: C
617: C ***** count fission neutrons *****
618: C
619: C *VOCL LOOP,NOVREC
620: C      do 300 I = 1, NFISS
621: C          WCNTR(2) = WCNTR(2) + WK3(I)
622: C          continue
623: C      do 310 I = NFISS + 1, N
624: C          WCNTR(2) = WCNTR(2) + WWW(IWK6(I))
625: C          continue
626: C      NCNTR(2) = NCNTR(2) + N
627: C
628: C
629: C ***** determine energy of fission neutrons *****
630: C
631: C
632: C      Incident neutron energy : WK4
633: C      Colliding nuclide       : IWK1
634: C      Fission neutron energy  ---> WK2
635: C
636: C
637: C      .... IWK1(I) IWK6(I) and WK4(I) for I=NFISS+1,N is set above in
638: C      parent/fission-n survival game
639: C
640: C      do 320 I = 1, NFISS
641: C          II = IWK6(I)
642: C          IWK1(I) = INUC(II)
643: C          WK4(I) = EEE(LSCOL(II))
644: C      continue
645: C
646: C      call SEF5N4( IOW, IWK1, N, WK4, WK2, CX, CX, MCX, KLIB1, XLIB1,
647: C      &          KLIB2, XLIB2, NBANK, NUC, NMT, IRAND, JDEBS, IBPF,
648: C      &          IWK7, IWK8, IWK9, IWK10, IWK11, IWK12, WK1, WK5, R,
649: C      &          WK6, IWK15, JDLYN, JTIME )
650: C
651: C      do 330 I = 1, N
652: C          WK2(I) = MIN(WK2(I),ETOP)
653: C      continue
654: C
655: C      if ( ETHMAX.ge.EBOT ) then
656: C          do 340 I = 1, N
657: C              WK2(I) = MAX(WK2(I),EBOT)
658: C          continue
659: C          NFISB = NFISS
660: C          NN = N
661: C
662: C      ... ENERGY CUT OFF .....
663: C
664: C      else
665: C
666: C          NN = 0
667: C
668: C          if ( JMNTR.ne.0 ) then
669: C              do 350 I = 1, NFISS
670: C                  if ( WK2(I).lt.EBOT ) then
671: C                      IREGI = IREG(IWK6(I))
672: C                      NECUT(IREGI) = NECUT(IREGI) + 1

```

```

673: C                      WECUT(IREGI) = WECUT(IREGI) + WK3(I)
674: C                  end if
675: C              continue
676: C          end if
677: C *VOCL LOOP,NOVREC
678: C      do 360 I = 1, NFISS
679: C          WK3I = WK3(I)
680: C          if ( WK2(I).lt.EBOT ) then
681: C              NCNTR(8) = NCNTR(8) + 1
682: C              WCNTR(8) = WCNTR(8) + WK3I
683: C          else
684: C              NN = NN + 1
685: C              IWK6(NN) = IWK6(I)
686: C              WK2(NN) = WK2(I)
687: C              WK3(NN) = WK3I
688: C              if ( JTIME.ne.0 ) WK6(NN) = WK6(I) ! dn
689: C              if ( JDLYN.ne.0 ) IWK15(NN) = IWK15(I) ! dn
690: C          end if
691: C      continue
692: C
693: C      NFISB = NN
694: C
695: C      if ( JMNTR.ne.0 ) then
696: C          do 370 I = NFISS + 1, N
697: C              if ( WK2(I).lt.EBOT ) then
698: C
699: C          ... region# here is saved in parent/fission survival game
700: C
701: C              IREGI = IWK13(I)
702: C              NECUT(IREGI) = NECUT(IREGI) + 1
703: C              WECUT(IREGI) = WECUT(IREGI) + WWW(IWK6(I))
704: C              end if
705: C          continue
706: C      end if
707: C
708: C *VOCL LOOP,NOVREC
709: C      do 380 I = NFISS + 1, N
710: C          IWK6I = IWK6(I)
711: C          if ( WK2(I).lt.EBOT ) then
712: C              NCNTR(8) = NCNTR(8) + 1
713: C              WCNTR(8) = WCNTR(8) + WWW(IWK6I)
714: C              NDEAD = NDEAD + 1
715: C              LSDED(NDEAD) = IWK6I
716: C              if ( KIBNK(17).ne.0 ) IBNK(IWK6I,KIBNK(17)) = 0 ! photo-nuc
717: C              if ( KIBNK(18).ne.0 ) IBNK(IWK6I,KIBNK(18)) = 0 ! photo-nuc
718: C              if ( KIBNK(19).ne.0 ) IBNK(IWK6I,KIBNK(19)) = 0 ! photo-nuc
719: C              if ( JPHNU.ne.0 ) KPNFG(IWK6I) = 0 ! photo-nuc
720: C          else
721: C              NN = NN + 1
722: C              IWK6(NN) = IWK6I
723: C              WK2(NN) = WK2(I)
724: C              IWK1(NN) = IWK1(I) ! photo-nuc
725: C              IWK13(NN) = IWK13(I) ! photo-nuc
726: C              if ( JTIME.ne.0 ) WK6(NN) = WK6(I) ! dn
727: C              if ( JDLYN.ne.0 ) IWK15(NN) = IWK15(I) ! dn
728: C          end if
729: C      continue
730: C      end if
731: C
732: C      if ( NN.eq.0 ) return
733: C
734: C      NDEAD = NDEAD - NFISB
735: C      call RANU2( IRAND, R, NN*2, ICON )
736: C
737: C

```

src/mvp/fisgen.f

```

738: C      .... for newly created fission neutron
739: C      IWK7(I) : bank pointer of parent neutron.
740: C      IBPF(I) : bank pointer of generated neutron.
741: C
742: *VOCL LOOP,NOVREC
743:   do 390 I = 1, NFISB
744:     I0      = IWK6(I)
745:     IWK7(I) = LSCOL(I0)
746:     L1      = LSDED(NDEAD+I)
747:     IBPF(I) = L1
748: C
749:     W      = 1. - 2.*R(I)
750:     A      = SQRT(1.-W*W)
751:     PHI     = PI2*R(NN+I)
752:     U      = A*COS(PHI)
753:     V      = A*SIN(PHI)
754:     S      = 1./SQRT(U*U+V*V+W*W)
755:     AAA(IBPF(I)) = U*S
756:     BBB(IBPF(I)) = V*S
757:     CCC(IBPF(I)) = W*S
758:     XXX(IBPF(I)) = XXX(IWK7(I))
759:     YYY(IBPF(I)) = YYY(IWK7(I))
760:     ZZZ(IBPF(I)) = ZZZ(IWK7(I))
761:     WWW(IBPF(I)) = WK3(I)
762:     KKP(IBPF(I)) = KKP(IWK7(I))
763:     IZZ(IBPF(I)) = IZZ(IWK7(I))
764:     EEE(IBPF(I)) = WK2(I)
765:     KLSF(IBPF(I)) = 0
766: C
767: C---- SET MMAC(L1,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
768: C      FOR THIS PARTICLE
769: C
770:     MMAC(IBPF(I),2) = 0
771:     if ( JLATT.ne.0 ) LEVL(IBPF(I)) = LEVL(IWK7(I))
772:     if ( JIMPT.ne.0 ) XIM(IBPF(I)) = XIM(IWK7(I))
773:     if ( JTLLT.ne.0 ) IBREG(IBPF(I)) = IBREG(IWK7(I))
774:     IBREG(IBPF(I)) = IBREG(IWK7(I))
775:     if ( JTIME.ne.0 ) then
776:       if ( WK6(I).gt.0.0 ) then
777:         TTT(IBPF(I)) = TTT(IWK7(I)) + WK6(I) ! add the de
778:       if ( JPTIM.ne.0.and.TTT(IBPF(I)).gt.TCUT ) then ! treatment
779:         ITREP = TTT(IBPF(I)) / TCUT
780:         TTT(IBPF(I)) = TTT(IBPF(I)) - ITREP * TCUT
781:       end if
782:       call BS0ISD( TIMEB, NTIME+1, TTT(IBPF(I)), IT, 1 )
783:       ITT(IBPF(I)) = max(1, min(NTIME, IT))
784:     else
785:       TTT(IBPF(I)) = TTT(IWK7(I))
786:       ITT(IBPF(I)) = ITT(IWK7(I))
787:     end if
788:   else
789:     TTT(IBPF(I)) = TTT(IWK7(I))
790:   end if
791: C
792: c##<2007/03/14:PN3:
793:   if ( KPNPRD.eq.0 ) then
794:     if ( KIBNK(17).ne.0 ) then
795:       IBNK(IBPF(I),KIBNK(17)) = - IREG(I0) ! produce-region
796:     end if
797:     if ( KIBNK(18).ne.0 ) then
798:       IBNK(IBPF(I),KIBNK(18)) = INUC(I0) ! produce-nuclide
799:     end if
800:     if ( KIBNK(19).ne.0 ) then
801:       if ( JDLYN.ne.0.and.IWK15(I).eq.98 ) then
802:         IBNK(IBPF(I),KIBNK(19)) = 98 ! produce-reaction (de
803:       else
804:         IBNK(IBPF(I),KIBNK(19)) = 18 ! produce-reaction (pr
805:       end if
806:     end if
807: c##<2007/03/14:PN4:
808:     if ( JPHNU.ne.0 ) KPNFG(IBPF(I)) = 0
809: c##>
810:   else
811:     if ( KIBNK(17).ne.0 ) then
812:       IBNK(IBPF(I),KIBNK(17)) = IBNK(IWK7(I),KIBNK(17)) ! prod
813:     end if
814:     if ( KIBNK(18).ne.0 ) then
815:       IBNK(IBPF(I),KIBNK(18)) = IBNK(IWK7(I),KIBNK(18)) ! prod
816:     end if
817:     if ( KIBNK(19).ne.0 ) then
818:       IBNK(IBPF(I),KIBNK(19)) = IBNK(IWK7(I),KIBNK(19)) ! prod
819:     end if
820: c##<2007/03/14:PN4:
821:     if ( JPHNU.ne.0 ) KPNFG(IBPF(I)) = KPNFG(IWK7(I))
822: c##>
823:   end if
824: c##>
825:   390 continue
826: C
827:   if ( JLATT.ne.0 ) then
828:     do 410 K = 1, NEST
829:       *VOCL LOOP,NOVREC
830:       do 400 I = 1, NFISB
831:         LZZ(IBPF(I),K) = LZZ(IWK7(I),K)
832:         LPOS(IBPF(I),K) = LPOS(IWK7(I),K)
833:         if ( JHLAT.ne.0 ) LCRS(IBPF(I),K) = LCRS(IWK7(I),K)
834:         if ( JTLLT.ne.0 ) IBSPC(IBPF(I),K) = IBSPC(IWK7(I),K)
835:       400 continue
836:     410 continue
837:   end if
838: C
839: C
840: C ==== resample angle only for fission neutrons generated by killing
841: C      parent neutrons
842: C
843: *VOCL LOOP,NOVREC
844:   do 420 I = NFISB + 1, NN
845:     IBPF(I) = IWK6(I)
846: C
847:     W      = 1. - 2.*R(I)
848:     A      = SQRT(1.-W*W)
849:     PHI     = PI2*R(NN+I)
850:     U      = A*COS(PHI)
851:     V      = A*SIN(PHI)
852:     S      = 1./SQRT(U*U+V*V+W*W)
853:     AAA(IBPF(I)) = U*S
854:     BBB(IBPF(I)) = V*S
855:     CCC(IBPF(I)) = W*S
856:     EEE(IBPF(I)) = WK2(I)
857: C
858: C---- SET MMAC(L1,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
859: C      FOR THIS PARTICLE
860: C

```

src/mvp/fisgen.f

```

861:      MMAC(IBPF(I),2) = 0
862: c##<2007/03/14:PN3:
863: C
864:      if ( KPNPRD.eq.0 ) then
865:      if ( KIBNK(17).ne.0 ) then
866:      IBNK(IBPF(I),KIBNK(17)) = - IWK13(I) ! produce-region
867:      end if
868:      if ( KIBNK(18).ne.0 ) then
869:      IBNK(IBPF(I),KIBNK(18)) = IWK1(I) ! produce-nuclide
870:      end if
871:      if ( KIBNK(19).ne.0 ) then
872:      if ( JDLYN.ne.0.and.IWK15(I).eq.98 ) then
873:      IBNK(IBPF(I),KIBNK(19)) = 98 ! produce-reaction (de
layed neutron)
874:      else
875:      IBNK(IBPF(I),KIBNK(19)) = 18 ! produce-reaction (pr
ompt neutron)
876:      end if
877:      end if
878: c##<2007/03/14:PN4:
879:      if ( JPHNU.ne.0 ) KPNFG(IBPF(I)) = 0
880: c##>
881:      end if
882: c##>
883: 420 continue
884: C
885: C ... copy or change optional bank parameters
886: C
887: C IWK7(1:NFISB) : bank pointer of parent neutron.
888: C IBPF(1:NN) : bank pointer of generated neutron.
889: C
890: do 470 K = 1, KDBNK(0)
891: C
892: if ( K.eq.KDBNK(1) ) then
893: C
894: C ... birth weight is current weight !!!
895: C
896: *VOCL LOOP,NOVREC
897: do 430 I = 1, NN
898: DBNK(IBPF(I),KDBNK(1)) = WWW(IBPF(I))
899: 430 continue
900: C
901: C ... time of birth is just now !!!
902: C
903: else if ( K.eq.KDBNK(2) ) then
904: *VOCL LOOP,NOVREC
905: do 440 I = 1, NN
906: DBNK(IBPF(I),KDBNK(2)) = TTT(IBPF(I))
907: 440 continue
908: C
909: C ... energy on birth !!!
910: C
911: else if ( K.eq.KDBNK(3) ) then
912: *VOCL LOOP,NOVREC
913: do 450 I = 1, NN
914: DBNK(IBPF(I),KDBNK(3)) = EEE(IBPF(I))
915: 450 continue
916: C
917: C ... other optional bank parameters are only copy of those
918: C of parents
919: C or gadget for I=NFISB+1,NN .....(don't know how to set)
920: C
921: else
922: *VOCL LOOP,NOVREC
923: do 460 I = 1, NFISB

```

```

924: DBNK(IBPF(I),K) = DBNK(IWK7(I),K)
925: 460 continue
926: end if
927: 470 continue
928: C
929: do 520 K = 1, KIBNK(0)
930: C
931: C ... increment "particle generation" if necessary
932: C
933: if ( K.eq.KIBNK(4) ) then
934: *VOCL LOOP,NOVREC
935: do 480 I = 1, NFISB
936: IBNK(IBPF(I),KIBNK(4)) = IBNK(IWK7(I),KIBNK(4)) + 1
937: 480 continue
938: *VOCL LOOP,NOVREC
939: do 490 I = NFISB + 1, NN
940: IBNK(IBPF(I),KIBNK(4)) = IBNK(IBPF(I),KIBNK(4)) + 1
941: 490 continue
942: C
943: C ... reset flight path counter
944: C
945: else if ( K.eq.KIBNK(5) ) then
946: do 500 I = 1, NFISB
947: IBNK(IBPF(I),K) = 0
948: 500 continue
949: c##<2007/03/14:PN3:
950: C
951: else if ( K.eq.KIBNK(17) .or. K.eq.KIBNK(18) .or.
952: & K.eq.KIBNK(19) ) then
953: C --- stored in above ---
954: c##>
955: else
956: *VOCL LOOP,NOVREC
957: do 510 I = 1, NFISB
958: IBNK(IBPF(I),K) = IBNK(IWK7(I),K)
959: 510 continue
960: end if
961: 520 continue
962: C
963: C NFISB = NN
964: C
965: C
966: else
967: C
968: C=====
969: C Eigenvalue problem
970: C=====
971: C Fission neutrons are banked in fission bank
972: C (XXXF,YYYF,ZZZF,ZZZF.....)
973: C
974: C***** Take tally for keff (analog estimator) *****
975: C
976: do 530 I = 1, NCOLS
977: if ( WK2(I).gt.0. ) then
978: WCNTR(21) = WCNTR(21) + WK2(I)
979: NCNTR(21) = NCNTR(21) + 1
980: c##<2007/03/14:PN4:
981: if ( JPHNU.ne.0 ) then
982: if ( KPNFG(LSCOL(I)).gt.0 ) then
983: WCNTR(33) = WCNTR(33) + WK2(I)
984: end if
985: end if
986: c##>
987: end if
988: if ( JFMUL.eq.0 .and. JDLYN.ne.0 .and. WK6(I).gt.0. ) then

```

src/mvp/fisgen.f

```

989:          WCNTR(21) = WCNTR(21) + WK6(I)
990:          NCNTR(21) = NCNTR(21) + 1
991: c##<2007/03/14:PN4:
992:          if ( JPHNU.ne.0 ) then
993:            if ( KPNFG(LSCOL(I)).gt.0 ) then
994:              WCNTR(33) = WCNTR(33) + WK6(I)
995:            end if
996:          end if
997: c##>
998:          end if
999:          WCNTR(22) = WCNTR(22) + WK1(I)*
1000: &          SMIC(LSCOL(I),INUC(I),EMIC(8))
1001: c+beff1
1002:          if( JBEFF.ne.0 ) then
1003:            WCBEF(3) = WCBEF(3) + WK6(I)
1004:            WCBEF(6) = WCBEF(6) + (WK2(I)+WK6(I))
1005: &            *dble( IBNK(LSCOL(I),KIBNK(20)))
1006:            WCBEF(9) = WCBEF(9)+WK7(I)*dble( IBNK(LSCOL(I),KIBNK(20)))
1007:            WCBEF(12) = WCBEF(12)+WK7(I)
1008:          end if
1009: c-beff1
1010: 530      continue
1011: C
1012:          if ( NFISS.le.0 ) return
1013: C
1014: C***** Bank fission neutrons *****
1015: C
1016: C ... NCS : start point in collision stack
1017: C ... NCOLS : length of collision stack
1018: C
1019:          NCS = 1
1020:          ICNTR = 0 ! counter of 540 loop.
1021: 540      continue ! start loop
1022:          ICNTR = ICNTR + 1
1023: C
1024: C .....
1025: C
1026: C          IBPF(I) : position in fission bank
1027: C          IWK1(I) : colliding nuclide
1028: C          IWK6(I) : mother neutron
1029: C          NN : number of fission neutrons to be banked
1030: C .....
1031: C
1032:          if ( NFISS.le.NBANK ) then
1033:            NFISR = 0
1034:            MA = 0
1035:            do 550 I = NCS, NCOLS
1036:              MA = max(MA, IWK4(I), IWK14(I)) ! dn: IWK14
1037: 550          end do
1038:            NN = 0
1039:            do 570 K = 1, MA
1040:              do 560 I = NCS, NCOLS
1041:                if ( IWK4(I).ge.K ) then
1042:                  NN = NN + 1
1043:                  IWK6(NN) = LSCOL(I)
1044:                  IWK1(NN) = INUC(I)
1045:                  IWK15(NN) = 18
1046:                end if
1047:                if ( IWK14(I).ge.K ) then ! dn
1048:                  NN = NN + 1 ! dn
1049:                  IWK6(NN) = LSCOL(I) ! dn
1050:                  IWK1(NN) = INUC(I) ! dn
1051:                  IWK15(NN) = 98 ! dn
1052:                end if ! dn
1053: 560          end do

```

```

1054: 570      end do
1055: C
1056: C ... else : NFISS > NBANK
1057: C
1058:          else
1059:            NNN = 0
1060:            NN = 0
1061:            do 590 I = NCS, NCOLS
1062:              NNN = NNN + IWK4(I) + IWK14(I) ! dn:IWK14
1063:              ICS = I
1064:              if ( NNN.gt.NBANK ) go to 600
1065:              do 580 K = 1, IWK4(I)
1066:                NN = NN + 1
1067:                IWK6(NN) = LSCOL(I)
1068:                IWK1(NN) = INUC(I)
1069:                IWK15(NN) = 18
1070: 580          end do
1071:              do 581 K = 1, IWK14(I) ! dn
1072:                NN = NN + 1 ! dn
1073:                IWK6(NN) = LSCOL(I) ! dn
1074:                IWK1(NN) = INUC(I) ! dn
1075:                IWK15(NN) = 98 ! dn
1076: 581          end do
1077: 590          end do
1078: C
1079:            NFISR = 0
1080:            go to 610
1081: C
1082: 600          continue
1083: C
1084: C ... detect possible infinite loop
1085: C <<comment>> Y.Nagaya 2016/10/31
1086: C If the exact resonance scattering models (JEXDP=2, or =3) are used,
1087: C one collision point may yield many fission neutrons. Use of the
1088: C models invokes weight window automatically, but splitting cannot
1089: C be always applied for all neutron weights. A weight may become more
1090: C than 400. Then loop 540 may go into an infinite loop because
1091: C NNN > NBANK. This logic should be modified in future.
1092: C
1093:          if(ICNTR.gt.0.and.NN.eq.0) then
1094:            write(IOW,*) ' XXX(FISGEN) Cannot prepare bank data of '//
1095: &            'fission neutrons. Possibly in infinte loop. Stop. '//
1096: &            'Larger NBANK may circumvent this error.'
1097:            stop 999
1098:          end if
1099: C
1100:            NFISR = NFISS - NN
1101:            NFISS = NN
1102:            NCS = ICS
1103: C
1104: 610          continue
1105:          end if ! if NFISS <= NBANK else NFISS > NBANK
1106: C
1107:          if ( NFISB.ge.NFBANK ) then
1108:            call RANU2( IRAND, R, NFISS, ICON )
1109: C
1110:            NN = 0
1111: C
1112: *VOCL LOOP,NOVREC
1113:          do 620 I = 1, NFISS
1114: C/#IF ROUNDOFF(NEAREST)
1115:            NCN2 = NCNTR(2) + I
1116:            IP = MIN( INT(NCN2*R(I)) +1,NCN2)
1117: C/#ELSE
1118: *            IP = (NCNTR(2)+I)*R(I) + 1

```

src/mvp/fisgen.f

```

1119: C/#ENDIF
1120:         if ( IP.le.NFBANK ) then
1121:             XXXF(IP) = XXX(IWK6(I))
1122:             YYYY(IP) = YY(IWK6(I))
1123:             ZZZF(IP) = ZZZ(IWK6(I))
1124:             EEEF(IP) = EEE(IWK6(I))
1125:             IZZF(IP) = IZZ(IWK6(I))
1126:             INUF(IP) = IWK1(I)
1127:             IMTF(IP) = IWK15(I)
1128:             if ( JLATT.ne.0 ) LEVLF(IP) = LEVL(IWK6(I))
1129:             if ( JTLLT.ne.0 ) IBRGF(IP) = IBREG(IWK6(I))
1130:             IBRGF(IP) = IBREG(IWK6(I))
1131:             if ( JPERT.ne.0 .and. NBATCH.ge.NSKIP ) then
1132: c ... FISGEN1 ...
1133: c -----
1134: c DENSITY & NUMBER DENSITY
1135: c -----
1136:             if ( JPDEN.ne.0 ) then
1137:
1138: c --- differential operator sampling ---
1139:
1140:                 do 691 IPT = 1, NPTDS
1141: c ... density perturbation ...
1142:                     DELTA = dble(JPTRR(IBREG(IWK6(I)),IPT))
1143:                     & * dble(JPTRN(IWK1(I),IPT))
1144:
1145:                     WD0(IP,0,IPT,1) = DELTA + WWD(IWK6(I),IPT,1)
1146:                     & + WSD(IWK6(I),0,IPT,1)
1147:
1148:                     WD0(IP,1,IPT,1) = DELTA + WWD(IWK6(I),IPT,1)
1149:                     WD0(IP,1,IPT,2) =
1150:                     & (DELTA + WWD(IWK6(I),IPT,1))*2
1151:                     & - DELTA + WWD(IWK6(I),IPT,2)
1152: c ____ 3rd order ____
1153:
1154:                     CONT1 = WWD(IWK6(I),IPT,1) + DELTA
1155:                     D1CT1 = WWD(IWK6(I),IPT,2) - DELTA
1156:                     D2CT1 = WWD(IWK6(I),IPT,3) + DELTA*2.d0
1157:                     D3CT1 = WWD(IWK6(I),IPT,4) - DELTA*6.d0
1158:                     D4CT1 = WWD(IWK6(I),IPT,5) + DELTA*24.d0
1159:                     D5CT1 = WWD(IWK6(I),IPT,6) - DELTA*120.d0
1160:                     D6CT1 = WWD(IWK6(I),IPT,7) + DELTA*720.d0
1161:                     D7CT1 = WWD(IWK6(I),IPT,8) - DELTA*5040.d0
1162:
1163:                     CONT2 = D1CT1 + CONT1*CONT1
1164:                     D1CT2 = D2CT1 + 2.d0*CONT1*D1CT1
1165:                     D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
1166:                     & + 2.d0*CONT1*D2CT1
1167:                     D3CT2 = D4CT1 + D3CT1*CONT1
1168:                     & + 3.d0*D2CT1*D1CT1
1169:                     & + 3.d0*D1CT1*D2CT1
1170:                     & + CONT1*D3CT1
1171:                     D4CT2 = D5CT1 + D4CT1*CONT1
1172:                     & + 4.d0*D3CT1*D1CT1
1173:                     & + 6.d0*D2CT1*D2CT1
1174:                     & + 4.d0*D1CT1*D3CT1
1175:                     & + CONT1*D4CT1
1176:                     D5CT2 = D6CT1 + D5CT1*CONT1
1177:                     & + 5.d0*D4CT1*D1CT1
1178:                     & + 10.d0*D3CT1*D2CT1
1179:                     & + 10.d0*D2CT1*D3CT1
1180:                     & + 5.d0*D1CT1*D4CT1
1181:                     & + CONT1*D5CT1
1182:                     D6CT2 = D7CT1 + D6CT1*CONT1
1183:                     & + 6.d0*D5CT1*D1CT1
1184:                     & + 15.d0*D4CT1*D2CT1
1185:                     & + 20.d0*D3CT1*D3CT1
1186:                     & + 15.d0*D2CT1*D4CT1
1187:                     & + 6.d0*D1CT1*D5CT1
1188:                     & + CONT1*D6CT1
1189:
1190:                     CONT3 = D1CT2 + CONT2*CONT1
1191:                     D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
1192:                     D2CT3 = D3CT2 + D2CT2*CONT1
1193:                     & + 2.d0*D1CT2*D1CT1
1194:                     & + CONT2*D2CT1
1195:                     D3CT3 = D4CT2 + D3CT2*CONT1
1196:                     & + 3.d0*D2CT2*D1CT1
1197:                     & + 3.d0*D1CT2*D2CT1
1198:                     & + CONT2*D3CT1
1199:                     D4CT3 = D5CT2 + D4CT2*CONT1
1200:                     & + 4.d0*D3CT2*D1CT1
1201:                     & + 6.d0*D2CT2*D2CT1
1202:                     & + 4.d0*D1CT2*D3CT1
1203:                     & + CONT2*D4CT1
1204:                     D5CT3 = D6CT2 + D5CT2*CONT1
1205:                     & + 5.d0*D4CT2*D1CT1
1206:                     & + 10.d0*D3CT2*D2CT1
1207:                     & + 10.d0*D2CT2*D3CT1
1208:                     & + 5.d0*D1CT2*D4CT1
1209:                     & + CONT2*D5CT1
1210:
1211:                     CONT4 = D1CT3 + CONT3*CONT1
1212:                     D1CT4 = D2CT3 + D1CT3*CONT1 + CONT3*D1CT1
1213:                     D2CT4 = D3CT3 + D2CT3*CONT1
1214:                     & + 2.d0*D1CT3*D1CT1
1215:                     & + CONT3*D2CT1
1216:                     D3CT4 = D4CT3 + D3CT3*CONT1
1217:                     & + 3.d0*D2CT3*D1CT1
1218:                     & + 3.d0*D1CT3*D2CT1
1219:                     & + CONT3*D3CT1
1220:                     D4CT4 = D5CT3 + D4CT3*CONT1
1221:                     & + 4.d0*D3CT3*D1CT1
1222:                     & + 6.d0*D2CT3*D2CT1
1223:                     & + 4.d0*D1CT3*D3CT1
1224:                     & + CONT3*D4CT1
1225:
1226:                     CONT5 = D1CT4 + CONT4*CONT1
1227:                     D1CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
1228:                     D2CT5 = D3CT4 + D2CT4*CONT1
1229:                     & + 2.d0*D1CT4*D1CT1
1230:                     & + CONT4*D2CT1
1231:                     D3CT5 = D4CT4 + D3CT4*CONT1
1232:                     & + 3.d0*D2CT4*D1CT1
1233:                     & + 3.d0*D1CT4*D2CT1
1234:                     & + CONT4*D3CT1
1235:
1236:                     CONT6 = D1CT5 + CONT5*CONT1
1237:                     D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
1238:                     D2CT6 = D3CT5 + D2CT5*CONT1
1239:                     & + 2.d0*D1CT5*D1CT1
1240:                     & + CONT5*D2CT1
1241:
1242:                     CONT7 = D1CT6 + CONT6*CONT1
1243:                     D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1
1244:
1245:                     CONT8 = D1CT7 + CONT7*CONT1
1246:
1247:                     WD0(IP,1,IPT,3) = CONT3
1248:                     WD0(IP,1,IPT,4) = CONT4
1249:                     WD0(IP,1,IPT,5) = CONT5

```

src/mvp/fisgen.f

```

1249:      WD0(IP,1,IPT,6) = CONT6
1250:      WD0(IP,1,IPT,7) = CONT7
1251:      WD0(IP,1,IPT,8) = CONT8
1252:
1253:      do ISP = 2, NGSP
1254:          WD0(IP,ISP,IPT,1) =
1255:              & DELTA + WWD(IWK6(I),IPT,1)
1256:              & + WSD(IWK6(I),ISP-1,IPT,1)
1257:          WD0(IP,ISP,IPT,2) =
1258:              & (DELTA + WWD(IWK6(I),IPT,1))*2
1259:              & - DELTA + WWD(IWK6(I),IPT,2)
1260:              & + WSD(IWK6(I),ISP-1,IPT,2)
1261:              & + 2.d0*(DELTA + WWD(IWK6(I),IPT,1))
1262:              & *WSD(IWK6(I),ISP-1,IPT,1)
1263:      c --- 1st order ---
1264:          PS1 = WSD(IWK6(I),ISP-1,IPT,1)
1265:          PS2 = WSD(IWK6(I),ISP-1,IPT,2)
1266:              & + 2.d0*CONT1*PS1
1267:          PS3 = WSD(IWK6(I),ISP-1,IPT,3)
1268:              & + 3.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,2)
1269:              & + 3.d0*CONT2*PS1
1270:          PS4 = WSD(IWK6(I),ISP-1,IPT,4)
1271:              & + 4.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,3)
1272:              & + 6.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,2)
1273:              & + 4.d0*CONT3*PS1
1274:          PS5 = WSD(IWK6(I),ISP-1,IPT,5)
1275:              & + 5.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,4)
1276:              & + 10.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,3)
1277:              & + 10.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,2)
1278:              & + 5.d0*CONT4*PS1
1279:          PS6 = WSD(IWK6(I),ISP-1,IPT,6)
1280:              & + 6.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,5)
1281:              & + 15.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,4)
1282:              & + 20.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,3)
1283:              & + 15.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,2)
1284:              & + 6.d0*CONT5*PS1
1285:          PS7 = WSD(IWK6(I),ISP-1,IPT,7)
1286:              & + 7.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,6)
1287:              & + 21.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,5)
1288:              & + 35.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,4)
1289:              & + 35.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,3)
1290:              & + 21.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,2)
1291:              & + 7.d0*CONT6*PS1
1292:          PS8 = WSD(IWK6(I),ISP-1,IPT,8)
1293:              & + 8.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,7)
1294:              & + 28.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,6)
1295:              & + 56.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,5)
1296:              & + 70.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,4)
1297:              & + 56.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,3)
1298:              & + 28.d0*CONT6*WSD(IWK6(I),ISP-1,IPT,2)
1299:              & + 8.d0*CONT7*PS1
1300:          WD0(IP,ISP,IPT,3) = CONT3 + PS3
1301:          WD0(IP,ISP,IPT,4) = CONT4 + PS4
1302:          WD0(IP,ISP,IPT,5) = CONT5 + PS5
1303:          WD0(IP,ISP,IPT,6) = CONT6 + PS6
1304:          WD0(IP,ISP,IPT,7) = CONT7 + PS7
1305:          WD0(IP,ISP,IPT,8) = CONT8 + PS8
1306:      end do
1307:
1308:      691      continue
1309:
1310:      c --- correlated sampling ---
1311:
1312:      do 591 IPT = 1, NPTCS
1313:          DLT01 =
1314:              & dble(JPTTR(IBREG(IWK6(I)),NPTDS+IPT))
1315:              & DELA1 = dble(DELA(1,NPTDS+IPT))
1316:
1317:          WR0(IP,0,IPT) =
1318:              & ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1319:              & *( WSC(IWK6(I),0,IPT) + 1.0d0 )
1320:
1321:          WR0(IP,1,IPT) =
1322:              & ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1323:
1324:          do 592 ISP = 2, NGSP
1325:              WR0(IP,ISP,IPT) =
1326:                  & ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1327:                  & *( WSC(IWK6(I),ISP-1,IPT) + 1.0d0 )
1328:          continue
1329:
1330:          591      continue
1331:          end if
1332:      c+beff2
1333:          if ( NPTBE.gt.0 ) then
1334:              c      RRNU = RNU(IWK6(I),IWK1(I),3)
1335:              c      & /RNU(IWK6(I),IWK1(I),1)
1336:              RRNU = 1.d0
1337:          c ... correlated sampling
1338:              WB0(IP,0) = RRNU*WWB(IWK6(I))
1339:                  & *(WSB(IWK6(I),0) + 1.0d0)
1340:              WB0(IP,1) = RRNU*WWB(IWK6(I))
1341:              do ISP = 2, NGSP
1342:                  WB0(IP,ISP) = RRNU*WWB(IWK6(I))
1343:                      & *(WSB(IWK6(I),ISP-1) + 1.0d0)
1344:              end do
1345:
1346:          c ... differential operator sampling
1347:
1348:              WBD0(IP,1) = WWBD(IWK6(I))
1349:              WLD0(IP,1) = WWLD(IWK6(I))
1350:              do ISP = 2, NGSP
1351:                  WBD0(IP,ISP) = WWBD(IWK6(I))
1352:                      & + WSD(IWK6(I),ISP-1)
1353:                  WLD0(IP,ISP) = WWLD(IWK6(I))
1354:                      & + WSLD(IWK6(I),ISP-1)
1355:              end do
1356:          end if
1357:      c-beff2
1358:          end if
1359:      C
1360:          NN = NN + 1
1361:          IBPF(NN) = IP
1362:          IWK6(NN) = IWK6(I)
1363:          end if
1364:          620      continue
1365:
1366:          if ( JLAT.ne.0.and.NN.gt.0 ) then
1367:              do 640 K = 1, NEST
1368:                  *VOCL LOOP,NOVREC
1369:
1370:                  do 630 I = 1, NN
1371:                      LZZF(IBPF(I),K) = LZZ(IWK6(I),K)
1372:                      LPOSF(IBPF(I),K) = LPOS(IWK6(I),K)
1373:                      if ( JHLAT.ne.0 ) then
1374:                          LCRSF(IBPF(I),K) = LCRS(IWK6(I),K)
1375:                      end if
1376:                      if ( JTLT.ne.0 ) then
1377:                          IBSPF(IBPF(I),K) = IBSPC(IWK6(I),K)
1378:                      end if
1379:                  end if
1380:              continue
1381:          end if
1382:      end do
1383:  
```

src/mvp/fisgen.f

```

1379:      640          continue
1380:          end if
1381:      C
1382:      C ... bank distance to lattice frame ...
1383:      C if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
1384:      C   do 612 K = 0, NEST
1385:      C     do 611 I = 1, NN
1386:      C       DFBK(IBPF(I),KDFBK(4)+k) = DBNK(IWK6(I),KDBNK(4)+k)
1387:      C     continue
1388:      C   continue
1389:      C   end if
1390:      C if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
1391:      C   do 615 K = 0, NEST-1
1392:      C     do 614 I = 1, NN
1393:      C       IFBK(IBPF(I),KIFBK(6)+k) = IBNK(IWK6(I),KIBNK(6)+k)
1394:      C     continue
1395:      C   continue
1396:      C   end if
1397:      C
1398:      C ... bank STG sphere position ...
1399:      C
1400:      C if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
1401:      C   KF = KDFBK(5)
1402:      C   KB = KDBNK(5)
1403:      C   do 660 K = 1, NEST
1404:      C     do 650 I = 1, NN
1405:      C       DFBK(IBPF(I),KF) = DBNK(IWK6(I),KB)
1406:      C       DFBK(IBPF(I),KF+1) = DBNK(IWK6(I),KB+1)
1407:      C       DFBK(IBPF(I),KF+2) = DBNK(IWK6(I),KB+2)
1408:      C     continue
1409:      C     KF = KF + 3
1410:      C     KB = KB + 3
1411:      C   continue
1412:      C   end if
1413:      C
1414:      C ... else : NFISB < NFBANK
1415:      C
1416:      C   else
1417:      C     NTEMP = MIN(NFISS,NFBANK-NFISB)
1418:      C
1419:      C   do 670 I = 1, NTEMP
1420:      C     IBPF(I) = NFISB + I
1421:      C     XXXF(IBPF(I)) = XXX(IWK6(I))
1422:      C     YYYY(IBPF(I)) = YYY(IWK6(I))
1423:      C     ZZZF(IBPF(I)) = ZZZ(IWK6(I))
1424:      C     EEEF(IBPF(I)) = EEE(IWK6(I))
1425:      C     IZZF(IBPF(I)) = IZZ(IWK6(I))
1426:      C     INUF(IBPF(I)) = IWK1(I)
1427:      C     IMTF(IBPF(I)) = IWK15(I) ! dn
1428:      C
1429:      C   if ( JLATT.ne.0 ) then
1430:      C     LEVLF(IBPF(I)) = LEVL(IWK6(I))
1431:      C   end if
1432:      C   if ( JTLT.ne.0 ) then
1433:      C     IBRGF(IBPF(I)) = IBREG(IWK6(I))
1434:      C   end if
1435:      C   IBRGF(IBPF(I)) = IBREG(IWK6(I))
1436:      C
1437:      C   if ( JPERT.ne.0 .and. NBATCH.ge.NSKIP ) then
1438:      C ... FISGEN2 ...
1439:      C -----
1440:      C DENSITY & NUMBER DENSITY
1441:      C -----
1442:      C   if ( JPDEN.ne.0 ) then
1443:
1444:      C --- differential operator sampling ---
1445:
1446:      C   do 721 IPT = 1, NPTDS
1447:      C ... density perturbation ...
1448:      C     DELTA = dble(JPTTR(IBREG(IWK6(I)),IPT))
1449:      C     & * dble(JPTNU(IWK1(I),IPT))
1450:
1451:      C     WDO(IBPF(I),0,IPT,1) =
1452:      C     & DELTA + WWD(IWK6(I),IPT,1)
1453:      C     & + WSD(IWK6(I),0,IPT,1)
1454:
1455:      C     WDO(IBPF(I),1,IPT,1) =
1456:      C     & DELTA + WWD(IWK6(I),IPT,1)
1457:      C     WDO(IBPF(I),1,IPT,2) =
1458:      C     & (DELTA + WWD(IWK6(I),IPT,1))*2
1459:      C     & - DELTA + WWD(IWK6(I),IPT,2)
1460:      C --- 3rd order ---
1461:
1462:      C     CONT1 = WWD(IWK6(I),IPT,1) + DELTA
1463:      C     D1CT1 = WWD(IWK6(I),IPT,2) - DELTA
1464:      C     D2CT1 = WWD(IWK6(I),IPT,3) + DELTA*2.d0
1465:      C     D3CT1 = WWD(IWK6(I),IPT,4) - DELTA*6.d0
1466:      C     D4CT1 = WWD(IWK6(I),IPT,5) + DELTA*24.d0
1467:      C     D5CT1 = WWD(IWK6(I),IPT,6) - DELTA*120.d0
1468:      C     D6CT1 = WWD(IWK6(I),IPT,7) + DELTA*720.d0
1469:      C     D7CT1 = WWD(IWK6(I),IPT,8) - DELTA*5040.d0
1470:
1471:      C     CONT2 = D1CT1 + CONT1*CONT1
1472:      C     D1CT2 = D2CT1 + 2.d0*CONT1*D1CT1
1473:      C     D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
1474:      C     & + 2.d0*CONT1*D2CT1
1475:      C     D3CT2 = D4CT1 + D3CT1*CONT1
1476:      C     & + 3.d0*D2CT1*D1CT1
1477:      C     & + 3.d0*D1CT1*D2CT1
1478:      C     & + CONT1*D3CT1
1479:      C     D4CT2 = D5CT1 + D4CT1*CONT1
1480:      C     & + 4.d0*D3CT1*D1CT1
1481:      C     & + 6.d0*D2CT1*D2CT1
1482:      C     & + 4.d0*D1CT1*D3CT1
1483:      C     & + CONT1*D4CT1
1484:      C     D5CT2 = D6CT1 + D5CT1*CONT1
1485:      C     & + 5.d0*D4CT1*D1CT1
1486:      C     & +10.d0*D3CT1*D2CT1
1487:      C     & +10.d0*D2CT1*D3CT1
1488:      C     & + 5.d0*D1CT1*D4CT1
1489:      C     & + CONT1*D5CT1
1490:      C     D6CT2 = D7CT1 + D6CT1*CONT1
1491:      C     & + 6.d0*D5CT1*D1CT1
1492:      C     & +15.d0*D4CT1*D2CT1
1493:      C     & +20.d0*D3CT1*D3CT1
1494:      C     & +15.d0*D2CT1*D4CT1
1495:      C     & + 6.d0*D1CT1*D5CT1
1496:      C     & + CONT1*D6CT1
1497:
1498:      C     CONT3 = D1CT2 + CONT2*CONT1
1499:      C     D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
1500:      C     D2CT3 = D3CT2 + D2CT2*CONT1
1501:      C     & + 2.d0*D1CT2*D1CT1
1502:      C     & + CONT2*D2CT1
1503:      C     D3CT3 = D4CT2 + D3CT2*CONT1
1504:      C     & + 3.d0*D2CT2*D1CT1
1505:      C     & + 3.d0*D1CT2*D2CT1
1506:      C     & + CONT2*D3CT1
1507:      C     D4CT3 = D5CT2 + D4CT2*CONT1
1508:      C     & + 4.d0*D3CT2*D1CT1
1509:      C     & + 6.d0*D2CT2*D2CT1

```

src/mvp/fisgen.f

```

1509:      &          + 4.d0*D1CT2*D3CT1
1510:      &          + CONT2*D4CT1
1511:      D5CT3 = D6CT2 + D5CT2*CONT1
1512:      &          + 5.d0*D4CT2*D1CT1
1513:      &          +10.d0*D3CT2*D2CT1
1514:      &          +10.d0*D2CT2*D3CT1
1515:      &          + 5.d0*D1CT2*D4CT1
1516:      &          + CONT2*D5CT1
1517:
1518:      CONT4 = D1CT3 + CONT3*CONT1
1519:      D1CT4 = D2CT3 + D1CT3*CONT1 + CONT3*D1CT1
1520:      D2CT4 = D2CT3 + D2CT3*CONT1
1521:      &          + 2.d0*D1CT3*D1CT1
1522:      &          + CONT3*D2CT1
1523:      D3CT4 = D4CT3 + D3CT3*CONT1
1524:      &          + 3.d0*D2CT3*D1CT1
1525:      &          + 3.d0*D1CT3*D2CT1
1526:      &          + CONT3*D3CT1
1527:      D4CT4 = D5CT3 + D4CT3*CONT1
1528:      &          + 4.d0*D3CT3*D1CT1
1529:      &          + 6.d0*D2CT3*D2CT1
1530:      &          + 4.d0*D1CT3*D3CT1
1531:      &          + CONT3*D4CT1
1532:
1533:      CONT5 = D1CT4 + CONT4*CONT1
1534:      D1CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
1535:      D2CT5 = D3CT4 + D2CT4*CONT1
1536:      &          + 2.d0*D1CT4*D1CT1
1537:      &          + CONT4*D2CT1
1538:      D3CT5 = D4CT4 + D3CT4*CONT1
1539:      &          + 3.d0*D2CT4*D1CT1
1540:      &          + 3.d0*D1CT4*D2CT1
1541:      &          + CONT4*D3CT1
1542:
1543:      CONT6 = D1CT5 + CONT5*CONT1
1544:      D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
1545:      D2CT6 = D3CT5 + D2CT5*CONT1
1546:      &          + 2.d0*D1CT5*D1CT1
1547:      &          + CONT5*D2CT1
1548:
1549:      CONT7 = D1CT6 + CONT6*CONT1
1550:      D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1
1551:
1552:      CONT8 = D1CT7 + CONT7*CONT1
1553:
1554:      WD0(IBPF(I),1,IPT,3) = CONT3
1555:      WD0(IBPF(I),1,IPT,4) = CONT4
1556:      WD0(IBPF(I),1,IPT,5) = CONT5
1557:      WD0(IBPF(I),1,IPT,6) = CONT6
1558:      WD0(IBPF(I),1,IPT,7) = CONT7
1559:      WD0(IBPF(I),1,IPT,8) = CONT8
1560:
1561:      do ISP = 2, NGSP
1562:          WD0(IBPF(I),ISP,IPT,1) =
1563:      &          DELTA + WWD(IWK6(I),IPT,1)
1564:      &          + WSD(IWK6(I),ISP-1,IPT,1)
1565:          WD0(IBPF(I),ISP,IPT,2) =
1566:      &          (DELTA + WWD(IWK6(I),IPT,1))**2
1567:      &          - DELTA + WWD(IWK6(I),IPT,2)
1568:      &          + WSD(IWK6(I),ISP-1,IPT,2)
1569:      &          + 2.d0*(DELTA + WWD(IWK6(I),IPT,1))
1570:      &          *WSD(IWK6(I),ISP-1,IPT,1)
1571:      c --- 3rd order ---
1572:      PS1 = WSD(IWK6(I),ISP-1,IPT,1)
1573:      PS2 = WSD(IWK6(I),ISP-1,IPT,2)

```

```

1574:      &          + 2.d0*CONT1*PS1
1575:      PS3 = WSD(IWK6(I),ISP-1,IPT,3)
1576:      &          + 3.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,2)
1577:      &          + 3.d0*CONT2*PS1
1578:      PS4 = WSD(IWK6(I),ISP-1,IPT,4)
1579:      &          + 4.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,3)
1580:      &          + 6.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,2)
1581:      &          + 4.d0*CONT3*PS1
1582:      PS5 = WSD(IWK6(I),ISP-1,IPT,5)
1583:      &          + 5.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,4)
1584:      &          +10.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,3)
1585:      &          +10.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,2)
1586:      &          + 5.d0*CONT4*PS1
1587:      PS6 = WSD(IWK6(I),ISP-1,IPT,6)
1588:      &          + 6.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,5)
1589:      &          +15.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,4)
1590:      &          +20.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,3)
1591:      &          +15.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,2)
1592:      &          + 6.d0*CONT5*PS1
1593:      PS7 = WSD(IWK6(I),ISP-1,IPT,7)
1594:      &          + 7.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,6)
1595:      &          +21.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,5)
1596:      &          +35.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,4)
1597:      &          +35.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,3)
1598:      &          +21.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,2)
1599:      &          + 7.d0*CONT6*PS1
1600:      PS8 = WSD(IWK6(I),ISP-1,IPT,8)
1601:      &          + 8.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,7)
1602:      &          +28.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,6)
1603:      &          +56.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,5)
1604:      &          +70.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,4)
1605:      &          +56.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,3)
1606:      &          +28.d0*CONT6*WSD(IWK6(I),ISP-1,IPT,2)
1607:      &          + 8.d0*CONT7*PS1
1608:      WD0(IBPF(I),ISP,IPT,3) = CONT3 + PS3
1609:      WD0(IBPF(I),ISP,IPT,4) = CONT4 + PS4
1610:      WD0(IBPF(I),ISP,IPT,5) = CONT5 + PS5
1611:      WD0(IBPF(I),ISP,IPT,6) = CONT6 + PS6
1612:      WD0(IBPF(I),ISP,IPT,7) = CONT7 + PS7
1613:      WD0(IBPF(I),ISP,IPT,8) = CONT8 + PS8
1614:      end do
1615:
1616:      721          continue
1617:      c --- correlated sampling ---
1618:
1619:      do 621 IPT = 1, NPTCS
1620:          DLT01 =
1621:      &          dble(JPTTR(IBREG(IWK6(I)),NPTDS+IPT))
1622:          DELA1 = dble(DELA(1,NPTDS+IPT))
1623:
1624:          WR0(IBPF(I),0,IPT) =
1625:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1626:      &          *( WSC(IWK6(I),0,IPT) + 1.0d0 )
1627:
1628:          WR0(IBPF(I),1,IPT) =
1629:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1630:
1631:      do 622 ISP = 2, NGSP
1632:          WR0(IBPF(I),ISP,IPT) =
1633:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1634:      &          *( WSC(IWK6(I),ISP-1,IPT) + 1.0d0 )
1635:
1636:      622          continue
1637:
1638:      621          continue

```


src/mvp/fisgen.f

```

1639:                end if
1640: c+beff2
1641:                if ( NPTBE.gt.0 ) then
1642: c                    RRNU =  RNU(IWK6(I),IWK1(I),3)
1643: c                    &      /RNU(IWK6(I),IWK1(I),1)
1644:                RRNU = 1.d0
1645: c ... correlated sampling
1646:                WB0(IBPF(I),0) = RRNU*WVB(IWK6(I))
1647:                &      *(WSB(IWK6(I),0) + 1.0d0)
1648:                WB0(IBPF(I),1) = RRNU*WVB(IWK6(I))
1649:                do ISP = 2, NGSP
1650:                WB0(IBPF(I),ISP) = RRNU*WVB(IWK6(I))
1651:                &      *(WSB(IWK6(I),ISP-1) + 1.0d0)
1652:                end do
1653:
1654: c ... differential operator sampling
1655:                WBD0(IBPF(I),1) = WWBD(IWK6(I))
1656:                WLD0(IBPF(I),1) = WWLD(IWK6(I))
1657:                do ISP = 2, NGSP
1658:                WBD0(IBPF(I),ISP) = WWBD(IWK6(I))
1659:                &      + WSD(IWK6(I),ISP-1)
1660:                WLD0(IBPF(I),ISP) = WWLD(IWK6(I))
1661:                &      + WSLD(IWK6(I),ISP-1)
1662:                end do
1663:                end if
1664:
1665: c-beff2
1666:                end if
1667:
1668: 670                continue
1669:
1670:                if ( JLATT.ne.0 ) then
1671:                do 690 K = 1, NEST
1672:                do 680 I = 1, NTEMP
1673:                LZZF(IBPF(I),K) = LZZ(IWK6(I),K)
1674:                LPOSF(IBPF(I),K) = LPOS(IWK6(I),K)
1675:                if ( JHLAT.ne.0 ) then
1676:                LCRSF(IBPF(I),K) = LCRS(IWK6(I),K)
1677:                end if
1678:                if ( JTLLT.ne.0 ) then
1679:                IBSPF(IBPF(I),K) = IBSPC(IWK6(I),K)
1680:                end if
1681:                680                continue
1682:                690                continue
1683:                end if
1684: C
1685: C ... bank distance to lattice frame ...
1686: C                if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
1687: C                do 642 K = 0, NEST
1688: C                do 641 I = 1, NTEMP
1689: C                DFBK(IBPF(I),KDFBK(4)+k) = DBNK(IWK6(I),KDBNK(4)+k)
1690: C                continue
1691: C                642                continue
1692: C                end if
1693: C                if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
1694: C                do 645 K = 0, NEST-1
1695: C                do 644 I = 1, NTEMP
1696: C                IFBK(IBPF(I),KIFBK(6)+k) = IBNK(IWK6(I),KIBNK(6)+k)
1697: C                644                continue
1698: C                645                continue
1699: C                end if
1700: C
1701: C ... bank STG sphere position ...
1702: C
1703:                if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then

```

```

1704:                KF      = KDFBK(5)
1705:                KB      = KDBNK(5)
1706:                do 710 K = 1, NEST
1707:                do 700 I = 1, NTEMP
1708:                DFBK(IBPF(I),KF) = DBNK(IWK6(I),KB)
1709:                DFBK(IBPF(I),KF+1) = DBNK(IWK6(I),KB+1)
1710:                DFBK(IBPF(I),KF+2) = DBNK(IWK6(I),KB+2)
1711:                700                continue
1712:                KF      = KF + 3
1713:                KB      = KB + 3
1714:                710                continue
1715:                end if
1716: C
1717:                NFISB = NFISB + NTEMP
1718:                NN      = NTEMP
1719: C
1720:                if ( NTEMP.lt.NFISS ) then
1721:                call RANU2( IRAND, R, NFISS-NTEMP, ICON )
1722:                NTEMP1 = NTEMP + 1
1723:                *VOCL LOOP,NOVREC
1724:                do 720 I = NTEMP1, NFISS
1725:                C/#IF ROUNDOFF(NEAREST)
1726:                NFNI = NFBANK - NTEMP + I
1727:                IP = MIN(INT(NFNI*R(I-NTEMP))+1,NFNI)
1728:                C/#ELSE
1729:                *                IP = (NFBANK-NTEMP+I)*R(I-NTEMP) + 1
1730:                C/#ENDIF
1731:                if ( IP.le.NFBANK ) then
1732:                XXXF(IP) = XXX(IWK6(I))
1733:                YYYYF(IP) = YYY(IWK6(I))
1734:                ZZZF(IP) = ZZZ(IWK6(I))
1735:                EEEF(IP) = EEE(IWK6(I))
1736:                IZZF(IP) = IZZ(IWK6(I))
1737:                INUF(IP) = IWK1(I)
1738:                IMTF(IP) = IWK15(I)
1739: C
1740:                if ( JLATT.ne.0 ) LEVLF(IP) = LEVL(IWK6(I))
1741:                CCC                if ( JTLLT.ne.0 ) IBRGF(IP) = IBREG(IWK6(I))
1742:                IBRGF(IP) = IBREG(IWK6(I))
1743: C
1744:                if ( JPERT.ne.0 .and. NBATCH.ge.NSKIP ) then
1745:                c... FISGEN3 ...
1746:                c-----
1747:                c DENSITY & NUMBER DENSITY
1748:                c-----
1749:                if ( JPDEN.ne.0 ) then
1750:
1751:                c --- differential operator sampling ---
1752:
1753:                do 751 IPT = 1, NPTDS
1754:                c ... density perturbation ...
1755:                DELTA =
1756:                &      dble(JPTTR(IBREG(IWK6(I)),IPT))
1757:                &      * dble(JPTNU(IWK1(I),IPT))
1758:
1759:                WD0(IP,0,IPT,1) =
1760:                &      DELTA + WWD(IWK6(I),IPT,1)
1761:                &      + WSD(IWK6(I),0,IPT,1)
1762:
1763:                WD0(IP,1,IPT,1) =
1764:                &      DELTA + WWD(IWK6(I),IPT,1)
1765:                WD0(IP,1,IPT,2) =
1766:                &      (DELTA + WWD(IWK6(I),IPT,1))*2
1767:                &      - DELTA + WWD(IWK6(I),IPT,2)
1768:                c ___ 3rd order ___

```

```

D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
+ 2.d0*CONT1*D2CT1
D3CT2 = D4CT1 + D3CT1*CONT1
+ 3.d0*D2CT1*D1CT1
+ 3.d0*D1CT1*D2CT1
+ CONT1*D3CT1
D4CT2 = D5CT1 + D4CT1*CONT1
+ 4.d0*D3CT1*D1CT1
+ 6.d0*D2CT1*D2CT1
+ 4.d0*D1CT1*D3CT1
+ CONT1*D4CT1
D5CT2 = D6CT1 + D5CT1*CONT1
+ 5.d0*D4CT1*D1CT1
+10.d0*D3CT1*D2CT1
+10.d0*D2CT1*D3CT1
+ 5.d0*D1CT1*D4CT1
+ CONT1*D5CT1
D6CT2 = D7CT1 + D6CT1*CONT1
+ 6.d0*D5CT1*D1CT1
+15.d0*D4CT1*D2CT1
+20.d0*D3CT1*D3CT1
+15.d0*D2CT1*D4CT1
+ 6.d0*D1CT1*D5CT1
+ CONT1*D6CT1

CONT3 = D1CT2 + CONT2*CONT1
D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
D2CT3 = D3CT2 + D2CT2*CONT1
+ 2.d0*D1CT2*D1CT1
+ CONT2*D2CT1
D3CT3 = D4CT2 + D3CT2*CONT1
+ 3.d0*D2CT2*D1CT1
+ 3.d0*D1CT2*D2CT1
+ CONT2*D3CT1
D4CT3 = D5CT2 + D4CT2*CONT1
+ 4.d0*D3CT2*D1CT1
+ 6.d0*D2CT2*D2CT1
+ 4.d0*D1CT2*D3CT1
+ CONT2*D4CT1
D5CT3 = D6CT2 + D5CT2*CONT1
+ 5.d0*D4CT2*D1CT1
+10.d0*D3CT2*D2CT1
+10.d0*D2CT2*D3CT1
+ 5.d0*D1CT2*D4CT1
+ CONT2*D5CT1

```

1834:	&
1835:	
1836:	&
1837:	&
1838:	&
1839:	&
1840:	
1841:	
1842:	
1843:	
1844:	&
1845:	&
1846:	
1847:	&
1848:	&
1849:	&
1850:	
1851:	
1852:	
1853:	
1854:	&
1855:	&
1856:	
1857:	
1858:	
1859:	
1860:	
1861:	
1862:	
1863:	
1864:	
1865:	
1866:	
1867:	
1868:	
1869:	
1870:	
1871:	&
1872:	&
1873:	
1874:	&
1875:	&
1876:	&
1877:	&
1878:	&
1879:	c ____ 3rd order ____
1880:	
1881:	
1882:	&
1883:	
1884:	&
1885:	&
1886:	
1887:	&
1888:	&
1889:	&
1890:	
1891:	&
1892:	&
1893:	&
1894:	&
1895:	
1896:	&
1897:	&
1898:	&

```

+ CONT3*D3CT1
D4CT4 = D5CT3 + D4CT3*CONT1
+ 4.d0*D3CT3*D1CT1
+ 6.d0*D2CT3*D2CT1
+ 4.d0*D1CT3*D3CT1
+ CONT3*D4CT1

CONT5 = D1CT4 + CONT4*CONT1
D4CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
D2CT5 = D3CT4 + D2CT4*CONT1
+ 2.d0*D1CT4*D1CT1
+ CONT4*D2CT1
D3CT5 = D4CT4 + D3CT4*CONT1
+ 3.d0*D2CT4*D1CT1
+ 3.d0*D1CT4*D2CT1
+ CONT4*D3CT1

CONT6 = D1CT5 + CONT5*CONT1
D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
D2CT6 = D3CT5 + D2CT5*CONT1
+ 2.d0*D1CT5*D1CT1
+ CONT5*D2CT1

CONT7 = D1CT6 + CONT6*CONT1
D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1

CONT8 = D1CT7 + CONT7*CONT1

WD0(IP,1,IPT,3) = CONT3
WD0(IP,1,IPT,4) = CONT4
WD0(IP,1,IPT,5) = CONT5
WD0(IP,1,IPT,6) = CONT6
WD0(IP,1,IPT,7) = CONT7
WD0(IP,1,IPT,8) = CONT8

do ISP = 2, NGSP
  WD0(IP,ISP,IPT,1) =
    DELTA + WWD(IWK6(I),IPT,1)
    + WSD(IWK6(I),ISP-1,IPT,1)
  WD0(IP,ISP,IPT,2) =
    (DELTA + WWD(IWK6(I),IPT,1))**2
    - DELTA + WWD(IWK6(I),IPT,2)
    + WSD(IWK6(I),ISP-1,IPT,2)
    + 2.d0*(DELTA + WWD(IWK6(I),IPT,1))
    *WSD(IWK6(I),ISP-1,IPT,1)

  PS1 = WSD(IWK6(I),ISP-1,IPT,1)
  PS2 = WSD(IWK6(I),ISP-1,IPT,2)
    + 2.d0*CONT1*PS1
  PS3 = WSD(IWK6(I),ISP-1,IPT,3)
    + 3.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,2)
    + 3.d0*CONT2*PS1
  PS4 = WSD(IWK6(I),ISP-1,IPT,4)
    + 4.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,3)
    + 6.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,2)
    + 4.d0*CONT3*PS1
  PS5 = WSD(IWK6(I),ISP-1,IPT,5)
    + 5.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,4)
    +10.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,3)
    +10.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,2)
    + 5.d0*CONT4*PS1
  PS6 = WSD(IWK6(I),ISP-1,IPT,6)
    + 6.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,5)
    +15.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,4)
    +20.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,3)

```

src/mvp/fisgen.f

```

1899:      &          +15.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,2)
1900:      &          + 6.d0*CONT5*PS1
1901:      PS7 = WSD(IWK6(I),ISP-1,IPT,7)
1902:      &          + 7.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,6)
1903:      &          +21.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,5)
1904:      &          +35.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,4)
1905:      &          +35.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,3)
1906:      &          +21.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,2)
1907:      &          + 7.d0*CONT6*PS1
1908:      PS8 = WSD(IWK6(I),ISP-1,IPT,8)
1909:      &          + 8.d0*CONT1*WSD(IWK6(I),ISP-1,IPT,7)
1910:      &          +28.d0*CONT2*WSD(IWK6(I),ISP-1,IPT,6)
1911:      &          +56.d0*CONT3*WSD(IWK6(I),ISP-1,IPT,5)
1912:      &          +70.d0*CONT4*WSD(IWK6(I),ISP-1,IPT,4)
1913:      &          +56.d0*CONT5*WSD(IWK6(I),ISP-1,IPT,3)
1914:      &          +28.d0*CONT6*WSD(IWK6(I),ISP-1,IPT,2)
1915:      &          + 8.d0*CONT7*PS1
1916:      WDO(IP,ISP,IPT,3) = CONT3 + PS3
1917:      WDO(IP,ISP,IPT,4) = CONT4 + PS4
1918:      WDO(IP,ISP,IPT,5) = CONT5 + PS5
1919:      WDO(IP,ISP,IPT,6) = CONT6 + PS6
1920:      WDO(IP,ISP,IPT,7) = CONT7 + PS7
1921:      WDO(IP,ISP,IPT,8) = CONT8 + PS8
1922:      end do
1923:      continue
1924: 751
1925:
1926: c --- correlated sampling ---
1927:
1928:      do 651 IPT = 1, NPTCS
1929:          DLT01 =
1930:      &          dble(JPTTR(IBREG(IWK6(I)),NPTDS+IPT))
1931:          DELA1 = dble(DELA(1,NPTDS+IPT))
1932:
1933:          WR0(IP,0,IPT) =
1934:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1935:      &          *( WSC(IWK6(I),0,IPT) + 1.0d0 )
1936:
1937:          WR0(IP,1,IPT) =
1938:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1939:
1940:      do 652 ISP = 2, NGSP
1941:          WR0(IP,ISP,IPT) =
1942:      &          ( 1.d0 + DELA1*DLT01 )*WWR(IWK6(I),IPT)
1943:      &          *( WSC(IWK6(I),ISP-1,IPT) + 1.0d0 )
1944: 652      continue
1945:
1946: 651      continue
1947:      end if
1948: c+beff2
1949:      if ( NPTBE.gt.0 ) then
1950:  c          RRNU = RNU(IWK6(I),IWK1(I),3)
1951:  c      &          /RNU(IWK6(I),IWK1(I),1)
1952:          RRNU = 1.d0
1953:  c ... correlated sampling
1954:          WB0(IP,0) = RRNU*WVB(IWK6(I))
1955:      &          *(WSB(IWK6(I),0) + 1.0d0)
1956:          WB0(IP,1) = RRNU*WVB(IWK6(I))
1957:      do ISP = 2, NGSP
1958:          WB0(IP,ISP) = RRNU*WVB(IWK6(I))
1959:      &          *(WSB(IWK6(I),ISP-1) + 1.0d0)
1960:      end do
1961:
1962: c ... differential operator sampling
1963:

```

```

1964:          WBD0(IP,1) = WWBD(IWK6(I))
1965:          WLD0(IP,1) = WWLD(IWK6(I))
1966:      do ISP = 2, NGSP
1967:          WBD0(IP,ISP) = WWBD(IWK6(I))
1968:      &          + WSD(IWK6(I),ISP-1)
1969:          WLD0(IP,ISP) = WWLD(IWK6(I))
1970:      &          + WSLD(IWK6(I),ISP-1)
1971:      end do
1972:      end if
1973: c-beff2
1974:      end if
1975:
1976:          NN = NN + 1
1977:          IBPF(NN) = IP
1978:          IWK6(NN) = IWK6(I)
1979:      end if
1980: 720      continue
1981:      if ( JLATT.ne.0 ) then
1982:          do 740 K = 1, NEST
1983:      *VOCL LOOP,NOVREC
1984:          do 730 I = NTEMPL, NN
1985:              LZZF(IBPF(I),K) = LZZ(IWK6(I),K)
1986:              LPOSF(IBPF(I),K) = LPOS(IWK6(I),K)
1987:              if ( JHLAT.ne.0 ) then
1988:                  LCRSF(IBPF(I),K) = LCRS(IWK6(I),K)
1989:              end if
1990:              if ( JTLLT.ne.0 ) then
1991:                  IBSPF(IBPF(I),K) = IBSPC(IWK6(I),K)
1992:              end if
1993: 730          continue
1994: 740          continue
1995:      end if
1996:  c
1997:  c ... bank distance to lattice frame ...
1998:  c      if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
1999:  c          do 672 K = 0, NEST
2000:  c              do 671 I = NTEMPL, NN
2001:  c                  DFBK(IBPF(I),KDFBK(4)+k)=
2002:  c      &                  DBNK(IWK6(I),KDBNK(4)+k)
2003:  c      C 671          continue
2004:  c      C 672          continue
2005:  c      end if
2006:  c      if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
2007:  c          do 675 K = 0, NEST-1
2008:  c              do 674 I = NTEMPL, NN
2009:  c                  IFBK(IBPF(I),KIFBK(6)+k) =
2010:  c      &                  IBNK(IWK6(I),KIBNK(6)+k)
2011:  c      C 674          continue
2012:  c      C 675          continue
2013:  c      end if
2014:  c
2015:  c ... bank STG sphere position ...
2016:  c
2017:      if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
2018:          KF = KDFBK(5)
2019:          KB = KDBNK(5)
2020:          do 760 K = 1, NEST
2021:              do 750 I = NTEMPL, NN
2022:                  DFBK(IBPF(I),KF) = DBNK(IWK6(I),KB)
2023:                  DFBK(IBPF(I),KF+1) = DBNK(IWK6(I),KB+1)
2024:                  DFBK(IBPF(I),KF+2) = DBNK(IWK6(I),KB+2)
2025: 750          continue
2026:          KF = KF + 3
2027:          KB = KB + 3
2028: 760          continue

```

src/mvp/fisgen.f

```
2029:                end if
2030:            end if
2031:        end if
2032: C
2033: C**** No. of total fission neutron generated in this batch
2034: C
2035:         NCNTR(2)    = NCNTR(2) + NFISS
2036: C         WCNTR( 2) = WCNTR( 2) + W
2037:         if ( NFISR.gt.0 ) then
2038:             NFISS    = NFISR
2039:             go to 540      ! return to loop top
2040:         end if
2041: C
2042:     end if
2043: C
2044:     return
2045: end
```

src/mvp/fismin.f

```

1:      subroutine FISMIN( IO,      LIFISM,NSOUR, IA,      LAST, NUCID, NUC,
2:      &                  JDEBG )
3: C=====
4: C PURPOSE: INPUT 'IFISM' PARAMETER BY NUCLIDE SYMBOL.
5: C CALLED IN: INTRO
6: C=====
7:      integer IA(*)
8: C
9:      character*16 NUCID(NUC), NAME
10:     integer JDEBG(*)
11: C
12:     real*8 DENTAK
13: C
14:     include '../shared/INC/IOUNIT'
15: C
16: C-----
17: C
18:     if ( NSOUR.eq.0 .or. NUC.eq.0 ) then
19:       write(IMG,'(/lx,a,a)') 'XXX "IFISM" DATA ENCOUNTERED BEFORE ',
20:       & 'CROSS SECTION INPUT OR "NSOUR" DETERMINATION.'
21:       call CNTERR( 'FATAL' )
22:     end if
23: C
24:     if ( LIFISM.eq.0 ) then
25:       call KEEP( 'IFISM', LIFISM, NSOUR, 14, LAST, JDEBG )
26:     end if
27: C
28:     ISOUR = 0
29: C
30:     100 call CHREAD( 'IFISM', NAME, ' ', LNM, ITERM, IEND )
31:     if ( IEND.ne.0 ) then
32:       write(IMG,'(/lx,a)') 'XXX NO DATA FOR "IFISM".'
33:       call PRSTOP( 1, 'UNEXPECTED END OF DATA' )
34:       stop 888
35:     end if
36: C
37:     if ( LNM.ne.0 ) then
38:       if ( ISOUR.lt.NSOUR ) then
39:         IA(LIFISM+ISOUR) = 0
40:         IN = 0
41:         do 110 IIM = 1, NUC
42: Ccccccccccc if ( NAME(:LNM).eq.NUCID(IN) ) go to 120
43:         if ( IMATCH(NAME(:LNM),NUCID(IIM)).eq.1 ) then
44:           if ( IN.eq.0 ) then
45:             IN = IIM
46:           else
47:             L1 = ICLEN(NUCID(IN))
48:             L2 = ICLEN(NUCID(IIM))
49:             write(IO,'(/lx,a,i3,5a)')
50:             & '<<MESSAGE>> "IFISM" of source ', ISOUR,
51:             & ' matches more than one nuclides. <',
52:             & NUCID(IN) (:L1), '> (selected) and <',
53:             & NUCID(IIM) (:L2), '>'
54:             call CNTERR( 'MESSAGE' )
55:           end if
56:         end if
57:       110 continue
58: C
59:       if ( IN.eq.0 ) then
60:         write(IMG,'(3x,a,a/3x,a,a)')
61:         & '!!! Your input for IFISM may be in the old',
62:         & '-fashioned style using nuclide sequential # or',
63:         & ' specifying non-existent nuclide-ID pattern.',
64:         & ' Please use correct nuclide-ID pattern next time!!'
65:         call CNTERR( 'WARNING' )

```

```

66: C
67: C .... may be old form (by nuclide # ) try to read as an integer data
68: C
69:       IN = 0
70:       II = DENTAK('&&SILENT ON',IO,IEE0)
71:       IN = DENTAK(NAME(:LNM),IO,IEEE)
72:       II = DENTAK('&&SILENT OFF',IO,IEEE)
73: C
74:       if ( IEEE.ne.0 ) then
75:         write(IMG,'(/lx,a,a,a)')
76:         & 'XXX Non existent nuclide name or matching pattern <',
77:         & NAME(:LNM), '> is given as "IFISM" data.'
78:         call CNTERR( 'FATAL' )
79:         go to 120
80:       else
81:         write(IMG,'(/lx,a,i3,a,a,a)') '!!! IFISM may be ',IN,
82:         & ' <', NAME(:LNM), '>'
83:         call CNTERR( 'WARNING' )
84: C
85:         if ( IN.le.0 .or. IN.gt.NUC ) then
86:           write(IMG,'(/lx,a,i5,a)')
87:           & 'XXX "IFISM" input is out of range =', IN,
88:           & ' (must be 0<IFISM<=no. of nuclides)'
89:           call CNTERR( 'FATAL' )
90:         end if
91:       end if
92:     end if
93: C
94: C
95:     IA(LIFISM+ISOUR) = IN
96:     120 ISOUR = ISOUR + 1
97:   else
98:     write(IMG,'(/lx,a,i4,a)') '!!! TOO MANY "IFISM" DATA (',
99:     & NSOUR, ' DATA REQUIRED)'
100:     call CNTERR( 'WARNING' )
101:   end if
102: end if
103: C
104: if ( ITERM.eq.0 ) go to 100
105: C
106: return
107: end

```

src/mvp/fisset.f

```

1:      subroutine FISSSET( IOW,  IFISB, WFFACT,
2:      &      IRAND, NFISB, NHIST, NHIST1, NHSUB, NBANK,
3:      &      NZONE, NFBANK, NFBNK0, NBATCH, NREG,  NEST,
4:      &      WGTF, KZREG,
5:      &      XSOC, NLENG,
6:      &      XXXF, YYF, ZZZF, EEEF, INUF, IZZF,
7:      &      LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF, KLSFF,
8:      &      DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK,
9:      &      XXXF2, YYF2, ZZZF2, EEEF2, INUF2, IZZF2,
10:     &      LEVLF2, LZZF2, LPOSF2, LCRSF2,
11:     &      IBRGF2, IBSPF2, KLSFF2,
12:     &      DFBK2, KDFBK2, MDFBK2, IFBK2, KIFBK2, MIFBK2,
13:     &      RWF )
14: C=<MVP>=====
15: C PURPOSE:  selection of fission source before each batch of eigenvalue
16: C calculation mode.
17: C when a batch needs more than one sub-batches
18: C (NHSUB < NHIST) make copies of fission sources.
19: C CALLED IN: SOURCE
20: C CALLS      : (none)
21: C-----
22: C i NHIST1      : current batch size (NHIST or less than it)
23: C o IFISB(NHIST1) : fission bank index selected for current batch.
24: C      If points XXXF2(*) ,YYYF2(*) ,... when NHSUB < NHIST1
25: C      else points XXXF(*), YYF(*), ...
26: C o WFFACT : factor to normalize fission weight.
27: C w RWF(NHIST) : working array
28: C=====
29:      implicit real*8(A-H,O-Z)
30:
31:      include 'INC/_FLAGS'
32:
33:      integer IFISB(NHIST)
34:      real*8 WFFACT
35:
36: C .... FISSION BANK .....
37:
38:      real*8 XXXF(NFBNK0), YYF(NFBNK0), ZZZF(NFBNK0)
39:      real EEEF(NFBNK0)
40:      integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
41:      &      LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST), INUF(NFBNK0),
42:      &      KLSFF(NFBNK0)
43:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
44:
45: C
46: C ... optional bank parameters
47:
48:      real*8 DFBK(NFBNK0,*)
49:      integer KDFBK(0:MDFBK)
50:      integer IFBK(NFBNK0,*)
51:      integer KIFBK(0:MIFBK)
52:
53: C ... fission weight copy
54:
55: c##<2007/03/14:PN3:
56: c##      real WGTF(NREG)
57:      real WGTF(NREG,2)
58: c##>
59:
60: C ... fission bank copy
61:
62:      real*8 XXXF2(NHIST), YYF2(NHIST), ZZZF2(NHIST)
63:      real EEEF2(NHIST)
64:      integer IZZF2(NHIST), LEVLF2(NHIST), LZZF2(NHIST,NEST),
65:      &      LPOSF2(NHIST,NEST), LCRSF2(NHIST,NEST), INUF2(NHIST),

```

```

66:      &      KLSFF2(NHIST)
67:      integer IBRGF2(NHIST), IBSPF2(NHIST,NEST)
68: C
69:      real*8 DFBK2(NHIST,*)
70:      integer KDFBK2(0:MDFBK2)
71:      integer IFBK2(NHIST,*)
72:      integer KIFBK2(0:MIFBK2)
73: C
74: C .... TALLY .....
75: C
76:      real*8 XSOC(NLENG)
77: C
78: C .... GEOMETRY ARRAY DATA
79: C
80:      integer KZREG(NZONE)
81: C
82: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 7*NBANK FOR USE
83: C      IN sef5n2 )
84: C
85:      real RWF(NHIST)
86: C
87: C-----
88: C
89:      if ( NBATCH.gt.0.and.NFISB.le.0 ) then
90:          write(IOW,7000)
91:          7000 format(/1X,' !!! No fission source has been generated. ',
92:          &      ' STOP calculation !!'/1X,
93:          &      ' Check values for "WGTF" to generate more fission',
94:          &      ' neutrons, or other problems in your input. ')
95:          stop 888
96:      end if
97: C
98:      if ( NFISB.le.NHIST1 ) then
99:          do 100 I = 1, NFISB
100:             IFISB(I) = I
101:             continue
102:             if ( NFISB.lt.NHIST1 ) then
103:                 call RANU2( IRAND, RWF, NHIST1-NFISB, ICON )
104:                 do 110 I = 1, NHIST1 - NFISB
105: C/#IF ROUNDOFF(NEAREST)
106:                     IP = MIN(INT(NFISB*RWF(I))+1,NFISB)
107: C/#ELSE
108:                     IP = NFISB*RWF(I) + 1
109: C/#ENDIF
110:                     IFISB(NFISB+I) = IP
111:             110 continue
112:             end if
113:             else
114:                 call RANU2( IRAND, RWF, NHIST1, ICON )
115:                 do 120 I = 1, NHIST1
116: C/#IF ROUNDOFF(NEAREST)
117:                     IP = MIN(INT(NFISB*RWF(I))+1,NFISB)
118: C/#ELSE
119:                     IP = NFISB*RWF(I) + 1
120: C/#ENDIF
121:                     IFISB(I) = IP
122:             120 continue
123:             end if
124: C
125: C ..... Weight normalization factor ...
126: C
127:      WSUMF = 0.0
128:      *VOCL LOOP,NOVREC
129:      do 130 J = 1, NHIST1
130: c##<2007/03/14:PN3:

```

src/mvp/fisset.f

```
131: c##      WSUMF = WSUMF + WGTF(IBRGF(IFISB(J)))
132:      WSUMF = WSUMF + WGTF(IBRGF(IFISB(J)),1)      ! only prompt ne
utron
133: c##>
134:      130 continue
135: C
136:      WSUMF2 = DBLE(NHIST1)
137:      WFFACT = WSUMF2/WSUMF
138: C
139: C .... Update sum of weights .....
140: C
141:      XSOC(NBATCH+1) = 0.0D0
142: C
143: C ... copy selected fission bank data
144: C
145:      if ( NHSNB.lt.NHIST1 ) then
146:      do 140 I=1,NHIST1
147:          XXXF2(I) = XXXF(IFISB(I))
148:          YYF2(I) = YYF(IFISB(I))
149:          ZZF2(I) = ZZF(IFISB(I))
150:          IZZF2(I) = IZZF(IFISB(I))
151:          KLSFF2(I) = KLSFF(IFISB(I))
152:          EEEF2(I) = EEEF(IFISB(I))
153:          INUF2(I) = INUF(IFISB(I))
154:          IBRGF2(I) = IBRGF(IFISB(I))
155:          if ( JLATT.ne.0 ) then
156:              LEVLF2(I) = LEVLF(IFISB(I))
157:          end if
158:      140 continue
159: C
160:      if ( JLATT.ne.0 ) then
161:      do 160 K=1,NEST
162:          do 150 I=1,NHIST1
163:              LZZF2(I,K) = LZZF(IFISB(I),K)
164:              LPOSF2(I,K) = LPOSF(IFISB(I),K)
165:              if ( JHLAT.ne.0 ) then
166:                  LCRSF2(I,K) = LCRSF(IFISB(I),K)
167:              end if
168:              if ( JTLT.ne.0 ) then
169:                  IBSPF2(I,K) = IBSPF(IFISB(I),K)
170:              end if
171:          150 continue
172:      160 continue
173:      end if
174:      do 180 K=1,KDFBK(0)
175:          do 170 I=1,NHIST1
176:              DFBK2(I,K) = DFBK(IFISB(I),K)
177:          170 continue
178:      180 continue
179:      do 200 K=1,KIFBK(0)
180:          do 190 I=1,NHIST1
181:              IFBK2(I,K) = IFBK(IFISB(I),K)
182:          190 continue
183:      200 continue
184:      do 210 I=1,NHIST1
185:          IFISB(I) = I
186:      210 continue
187:      end if
188: C
189: C ... clear fission bank entry number for current batch ...
190: C
191:      NFISB = 0
192: C
193:      return
194:      end
```

src/mvp/fissrc.f

```

1:      subroutine FISSRC( IOW, IRAND, NFISB, NGENE, NFB0,WFFACT,IFISB,
2:      &      NFBNK0, IBP,
3:      &      NHIST, NBANK, NUC,      NZONE, NFBANK,
4:      &      NBATCH,NGROUP,NGP1, NREG, NEST,
5:      &      NLENG, WSUM, KZREG, KZMAT, LTYF, WGTF, EBOT, ETOP,
6:      &      ENGYB, XIMP, XSOC, NCNTR, WCNTR, NEVENT,CX,ICX,MCX,
7:      &      NMT, KLIB1, KLIB1, KLIB2, XLIB2,
8:      &      NFFL, LSFFL, IZFFL, NDEAD, LSDED,
9:      &      XXX, YYY, ZZZ, AAA, BBB, CCC, EEE,
10:     &      WWW, IGG, IZZ, TTT, XIM, IBREG, IBSPC, KLSF,
11:     &      LEVL, LZZ, LPOS, LCRS,
12:     &      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
13:     &      XXXF, YYYF, ZZZF, EEEF, INUF, IZZF,
14:     &      LEVLF, LZZF, LPOSF,
15:     &      LCRSF, IBRGF, IBSPF, KLSFF,
16:     &      DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK,
17:     &      R, IWK2,
18:     &      IWK3, IWK4, IWK5, IWK6, IWK7, RWK1, EW1,
19:     &      IMTF, RWK2, IWK8)
20: C=<MVP>=====
21: C PURPOSE: Set energy and other parameters necessary to start
22: C random walk for fission source of eigenvalue problem.
23: C this routine may be called more than once for a batch (probably
24: C before each sub-batch).
25: C
26: C (formerly working as "Preparation of fission source before
27: C each batch")
28: C
29: C Selection of fission sources from fission neutron bank is moved
30: C to FISSET routine wich should be called at the beginnig of each
31: C batch .
32: C
33: C CALLED IN: SOURCE
34: C CALLS : SEF5N2
35: C HISTORY: EXTRACTED FROM 'SOURCE' ROUTINE. ( FEB 1992 )
36: C-----
37: C arguments:
38: C i NGENE : number of fission sources set in this routine.
39: C
40: C i IFISB(*) : index pointer to fission neutron bank selected for
41: C fission source of current batch.
42: C More than one elemnts of this array may be pointing
43: C the same fission bank elements.(take care for vectorization)
44: C
45: C i NFB0 : use fission bank of indecies IFISB(NFB0+1:NFB0+NGENE).
46: C
47: C i WFFACT : normalization factor of fission source weight.
48: C i NFBNK0: fission bank array length
49: C i XXXF,YYYF,ZZZF,EEEF,INUF,IZZF,LEVLF,LZZF,LPOSF,LCRSF,IBRGF,IBSPF,
50: C KLSFF, DFBK,KDFBK,IFBK,KIFBK :
51: C Fission neutron bank arrays.
52: C
53: C <<Caution>>
54: C
55: C When NHSUB < NHIST, which means more than one sub-batch are
56: C necessary for a batch, another set of fission bank arrays
57: C XXXF2, YYYF2, .... are passed as XXXF, YYYF, ... and
58: C array size NFBNK0 is not the global data for XXXF, YYYF,...
59: C but that for XXXF2, YYYF2 (must be same as NHIST).
60: C
61: C o IBP(*) : bank pointer of generated source.
62: C o WSUM : source weight sum
63: C o XSOC : source weight sum of current batch id XSOC(NBATCH+1)
64: C o NCNTR(1,1), WCNTR(1,1) : total number and weight of fission source
65: C

```

```

66: C o NFFL, IZFFL, LSFFL : free flight particle stack.
67: C-----
68: C=====
69:     implicit real*8(A-H,O-Q,S-Z)
70:     implicit real(R)
71: c
72:     parameter( PI2 = 2*3.141592653589793238D0 )
73: C
74:     include 'INC/_FLAGS'
75: C
76:     real*8 WFFACT
77: C
78:     integer IFISB(NHIST)
79: C
80: C .... bank pointers returned ...
81: C
82:     integer IBP(NGENE)
83: C
84: C .... PARTICLE BANK .....
85: C
86:     real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
87:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
88:     real WWW(NBANK), EEE(NBANK), XIM(NBANK)
89:     real*8 TTT(NBANK)
90: c##<2007/03/14:PN3:
91: c##     real WGTF(NREG)
92:     real WGTF(NREG,2)
93: c##>
94:     integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
95:     &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
96:     integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
97: C
98: C ... optional bank parameters
99: C
100:     real*8 DBNK(NBANK,*)
101:     integer KDBNK(0:MDBNK)
102:     integer IBNK(NBANK,*)
103:     integer KIBNK(0:MIBNK)
104: C
105: C .... FISSION BANK .....
106: C
107:     real*8 XXXF(NFBNK0), YYYF(NFBNK0), ZZZF(NFBNK0)
108:     real EEEF(NFBNK0)
109:     integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
110:     &      LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST), INUF(NFBNK0),
111:     &      KLSFF(NFBNK0)
112:     integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
113:     integer IMTF(NFBNK0)
114: C
115: C ... optional bank parameters
116: C
117:     real*8 DFBK(NFBNK0,*)
118:     integer KDFBK(0:MDFBK)
119:     integer IFBK(NFBNK0,*)
120:     integer KIFBK(0:MIFBK)
121: C
122: C .... TALLY .....
123: C
124:     real*8 WSUM, XSOC(NLENG), WCNTR(NEVENT,2), NCNTR(NEVENT,2)
125: C
126: C .... VARIANCE REDUCTION DATA .....
127: C
128:     real XIMP(NGROUP,NREG)
129: C
130: C .... STACKS ...(FREE FLIGHT & MORGUE) .....

```


src/mvp/fissrc.f

```

131: C
132:   integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSDED(NBANK)
133: C
134: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 7*NBANK FOR USE
135: C       IN sef5n2 )
136: C
137:   real R(7*NBANK), EWL(NBANK), RWK1(NBANK)
138:   integer IWK2(NBANK), IWK3(NBANK), IWK4(NBANK), IWK5(NBANK),
139:   &       IWK6(NBANK), IWK7(NBANK)
140:   real   RWK2(NBANK)
141:   integer IWK8(NBANK)
142: C
143: C .... ENERGY BOUNDARY .....
144: C
145:   real ENGYB(NGP1+1)
146: C
147: C .... GEOMETRY ARRAY DATA
148: C
149:   integer KZREG(NZONE)
150:   integer KZMAT(NZONE*2)
151:   integer LTYP(*)
152: C
153: C CROSS SECTION ..... (NEUTRON)
154: C
155:   real EBOT, ETOP
156:   real CX(MCX)
157:   integer ICX(MCX)
158:   real XLIB1(NUC,11), XLIB2(NUC,NMT,4)
159:   integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
160: C
161:   include '../shared/INC/_PMLATT'
162:   include '../shared/INC/_SFLATT'
163: C
164: C-----
165: C .... SOURCE IS TAKEN FROM FISSION BANK (JEIGN.EQ.1.AND.NBATCH=0)
166: C-----
167: C
168:   do 100 I=1,NGENE
169:     IWK2(I) = IFISB(NFB0+I)
170:   100 continue
171: C
172: C .....
173: C
174:   call RANU2( IRAND, R, 4*NGENE, ICON )
175: C
176:   IPP      = NDEAD - NGENE
177:   NGEN2    = NGENE*2
178:   NGEN3    = NGENE*3
179:   WSUMF    = 0.0d0
180: *VOCL LOOP,NOVREC
181:   do 130 I = 1, NGENE
182:     IBP(I) = LSDED(IPP+I)
183: C
184:     XXX(IBP(I)) = XXXF(IWK2(I))
185:     YYY(IBP(I)) = YYYF(IWK2(I))
186:     ZZZ(IBP(I)) = ZZZF(IWK2(I))
187:     D      = 2.0*R(I) - 1.0
188:     AAA(IBP(I)) = D
189:     D      = SQRT(1.0-D**2)
190:     D2     = PI2*R(NGENE+I)
191:     BBB(IBP(I)) = COS(D2)*D
192:     CCC(IBP(I)) = SIN(D2)*D
193:     IZ     = IZZF(IWK2(I))
194:     IZZ(IBP(I)) = IZ
195: C

```

```

196:       KLSF(IBP(I)) = KLSFF(IWK2(I))
197: C
198:       IREG = KZREG(IZ)
199: C
200: C ... IBREG is always set (from Jan 2000)
201:       IBREG(IBP(I)) = IREG
202:       if ( JTLT.eq.0 ) then
203: c##<2007/03/14:PN3:
204: c##       RWK1(I) = WGTF(IREG)
205:       RWK1(I) = WGTF(IREG,1) ! by prompt fission neutron
206: c##>
207:       else
208:       IBREG(IBP(I)) = IBRGF(IWK2(I))
209: c##<2007/03/14:PN3:
210: c##       RWK1(I) = WGTF(IBREG(IBP(I)))
211:       RWK1(I) = WGTF(IBREG(IBP(I)),1) ! by prompt fission neutron
212: c##>
213:       end if
214: C
215:       WSUMF = WSUMF + RWK1(I)
216: C
217:       if ( JTIME.ne.0 ) TTT(IBP(I)) = 0.0D0
218: C
219:   130 continue
220: C
221: C ..... WEIGHT NORMALIZATION ...
222: C
223: *VOCL LOOP,NOVREC
224:   do 140 I = 1, NGENE
225:     WWW(IBP(I)) = RWK1(I)*WFFACT
226:   140 continue
227: C
228: C ..... LATTICE PARAMETER BANK .....
229: C
230:   if ( JLATT.ne.0 ) then
231: *VOCL LOOP,NOVREC
232:   do 150 I = 1, NGENE
233:     LEVL(IBP(I)) = LEVLF(IWK2(I))
234:     if ( JTLT.ne.0 ) IBSPC(IBP(I),0) = 0
235:   150 continue
236: C
237:   do 170 K = 1, NEST
238: *VOCL LOOP,NOVREC
239:   do 160 I = 1, NGENE
240:     LZZ(IBP(I),K) = LZZF(IWK2(I),K)
241:     LPOS(IBP(I),K) = LPOSF(IWK2(I),K)
242:     if ( JHLAT.ne.0 ) LCRS(IBP(I),K) = LCRSF(IWK2(I),K)
243:     if ( JTLT.ne.0 ) IBSPC(IBP(I),K) = IBSPF(IWK2(I),K)
244:   160 continue
245:   170 continue
246:   end if
247: C
248: C ... retrieve distance to lattice frame ...
249: C
250:   if ( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
251:     KF = KDFBK(4)
252:     KB = KDBNK(4)
253:     do 190 K = 0, NEST
254:       do 180 I = 1, NGENE
255:         DBNK(IBP(I),KB+K) = DFBK(IWK2(I),KF+K)
256:   180 continue
257:   190 continue
258:   end if
259: C
260:   if ( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then

```

src/mvp/fissrc.f

```

261: C      KF      = KIFBK(6)
262: C      KB      = KIBNK(6)
263: C      do 210 K = 0, NEST - 1
264: C          do 200 I = 1, NGENE
265: C              IBNK(IBP(I),KB+K) = IFBK(IWK2(I),KF+K)
266: C          continue
267: C      210 continue
268: C      end if
269: C
270: C      ... retrieve STG sphere position ...
271: C
272: C      if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
273: C          KF      = KDFBK(5)
274: C          KB      = KDBNK(5)
275: C          do 230 K = 1, NEST
276: C              do 220 I = 1, NGENE
277: C                  DBNK(IBP(I),KB) = DFBK(IWK2(I),KF)
278: C                  DBNK(IBP(I),KB+1) = DFBK(IWK2(I),KF+1)
279: C                  DBNK(IBP(I),KB+2) = DFBK(IWK2(I),KF+2)
280: C              220 continue
281: C              KF      = KF + 3
282: C              KB      = KB + 3
283: C          230 continue
284: C      end if
285: C
286: C      .... STG placement flag (which type of NND should be used)
287: C
288: C      Base material (matrix) of STG region needs NND type2 sampling
289: C
290: C      if ( JSTGR.ne.0 ) then
291: C          do 240 I = 1, NGENE
292: C              INND      = 0
293: C              LVL      = LEVL(IBP(I))
294: C              if ( LVL.gt.0 ) then
295: C                  if ( LTYP(LATTNM(KZMAT(LZZ(IBP(I),LVL),1))).eq.10 ) then
296: C                      if ( LPOS(IBP(I),LVL).eq.0 ) INND      = 2
297: C                  end if
298: C              end if
299: C              IBNK(IBP(I),KIBNK(7)) = INND
300: C          240 continue
301: C      end if
302: C
303: C      ... clear flight count in flight path
304: C
305: C      if ( JFISX.ne.0 .or. JTSRF.ne.0 ) then
306: C          do 250 I = 1, NGENE
307: C              IBNK(IBP(I),KIBNK(5)) = 0
308: C          250 continue
309: C      end if
310: C
311: C-----
312: C      SAMPLING OF ENERGY & DETERMINE ENERGY GROUP
313: C-----
314: C
315: C      do 260 I = 1, NGENE
316: C          EW1(I) = EEEF(IWK2(I))
317: C          IWK3(I) = INUF(IWK2(I))
318: C          IWK8(I) = IMTF(IWK2(I))
319: C      260 continue
320: C      NN1      = NGENE + 1
321: C      NN2      = 2*NGENE + 1
322: C      NN3      = 3*NGENE + 1
323: C
324: C
325: C      ... RWK2 : dummy array

```

```

326: C
327: C      call SEF5N4( IOW, IWK3, NGENE, EW1, EW1, CX, ICX, MCX, KLIB1,
328: C          &      XLIB1, KLIB2, XLIB2, NBANK, NUC, NMT, IRAND, JDEBG, R(1),
329: C          &      R(NN1), R(NN2), RWK1, IWK2, IWK4, IWK5, IWK6, IWK7, R(NN3),
330: C          &      RWK2, IWK8, JDLYN, JTIME )
331: C
332: C
333: C      do 270 I = 1, NGENE
334: C          EW1(I) = MAX(EBOT,MIN(EW1(I),ETOP))
335: C      270 continue
336: C
337: C
338: C      call BSVDEC( ENGYB, NGP1+1, EW1, IWK3, NGENE )
339: C      *VOCL LOOP,NOVREC
340: C      do 280 I = 1, NGENE
341: C          EEE(IBP(I)) = EW1(I)
342: C          IGG(IBP(I)) = IWK3(I)
343: C      280 continue
344: C
345: C      .... MAKE FREE FLIGHT STACK .....
346: C
347: C      do 290 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
348: C          LSFFL(I) = IBP(I-NFFL(NZONE+1))
349: C          IZFFL(I) = IZZ(LSFFL(I))
350: C          if ( JIMPT.ne.0 ) then
351: C              if ( JTLLT.eq.0 ) then
352: C                  IREG = KZREG(IZFFL(I))
353: C              else
354: C                  IREG = IBREG(LSFFL(I))
355: C              end if
356: C              XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),IREG)
357: C          end if
358: C      290 continue
359: C      *VOCL LOOP,SCALAR
360: C      do 300 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
361: C          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
362: C      300 continue
363: C      NFFL(NZONE+1) = NFFL(NZONE+1) + NGENE
364: C
365: C      .... Update sum of weights .....
366: C
367: C      WSUMF2 = DBLE(NGENE)
368: C
369: C      WSUM = WSUM + WSUMF2
370: C      XSOC(NBATCH+1) = XSOC(NBATCH+1) + WSUMF2
371: C      NCNTR(1,1) = NCNTR(1,1) + NGENE
372: C      WCNTR(1,1) = WCNTR(1,1) + WSUMF2
373: C
374: C      NFISB = 0
375: C
376: C      return
377: C      end

```

src/mvp/fissrcpt.f

```

1:      subroutine FISSRCPT(NGENE, IBP, NHIST, IFISB, NFB0,
2:      &      NFBNK0, IZZF, IBRGF, IMTF,
3:      &      NBANK, LSDED, IBREG, IBNK, KIBNK, MIBNK,
4:      &      NZONE, KZREG, NREG, WGTf, NBATCH, NDEAD, IWK2, RWK1,
5:      &      NPTDS, WWD, WDO, WCTRP, WWD2, NPTCS, WWR, WR0, NUCPT,
6:      &      NGSP, WSD, NTPT, SWD0, WSC, SWR0, NSKIP, NORDDs,
7:      &      NOPSDS,
8:      &      RWK2, IWK8,
9:      &      WWB, WB0, WSB, SWB0, NPTBE,
10:     &      WWBD, WBD0, WSBD, SWBD0, WWLD, WLD0, WSLD, SWLD0)
11: C=<MVP>=====
12: C PURPOSE: Set initial parameters for perturbation and beff
13: C      calculations.
14: C
15: C CALLED IN: SOURCE
16: C CALLS:
17: C HISTORY: EXTRACTED FROM 'SOURCE' ROUTINE. (2009/05/29)
18: C-----
19: C arguments
20: C i NGENE : number of fission sources set in this routine.
21: C
22: C i IFISB(*) : index pointer to fission neutron bank selected for
23: C      fission source of current batch.
24: C      More than one elements of this array may be pointing
25: C      the same fission bank elements.(take care for vectorization)
26: C
27: C I NFBNK0: fission bank array length
28: C i XXXF,YYF,ZZF,EEF,INUF,IZZF,LEVLf,LZZF,LPOSF,LCSF,IBRGF,IBSPF,
29: C      KLSFF, DFBK,KDFBK,IFBK,KIFBK :
30: C      Fission neutron bank arrays.
31: C
32: C <<Caution>>
33: C
34: C      When NHSUB < NHIST, which means more than one sub-batch are
35: C      necessary for a batch, another set of fission bank arrays
36: C      XXXF2, YYF2, ... are passed as XXXF, YYF, ... and
37: C      array size NFBNK0 is not the global data for XXXF, YYF, ...
38: C      but that for XXXF2, YYF2 (must be same as NHIST).
39: C
40: C o IBP(*) : bank pointer of generated source.
41: C o WSUM : source weight sum
42: C o XSOC : source weight sum of current batch id XSOC(NBATCH+1)
43: C o NCNTR(1,1), WCNTR(1,1) : total number and weight of fission source
44: C
45: C o NFFL, IZFFL, LSFFL : free flight particle stack.
46: C-----
47: C=====
48:      implicit real*8(A-H,O-Z)
49: C
50:      include 'INC/_FLAGS'
51: C
52:      integer IFISB(NHIST)
53: C
54: C .... bank pointers returned ...
55: C
56:      integer IBP(NGENE)
57: C
58: C .... PARTICLE BANK .....
59: C
60:      real WGTf(NREG,2)
61:      integer IBREG(NBANK)
62: C
63: C ... optional bank parameters
64: C
65:      integer IBNK(NBANK,*)

```

```

66:      integer KIBNK(0:MIBNK)
67: C
68: C .... FISSION BANK .....
69: C
70:      integer IZZF(NFBNK0)
71:      integer IBRGF(NFBNK0)
72:      integer IMTF(NFBNK0)
73: C
74: C .... STACKS ...(FREE FLIGHT & MORGUE) .....
75: C
76:      integer LSDED(NBANK)
77: C
78: C .... etc.
79: C
80:      real RWK1(NBANK)
81:      integer IWK2(NBANK)
82:      real RWK2(NBANK)
83:      integer IWK8(NBANK)
84: C
85: C .... GEOMETRY ARRAY DATA
86: C
87:      integer KZREG(NZONE)
88: C
89: C ... arrays for perturbation calculation ...
90: C
91:      real*8 WWD(NBANK,NPTDS,NORDDs), WWD2(NBANK,NPTDS)
92:      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
93:      real*8 WWR(NBANK,NPTCS)
94:      real*8 WSC(NBANK,0:NGSP,NPTCS)
95:      real*8 WDO(NFBNK0,0:NGSP,NPTDS,NOPSDS)
96:      real*8 WCTRP(NUCPT,7,NTPT)
97:      real*8 WR0(NFBNK0,0:NGSP,NPTCS)
98:      real*8 SWD0(0:NGSP,NPTDS,NOPSDS)
99:      real*8 SWR0(0:NGSP,NPTCS)
100: C
101: C ... arrays for beta-effective calculation ...
102: C
103:      real*8 WWB(NBANK)
104:      real*8 WB0(NFBNK0,0:NGSP)
105:      real*8 WSB(NBANK,0:NGSP)
106:      real*8 SWB0(0:NGSP)
107:      real*8 WWBD(NBANK)
108:      real*8 WBD0(NFBNK0,NGSP)
109:      real*8 WSBD(NBANK,NGSP)
110:      real*8 SWBD0(NGSP)
111:      real*8 WWLD(NBANK)
112:      real*8 WLD0(NFBNK0,NGSP)
113:      real*8 WSLD(NBANK,NGSP)
114:      real*8 SWLD0(NGSP)
115: C
116: C ... local variables ...
117: C
118: C-----
119: C .... SOURCE IS TAKEN FROM FISSION BANK (JEIGN.EQ.1.AND.NBATCH>0)
120: C-----
121: C-----
122: C
123:      do 100 I=1,NGENE
124:          IWK2(I) = IFISB(NFB0+I)
125:      100 continue
126: C
127: C
128: C .....
129: C
130: C

```

src/mvp/fissrcpt.f

```

131:     IPP      = NDEAD - NGENE
132:     WSUMF    = 0.0d0
133:
134:     if (JPERT.ne.0) then
135:
136:         do NPT = 1, NUCPT
137:             do IPT = 1, NPT
138:                 do JJ = 1, 7
139:                     WCTRP(NPT,JJ,IPT) = 0.0d0
140:                 end do
141:             end do
142:         end do
143:
144:         do IPT = 1, NPTDS
145:             do ISP = 0, NGSP
146:                 do IOP = 1, NOPSDS
147:                     SWD0(ISP,IPT,IOP) = 0.0d0
148:                 end do
149:             end do
150:         end do
151:
152:         do IPT = 1, NPTCS
153:             do ISP = 0, NGSP
154:                 SWR0(ISP,IPT) = 0.0d0
155:             end do
156:         end do
157:
158:         if (NPTBE.gt.0) then
159:             do ISP = 0, NGSP
160:                 SWB0(ISP) = 0.0d0
161:             end do
162:             do ISP = 1, NGSP
163:                 SWBD0(ISP) = 0.0d0
164:                 SWLD0(ISP) = 0.0d0
165:             end do
166:         end if
167:
168:     end if
169:
170: *VOCL LOOP,NOVREC
171: do 130 I = 1, NGENE
172:     IBP(I) = LSDED(IPP+I)
173:     IZ     = IZZF(IWK2(I))
174:     IREG   = KZREG(IZ)
175:     IBREG(IBP(I)) = IREG
176:     if (JTLLT.eq.0) then
177:         RWK1(I) = WGTF(IREG,1)
178:     else
179:         IBREG(IBP(I)) = IBRGF(IWK2(I))
180:         RWK1(I) = WGTF(IBREG(IBP(I)),1)
181:     end if
182:     WSUMF = WSUMF + RWK1(I)
183: 130 continue
184: C
185: C ... store delayed neutron flag ...
186: C
187:     if (JBEFF.ne.0 .or. NPTBE.gt.0) then
188:         do I = 1, NGENE
189:             if (IMTF(IWK2(I)).eq.98) then
190:                 IBNK(IBP(I),KIBNK(20)) = 1
191:             else
192:                 IBNK(IBP(I),KIBNK(20)) = 0
193:             end if
194:         end do
195:     end if

```

! by prompt fission neutron

! by prompt fission neutron

```

196:
197:     if (JPERT.ne.0) then
198:
199: *VOCL LOOP,NOVREC
200:         do 134 I = 1, NGENE
201:             do IPT = 1, NPTDS
202:                 SWD0(0,IPT,1) = SWD0(0,IPT,1)
203:                 & + WD0(IWK2(I),0,IPT,1)*RWK1(I)
204:             do ISP = 1, NGSP
205:                 do IOP = 1, NOPSDS
206:                     SWD0(ISP,IPT,IOP) = SWD0(ISP,IPT,IOP)
207:                     & + WD0(IWK2(I),ISP,IPT,IOP)*RWK1(I)
208:                 end do
209:             end do
210:         end do
211:
212:         do IPT = 1, NPTCS
213:             do ISP = 0, NGSP
214:                 SWR0(ISP,IPT) = SWR0(ISP,IPT)
215:                 & + WR0(IWK2(I),ISP,IPT)*RWK1(I)
216:             end do
217:         end do
218:
219:         if (NPTBE.gt.0) then
220:             do ISP = 0, NGSP
221:                 SWB0(ISP) = SWB0(ISP) + WB0(IWK2(I),ISP)*RWK1(I)
222:             end do
223:             do ISP = 1, NGSP
224:                 SWBD0(ISP) = SWBD0(ISP) + WBD0(IWK2(I),ISP)*RWK1(I)
225:                 SWLD0(ISP) = SWLD0(ISP) + WLD0(IWK2(I),ISP)*RWK1(I)
226:             end do
227:         end if
228: 134 continue
229:
230:         do IPT = 1, NPTDS
231:             SWD0(0,IPT,1) = SWD0(0,IPT,1)/WSUMF
232:             do ISP = 1, NGSP
233:                 do IOP = 1, NOPSDS
234:                     SWD0(ISP,IPT,IOP) = SWD0(ISP,IPT,IOP)/WSUMF
235:                 end do
236:             end do
237:         end do
238:
239:         do IPT = 1, NPTCS
240:             do ISP = 0, NGSP
241:                 SWR0(ISP,IPT) = SWR0(ISP,IPT)/WSUMF
242:             end do
243:         end do
244:
245:         if (NPTBE.gt.0) then
246:             do ISP = 0, NGSP
247:                 SWB0(ISP) = SWB0(ISP)/WSUMF
248:             end do
249:             do ISP = 1, NGSP
250:                 SWBD0(ISP) = SWBD0(ISP)/WSUMF
251:                 SWLD0(ISP) = SWLD0(ISP)/WSUMF
252:             end do
253:             NND = 0.d0
254:             do I = 1, NGENE
255:                 NND = NND+IBNK(IBP(I),KIBNK(20))
256:             end do
257:             WFCT = dble(NGENE)/dble(NGENE-NND)
258:         end if
259:
260: *VOCL LOOP,NOVREC

```

src/mvp/fissrcpt.f

```

261:         do 135 I = 1, NGENE
262:
263: c ... Set initial additional weightht ...
264:
265:         do IPT = 1, NPTDS
266:             do IOD = 1, NORDDS
267:                 WWD(IBP(I),IPT,IOD) = 0.0d0
268:             end do
269:             WWD2(IBP(I),IPT) = -(WWD(IBP(I),IPT,1))*2
270:
271:             WSD(IBP(I),0,IPT,1) = WD0(IWK2(I),0,IPT,1)
272:             & - SWD0(0,IPT,1)
273:             do ISP = 1, NGSP
274:                 WSD(IBP(I),ISP,IPT,1) =
275:                 & WD0(IWK2(I),ISP,IPT,1) - SWD0(ISP,IPT,1)
276:                 WSD(IBP(I),ISP,IPT,2) =
277:                 & WD0(IWK2(I),ISP,IPT,2) - SWD0(ISP,IPT,2)
278:                 & + 2.d0*SWD0(ISP,IPT,1)
279:                 & *(SWD0(ISP,IPT,1) - WD0(IWK2(I),ISP,IPT,1))
280:                 WSD(IBP(I),ISP,IPT,3) =
281:                 & WD0(IWK2(I),ISP,IPT,3) - SWD0(ISP,IPT,3)
282:                 & - 3.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,2)
283:                 & - 3.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,1)
284:                 WSD(IBP(I),ISP,IPT,4) =
285:                 & WD0(IWK2(I),ISP,IPT,4) - SWD0(ISP,IPT,4)
286:                 & - 4.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,3)
287:                 & - 6.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,2)
288:                 & - 4.d0*SWD0(ISP,IPT,3)*WSD(IBP(I),ISP,IPT,1)
289:                 WSD(IBP(I),ISP,IPT,5) =
290:                 & WD0(IWK2(I),ISP,IPT,5) - SWD0(ISP,IPT,5)
291:                 & - 5.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,4)
292:                 & -10.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,3)
293:                 & -10.d0*SWD0(ISP,IPT,3)*WSD(IBP(I),ISP,IPT,2)
294:                 & - 5.d0*SWD0(ISP,IPT,4)*WSD(IBP(I),ISP,IPT,1)
295:                 WSD(IBP(I),ISP,IPT,6) =
296:                 & WD0(IWK2(I),ISP,IPT,6) - SWD0(ISP,IPT,6)
297:                 & - 6.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,5)
298:                 & -15.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,4)
299:                 & -20.d0*SWD0(ISP,IPT,3)*WSD(IBP(I),ISP,IPT,3)
300:                 & -15.d0*SWD0(ISP,IPT,4)*WSD(IBP(I),ISP,IPT,2)
301:                 & - 6.d0*SWD0(ISP,IPT,5)*WSD(IBP(I),ISP,IPT,1)
302:                 WSD(IBP(I),ISP,IPT,7) =
303:                 & WD0(IWK2(I),ISP,IPT,7) - SWD0(ISP,IPT,7)
304:                 & - 7.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,6)
305:                 & -21.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,5)
306:                 & -35.d0*SWD0(ISP,IPT,3)*WSD(IBP(I),ISP,IPT,4)
307:                 & -35.d0*SWD0(ISP,IPT,4)*WSD(IBP(I),ISP,IPT,3)
308:                 & -21.d0*SWD0(ISP,IPT,5)*WSD(IBP(I),ISP,IPT,2)
309:                 & - 7.d0*SWD0(ISP,IPT,6)*WSD(IBP(I),ISP,IPT,1)
310:                 WSD(IBP(I),ISP,IPT,8) =
311:                 & WD0(IWK2(I),ISP,IPT,8) - SWD0(ISP,IPT,8)
312:                 & - 8.d0*SWD0(ISP,IPT,1)*WSD(IBP(I),ISP,IPT,7)
313:                 & -28.d0*SWD0(ISP,IPT,2)*WSD(IBP(I),ISP,IPT,6)
314:                 & -56.d0*SWD0(ISP,IPT,3)*WSD(IBP(I),ISP,IPT,5)
315:                 & -70.d0*SWD0(ISP,IPT,4)*WSD(IBP(I),ISP,IPT,4)
316:                 & -56.d0*SWD0(ISP,IPT,5)*WSD(IBP(I),ISP,IPT,3)
317:                 & -28.d0*SWD0(ISP,IPT,6)*WSD(IBP(I),ISP,IPT,2)
318:                 & - 8.d0*SWD0(ISP,IPT,7)*WSD(IBP(I),ISP,IPT,1)
319:             end do
320:
321:         end do
322:
323: c This if statement is temporary.
324:         if (JPDEN.ne.0) then
325:             do IPT = 1, NPTCS
326:
327:                 WWR(IBP(I),IPT) = 1.0d0
328:                 if (NBATCH.gt.NSKIP) then
329:                     do ISP = 0, NGSP
330:                         WSC(IBP(I),ISP,IPT) =
331:                         & WRO(IWK2(I),ISP,IPT)/SWRO(ISP,IPT) - 1.0d0
332:                     end do
333:                 end if
334:             end if
335:
336:             if (NPTBE.gt.0) then
337:
338:                 ccc WWB(IBP(I)) = 1.0d0
339:                 c ... Prompt k-ratio method ...
340:                 c --- scheme 1 ---
341:                 WWB(IBP(I)) = 1.0d0 - dble(IBNK(IBP(I),KIBNK(20)))
342:                 c --- scheme 2 ---
343:                 WWB(IBP(I)) = (1.0d0-dble(IBNK(IBP(I),KIBNK(20))))*WFCT
344:                 c ... Delayed k-ratio method ...
345:                 c WWB(IBP(I)) = dble(IBNK(IBP(I),KIBNK(20)))
346:
347:                 if (NBATCH.gt.NSKIP) then
348:                     do ISP = 0, NGSP
349:                         WSB(IBP(I),ISP) = WB0(IWK2(I),ISP)/SWB0(ISP) - 1.0d0
350:                     end do
351:                 end if
352:
353: c ... beta-effective with differential operator sampling
354:
355:                 WWBD(IBP(I)) = dble(IBNK(IBP(I),KIBNK(20)))
356:                 WWLD(IBP(I)) = 0.d0
357:                 if (NBATCH.gt.NSKIP) then
358:                     do ISP = 1, NGSP
359:                         WSBD(IBP(I),ISP) = WBD0(IWK2(I),ISP) - SWBD0(ISP)
360:                         WSLD(IBP(I),ISP) = WLD0(IWK2(I),ISP) - SWLD0(ISP)
361:                     end do
362:                 end if
363:
364:             end if
365:
366:             135 continue
367:         end if
368:
369:         if (JPERT.ne.0) then
370:             do I = 1, NFBNK0
371:                 do IPT = 1, NPTDS
372:                     do ISP = 0, NGSP
373:                         do IOP = 1, NOPSDS
374:                             WD0(I,ISP,IPT,IOP) = 0.0d0
375:                         end do
376:                     end do
377:                 end do
378:             end if
379:
380:
381:         return
382:         end

```

src/mvp/flagin.f

```

1:      subroutine FLAGIN
2: C=====
3: C purpose: initial setting of option flags (MVP)
4: C called in: main
5: C=====
6:      include 'INC/_FLAGS'
7:      include 'INC/_KPIDS'
8:
9:      JREST   = 0
10:     JARST   = 0
11:
12:     do 150 I = 1, MKPAR
13:       JKHAR(I) = 0
14: 150 continue
15:     JNEUT   = 1
16:     JXPAR(KPNEUT) = 1
17:     JPHOT   = 0
18:
19:     JIMAG   = 0
20:     JTIME   = 0
21:     JDELT   = 0
22:
23:     JFISS   = 0
24:     JEIGN   = 0
25:     JFIXD   = 1
26:     JDDX    = 1
27:     JADJM   = 0
28:
29:     JRRLT   = 1
30:     JIMPT   = 0
31:     JWWND   = 0
32:     JPSTR   = 0
33:     JWTIM   = 0
34:
35:     JFCOL   = 0
36:     JNCOL   = 0
37:     JBIAS   = 0
38:     JRWVR   = 0
39:
40:     JRESP   = 0
41:     JMNTR   = 0
42:     JVMNT   = 0
43:     JOSRC   = 0
44:     JRSTF   = 1
45:     JSCRT   = 1
46:
47:     JALLZ   = 0
48:     JONEZ   = 1
49:     JPICL   = 0
50:     JLATT   = 0
51:     JFPRT   = 0
52:     JRTTR   = 1
53:     JRTCL   = 0
54:     JTLLT   = 0
55:     JREPR   = 1
56:     JTSKR   = 1
57:     JBREM   = 0
58:     JPHNU   = 0 ! photonuc
59:     JRUNM   = 0
60:     JUNDG   = 0
61:
62:     JFISX   = 0
63:     JSTGR   = 0
64:
65:     JANLF   = 0
66:
67:     JPTIM   = 0
68:     JAMXC   = 0
69:     JPTDT   = 0
70:
71:     JTSTRF   = 0
72:
73:     JFMUL   = 0
74:     JEXDP   = 0 ! res-scat
75:     JPERT   = 0 ! pert
76:     JDLYN   = 1 ! dn
77:     JNRUN   = 0
78:     JSCTM   = 0 ! scat-mtx
79:     JSCTMU  = 0 ! scat-mtx
80:     JBEFF   = 0 ! beff
81:     JTLST   = 0 ! time-list
82:
83:     do 100 I = 1, 8
84:       JMICE(I) = 0
85: 100 continue
86:     do 110 I = 1, 8
87:       JMACE(I) = 0
88: 110 continue
89:     JPRTS(1) = 1
90:     do 120 I = 2, 20
91:       JPRTS(I) = 0
92: 120 continue
93:     do 130 I = 1, MAXJDB
94:       JDEBG(I) = 0
95: 130 continue
96:
97:     do 140 I = 1, MXJVP
98:       JVPSP(I) = 0
99: 140 continue
100:
101:     JSTPRN(1) = 1
102: C/#IF PARA(SX* CRAY*)
103:     JSTPRN(1) = 2
104: C/#ENDIF
105:     JSTPRN(2) = 2
106:
107:     JNWMX   = 0
108:
109:     return
110: end

```

src/mvp/fldist.f

```

1:      subroutine FLDIST(IOW,MZONE, ISAFE, MAT, IHF, JJJNND,
2:      N      NPKIND, NGROUP,NBANK,NZONE, DINF,
3:      N      NDEAD,NCOLS,NCOLP,
4:      N      NUNV, NSPACE,NKTCSP,
5:      N      NEST,NEVENT,NTIME,TCUT,
6:      &      X,      Y,      Z,      AI,      BI,      CI,
7:      &      W,      DI,      IGI,      IBP,      IFC,
8:      &      DLOC1, DLOC2, IKL,      IKL2, XUP,      YUP,      ZUP,
9:      &      AUP,      BUP,      CUP,      TI,VI, ISTG, ILUP,
10:     &      KSORT,IKSORT,KS,
11:     B      KKP, IZZ,      EEE,      TTT,      ITT,      LEVL,
12:     F      LPOS, IBSPC,IBREG,
13:     B      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
14:     S      LSCOL, LSCLP, LSSRC, NNXT, IZNXT,
15:     S      LSDED,
16:     G      KZREG,KSPSU,KTCSP,
17: c##<2007/03/14:PN3:PN4:
18: c## T      TFLH ,NCNTR, WCNTR)
19: c## T      TFLH ,NCNTR, WCNTR,
20: c## &      NREG, IGG, NTCUT, WTCUT, KPNFG )
21: c##>
22: C=<MVP>=====
23: C PURPOSE: Determination of flight distance and stack handling.
24: C CALLED IN: FLIONE
25: C CALLS:
26: C=====
27: C
28: C === INTER-STACK DATA FLOW ===
29: C
30: C      Free-Flight Stack  -----+-----> Next-Zone-Search Stack
31: C      (LSFFL,IZFFL,NFFL)      |      (LSSRC,IZNXT,NNXT)
32: C                               +-----> Collision Stack
33: C                               |      (LSCOL,NCOLS)
34: C                               |      (LSCLP,NCOLP)
35: C      (LOST) +-----> Dead Particle Stack
36: C                               |      (LSDED,NDEAD)
37: C
38: C === BANK DATA TO BE UPDATED ===
39: C
40: C      (XXX,YYY,ZZZ) : Position
41: C      (AAA,BBB,CCC) : Direction      (Hex-lattice or overlapping frame)
42: C      LEVL          : Lattice Level (Hex-lattice or overlapping frame)
43: C      IZZ           : Zone #        (Hex-lattice or overlapping frame)
44: C
45: C === BANK DATA ADDED NEWLY ===
46: C
47: C      DATA IN SIGMA BANK
48: C=====
49: c      implicit real*8(D)
50: C
51: c      include 'INC/_KPIDS'
52: C
53: c      include 'INC/_FLAGS'
54: c      include '../shared/INC/_TASKDT'
55: C
56: C **** PARTICLE BANK ****
57: C
58: c      real EEE(NBANK)
59: c      real*8 TTT(NBANK)
60: c      integer ITT(NBANK)
61: c      integer KKP(NBANK)
62: c      integer IZZ(NBANK), LEVL(NBANK), LPOS(NBANK,NEST)
63: C
64: c      integer IBSPC(NBANK,0:NEST)
65: c      integer IBREG(NBANK)

```

```

66: c##<2007/03/14:PN3:
67: c      integer IGG(NBANK)
68: c##>
69: C
70: C ... optional bank parameters
71: C
72: c      real*8 DBNK(NBANK,*)
73: c      integer KDBNK(0:MDBNK)
74: c      integer IBNK(NBANK,*)
75: c      integer KIBNK(0:MIBNK)
76: c##<2007/03/14:PN4:
77: c      integer KPNFG(NBANK)
78: c##>
79: C
80: C **** STACKS ****
81: C
82: c      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
83: c      integer LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK)
84: c      integer LSCOL(NBANK)
85: c      integer LSCLP(NBANK)
86: c      integer LSDED(NBANK)
87: C
88: C **** GEOMETRY ****
89: C
90: c      integer KZREG(NZONE)
91: c      integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
92: c      real*8 DINF
93: C
94: C **** TALLY ****
95: C
96: c      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
97: c      real*8 TFLH(KPLIM)
98: c##<2007/03/14:PN3:
99: c      real*8 NTCUT(NGROUP,NREG), WTCUT(NGROUP,NREG)
100: c##>
101: C
102: C **** Input output items ( working arrays in FLIONE ) ****
103: C
104: c      real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
105: c      &      CI(NBANK), W(NBANK), DI(NBANK), DLOC1(NBANK), DLOC2(NBANK)
106: c      integer IGI(NBANK), IBP(NBANK), IFC(NBANK)
107: c      integer IKL(NBANK), IKL2(NBANK)
108: C
109: C ... particle sorting information
110: C
111: c      integer KSORT(KS), IKSORT(KS+1)
112: C
113: c      .... USED FOR HEXAGONAL LATTICE GEOMETRY and STG region ...
114: C
115: c      integer ILUP(NBANK)
116: c      real*8 XUP(NBANK), YUP(NBANK), ZUP(NBANK)
117: c      real*8 AUP(NBANK), BUP(NBANK), CUP(NBANK)
118: C
119: c      integer ISTG(NBANK)
120: C
121: c      .... only for time depende case ...
122: C
123: c      real*8 TI(NBANK), VI(NBANK)
124: C
125: C .... constants ....
126: C
127: C --- light speed --- (cm/s)
128: C
129: c      real*8 CLIGHT
130: c      parameter( CLIGHT = 2.99792458D10 )

```

src/mvp/fldist.f

```

131: C
132: C --- neutron mass --- (gram)
133: C
134:       real*8 NMASS
135:       parameter( NMASS      = 1.6749286D-24 )
136: C
137: C --- erg / eV --- electron charge
138: C
139:       real*8 EECHRG
140:       parameter( EECHRG     = 1.60217733D-12 )
141: C
142: C --- neutron's mc**2 in eV
143: C
144:       real*8 MC2
145:       parameter( MC2      = NMASS*CLIGHT**2/EECHRG )
146: C
147: C **** LOCAL ARRAY ****
148: C
149: C
150: C -----
151: C
152: C=====
153: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) ....
154: C      (non time dependent case)
155: C=====
156: C
157: C
158: C .... count up path flight counter here (for collided particle)
159: C      it is cleared to zero afterwards)
160: C
161:       KI5      = KIBNK(5)
162:       KI7      = KIBNK(7)
163:       if ( KI5.ne.0 ) then
164: *VOCL LOOP,NOVREC
165:         do 100 I = 1, ISAFE
166:           if ( IBNK(IBP(I),KI5).lt.0 ) then
167:             IBNK(IBP(I),KI5) = 1
168:           else
169:             IBNK(IBP(I),KI5) = IBNK(IBP(I),KI5) + 1
170:           end if
171:         100 continue
172:       end if
173: C
174: C
175: C ... ipart is parcticle kind index for WCNTR in non-coupled problem
176: C
177:       IPART = 1
178:       if ( NPKIND.eq.1 ) then
179:         do 110 IK = 1, KPLIM
180:           if ( JKPAR(IK).ne.0 ) then
181:             IPART = IK
182:             go to 120
183:           end if
184:         110 continue
185:         120 continue
186:       end if
187: C
188:       if ( JTIME.eq.0 ) then
189:         ICOLS = 0
190:         ICOLP = 0
191: C
192: C=====
193: C=====
194: C===== When hexagonal lattice or overlapping lattice is used
195: C=====

```

```

196: C =====
197: C
198:       if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
199: C
200:         INZ1 = NNXT(NZONE+1)
201: C
202: C-----
203: C .... MAT > 0 : REAL MEDIUM .....
204: C .... JNCOL = 0 : ordinary treatment for collision .....
205: C-----
206: C
207:       if ( MAT.gt.0.and.JNCOL.eq.0 ) then
208:         INXT = 0
209:         IHH = 0
210:         IST = 0
211: C
212: C-----
213: C      NEUTRON/PHOTON COUPLED
214: C-----
215: C
216:       if ( NPKIND.gt.1 ) then
217: C
218: *VOCL LOOP,NOVREC
219:         do 130 I = 1, ISAFE
220: C
221:           D1 = DLOC1(I)
222: C
223: C ----- COLLISION -----
224: C
225:           if ( DI(I).gt.D1 ) then
226:             DI(I) = D1
227: C
228:           ... assuming only neutorn and photon are treated ...
229:           if ( KKP(IBP(I)).eq.KPNEUT ) then
230:             ICOLS = ICOLS + 1
231:             LSCOL(NCOLS+ICOLS) = IBP(I)
232:           else
233:             ICOLP = ICOLP + 1
234:             LSCLP(NCOLP+ICOLP) = IBP(I)
235:           end if
236: C
237:           IKL2(I) = 0
238: C
239:           ... flight path count is cleared for collision
240:           if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
241:           ... NND type for next STG sampling
242:           if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
243: C ----- REACHED ZONE BOUNDARY -----
244: C
245:           else
246:             IZNN = MZONE
247: C
248: C .....IHF = 1 : Zone 'MZONE' is in lattice .....
249: C
250:           if ( IHF.ne.0 ) then
251: C
252:           ... reached lattice boundary ...
253: C
254:           if ( IFC(I).ne.0 ) then
255:             IHH = IHH + 1
256:             IZNN = IFC(I)
257: C
258: C      CAUTION : IZZ (ZONE #) DOES NOT MATCH MZONE IN THIS CASE
259: C      TO THE END OF THIS ROUTINE.
260: C

```


src/mvp/fldist.f

```

261:          IZZ(IBP(I)) = IFC(I)
262:          LEVL(IBP(I)) = ILUP(I)
263: C      .... make IFC(I) negative to calculate NNXT
264:          IFC(I) = -IFC(I)
265: C
266: C      .... COORDINATES & DIRECTION IN UPPER LEVEL
267: C
268:          X(I) = XUP(I)
269:          Y(I) = YUP(I)
270:          Z(I) = ZUP(I)
271:          AI(I) = AUP(I)
272:          BI(I) = BUP(I)
273:          CI(I) = CUP(I)
274:          IKL2(I) = 0
275: C      end if
276: C      end if
277: C
278:          INXT = INXT + 1
279:          LSSRC(INZ1+INXT) = IBP(I)
280:          IZNN(INZ1+INXT) = IZNN
281: C      end if
282: 130      continue
283: C
284: C-----
285: C      NON-NEUTRON/PHOTON COUPLED PROBLEM
286: C-----
287: C
288: C      else
289: C      *VOCL LOOP,NOVREC
290: C      do 170 I = 1, ISAFE
291: C
292: C          D1 = DLOC1(I)
293: C
294: C----- COLLISION -----
295: C
296: C      if ( DI(I).gt.D1 ) then
297: C          DI(I) = D1
298: C      if ( JKPAR(KPNEUT).ne.0 ) then
299: C          ICOLS = ICOLS + 1
300: C          LSCOL(NCOLS+ICOLS) = IBP(I)
301: C      else if ( JKPAR(KPPHOT).ne.0 ) then
302: C          ICOLP = ICOLP + 1
303: C          LSCLP(NCOLP+ICOLP) = IBP(I)
304: C      end if
305: C
306: C      ... flight path count is cleared for collision
307: C      if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
308: C      ... NND type for next STG sampling
309: C      if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
310: C
311: C----- REACHED ZONE BOUNDARY -----
312: C
313: C      else
314: C          IZNN = MZONE
315: C
316: C      CCCCC ---- IHF = 1 : ZONE 'MZONE' IS IN A HEXAGONAL CELL. ----
317: C
318: C      if ( IHF.ne.0 ) then
319: C
320: C      ..... REACHED LATTICE BOUNDARY .....
321: C
322: C      if ( IFC(I).ne.0 ) then
323: C          IHH = IHH + 1
324: C          IZNN = IFC(I)
325: C          IZZ(IBP(I)) = IFC(I)

```

```

326:          LEVL(IBP(I)) = ILUP(I)
327: C
328: C      .... make IFC(I) negative to calculate NNXT
329: C          IFC(I) = -IFC(I)
330: C          X(I) = XUP(I)
331: C          Y(I) = YUP(I)
332: C          Z(I) = ZUP(I)
333: C          AI(I) = AUP(I)
334: C          BI(I) = BUP(I)
335: C          CI(I) = CUP(I)
336: C          IKL2(I) = 0
337: C      end if
338: C      end if
339: C
340: C          INXT = INXT + 1
341: C          LSSRC(INZ1+INXT) = IBP(I)
342: C          IZNN(INZ1+INXT) = IZNN
343: C      end if
344: C      continue
345: C      end if
346: C-----
347: C      update NNXT, LPOS, IBSPC if IHH>0
348: C-----
349: C
350: C      if ( IHH.gt.0 ) then
351: C      *VOCL LOOP,SCALAR
352: C      do 180 I = 1, ISAFE
353: C      if ( IFC(I).lt.0 ) then
354: C          NNXT(-IFC(I)) = NNXT(-IFC(I)) + 1
355: C      end if
356: C      continue
357: C
358: C      ... "cell position" LPOS is changed here for STG cell
359: C
360: C      if ( JJJNND.ne.0 ) then
361: C      *VOCL LOOP,NOVREC
362: C      do 190 I = 1, ISAFE
363: C      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
364: C          LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
365: C      end if
366: C      continue
367: C      end if
368: C      if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
369: C      *VOCL LOOP,NOVREC
370: C      do 200 I = 1, ISAFE
371: C      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
372: C          IBSPC(IBP(I),LEVL(IBP(I))) =
373: C          &          KTCSP(
374: C          &          KPSU(KZREG(MZONE),
375: C          &          IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
376: C      end if
377: C      continue
378: C      end if
379: C      end if
380: C      NNXT(MZONE) = NNXT(MZONE) + INXT - IHH
381: C
382: C-----
383: C      .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
384: C      .... JNCOL > 0 : "no-collision" treatment
385: C-----
386: C
387: C      else
388: C          INXT = ISAFE
389: C      if ( IHF.eq.0 ) then
390: C          do 210 I = 1, INXT

```

src/mvp/fldist.f

```

391:          LSSRC(INZ1+I) = IBP(I)
392:          IZNXT(INZ1+I) = MZONE
393:      210      continue
394:          NNXT(MZONE) = NNXT(MZONE) + INXT
395:      else
396:          IHHHH = 0
397:      *VOCL LOOP,NOVREC
398:          do 220 I = 1, INXT
399:              LSSRC(INZ1+I) = IBP(I)
400:              if ( IFC(I).eq.0 ) then
401:                  IZNXT(INZ1+I) = MZONE
402:              else
403:                  X(I) = XUP(I)
404:                  Y(I) = YUP(I)
405:                  Z(I) = ZUP(I)
406:                  AI(I) = BUP(I)
407:                  BI(I) = BUP(I)
408:                  CI(I) = CUP(I)
409:                  IKL2(I) = 0
410:      C
411:                  IHHHH = IHHHH + 1
412:                  IZNXT(INZ1+I) = IFC(I)
413:                  IZZ(IBP(I)) = IFC(I)
414:                  LEVL(IBP(I)) = ILUP(I)
415:      C          .... make IFC(I) negative to calculate NNXT
416:                  IFC(I) = -IFC(I)
417:              end if
418:          220      continue
419:
420:          if ( IHHHH.gt.0 ) then
421:      *VOCL LOOP,SCALAR
422:              do 230 I = 1, INXT
423:                  if ( IFC(I).lt.0 ) then
424:                      NNXT(-IFC(I)) = NNXT(-IFC(I)) + 1
425:                  end if
426:              230      continue
427:      C
428:      C          ... "cell position" LPOS is changed here for STG cell
429:      C
430:          if ( JJJNND.ne.0 ) then
431:      *VOCL LOOP,NOVREC
432:              do 240 I = 1, INXT
433:                  if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
434:                      LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
435:                  end if
436:              240      continue
437:          end if
438:          if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
439:      *VOCL LOOP,NOVREC
440:              do 250 I = 1, INXT
441:                  if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
442:                      IBSPC(IBP(I),LEVL(IBP(I))) =
443:      &                      KTCSP(
444:      &                      KSPSU(KZREG(MZONE),
445:      &                      IBSPC(IBP(I),LEVL(IBP(I)))-1))+IKL(I))
446:                  end if
447:              250      continue
448:          end if
449:          end if
450:          NNXT(MZONE) = NNXT(MZONE) + INXT - IHHHH
451:      end if
452:  end if
453:  C
454:  C
455:      if ( JKPAR(KPNEUT).ne.0 ) NCOLS = NCOLS + ICOLS

```

```

456:          if ( JKPAR(KPPHOT).ne.0 ) NCOLP = NCOLP + ICOLP
457:          NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
458:      C
459:      C
460:      C =====
461:      C =====
462:      C ===== When no hexagonal lattice or overlapping frame is used =====
463:      C =====
464:      C =====
465:      C
466:      C
467:      else
468:          INXT = 0
469:      C
470:      C-----
471:      C .... MAT > 0 : REAL MEDIUM .....
472:      C .... JNCOL = 0 : ordinary collision treatment .....
473:      C-----
474:      C
475:          if ( MAT.gt.0.and.JNCOL.eq.0 ) then
476:      C
477:      C-----
478:      C          NEUTRON/PHOTON COUPLED PROBLEM
479:      C-----
480:      C
481:          if ( NPkind.gt.1 ) then
482:      *VOCL LOOP,NOVREC
483:              do 260 I = 1, ISAFE
484:      C
485:                  D1 = DLOC1(I)
486:      C
487:      C----- COLLISION -----
488:      C
489:          if ( DI(I).gt.D1 ) then
490:              DI(I) = D1
491:              if ( KKP(IBP(I)).eq.KPNEUT ) then
492:                  ICOLS = ICOLS + 1
493:                  LSCOL(NCOLS+ICOLS) = IBP(I)
494:              else
495:                  ICOLP = ICOLP + 1
496:                  LSCLP(NCOLP+ICOLP) = IBP(I)
497:              end if
498:              IKL2(I) = 0
499:      C
500:      C          ... flight path count is cleared for collision
501:          if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
502:      C          ... NND type for next STG sampling
503:          if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
504:      C
505:      C----- REACHED ZONE BOUNDARY -----
506:      C
507:          else
508:              INXT = INXT + 1
509:              LSSRC(NNXT(NZONE+1)+INXT) = IBP(I)
510:          end if
511:      260      continue
512:      C
513:      C-----
514:      C          NON NEUTRON/PHOTON COUPLED PROBLEM ====
515:      C-----
516:      C
517:      else
518:      *VOCL LOOP,NOVREC
519:          do 270 I = 1, ISAFE
520:      C

```

src/mvp/fldist.f

```

521:          D1      = DLOC1(I)
522: C
523: C ----- COLLISION -----
524: C
525:          if ( DI(I).gt.D1 ) then
526:            DI(I) = D1
527:            if ( JKPAR(KPNEUT).ne.0 ) then
528:              ICOLS = ICOLS + 1
529:              LSCOL(NCOLS+ICOLS) = IBP(I)
530:            else if ( JKPAR(KPPHOT).ne.0 ) then
531:              ICOLP = ICOLP + 1
532:              LSCLP(NCOLP+ICOLP) = IBP(I)
533:            end if
534:            IKL2(I) = 0
535: C
536: C ... flight path count is cleared for collision
537: C if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
538: C ... NND type for next STG sampling
539: C if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
540: C
541: C
542: C ----- REACHED ZONE BOUNDARY -----
543: C
544:          else
545:            INXT = INXT + 1
546:            LSSRC(NNXT(NZONE+1)+INXT) = IBP(I)
547:          end if
548: 270      continue
549:        end if
550: C
551: C -----
552: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
553: C .... JNCOL >< 0 : "no-collision" treatment
554: C -----
555: C
556:          else
557:            INXT = ISAFE
558:            do 280 I = 1, INXT
559:              LSSRC(NNXT(NZONE+1)+I) = IBP(I)
560: 280      continue
561:          end if
562: C
563:          if ( JKPAR(KPNEUT).ne.0 ) NCOLS = NCOLS + ICOLS
564:          if ( JKPAR(KPPHOT).ne.0 ) NCOLP = NCOLP + ICOLP
565: C
566:          do 290 I = 1, INXT
567:            IZNXT(NNXT(NZONE+1)+I) = MZONE
568: 290      continue
569:            NNXT(MZONE) = NNXT(MZONE) + INXT
570:            NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
571:          end if
572: C
573: C =====
574: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) ....
575: C (time dependent case)
576: C =====
577: C
578:          else
579:            ICOLS = 0
580:            ICOLP = 0
581: C
582: C ... time remained until time-cutoff & velocity
583: C
584:          do 300 I = 1, ISAFE
585:            TI(I) = TCUT - TTT(IBP(I))

```

```

586:          if ( TI(I).lt.0.d0 ) TI(I) = 0.d0
587: 300      continue
588: C
589:          do 330 K = 1, KS
590:            IKPID = KSORT(K)
591:            if ( IKPID.eq.KPNEUT ) then
592:              do 310 I = IKSORT(K), IKSORT(K+1) - 1
593:                DDE = EEE(IBP(I))
594:                VI(I) = CLIGHT*SQRT(DDE*(DDE+2*MC2)) / (DDE+MC2)
595: 310          continue
596:            else if ( IKPID.eq.KPPHOT ) then
597:              do 320 I = IKSORT(K), IKSORT(K+1) - 1
598:                VI(I) = CLIGHT
599: 320          continue
600:            end if
601: 330          continue
602: C
603: C =====
604: C ===== WHEN HEXAGONAL LATTICE USED =====
605: C =====
606: C =====
607: C =====
608: C
609: C
610:          if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
611: C
612:            INZ1 = NNXT(NZONE+1)
613: C
614: C -----
615: C .... MAT > 0 : REAL MEDIUM .....
616: C -----
617: C
618:          if ( MAT.gt.0.and.JNCOL.eq.0 ) then
619:            INXT = 0
620:            IHH = 0
621: C
622: C -----
623: C NEUTRON/PHOTON COUPLED
624: C -----
625: C
626:          if ( NPKIND.gt.1 ) then
627: C
628: *VOCL LOOP,NOVREC
629:          do 340 I = 1, ISAFE
630: C
631:            D1 = DLOC1(I)
632:            D2 = TI(I)*VI(I)
633:            if ( JPTIM.ne.0 ) D2 = DINF
634: C
635: C ----- TIME cutoff -----
636: C
637:          if ( D2.le.MIN(DI(I),D1) ) then
638:            DI(I) = D2
639:            NDEAD = NDEAD + 1
640:            LSDED(NDEAD) = IBP(I)
641:            if ( KKP(IBP(I)).eq.KPNEUT ) then
642:              NCNTR(26,KPNEUT) = NCNTR(26,KPNEUT) + 1
643:              WCNTR(26,KPNEUT) = WCNTR(26,KPNEUT) + W(I)
644:            else
645:              NCNTR(26,KPPHOT) = NCNTR(26,KPPHOT) + 1
646:              WCNTR(26,KPPHOT) = WCNTR(26,KPPHOT) + W(I)
647:            end if
648: c##<2007/03/14:PN3:
649:          if ( JMNTR.ne.0 ) then
650:            IGT = IGG(IBP(I))

```

src/mvp/fldist.f

```

651:         if ( JTLLT.eq.0 ) then
652:             IRT      = KZREG(IZZ(IBP(I)))
653:         else
654:             IRT      = IBREG(IBP(I))
655:         end if
656:         NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
657:         WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
658:     end if
659:     if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
660:     if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
661:     if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
662: c##>
663: c##<2007/03/14:PN4:
664:         if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
665: c##>
666: C
667: C ----- COLLISION -----
668: C
669:         else if ( DI(I).gt.D1 ) then
670:             DI(I) = D1
671:             if ( KKP(IBP(I)).eq.KPNEUT ) then
672:                 ICOLS = ICOLS + 1
673:                 LSCOL(NCOLS+ICOLS) = IBP(I)
674:             else
675:                 ICOLP = ICOLP + 1
676:                 LSCLP(NCOLP+ICOLP) = IBP(I)
677:             end if
678: C
679:             IKL2(I) = 0
680: C
681: C     ... flight path count is cleared for collision
682:             if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
683: C     ... NND type for next STG sampling
684:             if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
685: C
686: C ----- REACHED ZONE BOUNDARY -----
687: C
688: C
689:         else
690:             IZNN      = MZONE
691: C
692: C     .....IHF = 1 : ZONE 'MZONE' IS IN A HEXAGONAL CELL. ....
693: C
694: C
695: C     ... reached lattice boundary ...
696: C
697:         if ( IHF.ne.0.and.IFC(I).ne.0 ) then
698:             IHH      = IHH + 1
699:             IZNN      = IFC(I)
700: C
701: C     CAUTION : IZZ (ZONE #) DOES NOT MATCH MZONE IN THIS CASE
702: C     TO THE END OF THIS ROUTINE.
703: C
704:             IZZ(IBP(I)) = IFC(I)
705:             LEVL(IBP(I)) = ILUP(I)
706: C     .... make IFC(I) negative to calculate NNXT
707:             IFC(I) = -IFC(I)
708: C
709: C     .... COORDINATES & DIRECTION IN UPPER LEVEL
710: C
711:             X(I)      = XUP(I)
712:             Y(I)      = YUP(I)
713:             Z(I)      = ZUP(I)
714:             AI(I)      = AUP(I)
715:             BI(I)      = BUP(I)

```

```

716:             CI(I)      = CUP(I)
717:             IKL2(I) = 0
718:         end if
719: C
720:             INXT      = INXT + 1
721:             LSSRC(INZ1+INXT) = IBP(I)
722:             IZNX1(INZ1+INXT) = IZNN
723:         end if
724:         340          continue
725: C
726: C -----
727: C     NON-NEUTRON/PHOTON COUPLED PROBLEM
728: C -----
729: C
730:         else
731:             *VOCL LOOP,NOVREC
732:             do 380 I = 1, ISAFE
733: C
734:                 D1      = DLOC1(I)
735:                 D2      = TI(I)*VI(I)
736:                 if ( JPTIM.ne.0 ) D2      = DINF
737: C
738: C ----- TIME cutoff -----
739: C
740:             if ( D2.le.MIN(DI(I),D1) ) then
741:                 DI(I) = D2
742:                 NDEAD = NDEAD + 1
743:                 LSDED(NDEAD) = IBP(I)
744:                 NCNTR(26,IPART) = NCNTR(26,IPART) + 1
745:                 WCNTR(26,IPART) = WCNTR(26,IPART) + W(I)
746: c##<2007/03/14:PN3:
747:             if ( JMNTR.ne.0 ) then
748:                 IGT      = IGG(IBP(I))
749:                 if ( JTLLT.eq.0 ) then
750:                     IRT      = KZREG(IZZ(IBP(I)))
751:                 else
752:                     IRT      = IBREG(IBP(I))
753:                 end if
754:                 NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
755:                 WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
756:             end if
757:             if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
758:             if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
759:             if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
760: c##>
761: c##<2007/03/14:PN4:
762:             if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
763: c##>
764: C
765: C ----- COLLISION -----
766: C
767:         else if ( DI(I).gt.D1 ) then
768:             DI(I) = D1
769:             if ( JKPAR(KPNEUT).ne.0 ) then
770:                 ICOLS = ICOLS + 1
771:                 LSCOL(NCOLS+ICOLS) = IBP(I)
772:             else if ( JKPAR(KPPHOT).ne.0 ) then
773:                 ICOLP = ICOLP + 1
774:                 LSCLP(NCOLP+ICOLP) = IBP(I)
775:             end if
776: C
777: C     ... flight path count is cleared for collision
778:             if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
779: C     ... NND type for next STG sampling
780:             if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2

```

src/mvp/fldist.f

```

781: C
782: C
783: C
784: C ----- REACHED ZONE BOUNDARY -----
785: C
786: C
787: C      else
788: C          IZNN      = MZONE
789: C
790: CCCCC ---- IHF = 1 : ZONE 'MZONE' IS IN A HEXAGONAL CELL. ----
791: C
792: C ..... REACHED LATTICE BOUNDARY .....
793: C
794: C      if ( IHF.ne.0.and.IFC(I).ne.0 ) then
795: C          IHH      = IHH + 1
796: C          IZNN      = IFC(I)
797: C          IZZ(IBP(I)) = IFC(I)
798: C          LEVL(IBP(I)) = ILUP(I)
799: C          .... make IFC(I) negative to calculate NNXT
800: C          IFC(I) = -IFC(I)
801: C          X(I) = XUP(I)
802: C          Y(I) = YUP(I)
803: C          Z(I) = ZUP(I)
804: C          AI(I) = AUP(I)
805: C          BI(I) = BUP(I)
806: C          CI(I) = CUP(I)
807: C          IKL2(I) = 0
808: C      end if
809: C
810: C          INXT      = INXT + 1
811: C          LSSRC(INZ1+INXT) = IBP(I)
812: C          IZNXT(INZ1+INXT) = IZNN
813: C      end if
814: C      380      continue
815: C      end if
816: C
817: C          NNXT(MZONE) = NNXT(MZONE) + INXT - IHH
818: C
819: C -----
820: C      update NNXT, LPOS, IBSPC if IHH>0
821: C -----
822: C
823: C      if ( IHH.gt.0 ) then
824: C          *VOCL LOOP,SCALAR
825: C          do 390 I = 1, ISAFE
826: C              if ( IFC(I).lt.0 ) then
827: C                  NNXT(-IFC(I)) = NNXT(-IFC(I)) + 1
828: C              end if
829: C          390      continue
830: C
831: C          ... "cell position" LPOS is changed here for STG cell
832: C
833: C          if ( JJJNND.ne.0 ) then
834: C          *VOCL LOOP,NOVREC
835: C          do 400 I = 1, ISAFE
836: C              if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
837: C                  LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
838: C              end if
839: C          400      continue
840: C          end if
841: C          if ( JTLLT.ne.0.and.JJJNND.ne.0 ) then
842: C          *VOCL LOOP,NOVREC
843: C          do 410 I = 1, ISAFE
844: C              if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
845: C                  IBSPC(IBP(I),LEVL(IBP(I))) =

```

```

846: C      &          KTCSP(
847: C      &          KSPSU(KZREG(MZONE),
848: C      &          IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
849: C      end if
850: C      410      continue
851: C      end if
852: C      end if
853: C
854: C -----
855: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
856: C -----
857: C
858: C      else
859: C          INXT      = 0
860: C          if ( IHF.eq.0 ) then
861: C              do 420 I = 1, ISAFE
862: C                  D2      = TI(I)*VI(I)
863: C                  if ( JPTIM.ne.0 ) D2      = DINF
864: C
865: C ----- TIME cutoff -----
866: C
867: C          if ( D2.le.DI(I) ) then
868: C              DI(I) = D2
869: C              NDEAD = NDEAD + 1
870: C              LSDED(NDEAD) = IBP(I)
871: C              if ( KKP(IBP(I)).eq.KPNEUT ) then
872: C                  NCNTR(26,KPNEUT) = NCNTR(26,KPNEUT) + 1
873: C                  WCNTR(26,KPNEUT) = WCNTR(26,KPNEUT) + W(I)
874: C              else
875: C                  NCNTR(26,KPPHOT) = NCNTR(26,KPPHOT) + 1
876: C                  WCNTR(26,KPPHOT) = WCNTR(26,KPPHOT) + W(I)
877: C              end if
878: C          c##<2007/03/14:PN3:
879: C          if ( JMNTR.ne.0 ) then
880: C              IGT      = IGG(IBP(I))
881: C              if ( JTLLT.eq.0 ) then
882: C                  IRT      = KZREG(IZZ(IBP(I)))
883: C              else
884: C                  IRT      = IBREG(IBP(I))
885: C              end if
886: C              NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
887: C              WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
888: C          end if
889: C          if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
890: C          if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
891: C          if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
892: C          c##>
893: C          c##<2007/03/14:PN4:
894: C          if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
895: C          c##>
896: C          else
897: C              INXT      = INXT + 1
898: C              LSSRC(INZ1+INXT) = IBP(I)
899: C              IZNXT(INZ1+INXT) = MZONE
900: C          end if
901: C          420      continue
902: C          NNXT(MZONE) = NNXT(MZONE) + INXT
903: C          else
904: C              IHFFF      = 0
905: C          *VOCL LOOP,NOVREC
906: C          do 430 I = 1, ISAFE
907: C              D2      = TI(I)*VI(I)
908: C              if ( JPTIM.ne.0 ) D2      = DINF
909: C
910: C ----- TIME cutoff -----

```

src/mvp/fldist.f

```

911: C
912:       if ( D2.le.DI(I) ) then
913:         DI(I) = D2
914:         NDEAD = NDEAD + 1
915:         LSDED(NDEAD) = IBP(I)
916:         if ( KKP(IBP(I)).eq.KPNEUT ) then
917:           NCNTR(26,KPNEUT) = NCNTR(26,KPNEUT) + 1
918:           WCNTR(26,KPNEUT) = WCNTR(26,KPNEUT) + W(I)
919:         else
920:           NCNTR(26,KPPHOT) = NCNTR(26,KPPHOT) + 1
921:           WCNTR(26,KPPHOT) = WCNTR(26,KPPHOT) + W(I)
922:         end if
923: c##<2007/03/14:PN3:
924:       if ( JMNTR.ne.0 ) then
925:         IGT = IGG(IBP(I))
926:         if ( JTLLT.eq.0 ) then
927:           IRT = KZREG(IZZ(IBP(I)))
928:         else
929:           IRT = IBREG(IBP(I))
930:         end if
931:         NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
932:         WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
933:       end if
934:       if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
935:       if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
936:       if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
937: c##>
938: c##<2007/03/14:PN4:
939:       if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
940: c##>
941:     else
942:       IZNN = MZONE
943:       if ( IFC(I).ne.0 ) then
944:         IZNN = IFC(I)
945:         X(I) = XUP(I)
946:         Y(I) = YUP(I)
947:         Z(I) = ZUP(I)
948:         AI(I) = AUP(I)
949:         BI(I) = BUP(I)
950:         CI(I) = CUP(I)
951:         IKL2(I) = 0
952: C
953:       IHHHH = IHHHH + 1
954:       IZZ(IBP(I)) = IFC(I)
955:       LEVL(IBP(I)) = ILUP(I)
956: C       .... make IFC(I) negative to calculate NNXT
957:       IFC(I) = -IFC(I)
958:       end if
959:       INXT = INXT + 1
960:       LSSRC(INZ1+INXT) = IBP(I)
961:       IZNXT(INZ1+INXT) = IZNN
962:     end if
963: 430     continue
964:     if ( IHHHH.gt.0 ) then
965: *VOCL LOOP,SCALAR
966:       do 440 I = 1, INXT
967:         if ( IFC(I).lt.0 ) then
968:           NNXT(-IFC(I)) = NNXT(-IFC(I)) + 1
969:         end if
970:       continue
971: C
972: C       ... "cell position" LPOS is changed here for STG cell
973: C
974:       if ( JJJNND.ne.0 ) then
975: *VOCL LOOP,NOVREC

```

```

976:       do 450 I = 1, ISAFE
977:         if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
978:           LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
979:         end if
980:       continue
981:     end if
982:     if ( JTLLT.ne.0.and.JJJNND.ne.0 ) then
983: *VOCL LOOP,NOVREC
984:       do 460 I = 1, ISAFE
985:         if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
986:           IBSPC(IBP(I),LEVL(IBP(I))) =
987:             KTCSP(
988:               KSPSU(KZREG(MZONE),
989:                 IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
990:         end if
991:       continue
992:     end if
993:     end if
994:     NNXT(MZONE) = NNXT(MZONE) + INXT - IHHHH
995:   end if
996: end if
997: C
998: C
999:   if ( JKPAR(KPNEUT).ne.0 ) NCOLS = NCOLS + ICOLS
1000:   if ( JKPAR(KPPHOT).ne.0 ) NCOLP = NCOLP + ICOLP
1001:   NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
1002: C
1003: C =====
1004: C =====
1005: C ===== WHEN NO-HEXAGONAL LATTICE USED =====
1006: C =====
1007: C =====
1008: C
1009:   else
1010:     INXT = 0
1011: C
1012: C-----
1013: C .... MAT > 0 : REAL MEDIUM .....
1014: C-----
1015: C
1016:       if ( MAT.gt.0.and.JNCOL.eq.0 ) then
1017: C-----
1018: C-----
1019: C       NEUTRON/PHOTON COUPLED PROBLEM
1020: C-----
1021: C
1022: C
1023:       if ( NPKIND.gt.1 ) then
1024: *VOCL LOOP,NOVREC
1025:         do 470 I = 1, ISAFE
1026: C
1027:           D1 = DLOC1(I)
1028:           D2 = TI(I)*VI(I)
1029:           if ( JPTIM.ne.0 ) D2 = DINF
1030: C
1031: C ----- TIME cutoff -----
1032: C
1033:           if ( D2.le.MIN(DI(I),D1) ) then
1034:             DI(I) = D2
1035:             NDEAD = NDEAD + 1
1036:             LSDED(NDEAD) = IBP(I)
1037:             if ( KKP(IBP(I)).eq.KPNEUT ) then
1038:               NCNTR(26,KPNEUT) = NCNTR(26,KPNEUT) + 1
1039:               WCNTR(26,KPNEUT) = WCNTR(26,KPNEUT) + W(I)
1040:             else

```

src/mvp/fldist.f

```

1041:          NCNTR(26,KPPHOT) = NCNTR(26,KPPHOT) + 1
1042:          WCNTR(26,KPPHOT) = WCNTR(26,KPPHOT) + W(I)
1043:        end if
1044: c##<2007/03/14:PN3:
1045:        if ( JMNTR.ne.0 ) then
1046:          IGT = IGG(IBP(I))
1047:          if ( JTLLT.eq.0 ) then
1048:            IRT = KZREG(IZZ(IBP(I)))
1049:          else
1050:            IRT = IBREG(IBP(I))
1051:          end if
1052:          NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
1053:          WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
1054:        end if
1055:        if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
1056:        if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
1057:        if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
1058: c##>
1059: c##<2007/03/14:PN4:
1060:        if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
1061: c##>
1062: C
1063: C ----- COLLISION -----
1064: C
1065:        else if ( DI(I).gt.D1 ) then
1066:          DI(I) = D1
1067:          if ( KKP(IBP(I)).eq.KPNEUT ) then
1068:            ICOLS = ICOLS + 1
1069:            LSCOL(NCOLS+ICOLS) = IBP(I)
1070:          else
1071:            ICOLP = ICOLP + 1
1072:            LSCLP(NCOLP+ICOLP) = IBP(I)
1073:          end if
1074:          IKL2(I) = 0
1075: C
1076: C ... flight path count is cleared for collision
1077:        if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
1078: C ... NND type for next STG sampling
1079:        if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
1080: C
1081: C
1082: C ----- REACHED ZONE BOUNDARY -----
1083: C
1084:        else
1085:          INXT = INXT + 1
1086:          LSSRC(NNXT(NZONE+1)+INXT) = IBP(I)
1087:        end if
1088: 470        continue
1089: C
1090: C -----
1091: C NON NEUTRON/PHOTON COUPLED PROBLEM ====
1092: C -----
1093: C
1094:        else
1095:          *VOCL LOOP,NOVREC
1096:          do 480 I = 1, ISAFE
1097: C
1098:            D1 = DLOC1(I)
1099:            D2 = TI(I)*VI(I)
1100:            if ( JPTIM.ne.0 ) D2 = DINF
1101: C
1102: C ----- TIME cutoff -----
1103: C
1104:            if ( D2.le.MIN(DI(I),D1) ) then
1105:              DI(I) = D2

```

```

1106:          NDEAD = NDEAD + 1
1107:          LSEDED(NDEAD) = IBP(I)
1108:          NCNTR(26,IPART) = NCNTR(26,IPART) + 1
1109:          WCNTR(26,IPART) = WCNTR(26,IPART) + W(I)
1110: c##<2007/03/14:PN3:
1111:        if ( JMNTR.ne.0 ) then
1112:          IGT = IGG(IBP(I))
1113:          if ( JTLLT.eq.0 ) then
1114:            IRT = KZREG(IZZ(IBP(I)))
1115:          else
1116:            IRT = IBREG(IBP(I))
1117:          end if
1118:          NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
1119:          WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
1120:        end if
1121:        if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
1122:        if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
1123:        if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
1124: c##>
1125: c##<2007/03/14:PN4:
1126:        if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
1127: c##>
1128: C
1129: C ----- COLLISION -----
1130: C
1131:        else if ( DI(I).gt.D1 ) then
1132:          DI(I) = D1
1133:          if ( JKPAR(KPNEUT).ne.0 ) then
1134:            ICOLS = ICOLS + 1
1135:            LSCOL(NCOLS+ICOLS) = IBP(I)
1136:          else if ( JKPAR(KPPHOT).ne.0 ) then
1137:            ICOLP = ICOLP + 1
1138:            LSCLP(NCOLP+ICOLP) = IBP(I)
1139:          end if
1140:          IKL2(I) = 0
1141: C
1142: C ... flight path count is cleared for collision
1143:        if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
1144: C ... NND type for next STG sampling
1145:        if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
1146: C
1147: C
1148: C ----- REACHED ZONE BOUNDARY -----
1149: C
1150:        else
1151:          INXT = INXT + 1
1152:          LSSRC(NNXT(NZONE+1)+INXT) = IBP(I)
1153:        end if
1154: 480        continue
1155:        end if
1156: C
1157: C -----
1158: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
1159: C -----
1160: C
1161:        else
1162:          INXT = 0
1163:          do 490 I = 1, ISAFE
1164:            D2 = TI(I)*VI(I)
1165:            if ( JPTIM.ne.0 ) D2 = DINF
1166: C
1167: C ----- TIME cutoff -----
1168: C
1169:            if ( D2.le.DI(I) ) then
1170:              DI(I) = D2

```

src/mvp/fldist.f

```
1171:      NDEAD = NDEAD + 1
1172:      LSDED(NDEAD) = IBP(I)
1173:      if ( KKP(IBP(I)).eq.KPNEUT ) then
1174:         NCNTR(26,KPNEUT) = NCNTR(26,KPNEUT) + 1
1175:         WCNTR(26,KPNEUT) = WCNTR(26,KPNEUT) + W(I)
1176:      else
1177:         NCNTR(26,KPPHOT) = NCNTR(26,KPPHOT) + 1
1178:         WCNTR(26,KPPHOT) = WCNTR(26,KPPHOT) + W(I)
1179:      end if
1180: c##<2007/03/14:PN3:
1181:      if ( JMNTR.ne.0 ) then
1182:         IGT = IGG(IBP(I))
1183:         if ( JTLT.eq.0 ) then
1184:            IRT = KIREG(IZZ(IBP(I)))
1185:         else
1186:            IRT = IBREG(IBP(I))
1187:         end if
1188:         NTCUT(IGT,IRT) = NTCUT(IGT,IRT) + 1
1189:         WTCUT(IGT,IRT) = WTCUT(IGT,IRT) + W(I)
1190:      end if
1191:      if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
1192:      if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
1193:      if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
1194: c##>
1195: c##<2007/03/14:PN4:
1196:      if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
1197: c##>
1198:      else
1199:         INXT = INXT + 1
1200:         LSSRC(NNXT(NZONE+1)+INXT) = IBP(I)
1201:      end if
1202: 490      continue
1203:      end if
1204: C
1205:      if ( JKPAR(KPNEUT).ne.0 ) NCOLS = NCOLS + ICOLS
1206:      if ( JKPAR(KPPHOT).ne.0 ) NCOLP = NCOLP + ICOLP
1207: C
1208:      do 500 I = 1, INXT
1209:         IZNXT(NNXT(NZONE+1)+I) = MZONE
1210: 500      continue
1211:         NNXT(MZONE) = NNXT(MZONE) + INXT
1212:         NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
1213:      end if
1214: C
1215: C ... flight time ...
1216: C
1217:      do 510 I = 1, ISAFE
1218:         TI(I) = DI(I) /VI(I)
1219: 510      continue
1220:
1221:
1222:      KK = 1
1223:      do 530 K = 1, KS
1224:         IKPID = KSORT(K)
1225:         do 520 I = IKSORT(K), IKSORT(K+1) - 1
1226:            TFLH(IKPID) = TFLH(IKPID) + TI(I)*W(I)
1227: 520          continue
1228: 530      continue
1229: C
1230:      end if
1231: C
1232:      return
1233:      end
```


src/mvp/flight.f

```

1: C/#!/IF ARGSAVE
2: *      subroutine FLIGH0( IOW, NBANK, NZONE, NSDA, NZDA, NGROUP,NGP1,
3: *      &                NGP2, NREG, NRESP, NCOLS, NDEAD, NLOST,
4: *      &                IRAND, DINF, NEST, NLATT, NLBZ, NMAT, MB,
5: *      &                NUC, NSMAC, NSMIC, NEMIC, NSTAL, XXX, YYY,
6: *      &                ZZZ, AAA, BBB, CCC, WWW, IGG, TTT,
7: *      &                LEVL, LZZ, LPOS, LCRS, SMAC, LMAC, KMAC,
8: *      &                MMAC, SMIC, LMIC, SGTAL, LPDEN, INUCT,
9: *      &                DENST, DNFLX, LSFFL, NFFL, IZFFL, LSCOL,
10: *      &                LSSRC, NNXT, IZNXT, LSDED, SDA, KZMAT,
11: *      &                KZREG, KZDA, KZAA, NKZAA, DALT, KDALT,
12: *      &                NVLAT, SZLAT, IPLAT, KSLAT, LTYP, MLBZZ,
13: *      &                KLATT, CELSZ, RESP, FLTR, RETR, RMIC, RSTR,
14: *      &                NLEAK, WLEAK, NCNTR, WCNTR, DNZON, LEMIC,
15: *      &                IBREG, KKSF, JKSF, IPZONE,X, Y, Z,
16: *      &                AI, BI, CI, W, DI, IGI, IBP,
17: *      &                R, D11, D22, DLOC1, DLOC2, LLS, IZT,
18: *      &                DSDA0, DSDA1, DSDA2, DSDA3, DSDA4, DSDA5,
19: *      &                DSDA6, DSDA7, DSDA8, DSDA9 )
20: C/#!/ELSE
21: *      subroutine FLIGHT( IOW, NBANK, NZONE, NSDA, NZDA, NGROUP,NGP1,
22: *      &                NGP2, NREG, NRESP, NCOLS, NDEAD, NLOST,
23: *      &                IRAND, DINF, NEST, NLATT, NLBZ, NMAT, MB,
24: *      &                NUC, NSMAC, NSMIC, NEMIC, NSTAL, XXX, YYY,
25: *      &                ZZZ, AAA, BBB, CCC, WWW, IGG, TTT,
26: *      &                LEVL, LZZ, LPOS, LCRS, SMAC, LMAC, KMAC,
27: *      &                MMAC, SMIC, LMIC, SGTAL, LPDEN, INUCT,
28: *      &                DENST, DNFLX, LSFFL, NFFL, IZFFL, LSCOL,
29: *      &                LSSRC, NNXT, IZNXT, LSDED, SDA, KZMAT,
30: *      &                KZREG, KZDA, KZAA, NKZAA, DALT, KDALT,
31: *      &                NVLAT, SZLAT, IPLAT, KSLAT, LTYP, MLBZZ,
32: *      &                KLATT, CELSZ, RESP, FLTR, RETR, RMIC, RSTR,
33: *      &                NLEAK, WLEAK, NCNTR, WCNTR, DNZON, LEMIC,
34: *      &                IBREG, KKSF, JKSF, IPZONE,X, Y, Z,
35: *      &                AI, BI, CI, W, DI, IGI, IBP,
36: *      &                R, D11, D22, DLOC1, DLOC2, LLS, IZT,
37: *      &                DSDA0, DSDA1, DSDA2, DSDA3, DSDA4, DSDA5,
38: *      &                DSDA6, DSDA7, DSDA8, DSDA9 )
39: C/#!/ENDIF
40:
41: C===== < FLIGHT >=====
42: C PURPOSE:  FREE-FLIGHT (ALL ZONE LOGIC )
43: C CALLED IN: ACTION
44: C CALLS:    RANU2,VMNTR1
45: C=====
46:
47:      implicit real*8(D)
48:
49:      include 'INC/_FLAGS'
50:      include '../shared/INC/_TASKDT'
51: C
52: C .... PARTICLE BANK .....
53: C
54:      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
55:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
56:      real WWW(NBANK)
57:      real*8 TTT(NBANK)
58:      integer IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
59:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KMAC(NBANK,MB,2),
60:      &      MMAC(NBANK,2), LMAC(8), LMIC(8), LEMIC(16)
61:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
62:      &      SGTAL(NBANK,NSTAL)
63:      integer IBREG(NBANK)
64: C
65: C .... STACKS .....

```

```

66: C
67:      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSCOL(NBANK),
68:      &      LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK)
69: C
70: C .... GEOMETRY .....
71: C
72:      real*8 SDA(NSDA), DALT(*), SZLAT(8,NLATT), CELSZ(6,*)
73:      integer KZMAT(NZONE), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
74:      &      NKZAA(NZONE), KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT),
75:      &      IPLAT(NLATT+1), KSLAT(*), KLATT(*), NVLAT(4,NLATT)
76: C
77: C .... MATERIAL DATA .....
78: C
79:      real DENST(*)
80:      integer INUCT(*), LPDEN(NMAT+1)
81: C
82: C .... TALLY .....
83: C
84:      real RESP(NGROUP,NRESP), DNZON(NUC,NZONE)
85:      real*8 FLTR(NGROUP,NREG), RETR(NREG,NRESP), WCNTR(*),
86:      &      WLEAK(NGROUP,NREG), RMIC(NGROUP,NREG,NUC,NEMIC),
87:      &      RSTR(NGROUP,NSTAL), NCNTR(*)
88:      real*8 DNFLX(NGROUP,NREG,NUC)
89:      real*8 NLEAK(NGROUP,NREG)
90: C
91: C .... WORKING AREA .....
92: C
93:      real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
94:      &      CI(NBANK), W(NBANK), DI(NBANK), DSDA0(NBANK),
95:      &      DSDA1(NBANK), DSDA2(NBANK), DSDA3(NBANK), DSDA4(NBANK),
96:      &      DSDA5(NBANK), DSDA6(NBANK), DSDA7(NBANK), DSDA8(NBANK),
97:      &      DSDA9(NBANK), D11(NBANK), D22(NBANK), DLOC1(NBANK),
98:      &      DLOC2(NBANK)
99:      real R(NBANK)
100:      integer IGI(NBANK), IBP(NBANK), LLS(NBANK), KKSF(NZONE+1),
101:      &      IZT(NBANK), JKSF(NZONE+1), IPZONE(NZONE+1)
102:
103: C/#!/IF ARGSAVE
104: *      return
105: C -----
106: *      entry FLIGHT
107: C/#!/ENDIF
108:
109:      if ( JVMNT.ne.0 ) call VMNTR1( 3, NFFL(NZONE+1) )
110: C
111: C -----
112: C .... GATHER VECTORS .....
113: C -----
114:
115:      100 continue
116:      NZK = 0
117:      do 110 K = 1, NZONE
118:      if ( NFFL(K).ne.0 ) then
119:      NZK = NZK + 1
120:      JKSF(NZK) = K
121:      end if
122:      110 continue
123: C
124:      if ( NZK.gt.20 ) then
125:      IPZONE(1) = 1
126:
127: *VOCL LOOP,SCALAR
128:      do 120 NK = 1, NZK
129:      IPZONE(NK+1) = IPZONE(NK) + NFFL(JKSF(NK))
130:      120 continue

```

src/mvp/flight.f

```

131:
132: *VOCL LOOP,NOVREC
133:   do 130 NK = 1, NZK
134:     KKSF(JKSF(NK)) = IPZONE(NK) - 1
135:   130   continue
136:     do 140 I = 1, NFFL(NZONE+1)
137:       KKSF(IZFFL(I)) = KKSF(IZFFL(I)) + 1
138:       IGI(I) = KKSF(IZFFL(I))
139:   140   continue
140:
141: *VOCL LOOP,NOVREC
142:   do 150 I = 1, NFFL(NZONE+1)
143:     IBP(IGI(I)) = LSFFL(I)
144:     IZT(IGI(I)) = IZFFL(I)
145:   150   continue
146:   else
147:     IPZONE(1) = 1
148:     do 160 NK = 1, NZK
149:       IPZONE(NK+1) = IPZONE(NK) + NFFL(JKSF(NK))
150:   160   continue
151:     KKK = 0
152:     do 180 NK = 1, NZK
153:       do 170 I = 1, NFFL(NZONE+1)
154:         if ( IZFFL(I).eq.JKSF(NK) ) then
155:           KKK = KKK + 1
156:           IBP(KKK) = LSFFL(I)
157:           IZT(KKK) = IZFFL(I)
158:         end if
159:       170   continue
160:     180   continue
161:   end if
162: C
163: C .....
164: C
165:   do 190 I = 1, NFFL(NZONE+1)
166:     LSFFL(I) = IBP(I)
167:     IZFFL(I) = IZT(I)
168:     X(I) = XXX(LSFFL(I))
169:     Y(I) = YYY(LSFFL(I))
170:     Z(I) = ZZZ(LSFFL(I))
171:     AI(I) = AAA(LSFFL(I))
172:     BI(I) = BBB(LSFFL(I))
173:     CI(I) = CCC(LSFFL(I))
174:     WI(I) = WWW(LSFFL(I))
175:     IGI(I) = IGG(LSFFL(I))
176:     DI(I) = DINF
177:     DLOC1(I) = -DINF
178:     DLOC2(I) = DINF
179:   190   continue
180: C
181: C-----
182: C  PREPARE MACRO CROSS SECTION  ( TOTAL )
183: C-----
184:   do 210 NK = 1, NZK
185:     MAT = KZMAT(JKSF(NK))
186:     if ( MAT.gt.0 ) then
187:       NN = IPZONE(NK+1) - IPZONE(NK)
188:       if ( JVMNT.ne.0 ) call VMNT00( 9, 3 )
189:       call GETMAC( IRAND, NN, IBP(IPZONE(NK)), MAT, JEIGN, JDEBG,
190: &               NBANK, MB, NUC, NSTAL, NSMAC, SMAC, LMAC, KMAC,
191: &               MMAC, NSMIC, SMIC, LMIC, SGTAL, LPDEN, INUCT, DENST,
192: &               NMAT, DSDA0, DSDA1, DSDA2, DSDA3, DSDA4, DSDA5 )
193:       if ( JVMNT.ne.0 ) call VMNT22( 9, 3 )
194:     else
195:       do 200 I = IPZONE(NK), IPZONE(NK+1) - 1

```

```

196:         MMAC(IBP(I),1) = 0
197:       200   continue
198:     end if
199:   210   continue
200: C-----
201: C  .... CALCULATE MAXIMUM SURFACE NUMBER OF ZONES .....
202: C-----
203:     MSF = 0
204:     do 220 NK = 1, NZK
205:       K = JKSF(NK)
206:       MSF = MAX(NKZAA(K),MSF)
207:   220   continue
208: C=====
209: C  .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
210: C=====
211:     do 390 N = 1, MSF
212: C-----
213: C  .... GATHER DATA OF SURFACES .....
214: C-----
215:     do 270 NK = 1, NZK
216:       KKSF(NK) = 0
217:       K = JKSF(NK)
218:       KK = NKZAA(K)
219:       if ( KK.ge.N ) then
220:         LS = KZDA(1,KZAA(K)+N-1)
221:       220   C  .... 0/1/2 = SIMPLE / ONE OF A BODY / LAST SURFACE OF A BODY
222:       223   C
224:         LLG = KZDA(2,KZAA(K)+N-1)
225:       225   C
226:       C  .... SIGN: (+) .... INSIDE OF A SURFACE (-) ... OUTSIDE
227:       227   C
228:         LSG = SIGN(1,LS)
229:         LSP = ABS(LS)
230:         KSF = INT(ABS(SDA(LSP)))
231:         KKSF(NK) = KSF
232:         if ( KSF.eq.1 ) then
233:           do 230 I = IPZONE(NK), IPZONE(NK+1) - 1
234:             LLS(I) = LSG
235:             IZT(I) = LLG
236:             DSDA0(I) = SDA(LSP+1)
237:             DSDA1(I) = SDA(LSP+2)
238:             DSDA2(I) = SDA(LSP+3)
239:             DSDA3(I) = SDA(LSP+4)
240:             DSDA4(I) = SDA(LSP+5)
241:       230   continue
242:         else if ( KSF.eq.2 .or. KSF.eq.5 ) then
243:           do 240 I = IPZONE(NK), IPZONE(NK+1) - 1
244:             LLS(I) = LSG
245:             IZT(I) = LLG
246:             DSDA0(I) = SDA(LSP+1)
247:             DSDA1(I) = SDA(LSP+2)
248:             DSDA2(I) = SDA(LSP+3)
249:             DSDA3(I) = SDA(LSP+4)
250:       240   continue
251:         else if ( KSF.eq.3 ) then
252:           do 250 I = IPZONE(NK), IPZONE(NK+1) - 1
253:             LLS(I) = LSG
254:             IZT(I) = LLG
255:             DSDA0(I) = SDA(LSP+1)
256:             DSDA1(I) = SDA(LSP+2)
257:             DSDA2(I) = SDA(LSP+3)
258:             DSDA3(I) = SDA(LSP+4)
259:             DSDA4(I) = SDA(LSP+5)
260:             DSDA5(I) = SDA(LSP+6)

```

src/mvp/flight.f

```

261:          DSDA6(I) = SDA(LSP+7)
262: 250      continue
263:      else if ( KSF.eq.4 ) then
264:          do 260 I = IPZONE(NK), IPZONE(NK+1) - 1
265:              LLS(I) = LSG
266:              IZT(I) = LLG
267:              DSDA0(I) = SDA(LSP+1)
268:              DSDA1(I) = SDA(LSP+2)
269:              DSDA2(I) = SDA(LSP+3)
270:              DSDA3(I) = SDA(LSP+4)
271:              DSDA4(I) = SDA(LSP+5)
272:              DSDA5(I) = SDA(LSP+6)
273:              DSDA6(I) = SDA(LSP+7)
274:              DSDA7(I) = SDA(LSP+8)
275:              DSDA8(I) = SDA(LSP+9)
276:              DSDA9(I) = SDA(LSP+10)
277: 260      continue
278:      end if
279:  end if
280: 270      continue
281: C
282: C-----
283: C ..... D11: SMALLER DISTANCE      D22: LARGER DISTANCE
284: C ..... DLOC1: MAXIMUM OF SMALLER DISTANCE TO A BODY
285: C ..... DLOC2: MINIMUM OF LARGER DISTANCE TO A BODY
286: C
287: C ..... IF A SURFACE HAS NO CROSSING POINT WITH A PARTICLE TRACK
288: C "MINIMUM DISTANCE (D11)" BECOMES GREATER THAN "MAXIMUM DISTANCE
289: C (D22)" AND THE CODE JUDGES THAT A BODY INCLUDING THE
290: C SURFACE HAS NO CROSSING POINT WITH THE PARTICLE TRACK.
291: C-----
292:      do 280 I = 1, NFFL(NZONE+1)
293:          D11(I) = -DINF
294:          D22(I) = DINF
295: 280      continue
296:          KKK = 1
297: C
298: 290      KSF = KKSF(KKK)
299:          do 300 NK = KKK, NZK
300:              if ( KKSF(NK).ne.KSF ) go to 310
301: 300      continue
302:          NK = NZK + 1
303: 310      KKK2 = NK
304: C-----
305: C ..... SLAB .....
306: C-----
307:      if ( KSF.eq.1 ) then
308:          do 320 I = IPZONE(KKK), IPZONE(KKK2) - 1
309:              DUV = DSDA0(I)*AI(I) + DSDA1(I)*BI(I) + DSDA2(I)*
310:              &      CI(I)
311:              DUP = -(DSDA0(I)*X(I)+DSDA1(I)*Y(I)+DSDA2(I)*Z(I))
312:              D1 = DSDA3(I) + DUP
313:              D2 = DSDA4(I) + DUP
314:              if ( DUV.ne.0.0 ) then
315:                  D1 = D1/DUV
316:                  D2 = D2/DUV
317:                  D11(I) = MIN(D1,D2)
318:                  D22(I) = MAX(D1,D2)
319:
320: C ..... NEVER CROSSING CASE WITH BODY ( IZT(I) > 0 ) .....
321: C (PARTICLE POSITION IS OUTSIDE OF SLAB)
322: *VOCL STMT,IF(0)
323:      else if ( IZT(I).gt.0 ) then
324:          if ( D1*D2.ge.0.0 ) then
325:              D11(I) = DINF

```

```

326:          D22(I) = -DINF
327:      end if
328:  end if
329: 320      continue
330:  end if
331: C-----
332: C ..... SPHERE .....
333: C-----
334:      if ( KSF.eq.2 ) then
335:          do 330 I = IPZONE(KKK), IPZONE(KKK2) - 1
336:              DPCX = X(I) - DSDA0(I)
337:              DPCY = Y(I) - DSDA1(I)
338:              DPCZ = Z(I) - DSDA2(I)
339:              DVPC = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
340:              DH = DSDA3(I) - DPCX*DPCX - DPCY*DPCY - DPCZ*DPCZ
341:              DD = DVPC*DVPC + DH
342:              if ( DD.ge.0.0 ) then
343:                  D11(I) = -DVPC - SQRT(DD)
344:                  D22(I) = -D11(I) - 2.0D0*DVPC
345:              else if ( IZT(I).gt.0 ) then
346: C-----
347: C ..... NEVER CROSSING CASE WITH BODY ( IZT(I) > 0 ) .....
348: C (PARTICLE POSITION IS OUTSIDE OF SPHERE)
349: C-----
350:                  D11(I) = DINF
351:                  D22(I) = -DINF
352:              end if
353: 330      continue
354:      end if
355: C-----
356: C ..... CYLINDER .....
357: C-----
358:      if ( KSF.eq.3 ) then
359:          do 340 I = IPZONE(KKK), IPZONE(KKK2) - 1
360:              DPCX = X(I) - DSDA0(I)
361:              DPCY = Y(I) - DSDA1(I)
362:              DPCZ = Z(I) - DSDA2(I)
363:              DUV = DSDA3(I)*AI(I) + DSDA4(I)*BI(I) + DSDA5(I)*
364:              &      CI(I)
365:              DA = 1.0D0 - DUV*DUV
366:              DUPC = DSDA3(I)*DPCX + DSDA4(I)*DPCY + DSDA5(I)*DPCZ
367:              DB = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ - DUV*DUPC
368:              DH = DPCX**2 + DPCY**2 + DPCZ**2 - DSDA6(I)
369:              &      - DUPC**2
370:              DD = DB*DB - DA*DH
371:              if ( DA.gt.0.0.and.DD.ge.0.0 ) then
372:                  DD = SQRT(DD)
373:                  D11(I) = (-DB-DD)/DA
374:                  D22(I) = (-DB+DD)/DA
375:              else if ( IZT(I).gt.0 ) then
376:                  if ( DH.ge.0.0 ) then
377:                      D11(I) = DINF
378:                      D22(I) = -DINF
379:                  end if
380:              end if
381: 340      continue
382:      end if
383: C
384: C-----
385: C .... QUADRATIC EXPRESSION ....
386: C S0*X**2 + S1*Y**2 + S2*Z**2 + S3*X*Y + S4*Y*Z + S5*Z*X
387: C S6*X + S7*Y + S8*Z + S9 = 0
388: C-----
389:      if ( KSF.eq.4 ) then
390:          do 350 I = IPZONE(KKK), IPZONE(KKK2) - 1

```

src/mvp/flight.f

```

391: *VOCL STMT,IF(50)
392:   if ( DLOC2(I).gt.0 ) then
393:     DRV1 = DSDA0(I)*AI(I) + DSDA3(I)*BI(I) + DSDA5(I)*
394:       & CI(I)
395:     DRV2 = DSDA1(I)*BI(I) + DSDA4(I)*CI(I)
396:     DRV3 = DSDA2(I)*CI(I)
397:     DRW1 = DSDA0(I)*X(I) + DSDA3(I)*Y(I) + DSDA5(I)*
398:       & Z(I) + DSDA6(I)
399:     DRW2 = DSDA1(I)*Y(I) + DSDA4(I)*Z(I) + DSDA7(I)
400:     DRW3 = DSDA2(I)*Z(I) + DSDA8(I)
401:     DA = AI(I)*DRV1 + BI(I)*DRV2 + CI(I)*DRV3
402:     DB =
403:       & (AI(I)*DRW1+BI(I)*DRW2+CI(I)*DRW3+X(I)*DRV1
404:       & +Y(I)*DRV2+Z(I)*DRV3)*0.5D0
405:     DC = X(I)*DRW1 + Y(I)*DRW2 + Z(I)*DRW3 + DSDA9(I)
406: *VOCL STMT,IF(100)
407:   if ( DA.ne.0.0 ) then
408:     DD = DB**2 - DC*DA
409:     if ( DD.ge.0.0 ) then
410:       DH = SQRT(DD)
411:       D11(I) = (-DB-DH)/DA
412:       D22(I) = (-DB+DH)/DA
413:
414: C ..... CONCAVE SURFACE ( DA < 0, D11 > D22 ) .....
415:
416:       if ( DA.lt.0.0 ) then
417:         if ( D22(I).gt.0.0.and.D22(I).gt.DLOC1(I) )
418:           & then
419:           D11(I) = -DINF
420:         else
421:           D22(I) = DINF
422:         end if
423:       end if
424:
425: C ..... NO REAL ROOT, AND PARTICLE TRACK IS OUTSIDE OF SURFACE
426:
427:       else if ( IZT(I).gt.0.and.DA.gt.0 ) then
428:         D11(I) = DINF
429:         D22(I) = -DINF
430:       end if
431:
432: C ..... ONLY ONE REAL ROOT (DA = 0)
433: *VOCL STMT,IF(0)
434:   else
435:     if ( DB.gt.0.0 ) then
436:       D22(I) = -DC/(DB*2.0D0)
437:     else if ( DB.lt.0.0 ) then
438:       D11(I) = -DC/(DB*2.0D0)
439:
440: C ..... NO ROOT, AND PARTICLE TRACK IS IN OUTSIDE OF THE SURFACE
441: *VOCL STMT,IF(0)
442:   else if ( IZT(I).gt.0.and.DC.ge.0.0 ) then
443:     D11(I) = DINF
444:     D22(I) = -DINF
445:   end if
446: end if
447: end if
448: 350 continue
449: end if
450: C-----
451: C .... PLANE (HALF SPACE) ....
452: C-----
453:   if ( KSF.eq.5 ) then
454:     do 360 I = IPZONE(KKK), IPZONE(KKK2) - 1
455:       DA = DSDA0(I)*AI(I) + DSDA1(I)*BI(I) + DSDA2(I)*

```

```

456:       & CI(I)
457:       = DSDA3(I)
458:       & - (DSDA0(I)*X(I)+DSDA1(I)*Y(I)+DSDA2(I)*Z(I))
459: *VOCL STMT,IF(100)
460:   if ( DA.ne.0.0 ) then
461:     DD = DB/DA
462:     if ( DA.lt.0.0 ) then
463:       D11(I) = DD
464:     else
465:       D22(I) = DD
466:     end if
467: *VOCL STMT,IF(0)
468:   else if ( IZT(I).gt.0.and.DB.le.0.0 ) then
469:     D11(I) = DINF
470:     D22(I) = -DINF
471:   end if
472: 360 continue
473: end if
474: C
475:   KKK = KKK2
476:   if ( KKK.le.NZK ) go to 290
477: C
478: C-----
479: C ..... CALCULATE MINIMUM DISTANCE
480: C-----
481: C
482: C =====
483: C =
484: C = DISTANCE DETERMINATION FOR COMBINATORIAL GEOMETRY (88/4/12)
485: C = VALUES OF IZT(I) (=KZDA(2,NK)) IS ASSIGNED SUCH AS:
486: C =
487: C = A.AND.B.AND. .NOT.( C.AND.D.AND.E ) .AND. .NOT.( F.AND.G )
488: C = | | | | |
489: C = 0 | 0 | 1 | 1 | 2 | | 1 | 2 |
490: C =
491: C = WHERE A, B, C ETC. ARE SURFACES COMPOSING A ZONE.
492: C =
493: C =====
494:   if ( JSIMP.eq.0 ) then
495:     do 370 I = 1, NFFL(NZONE+1)
496:       if ( IZT(I).eq.0 ) then
497:         if ( LLS(I).gt.0.and.D22(I).ge.0.0 ) DI(I) =
498:           & MIN(DI(I),D22(I))
499:         if ( LLS(I).lt.0.and.D11(I).ge.0.0 ) DI(I) =
500:           & MIN(DI(I),D11(I))
501:       else
502:         DLOC1(I) = MAX(DLOC1(I),D11(I))
503:         DLOC2(I) = MIN(DLOC2(I),D22(I))
504:       end if
505:       if ( IZT(I).eq.2 ) then
506:         if ( DLOC1(I).lt.DLOC2(I).and.DLOC1(I).ge.0.0 ) DI(I)
507:           & = MIN(DI(I),DLOC1(I))
508:         DLOC1(I) = -DINF
509:         DLOC2(I) = DINF
510:       end if
511: 370 continue
512:     else
513:       do 380 I = 1, NFFL(NZONE+1)
514:         if ( LLS(I).gt.0 ) then
515:           if ( D22(I).ge.0.0 ) DI(I) = MIN(DI(I),D22(I))
516:         else
517:           if ( D11(I).ge.0.0 ) DI(I) = MIN(DI(I),D11(I))
518:         end if
519: 380 continue
520:       end if

```

src/mvp/flight.f

```

521: C
522:   390 continue
523: C-----
524: C .... CHECK LOST PARTICLE .....
525: C-----
526:       II      = 0
527:       do 400 I = 1, NFFL(NZONE+1)
528:         if ( DI(I).eq.DINF ) II = II + 1
529:       400 continue
530: C
531: C .... COUNT THE NUMBER OF FREE FLIGHT
532: C
533:       NCNTR(16) = NCNTR(16) + NFFL(NZONE+1) - II
534: C
535:       if ( II.gt.0 ) then
536: C/#IF PARA(SX* CRAY)
537: *       call MVPSYNC_LOCK(2)
538: C/#ENDIF
539:       write(IOW,*) ' (FLIGHT) ', II, ' PARTICLES ARE LOST ! '
540:       NLOST = NLOST + II
541:       ILOST = NDEAD
542:       do 410 I = 1, NFFL(NZONE+1)
543:         if ( DI(I).eq.DINF ) then
544:           ILOST = ILOST + 1
545:           LSDED(ILOST) = LSFFL(I)
546:           IGI(I) = -IGI(I)
547:         end if
548:       410 continue
549:       do 420 I = NDEAD + 1, ILOST
550:         LL = LSDED(I)
551:         write(IOW,7000) XXX(LL), YYY(LL), ZZZ(LL), AAA(LL), BBB(LL),
552:           & CCC(LL)
553:       7000 format(/1X,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',
554:         & E12.5,' ETA=',E12.5,' XI=',E12.5)
555:       420 continue
556: C/#IF PARA(SX* CRAY)
557: *       call MVPSYNC_UNLOCK(2)
558: C/#ENDIF
559:       NDEAD = ILOST
560: C-----
561: C ..... DELETE LOST PARTICLES .....
562: C-----
563:       ISAFE = 0
564: *VOCL LOOP,NOVREC
565:       do 430 I = 1, NFFL(NZONE+1)
566:         if ( IGI(I).gt.0 ) then
567:           ISAFE = ISAFE + 1
568:           LSFFL(ISAFE) = LSFFL(I)
569:           IZFFL(ISAFE) = IZFFL(I)
570:           DI(ISAFE) = DI(I)
571:         end if
572:       430 continue
573: C-----
574: C ..... GATHER UNLOST PARTICLES AGAIN .....
575: C-----
576:       do 440 K = 1, NZONE
577:         NFFL(K) = 0
578:       440 continue
579:       do 450 I = 1, ISAFE
580:         X(I) = XXX(LSFFL(I))
581:         Y(I) = YYY(LSFFL(I))
582:         Z(I) = ZZZ(LSFFL(I))

```

```

583:       AI(I) = AAA(LSFFL(I))
584:       BI(I) = BBB(LSFFL(I))
585:       CI(I) = CCC(LSFFL(I))
586:       IGI(I) = IGG(LSFFL(I))
587:       W(I) = WWW(LSFFL(I))
588:       NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
589:   450 continue
590:       NFFL(NZONE+1) = ISAFE
591:       KK = 0
592:       NZK = 0
593:       do 470 K = 1, NZONE
594:         if ( NFFL(K).ne.0 ) then
595:           NZK = NZK + 1
596:           JKSF(NZK) = K
597:           IPZONE(NZK) = KK + 1
598:           do 460 I = 1, NFFL(NZONE+1)
599:             if ( IZFFL(I).eq.K ) KK = KK + 1
600:           460 continue
601:           end if
602:       470 continue
603:       IPZONE(NZK+1) = NFFL(NZONE+1) + 1
604:       end if
605: C-----
606: C .... CALCULATE DISTANCE TO LATTICE FRAME & COORDINATES IN UPPER .....
607: C
608:       IBP(I) = 1/0 = COLLISION/ NON COLLISION
609: C
610: C ( IZS=1, IZE=NZK, LLS AS IFC, DSDA0 TO DSDA5 AS XUP TO CUP )
611: C-----
612:       if ( JHLAT.ne.0 ) then
613:         if ( JVMNT.ne.0 ) call VMNT00( 11, 3 )
614: C
615:         call LFRAME( IOW, JDEBG, NBANK,NEST,NLATT,NLBZ,NZONE,
616:           & DINF, IBP, IPZONE, 1, NZK, DI, LLS,
617:           & DSDA0, DSDA1, DSDA2, DSDA3, DSDA4, DSDA5,
618:           & LEVL, LZZ, LPOS, LCRS,
619:           & KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ,
620:           & IPLAT, KLATT, KSLAT, LTYP, MLBZZ,
621:           & X, Y, Z, AI, BI, CI, R )
622:         if ( JVMNT.ne.0 ) call VMNT22( 11, 3 )
623:       end if
624: C=====
625: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) .....
626: C
627:       IBP(I) = 1/0 = COLLISION/ NON COLLISION
628: C=====
629: C-----
630: C ---- HEXAGONAL LATTICE -----
631: C
632:       if ( JHLAT.ne.0 ) then
633:         ICOLS = 0
634:         call RANU2( IRAND, R, NFFL(NZONE+1), ICON )
635: C-----
636: C USE IZT AS COLLISION FLAG ! (IZT = 1/0 = COLISION/NO COLISION)
637: C-----
638:       do 480 I = 1, NFFL(NZONE+1)
639:         IZT(I) = 0
640:       480 continue
641:       KKK = 1
642:       MAT = KZMAT(JKSF(KKK))
643:       do 500 NK = KKK, NZK
644:         if ( KZMAT(JKSF(NK)).ne.MAT ) go to 510
645:       500 continue
646:       NK = NZK + 1
647:       KKK2 = NK
648:   510
649: C
650:       if ( MAT.gt.0 ) then

```

src/mvp/flight.f

```

551:         do 520 I = IPZONE(KKK), IPZONE(KKK2) - 1
552:
553: C         ..... SMAC(IBP(I),MMAC(IBP(I),1),1) : TOTAL XSEC. ....
554:
555:         D1      = -LOG(R(I)) /SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
556:         if ( DI(I).gt.D1 ) then
557:             IZT(I) = 1
558:             DI(I)  = D1
559:             ICOLS  = ICOLS + 1
560:             LSCOL(NCOLS+ICOLS) = LSFFL(I)
561:         end if
562: 520      continue
563:     end if
564:     KKK   = KKK2
565:     if ( KKK.le.NZK ) go to 490
566: C-----
567: C .... SEND UNCOLLIDED PARTICLES TO SEARCH STACK .....
568: C-----
569:     INXT  = 0
570: *VOCL LOOP,NOVREC
571:     do 530 I = 1, NFFL(NZONE+1)
572:         if ( IZT(I).eq.0 ) then
573:
574:             IZNN  = IZFFL(I)
575: C
576: C ..... IN THE CASE OF HEXAGONAL LATTICE GEOMETRY .....
577: C
578:             if ( LLS(I).ne.0 ) then
579:                 IZNN  = LLS(I)
580:                 LEVL(LSFFL(I)) = LEVL(LSFFL(I)) - 1
581:                 IZFFL(I) = LLS(I)
582:                 X(I)    = DSDA0(I)
583:                 Y(I)    = DSDA1(I)
584:                 Z(I)    = DSDA2(I)
585:                 AI(I)   = DSDA3(I)
586:                 BI(I)   = DSDA4(I)
587:                 CI(I)   = DSDA5(I)
588:             end if
589: C
590:             INXT  = INXT + 1
591:             LSSRC(NNXT(NZONE+1)+INXT) = LSFFL(I)
592:             IZNXT(NNXT(NZONE+1)+INXT) = IZNN
593:         end if
594: 530      continue
595:
596: *VOCL LOOP,SCALAR
597:     do 540 I = 1, NFFL(NZONE+1)
598:         if ( IZT(I).eq.0 ) NNXT(IZFFL(I)) = NNXT(IZFFL(I)) + 1
599: 540      continue
600: C
601:         NCOLS = NCOLS + ICOLS
602:         NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
603: C
604: C ---- NO-HEXAGONAL LATTICE
605: C
606:     else
607:         ICOLS = 0
608:         call RANU2( IRAND, R, NFFL(NZONE+1), ICON )
609: C-----
610: C      USE IZT AS COLLISION FLAG ! (IZT = 1/0 = COLISION/NO COLISION)
611: C-----
612:         do 550 I = 1, NFFL(NZONE+1)
613:             IZT(I) = 0
614: 550      continue
615:         KKK      = 1

```

```

716: 560      MAT      = KZMAT(JKSF(KKK))
717:         do 570 NK = KKK, NZK
718:             if ( KZMAT(JKSF(NK)).ne.MAT ) go to 580
719: 570      continue
720:         NK      = NZK + 1
721: 580      KKK2    = NK
722: C
723:         if ( MAT.gt.0 ) then
724:             do 590 I = IPZONE(KKK), IPZONE(KKK2) - 1
725:
726: C         ..... SMAC(IBP(I),MMAC(IBP(I),1),1) : TOTAL XSEC. ....
727:
728:             D1      = -LOG(R(I)) /SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
729:             if ( DI(I).gt.D1 ) then
730:                 IZT(I) = 1
731:                 DI(I)  = D1
732:                 ICOLS  = ICOLS + 1
733:                 LSCOL(NCOLS+ICOLS) = LSFFL(I)
734:             end if
735: 590      continue
736:         end if
737:         KKK      = KKK2
738:         if ( KKK.le.NZK ) go to 560
739: C-----
740: C .... SEND UNCOLLIDED PARTICLES TO SEARCH STACK .....
741: C-----
742:         INXT  = 0
743:         do 600 I = 1, NFFL(NZONE+1)
744:             if ( IZT(I).eq.0 ) then
745:                 INXT  = INXT + 1
746:                 LSSRC(NNXT(NZONE+1)+INXT) = LSFFL(I)
747:                 IZNXT(NNXT(NZONE+1)+INXT) = IZFFL(I)
748:             end if
749: 600      continue
750: *VOCL LOOP,SCALAR
751:         do 610 I = 1, NFFL(NZONE+1)
752:             if ( IZT(I).eq.0 ) NNXT(IZFFL(I)) = NNXT(IZFFL(I)) + 1
753: 610      continue
754: C
755:             NCOLS = NCOLS + ICOLS
756:             NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
757:         end if
758: C-----
759: C .... MOVE PARTICLES TO NEXT POINTS AND PUT VALUES IN BANK.....
760: C-----
761: *VOCL LOOP,NOVREC
762:         do 620 I = 1, NFFL(NZONE+1)
763:             XXX(LSFFL(I)) = X(I) + AI(I)*DI(I)
764:             YYY(LSFFL(I)) = Y(I) + BI(I)*DI(I)
765:             ZZZ(LSFFL(I)) = Z(I) + CI(I)*DI(I)
766:
767: C         .... DI(I) -> W * DI for use in TALLY ...
768:
769:             DI(I) = DI(I)*W(I)
770: 620      continue
771: C-----
772: C .... TAKE TALLIES (FLUX & RATE) .....
773: C-----
774:         do 630 I = 1, NFFL(NZONE+1)
775:             LLS(I) = MMAC(LSFFL(I),1)
776:             if ( JTLT.eq.0 ) IZT(I) = KZREG(IZFFL(I))
777: 630      continue
778: C
779:         if ( JTLT.ne.0 ) then
780:             do 640 I = 1, NFFL(NZONE+1)

```

src/mvp/flight.f

```
781:          IZT(I) = IBREG(LSFFL(I))
782: 640    continue
783:    end if
784: C
785:    JREGN = 1
786:    JZONE = 1
787:    if ( JVMNT.ne.0 ) call VMNT00( 10, 3 )
788:    call TALLY( IOW,JEIGN,JRTTR,JREGN,JZONE,JRESP, DI, NFFL(NZONE+1),
789:      &      IGI, IZT, LSFFL, LLS, W, NGROUP, NGP1, NREG, NRESP, NUC,
790:      &      NBANK, NSMIC, NSMAC, NSTAL, MB, IZFFL, DNZON, NZONE,
791:      &      LEMIC(9), NEMIC, RESP, SMIC, LMIC, SMAC, LMAC, SGTAL,
792:      &      FLTR, RETR, RMIC, DMFLX, RSTR, WCNTR(15), WCNTR(20) )
793:    if ( JVMNT.ne.0 ) call VMNT22( 10, 3 )
794: C
795: C-----
796: C .... DELETE PARTICLES FROM FLIGHT STACK .....
797: C-----
798:    do 650 I = 1, NZONE + 1
799:      NFFL(I) = 0
800: 650 continue
801:    return
802:    end
```

src/mvp/flione.f

```

1: C/#IF ARGSAVE
2: *      subroutine FLION0(IOW,
3: *      N NDEAD,NCOLS,NCOLP,NLOST,
4: *      N NKTCSP,MB,NSMAC,NSMIC,NEMIC,NEVENT,
5: *      B   XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,KKP,IZZ,EEE,IGG,TTT,ITT,LEVL,LZZ,
6: *      B   LPOS,LCRS,KLSF,IBSPC,IBREG,SMAC,LMAC,KMAC,MMAC,SMIC,LMIC,SGTAL,
7: *      B   KSPI, DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
8: *      M   LPDEN ,INUCT ,DENST ,DNFLX, LPDNP ,IATMT ,DNSTP ,
9: *      S   LSFFL,NFFL ,IZFFL ,
10: *      S   LSCOL ,LSCLP, LSSRC ,NNXT ,IZNXT ,LSDED ,
11: *      G   SDA ,KZMAT ,KZREG ,KSPSU,KTCSP,KZDA ,KZAA ,
12: *      G   KDALT ,KDALT ,NVLAT ,SZLAT ,IPCEL ,
13: *      G   KSLAT ,LTYP ,MLBZZ ,KZLBZ,KCELL ,ICTYP ,KLATT ,CELSZ ,
14: *      G   PNND ,KNND ,PPPF ,KPPF ,
15: *      T   JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, PSALP, PSXYZ,
16: *      T   DTALY, RESP ,FLTR,RETR, RMIC ,RSTR ,TFLH ,
17: *      T   NCNTR ,WCNTR ,DNZON ,LEMIC, TIMEB, NMKREG, MKREG,
18: *      &   NTSRF, KTSRF, ITSRF, IDSRF, ANGLB, NANGLE,
19: *      T   ENGYB, DNFLXSM, RMICSM, ! sc-mt

x
20: *      &      CX, KLIB1, KLIB2, XLIB1,
21: *      &      XLIB2, ! sc-mt

x
22: *      &      CXP, KLBPN1, KLBPN2, XLBPN2, ! ph-nu

c
23: *      &      NSMICPN, NUC_MAX, SMICPN, NATMT, MNUCPN, ! ph-nu

c
24: *      &      LPIDPN, IZTBPN, DNSTPN, MNEUTPN, NNCSTPN, ! ph-nu

c
25: *      &      INCSTPN, NMTMPN, MTMPN, NSTALY, NTCUT, WTCUT, ! ph-nu

c
26: *      &      CXPN, KLBPN1, KLBPN2, XLBPN1, KPNFG, ! ph-nu

c
27: C/#ELSE
28:      subroutine FLIONE( IOW, NDEAD, NCOLS, NCOLP, NLOST,
29:      &      NKTCSP, MB, NSMAC, NSMIC, NEMIC, NEVENT,
30:      &      XXX, YYY, ZZZ, AAA, BBB, CCC, WWW,
31:      &      KKP, IZZ, EEE, IGG, TTT,
32:      &      ITT, LEVL, LZZ, LPOS, LCRS, KLSF,
33:      &      IBSPC, IBREG, SMAC, LMAC, KMAC, MMAC, SMIC,
34:      &      LMIC, SGTAL, KSPI, DBNK, KDBNK, MDBNK, IBNK,
35:      &      KIBNK, MIBNK, LPDEN, INUCT, DENST, DNFLX,
36:      &      LPDNP, IATMT, DNSTP, LSFFL, NFFL, IZFFL,
37:      &      LSCOL, LSCLP, LSSRC, NNXT, IZNXT, LSDED, SDA,
38:      &      KZMAT, KZREG, KSPSU, KTCSP, KZDA, KZAA, DALT,
39:      &      KDALT, NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP,
40:      &      MLBZZ, KZLBZ, KCELL, ICTYP, KLATT, CELSZ, PNND,
41:      &      KNND, PPPF, KPPF, JDTRG, IDTAL, JTEVE, LTEVE,
42:      &      KTEVE, NTEVE, PSALP, PSXYZ, DTALY, RESP, FLTR,
43:      &      RETR, RMIC, RSTR, TFLH, NCNTR, WCNTR,
44:      &      DNZON, LEMIC, TIMEB, NMKREG, MKREG,
45:      &      NTSRF, KTSRF, ITSRF, IDSRF, ANGLB, NANGLE,
46:      &      ENGYB, DNFLXSM, RMICSM, ! sc-mtx
47:      &      CX, KLIB1, KLIB2, XLIB1,
48:      &      XLIB2, ! sc-mtx
49:      &      CXP, KLBPN1, KLBPN2, XLBPN2,
50:      &      X, Y, Z, AI, BI, CI, W,
51:      &      DI, IGI, IBP, IFC,
52:      &      R, DLOC1, DLOC2, IKL, IKL2, XUP, YUP,
53:      &      ZUP, AUP, BUP, CUP, IRG, TI, VI,
54:      &      ISTG, ILUP, ILST, XYZ1, ABC1, XYZ2, ABC2,
55:      &      DDI, IBP0, IBP1, IBP2, KPLT, JKSF, SMCW,
56:      &      IWKF1, WKF1, ! sc-mtx
57:      &      MZONE,
58:      &      STOTP, SNUFP, SLOSP, ! pert

59:      &      SMICP, SMACP, ! pert
60:      &      NPTDS, WWD, JPTTR, JPTNU, NPTCS, WWR, MAXDA, ! pert
61:      &      DELA , ! pert
62:      &      WWT,NUCPT,MNUC, NGSP, WSC, NORDD, ! pert
63:      &      RNU, NEBEF, WCBEF, ! b-eff
64:      &      NSMICPN, NUC_MAX, SMICPN, NATMT, MNUCPN, ! ph-nu

c
65:      &      LPIDPN, IZTBPN, DNSTPN, MNEUTPN, NNCSTPN, ! ph-nu

c
66:      &      INCSTPN, NMTMPN, MTMPN, NSTALY, NTCUT, WTCUT, ! ph-nu

c
67:      &      CXPN, KLBPN1, KLBPN2, XLBPN1, KPNFG, ! ph-nu

c
68:      &      NPTBE, WWLD) ! lambda
69: C/#ENDIF
70: C=<MVP>=====
71: C PURPOSE: FREE-FLIGHT OF ONE-ZONE LOGIC
72: C CALLED IN: ACTION
73: C CALLS: GETMAC,LFRAME,TALLY, RANU2,VMNTR0,VMNTR1,VMNTR2
74: C
75: C=====

76: C
77: C === INTER-STACK DATA FLOW ===
78: C
79: C Free-Flight Stack -----+-----> Next-Zone-Search Stack
80: C (LSFFL,IZFFL,NFFL) | (LSSRC,IZNXT,NNXT)
81: C +-----> Collision Stack
82: C | (LSCOL,NCOLS)
83: C | (LSCLP,NCOLP)
84: C (LOST) +-----> Dead Particle Stack
85: C (LSDED,NDEAD)
86: C
87: C === BANK DATA TO BE UPDATED ===
88: C
89: C (XXX,YYY,ZZZ) : Position
90: C (AAA,BBB,CCC) : Direction (Hex-lattice or overlapping frame)
91: C LEVL : Lattice Level (Hex-lattice or overlapping frame)
92: C IZZ : Zone # (Hex-lattice or overlapping frame)
93: C
94: C === BANK DATA ADDED NEWLY ===
95: C
96: C DATA IN SIGMA BANK
97: C=====
98: C implicit real*8(D)
99: C
100: C include 'INC/_KPIDS'
101: C include 'INC/_NGPS'
102: C
103: C include '../shared/INC/_SIZES'
104: C include '../shared/INC/_PGEOM'
105: C include 'INC/_CXSEC'
106: C include 'INC/_FLAGS'
107: C include '../shared/INC/_TASKDT'
108: C
109: C include '../shared/INC/_IDXOFF'
110: C
111: C **** PARTICLE BANK ****
112: C
113: C real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
114: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
115: C real EEE(NBANK)
116: C real WWW(NBANK)
117: C real*8 TTT(NBANK)
118: C integer ITT(NBANK)
119: C integer KKP(NBANK)

```


src/mvp/flione.f

```

120: integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
121: & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KMAC(NBANK,MB,2),
122: & MMAC(NBANK,2), KLSF(NBANK), LMAC(8), LMIC(8), LEMIC(8,2,2)
123: real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
124: & SGTAL(NBANK,NSTAL)
125: integer IBSPC(NBANK,0:NEST)
126: integer IBREG(NBANK)
127: integer KSPI(NBANK,NUC)
128:
129: real SMICPN(NBANK,NUCPN,NSMICPN) ! ph-nuc-3
130: real RNU(NBANK,NUC_MAX,3) ! b-eff, ph-nuc-3
131: C
132: C ... optional bank parameters
133: C
134: real*8 DBNK(NBANK,*)
135: integer KDBNK(0:MDBNK)
136: integer IBNK(NBANK,*)
137: integer KIBNK(0:MIBNK)
138: integer KPNFG(NBANK) ! ph-nuc-4
139: C
140: C *** STACKS *****
141: C
142: integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
143: integer LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK)
144: integer LSCOL(NBANK)
145: integer LSCLP(NBANK)
146: integer LSDED(NBANK)
147: C
148: C *** GEOMETRY *****
149: C
150: real*8 SDA(NSDA), DALT(*), SZLAT(8,NLATT), CELSZ(6,*)
151: integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
152: integer KCELL(NZONE), ICTYP(NCELL), NVLAT(4,*)
153: integer KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT)
154: integer IPLAT(NLATT+1), IPCEL(NCELL), KLATT(*), KSLAT(*)
155: integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
156: C
157: C ... Nearest Neighbour Distribution for STGM region.
158: C
159: real PNND(NPNND)
160: integer KNND(NPNND)
161: C
162: C ... Partial Packing Fraction for STGM-region
163: C
164: real PPPF(NPPPF)
165: integer KPFF(NPPPF)
166: C
167: C *** MATERIAL DATA *****
168: C
169: real DENST(*), DNSTP(*)
170: real DNSTPN(NUCPNA) ! ph-nuc-3
171: integer NATMT(NPATOM), MNUCPN(NMAT+1), IZTBPN(NUCPNA), ! ph-nuc-3
172: & LPIDPN(NUCPNA), MNEUTPN(NUCPNA) ! ph-nuc-3
173: integer INUCT(*), LPDEN(NMAT+1), IATMT(*), LPDNP(NMAT+1)
174: C
175: C *** CROSS SECTION & POINTERS *****
176: C
177: real CX(MCX), XLIB1(NUC,11)
178: real XLIB2(NUC,NMT,4) ! sc-mtx
179: integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
180: C
181: real CXP(MCXP)
182: integer KLBPN1(NPATOM,20), KLBPN2(NPATOM,NMTP,6)
183: real XLBP2(NPATOM,NMTP,2)
184: real CXPN(MCXP), XLBPN1(NUCPNI,6) ! ph-nuc-3

```

```

185: integer KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13) ! ph-nuc-3
186: C
187: C *** TALLY *****
188: C
189: integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
190: real PSALP(NREG)
191: real*8 PSXYZ(3)
192: integer IDTAL(*)
193: real*8 DTALY(*)
194: C
195: real RESP(NGROUP,NRESP), DNZON(NUC,NZONE,2)
196: real*8 FLTR(NGROUP,NREG), RETR(NREG,NRESP),
197: & RMIC(NGROUP,NREG,NUC,NEMIC), RSTR(NGROUP,NSTAL)
198: real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
199: real*8 DNFLX(NGROUP,NREG,NUC)
200: real*8 TPLH(KPLIM)
201: real*8 NTCUT(NGROUP,NREG), WTCUT(NGROUP,NREG) ! ph-nuc-3
202: C
203: real TIMEB(NTIME+1)
204: integer MKREG(NREG,2)
205: integer INCSTPN(NSTALY,2), MTMPN(NMTMPN) ! ph-nuc-3
206: C
207: C
208: C ... Tally surface .....
209: C
210: integer KTSRF(2,*), ITSRF(NTSRF+1,2), IDSRF(NTSRF)
211: C
212: C ... angle bin (cosine) ....
213: C
214: real*8 ANGLB(NANGLE+1)
215: C
216: C
217: C ... beta-effective ...
218: C
219: real*8 WCBEF(NEBEF) ! b-eff
220: C
221: C ... scattering matrix ...
222: real ENGYB(*) ! sc-mtx
223: real*8 DNFLXSM(*), RMICSM(*) ! sc-mtx
224: C
225: C *** WORKING AREA *****
226: C (not all arrays may be available for any condition.
227: C See WRKARY routine.)
228: C
229: real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
230: & CI(NBANK), W(NBANK), DI(NBANK), DLOC1(NBANK), DLOC2(NBANK)
231: real R(3*NBANK)
232: integer IGI(NBANK), IBP(NBANK), IFC(NBANK)
233: integer IRG(NBANK), IKL(NBANK), IKL2(NBANK)
234: integer ILST(NBANK)
235: C
236: C .... USED FOR HEXAGONAL LATTICE GEOMETRY and STG region ...
237: C
238: integer ILUP(NBANK)
239: real*8 XUP(NBANK), YUP(NBANK), ZUP(NBANK)
240: real*8 AUP(NBANK), BUP(NBANK), CUP(NBANK)
241: C
242: integer ISTG(NBANK)
243: integer IBP0(NBANK), IBP1(NBANK), IBP2(NBANK)
244: integer KPLT(NLBZ+1)
245: integer JKSF(NLBZ)
246: real*8 XYZ1(NBANK,3), ABC1(NBANK,3)
247: real*8 XYZ2(NBANK,3), ABC2(NBANK,3)
248: real*8 DDI(NBANK)
249: C

```

src/mvp/flione.f

```

250:      real SMCW(NBANK,NSMIC)
251: C
252: C      .... only for time depende case ...
253: C
254:      real*8 TI(NBANK), VI(NBANK)
255: C
256:      integer IWKF1(NBANK)          ! sc-mtx
257:      real      WKF1(NBANK)         ! sc-mtx
258: C
259: C      ! pert >>
260: C **** PERTURBATION *****
261: C
262:      real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
263:      real*8 WWD(NBANK,NPTDS,NORDD), WWR(NBANK,NPTCS)
264:      real*8 WSC(NBANK,0:NQSP,NPTCS)
265:      integer IPTTR(NREG,NPTDS+NPTCS), JPTNU(NUC,NPTDS)
266:      real DELA(MAXDA,NPTDS+NPTCS)
267:      real*8 WWT(NBANK,NPTDS)
268: C
269: C      .... material data
270: C
271:      integer MNUC(NMAT)
272: C
273: C      .... working area
274: C
275:      real*8 STOTP(NBANK),SMUPP(NBANK),SEOSP(NBANK) ! pert <<
276: c+beff2
277:      real*8 WWLD(NBANK)
278: c      ... local variables ...
279:      real*8 DDE, VVV
280: c-beff2
281: C
282: C **** LOCAL ARRAY *****
283: C
284:      integer IPZZ(2)
285: C
286:      integer IKSORT(KPLIM+1), KSORT(KPLIM)
287: C
288: C **** constants *****
289: C
290:      real*8 SMALL
291:      parameter( SMALL      = 1.0D-35 )
292: C
293: C --- light speed --- (cm/s)
294: C
295:      real*8 CLIGHT
296:      parameter( CLIGHT     = 2.99792458D10 )
297: C
298: C --- neutron mass --- (gram)
299: C
300:      real*8 NMASS
301:      parameter( NMASS      = 1.6749286D-24 )
302: C
303: C --- erg / eV ---  electron charge
304: C
305:      real*8 EECHRG
306:      parameter( EECHRG     = 1.60217733D-12 )
307: C
308: C --- neutron's mc**2 in eV
309: C
310:      real*8 MC2
311:      parameter( MC2        = NMASS*CLIGHT**2/EECHRG )
312: C
313: C ... statement functions ...
314: C

```

```

315:      include '../shared/INC/_PMLATT'
316:      include '../shared/INC/_SFLATT'
317: C
318: C
319: C/#IF ARGSAVE
320: *      return
321: C
322: C -----
323: C
324: *      entry      FLIONE( MZONE,
325: *      W          X          ,Y          ,Z          ,AI          ,BI          ,CI          ,
326: *      W          W          ,DI          ,IGI          ,IBP          ,IFC          ,R          ,DLOC1 ,
327: *      W          DLOC2, IKL,IKL2 ,XUP          ,YUP          ,ZUP          ,AUP          ,BUP          ,CUP          ,
328: *      W          IRG, TI, VI, ISTG, ILUP, ILST ,
329: *      W          XYZ1, ABC1, XYZ2, ABC2,
330: *      W          DDI, IBP0, IBP1, IBP2, KPLT, JKSF, SMCW, IWKF1, WKF1,
331: c##<2007/03/14:PN3:
332: *      &          RNU )
333: c##>
334: C/#ENDIF
335: C
336:      if ( JVMNT.ne.0 ) call VMNTRI( 3, NFFL(MZONE) )
337: C-----
338: C ... GATHER BANK POINTER .....
339: C-----
340:      II      = 0
341: *VOCL LOOP,NOVREC
342:      do 100 I = 1, NFFL(NZONE+1)
343:          if ( IZFFL(I).eq.MZONE ) then
344:              II      = II + 1
345:              IBP(II) = LSFFL(I)
346:          end if
347:      100 continue
348: C
349: C
350: C ... sort particle bank index by particle species if necessary
351: C
352: C      KKPID : particle species ID for single particle species problem
353: C
354:      KS      = 0
355:      KKPID   = 0
356:      if ( NPKIND.eq.1 ) then
357:          KS = 1
358:          KKPID   = KKP(IBP(1))
359:          KSORT(1) = KKPID
360:          IKSORT(1) = 1
361:          IKSORT(2) = NFFL(MZONE) + 1
362:      else
363:          call KPSORT( KKP, NBANK, KPLIM, IBP, NFFL(MZONE), KS, KSORT(1),
364:          &          IKSORT(1), IFC )
365:      end if
366: C
367: C-----
368: C ... Material #
369: C-----
370: C
371:      MAT      = KZMAT(MZONE,1)
372: C
373: C ... this is a base material (matrix) region of STG region.
374: C
375:      JJJNND   = 0
376:      if ( MAT.lt.0 ) then
377:          if ( ISLATT(MAT).and.LTYP(LATTNM(MAT)).eq.10 ) then
378:              MAT      = KZMAT(MZONE,2)
379:              JJJNND   = 1

```

src/mvp/flione.f

```

380:         else
381:         write(IOW,'(1X,A,A,I5,A,2I6,A,I6)')
382:         &      'XXX(FLIONE) Program error ? Invalid zone :',
383:         &      ' MZONE=', MZONE, ' KZMAT(MZONE,*) ',
384:         &      KZMAT(MZONE,1), KZMAT(MZONE,2), ' LTYPE ',
385:         &      LTYPE(LATTNM(MAT))
386:         stop 666
387:         end if
388:     end if
389: C
390: C-----
391: C PREPARE MACRO CROSS SECTION ( TOTAL )
392: C-----
393: C
394:     if ( MAT.gt.0 ) then
395: C
396: C ..... NEUTRONS .....
397: C
398:         do 110 K = 1, KS
399:             IKPID = KSORT(K)
400:             KK     = IKSORT(K)
401:             NM     = IKSORT(K+1) - IKSORT(K)
402: C
403:             if ( IKPID.eq.KPNEUT ) then
404:                 if ( JVMNT.ne.0 ) call VMNT00( 9, 3 )
405:                 if ( JVMNT.ne.0 ) call VMNTR1( 9, NM )
406: C
407:                 call GETMAC( IRAND, NM, IBP(KK), MAT, JEIGN, JAMXC,
408:                 &      JDEBG, NBANK, EEE, MB, NUC, NSTAL, NSMAC, SMAC,
409:                 &      LMAC, KMAC, MMAC, NSMIC, SMIC, LMIC, KSPI, LPDEN,
410:                 &      INUCT, DENST, NMAT, NMT, CX, MCX, KLBI1, KLBI2,
411:                 &      XLBI1, IFC, X, Y, Z, AI, BI, CI, DI, R, SMCW )
412:                 if( JPERT.ne.0 .and. JPTMP.ne.0 ) then
413:                     call GMACPT( IRAND, INEUT, IBP, MAT, JEIGN, JDEBG, NBANK,
414:                     &      MB, NUC, NSTAL, NSMAC, SMAC, LMAC, KMAC, MMIC,
415:                     &      NSMIC, SMIC, LMIC, SGTAL, LPDEN, INUCT, DENST, NMAT,
416:                     &      IFC, R, X, Y, Z, AI,
417:                     &      SMICP, SMACP, STOTP, SNUFP, SLOSP )
418:                 else
419:                     call GETMAC( IRAND, NM, IBP(KK), MAT, JEIGN, JAMXC,
420:                     &      JDEBG, NBANK, EEE, MB, NUC, NSTAL, NSMAC, SMAC,
421:                     &      LMAC, KMAC, MMAC, NSMIC, SMIC, LMIC, KSPI, LPDEN,
422:                     &      INUCT, DENST, NMAT, NMT, CX, MCX, KLBI1, KLBI2,
423:                     &      XLBI1, IFC, X, Y, Z, AI, BI, CI, DI, R, SMCW )
424:                 end if
425: C
426:                 if ( JVMNT.ne.0 ) call VMNT22( 9, 3 )
427: C
428: C ..... PHOTONS .....
429: C
430:             else if ( IKPID.eq.KPPHOT ) then
431: C
432:                 if ( JVMNT.ne.0 ) call VMNT00( 9, 3 )
433:                 if ( JVMNT.ne.0 ) call VMNTR1( 9, NM )
434:                 if ( JPHNU.eq.0 ) then
435:                     call GTMACP( IRAND, NM, IBP(KK), MAT, JAMXC, JDEBG,
436:                     &      NBANK, EEE, MB, NUC, NSTAL, NSMAC, SMAC, LMAC,
437:                     &      KMAC, MMAC, NSMIC, SMIC, KSPI, LPDNP, IATMT,
438:                     &      DNSTP, NMAT, NPATOM, NMTP, CXP, MCXP, KLBP1,
439:                     &      KLBP2, XLBP2, IFC, R, X, Y, AI, BI, CI, LMIC)
440:                 else ! ph-nuc-3 >>
441:                     call GTMACPN( IRAND, NM, IBP(KK), MAT, JAMXC, JDEBG,
442:                     &      NBANK, EEE, MB, NUC, NSTAL, NSMAC, SMAC, LMAC,
443:                     &      KMAC, MMAC, NSMIC, SMIC, KSPI, LPDNP, IATMT,
444:                     &      NATMT, DNSTP, NMAT, NPATOM, NMTP, CXP, MCXP,

```

```

445:         &      KLBP1, KLBP2, XLBP2, NUCPN, NUCPNA, NUC_MAX,
446:         &      NSMICPN, SMICPN, MNUCPN, IZTBPN, LPIDPN,
447:         &      DNSTPN, EBOTLPN, CXP, MCXP, KLBP1, KLBP2,
448:         &      XLBP1, NUCPNI, NMTPN, RNU, IFC, R, X, Y, AI,
449:         &      BI, CI, SMCW )
450:         end if ! ph-nuc-3 <<
451:         if ( JVMNT.ne.0 ) call VMNT22( 9, 3 )
452: C
453: C ... ???
454: C
455:         else
456:             write(IOW,'(1X,A,A,I6)')
457:             &      'XXX(FLIONE) Program error: Particle other than',
458:             &      ' neutron/photon is not supported!!! IKPID=', IKPID
459:             stop 666
460:         end if
461: 110 continue
462: C
463:     else
464:         *VOCL LOOP,NOVREC
465:         do 120 I = 1, NFFL(MZONE)
466:             MMAC(IBP(I),1) = 0
467: 120 continue
468:         end if
469: C
470: C      ( IFC, R, X, Y, Z & AI ARE FREE TO USE HEREFTER )
471: C
472: C-----
473: C .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
474: C      (Nov 1998: initialization of DLOC1,DLOC2 is moved into SPEAR1)
475: C-----
476: C
477:         *VOCL LOOP,NOVREC
478:         do 130 I = 1, NFFL(MZONE)
479:             X(I) = XXX(IBP(I))
480:             Y(I) = YYY(IBP(I))
481:             Z(I) = ZZZ(IBP(I))
482:             AI(I) = AAA(IBP(I))
483:             BI(I) = BBB(IBP(I))
484:             CI(I) = CCC(IBP(I))
485:             W(I) = WWW(IBP(I))
486:             IGI(I) = IGG(IBP(I))
487:             IKL(I) = KLSF(IBP(I))
488:             IKL2(I) = 0
489:             DI(I) = DINF
490:             ILST(I) = 0
491: 130 continue
492: C
493: C      .... distance to current zone boundary ....
494: C
495: C      for matrix zone of STG region, distance to zone boundary
496: C      is that to STG region boundary and not calculated here
497: C      but in DFRAME. (also means that arrival to STG region frame
498: C      must be treated as a movement to an upper lattice level.)
499: C
500:         if ( JFISX.eq.0 .or. JJJNND.eq.0 ) then
501:             JSS = 0
502:             call SPEAR1( MZONE, KZDA, KZAA, SDA, NFFL(MZONE), NBANK, JSS,
503:             &      X, Y, Z, AI, BI, CI, DI, DLOC1, DLOC2, IKL, IKL2,
504:             &      DUMMY1, DINF, DEPS )
505: C
506: C      ... "lost" particle index to "1"
507: C
508:         *VOCL LOOP,NOVREC
509:         do 140 I = 1, NFFL(MZONE)

```

src/mvp/flione.f

```

510:         if ( DI(I).eq.DINF ) then
511:             ILST(I) = 1
512:         end if
513: 140      continue
514:     end if
515: C
516: C         ( DLOC1 and DLOC2 are free to use hereafter )
517: C
518: C         .... distance to lattice frame boundary ....
519: C
520:     IHF      = 0
521:     if ( JFISX.eq.0.and.JHLAT.ne.0 ) then
522:         if ( KCELL(MZONE).gt.0.and.ICYTP(KCELL(MZONE)).eq.2 ) then
523:             IHF      = 1
524:             IPZZ(1) = 1
525:             IPZZ(2) = NFFL(MZONE) + 1
526:             if ( JVMNT.ne.0 ) call VMNT00( 11, 3 )
527:             if ( JVMNT.ne.0 ) call VMNTR1( 11, NFFL(MZONE) )
528: C
529:             call LFRAME( IOW, JDEBG, NBANK, NEST, NLATT, NLBZ, NZONE,
530: &                     DINF, IBP, IPZZ, 1, 1, DI, IFC, XUP, YUP, ZUP, AUP,
531: &                     BUP, CUP, LEVL, LZZ, LPOS, LCRS, KZMAT, KDALT, DALT,
532: &                     NVLAT, SZLAT, CELSZ, IPLAT, KLATT, KSLAT, LTYP,
533: &                     MLBZZ, X, Y, Z, AI, BI, CI, R )
534: C ..... IFC(I) = N/0
535: C             = cross the outer frame (N=MZONE# in upper level) // NO
536: C ..... DI(I) may have been changed.
537: C
538: *VOCL LOOP,NOVREC
539:     do 150 I = 1, NFFL(MZONE)
540: C
541: C         .... this case moves only one upper lattice level
542: C
543:             ILUP(I) = LEVL(IBP(I)) - 1
544: C
545: C         ... lost in frame distance calculation
546: C
547:             if ( DI(I).eq.DINF ) then
548:                 ILST(I) = ILST(I) + 2
549:             end if
550: 150      continue
551:             if ( JVMNT.ne.0 ) call VMNT22( 11, 3 )
552:         end if
553: C
554: C         ... distance to frames until current lattice level is
555: C         calculated if flight path count is zero.
556: C
557:     else if ( JFISX.ne.0.and.(KCELL(MZONE).gt.0.or.JJNNND.gt.0) ) then
558:         IHF      = 1
559:         KK        = 0
560: *VOCL LOOP,NOVREC
561:     do 160 I = 1, NFFL(MZONE)
562:         IFC(I) = 0
563:         if ( IBNK(IBP(I),KIBNK(5)).eq.0 ) then
564:             KK      = KK + 1
565:         end if
566: 160      continue
567:         if ( KK.gt.0 ) then
568:             if ( JVMNT.ne.0 ) call VMNT00( 11, 3 )
569:             if ( JVMNT.ne.0 ) call VMNTR1( 11, NFFL(MZONE) )
570:             call DFRAME( IOW, MZONE, NFFL(MZONE), JDEBG,
571: N             NBANK, NEST, NLATT, NLBZ,
572: N             NZONE, DINF, DEPS,
573: I             X, Y, Z, AI, BI, CI, IBP,
574: O             DI, IFC, XUP, YUP, ZUP, AUP, BUP, CUP, ILUP, ILST,

```

```

575: B             LEVL, LZZ, LPOS, LCRS,
576: B             DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
577: G             KZMAT, KZDA, KZAA, SDA, KCELL,
578: G             KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT,
579: G             KLATT, KSLAT, LTYP, MLBZZ, KZLBZ,
580: W             KPLT, JKSF, IBP0, IBP1, IBP2,
581: W             XYZ1(1,1), XYZ1(1,2), XYZ1(1,3),
582: W             ABC1(1,1), ABC1(1,2), ABC1(1,3),
583: W             XYZ2(1,1), XYZ2(1,2), XYZ2(1,3),
584: W             ABC2(1,1), ABC2(1,2), ABC2(1,3),
585: W             DDI, IKL, IKL2, DLOC1, DLOC2 )
586:         if ( JVMNT.ne.0 ) call VMNT22( 11, 3 )
587:     end if
588: C
589: C         ... get frame distance and upper level postion etc. from
590: C         banked data
591: C
592:         if ( NFFL(MZONE)-KK.gt.0 ) then
593: *VOCL LOOP,NOVREC
594:         do 170 I = 1, NFFL(MZONE)
595:             KL      = IBNK(IBP(I),KIBNK(6)+LEVL(IBP(I))-1)
596:             KI      = KDBNK(4) + KL - 1
597:             DDDD     = DBNK(IBP(I),KI)
598:             if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.DDDD.lt.DI(I) ) then
599:                 IFC(I) = LZZ(IBP(I),KL)
600:                 ILUP(I) = KL - 1
601:                 K6     = KDBNK(6) + 6*(KL-1)
602:                 DAA     = DBNK(IBP(I),K6+3)
603:                 DBB     = DBNK(IBP(I),K6+4)
604:                 DCC     = DBNK(IBP(I),K6+5)
605:                 XUP(I) = DBNK(IBP(I),K6) - DDDD*DAA
606:                 YUP(I) = DBNK(IBP(I),K6+1) - DDDD*DBB
607:                 ZUP(I) = DBNK(IBP(I),K6+2) - DDDD*DCC
608:                 AUP(I) = DAA
609:                 BUP(I) = DBB
610:                 CUP(I) = DCC
611:                 DI(I) = DDDD
612:             end if
613: 170      continue
614:         end if
615:     end if
616: C
617: C-----
618: C .... CHECK LOST PARTICLE .....
619: C-----
620: C
621:     ISAFE      = NFFL(MZONE)
622: C
623:     if ( MAT.gt.0 ) then
624:         do 180 I = 1, NFFL(MZONE)
625: C
626: C         ... negative or zero total xsec leads to unpredictable behaviour.
627: C         treat it as lost.
628: C
629:             if ( SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1)).le.0.0 ) then
630:                 ILST(I) = ILST(I) + 4
631:             end if
632: 180      continue
633:         end if
634: C
635:         II      = 0
636:         do 190 I = 1, ISAFE
637:             if ( ILST(I).ne.0 ) II = II + 1
638: 190      continue
639: C

```

src/mvp/flione.f

```

640:      if ( II.gt.0 ) then
641:
642: C/#IF PARA(SX* CRAY)
643: *      call MVPSYNC_LOCK(2)
644: C/#ENDIF
645:
646:      write(IOW,'(/1X,A,I5,A,I5)') ' (FLIONE) ', II,
647: &      ' particles are LOST !! in ZONE ', MZONE
648:      NLOST = NLOST + II
649:      ILOST = NDEAD
650: C
651:      do 200 I = 1, NFFL(MZONE)
652:      IBPI = IBP(I)
653:      if ( ILST(I).ne.0 ) then
654:      ILOST = ILOST + 1
655:      LSDED(ILOST) = IBPI
656:      if ( KIBNK(17).ne.0 ) IBNK(IBPI,KIBNK(17)) = 0 ! ph-nuc-3
657:      if ( KIBNK(18).ne.0 ) IBNK(IBPI,KIBNK(18)) = 0 ! ph-nuc-3
658:      if ( KIBNK(19).ne.0 ) IBNK(IBPI,KIBNK(19)) = 0 ! ph-nuc-3
659:      if ( JPHNU.ne.0 ) KPNFG(IBPI) = 0 ! ph-nuc-3
660:      end if
661: 200 continue
662: C
663:      do 210 I = NDEAD + 1, ILOST
664:      LL = LSDED(I)
665:      write(IOW,7000) XXX(LL), YYY(LL), ZZZ(LL), AAA(LL), BBB(LL)
666: &      CCC(LL), KLSF(LL), LL
667:
668:      if ( MAT.gt.0.and.SMAC(LL,MMAC(LL,1),LMAC(LL,1)).le.0.0 ) then
669:      write(IOW,7020) SMAC(LL,MMAC(LL,1),LMAC(LL,1), EEE(LL)
670:      end if
671: C
672: 210 continue
673: C
674: 7000 format(1X,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',E12.5,
675: &      ' ETA=',E12.5,' XI=',E12.5,' SURFACE=',I5,' NC=',I6)
676: 7020 format(1X,
677: &      ' ** Non-positive total cross section for the particle'
678: &      ', above : SIG-T = ',E12.5,' Energy ',E12.5)
679: C
680: C/#IF PARA(SX* CRAY)
681: *      call MVPSYNC_UNLOCK(2)
682: C/#ENDIF
683: C
684:      NDEAD = ILOST
685: C
686: C ..... Delete lost particles .....
687: C
688:      ISAFE = 0
689: *VOCL LOOP,NOVREC
690:      do 220 I = 1, NFFL(MZONE)
691:      if ( ILST(I).eq.0 ) then
692:      ISAFE = ISAFE + 1
693:      IBP(ISAFE) = IBP(I)
694:      DI(ISAFE) = DI(I)
695:      IKL2(ISAFE) = IKL2(I)
696:      if ( IHF.ne.0 ) then
697:      ILUP(ISAFE) = ILUP(I)
698:      IFC(ISAFE) = IFC(I)
699:      XUP(ISAFE) = XUP(I)
700:      YUP(ISAFE) = YUP(I)
701:      ZUP(ISAFE) = ZUP(I)
702:      AUP(ISAFE) = AUP(I)
703:      BUP(ISAFE) = BUP(I)
704:      CUP(ISAFE) = CUP(I)

```

```

705:      end if
706:      end if
707: 220 continue
708: C
709: C ..... GATHER UNLOST PARTICLES AGAIN .....
710: C
711:      do 230 I = 1, ISAFE
712:      X(I) = XXX(IBP(I))
713:      Y(I) = YYY(IBP(I))
714:      Z(I) = ZZZ(IBP(I))
715:      AI(I) = AAA(IBP(I))
716:      BI(I) = BBB(IBP(I))
717:      CI(I) = CCC(IBP(I))
718:      IGI(I) = IGG(IBP(I))
719:      W(I) = WWW(IBP(I))
720: 230 continue
721: C
722: C .... relocate particle species boundary ....
723: C
724:      call KPSORT( KKP, NBANK, KPLIM, IBP, ISAFE, KS, KSORT(1),
725: &      KKSORT(1), IFC )
726: C
727:      end if
728: C
729: C-----
730: C..... TALLY FLIGHT COUNT ....
731: C-----
732: C
733:      do 240 K = 1, KS
734:      IKPID = KSORT(K)
735:      NCNTR(16,IKPID) = NCNTR(16,IKPID) + IKSORT(K+1) - IKSORT(K)
736: 240 continue
737: C
738: C-----
739: C .... sampling of STG sphere position from
740: C      Nearest Neighbour Distribution (NND)
741: C-----
742: C
743:      if ( JJJNND.ne.0 ) then
744:      MLT = LATTNM(KZMAT(MZONE,1))
745:      NN = ISAFE
746: C
747: C ... ISTG (cell buffer zone of STG cell region) is returned
748: C ... IKL (cell position in STG lattice) is returned
749: C      if sampled distance to STG is smaller than DI(I)
750: C
751: C      XUP,YUP,ZUP here is not a coodinate in "upper level" in this
752: C      case.
753: C
754:      call SMPSTG( IOW, MLT, X, Y, Z, AI, BI, CI, DI, ISTG, IBP,
755: &      NN, PNND, KNND, NPNND, IPLAT, KLATT, KSLAT, SZLAT,
756: &      NLATT, CELSZ, IPCEL, PPPF, KPPF, NPPPF,
757: &      IRAND, LEVL, DBNK, KDBNK, MDBNK, IBNK,
758: &      KIBNK, MIBNK, NBANK, XUP, YUP, ZUP, IKL, IBP2, TI, R )
759: C
760: C ... override frame distance flag
761: C
762:      do 250 I = 1, ISAFE
763:      if ( ISTG(I).ne.0 ) then
764:      IFC(I) = ISTG(I)
765:      AUP(I) = AI(I)
766:      BUP(I) = BI(I)
767:      CUP(I) = CI(I)
768:      ILUP(I) = LEVL(IBP(I))
769:      end if

```

src/mvp/flione.f

```

770: 250 continue
771: end if
772: C
773: C
774: C=====
775: C Sample flight distance on DLOC1
776: C and save path stretching factor on DLOC2 if necessary
777: C=====
778: C
779: DAJUST = DEPS*0.1D0
780: C
781: C ... path stretching (exponential transformation) is necessary or not
782: C
783: JEXP = 0
784: if ( MAT.gt.0.and.JNCOL.eq.0 ) then
785: if ( JPSTR.eq.0
786: & .or.
787: & (JPSTR.ne.0.and.JTILT.eq.0.and.PSALP(KZREG(MZONE)).eq.0.0) )
788: & then
789: JEXP = 0
790: else
791: JEXP = 1
792: end if
793: C
794: call RANU2( IRAND, R, ISAFE, ICON )
795: C
796: C ... no PATH-STRETCHING in this case
797: C
798: if ( JEXP.eq.0 ) then
799: *VOCL LOOP,NOVREC
800: do 260 I = 1, ISAFE
801: C/#IF .NOT.SSL2
802: DLOC1(I) = -LOG(R(I)+SMALL) /
803: & SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
804: C/#ELSE
805: DLOC1(I) = -LOG(R(I)) /
806: & SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
807: C/#ENDIF
808: C/#IF .NOT.RANDOM( BOUND )
809: if ( DLOC1(I).le.0.0 ) DLOC1(I) = DAJUST
810: C/#ENDIF
811: 260 continue
812: C
813: C ... PATH-STRETCHING is necessary
814: C
815: else
816: *VOCL LOOP,NOVREC
817: do 270 I = 1, ISAFE
818: C
819: C ... IBREG is always set (from Jan 2000)
820: IRG1 = IBREG(IBP(I))
821: C
822: C ... get absolute (level 0) coordinates.
823: C (assuming JFISX>0 for path stretching)
824: C
825: DXXX = X(I)
826: DYYY = Y(I)
827: DZZZ = Z(I)
828: DAAA = AI(I)
829: DBBB = BI(I)
830: DCCC = CI(I)
831: if ( JLATT.gt.0 ) then
832: LVL = LEVL(IBP(I))
833: if ( LVL.gt.0 ) then
834: KD4 = KDBNK(4)

```

```

835: KD6 = KDBNK(6)
836: DDDD = DBNK( IBP(I),KD4)
837: DAAA = DBNK( IBP(I),KD6+3)
838: DBBB = DBNK( IBP(I),KD6+4)
839: DCCC = DBNK( IBP(I),KD6+5)
840: DXXX = DBNK( IBP(I),KD6) - DDDD*DAAA
841: DYYY = DBNK( IBP(I),KD6+1) - DDDD*DBBB
842: DZZZ = DBNK( IBP(I),KD6+2) - DDDD*DCCC
843: end if
844: end if
845: C
846: DLL =
847: & Sqrt((DXXX-PSXYZ(1))**2+(DYYY-PSXYZ(2))**2
848: & +(DZZZ-PSXYZ(3))**2)
849: DOO = 0.0D0
850: if ( DLL.ne.0.0D0 ) then
851: DOO =
852: & (DAAA*(DXXX-PSXYZ(1))+DBBB*(DYYY-PSXYZ(2))
853: & +DCCC*(DZZZ-PSXYZ(3))) /DLL
854: end if
855: DOO = PSALP(IRG1)*DOO
856: if ( ABS(DOO).gt.1.0D0 ) DOO = SIGN(0.99999D0,DOO)
857: C
858: C/#IF .NOT.SSL2
859: DLOC1(I) = -LOG(R(I)+SMALL) /
860: & SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1)) /(1.0D0-DOO)
861: C/#ELSE
862: DLOC1(I) = -LOG(R(I)) /
863: & SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1)) /(1.0D0-DOO)
864: C/#ENDIF
865: C/#IF .NOT.RANDOM( BOUND )
866: if ( DLOC1(I).le.0.0 ) DLOC1(I) = DAJUST
867: C/#ENDIF
868: DLOC2(I) = DOO
869: 270 continue
870: end if
871: end if
872: C
873: C ... store IBNK(*,KIBNK(5)) for use in surface crossing tally
874: C
875: if ( JTSRF.ne.0 ) then
876: do 280 I = 1, ISAFE
877: IBP2(I) = IBNK(IBP(I),KIBNK(5))
878: 280 continue
879: end if
880: C
881: C=====
882: C .... Calculate distance to collision point & compare with DI(I) ....
883: C and send particles to stacks etc.
884: C
885: C X,Y,Z,AI,BI,CI may be changed to that of upper lattice geometry
886: C level when flight path reaches frame boundary.
887: C
888: C=====
889: C
890: call FLDIST(IOW,MZONE, ISAFE, MAT, IHF, JJJNND,
891: N NPKIND, NGROUP,NBANK,NZONE, DINF,
892: N NDEAD,NCOLS,NCOLP,
893: N NUNV, NSPACE,NKTCSP,
894: N NEST,NEVENT,NTIME,TCUT,
895: & X, Y, Z, AI, BI, CI,
896: & W, DI, IGI, IBP, IFC,
897: & DLOC1, DLOC2, IKL, IKL2, XUP, YUP, ZUP,
898: & AUP, BUP, CUP, TI,VI, ISTG, ILUP,
899: & KSORT(1),IKSORT(1),KS,

```

src/mvp/flione.f

```

900:      B          KKP, IZZ,   EEE,   TTT,   ITT,   LEVL,
901:      B          LPOS, IBSPC, IBREG,
902:      B          DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
903:      S          LSCOL, LSCLP, LSSRC, NNXT, IZNXT,
904:      S          LSDED,
905:      G          KZREG, KSPSU, KTCSP,
906:      T          TFLH , NCNTR, WCNTR,
907:      &          NREG, IGG, NTCUT, WTCUT, KPNFG) ! ph-nuc
908: C
909: C-----
910: C .... MOVE PARTICLES TO NEXT POINTS AND MODIFY COORDINATES IN BANK ...
911: C-----
912: C
913: *VOCL LOOP,NOVREC
914:      do 290 I = 1, ISAFE
915:          XXX(IBP(I)) = X(I) + AI(I)*DI(I)
916:          YYY(IBP(I)) = Y(I) + BI(I)*DI(I)
917:          ZZZ(IBP(I)) = Z(I) + CI(I)*DI(I)
918:          KLSF(IBP(I)) = IKL2(I)
919:      290 continue
920:      if( JPERT.ne.0 ) then
921:          do 291 I = 1, ISAFE
922:      c
923: C-----
924: c TEMPERATURE
925: C-----
926:      if( JPTMP.ne.0 ) then
927:          WWT(IBP(I),1) = WWT(IBP(I),1) -
928:      &          DI(I)*SMACP(IBP(I),MMAC(IBP(I),1),LMAC(1))
929:          WWT(IBP(I),3) = WWT(IBP(I),3) -
930:      &          DI(I)*SMACP(IBP(I),MMAC(IBP(I),1),LMAC(1))
931:          WWT(IBP(I),5) = WWT(IBP(I),5) -
932:      &          DI(I)*SMACP(IBP(I),MMAC(IBP(I),1),LMAC(1))
933:          WWT(IBP(I),7) = WWT(IBP(I),7) -
934:      &          DI(I)*SMACP(IBP(I),MMAC(IBP(I),1),LMAC(1))
935:      end if
936: C-----
937: c DENSITY & NUMBER DENSITY
938: C-----
939:      if ( JPDEN.ne.0 ) then
940:
941:          DSMAC = DI(I)*SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
942:
943:          if ( JTLLT.ne.0 ) then
944:              IREG = IBREG(IBP(I))
945:          else
946:              IREG = KZREG(MZONE)
947:          end if
948:
949:          MM = KZMAT(MZONE,1)
950:          if ( MM.gt.0 ) then
951:              do IPT = 1, NPTDS
952:      c          DELKR = dble( JPTTR( KZREG(MZONE), IPT ) )
953:          DELKR = dble( JPTTR( IREG, IPT ) )
954:
955: c ... density perturbation ...
956: c          WWD(IBP(I),IPT) = WWD(IBP(I),IPT) - DSMAC*DELKR
957:
958:          do J = 1, MNUC(MM)
959:              IDN = LPDEN( MM ) - 1 + J
960:              KN = INUCT( IDN )
961:              RHO = DENST( IDN )
962:
963:              WWD(IBP(I),IPT,1) = WWD(IBP(I),IPT,1)
964:      &          - DELKR*dble(JPTNU(KN,IPT))

```

```

965:      &          * DI(I)*RHO*SMIC(IBP(I),KN,LMIC(1))
966:      end do
967:
968:      end do
969:      else if ( MM.lt.0 ) then
970:          write(6,*) ' XXX(FLIONE) MM less than 0 in ',
971:      &          'pertrubation calculation. MM=',MM
972:      stop
973:      end if
974:
975:      do IPT = 1, NPTCS
976:      c          DLT01 = dble(JPTTR(KZREG(MZONE),NPTDS+IPT))
977:          DLT01 = dble(JPTTR(IREG,NPTDS+IPT))
978:          DELA1 = dble(DELA(1,NPTDS+IPT))
979:          WWR(IBP(I),IPT) =
980:      &          WWR(IBP(I),IPT)*exp(-DSMAC*DELA1*DLT01)
981:          do ISP = 0, NGSP
982:              WSC(IBP(I),ISP,IPT) =
983:      &          WSC(IBP(I),ISP,IPT)*exp(-DSMAC*DELA1*DLT01)
984:          end do
985:      end do
986:
987:      end if
988:
989:      291 continue
990:      end if
991: c+beff2
992:      if (NPTBE.ne.0) then
993:          do I = 1, ISAFE
994:              DDE = EEE(IBP(I))
995:              VVV = CLIGHT*sqrt(DDE*(DDE+2*MC2))/(DDE+MC2)
996:              if (JTLLT.ne.0) then
997:                  IREG = IBREG(IBP(I))
998:              else
999:                  IREG = KZREG(MZONE)
1000:              end if
1001:              MM = KZMAT(MZONE,1)
1002:              if (MM.gt.0) then
1003:                  RHO = 0.d0
1004:                  do J = 1, MNUC(MM)
1005:                      IDN = LPDEN(MM) - 1 + J
1006:                      RHO = RHO + DENST(IDN)
1007:                  end do
1008:                  WWLD(IBP(I)) = WWLD(IBP(I)) - DI(I)/VVV
1009:              end if
1010:          end do
1011:      end if
1012: c-beff2
1013: C
1014: C ... Flight direction may be changed in free lattice frame problem.
1015: C
1016:      if ( IHF.ne.0 ) then
1017:      *VOCL LOOP,NOVREC
1018:          do 300 I = 1, ISAFE
1019:              AAA(IBP(I)) = AI(I)
1020:              BBB(IBP(I)) = BI(I)
1021:              CCC(IBP(I)) = CI(I)
1022:          300 continue
1023:      end if
1024: C
1025: C-----
1026: C .... adjust weights for path-stretching mode .....
1027: C          (is it right correcting here ?)
1028: C-----
1029: C

```

src/mvp/flione.f

```

1030:      if ( JEXP.ne.0 ) then
1031: *VOCL LOOP,NOVREC
1032:      do 310 I = 1, ISAFE
1033: C
1034: C      ... collided in this zone ...
1035:      if ( DLOC1(I).le.DI(I) ) then
1036:          DEE =
1037:      &      EXP(-DLOC2(I)*SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
1038:      &      *DLOC1(I)) /(1.0D0-DLOC2(I))
1039:      else
1040: C
1041: C      ... escaped from this zone or cutoff ...
1042:      DEE =
1043:      &      EXP(-DLOC2(I)*SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
1044:      &      *DI(I))
1045:      end if
1046:      W(I) = W(I)*DEE
1047:      WWW(IBP(I)) = W(I)
1048: 310 continue
1049: end if
1050: C
1051: C-----
1052: C .... reduce weights for "collisionless" mode .....
1053: C-----
1054: C
1055:      if ( JNCOL.eq.1.and.MAT.gt.0 ) then
1056: *VOCL LOOP,NOVREC
1057:      do 320 I = 1, ISAFE
1058:          WWW(IBP(I)) = W(I)*
1059:      &      EXP(-DI(I)*SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1)))
1060: 320 continue
1061:      end if
1062: C
1063: C-----
1064: C .... collect data for tallies (region # etc.)
1065: C-----
1066: C
1067: C      ... IFC(I) IS POSITION OF MACRO X-SEC IN X-SEC BANK HERE.
1068: C
1069:      if ( MAT.gt.0 ) then
1070:          do 330 I = 1, ISAFE
1071:              IFC(I) = MMAC(IBP(I),1)
1072: 330 continue
1073:      else
1074:          do 340 I = 1, ISAFE
1075:              IFC(I) = 0
1076: 340 continue
1077:      end if
1078: C
1079: C      .... Gather region # for universe-dependent tally ...
1080: C
1081:      JREGN = 0
1082:      if ( JSCTM.ne.0 ) then
1083:          do I = 1, ISAFE
1084:              IRG(I) = KZREG(MZONE)
1085:          end do
1086:      else
1087:          IRG(1) = KZREG(MZONE)
1088:      end if
1089: C
1090:      if ( JTLT.ne.0.and.(KCELL(MZONE).ne.0.or.JJJNND.ne.0) ) then
1091: *VOCL LOOP,NOVREC
1092:      do 350 I = 1, ISAFE
1093:          IRG(I) = IBREG(IBP(I))
1094: 350 continue

```

```

1095: C
1096: C      .... Check multi REGION / single REGION ....
1097: C      ( JREGN = 1 / 0 )
1098: C
1099:      do 360 I = 2, ISAFE
1100:          if ( IRG(I).ne.IRG(1) ) go to 370
1101: 360 continue
1102: C
1103: C      .... ALL PARTICLES ARE IN THE SAME REGION. ....
1104:      go to 380
1105: C
1106: 370 JREGN = 1
1107: C
1108: 380 continue
1109:      else if ( NTEVE.gt.0.and.(JTEVE(2).gt.0.or.JTEVE(3).gt.0) ) then
1110:          do 390 I = 1, ISAFE
1111:              IRG(I) = KZREG(MZONE)
1112: 390 continue
1113:          end if
1114: C
1115: C-----
1116: C .... Take surface tallys .....
1117: C-----
1118: C
1119: C      variables used for surface tally:
1120: C
1121: C      X(*),Y(*),Z(*) : particle position in "local" coordinates
1122: C      AI(*),BI(*),CI(*) : particle direction in "local" coordinates
1123: C      (position and direction may be changed to those of
1124: C      "global" (absolute) coordinates)
1125: C      TI(*) : time of flight
1126: C      VI(*) : velocity
1127: C
1128: C      DI(I) : flight distance to next collision or boundary crossing point
1129: C      W(I) : particle weight
1130: C      (DI and W must not be changed their value, for use in track length
1131: C      tally.)
1132: C
1133: C      and other pre-collected data:
1134: C
1135: C      IBP(I) : bank pointers for particles to be tallied
1136: C
1137: C      IBP2(I) : flight path counter value (IBNK(*,KIBNK(5)) before
1138: C      change in collision detection procedure.
1139: C
1140: C      IFC(I) : macro cross section set #
1141: C      IGI(I) : energy group(bin) #
1142: C      IRG(I) : region #
1143: C
1144: C      available working arrays (see WRKARY routine):
1145: C
1146: C      (without any condition)
1147: C
1148: C      integer : IKL(NBANK), IKL2(NBANK), ILST(NBANK)
1149: C      real : R(3*)
1150: C      double precision: DLOC1, DLOC2
1151: C
1152: C      (when JTSRF.ne.0 or other condition)
1153: C      double precision: XUP,YUP,ZUP, AUP, BUP, CUP
1154: C      XYZ1(*,3),XYZ2(*,3), ABC1(*,3), ABC2(*,3)
1155: C      DDI
1156: C      integer : ISTG, ILUP, IBP0, IBP1
1157: C
1158: C
1159:      if ( NTEVE.gt.0.and.JTEVE(3).gt.0 ) then

```


src/mvp/flione.f

```

1160: C
1161:       do 570 ITS = 1, NTSRF
1162:         KI10 = KIBNK(10) + ITS - 1
1163:         KD7 = KDBNK(7) + ITS - 1
1164:         ... first flight after birth or collision
1165:         set surface flag to 3
1166:         ICROSS = 0
1167: *VOCL LOOP,NOVREC
1168:       do 400 I = 1, ISAFE
1169:         if ( IBP2(I).le.0 ) then
1170:           IBNK(IBP(I),KI10) = 3
1171:         end if
1172:         if ( IBNK(IBP(I),KI10).ne.0 ) then
1173:           ICROSS = ICROSS + 1
1174:         end if
1175:       400 continue
1176: C
1177: C ... all flight path never cross this surface !!
1178: if ( ICROSS.eq.0 ) go to 570
1179: C
1180:       ICROSS = 0
1181: *VOCL LOOP,NOVREC
1182:       do 410 I = 1, ISAFE
1183:         if ( IBNK(IBP(I),KI10).ne.0 ) then
1184:           ICROSS = ICROSS + 1
1185:           XUP(ICROSS) = X(I)
1186:           YUP(ICROSS) = Y(I)
1187:           ZUP(ICROSS) = Z(I)
1188:           AUP(ICROSS) = AI(I)
1189:           BUP(ICROSS) = BI(I)
1190:           CUP(ICROSS) = CI(I)
1191:           DDI(ICROSS) = DI(I)
1192:           if ( JTIME.ne.0 ) then
1193:             DLOC1(ICROSS) = TTT(IBP(I))
1194:             DLOC2(ICROSS) = VI(I)
1195:           end if
1196: C
1197:           IBP1(ICROSS) = IBP(I)
1198:           IBP0(ICROSS) = I
1199:         end if
1200:       410 continue
1201: C
1202: C ... convert to absolute (level 0) coordinates if necessary
1203: C
1204:       if ( JFISX.ne.0.and.JLATT.gt.0 ) then
1205:         KD4 = KDBNK(4)
1206:         KD6 = KDBNK(6)
1207: *VOCL LOOP,NOVREC
1208:       do 420 I = 1, ICROSS
1209:         LVL = LEVL(IBP1(I))
1210:         if ( LVL.gt.0 ) then
1211:           DDDD = DBNK(IBP1(I),KD4)
1212:           AUP(I) = DBNK(IBP1(I),KD6+3)
1213:           BUP(I) = DBNK(IBP1(I),KD6+4)
1214:           CUP(I) = DBNK(IBP1(I),KD6+5)
1215:           XUP(I) = DBNK(IBP1(I),KD6) - DDDD*AUP(I)
1216:           YUP(I) = DBNK(IBP1(I),KD6+1) - DDDD*BUP(I)
1217:           ZUP(I) = DBNK(IBP1(I),KD6+2) - DDDD*CUP(I)
1218:         end if
1219:       420 continue
1220:       end if
1221: C
1222:       ITSF = ICROSS
1223: C
1224: C ... surface crossing judgement may need more than

```

```

1225: C           one cycle per surface, because a flight path may
1226: C           cross a surface more than once!!
1227: C
1228:       ISCYCL = 0
1229:       430 continue
1230:       ISCYCL = ISCYCL + 1
1231:       if ( ISCYCL.gt.20 ) then
1232:         write(IOW,*) '!!!(FLIONE) Too many cycles (', ISCYCL,
1233:           & ' ) for tally-surface',
1234:           & ' crossing judgement. Surface: ', ITS
1235:         write(IOW,*) ' Number of remaining particle : ', ITSF
1236:       Check %%%
1237:       write(*,*) '%% flione JFISX ', JFISX
1238:       write(*,*) '%% flione KIBNK ', KIBNK
1239:       write(*,*) '%% flione KDBNK ', KDBNK
1240:       write(*,*) '%% flione IBP1 ', (IBP1(I),I=1,ITSF)
1241:       write(*,*) '%% flione IBNK(*,KI10) ',
1242:         & (IBNK(IBP1(I),KI10),I=1,ITSF)
1243:       write(*,*) '%% flione DBNK(*,KD7) ',
1244:         & (DBNK(IBP1(I),KD7),I=1,ITSF)
1245:       do I = 1, ITSF
1246:         write(*,*) '%% DDI XUP,YUP,ZUP ', DDI(I), XUP(I),
1247:           & YUP(I), ZUP(I)
1248:         write(*,*) '%% AUP,BUP,CUP ', AUP(I), BUP(I), CUP(I)
1249:       end do
1250: C %%%%%%%%%
1251:       if ( ISCYCL.gt.25 ) go to 570
1252:       end if
1253: C
1254: C ... check distance to tally-surface
1255: C
1256: C surface crossing status IBNK(*,KIBNK(10)+*) is expected to
1257: C set +1/-1 (inside/outside is determined and distance is calculated)
1258: C or 0 (never cross).
1259: C
1260:       call FLSURF( ITS, NTSRF, KTSRF, ITSRF, IDSRF,
1261:         & ITSF, IBP1, NBANK,
1262:         & XUP, YUP, ZUP, AUP, BUP, CUP,
1263:         & SDA, NSDA, DINF, DEFS, ANGLB, NANGLE,
1264:         & DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
1265:         W XYZ1(1,1),XYZ1(1,2),XYZ1(1,3),
1266:         W ABC1(1,1),ABC1(1,2),ABC1(1,3), R(1),
1267:         W XYZ2(1,1),XYZ2(1,2),XYZ2(1,3),
1268:         W ABC2(1,1),ABC2(1,2),ABC2(1,3),
1269:         W ILUP, IKL, IKL2, ISTG)
1270: C
1271: C ... check crossing path
1272: C
1273:       ICROSS = 0
1274: *VOCL LOOP,NOVREC
1275:       do 440 I = 1, ITSF
1276:         III = IBP1(I)
1277:         IFSS = IBNK(III,KI10)
1278: C
1279:         DD0 = DBNK(III,KD7)
1280:         DBNK(III,KD7) = DD0 - DDI(I)
1281: C
1282: CM2016 if ( IFSS.ne.0.and.DBNK(III,KD7).lt.0.0d0 ) then
1283:       DERR = DD0 * 1.0d-8
1284:       if ( IFSS.ne.0.and.DBNK(III,KD7).le.DERR ) then
1285:         ICROSS = ICROSS + 1
1286: C
1287: C ... crossing from outside to inside
1288: C
1289:       if ( IFSS.lt.0 ) then

```

src/mvp/flione.f

```

1290:          IBNK(III,KI10) = +2
1291: C
1292: C          ... crossing from inside to outside
1293: C          ... IFSS == 1
1294: C          else
1295: C          IBNK(III,KI10) = -2
1296: C          end if
1297: C
1298: C          ... time of crossing (DLOC2: velocity)
1299: C          if ( JTIME.ne.0 ) then
1300: C             DLOC1(I) = DLOC1(I) + DD0/DLOC2(I)
1301: C          end if
1302: C
1303: C          XUP(ICROSS) = XUP(I) + DD0*AUP(I)
1304: C          YUP(ICROSS) = YUP(I) + DD0*BUP(I)
1305: C          ZUP(ICROSS) = ZUP(I) + DD0*CUP(I)
1306: C          AUP(ICROSS) = AUP(I)
1307: C          BUP(ICROSS) = BUP(I)
1308: C          CUP(ICROSS) = CUP(I)
1309: C          DDI(ICROSS) = DDI(I) - DD0
1310: C          if ( JTIME.ne.0 ) then
1311: C             DLOC1I = DLOC1(I) + DD0/DLOC2(I)
1312: C             DLOC1(ICROSS) = DLOC1I
1313: C             DLOC2(ICROSS) = DLOC2(I)
1314: C          end if
1315: C          IBP1(ICROSS) = III
1316: C          IBP0(ICROSS) = IBP0(I)
1317: C
1318: C          end if
1319: C
1320: 440      continue
1321: C
1322: C          if ( ICROSS.eq.0 ) go to 570
1323: C
1324: C          ITSF = ICROSS
1325: C
1326: C          -----
1327: C          ... take surface tally, huhhhhhh ...
1328: C          -----
1329: C
1330: C          ... flag set after time bin check ...
1331: C          ITCHK = 0
1332: C
1333: C          do 560 J = 0, JTEVE(3) - 1
1334: C          ... pointer to direct tally (idt) & d-tally # (it) ...
1335: C             IDT = KTEVE(LTEVE(3)+J) - 1
1336: C             IT = IDTAL(IDT+9)
1337: C          ... this tally is not for this surface ;-) ...
1338: C             if ( ABS(IDTAL(IDT+11)).ne.ITS ) go to 560
1339: C
1340: C          ... check particle species ...
1341: C          (assuming IBP0(I) is sorted by particle species)
1342: C
1343: C          if ( NPKIND.eq.1 ) then
1344: C             I1 = 1
1345: C             I2 = ITSF
1346: C          else
1347: C             IKPID = IDTAL(IDT+5)
1348: C             I1 = 1
1349: C             I2 = ITSF
1350: C             do 450 I = 1, ITSF
1351: C                if ( KKP(IBP1(I)).eq.IKPID ) then
1352: C                   I1 = I
1353: C                   go to 460
1354: C                end if
1355: 450      continue
1356: C          go to 560
1357: C          continue
1358: C          do 470 I = I1, ITSF
1359: C             if ( KKP(IBP1(I)).ne.IKPID ) then
1360: C                I2 = I
1361: C                go to 480
1362: C             end if
1363: 470      continue
1364: C             I2 = ITSF + 1
1365: 480      continue
1366: C             I2 = I2 - I1
1367: C          end if
1368: C
1369: C          ... skip if current region does not need this tally ;-) ...
1370: C
1371: C          if ( JREGN.eq.0 ) then
1372: C             if ( JDTRG(IRG(IBP0(1)),IT).eq.0 ) go to 560
1373: C          else
1374: C             do 490 I = 1, ITSF
1375: C                if ( JDTRG(IRG(IBP0(I)),IT).ne.0 ) go to 500
1376: 490      continue
1377: C                go to 560
1378: 500      continue
1379: C          end if
1380: C
1381: C          ... calculate time bin on crossing
1382: C
1383: C          if ( JTIME.ne.0.and.ITCHK.eq.0 ) then
1384: C             ITC2 = 0
1385: C             IDT2 = IDT + IDTOFF
1386: C             NNBIN = IDTAL(IDT2+6)
1387: C             do 510 JJ = 1, NNBIN
1388: C                KBIN = IDTAL(IDT2+1)
1389: C                if ( KBIN.eq.3 ) then
1390: C                   ITC2 = 1
1391: C                   go to 520
1392: C                end if
1393: C             end if
1394: C             continue
1395: C
1396: 510      continue
1397: C          ... time bin is set on ILUP(*)
1398: C          continue
1399: C          if ( ITC2.ne.0 ) then
1400: C             ITCHK = 1
1401: C             call BS0ISD( TIMEB, NTIME+1, DLOC1(1), ILUP(1),
1402: C                & ITSF )
1403: C             do 530 I = 1, ITSF
1404: C                ILUP(I) = MAX(1,MIN(NTIME,ILUP(I)))
1405: C             continue
1406: C             end if
1407: C          end if
1408: C
1409: C          if ( I2.gt.0 ) then
1410: C
1411: C          ... put current/flux on working array XYZ1(*,1)
1412: C
1413: C          ... current (just count weight!!)
1414: C          if ( IDTAL(IDT+11).gt.0 ) then
1415: C             KD72 = KD7 + NTSRF
1416: C             do 540 I = I1, I1 + I2 - 1
1417: C                XYZ1(I,1) = W(IBP0(I))
1418: C                XYZ1(I,1) = W(IBP0(I)) *
1419: C                SIGN( 1.0d0, DBNK(IBP1(I),KD72) )
1420: C             continue
1421: C          else

```

src/mvp/flione.f

```

1420:      KD72 = KD7 + NTSRF
1421:      do 550 I = I1, I1 + I2 - 1
1422:        DCC = ABS(DBNK(IBP1(I),KD72))
1423:        if ( DCC.gt.0.01D0 ) then
1424:          XYZ1(I,1) = W(IBP0(I)) /DCC
1425:        else
1426:          XYZ1(I,1) = W(IBP0(I)) /0.005D0
1427:        end if
1428:      550      continue
1429:    end if
1430: C
1431: C
1432:      call STALN3( IOW, ITS,NTSRF,
1433: &      JTIME, MZONE, IDTAL(IDT+1),XYZ1(I1,1),I2,
1434: &      IBP1(I1),IBP0(I1),
1435: &      IGI, IRG, IFC, ILUP(I1),
1436: &      NGROUP,NREG, NRESP, NUC, NBANK,
1437: &      NSMIC, NSMAC, NSTAL, MB, DNZON, NZONE,
1438: &      NTIME, NEMIC, RESP, NMKREG,MKREG,
1439: &      DTALY,
1440: &      SMIC, LMIC, SMAC, LMAC, SGTAL,
1441: &      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
1442: &      W
1443:      end if
1444: 560      continue
1445: C
1446: C      ... go to next surface distance calculation cycle....
1447: C
1448:      go to 430
1449: C
1450: 570      continue
1451:    end if
1452: C
1453: C      ... reduce distance to frame in lattice levels
1454: C      (this procedure should be here, because the distance
1455: C      may be used in surface crossing tally to get absolute
1456: C      coordinates and direction.)
1457: C
1458:      if ( JFISX.ne.0 ) then
1459:        MXLV = 0
1460:        KK = 0
1461:      *VOCL LOOP,NOVREC
1462:        do 580 I = 1, ISAFE
1463:          if ( IBNK(IBP(I),KIBNK(5)).ne.0 ) then
1464:            MXLV = MAX(MXLV,LEVL(IBP(I)))
1465:            KK = KK + 1
1466:          end if
1467: 580      continue
1468:          if ( MXLV.gt.0.and.KK.gt.0 ) then
1469:            KD4 = KDBNK(4)
1470:            do 600 LV = 1, MXLV
1471:      *VOCL LOOP,NOVREC
1472:              do 590 I = 1, ISAFE
1473:                if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.LEVL(IBP(I)).ge.LV
1474: &      ) then
1475:                  DBNK(IBP(I),KD4+LV-1) = DBNK(IBP(I),KD4+LV-1)
1476: &      - DI(I)
1477:                end if
1478: 590      continue
1479: 600      continue
1480:              end if
1481:            end if
1482: C
1483: C-----
1484: C .... Take track length tallys .....
```

```

1485: C-----
1486: C
1487: C      ... change value of DI(I) for track length tally
1488: C      DI(I) (distance) --> DI(I) * weight
1489: C
1490:      if ( JNCOL.eq.1.and.MAT.gt.0 ) then
1491: C
1492: C      ... collisionless mode and non void region ....
1493: C
1494:      *VOCL LOOP,NOVREC
1495:        do 610 I = 1, ISAFE
1496:          DSIGT = SMAC(IBP(I),MMAC(IBP(I),1),LMAC(1))
1497:          DEE = DI(I)*DSIGT
1498:          if ( DEE.lt.1.0D-10 ) then
1499:            DI(I) = DI(I)*W(I)
1500:          else
1501:            DI(I) = ((1D0-EXP(-DEE))/DSIGT)*W(I)
1502:          end if
1503: 610      continue
1504:        else
1505: C
1506:      *VOCL LOOP,NOVREC
1507:        do 620 I = 1, ISAFE
1508:          DI(I) = DI(I)*W(I)
1509: 620      continue
1510:        end if
1511: C
1512: C -----
1513: C      ... SPECIAL tallies by track length
1514: C -----
1515: C
1516:      if ( NTEVE.gt.0.and.JTEVE(2).gt.0 ) then
1517: C
1518: C      ... gather current time & time bin here in X & IKL !!
1519: C
1520:      if ( JTIME.ne.0 ) then
1521:      *VOCL LOOP,NOVREC
1522:        do 630 I = 1, ISAFE
1523:          X(I) = TTT(IBP(I))
1524:          IKL(I) = ITT(IBP(I))
1525:          if ( JPTIM.ne.0 ) DLOC1(I) = TI(I)
1526: 630      continue
1527:        end if
1528: C
1529:      do 710 J = 0, JTEVE(2) - 1
1530: C
1531: C      ... pointer to direct tally (idt) & d-tally # (it) ...
1532:          IDT = KTEVE(LTEVE(2)+J) - 1
1533:          IT = IDTAL(IDT+9)
1534: C
1535: C      ... check particle species ...
1536: C      (assuming IBP(I) is sorted by particle species)
1537: C
1538:      if ( NPKIND.eq.1 ) then
1539:        I1 = 1
1540:        N2 = ISAFE
1541:      else
1542:        IKPID = IDTAL(IDT+5)
1543:        I1 = 1
1544:        I2 = ISAFE
1545:        do 640 I = 1, ISAFE
1546:          if ( KKP(IBP(I)).eq.IKPID ) then
1547:            I1 = I
1548:            go to 650
1549:          end if
```

src/mvp/flione.f

```

1550: 640      continue
1551:          go to 710
1552: C
1553: 650      continue
1554:          do 660 I = I1, ISAFE
1555:              if ( KKP(IBP(I)).ne.IKPID ) then
1556:                  I2      = I
1557:                  go to 670
1558:              end if
1559: 660      continue
1560:          I2      = ISAFE + 1
1561: 670      continue
1562:          N2      = I2 - I1
1563:      end if
1564: C
1565: C .. skip if current region does not need this tally ...
1566: C
1567:      if ( JREGN.eq.0 ) then
1568:          if ( JDTRG(IRG(1),IT).eq.0 ) go to 710
1569:      else
1570:          do 680 I = 1, ISAFE
1571:              if ( JDTRG(IRG(I),IT).ne.0 ) go to 690
1572: 680      continue
1573:          go to 710
1574: 690      continue
1575:      end if
1576: C
1577:      if ( N2.gt.0 ) then
1578:          JPTIM2 = 0
1579: C
1580: C ... IZDUM is used as spacer for zone# array passed to
1581: C STALN1 and it is not used when MZONE > 0.
1582: C
1583:      call STALN1( IOW, JTIME, JPTIM, MZONE, IDTAL(IDT+1),
1584: & DI(I1), N2, IZDUM, IGI(I1), IRG(I1), IBP(I1),
1585: & IFC(I1), IKL(I1), X(I1), TI(I1), NGROUP, NREG,
1586: & NRESP, NUC, NBANK, NSMIC, NSMAC, NSTAL, MB,
1587: & DNZON, NZONE, NTIME, NEMIC, RESP, TIMES, NMKREG,
1588: & MKREG, DTALY, SMIC, LMIC, SMAC, LMAC, SGTAL,
1589: & TCUT, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, Y,
1590: & Z, R, JPTIM2,
1591: & EEE, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, CXPN, ! ph-
nuc-3
1592: & MCXPN, INCSTPN, MTMPN, NMTPN, KLBPN1, KLBPN2, ! ph-
nuc-3
1593: & XLBPN1, EBOTLPN, NUCPNA, MNEUTPN, DNSTPN, INUCPN, ! ph-
nuc-3
1594: & LPIDPN, SMCW(1,1), SMCW(1,2), SMCW(1,3), ! ph-
nuc-3
1595: & R(NBANK+1) ) ! ph-
nuc-3
1596: C
1597:      if ( JPTIM2.ne.0 ) then
1598:          do 700 I = I1, I1 + N2 - 1
1599:              X(I) = TTT(IBP(I))
1600:              IKL(I) = ITT(IBP(I))
1601:              TI(I) = DLOC1(I)
1602: 700      continue
1603:          end if
1604:      end if
1605: 710      continue
1606:      end if
1607: C
1608: C -----
1609: C .... update time and time-bin -----

```

```

1610: C -----
1611: C
1612:      if ( JTIME.ne.0 ) then
1613:          if ( JPTIM.eq.0 ) then
1614:              *VOCL LOOP,NOVREC
1615:              do 720 I = 1, ISAFE
1616:                  TTT(IBP(I)) = TTT(IBP(I)) + TI(I)
1617:                  X(I) = TTT(IBP(I))
1618: 720      continue
1619:          else
1620: C
1621: C .... case of PERIODIC TIME
1622: C
1623:              *VOCL LOOP,NOVREC
1624:              do 730 I = 1, ISAFE
1625:                  TTT(IBP(I)) = TTT(IBP(I)) + TI(I)
1626:                  if ( TTT(IBP(I)).ge.TCUT ) then
1627:                      ITREP = TTT(IBP(I)) /TCUT
1628:                      TTT(IBP(I)) = TTT(IBP(I)) - TCUT*ITREP
1629:                  end if
1630:                  X(I) = TTT(IBP(I))
1631: 730      continue
1632:          end if
1633: C
1634:          call BS0ISD( TIMEB, NTIME+1, X(1), IKL(1), ISAFE )
1635:          do 740 I = 1, ISAFE
1636:              ITT(IBP(I)) = MAX(1,MIN(NTIME,IKL(I)))
1637: 740      continue
1638:          end if
1639: C
1640: C -----
1641: C .... BASIC TALLYS (FLUX & RATE) -----
1642: C -----
1643:      do 750 K = 1, KS
1644:          IKPID = KSORT(K)
1645:          KK = IKSORT(K)
1646:          NM = IKSORT(K+1) - IKSORT(K)
1647:          if ( NM.eq.0 ) go to 750
1648: C
1649: C ..... NEUTRON .....
1650: C
1651:          if ( IKPID.eq.KPNEUT ) then
1652: C
1653:              JZONE = 0
1654: C
1655:              if ( JVMNT.ne.0 ) call VMNT00( 10, 3 )
1656:              if ( JVMNT.ne.0 ) call VMNTR1( 10, NM )
1657: C
1658:              call TALLY( IOW, JEIGN, JRTTR, JREGN, JZONE, JRESP, DI(KK),
1659: & NM, IGI(KK), IRG(KK), IBP(KK), IFC(KK), Y, NGROUP,
1660: & NSTAL, MB, MZONE, DNZON, NZONE, LEMIC(1,1,1), NEMIC,
1661: & RESP, SMIC, LMIC, SMAC, LMAC, SGTAL, FLTR, RETR,
1662: & RMIC, DNFLX, RSTR, WCNTR(15,KPNEUT),
1663: & WCNTR(20,KPNEUT),
1664: &
1665: c+beff1
1666: & JBEFF, RNU, IBNK, KIBNK, MIBNK, WCBEF(1), WCBEF(4),
1667: & WCBEF(7), WCBEF(10),
1668: c-beff1
1669: c##<2007/03/14:PN4:
1670: & JPHNU, KPNGF, WCNTR(32,KPNEUT), NUC_MAX )
1671: c##>
1672: C
1673:          if ( JVMNT.ne.0 ) call VMNT22( 10, 3 )
1674: C

```

src/mvp/flione.f

```

1675: c      if ( JSCTM.ne.0 ) then
1676:      if ( JSCTM.ne.0.and.JRTTR.ne.0 ) then
1677:          NB1 = NBANK/2 + 1
1678:          call TALSME( IOW, JSCTM, DI(KK), NM, IGI(KK), IRG(KK),
1679:              &          IBP(KK), NUC, NMT, NGROUP, NZONE, NREG, NBANK,
1680:              &          ETOP, EBOT, ETHMAX, NGP(KPNEUT),
1681:              &          ENGYB(KENGP(KPNEUT))),
1682:              &          JMICE, JMACE, DNZON, MZONE, KSPI, EEE, AAA,
1683:              &          BBB, CCC, CX, CX, MCX, SMIC, LMIC, NSMIC,
1684:              &          KLIB1, KLIB2, XLIB1, XLIB2,
1685:              &          DNFLXSM, RMICSM,
1686:              &          ILST, ILUP, ISTG, IXL, IKL2, CI(1), CI(NB1),
1687:              &          TI(1), TI(NB1), W(1), W(NB1), IFC, IWKF1,
1688:              &          R, X(1), X(NB1), Y(1), Y(NB1), Z(1), Z(NB1),
1689:              &          AI(1), AI(NB1), BI(1), BI(NB1), DLOC1, DLOC2,
1690:              &          WKF1 )
1691:      end if
1692: C
1693: C ..... PHOTON .....
1694: C
1695:      else if ( IKPID.eq.KPPHOT ) then
1696: CTEMP----
1697:          NEMICP = 0
1698: C -----
1699:          if ( JREGN.ne.0 ) then
1700:              JZONE = 0
1701:              if ( JVMNT.ne.0 ) call VMNT00( 10, 3 )
1702:              if ( JVMNT.ne.0 ) call VMNTR1( 10, NM )
1703: C
1704:              call TALLYP( IOW, JRTTR, JREGN, JZONE, JRESP, DI(KK), NM,
1705:                  &          IGI(KK), IRG(KK), IBP(KK), Y, NGROUP,
1706:                  &          NGP(KPPHOT), NREG, NRESP, NUC, NPATOM, NBANK,
1707:                  &          NSMIC, NSTAL, MZONE, DNZON, NZONE, LEMIC(1,1,1),
1708:                  &          NEMICP, RESP, SMIC, SGTAL, FLTR, RETR, RMIC,
1709:                  &          DNFLX, RSTR )
1710: C
1711:              if ( JVMNT.ne.0 ) call VMNT22( 10, 3 )
1712: C
1713:          else
1714:              JZONE = 0
1715:              if ( JVMNT.ne.0 ) call VMNT00( 10, 3 )
1716:              if ( JVMNT.ne.0 ) call VMNTR1( 10, NM )
1717: C
1718:              call TALLYP( IOW, JRTTR, JREGN, JZONE, JRESP, DI(KK), NM,
1719:                  &          IGI(KK), IRG, IBP(KK), Y, NGROUP, NGP(KPPHOT),
1720:                  &          NREG, NRESP, NUC, NPATOM, NBANK, NSMIC, NSTAL,
1721:                  &          MZONE, DNZON, NZONE, LEMIC(1,1,1), NEMICP, RESP,
1722:                  &          SMIC, SGTAL, FLTR, RETR, RMIC, DNFLX, RSTR )
1723: C
1724:              if ( JVMNT.ne.0 ) call VMNT22( 10, 3 )
1725:          end if
1726:      end if
1727:      750 continue
1728: C
1729: C-----
1730: C .... remove processed particles from flight stack .....
1731: C-----
1732: C
1733:      II = 0
1734:      *VOCL LOOP,NOVREC
1735:      do 760 N = 1, NFFL(NZONE+1)
1736:          if ( IZFFL(N).ne.MZONE ) then
1737:              II = II + 1
1738:              LSFFL(II) = LSFFL(N)
1739:              IZFFL(II) = IZFFL(N)

```

```

1740:          end if
1741:      760 continue
1742:          NFFL(NZONE+1) = NFFL(NZONE+1) - NFFL(MZONE)
1743:          NFFL(MZONE) = 0
1744:
1745:      return
1746:  end

```

src/mvp/fregas.f

```

1:      subroutine FREGAS( IOW, IBP, ICLN, CTH, ISOSC, IFRGS,
2:      &
3:      &
4:      &
5:      &
6:      &
7:      &
8:      &
9:      &
10:     &
11:     &
12:     &
13:     &
14: C=====
15: C  PURPOSE: ANALYSIS OF THERMAL SCATTERING BY FREE GAS MODEL
16: C  IBP(I) : STACK, ICLN(I) : COLLIDING NUCLIDE,
17: C  NCOLF : NUMBER OF PARTICLES
18: C  ----> CTH(I) : SCATTERING ANGLE COSINE IN LAB SYSTEM,
19: C  ISOSC : STACK FOR ISOTROPICALLY SCATTERED NEUTRONS
20: C  WITHOUT ENERGY CHANGE (NWOEC)
21: C  IFRGS : STACK FOR NEUTRONS PROCESSED BASED ON FREE-GAS
22: C  MODEL (NCOLF CHANGED)
23: C=====
24: C CALLED IN: NEUTR
25: C CALLS : RANU2
26: C
27: C=====
28: C
29: C  === BANK DATA TO BE UPDATED ===
30: C
31: C  AAA,BBB,CCC: DIRECTION COSINE
32: C  EEE : ENERGY
33: C
34: C=====
35: C  implicit real*8(A-H,O-Z)
36: C
37: C  integer JDEBG(*)
38: C
39: C  ..... flag options
40: C  integer JMICE(*), JMACE(*)
41: C
42: C  ..... geometry data
43: C  integer KZREG(NZONE)
44: C
45: C**** INPUT
46: C
47: C  integer IBP(NCOLF), ICLN(NCOLF)
48: C
49: C**** OUTPUT
50: C
51: C  integer ISOSC(NBANK), IFRGS(NBANK)
52: C  real CTH(NCOLF)
53: C
54: C**** CROSS SECTION DATA IN CX ARRAY
55: C
56: C  real ETOP, EBOT, ETHMAX, AMLIM
57: C  real XLIB1(NUC,11)
58: C  real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC)
59: C  integer MMAC(NBANK,2), LMAC(8), LMIC(8)
60: C
61: C**** STACK
62: C
63: C  integer LSDED(NBANK)
64: C
65: C**** PARTICLE BANK

```

```

66: C
67: C  real EEE(NBANK), WWW(NBANK)
68: C  real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
69: C  integer IZZ(NBANK), IBREG(NBANK), IGG(NBANK)
70: C
71: C**** TALLY ARRAY
72: C
73: CCCC real*8 WCNTR(*), NCNTR(*)
74: C  real DNZON(NUC,NZONE,2), ENGYB(*)
75: C  real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*),
76: C  & RMICSM(NGP1,NGP1,NREG,NUC,JSCTM,*)
77: C  integer KYPOS(3)
78: C  real*8 RMIMU(NGP1,NGP1+1,NREG,NUC,*),WMIMU(NGP1,NGP1+1,NREG,NUC,*)
79: C
80: C**** WORKING AREA
81: C
82: C  integer IWK10(NBANK), IWK11(NBANK), IWK12(NBANK)
83: C  real WK6(NBANK), WK7(NBANK), WK8(NBANK), WK9(NBANK)
84: C  real WK10(NBANK)
85: C  real WK2(NBANK), WK3(NBANK), WK4(NBANK), WK5(NBANK)
86: C  real R(7*NBANK)
87: C
88: C  parameter( PI = 3.141592653589793200D+00 )
89: C  parameter( PI2 = 2.0D0*PI )
90: C  parameter( PIH = 0.5D0*PI )
91: C
92: C
93: C**** MAKE TEMPORARY STACKS ISOSC FOR ATW > AMLIM NWOEC
94: C  CTH : SCATTERING ANGLE COSINE
95: C  IFRGS FOR FREE GAS NF (NCOLF MODIFIED)
96: C  WK2 : ATW
97: C  WK3 : 1/(ATW+1)/(ATW+1)
98: C  WK4 : ATW/KTeff*EEE WHEN EEE<=5eV
99: C  WK10: ATW/KT*EEE
100: C
101: C
102: C  DZERO = 0
103: C  NWOEC = 0
104: C  NF = 0
105: C
106: C*VOCL LOOP,NOVREC
107: C  do 100 I = 1, NCOLF
108: C    ATW = XLIB1(ICLN(I),1)
109: C    if ( ATW.le.AMLIM ) then
110: C      NF = NF + 1
111: C      IFRGS(NF) = IBP(I)
112: C      WK2(NF) = ATW
113: C      WK3(NF) = XLIB1(ICLN(I),7)
114: C      WK10(NF) = XLIB1(ICLN(I),9)*EEE(IBP(I))
115: C      if ( EEE(IBP(I)).le.5. ) then
116: C        WK4(NF) = XLIB1(ICLN(I),11)*EEE(IBP(I))
117: C      else
118: C        WK4(NF) = WK10(NF)
119: C      end if
120: C      IWK12(NF) = I
121: C    else
122: C      NWOEC = NWOEC + 1
123: C      ISOSC(NWOEC) = IBP(I)
124: C      WK5(NWOEC) = ATW
125: C    end if
126: C  100 continue
127: C
128: C  NCOLF = NF
129: C
130: C**** ISOTROPIC SCATTERING IN CM SYSTEM WITHOUT ANY ENERGY CHANGE

```

src/mvp/fregas.f

```

131: C      SCATTERING ANGLE COSINE IN LAB SYSTEM -----> CTH
132: C
133:       if ( NWOEC.gt.0 ) then
134:         call RANU2( IRAND, R, NWOEC, ICON )
135: *VOCL LOOP,NOVREC
136:       do 110 I = 1, NWOEC
137:         XMU      = 2.*R(I) - 1.
138:         XXX      = 1. + 2.*WK5(I)*XMU + WK5(I)*WK5(I)
139:         CTH(I)   = (1.+XMU*WK5(I)) /SQRT(XXX)
140:         if ( EEE(ISOSC(I)).gt.0.625 ) then
141:           ENEW    = EEE(ISOSC(I))*XXX/((1.+WK5(I))*(1.+WK5(I)))
142:           ENEW    = MAX(DBLE(EBOT),ENEW)
143:           EEE(ISOSC(I)) = ENEW
144:         end if
145:       110 continue
146:     end if
147: C
148: C
149: C**** CALCULATION BASED ON FREE GAS MODEL
150: C
151:       if ( NCOLF.gt.0 ) then
152: C
153: C..... JFREE =0 : FIRST TRIAL FOR SHORT COLLISION TIME APPROXIMATION
154: C          : BY USING THE SAME METHOD AS THAT FOR FREE GAS MODEL
155: C          JFREE = 0
156: C
157:       call RANU2( IRAND, R, NF*7, ICON )
158: C
159:       NF2      = NF*2
160:       NF3      = NF*3
161:       NF4      = NF*4
162:       NF5      = NF*5
163:       NF6      = NF*6
164: C
165: C..... WK5      : V(TARGET NUCLIDE) / V(NEUTRON) : XT
166: C          X2     : WK5**2
167: C          (WK6,WK7,WK8) : DIRECTION COSINE OF TARGET
168: C          WK9    : SQRT(1.+X2-2*XMU*XT)
169: C          IWK10(NN) : IFRGS(IWK10(J))---> STACK
170: C
171:       NN      = 0
172: C
173: *VOCL LOOP,NOVREC
174:       do 120 I = 1, NF
175:         if ( (4.*R(I)*R(I)).gt.(PI*WK4(I)*(1.-R(I))*(1.-R(I))) )
176:           &       then
177:           X2      = -LOG(R(NF+I)*R(NF2+I)) /WK4(I)
178:         else
179:           COS2    = COS(PIH*R(NF+I))
180:           COS2    = COS2*COS2
181:           X2      = -(LOG(R(NF2+I))+LOG(R(NF3+I))*COS2) /WK4(I)
182:         end if
183: C
184:       WK5(I)    = SQRT(X2)
185: C
186: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
187: C
188:       WK6(I)    = 2.*R(NF4+I) - 1.0
189:       SINT      = SQRT(1.-WK6(I)*WK6(I))
190:       FAI       = PI2*R(NF5+I)
191:       WK7(I)    = SINT*COS(FAI)
192:       WK8(I)    = SINT*SIN(FAI)
193:       IP        = IFRGS(I)
194:       XMU       = WK6(I)*AAA(IP) + WK7(I)*BBB(IP) + WK8(I)*CCC(IP)
195: C

```

```

196:       WK9SQ    = max(1.+X2-2.*XMU*WK5(I),0.d0)
197:       WK9(I)   = sqrt(WK9SQ)
198: C
199:       if ( WK9(I).lt.R(NF6+I)*(1.+WK5(I)) ) then
200:         NN      = NN + 1
201:         IWK10(NN) = I
202:       end if
203: 120 continue
204: C
205: C**** RESAMPLING FOR REJECTED PARTICLES
206: C
207:       NREJ     = 0
208: 130 continue
209: C
210:       if ( NN.gt.0 ) then
211: C
212:         NF      = NN
213:         NN      = 0
214:         call RANU2( IRAND, R, NF*7, ICON )
215: C
216:         NF2     = NF*2
217:         NF3     = NF*3
218:         NF4     = NF*4
219:         NF5     = NF*5
220:         NF6     = NF*6
221: C
222: *VOCL LOOP,NOVREC
223:       do 140 J = 1, NF
224:         I       = IWK10(J)
225:         AEKT    = WK4(I)
226:         if ( (4.*R(J)*R(J)).gt.(PI*AEKT*(1.-R(J))*(1.-R(J))) )
227:           &       then
228:           X2     = -LOG(R(NF+J)*R(NF2+J)) /WK4(I)
229:         else
230:           COS2   = COS(PIH*R(NF+J))
231:           COS2   = COS2*COS2
232:           X2     = -(LOG(R(NF2+J))+LOG(R(NF3+J))*COS2) /AEKT
233:         end if
234:         WK5(I)  = SQRT(X2)
235: C
236: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
237: C
238:       WK6(I)    = 2.*R(NF4+J) - 1.0
239:       SINT      = SQRT(1.-WK6(I)*WK6(I))
240:       FAI       = PI2*R(NF5+J)
241:       WK7(I)    = SINT*COS(FAI)
242:       WK8(I)    = SINT*SIN(FAI)
243:       IP        = IFRGS(I)
244:       XMU       = WK6(I)*AAA(IP) + WK7(I)*BBB(IP) + WK8(I)*
245:       &         CCC(IP)
246: C
247:       WK9SQ    = max(1.+X2-2.*XMU*WK5(I),0.d0)
248:       WK9(I)   = sqrt(WK9SQ)
249: C
250:       if ( WK9(I).lt.R(NF6+J)*(1.+WK5(I)) ) then
251:         NN      = NN + 1
252:         IWK10(NN) = I
253:       end if
254: 140 continue
255: C
256:       go to 130
257: C
258:     end if
259: C
260: C

```

src/mvp/fregas.f

```

261: C**** CALCULATION OF EEE AND (AAA,BBB,CCC)
262: C
263: C      WK2      : ATW
264: C      WK3      : 1/(ATW+1)/(ATW+1)
265: C..... WK5      : V(TARGET NUCLIDE) / V(NEUTRON) : XT
266: C      (WK6,WK7,WK8) : DIRECTION COSINE OF TARGET
267: C      WK9      : SQRT(1.+X2-2*XMU*XT)
268: C      IFRGS      : STACK
269: C
270: C**** FIRST TRIAL IN CASE OF SHORT COLLISION TIME APPROXIMATION
271: C      if ( JFREE.eq.0 ) then
272: C          JFREE = 1
273: C
274: C      cm1994      CALL RANU2( IRAND, R, NCOLF*2, ICON )
275: C      call RANU2( IRAND, R, NCOLF*3, ICON )
276: C      N2          = NCOLF*2
277: C
278: C      *VOCL LOOP,NOVREC
279: C      do 150 I = 1, NCOLF
280: C          IP      = IFRGS(I)
281: C          U1      = 2.*R(I) - 1.0
282: C          SINT    = SQRT(1.-U1*U1)
283: C          FAI     = PI2*R(NCOLF+I)
284: C          U2      = SINT*COS(FAI)
285: C          U3      = SINT*SIN(FAI)
286: C          O1      = (WK9(I)*U1+WK5(I)*WK6(I))*WK2(I)
287: C          O2      = (WK9(I)*U2+WK5(I)*WK7(I))*WK2(I)
288: C          O3      = (WK9(I)*U3+WK5(I)*WK8(I))*WK2(I)
289: C          X1      = O1 + AAA(IP)
290: C          X2      = O2 + BBB(IP)
291: C          X3      = O3 + CCC(IP)
292: C
293: C          XX2      = X1*X1 + X2*X2 + X3*X3
294: C
295: C          RATIO    = XX2*WK3(I)
296: C          C1       = (WK4(I)-WK10(I))*(RATIO-1.D0)
297: C
298: C          if ( C1.lt.LOG(R(I+N2))*WK2(I) ) then
299: C              NN      = NN + 1
300: C              IWK10(NN) = I
301: C              IWK11(NN) = I
302: C          else
303: C      CM1994      ENEW      = EEE(IP) * XX2 * WK3(I)
304: C      ENEW      = EEE(IP)*RATIO
305: C      CM1991-12-18
306: C      ENEW      = MAX( EBOT,ENEW )
307: C      CM      EEE(IP) = MIN( ETHMAX,ENEW )
308: C      ccccc      EEE(IP) = MAX( EBOT,ENEW )
309: C      EEE(IP) = MAX(DBLE(EBOT),ENEW)
310: C
311: C          XNOLM    = 1./SQRT(XX2)
312: C          CTH(NWOEC+I) = (X1*AAA(IP)+X2*BBB(IP)+X3*CCC(IP))
313: C          &          *XNOLM
314: C          AAA(IP) = X1*XNOLM
315: C          BBB(IP) = X2*XNOLM
316: C          CCC(IP) = X3*XNOLM
317: C          end if
318: C
319: C      150      continue
320: C
321: C          if ( NN.gt.0 ) then
322: C              NREJ      = NN
323: C              go to 130
324: C          end if
325: C

```

```

326: C**** SUBSEQUENT TRIAL IN CASE OF SHORT COLLISION TIME APPROXIMATION
327: C      else
328: C
329: C          call RANU2( IRAND, R, NREJ*3, ICON )
330: C          N2      = NREJ*2
331: C
332: C      *VOCL LOOP,NOVREC
333: C      do 160 J = 1, NREJ
334: C          I      = IWK11(J)
335: C          IP      = IFRGS(I)
336: C          U1      = 2.*R(J) - 1.0
337: C          SINT    = SQRT(1.-U1*U1)
338: C          FAI     = PI2*R(NREJ+J)
339: C          U2      = SINT*COS(FAI)
340: C          U3      = SINT*SIN(FAI)
341: C          O1      = (WK9(I)*U1+WK5(I)*WK6(I))*WK2(I)
342: C          O2      = (WK9(I)*U2+WK5(I)*WK7(I))*WK2(I)
343: C          O3      = (WK9(I)*U3+WK5(I)*WK8(I))*WK2(I)
344: C          X1      = O1 + AAA(IP)
345: C          X2      = O2 + BBB(IP)
346: C          X3      = O3 + CCC(IP)
347: C
348: C          XX2      = X1*X1 + X2*X2 + X3*X3
349: C
350: C          RATIO    = XX2*WK3(I)
351: C          C1       = (WK4(I)-WK10(I))*(RATIO-1.D0)
352: C
353: C          if ( C1.lt.LOG(R(J+N2))*WK2(I) ) then
354: C              NN      = NN + 1
355: C              IWK10(NN) = I
356: C              IWK11(NN) = I
357: C          else
358: C              ENEW      = EEE(IP)*RATIO
359: C      CM1991-12-18
360: C      ENEW      = MAX( EBOT,ENEW )
361: C      CM      EEE(IP) = MIN( ETHMAX,ENEW )
362: C      ccccc      EEE(IP) = MAX( EBOT,ENEW )
363: C      EEE(IP) = MAX(DBLE(EBOT),ENEW)
364: C
365: C          XNOLM    = 1./SQRT(XX2)
366: C          CTH(NWOEC+I) = (X1*AAA(IP)+X2*BBB(IP)+X3*CCC(IP))
367: C          &          *XNOLM
368: C          AAA(IP) = X1*XNOLM
369: C          BBB(IP) = X2*XNOLM
370: C          CCC(IP) = X3*XNOLM
371: C          end if
372: C
373: C      160      continue
374: C
375: C          if ( NN.gt.0 ) then
376: C              NREJ      = NN
377: C              go to 130
378: C          end if
379: C
380: C          end if
381: C
382: C      C***** scattering matrix tally by collision
383: C
384: C          if ( JSCTM.ne.0.and.JRTCL.ne.0.and.
385: C      &      (JMICE(4).ne.0.or.JMACE(4).ne.0) ) then
386: C              do J = 1, NCOLF
387: C                  WK9(J) = EEE(IFRGS(J))
388: C              end do
389: C              call BSVDEC( ENGYB, NGP1+1, WK9, IWK10, NCOLF )
390: C              do J = 1, NCOLF

```


src/mvp/fregas.f

```
391:          I = IWK12(J)
392:          N = ICLN(I)
393:          IP = IFRGS(J)
394:          DN = DNZON(N,IZZ(IP),1)
395:          if ( DN.gt.DZERO ) then
396:             IE = IGG(IP)
397:             JE = IWK10(J)
398:             IR = KZREG(IZZ(IP))
399:             if ( JTLLT.ne.0 ) IR = IBREG(IP)
400:             DNF = DN*WWW(IP)/SMAC(IP,MMAC(IP,1),LMAC(1))
401:             DNFLXSM(IE,JE,IR,N,KYPOS(1)) =
402: &             DNFLXSM(IE,JE,IR,N,KYPOS(1)) + DNF
403: c             RMICSM(IE,JE,IR,N,KYPOS(1)) =
404: c &             RMICSM(IE,JE,IR,N,KYPOS(1)) +
405: c &             SMIC(IP,N,LMIC(4))*DNF
406:             RMICSM(IE,JE,IR,N,1,KYPOS(1)) =
407: &             RMICSM(IE,JE,IR,N,1,KYPOS(1)) + WWW(IP)
408:             if ( JSCTM.gt.1 ) then
409:                XMU = CTH(NWOEC+I)
410:                do K = 2, JSCTM
411:                   XMUN = XMU**(K-1)*WWW(IE)
412:                   RMICSM(IE,JE,IR,N,K,KYPOS(1)) =
413: &                   RMICSM(IE,JE,IR,N,K,KYPOS(1)) + XMUN
414:                end do
415:             end if
416:         end if
417:     end do
418: end if
419: C
420: C***** scattering mubar tally by collision
421: C
422:     if (JSCMU.ne.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0) ) then
423:         if ( JSCTM.eq.0.or.JRTCL.eq.0 ) then
424:             do J = 1, NCOLF
425:                WK9(J) = EEE(IFRGS(J))
426:             end do
427:             call BSVDEC( ENGYB, NGP1+1, WK9, IWK10, NCOLF )
428:         end if
429:         do J = 1, NCOLF
430:             I = IWK12(J)
431:             N = ICLN(I)
432:             IP = IFRGS(J)
433:             IE = IGG(IP)
434:             JE = IWK10(J)
435:             IR = KZREG(IZZ(IP))
436:             if ( JTLLT.ne.0 ) IR = IBREG(IP)
437:             XMU = CTH(NWOEC+I) + 1
438:             WMIMU(IE,JE,IR,N,KYPOS(1)) =
439: &             WMIMU(IE,JE,IR,N,KYPOS(1)) + WWW(IP)
440:             RMIMU(IE,JE,IR,N,KYPOS(1)) =
441: &             RMIMU(IE,JE,IR,N,KYPOS(1)) + WWW(IP)*XMU
442:             WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
443: &             WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)
444:             RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
445: &             RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)*XMU
446:         end do
447:     end if
448: C
449: end if
450: C
451: return
452: end
```

src/mvp/getmac.f

```

1:      subroutine GETMAC( IRAND, NN,      IBP,      MAT,      JEIGN, JAMXC,
2:      &                  JDEBG, NBANK, EEE, MB,      NUC,      NSTAL,
3:      &                  NSMAC, SMAC,      LMIC,      KMAC,      MMAC,      NSMIC, SMIC,
4:      &                  LMIC, KSPI,
5:      &                  LPDEN, INUCT, DENST, NMAT,
6:      &                  NMT,      CX,      MCX,      KLIB1, KLIB2, XLIB1,
7:      W      IF3,      IFI,      STOT,      SNUF,      SLOS,      WK6,
8:      W      WK7,      WK8,      R,      SMCW )
9: C=<MVP>=====
10: C PURPOSE      : TO GET OR PREPARE MACRO CROSS SECTION
11: C CALLED IN : FLIONE, FLIGHT, VMNTR1
12: C CALLS      : GMICN1
13: C-----
14: C <comment> calculations specifying different MB values may not produce
15: C exactly the same result because KMAC(*,*,2) - colliding nuclide -
16: C is determined in the course of sigma-total calculation using random
17: C numbers and sigma-total may be re-calculated even in a particle
18: C flight path when MB is less than NMAT. (July 1997 M.S.)
19: C-----
20: C=====
21:      integer JDEBG(*)
22: C
23: C .... NN : PARTICLE NUMBER      IBP : BANK POINTERS      MAT: MAT #
24: C
25:      integer NN, IBP(NBANK), MAT
26: C
27: C .... MATERIAL DATA .....
28: C
29:      integer LPDEN(NMAT+1), INUCT(*)
30:      real DENST(*)
31: C
32: C .... Cross section .....
33: C
34:      real CX(MCX), XLIB1(NUC,11)
35: C##<2007/03/14:PN3:
36: C##      integer KLIB1(NUC,30), KLIB2(NUC,NMT,13)
37: C##      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
38: C##>
39: C
40: C .... BANK .....
41: C
42:      real EEE(NBANK)
43: C
44: C .... SIGMA BANK .....
45: C
46:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC)
47:      integer MB, KMAC(NBANK,MB,2), MMAC(NBANK,2), LMIC(8), LMIC(8)
48:      integer KSPI(NBANK,NUC)
49: C
50: C .... WORKING ARRAYS ...
51: C
52:      integer IF3(NBANK), IFI(NBANK)
53:      real R(2*NBANK), STOT(NBANK), SNUF(NBANK), SLOS(NBANK)
54:      real WK6(NBANK), WK7(NBANK), WK8(NBANK)
55:      real SMCW(NBANK,NSMIC)
56: C
57:      include '../shared/INC/_IOUNIT'
58:      include 'INC/_IOUNIT2'
59: C
60: C
61: C=====
62: C DETERMINE BANK NUMBER TO BE USED ( MMAC(IBP(I),1) )
63: C=====
64: C
65: C

```

```

66:      N3      = 0
67:      do 100 I = 1, NN
68:          IF3(I) = 0
69:          100 continue
70: C-----
71: C IF3 : STATUS INDICATOR      0 = TREATMENT UNDETERMINED YET
72: C      1 = MACRO DATA MUST BE CALCULATED NEWLY
73: C     -1 = MACRO DATA ALREADY PREPARED
74: C-----
75: C-----
76: C COUNT THE NUMBER OF PARTICLES WHOSE DATA MUST BE CALCULATED NEWLY.
77: C IFI : GATHERED BANK POINTERS
78: C-----
79:      N2      = 0
80: C
81:      do 120 K = 1, MB + 1
82:          if ( N3.eq.NN ) go to 130
83: *VOCL LOOP,NOVREC
84:          do 110 I = 1, NN
85:              if ( IF3(I).eq.0 ) then
86: C-----
87: C MACRO DATA ARE NOT IN BANK, MUST BE CALCULATED NEWLY!
88: C-----
89:                  if ( MMAC(IBP(I),2).lt.K ) then
90:                      IF3(I) = 1
91:                      KMAC0 = MIN(MMAC(IBP(I),2)+1,MB)
92:                      MMAC(IBP(I),2) = KMAC0
93:                      MMAC(IBP(I),1) = KMAC0
94:                      KMAC(IBP(I),KMAC0,2) = MAT
95:                      N2 = N2 + 1
96:                      IFI(N2) = IBP(I)
97: C-----
98: C MACRO DATA FOR MATERIAL 'MAT' MAY EXIST IN BANK
99: C-----
100:                      else
101:                          if ( KMAC(IBP(I),K,2).eq.MAT ) then
102:                              IF3(I) = -1
103:                              MMAC(IBP(I),1) = K
104:                          end if
105:                      end if
106: C
107:                  if ( IF3(I).ne.0 ) N3 = N3 + 1
108:                  end if
109:                  110 continue
110:                  120 continue
111: C-----
112: C 130 continue
113: C-----
114: C N2 = 0 : NO NEED TO CALCULATE MACRO DATA !
115: C-----
116:      if ( N2.eq.0 ) return
117: C
118: C=====
119: C CALCULATE MACRO DATA FOR SOME PARTICLES
120: C=====
121: C
122:      if ( JAMXC.ne.0 ) then
123:          do 150 N = LPDEN(MAT), LPDEN(MAT+1) - 1
124:              NI = INUCT(N)
125:              NNN = 0
126:              do 140 I = 1, N2
127:                  if ( KSPI(IFI(I),NI).eq.0 ) then
128:                      NNN = NNN + 1
129:                      IF3(NNN) = IFI(I)
130:                      WK6(NNN) = EEE(IFI(I))

```

src/mvp/getmac.f

```

131:         end if
132: 140      continue
133:         if ( NNN.gt.0 ) then
134:             call GMICN1( NI, NNN, IF3, WK6, IRAND, NUC, NMT,
135: &                     NBANK, NSMIC, SMIC, LMIC, KSPI,
136: &                     CX, CX, MCX, KL1B1, KL1B2, XL1B1,
137: &                     STOT, SNUF, SLOS, WK7, WK8, R, SMCW,
138: &                     RNU, NUC_MAX )
139:         end if
140: 150      continue
141:     end if
142: C-----
143: C CLEAR OR INITIALIZE TEMPORARY ARRAYS
144: C (IF3 IS USED AS TEMPORARY STORAGE OF COLLIDING NUCLIDE NUMBER)
145: C-----
146:     NI = INUCT(LPDEN(MAT))
147:     RHO = DENST(LPDEN(MAT))
148:     do 160 I = 1, N2
149:         IF3(I) = NI
150: C         STOT(I) = 0.0
151:         STOT(I) = SMIC(IFI(I),NI,LMIC(1))*RHO
152:         if ( JEIGN.ne.0 ) then
153: C             SNUF(I) = 0.0
154: C             SLOS(I) = 0.0
155:             SNUF(I) = SMIC(IFI(I),NI,LMIC(2))*RHO
156:             SLOS(I) = SMIC(IFI(I),NI,LMIC(8))*RHO
157:         end if
158: 160 continue
159: C-----
160: C CALCULATE MACRO CROSS SECTION BY SUMMATION OF MICRO CROSS SECTIONS
161: C-----
162:         if ( (LPDEN(MAT+1)-LPDEN(MAT)).ge.2 ) then
163: C
164:             do 180 N = LPDEN(MAT) + 1, LPDEN(MAT+1) - 1
165:                 call RANU2( IRAND, R, N2, ICON )
166:                 NI = INUCT(N)
167:                 RHO = DENST(N)
168:                 do 170 I = 1, N2
169:                     ST = SMIC(IFI(I),NI,LMIC(1))*RHO
170: C ..... IF3 : NUCLIDE TO COLLIDE .....
171: C
172:                     STOT(I) = STOT(I) + ST
173:                     if ( ST.gt.STOT(I)*R(I) ) IF3(I) = NI
174:                     if ( JEIGN.ne.0 ) then
175:                         SNUF(I) = SNUF(I) + SMIC(IFI(I),NI,LMIC(2))*RHO
176:                         SLOS(I) = SLOS(I) + SMIC(IFI(I),NI,LMIC(8))*RHO
177:                     end if
178: 170          continue
179: 180          continue
180:             end if
181: C-----
182: C PUT CALCULATED MACROSCOPIC DATA IN BANK
183: C-----
184: *VOCL LOOP,NOVREC
185:     do 190 I = 1, N2
186:         KMAC(IFI(I),MMAC(IFI(I),1),1) = IF3(I)
187: c##<2007/03/14:PN3:
188:         KMAC(IFI(I),MMAC(IFI(I),1),2) = MAT
189: c##>
190:         SMAC(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOT(I)
191:         if ( JEIGN.ne.0 ) then
192:             SMAC(IFI(I),MMAC(IFI(I),1),LMAC(2)) = SNUF(I)
193:             SMAC(IFI(I),MMAC(IFI(I),1),LMAC(8)) = SLOS(I)
194:         end if
195: 190 continue

```

```

196: C
197: C-----
198: C OTHER MACROSCOPIC CROSS SECTIONS IF NECESSARY
199: C-----
200:         do 240 K = 2, 8
201:             if ( JEIGN.eq.0 .or. (K.ne.2.and.K.ne.8) ) then
202:                 if ( LMAC(K).gt.0 ) then
203:                     do 200 I = 1, N2
204:                         STOT(I) = 0.0
205: 200          continue
206:                     do 220 N = LPDEN(MAT), LPDEN(MAT+1) - 1
207:                         NI = INUCT(N)
208:                         RHO = DENST(N)
209:                         do 210 I = 1, N2
210:                             STOT(I) = STOT(I) + SMIC(IFI(I),NI,LMIC(K))*RHO
211: 210          continue
212: 220          continue
213: *VOCL LOOP,NOVREC
214:                 do 230 I = 1, N2
215:                     SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOT(I)
216: 230          continue
217:                 end if
218:             end if
219: 240 continue
220: C-----
221: C DATA CHECK ONLY FOR DEBUGGING (1990/3/15)
222: C-----
223:         if ( JDEBG(1).ne.0 ) then
224:             do 250 I = 1, N2
225:                 if ( MMAC(IFI(I),1).lt.1
226: &                 .or. MMAC(IFI(I),1).gt.MMAC(IFI(I),2)
227: &                 .or. MMAC(IFI(I),2).lt.0 .or. MMAC(IFI(I),2).gt.MB ) then
228:                     write(IOG,*) ' == INVALID MMAC DATA IN GETMAC ! == '
229:                     write(IOG,*) ' POS = ', IFI(I), ' MMAC = ',
230: &                     MMAC(IFI(I),1), MMAC(IFI(I),2)
231:                 end if
232: 250          continue
233:                 do 270 M = 1, MB
234:                     do 260 I = 1, N2
235:                         if ( KMAC(IFI(I),M,1).lt.0 .or. KMAC(IFI(I),M,1).gt.NUC
236: &                         .or. KMAC(IFI(I),M,2).lt.0
237: &                         .or. KMAC(IFI(I),M,2).gt.NMAT ) then
238:                             write(IOG,*) ' == INVALID KMAC DATA IN GETMAC ! == '
239:                             write(IOG,*) ' POS = ', IFI(I), ' M= ', M,
240: &                             ' KMAC = ', KMAC(IFI(I),M,1),
241: &                             KMAC(IFI(I),M,2)
242:                         end if
243: 260          continue
244: 270          continue
245:                 end if
246: C
247:         return
248:     end

```

src/mvp/getmic.f

```

1:      subroutine GETMIC( NN,   IBP,   IRAND, NUC,   NMT,
2:      &                  NBANK, NSMIC, EEE,   SMIC, LMIC, KSPI,  CX,
3:      &                  ICX,   MCX,   KLIB1, KLIB2, XLIB1,
4:      &                  E,     E2,   IEP,   IEP2,
5:      &                  IBP2, DE,   IP1,   R,     SMCW,  RNU,
6:      c##<2007/03/14:PN3:
7:      &                  NUC_MAX )
8:      c##>
9:      C
10:     C   &   SGTAL, CRES, KCRES, NSTAL, MCRES,
11:     C=<MVP>=====
12:     C PURPOSE: GET MICRO CROSS SECTIONS AND PUT THEM IN SIGMA-BANK.
13:     C (NEUTRON )
14:     C CALLED IN: NEUTR, SOURCE
15:     C
16:     C-----
17:     C
18:     C .... NN: NUMBER OF PARTICLES  IBP: POINTER .....
19:     C
20:     C      integer IBP(NN)
21:     C
22:     C .... BANK .....
23:     C
24:     C      real EEE(NBANK), SMIC(NBANK,NUC,NSMIC)
25:     C      real SGTAL(NBANK,NSTAL)
26:     C      integer KSPI(NBANK,NUC), LMIC(8)
27:     C ... RNU : nu-bar
28:     c##<2007/03/14:PN3:
29:     c##C      RNU(NBANK,NUC,1) : nu-total
30:     c##C      RNU(NBANK,NUC,2) : nu-delayed
31:     c##C      RNU(NBANK,NUC,3) : nu-prompt
32:     c##      real RNU(NBANK,NUC,3)
33:     C      RNU(NBANK,NUC_MAX,1) : nu-total
34:     C      RNU(NBANK,NUC_MAX,2) : nu-delayed
35:     C      RNU(NBANK,NUC_MAX,3) : nu-prompt
36:     C      real RNU(NBANK,NUC_MAX,3)
37:     c##>
38:     C
39:     C .... CROSS SECTION & POINTERS .....
40:     C
41:     C      real CX(MCX), XLIB1(NUC,11)
42:     c##<2007/03/14:PN3:
43:     c##      integer ICX(MCX), KLIB1(NUC,30), KLIB2(NUC,NMT,13)
44:     c##      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
45:     c##>
46:     C
47:     C      real CRES(MCRES)
48:     C      integer KCRES(NSTAL,4)
49:     C
50:     C .... WORKING ARRAY ....
51:     C
52:     C      real E(NBANK), E2(NBANK), R(2*NBANK), DE(NBANK)
53:     C      integer IEP(NBANK), IEP2(NBANK), IBP2(NBANK), IP1(NBANK)
54:     C      real SMCW(NBANK,NSMIC)
55:     C
56:     C      integer MNEG
57:     C      parameter( MWARN   = 2000 )
58:     C
59:     C      parameter( MT1   = 1, MT2   = 2, MT4   = 4, MT18 = 18, MT101 =
60:     C      &      101, MT94   = 94, MT95   = 95, MT96   = 96 )
61:     C
62:     C      include 'INC/_FLAGS'
63:     C      include '../shared/INC/_IUNIT'
64:     C
65:     C      data MNEG /0/

```

```

66:     C
67:     C-----
68:     C      GATHER ENERGY
69:     C-----
70:     C
71:     C      do 100 I = 1, NN
72:     C          E(I)   = EEE(IBP(I))
73:     C      100 continue
74:     C
75:     C-----
76:     C--- LOOP FOR NUCLIDES ---
77:     C-----
78:     C
79:     C      MT1   = 1
80:     C      MT2   = 2
81:     C
82:     C      do 360 N = 1, NUC
83:     C          NPTS = KLIB1(N,2)
84:     C          LNU  = KLIB1(N,14)
85:     C          NNU  = KLIB1(N,9)
86:     C          N2   = LNU*NNU
87:     C          LNUD = KLIB1(N,23)
88:     C          NNUD = KLIB1(N,22)
89:     C
90:     C-----
91:     C      ENERGY MESH POINT #
92:     C-----
93:     C
94:     C          LEMESH = KLIB1(N,1)
95:     C          call BSVDEC( CX(LEMESH), NPTS, E, IEP, NN )
96:     C
97:     C *VOCL LOOP,NOVREC
98:     C      do 110 I = 1, NN
99:     C          KSPI(IBP(I),N) = IEP(I)
100:     C          LE   = LEMESH + IEP(I)
101:     C          DE(I) = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
102:     C      110 continue
103:     C
104:     C ..... CLEAR ALL CROSS SECTIONS .....
105:     C ( FROM JAN 15 1992 : TO PREVENT ERRONEOUS REACTION IN
106:     C MIXED PARTICLE MODE )
107:     C
108:     C      do 130 K = 1, NSMIC
109:     C          do 120 I = 1, NN
110:     C              SMCW(I,K) = 0.0
111:     C          120 continue
112:     C      130 continue
113:     C      do 132 K = 1, 3
114:     C          do 131 I = 1, NN
115:     C              RNU(IBP(I),N,K) = 0.0
116:     C          131 continue
117:     C      132 continue
118:     C
119:     C-----
120:     C      TOTAL      (MT = 1)  & ELASTIC (MT = 2)
121:     C-----
122:     C
123:     C          L1      = KLIB2(N,MT1,11) - KLIB2(N,MT1,3)
124:     C          L2      = KLIB2(N,MT2,11) - KLIB2(N,MT2,3)
125:     C *VOCL LOOP,NOVREC
126:     C      do 140 I = 1, NN
127:     C          LL      = L1 + IEP(I)
128:     C          SMCW(I,LMIC(1)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
129:     C          if ( KLIB1(N,15).ne.1 .or. E(I).gt.XLIB1(N,4) ) then
130:     C              LL2      = L2 + IEP(I)

```

src/mvp/getmic.f

```

131:          SMCW(I,LMIC(4)) = CX(LL2+1) + DE(I)*(CX(LL2)-CX(LL2+1))
132:          end if
133:
134: 140      continue
135:
136: C-----
137: C      INELASTIC (MT = 4)
138: C-----
139:
140:          if ( LMIC(6).ne.0 ) then
141: CCCC      MT      = 4
142:          if ( KLIB2(N,MT4,1).ne.0 ) then
143:              L0      = KLIB2(N,MT4,11) - KLIB2(N,MT4,3)
144: *VOCL LOOP,NOVREC
145:              do 150 I = 1, NN
146:                  if ( IEP(I).ge.KLIB2(N,MT4,3)
147:                      & .and.IEP(I).lt.KLIB2(N,MT4,4) ) then
148:                      LL      = L0 + IEP(I)
149:                      SMCW(I,LMIC(6)) = CX(LL+1) + DE(I)*
150:                      (CX(LL)-CX(LL+1))
151:                  end if
152:              150      continue
153:          end if
154:      end if
155: C-----
156: C-----
157: C      FISSION (MT = 18)
158: C-----
159: C
160: Ccc      MT      = 18
161:
162:          if ( KLIB2(N,MT18,1).ne.0 ) then
163:
164: Ccccc      CALL      CXSINT(NN,IBP, N,MT18,
165: CccccX          SMIC(1,N,LMIC(3)), CX, MCX, KLIB2,NUC,NMT,
166: CccccW          IEP ,DE, DE1, NBANK)
167:
168:              L0      = KLIB2(N,MT18,11) - KLIB2(N,MT18,3)
169:
170: *VOCL LOOP,NOVREC
171:              do 160 I = 1, NN
172:                  if ( IEP(I).ge.KLIB2(N,MT18,3)
173:                      & .and.IEP(I).lt.KLIB2(N,MT18,4) ) then
174:                      LL      = L0 + IEP(I)
175:                      SMCW(I,LMIC(3)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
176:                  end if
177:              160      continue
178:          end if
179:
180: C-----
181: C      CAPTURE (MT = 101 )
182: C-----
183:
184: CCC      MT      = 101
185:
186:          if ( KLIB2(N,MT101,1).ne.0 ) then
187:              L0      = KLIB2(N,MT101,11) - KLIB2(N,MT101,3)
188: *VOCL LOOP,NOVREC
189:              do 170 I = 1, NN
190:                  if ( IEP(I).ge.KLIB2(N,MT101,3)
191:                      & .and.IEP(I).lt.KLIB2(N,MT101,4) ) then
192:                      LL      = L0 + IEP(I)
193:                      SMCW(I,LMIC(5)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
194:                  end if
195:              170      continue

```

```

196:          end if
197: C
198: C-----
199: C      UNRESOLVED RESONANCE (TOTAL , ELASTIC, FISSION )
200: C-----
201: C
202:          NUNR      = KLIB1(N,4)
203:          if ( NUNR.ne.0 ) then
204:              NUNR2      = KLIB1(N,5)
205:              LSUNR      = KLIB1(N,18)
206:              LLSSF      = KLIB1(N,30)
207:
208: C      ..... SELECT PARTICLES IN UNRESOLVED RESONANCE ENERGY RANGE ....
209:
210:          MM      = 0
211:          EL      = CX(LSUNR+NUNR-1)
212: *VOCL LOOP,NOVREC
213:          do 180 I = 1, NN
214:              if ( E(I).le.CX(LSUNR).and.E(I).ge.EL ) then
215:                  MM      = MM + 1
216:                  IBP2(MM) = I
217:                  E2(MM)  = E(I)
218:              end if
219:          180      continue
220: C
221: C      ..... DETERMINE TABLE POSITION
222: C
223:          if ( MM.gt.0 ) then
224:              call BSVDEC( CX(LSUNR), NUNR, E2, IEP2, MM )
225: C
226: C      ..... SELECT TABLE .....
227: C
228:          call RANU2( IRAND, R, MM, ICON )
229:          LINT      = LSUNR + NUNR
230:
231:          do 190 I = 1, MM
232:              EE2      = CX(LSUNR+IEP2(I))
233:              EE1      = CX(LSUNR+IEP2(I)-1)
234:              INTUN     = ICX(LINT+IEP2(I))
235:              if ( INTUN.eq.5 .or. INTUN.eq.3 ) then
236:                  EE2      = LOG(EE2)
237:                  EE1      = LOG(EE1)
238:                  E2(I)    = LOG(E2(I))
239:              end if
240:              if ( R(I).gt.(E2(I)-EE2)/(EE1-EE2) ) IEP2(I) = IEP2(I)
241:                  + 1
242:          190      continue
243: C
244:          call RANU2( IRAND, R, 2*MM, ICON )
245: C
246:          LS      = LSUNR + 2*NUNR
247: CM1995-2-15
248:          NNEG     = 0
249: C
250: C
251: C      ..... CASE OF LLSSF=0 (BACKGROUND IS GIVEN) .....
252: C
253:          if ( LLSSF.eq.0 ) then
254: *VOCL LOOP,NOVREC
255:              do 200 I = 1, MM
256:                  LQ      = LS + (IEP2(I)-1)*6*NUNR2
257:                  LIPOS    = LQ + NUNR2
258: C
259: C      ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
260: C

```

src/mvp/getmic.f

```

261: C/#IF ROUNDOFF(NEAREST)
262:         KK      = MIN(NUNR2-1,INT(NUNR2*R(I)))
263:         K        = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
264: C/#ELSE
265:         *        KK      = NUNR2*R(I)
266:         *        K        = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
267: C/#ENDIF
268:         K        = ICX(LIPOS+K)
269:
270:         LCRX     = LIPOS + 2*NUNR2 + 3*(K-1)
271:
272: C      .... TOTAL .....
273:
274:         SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1))
275:         &      + CX(LCRX)
276:
277: C      .... ELASTIC .....
278:
279:         SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4))
280:         &      + CX(LCRX+1)
281:
282: C      .... FISSION .....
283:
284:         SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3))
285:         &      + CX(LCRX+2)
286:
287: C      .... CAPTURE .....
288:
289:         SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5))
290:         &      + CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
291: C
292: C      .... check non-positive cross section ...
293: C
294:         if ( SMCW(IBP2(I),LMIC(1)).le.0.0
295:         &      .or. SMCW(IBP2(I),LMIC(4)).lt.0.0
296:         &      .or. SMCW(IBP2(I),LMIC(3)).lt.0.0
297:         &      .or. SMCW(IBP2(I),LMIC(5)).lt.0.0 ) then
298:             NNEG = NNEG + 1
299:             CX0  = CX(LCRX)
300:             CX1  = CX(LCRX+1)
301:             CX2  = CX(LCRX+2)
302:             SMCW(IBP2(I),LMIC(1)) = 1.0E-5
303:             SMCW(IBP2(I),LMIC(4)) = 1.0E-5*CX1/CX0
304:             SMCW(IBP2(I),LMIC(3)) = 1.0E-5*CX2/CX0
305:             SMCW(IBP2(I),LMIC(5)) = 1.0E-5*(CX0-CX1
306:             &      -CX2) /CX0
307:         end if
308: 200      continue
309: C
310: C      .... CASE OF LLSSF=1 (AVERAGED IS GIVEN) ....
311: C
312:         else
313: C
314:         *VOCL LOOP,NOVREC
315:         do 210 I = 1, MM
316:             LQ = LS + (IEP2(I)-1)*(6*NUNR2+3)
317:             LIPOS = LQ + NUNR2
318:             LSAV = LQ + 6*NUNR2
319: C
320: C      .... B.M.C. SAMPLING OF PROBABILITY TABLE .....
321: C
322: C/#IF ROUNDOFF(NEAREST)
323:         KK      = MIN(NUNR2-1,INT(NUNR2*R(I)))
324:         K        = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
325: C/#ELSE

```

```

326: *        KK      = NUNR2*R(I)
327: *        K        = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
328: C/#ENDIF
329:         K        = ICX(LIPOS+K)
330:
331:         LCRX     = LIPOS + 2*NUNR2 + 3*(K-1)
332:
333:         SIGC     = CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
334:         SIGCAV   = CX(LSAV) - CX(LSAV+1) - CX(LSAV+2)
335: C
336: C      .... check non-positive capture cross section ...
337: C      if it or averaged one is non-positive,
338: C      infinite dilution cross sections are used.
339: C
340:         if ( SIGC.le.0.0 .or. SIGCAV.le.0.0 ) then
341:             NNEG = NNEG + 1
342:         else
343:
344: C      .... TOTAL .....
345:
346:         SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1))
347:         &      /CX(LSAV)*CX(LCRX)
348:
349: C      .... ELASTIC .....
350:
351:         SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4))
352:         &      /CX(LSAV+1)*CX(LCRX+1)
353:
354: C      .... FISSION .....
355:
356:         if ( CX(LSAV+2).gt.0.0 )
357:         &      SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3))
358:         &      /CX(LSAV+2)*CX(LCRX+2)
359:
360: C      .... CAPTURE .....
361:
362:         SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5))
363:         &      /SIGCAV*SIGC
364:
365:         end if
366: 210      continue
367:         end if
368: C
369: C      .... warning message for non-positive cross section fixup...
370: C
371:         if ( NNEG.gt.0 ) then
372:             if ( NNEG.lt.MWARN ) then
373:                 write(IOW,7000) NNEG, N
374: 7000      format('!!!(GETMIC) FOR',I5,
375:             &      ' particles, cross sections',
376:             &      ' are negative in unresolved resonance region'/
377:             &      ' for ',I4,'-th nuclide.')
378:             MNEG = NNEG + NNEG
379:
380:             if ( MNEG.ge.MWARN ) then
381:                 write(IOW,*)
382:                 '!!!(GETMIC) NEGATIVE X-SEC MESSAGES ',
383:                 &      'WILL BE SUPPRESSED HEREAFTER.'
384:             end if
385:         end if
386:         end if
387:         end if
388:         end if
389: C-----
390: C      NU * FISSION (NU = nu-total)

```

src/mvp/getmic.f

```

391: C-----
392:
393:         LSNU      = KLIB1(N,19)
394: C
395: C ..... COEFFICIENTS .....
396:
397:         if ( LNU.eq.1 ) then
398:             do 220 I = 1, NN
399:                 E2(I) = CX(LSNU+NNU-1)
400: 220         continue
401:             do 240 K = NNU - 2, 0, -1
402:                 do 230 I = 1, NN
403:                     E2(I) = E2(I)*E(I) + CX(LSNU+K)
404: 230             continue
405: 240         continue
406: *VOCL LOOP,NOVREC
407:             do 250 I = 1, NN
408:                 RNU(IBP(I),N,1) = E2(I)
409: 250         continue
410: C
411: C ..... TABULATION .....
412:
413:         else if ( LNU.eq.2 ) then
414:             call BSVDEC( CX(LSNU), NNU, E, IEP2, NN )
415: *VOCL LOOP,NOVREC
416:             do 260 I = 1, NN
417:                 LE = LSNU + IEP2(I)
418:                 LL = LE + NNU
419:                 D0 = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
420:                 RNU(IBP(I),N,1) = CX(LL)+D0*(CX(LL-1)-CX(LL))
421: 260         continue
422:             end if
423: C-----
424: C   Nu-delayed ( MT = 98 )
425: C-----
426: C
427:         LSNU      = KLIB1(N,25)
428: C
429: C ..... COEFFICIENTS .....
430: C
431:         if ( LNUD.eq.1 ) then
432:             do 420 I = 1, NN
433:                 E2(I) = CX(LSNU+NNUD-1)
434: 420         continue
435:             do 440 K = NNUD - 2, 0, -1
436:                 do 430 I = 1, NN
437:                     E2(I) = E2(I)*E(I) + CX(LSNU+K)
438: 430             continue
439: 440         continue
440:             do 450 I = 1, NN
441:                 RNU(IBP(I),N,2) = E2(I)
442:                 RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
443: 450         continue
444: C
445: C ..... TABULATION .....
446: C
447:         else if ( LNUD.eq.2 ) then
448:             call BSVDEC( CX(LSNU), NNUD, E, IEP2, NN )
449: *VOCL LOOP,NOVREC
450:             do 460 I = 1, NN
451:                 LE = LSNU + IEP2(I)
452:                 LL = LE + NNUD
453:                 D0 = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
454:                 RNU(IBP(I),N,2) = CX(LL)+D0*(CX(LL-1)-CX(LL))
455:                 RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)

```

```

456: 460         continue
457:         end if
458: C
459:         if ( LNU.ne.0 ) then
460:             if ( JFMUL.eq.0 .or. LNUD.eq.0 ) then
461:                 do 470 I = 1, NN
462:                     SMCW(I,LMIC(2)) = RNU(IBP(I),N,1)*SMCW(I,LMIC(3))
463: 470             continue
464:             else if ( JFMUL.eq.1 .and. LNUD.ne.0 ) then
465:                 do 480 I = 1, NN
466:                     SMCW(I,LMIC(2)) = RNU(IBP(I),N,3)*SMCW(I,LMIC(3))
467: 480             continue
468:             end if
469:         end if
470: C-----
471: C   (N,2N) (MT = 94 )
472: C-----
473: C
474:         if ( LMIC(7).ne.0 ) then
475: Cccc         MT      = 94
476:             if ( KLIB2(N,MT94,1).ne.0 ) then
477: Ccccc         CALL     CXSINT(NN,IBP, N,MT94,
478: CccccX         SMIC(1,N,LMIC(7)), CX, MCX, KLIB2,NUC,NMT,
479: CccccW         IEP ,DE, DEL, NBANK)
480:                 L0      = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
481: *VOCL LOOP,NOVREC
482:                 do 270 I = 1, NN
483:                     if ( IEP(I).ge.KLIB2(N,MT94,3)
484:                         & .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
485:                         LL      = L0 + IEP(I)
486:                         SMCW(I,LMIC(7)) = CX(LL+1) + DE(I)*
487:                         & (CX(LL)-CX(LL+1))
488:                     end if
489: 270             continue
490:             end if
491:         end if
492: C-----
493: C-----
494: C   LOSS : CAPTURE + FISSION - (N,2N) - 2*(N,3N) - 3*(N,4N)
495: C-----
496: C
497: C   IF(JEIGN.NE.0) THEN
498:
499:         if ( LMIC(8).ne.0 ) then
500: C
501: C ..... (N,2N) HAS BEEN CALCULATED (NSMIC >= 7) ....
502:
503:             if ( LMIC(7).ne.0 ) then
504: *VOCL LOOP,NOVREC
505:                 do 280 I = 1, NN
506:                     R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
507:                     & - SMCW(I,LMIC(7))
508: 280             continue
509:             else
510: C
511: C ..... (N,2N) IS CALCULATED HERE (NSMIC < 7) ....
512:
513: Cccc         MT      = 94
514:
515:             if ( KLIB2(N,MT94,1).ne.0 ) then
516:
517: C         CALL     CXSINT(NN,IBP, N,MT94,
518: C   X         R         , CX, MCX, KLIB2,NUC,NMT,
519: C   W         IEP ,DE, DEL, NBANK)
520:

```

src/mvp/getmic.f

```

521:          L0      = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
522: C
523: *VOCL LOOP,NOVREC
524:       do 290 I = 1, NN
525:         if ( IEP(I).ge.KLIB2(N,MT94,3)
526:           & .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
527:           LL      = L0 + IEP(I)
528:           E2(I)   = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
529:         else
530:           E2(I)   = 0.0
531:         end if
532:
533:         R(I)      = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3)) - E2(I)
534:       290 continue
535:     else
536: *VOCL LOOP,NOVREC
537:       do 300 I = 1, NN
538:         R(I)      = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
539:       300 continue
540:     end if
541:   end if
542: C
543: C-----
544: C   (N,3N) , (N,4N)
545: C-----
546: C
547: CCCC      MT      = 95
548: C      if ( KLIB2(N,MT95,1).ne.0 ) then
549: C
550: Ccccc      CALL   CXSINT(NN,IBP, N,MT95,
551: CccccX      R      , CX, MCX, KLIB2,NUC,NMT,
552: CccccW      IEP ,DE, DE1, NBANK)
553: C
554:       L0      = KLIB2(N,MT95,11) - KLIB2(N,MT95,3)
555: *VOCL LOOP,NOVREC
556:       do 310 I = 1, NN
557:         if ( IEP(I).ge.KLIB2(N,MT95,3)
558:           & .and.IEP(I).lt.KLIB2(N,MT95,4) ) then
559:           LL      = L0 + IEP(I)
560:           E2(I)   = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
561:         else
562:           E2(I)   = 0.0
563:         end if
564:
565:         R(I)      = R(I) - 2.0*E2(I)
566:       310 continue
567:     end if
568: C
569: CCC      MT      = 96
570: C
571: C      if ( KLIB2(N,MT96,1).ne.0 ) then
572: C
573: Ccccc      CALL   CXSINT(NN,IBP, N,MT96,
574: CccccX      E2      , CX, MCX, KLIB2,NUC,NMT,
575: CccccW      IEP ,DE, DE1, NBANK)
576: C
577:       L0      = KLIB2(N,MT96,11) - KLIB2(N,MT96,3)
578: *VOCL LOOP,NOVREC
579:       do 320 I = 1, NN
580:         if ( IEP(I).ge.KLIB2(N,MT96,3)
581:           & .and.IEP(I).lt.KLIB2(N,MT96,4) ) then
582:           LL      = L0 + IEP(I)
583:           E2(I)   = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
584:         else
585:           E2(I)   = 0.0

```

```

586:         end if
587:         R(I)      = R(I) - 3.0*E2(I)
588:       320 continue
589:     end if
590: C
591: C-----
592: C   COMPLETE CALCULATION OF SIGMA-LOSS
593: C-----
594: C
595: *VOCL LOOP,NOVREC
596:       do 330 I = 1, NN
597:         SMCW(I,LMIC(8)) = R(I)
598:       330 continue
599:     end if
600: C
601: C   ... save SMCW in bank array ...
602: C
603:       do 350 K = 1, NSMIC
604:         do 340 I = 1, NN
605:           SMIC(IBP(I),N,K) = SMCW(I,K)
606:         340 continue
607:       350 continue
608: C
609: C 360 continue
610: C
611: C-----
612: C ... point-wise response : SGTAL
613: C   Energy in CRES is from low to high
614: C-----
615: C
616: C   do 390 N = 1, NSTAL
617: C     if ( KCRES(N,4).ne.0 ) then
618: C *VOCL LOOP,NOVREC
619: C       do 370 I = 1, NN
620: C         SGTAL(IBP(I),N) = 0.0
621: C       370 continue
622: C       if ( KCRES(N,3).eq.1 ) then
623: C         NPTS = KCRES(N,2)
624: C         LEMESH = KCRES(N,1)
625: C         LEM = LEMESH + NPTS - 1
626: C         call BSVINC( CRES(LEMESH), NPTS, E, IEP, NN )
627: C
628: C *VOCL LOOP,NOVREC
629: C       do 380 I = 1, NN
630: C         if ( E(I).ge.CRES(LEMESH).and.E(I).le.CRES(LEM) ) then
631: C           LE = LEMESH + IEP(I)
632: C           LL = LE + NPTS
633: C           SGTAL(IBP(I),N) = CRES(LL) + (CRES(LE)-E(I)) /
634: C             & (CRES(LE)-CRES(LE-1))*(CRES(LL-1)-CRES(LL))
635: C         end if
636: C       380 continue
637: C     end if
638: C   end if
639: C 390 continue
640: C
641:       return
642:     end

```


src/mvp/gmacnx.f

```

1:      subroutine GMACNX( IRAND, NN,      IBP,      MAT,      JMIC,      JAMXC,
2:      &                  JNP,      JGAMM,      JDEBG,
3:      &                  NBANK,      EEE,      MB,      NUC,      NSTAL,      NSMAC,
4:      &                  SMAC,      LMIC,      KMAC,      MMAC,      NSMIC,      SMIC,
5:      &                  LMIC,      KSPI,      SGTAL,
6:      &                  LPDEN,      INUCT,      DENST,      NMAT,      NUCL,
7:      &                  NMT,      CX,      MCX,      KLIB1,      KLIB2,      XLIB1,
8:      &                  CRES,      KCRES,      MCRES,      NNCST,      INCST,
9:      &                  IF3,      IFI,      STOT,      WK1,      WK2,      WK3,
10:     &                  WK4,      WK5,      R,      SMCW )
11: C=<MVP>=====
12: C  PURPOSE      : to prepare macroscopic cross sections including micro
13: C                  not select collision nuclide.
14: C  IBP(NN)      : bank pointers
15: C  JMIC         : 0/1 = in free flight / after source or collision
16: C  JAMXC        : 0/1 = calculate micro cross section for all nuclides /
17: C                  calculate micro cross section for nuclides in current zone
18: C  JNP          : 1/2 = neutron/photon
19: C  NUC          : number of nuclides
20: C  NUCL         : number of cross section library
21: C                  this is the number of atoms for photon.
22: C  CALLED IN   : SUNCL, NXTNR, NXTFLI
23: C  CALLS      : GMICNX
24: C -----
25: C  UPDATE :
26: C -----
27:      integer JDEBG(*)
28: C
29: C .... NN : PARTICLE NUMBER      IBP : BANK POINTERS      MAT : MAT #
30: C
31:      integer NN, IBP(NN), MAT
32: C
33: C .... MATERIAL DATA .....
34: C
35:      integer LPDEN(NMAT+1), INUCT(*)
36:      real DENST(*)
37: C
38: C .... Cross section .....
39: C
40:      real CX(MCX), XLIB1(NUCL,*)
41:      integer KLIB1(NUCL,*), KLIB2(NUCL,NMT,*)
42: C
43:      real CRES(MCRES)
44:      integer KCRES(NSTAL,4), INCST(NUC,*)
45: C
46: C .... BANK .....
47: C
48:      real EEE(NBANK)
49: C
50: C .... SIGMA BANK .....
51: C
52:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC)
53:      integer MB, KMAC(NBANK,MB), MMAC(NBANK,2), LMIC(8), LMIC(8)
54:      integer KSPI(NBANK,NUC)
55:      real SGTAL(NBANK,NSTAL)
56: C
57: C .... WORKING ARRAYS ...
58: C
59:      integer IF3(NN), IFI(NN)
60:      real R(2*NN), STOT(NN)
61:      real WK1(NN), WK2(NN), WK3(NN), WK4(NN), WK5(NN)
62: c##<2007/03/14:PN4:
63: c##      real SMCW(NN,NSMIC)
64:      real SMCW(NBANK,NSMIC)
65: c##>

```

```

66: C
67:      include '../shared/INC/_IUNIT'
68:      include 'INC/_IUNIT2'
69: C
70:      if ( NN.le.0 ) return
71: C
72: C=====
73: C  Calculate microscopic cross sections
74: C=====
75: C
76:      if ( JMIC.eq.1 ) then
77: C
78: C ... SET MMAC( ,2)=0 : number of macroscopic cross sections prepared
79: C                  for this particle
80:      do 100 I = 1, NN
81:          MMAC(IBP(I),2) = 0
82:      100 continue
83: C
84:      if ( JAMXC.ne.0 ) then
85:          do 110 N = 1, NUC
86:              do 120 I = 1, NN
87:                  KSPI(IBP(I),N) = 0
88:              120 continue
89:          110 continue
90: C
91:      if ( NNCST.gt.0 ) then
92: C          if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
93: C          if ( JVMNT.ne.0 ) call VMNTR1( 8, NCOLS )
94: C          if ( JNP.eq.1 ) then
95: C              do 130 I = 1, NN
96: C                  WK1(I) = EEE(IBP(I))
97: C              130 continue
98: C          end if
99: C          if ( JNP.eq.2 ) then
100: C              do 140 I = 1, NN
101: C                  WK1(I) = LOG(EEE(IBP(I)))
102: C              140 continue
103: C          end if
104: C          do 150 NC = 1, NNCST
105: C              N = INCST(NC,JNP)
106: C              if ( N.ne.0 ) then
107: C                  call GMICNX
108: C                      ( N, NN, IBP,      WK1,      IRAND, JNP,      JGAMM,
109: C                      NBANK, NSMIC, SMIC,      LMIC,      KSPI,      NUCL,
110: C                      NUC,      NMT,      CX,      MCX,      KLIB1,
111: C                      KLIB2, XLIB1,
112: C                      STOT, WK2,      WK3,      WK4,      WK5,      R, SMCW )
113: C              end if
114: C          150 continue
115: C          if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
116: C          end if
117: C
118:      else
119: C -----
120: C  GATHER ENERGY
121: C -----
122: C
123:      if ( JNP.eq.1 ) then
124: C          do 160 I = 1, NN
125: C              WK1(I) = EEE(IBP(I))
126: C          160 continue
127: C          end if
128: C          if ( JNP.eq.2 ) then
129: C              do 170 I = 1, NN
130: C                  WK1(I) = LOG(EEE(IBP(I)))

```

src/mvp/gmacnx.f

```

131: 170      continue
132:      end if
133: C
134: C      .... NUCL = NUC for neutron, NPATOM for photon
135:      do 180 N = 1, NUCL
136:          call GMICNX( N, NN, IBP, WK1, IRAND, JNP, JGAMM,
137:                      & NBANK, NSMIC, SMIC, LMIC, KSPI, NUCL,
138:                      & NUC, NMT, CX, CX, MCX, KLIB1,
139:                      & KLIB2, XLIB1,
140:                      & STOT, WK2, WK3, WK4, WK5, R,SMCW )
141: 180      continue
142:
143:      end if
144: C
145: C      .. pointwise response function ...
146: C
147:      if ( NSTAL.gt.0 ) then
148:          call GTSCTL( JNP, NN, IBP, NBANK, EEE,
149:                      & SGTAL, CRES, KCRES, NSTAL, MCRES, WK2, WK3 )
150:      end if
151: C
152:      end if
153: C
154: C=====
155: C When MAT.le.0, no macroscopic cross section are calculated.
156: C=====
157: C
158:      if ( MAT.le.0 ) return
159: C
160: C
161: C=====
162: C DETERMINE BANK NUMBER TO BE USED ( MMAC(IBP(I),1) )
163: C=====
164: C
165: C
166:      N3 = 0
167:      do 200 I = 1, NN
168:          IF3(I) = 0
169: 200      continue
170: C-----
171: C IF3 : STATUS INDICATOR 0 = TREATMENT UNDETERMINED YET
172: C 1 = MACRO DATA MUST BE CALCULATED NEWLY
173: C -1 = MACRO DATA ALREADY PREPARED
174: C-----
175: C
176: C COUNT THE NUMBER OF PARTICLES WHOSE DATA MUST BE CALCULATED NEWLY.
177: C IFI : GATHERED BANK POINTERS
178: C-----
179:      N2 = 0
180: C
181:      do 220 K = 1, MB + 1
182:          if ( N3.eq.NN ) go to 230
183: *VOCL LOOP,NOVREC
184:          do 210 I = 1, NN
185:              if ( IF3(I).eq.0 ) then
186: C-----
187: C MACRO DATA ARE NOT IN BANK, MUST BE CALCULATED NEWLY!
188: C-----
189:              if ( MMAC(IBP(I),2).lt.K ) then
190:                  IF3(I) = 1
191:                  KMAC0 = MIN(MMAC(IBP(I),2)+1,MB)
192:                  MMAC(IBP(I),2) = KMAC0
193:                  MMAC(IBP(I),1) = KMAC0
194:                  KMAC(IBP(I),KMAC0) = MAT
195:                  N2 = N2 + 1

```

```

196:          IFI(N2) = IBP(I)
197: C-----
198: C MACRO DATA FOR MATERIAL 'MAT' MAY EXIST IN BANK
199: C-----
200:          else
201:              if ( KMAC(IBP(I),K).eq.MAT ) then
202:                  IF3(I) = -1
203:                  MMAC(IBP(I),1) = K
204:              end if
205:          end if
206: C
207:          if ( IF3(I).ne.0 ) N3 = N3 + 1
208:      end if
209: 210      continue
210: 220      continue
211: C-----
212: 230      continue
213: C-----
214: C N2 = 0 : NO NEED TO CALCULATE MACRO DATA !
215: C-----
216:      if ( N2.eq.0 ) return
217: C
218: C=====
219: C CALCULATE MACRO DATA FOR SOME PARTICLES
220: C=====
221: C
222:      if ( JAMXC.ne.0 ) then
223:          do 260 N = LPDEN(MAT), LPDEN(MAT+1) - 1
224:              NI = INUCT(N)
225:              NNN = 0
226:              if ( JNP.eq.1 ) then
227:                  do 240 I = 1, N2
228:                      if ( KSPI(IFI(I),NI).eq.0 ) then
229:                          NNN = NNN + 1
230:                          IF3(NNN) = IFI(I)
231:                          WK1(NNN) = EEE(IFI(I))
232:                      end if
233: 240                  continue
234:              end if
235:              if ( JNP.eq.2 ) then
236:                  do 250 I = 1, N2
237:                      if ( KSPI(IFI(I),NI).eq.0 ) then
238:                          NNN = NNN + 1
239:                          IF3(NNN) = IFI(I)
240:                          WK1(NNN) = LOG(EEE(IFI(I)))
241:                      end if
242: 250                  continue
243:              end if
244:              if ( NNN.gt.0 ) then
245:                  call GMICNX( NI, NNN, IF3, WK1, IRAND, JNP, JGAMM,
246:                              & NBANK, NSMIC, SMIC, LMIC, KSPI, NUCL, NUC,
247:                              & NMT, CX, CX, MCX, KLIB1, KLIB2, XLIB1,
248:                              & STOT, WK2, WK3, WK4, WK5, R,SMCW )
249:              end if
250: 260          continue
251:      end if
252: C-----
253: C CLEAR OR INITIALIZE TEMPORARY ARRAYS
254: C-----
255:      NI = INUCT(LPDEN(MAT))
256:      RHO = DENST(LPDEN(MAT))
257:      do 270 I = 1, N2
258:          STOT(I) = SMIC(IFI(I),NI,LMIC(1))*RHO
259: 270      continue
260: C-----

```

src/mvp/gmacnx.f

```

261: C    CALCULATE MACRO CROSS SECTION BY SUMMATION OF MICRO CROSS SECTIONS
262: C-----
263:       if ( (LPDEN(MAT+1)-LPDEN(MAT)).ge.2 ) then
264: C
265:         do 290 N = LPDEN(MAT) + 1, LPDEN(MAT+1) - 1
266:           NI      = INUCT(N)
267:           RHO     = DENST(N)
268:           do 280 I = 1, N2
269:             STOT(I) = STOT(I) + SMIC(IFI(I),NI,LMIC(1))*RHO
270:           280 continue
271:         290 continue
272:       end if
273: C-----
274: C    PUT CALCULATED MACROSCOPIC DATA IN BANK
275: C-----
276: *VOCL LOOP,NOVREC
277:       do 300 I = 1, N2
278:         SMAC(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOT(I)
279:       300 continue
280: C
281: C-----
282: C    OTHER MACROSCOPIC CROSS SECTIONS IF NECESSARY
283: C-----
284:       do 350 K = 2, 8
285:         if ( LMAC(K).gt.0 ) then
286:           do 310 I = 1, N2
287:             STOT(I) = 0.0
288:           310 continue
289:           do 330 N = LPDEN(MAT), LPDEN(MAT+1) - 1
290:             NI      = INUCT(N)
291:             RHO     = DENST(N)
292:             do 320 I = 1, N2
293:               STOT(I) = STOT(I) + SMIC(IFI(I),NI,LMIC(K))*RHO
294:             320 continue
295:           330 continue
296: *VOCL LOOP,NOVREC
297:           do 340 I = 1, N2
298:             SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOT(I)
299:           340 continue
300:         end if
301:       350 continue
302: C-----
303: C    DATA CHECK  ONLY FOR DEBUGGING  (1990/3/15)
304: C-----
305:       if ( JDEBG(1).ne.0 ) then
306:         do 550 I = 1, N2
307:           if ( MMAC(IFI(I),1).lt.1
308:             & .or. MMAC(IFI(I),1).gt.MMAC(IFI(I),2)
309:             & .or. MMAC(IFI(I),2).lt.0 .or. MMAC(IFI(I),2).gt.MB ) then
310:             write(IOG,*) ' == INVALID MMAC DATA IN GETMAC ! == '
311:             write(IOG,*) '   POS = ', IFI(I), ' MMAC = ',
312:             & MMAC(IFI(I),1), MMAC(IFI(I),2)
313:           end if
314:         550 continue
315:         do 570 M = 1, MB
316:           do 560 I = 1, N2
317:             if ( KMAC(IFI(I),M).lt.0 .or. KMAC(IFI(I),M).gt.NMAT )
318:             & then
319:             write(IOG,*) ' == INVALID KMAC DATA IN GMACNX ! == '
320:             write(IOG,*) '   POS = ', IFI(I), ' M= ', M,
321:             & ' KMAC = ', KMAC(IFI(I),M)
322:           end if
323:         560 continue
324:       570 continue
325:       end if
326: C
327:       return
328: end

```

src/mvp/gmacpt.f

```

1:      subroutine GMACPT( IRAND, NN,      IBP,      MAT,      JEIGN, JDEBG,
2:      &                  NBANK, MB,      NUC,      NSTAL, NSMAC, SMAC,  LMAC,
3:      &                  KMAC, MMAC, NSMIC, SMIC,  LMIC,  SGTAL,
4:      &                  LPDEN, INUCT, DENST, NMAT,  IF3,   IFI,   R,
5:      &                  STOT, SNUF, SLOS,
6:      &                  SMICP, SMACP, STOTP, SNUFP, SLOSP )
7: C=<MVP>=====
8: C  PURPOSE      :  TO GET OR PREPARE MACRO CROSS SECTION
9: C  CALLED IN   :  FLIONE, FLIGHT, VMNTR1
10: C  CALLS       :  NONE
11: C-----
12: C <comment> Calculations specifying different MB values may not produce
13: C exactly the same result because KMAC(*,*,2) - colliding nuclide -
14: C is determined in the course of sigma-total calculation using random
15: C numbers and sigma-total may be re-calculated even in a particle
16: C flight path when MB is less than NMAT. (July 1997 M.S.)
17: C-----
18: C  UPDATE      :
19: C  7 SEP 1992 : I/O UNIT TI PARAMETER
20: C  31 May 1993 : add dummy argument 'IRAND' (random number seed)
21: C  27 Apr 1995: give "CONTINUE" for all "DO"
22: C  27 Jul 1997: declare JDEBG as an array.
23: C=====
24:      integer JDEBG(*)
25: C
26: C .... NN : PARTICLE NUMBER      IBP : BANK POINTERS      MAT: MAT #
27: C
28:      integer NN, IBP(NBANK), MAT
29: C
30: C .... MATERIAL DATA .....
31: C
32:      integer LPDEN(NMAT+1), INUCT(*)
33:      real DENST(*)
34: C
35: C .... SIGMA BANK .....
36: C
37:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
38:      &      SGTAL(NBANK,NSTAL)
39:      integer MB, KMAC(NBANK,MB,2), MMAC(NBANK,2), LMAC(8), LMIC(8)
40:      real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
41: C
42: C .... WORKING ARRAYS ...
43: C
44:      integer IF3(NBANK), IFI(NBANK)
45:      real R(NBANK), STOT(NBANK), SNUF(NBANK), SLOS(NBANK)
46:      real STOTP(NBANK), SNUFP(NBANK), SLOSP(NBANK)
47: C
48:      include '../shared/INC/_IOUNIT'
49:      include 'INC/_IOUNIT2'
50: C
51: C
52: C=====
53: C  DETERMINE BANK NUMBER TO BE USED  ( MMAC(IBP(I),1) )
54: C=====
55: C
56: C
57: C+f.k 1999.2.25
58: C  write(*,*) 'SMAC(2,1,0):',SMAC(2,1,0),' SMACP():',SMACP(2,1,0)
59: C  write(*,*) 'SMAC(2,1,1):',SMAC(2,1,1),' SMACP():',SMACP(2,1,1)
60: C  write(*,*) '----- GMACPT -----'
61: C  write(*,*) 'SMIC(2,1,1):',SMIC(2,1,1),' SMICP():',SMICP(2,1,1)
62: C  write(*,*) 'NBANK:',nbank,' MB:',mb,' NSMAC:',nsmac
63: C  do i = 1, NBANK
64: C    do j = 1, MB
65: C      do k = 1, NSMAC

```

```

66: C      smacp(i,j,k) = 0.0
67: C      enddo
68: C      enddo
69: C      enddo
70: C-f.k 1999.2.25
71: C      N3      = 0
72: C      do 100 I = 1, NN
73: C        IF3(I) = 0
74: C      100 continue
75: C-----
76: C  IF3 : STATUS INDICATOR      0 = TREATMENT UNDETERMINED YET
77: C      1 = MACRO DATA MUST BE CALCULATED NEWLY
78: C     -1 = MACRO DATA ALREADY PREPARED
79: C-----
80: C-----
81: C  COUNT THE NUMBER OF PARTICLES WHOSE DATA MUST BE CALCULATED NEWLY.
82: C  IFI : GATHERED BANK POINTERS
83: C-----
84: C      N2      = 0
85: C
86: C      do 120 K = 1, MB + 1
87: C        if ( N3.eq.NN ) go to 130
88: C      *VOCL LOOP,NOVREC
89: C      do 110 I = 1, NN
90: C        if ( IF3(I).eq.0 ) then
91: C-----
92: C      MACRO DATA ARE NOT IN BANK, MUST BE CALCULATED NEWLY!
93: C-----
94: C        if ( MMAC(IBP(I),2).lt.K ) then
95: C          IF3(I) = 1
96: C          KMAC0 = MIN(MMAC(IBP(I),2)+1,MB)
97: C          MMAC(IBP(I),2) = KMAC0
98: C          MMAC(IBP(I),1) = KMAC0
99: C          KMAC(IBP(I),KMAC0,2) = MAT
100: C          N2      = N2 + 1
101: C          IFI(N2) = IBP(I)
102: C-----
103: C      MACRO DATA FOR MATERIAL 'MAT' MAY EXIST IN BANK
104: C-----
105: C        else
106: C          if ( KMAC(IBP(I),K,2).eq.MAT ) then
107: C            IF3(I) = -1
108: C            MMAC(IBP(I),1) = K
109: C          end if
110: C        end if
111: C
112: C        if ( IF3(I).ne.0 ) N3      = N3 + 1
113: C      end if
114: C    110 continue
115: C  120 continue
116: C-----
117: C  130 continue
118: C-----
119: C  N2 = 0 : NO NEED TO CALCULATE MACRO DATA !
120: C-----
121: C+f.k 1999.2.25
122: C  write(*,*) 'N2:',N2
123: C-f.k 1999.2.25
124: C  if ( N2.eq.0 ) return
125: C
126: C=====
127: C  CALCULATE MACRO DATA FOR SOME PARTICLES
128: C=====
129: C
130: C-----

```

src/mvp/gmacpt.f

```

131: C      CLEAR OR INITIALIZE TEMPORARY ARRAYS
132: C      (IF3 IS USED AS TEMPORARY STORAGE OF COLLIDING NUCLIDE NUMBER)
133: C-----
134:      NI      = INUCT(LP DEN(MAT))
135:      RHO      = DENST(LP DEN(MAT))
136: c+f.k 1999.5.12
137: c      write(*,*) '--- summary --- gmacpt',NI,RHO,MAT,LP DEN(MAT)
138: c      write(*,*) '--- summary ---', (DENST(JJJ),JJJ=1,9)
139: c-f.k 1999.5.12
140:      do 140 I = 1, N2
141:      IF3(I) = NI
142: C      STOT(I) = 0.0
143:      STOT(I) = SMIC(IFI(I),NI,LMIC(1))*RHO
144:      STOTP(I) = SMICP(IFI(I),NI,LMIC(1))*RHO
145:      if ( JEIGN.ne.0 ) then
146: C      SNUF(I) = 0.0
147: C      SLOS(I) = 0.0
148:      SNUF(I) = SMIC(IFI(I),NI,LMIC(2))*RHO
149:      SLOS(I) = SMIC(IFI(I),NI,LMIC(8))*RHO
150:      SNUFP(I) = SMICP(IFI(I),NI,LMIC(2))*RHO
151:      SLOSP(I) = SMICP(IFI(I),NI,LMIC(8))*RHO
152:      end if
153:      140 continue
154: C-----
155: C      CALCULATE MACRO CROSS SECTION BY SUMMATION OF MICRO CROSS SECTIONS
156: C-----
157:      if ( (LP DEN(MAT+1)-LP DEN(MAT)).ge.2 ) then
158: C
159:      do 160 N = LP DEN(MAT) + 1, LP DEN(MAT+1) - 1
160:      call RANU2( IRAND, R, N2, ICON )
161:      NI      = INUCT(N)
162:      RHO      = DENST(N)
163:      do 150 I = 1, N2
164:      ST      = SMIC(IFI(I),NI,LMIC(1))*RHO
165:      STP      = SMICP(IFI(I),NI,LMIC(1))*RHO
166: C      ..... IF3 : NUCLIDE TO COLLIDE .....
167: C
168:      STOT(I) = STOT(I) + ST
169:      STOTP(I) = STOTP(I) + STP
170:      if ( ST.gt.STOT(I)*R(I) ) IF3(I) = NI
171:      if ( JEIGN.ne.0 ) then
172:      SNUF(I) = SNUF(I) + SMIC(IFI(I),NI,LMIC(2))*RHO
173:      SLOS(I) = SLOS(I) + SMIC(IFI(I),NI,LMIC(8))*RHO
174:      SNUFP(I) = SNUFP(I) + SMICP(IFI(I),NI,LMIC(2))*RHO
175:      SLOSP(I) = SLOSP(I) + SMICP(IFI(I),NI,LMIC(8))*RHO
176:      end if
177:      150 continue
178:      160 continue
179:      end if
180: C-----
181: C      PUT CALCULATED MACROSCOPIC DATA IN BANK
182: C-----
183: *VOCL LOOP,NOVREC
184:      do 170 I = 1, N2
185:      KMAC(IFI(I),MMAC(IFI(I),1),1) = IF3(I)
186:      SMAC(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOT(I)
187:      SMACP(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOTP(I)
188:      if ( JEIGN.ne.0 ) then
189:      SMAC(IFI(I),MMAC(IFI(I),1),LMAC(2)) = SNUF(I)
190:      SMAC(IFI(I),MMAC(IFI(I),1),LMAC(8)) = SLOS(I)
191:      SMACP(IFI(I),MMAC(IFI(I),1),LMAC(2)) = SNUFP(I)
192:      SMACP(IFI(I),MMAC(IFI(I),1),LMAC(8)) = SLOSP(I)
193:      end if
194:      170 continue
195: C

```

```

196: C-----
197: C      OTHER MACROSCOPIC CROSS SECTIONS IF NECESSARY
198: C-----
199:      do 220 K = 2, 8
200:      if ( JEIGN.eq.0 .or. (K.ne.2.and.K.ne.8) ) then
201: c+f.k 1999.3.31
202: c      write(*,*) '==== SMAC : ',SMAC(2,1,0),' STOT: ',STOT(1)
203: c      write(*,*) '==== SMACP: ',SMACP(2,1,0),' STOTP: ',STOTP(1)
204: c      do 111 I = 1, N2
205: c      STOT(I) = 0.0
206: c      STOTP(I) = 0.0
207: c      SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOT(I)
208: c      SMACP(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOTP(I)
209: c      write(*,*) SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K))
210: c      write(*,*) SMACP(IFI(I),MMAC(IFI(I),1),LMAC(K))
211: c 111 continue
212: c-f.k 1999.3.31
213:      if ( LMAC(K).gt.0 ) then
214:      do 180 I = 1, N2
215:      STOT(I) = 0.0
216:      STOTP(I) = 0.0
217:      180 continue
218:      do 200 N = LP DEN(MAT), LP DEN(MAT+1) - 1
219:      NI      = INUCT(N)
220:      RHO      = DENST(N)
221:      do 190 I = 1, N2
222:      STOT(I) = STOT(I) + SMIC(IFI(I),NI,LMIC(K))*RHO
223:      STOTP(I) = STOTP(I) + SMICP(IFI(I),NI,LMIC(K))*RHO
224:      190 continue
225:      200 continue
226: *VOCL LOOP,NOVREC
227:      do 210 I = 1, N2
228:      SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOT(I)
229:      SMACP(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOTP(I)
230:      210 continue
231:      end if
232:      end if
233:      220 continue
234: c+f.k 1999.2.25
235: c      write(*,*) 'SMAC(2,1,0):',SMAC(2,1,0),' SMACP():',SMACP(2,1,0)
236: c      write(*,*) 'SMAC(2,1,1):',SMAC(2,1,1),' SMACP():',SMACP(2,1,1)
237: c-f.k 1999.2.25
238: C-----
239: C      DATA CHECK ONLY FOR DEBUGGING (1990/3/15)
240: C-----
241:      if ( JDEBG(1).ne.0 ) then
242:      do 230 I = 1, N2
243:      if ( MMAC(IFI(I),1).lt.1
244:      & .or. MMAC(IFI(I),1).gt.MMAC(IFI(I),2)
245:      & .or. MMAC(IFI(I),2).lt.0 .or. MMAC(IFI(I),2).gt.MB ) then
246: c      write(IOG,*) ' == INVALID MMAC DATA IN GETMAC ! == '
247: c      write(IOG,*) ' POS = ', IFI(I), ' MMAC = ',
248: c      & MMAC(IFI(I),1), MMAC(IFI(I),2)
249:      end if
250:      230 continue
251:      do 250 M = 1, MB
252:      do 240 I = 1, N2
253:      if ( KMAC(IFI(I),M,1).lt.0 .or. KMAC(IFI(I),M,1).gt.NUC
254:      & .or. KMAC(IFI(I),M,2).lt.0
255:      & .or. KMAC(IFI(I),M,2).gt.NMAT ) then
256: c      write(IOG,*) ' == INVALID KMAC DATA IN GETMAC ! == '
257: c      write(IOG,*) ' POS = ', IFI(I), ' M = ', M,
258: c      & KMAC = ', KMAC(IFI(I),M,1),
259: c      & KMAC(IFI(I),M,2)
260:      end if

```

src/mvp/gmacpt.f

```
261:      240      continue
262:      250      continue
263:      end if
264: C
265: C-----
266: C check write
267: C-----
268: C
269: c      write(*,*) 'LMAC: ',(LMAC(i),i=1,8)
270: c      write(*,*) 'N2=',N2,' NSMAC=',NSMAC
271: c      write(*,*) 'IFI: ',(IFI(i),i=1,N2)
272: c      do k= 1,8
273: c          do i = 1,N2
274: c              if( SMAC(IFI(i),MMAC(IFI(i),1),LMAC(k)) .ne.
275: c &              SMACP(IFI(i),MMAC(IFI(1),1),LMAC(k)) .and.
276: c &              LMAC(K).gt.0 ) then
277: c                  write(*,*) ' == INVALID DATA IN SMAC =='
278: c                  write(*,*) ' Pos= ',IFI(i),' MMAC=',MMAC(IFI(i),1),
279: c &                  ' K= ',k,
280: c &                  ' SMAC :',SMAC(IFI(i),MMAC(IFI(i),1),LMAC(k)),
281: c &                  ' SMACP:',SMACP(IFI(i),MMAC(IFI(i),1),LMAC(k))
282: c              endif
283: c          enddo
284: c      enddo
285:
286:      return
287:      end
```

src/mvp/gmicn1.f

```

1:      subroutine GMICN1( N, NN, IBP,E, IRAND, NUC,   NMT,
2:      &                  NBANK, NSMIC, SMIC, LMIC, KSPI, CX,
3:      &                  ICX,  MCX,   KLIB1, KLIB2, XLIB1,
4:      &                  E2,   IEP,   IEP2,
5:      &                  IBP2, DE,    R,      SMCW,
6:      &                  RNU,   NUC_MAX )
7: C=<MVP>=====
8: C PURPOSE: Get micro cross sections and put them in sigma-bank.
9: C           for N'th nuclide
10: C          (NEUTRON)
11: C CALLED IN:  NEUTR, SOURCE
12: C=====
13: C
14: C .... NN: NUMBER OF PARTICLES  IBP: POINTER .....
15: C
16: C      integer IBP(NN)
17: C
18: C .... energy of neutrons
19: C
20: C      real E(NBANK)
21: C
22: C .... BANK .....
23: C
24: C      real SMIC(NBANK,NUC,NSMIC)
25: C      integer KSPI(NBANK,NUC), LMIC(8)
26: C
27: C .... CROSS SECTION & POINTERS .....
28: C
29: C      real CX(MCX), XLIB1(NUC,11)
30: C##<2007/03/14:PN3:
31: C##      integer ICX(MCX), KLIB1(NUC,30), KLIB2(NUC,NMT,13)
32: C      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
33: C##>
34: C      RNU(NBANK,NUC_MAX,1) : nu-total
35: C      RNU(NBANK,NUC_MAX,2) : nu-delayed
36: C      RNU(NBANK,NUC_MAX,3) : nu-prompt
37: C      real RNU(NBANK,NUC_MAX,3)
38: C
39: C .... WORKING ARRAY .....
40: C
41: C      real E2(NBANK), R(2*NBANK), DE(NBANK)
42: C      integer IEP(NBANK), IEP2(NBANK), IBP2(NBANK)
43: C      real SMCW(NBANK,NSMIC)
44: C
45: C      integer MNEG
46: C      parameter( MWARN      = 20 )
47: C      parameter( MWARN      = 2000 )
48: C
49: C      parameter( MT1 = 1, MT2 = 2, MT4 = 4, MT18 = 18, MT101 =
50: C      &          101, MT94 = 94, MT95 = 95, MT96 = 96 )
51: C
52: C      include 'INC/_FLAGS'
53: C      include '../shared/INC/_IUNIT'
54: C
55: C      data MNEG /0/
56: C
57: C-----
58: C      do 100 I = 1, NN
59: C          E(I) = EEE(IBP(I))
60: C 100 continue
61: C
62: C      NPTS = KLIB1(N,2)
63: C      LNU = KLIB1(N,14)
64: C      NNU = KLIB1(N,9)
65: C      N2 = LNU*NNU

```

```

66:
67: C-----
68: C      ENERGY MESH POINT #
69: C-----
70:
71: C      LEMESH = KLIB1(N,1)
72: C      call BSVDEC( CX(LEMESH), NPTS, E, IEP, NN )
73:
74: C*VOCL LOOP,NOVREC
75: C      do 100 I = 1, NN
76: C          KSPI(IBP(I),N) = IEP(I)
77: C          LE = LEMESH + IEP(I)
78: C          DE(I) = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
79: C 100 continue
80: C
81: C      ..... CLEAR ALL CROSS SECTIONS .....
82: C
83: C      do 120 K = 1, NSMIC
84: C          do 110 I = 1, NN
85: C              SMCW(I,K) = 0.0
86: C 110 continue
87: C 120 continue
88: C      do 122 K = 1, 3
89: C          do 121 I = 1, NN
90: C              RNU(IBP(I),N,K) = 0.0
91: C 121 continue
92: C 122 continue
93:
94: C-----
95: C      TOTAL      (MT = 1) & ELASTIC (MT = 2)
96: C-----
97:
98: C      L1 = KLIB2(N,MT1,11) - KLIB2(N,MT1,3)
99: C      L2 = KLIB2(N,MT2,11) - KLIB2(N,MT2,3)
100: C*VOCL LOOP,NOVREC
101: C      do 130 I = 1, NN
102: C          LL = L1 + IEP(I)
103: C          SMCW(I,LMIC(1)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
104: C          if ( KLIB1(N,15).ne.1 .or. E(I).gt.XLIB1(N,4) ) then
105: C              LL2 = L2 + IEP(I)
106: C              SMCW(I,LMIC(4)) = CX(LL2+1) + DE(I)*(CX(LL2)-CX(LL2+1))
107: C          end if
108: C
109: C 130 continue
110:
111: C-----
112: C      INELASTIC (MT = 4)
113: C-----
114:
115: C      if ( LMIC(6).ne.0 ) then
116: C          if ( KLIB2(N,MT4,1).ne.0 ) then
117: C              L0 = KLIB2(N,MT4,11) - KLIB2(N,MT4,3)
118: C*VOCL LOOP,NOVREC
119: C          do 140 I = 1, NN
120: C              if ( IEP(I).ge.KLIB2(N,MT4,3)
121: C      &          .and.IEP(I).lt.KLIB2(N,MT4,4) ) then
122: C                  LL = L0 + IEP(I)
123: C                  SMCW(I,LMIC(6)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
124: C              end if
125: C 140 continue
126: C          end if
127: C      end if
128: C
129: C-----
130: C      FISSION (MT = 18)

```

src/mvp/gmicn1.f

```

131: C-----
132: C
133:       if ( KLIB2(N,MT18,1).ne.0 ) then
134:         L0      = KLIB2(N,MT18,11) - KLIB2(N,MT18,3)
135: *VOCL LOOP,NOVREC
136:       do 150 I = 1, NN
137:         if ( IEP(I).ge.KLIB2(N,MT18,3).and.IEP(I).lt.KLIB2(N,MT18,4)
138:           &      ) then
139:           LL      = L0 + IEP(I)
140:           SMCW(I,LMIC(3)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
141:         end if
142:       150 continue
143:     end if
144:
145: C-----
146: C  CAPTURE  (MT = 101 )
147: C-----
148:
149:       if ( KLIB2(N,MT101,1).ne.0 ) then
150:         L0      = KLIB2(N,MT101,11) - KLIB2(N,MT101,3)
151: *VOCL LOOP,NOVREC
152:       do 160 I = 1, NN
153:         if ( IEP(I).ge.KLIB2(N,MT101,3)
154:           .and.IEP(I).lt.KLIB2(N,MT101,4) ) then
155:           LL      = L0 + IEP(I)
156:           SMCW(I,LMIC(5)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
157:         end if
158:       160 continue
159:     end if
160:
161: C-----
162: C  UNRESOLVED RESONANCE  (TOTAL , ELASTIC, FISSION )
163: C-----
164: C
165:       NUNR      = KLIB1(N,4)
166:       if ( NUNR.ne.0 ) then
167:         NUNR2    = KLIB1(N,5)
168:         LSUNR    = KLIB1(N,18)
169:         LLSSF    = KLIB1(N,30)
170:
171: C  .... SELECT PARTICLES IN UNRESOLVED RESONANCE ENERGY RANGE ...
172:
173:       MM      = 0
174:       EL      = CX(LSUNR+NUNR-1)
175: *VOCL LOOP,NOVREC
176:       do 170 I = 1, NN
177:         if ( E(I).le.CX(LSUNR).and.E(I).ge.EL ) then
178:           MM      = MM + 1
179:           IBP2(MM) = I
180:           E2(MM)  = E(I)
181:         end if
182:       170 continue
183: C
184: C  .... DETERMINE TABLE POSITION
185: C
186:       if ( MM.gt.0 ) then
187:         call BSVDEC( CX(LSUNR), NUNR, E2, IEP2, MM )
188: C
189: C  .... SELECT TABLE .....
190:
191:       call RANU2( IRAND, R, MM, ICON )
192:       LINT      = LSUNR + NUNR
193:
194:       do 180 I = 1, MM
195:         EE2      = CX(LSUNR+IEP2(I))

```

```

196:         EE1      = CX(LSUNR+IEP2(I)-1)
197:         INTUN     = ICX(LINT+IEP2(I))
198:         if ( INTUN.eq.5 .or. INTUN.eq.3 ) then
199:           EE2      = LOG(EE2)
200:           EE1      = LOG(EE1)
201:           E2(I)    = LOG(E2(I))
202:         end if
203:         if ( R(I).gt.(E2(I)-EE2)/(EE1-EE2) ) IEP2(I) = IEP2(I) +
204:           &      1
205:       180 continue
206: C
207:       call RANU2( IRAND, R, 2*MM, ICON )
208: C
209:       LS      = LSUNR + 2*NUNR
210:       NNEG     = 0
211: C
212: C  .... CASE OF LLSSF=0 (BACKGROUND IS GIVEN) ....
213: C
214:       if ( LLSSF.eq.0 ) then
215: *VOCL LOOP,NOVREC
216:       do 190 I = 1, MM
217:         LQ      = LS + (IEP2(I)-1)*6*NUNR2
218:         LIPOS    = LQ + NUNR2
219: C
220: C  .... B.M.C. SAMPLING OF PROBABILITY TABLE .....
221: C
222: C/#IF ROUNDOff(NEAREST)
223:       KK      = MIN(NUNR2-1,INT(NUNR2*R(I)))
224:       K        = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
225: C/#ELSE
226:       KK      = NUNR2*R(I)
227:       K        = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
228: C/#ENDIF
229:       K        = ICX(LIPOS+K)
230:
231: C       LCRX    = LIPOS + 2*NUNR2 - 1
232:
233:       LCRX    = LIPOS + 2*NUNR2 + 3*(K-1)
234:
235: C  .... TOTAL .....
236:
237:       SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1))
238:       &      + CX(LCRX)
239: C
240: C  .... ELASTIC .....
241:
242:       SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4))
243:       &      + CX(LCRX+1)
244: C
245: C  .... FISSION .....
246:
247:       SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3))
248:       &      + CX(LCRX+2)
249: C
250: C  .... CAPTURE .....
251:
252:       SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5))
253:       &      + CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
254: C
255: C  .... check non-positive cross section ...
256: C
257:       if ( SMCW(IBP2(I),LMIC(1)).le.0.0
258:         .or. SMCW(IBP2(I),LMIC(4)).lt.0.0
259:         .or. SMCW(IBP2(I),LMIC(3)).lt.0.0
260:         .or. SMCW(IBP2(I),LMIC(5)).lt.0.0 ) then

```


src/mvp/gmicn1.f

```

261:      NNEG      = NNEG + 1
262:      CX0       = CX(LCRX)
263:      CX1       = CX(LCRX+1)
264:      CX2       = CX(LCRX+2)
265:      SMCW(IBP2(I),LMIC(1)) = 1.0E-5
266:      SMCW(IBP2(I),LMIC(4)) = 1.0E-5*CX1/CX0
267:      SMCW(IBP2(I),LMIC(3)) = 1.0E-5*CX2/CX0
268:      SMCW(IBP2(I),LMIC(5)) = 1.0E-5*(CX0-CX1-CX2) /CX0
269:      end if
270: 190      continue
271: C
272: C      .... CASE OF LLSSF=1 (AVERAGED IS GIVEN) ....
273: C
274: C      else
275: C
276: *VOCL LOOP,NOVREC
277: do 200 I = 1, MM
278:   LQ = LS + (IEP2(I)-1)*(6*NUNR2+3)
279:   LIPOS = LQ + NUNR2
280:   LSAV = LQ + 6*NUNR2
281: C
282: C      ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
283: C
284: C/#IF ROUNDOFF(NEAREST)
285:      KK      = MIN(NUNR2-1,INT(NUNR2*R(I)))
286:      K       = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
287: C/#ELSE
288:      KK      = NUNR2*R(I)
289:      K       = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
290: C/#ENDIF
291:      K       = ICX(LIPOS+K)
292:
293:      LCRX    = LIPOS + 2*NUNR2 + 3*(K-1)
294:
295:      SIGC    = CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
296:      SIGCAV  = CX(LSAV) - CX(LSAV+1) - CX(LSAV+2)
297: C
298: C      .... check non-positive capture cross section ...
299: C      if it or averaged one is non-positive,
300: C      infinite dilution cross sections are used.
301: C
302:      if ( SIGC.le.0.0 .or. SIGCAV.le.0.0 ) then
303:         NNEG = NNEG + 1
304:      else
305:
306: C      ..... TOTAL .....
307:
308:         SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1)) /
309: &         CX(LSAV)*CX(LCRX)
310:
311: C      ..... ELASTIC .....
312:
313:         SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4)) /
314: &         CX(LSAV+1)*CX(LCRX+1)
315:
316: C      ..... FISSION .....
317:
318:         if ( CX(LSAV+2).gt.0.0 )
319: &         SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3)) /
320: &         CX(LSAV+2)*CX(LCRX+2)
321:
322: C      ..... CAPTURE .....
323:
324:         SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5)) /
325: &         SIGCAV*SIGC

```

```

326:
327:      end if
328: 200      continue
329:      end if
330: C
331: C      .... warning message for non-positive cross section fixup...
332: C
333:      if ( NNEG.gt.0 ) then
334:         if ( MNEG.lt.MWARN ) then
335:            write(IOW,7000) NNEG, N
336: 7000      format('!!! (GMICN1) FOR',I5,
337: &            ' particles, cross sections',
338: &            ' are negative in unresolved resonance region'
339: &            /' for ',I4,'-th nuclide.')
340:            MNEG = MNEG + NNEG
341:
342:         if ( MNEG.ge.MWARN ) then
343:            write(IOW,*)
344: &            '!!! (GMICN1) NEGATIVE X-SEC MESSAGES ',
345: &            'WILL BE SUPPRESSED HEREAFTER.'
346:         end if
347:      end if
348:      end if
349:      end if
350:      end if
351: C-----
352: C      NU * FISSION
353: C-----
354:
355:      LSNU    = KLIB1(N,19)
356: C
357: C      ..... COEFFICIENTS .....
358:
359:      if ( LNU.eq.1 ) then
360:         do 210 I = 1, NN
361:            E2(I) = CX(LSNU+NNU-1)
362: 210      continue
363:         do 230 K = NNU - 2, 0, -1
364:            do 220 I = 1, NN
365:               E2(I) = E2(I)*E(I) + CX(LSNU+K)
366: 220      continue
367: 230      continue
368: *VOCL LOOP,NOVREC
369:         do 240 I = 1, NN
370:            RNU(IBP(I),N,1) = E2(I)
371: 240      continue
372: C
373: C      ..... TABULATION .....
374:
375:         else if ( LNU.eq.2 ) then
376:            call BSVDEC ( CX(LSNU), NNU, E, IEP2, NN )
377: *VOCL LOOP,NOVREC
378:            do 250 I = 1, NN
379:               LE = LSNU + IEP2(I)
380:               LL = LE + NNU
381:               D0 = (CX(LE)-E(I)) / (CX(LE)-CX(LL-1))
382:               RNU(IBP(I),N,1) = CX(LL)+D0*(CX(LL-1)-CX(LL))
383: 250      continue
384:         end if
385: C
386: C-----
387: C      Nu-delayed ( MT = 98 )
388: C-----
389: C
390:      LSNUD   = KLIB1(N,25)

```

src/mvp/gmicn1.f

```

391: C
392: C ..... COEFFICIENTS .....
393: C
394:   if ( LNUD.eq.1 ) then
395:     do 420 I = 1, NN
396:       E2(I) = CX(LSNUD+NNUD-1)
397: 420   continue
398:     do 440 K = NNUD - 2, 0, -1
399:       do 430 I = 1, NN
400:         E2(I) = E2(I)*E(I) + CX(LSNUD+K)
401: 430   continue
402: 440   continue
403:     do 450 I = 1, NN
404:       RNU(IBP(I),N,2) = E2(I)
405:       RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
406: 450   continue
407: C
408: C ..... TABULATION .....
409: C
410:   else if ( LNUD.eq.2 ) then
411:     call BSVDEC( CX(LSNUD), NNUD, E, IEP2, NN )
412: *VOCL LOOP,NOVREC
413:     do 460 I = 1, NN
414:       LE = LSNUD + IEP2(I)
415:       LL = LE + NNUD
416:       D0 = (CX(LL)-E(I)) / (CX(LE)-CX(LL-1))
417:       RNU(IBP(I),N,2) = CX(LL)+D0*(CX(LL-1)-CX(LL))
418:       RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
419: 460   continue
420:   end if
421: C
422:   if ( LNU.ne.0 ) then
423:     if ( JFMUL.eq.0 .or. LNUD.eq.0 ) then
424:       do 470 I = 1, NN
425:         SMCW(I,LMIC(2)) = RNU(IBP(I),N,1)*SMCW(I,LMIC(3))
426: 470     continue
427:     else if ( JFMUL.eq.1 .and. LNUD.ne.0 ) then
428:       do 480 I = 1, NN
429:         SMCW(I,LMIC(2)) = RNU(IBP(I),N,3)*SMCW(I,LMIC(3))
430: 480     continue
431:   end if
432:   end if
433: C
434: C-----
435: C (N,2N) (MT = 94 )
436: C-----
437:   if ( LMIC(7).ne.0 ) then
438:     if ( KLIB2(N,MT94,1).ne.0 ) then
439:       L0 = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
440: *VOCL LOOP,NOVREC
441:       do 260 I = 1, NN
442:         if ( IEP(I).ge.KLIB2(N,MT94,3)
443:           & .and. IEP(I).lt.KLIB2(N,MT94,4) ) then
444:           LL = L0 + IEP(I)
445:           SMCW(I,LMIC(7)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
446:         end if
447: 260       continue
448:     end if
449:   end if
450:   end if
451: C
452: C-----
453: C LOSS : CAPTURE + FISSION - (N,2N) - 2*(N,3N) - 3*(N,4N)
454: C-----
455: C

```

```

456: C      IF(JEIGN.NE.0) THEN
457: C
458:       if ( LMIC(8).ne.0 ) then
459: C
460: C ..... (N,2N) HAS BEEN CALCULATED (NSMIC >= 7) ....
461: C
462:       if ( LMIC(7).ne.0 ) then
463: *VOCL LOOP,NOVREC
464:       do 270 I = 1, NN
465:         R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
466:         & - SMCW(I,LMIC(7))
467: 270       continue
468:       else
469: C
470: C ..... (N,2N) IS CALCULATED HERE (NSMIC < 7) ....
471: C
472:       if ( KLIB2(N,MT94,1).ne.0 ) then
473:         L0 = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
474: C
475: *VOCL LOOP,NOVREC
476:       do 280 I = 1, NN
477:         if ( IEP(I).ge.KLIB2(N,MT94,3)
478:           & .and. IEP(I).lt.KLIB2(N,MT94,4) ) then
479:           LL = L0 + IEP(I)
480:           E2(I) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
481:         else
482:           E2(I) = 0.0
483:         end if
484: 280       continue
485:       R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3)) - E2(I)
486:       else
487: *VOCL LOOP,NOVREC
488:       do 290 I = 1, NN
489:         R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
490: 290       continue
491:       end if
492:     end if
493: C
494: C-----
495: C (N,3N) , (N,4N)
496: C-----
497: C
498: C
499:       if ( KLIB2(N,MT95,1).ne.0 ) then
500:         L0 = KLIB2(N,MT95,11) - KLIB2(N,MT95,3)
501: *VOCL LOOP,NOVREC
502:       do 300 I = 1, NN
503:         if ( IEP(I).ge.KLIB2(N,MT95,3)
504:           & .and. IEP(I).lt.KLIB2(N,MT95,4) ) then
505:           LL = L0 + IEP(I)
506:           E2(I) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
507:         else
508:           E2(I) = 0.0
509:         end if
510: 300       continue
511:       R(I) = R(I) - 2.0*E2(I)
512:       end if
513: C
514: C
515:       if ( KLIB2(N,MT96,1).ne.0 ) then
516:         L0 = KLIB2(N,MT96,11) - KLIB2(N,MT96,3)
517: *VOCL LOOP,NOVREC
518:       do 310 I = 1, NN
519:         if ( IEP(I).ge.KLIB2(N,MT96,3)
520:           & .and. IEP(I).lt.KLIB2(N,MT96,4) ) then

```

src/mvp/gmicn1.f

```
521:          LL      = L0 + IEP(I)
522:          E2(I)    = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
523:          else
524:            E2(I)    = 0.0
525:          end if
526:          R(I)      = R(I) - 3.0*E2(I)
527: 310      continue
528:      end if
529: C
530: C-----
531: C  COMPLETE CALCULATION OF SIGMA-LOSS
532: C-----
533: C
534: *VOCL LOOP,NOVREC
535:       do 320 I = 1, NN
536:         SMCW(I,LMIC(8)) = R(I)
537: 320      continue
538:       end if
539: C
540: C  ... save SMCW in bank array ...
541: C
542:       do 340 K = 1, NSMIC
543:         do 330 I = 1, NN
544:           SMIC(IBP(I),N,K) = SMCW(I,K)
545: 330      continue
546: 340      continue
547: C
548:       return
549:       end
```

src/mvp/gmicnx.f

```

1:      subroutine GMICNX( N, NN, IBP,  E ,  IRAND, JNP,  JGAMM,
2:      &                  NBANK, NSMIC, SMIC,  LMIC, KSPI,  NUCL,
3:      &                  NUC,  NMT,  CX,  ICX,  MCX,  KLIB1,
4:      &                  KLIB2, XLIB1,
5:      &                  E2,  IEP,  IEP2,
6:      &                  IBP2, DE,  R,  SMCW )
7: C=<MVP>=====
8: C PURPOSE: Get micro cross sections and put them in sigma-bank.
9: C          for N'th nuclide or N'th atom
10: C          (neutron and photon)
11: C          JNP : 1/2 = neutron/photon
12: C COMMENT: If nusigf is necessary without sigf, LMIC(3) should be set
13: C          to LMIC(2) in advance (EDITFL).
14: C          Only total cross section is calculated for photon.
15: C CALLED IN: GMACNX
16: C UPDATE:
17: C 22 Nov 2002: BUG-fixed-up for LSSSF=1 (only fission)
18: C
19: C=====
20: C
21: C .... NN: NUMBER OF PARTICLES  IBP: POINTER .....
22: C
23: C          integer IBP(NN)
24: C
25: C .... energy of neutrons
26: C
27: C          real E(NN)
28: C
29: C .... BANK .....
30: C
31: C          real SMIC(NBANK,NUC,NSMIC)
32: C          integer KSPI(NBANK,NUC)
33: C          integer LMIC(8)
34: C
35: C .... CROSS SECTION & POINTERS .....
36: C
37: C          real CX(MCX), XLIB1(NUCL,*)
38: C          integer ICX(MCX), KLIB1(NUCL,*), KLIB2(NUCL,NMT,*)
39: C
40: C .... WORKING ARRAY .....
41: C
42: C          real E2(NN), R(2*NN), DE(NN)
43: C          integer IEP(NN), IEP2(NN), IBP2(NN)
44: C##<2007/03/14:PN4:
45: C##  real SMCW(NN,NSMIC)
46: C##  real SMCW(NBANK,NSMIC)
47: C##>
48: C
49: C          integer MNEG
50: C          parameter( MWARN = 0 )
51: C
52: C          parameter( MT1 = 1, MT2 = 2, MT4 = 4, MT18 = 18, MT101 =
53: C          &          101, MT94 = 94, MT95 = 95, MT96 = 96 )
54: C
55: C          include '../shared/INC/_IOUNIT'
56: C
57: C          data MNEG /0/
58: C
59: C-----
60: C          ENERGY MESH POINT #
61: C-----
62: C
63: C          LEMESH = KLIB1(N,1)
64: C          NPTS = KLIB1(N,2)
65: C          call BSVDEC( CX(LEMESH), NPTS, E, IEP, NN )

```

```

66:
67: *VOCL LOOP,NOVREC
68:       do 100 I = 1, NN
69:         KSPI(IBP(I),N) = IEP(I)
70:         LE = LEMESH + IEP(I)
71:         DE(I) = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
72:       100 continue
73: C
74: C=====
75: C          ..... photon .....
76: C=====
77: C
78:       if ( JNP.eq.2 ) then
79: C
80:         L2 = KLIB2(N,2,5) - KLIB2(N,2,3)
81:         L3 = KLIB2(N,4,5) - KLIB2(N,4,3)
82:         L4 = KLIB2(N,6,5) - KLIB2(N,6,3)
83:         L5 = KLIB2(N,8,5) - KLIB2(N,8,3)
84: *VOCL LOOP,NOVREC
85:       do 110 I = 1, NN
86:         SMIC(IBP(I),N,1) = 0.0
87: C
88: C .... INCOHERENT SCATTERING .....
89: C
90:         if ( KLIB2(N,4,1).ne.0.and.IEP(I).ge.KLIB2(N,4,3)
91:         &      .and.IEP(I).lt.KLIB2(N,4,4) ) then
92:           LL3 = L3 + IEP(I)
93:           SMIC(IBP(I),N,1) =
94:         &      EXP(CX(LL3+1)+DE(I)*(CX(LL3)-CX(LL3+1)))
95:         end if
96: C
97: C .... PAIR PRODUCTION .....
98: C          (HAVING THRESHOULD AT 1.022 MEV
99: C
100:         if ( KLIB2(N,6,1).ne.0.and.IEP(I).ge.KLIB2(N,6,3)
101:         &      .and.IEP(I).lt.KLIB2(N,6,4) ) then
102:           LL4 = L4 + IEP(I)
103:           SMIC(IBP(I),N,1) = SMIC(IBP(I),N,1) +
104:         &      EXP(CX(LL4+1)+DE(I)*(CX(LL4)-CX(LL4+1)))
105:         end if
106: C
107: C .... PHOTOELECTIC ABSORPTION .....
108: C
109:         if ( KLIB2(N,8,1).ne.0.and.IEP(I).ge.KLIB2(N,8,3)
110:         &      .and.IEP(I).lt.KLIB2(N,8,4) ) then
111:           LL5 = L5 + IEP(I)
112:           SMIC(IBP(I),N,1) = SMIC(IBP(I),N,1) +
113:         &      EXP(CX(LL5+1)+DE(I)*(CX(LL5)-CX(LL5+1)))
114:         end if
115:       110 continue
116: C
117: C .... COHERENT SCATTERING .....
118: C
119:       if ( JGAMM.eq.0 ) then
120: *VOCL LOOP,NOVREC
121:       do 120 I = 1, NN
122:         if ( KLIB2(N,2,1).ne.0.and.IEP(I).ge.KLIB2(N,2,3)
123:         &      .and.IEP(I).lt.KLIB2(N,2,4) ) then
124:           LL2 = L2 + IEP(I)
125:           SMIC(IBP(I),N,1) = SMIC(IBP(I),N,1) +
126:         &      EXP(CX(LL2+1)+DE(I)*(CX(LL2)-CX(LL2+1)))
127:         end if
128:       120 continue
129:       end if
130: C

```

src/mvp/gmicnx.f

```

131:      return
132:      end if
133: C
134: C=====
135: C      ..... neutron .....
136: C=====
137: C
138: C      ..... CLEAR ALL CROSS SECTIONS .....
139: C
140:      do 140 K = 1, NSMIC
141:      do 130 I = 1, NN
142:          SMCW(I,K) = 0.0
143:      130 continue
144:      140 continue
145:
146:      LNU = KLIB1(N,14)
147:      NNU = KLIB1(N,9)
148:      N2 = LNU*NNU
149:      IUNRES = 0
150:
151: C-----
152: C      TOTAL      (MT = 1)
153: C-----
154:
155:      if ( LMIC(1).ne.0 ) then
156:          L1 = KLIB2(N,MT1,11) - KLIB2(N,MT1,3)
157:          IUNRES = IUNRES + 1
158: *VOCL LOOP,NOVREC
159:          do 150 I = 1, NN
160:              LL = L1 + IEP(I)
161:              SMCW(I,LMIC(1)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
162:      150 continue
163:          end if
164:
165: C-----
166: C      ELASTIC (MT = 2)
167: C-----
168:
169:      if ( LMIC(4).ne.0 ) then
170:          L2 = KLIB2(N,MT2,11) - KLIB2(N,MT2,3)
171:          IUNRES = IUNRES + 1
172: *VOCL LOOP,NOVREC
173:          do 160 I = 1, NN
174:              if ( KLIB1(N,15).ne.1 .or. E(I).gt.XLIB1(N,4) ) then
175:                  LL2 = L2 + IEP(I)
176:                  SMCW(I,LMIC(4)) = CX(LL2+1) + DE(I)*(CX(LL2)-CX(LL2+1))
177:              end if
178:      160 continue
179:          end if
180:
181: C-----
182: C      INELASTIC (MT = 4)
183: C-----
184:
185:      if ( LMIC(6).ne.0 ) then
186:          if ( KLIB2(N,MT4,1).ne.0 ) then
187:              L0 = KLIB2(N,MT4,11) - KLIB2(N,MT4,3)
188: *VOCL LOOP,NOVREC
189:              do 170 I = 1, NN
190:                  if ( IEP(I).ge.KLIB2(N,MT4,3)
191:                      & .and.IEP(I).lt.KLIB2(N,MT4,4) ) then
192:                      LL = L0 + IEP(I)
193:                      SMCW(I,LMIC(6)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
194:                  end if
195:      170 continue

```

```

196:      end if
197:      end if
198: C
199: C-----
200: C      FISSION (MT = 18)
201: C-----
202: C
203:      if ( LMIC(3).ne.0.and.KLIB2(N,MT18,1).ne.0 ) then
204:          L0 = KLIB2(N,MT18,11) - KLIB2(N,MT18,3)
205:          IUNRES = IUNRES + 1
206: *VOCL LOOP,NOVREC
207:          do 180 I = 1, NN
208:              if ( IEP(I).ge.KLIB2(N,MT18,3).and.IEP(I).lt.KLIB2(N,MT18,4)
209:                  & ) then
210:                  LL = L0 + IEP(I)
211:                  SMCW(I,LMIC(3)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
212:              end if
213:      180 continue
214:          end if
215:
216: C-----
217: C      CAPTURE (MT = 101)
218: C-----
219:
220:      if ( LMIC(5).ne.0.and.KLIB2(N,MT101,1).ne.0 ) then
221:          L0 = KLIB2(N,MT101,11) - KLIB2(N,MT101,3)
222:          IUNRES = IUNRES + 1
223: *VOCL LOOP,NOVREC
224:          do 190 I = 1, NN
225:              if ( IEP(I).ge.KLIB2(N,MT101,3)
226:                  & .and.IEP(I).lt.KLIB2(N,MT101,4) ) then
227:                  LL = L0 + IEP(I)
228:                  SMCW(I,LMIC(5)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
229:              end if
230:      190 continue
231:          end if
232: C
233: C-----
234: C      UNRESOLVED RESONANCE (TOTAL , ELASTIC, FISSION )
235: C-----
236: C
237:      NUNR = KLIB1(N,4)
238:      if ( IUNRES.ne.0.and.NUNR.ne.0 ) then
239:          NUNR2 = KLIB1(N,5)
240:          LSUNR = KLIB1(N,18)
241:          LLSSF = KLIB1(N,30)
242:
243: C      ..... SELECT PARTICLES IN UNRESOLVED RESONANCE ENERGY RANGE ....
244:
245:          MM = 0
246:          EL = CX(LSUNR+NUNR-1)
247: *VOCL LOOP,NOVREC
248:          do 200 I = 1, NN
249:              if ( E(I).le.CX(LSUNR).and.E(I).ge.EL ) then
250:                  MM = MM + 1
251:                  IBP2(MM) = I
252:                  E2(MM) = E(I)
253:              end if
254:      200 continue
255: C
256: C      ..... DETERMINE TABLE POSITION
257: C
258:          if ( MM.gt.0 ) then
259:              call BSVDEC( CX(LSUNR), NUNR, E2, IEP2, MM )
260: C

```

src/mvp/gmicnx.f

```

261: C      ..... SELECT TABLE .....
262:
263:      call RANU2( IRAND, R, MM, ICON )
264:      LINT      = LSUNR + NUNR
265:
266:      do 210 I = 1, MM
267:          EE2      = CX(LSUNR+IEP2(I))
268:          EE1      = CX(LSUNR+IEP2(I)-1)
269:          INTUN     = ICX(LINT+IEP2(I))
270:          if ( INTUN.eq.5 .or. INTUN.eq.3 ) then
271:              EE2      = LOG(EE2)
272:              EE1      = LOG(EE1)
273:              E2(I)     = LOG(EE2(I))
274:          end if
275:          if ( R(I).gt. (E2(I)-EE2)/(EE1-EE2) ) IEP2(I) = IEP2(I) +
276:      &      1
277:      210 continue
278: C
279:      call RANU2( IRAND, R, 2*MM, ICON )
280: C
281:      LS          = LSUNR + 2*NUNR
282:      NNEG        = 0
283: C
284: C      ..... CASE OF LLSSF=0 (BACKGROUND IS GIVEN) .....
285: C
286:      if ( LLSSF.eq.0 ) then
287:      *VOCL LOOP,NOVREC
288:          do 220 I = 1, MM
289:              LQ      = LS + (IEP2(I)-1)*6*NUNR2
290:              LIPOS    = LQ + NUNR2
291: C
292: C      ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
293: C
294: C/#IF ROUNDOff(NEAREST)
295:          KK          = MIN(NUNR2-1,INT(NUNR2*R(I)))
296:          K            = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
297: C/#ELSE
298:          *          KK          = NUNR2*R(I)
299:          *          K            = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
300: C/#ENDIF
301:          K            = ICX(LIPOS+K)
302:          LCRX         = LIPOS + 2*NUNR2 + 3*(K-1)
303:          JNEG         = 0
304:
305: C      ..... TOTAL .....
306:
307:          if ( LMIC(1).ne.0 ) then
308:              SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1))
309:      &      + CX(LCRX)
310:          if ( SMCW(IBP2(I),LMIC(1)).le.0.0 ) JNEG = 1
311:          end if
312:
313: C      ..... ELASTIC .....
314:
315:          if ( LMIC(4).ne.0 ) then
316:              SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4))
317:      &      + CX(LCRX+1)
318:          if ( SMCW(IBP2(I),LMIC(4)).lt.0.0 ) JNEG = 1
319:          end if
320:
321: C      ..... FISSION .....
322:
323:          if ( LMIC(3).ne.0 ) then
324:              SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3))
325:      &      + CX(LCRX+2)

```

```

326:          if ( SMCW(IBP2(I),LMIC(3)).lt.0.0 ) JNEG = 1
327:          end if
328:
329: C      ..... CAPTURE .....
330:
331:          if ( LMIC(5).ne.0 ) then
332:              SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5))
333:      &      + CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
334:          if ( SMCW(IBP2(I),LMIC(5)).lt.0.0 ) JNEG = 1
335:          end if
336: C
337: C      .... check non-positive cross section ...
338: C
339:          if ( JNEG.ne.0 ) then
340:              NNEG      = NNEG + 1
341:              CX0        = CX(LCRX)
342:              CX1        = CX(LCRX+1)
343:              CX2        = CX(LCRX+2)
344:              if ( LMIC(1).ne.0 )
345:      &      SMCW(IBP2(I),LMIC(1)) = 1.0E-5
346:              if ( LMIC(4).ne.0 )
347:      &      SMCW(IBP2(I),LMIC(4)) = 1.0E-5*CX1/CX0
348:              if ( LMIC(3).ne.0 )
349:      &      SMCW(IBP2(I),LMIC(3)) = 1.0E-5*CX2/CX0
350:              if ( LMIC(5).ne.0 )
351:      &      SMCW(IBP2(I),LMIC(5)) = 1.0E-5*(CX0-CX1-CX2) /CX0
352:          end if
353:      220 continue
354: C
355: C      ..... CASE OF LLSSF=1 (AVERAGED IS GIVEN) .....
356: C
357:      else
358: C
359:      *VOCL LOOP,NOVREC
360:          do 230 I = 1, MM
361:              LQ      = LS + (IEP2(I)-1)*(6*NUNR2+3)
362:              LIPOS    = LQ + NUNR2
363:              LSAV     = LQ + 6*NUNR2
364: C
365: C      ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
366: C
367: C/#IF ROUNDOff(NEAREST)
368:          KK          = MIN(NUNR2-1,INT(NUNR2*R(I)))
369:          K            = MIN(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
370: C/#ELSE
371:          *          KK          = NUNR2*R(I)
372:          *          K            = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
373: C/#ENDIF
374:          K            = ICX(LIPOS+K)
375:
376:          LCRX         = LIPOS + 2*NUNR2 + 3*(K-1)
377:
378:          SIGC         = CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
379:          SIGCAV        = CX(LSAV) - CX(LSAV+1) - CX(LSAV+2)
380: C
381: C      .... check non-positive capture cross section ...
382: C      if it or averaged one is non-positive,
383: C      infinite dilution cross sections are used.
384: C
385:          if ( SIGC.le.0.0 .or. SIGCAV.le.0.0 ) then
386:              NNEG      = NNEG + 1
387:          else
388:
389: C      ..... TOTAL .....
390:

```

src/mvp/gmicnx.f

```

391:      if ( LMIC(1).ne.0 )
392:      &      SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1)) /
393:      &      CX(LSAV)*CX(LCRX)
394:
395: C      .... ELASTIC .....
396:
397:      if ( LMIC(4).ne.0 )
398:      &      SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4)) /
399:      &      CX(LSAV+1)*CX(LCRX+1)
400:
401: C      .... FISSION .....
402:
403:      if ( LMIC(3).ne.0.and. CX(LSAV+2).gt.0.0 )
404:      &      SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3)) /
405:      &      CX(LSAV+2)*CX(LCRX+2)
406:
407: C      .... CAPTURE .....
408:
409:      if ( LMIC(5).ne.0 )
410:      &      SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5)) /
411:      &      SIGCAV*SIGC
412:
413:      end if
414: 230      continue
415:      end if
416: C
417: C      .... warning message for non-positive cross section fixup. ....
418: C
419:      if ( NNEG.gt.0 ) then
420:      if ( MNEG.lt.MWARN ) then
421:      write(IOW,7000) NNEG, N
422: 7000      format('!!! (GMICNX) FOR',I5,
423:      &      ' particles, cross sections',
424:      &      ' are negative in unresolved resonance region'
425:      &      /' for ',I4,'-th nuclide.')
426:      MNEG = MNEG + NNEG
427:
428:      if ( MNEG.ge.MWARN ) then
429:      write(IOW,*)
430:      &      '!!! (GMICN1) NEGATIVE X-SEC MESSAGES ',
431:      &      'WILL BE SUPPRESSED HEREAFTER'
432:
433:      end if
434:      end if
435:      end if
436:      end if
437: C-----
438: C      NU * FISSION
439: C-----
440:
441: C
442: C      .... COEFFICIENTS .....
443:
444:      LSNU = KLIB1(N,19)
445: C
446:      if ( LMIC(2).ne.0.and.LNU.eq.1 ) then
447:      do 240 I = 1, NN
448:      E2(I) = CX(LSNU+NNU-1)
449: 240      continue
450:      do 250 K = NNU - 2, 0, -1
451:      do 260 I = 1, NN
452:      E2(I) = E2(I)*E(I) + CX(LSNU+K)
453: 260      continue
454: 250      continue
455: *VOCL LOOP,NOVREC

```

```

456:      do 270 I = 1, NN
457:      SMCW(I,LMIC(2)) = E2(I)*SMCW(I,LMIC(3))
458: 270      continue
459: C
460: C      .... TABULATION .....
461:
462:      else if ( LMIC(2).ne.0.and.LNU.eq.2 ) then
463:      call BSVDEC (CX(LSNU), NNU, E, IEP2, NN )
464: *VOCL LOOP,NOVREC
465:      do 280 I = 1, NN
466:      LE = LSNU + IEP2(I)
467:      LL = LE + NNU
468:      D0 = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
469:      SMCW(I,LMIC(2)) = (CX(LL)+D0*(CX(LL-1)-CX(LL)))*
470:      &      SMCW(I,LMIC(3))
471: 280      continue
472:      end if
473:
474: C-----
475: C      (N,2N) (MT = 94 )
476: C-----
477:
478:      if ( LMIC(7).ne.0 ) then
479:      if ( KLIB2(N,MT94,1).ne.0 ) then
480:      L0 = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
481: *VOCL LOOP,NOVREC
482:      do 290 I = 1, NN
483:      if ( IEP(I).ge.KLIB2(N,MT94,3)
484:      &      .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
485:      LL = L0 + IEP(I)
486:      SMCW(I,LMIC(7)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
487:      end if
488: 290      continue
489:      end if
490:      end if
491: C
492: C-----
493: C      LOSS : CAPTURE + FISSION - (N,2N) - 2*(N,3N) - 3*(N,4N)
494: C-----
495: C
496:      if ( LMIC(8).ne.0 ) then
497: C
498: C      .... (N,2N) HAS BEEN CALCULATED (NSMIC >= 7) ....
499:
500:      if ( LMIC(7).ne.0 ) then
501: *VOCL LOOP,NOVREC
502:      do 300 I = 1, NN
503:      R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
504:      &      - SMCW(I,LMIC(7))
505: 300      continue
506:      else
507: C
508: C      .... (N,2N) IS CALCULATED HERE (NSMIC < 7) ....
509:
510:      if ( KLIB2(N,MT94,1).ne.0 ) then
511:      L0 = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
512: C
513: *VOCL LOOP,NOVREC
514:      do 310 I = 1, NN
515:      if ( IEP(I).ge.KLIB2(N,MT94,3)
516:      &      .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
517:      LL = L0 + IEP(I)
518:      E2(I) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
519:      else
520:      E2(I) = 0.0

```

src/mvp/gmicnx.f

```

521:                end if
522:
523:                R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3)) - E2(I)
524:    310          continue
525:          else
526:    *VOCL LOOP,NOVREC
527:            do 320 I = 1, NN
528:              R(I) = SMCW(I,LMIC(5)) + SMCW(I,LMIC(3))
529:    320          continue
530:            end if
531:          end if
532: C
533: C-----
534: C (N,3N) , (N,4N)
535: C-----
536: C
537:       if ( KLIB2(N,MT95,1).ne.0 ) then
538:         L0 = KLIB2(N,MT95,11) - KLIB2(N,MT95,3)
539:    *VOCL LOOP,NOVREC
540:       do 330 I = 1, NN
541:         if ( IEP(I).ge.KLIB2(N,MT95,3)
542:           & .and.IEP(I).lt.KLIB2(N,MT95,4) ) then
543:           LL = L0 + IEP(I)
544:           E2(I) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
545:         else
546:           E2(I) = 0.0
547:         end if
548:
549:         R(I) = R(I) - 2.0*E2(I)
550:    330       continue
551:       end if
552: C
553:       if ( KLIB2(N,MT96,1).ne.0 ) then
554:         L0 = KLIB2(N,MT96,11) - KLIB2(N,MT96,3)
555:    *VOCL LOOP,NOVREC
556:       do 340 I = 1, NN
557:         if ( IEP(I).ge.KLIB2(N,MT96,3)
558:           & .and.IEP(I).lt.KLIB2(N,MT96,4) ) then
559:           LL = L0 + IEP(I)
560:           E2(I) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
561:         else
562:           E2(I) = 0.0
563:         end if
564:         R(I) = R(I) - 3.0*E2(I)
565:    340       continue
566:       end if
567: C
568: C-----
569: C COMPLETE CALCULATION OF SIGMA-LOSS
570: C-----
571: C
572:    *VOCL LOOP,NOVREC
573:       do 350 I = 1, NN
574:         SMCW(I,LMIC(8)) = R(I)
575:    350       continue
576:       end if
577: C
578: C ... save SMCW in bank array ...
579: C
580:       do 360 K = 1, NSMIC
581:         do 370 I = 1, NN
582:           SMIC(IBP(I),N,K) = SMCW(I,K)
583:         370       continue
584:       360     continue
585: C
586:       return
587:       end

```


src/mvp/gmicp1.f

```

1:      subroutine GMICP1( N, NN,IBP, E, JGAMM, NUC,  NMTP, NPATOM,
2:      &                NBANK, NSMIC, SMIC,  KSPI,
3:      X                CXP,  ICXP, MCXP,  KLBP1, KLBP2, XLBP2,
4:      W                IEP,  DE,  IP1 )
5: C=====
6: C PURPOSE: Get micro cross sections and put them in sigma-bank.
7: C         for N'th nuclide
8: C         FOR PHOTONS
9: C CALLED IN:  PHOTR, SOURCE
10: C-----
11: C
12: C .... NN: NUMBER OF PARTICLES  IBP: POINTER .....
13: C
14: C integer IBP(NN)
15: C
16: C .... lethargy
17: C
18: C real E(NN)
19: C
20: C .... BANK .....
21: C
22: C real SMIC(NBANK,NUC,NSMIC)
23: C integer KSPI(NBANK,NUC)
24: C
25: C .... CROSS SECTION & POINTERS .....
26: C
27: C real CXP(MCXP)
28: C integer ICXP(MCXP), KLBP1(NPATOM,20), KLBP2(NPATOM,NMTP,6)
29: C real XLBP2(NPATOM,NMTP,2)
30: C
31: C-----
32: C POSITIONING FOR REACTIONS
33: C
34: C 1 : TOTAL
35: C 2 : COHERENT SCATTERING
36: C 3 : INCOHERENT SCATTERING
37: C 4 : PAIR PRODUCTION TOTAL
38: C 5 : PHOTOELECTRIC
39: C-----
40: C
41: C .... WORKING ARRAY .....
42: C
43: C real DE(NBANK)
44: C integer IEP(NBANK), IP1(NBANK)
45: C
46: C parameter( MT1 = 1 )
47: C parameter( MT2 = 2 )
48: C parameter( MT3 = 4 )
49: C parameter( MT4 = 6 )
50: C parameter( MT5 = 8 )
51: C
52: C-----
53: C GATHER ENERGY ( -> LOG )
54: C-----
55: C
56: C do 100 I = 1, NN
57: C   E(I) = LOG(EEE(IBP(I)))
58: C 100 continue
59: C
60: C NPTS = KLBP1(N,2)
61: C-----
62: C ENERGY MESH POINT #
63: C-----
64: C LEMESH = KLBP1(N,1)
65: C call BSVDEC( CXP(LEMESH), NPTS, E, IEP, NN )

```

```

66: C
67: C do 100 I = 1, NN
68: C   KSPI(IBP(I),N) = IEP(I)
69: C   LE = LEMESH + IEP(I)
70: C   DE(I) = (CXP(LE)-E(I)) / (CXP(LE)-CXP(LE-1))
71: C 100 continue
72: C
73: C
74: C-----
75: C
76: C ..... LSTF3 - ISTID .....
77: C***** L1 = KLBP2(N,MT1,5) - KLBP2(N,MT1,3)
78: C L2 = KLBP2(N,MT2,5) - KLBP2(N,MT2,3)
79: C L3 = KLBP2(N,MT3,5) - KLBP2(N,MT3,3)
80: C L4 = KLBP2(N,MT4,5) - KLBP2(N,MT4,3)
81: C L5 = KLBP2(N,MT5,5) - KLBP2(N,MT5,3)
82: C*VOCL LOOP,NOVREC
83: C do 110 I = 1, NN
84: C   SMIC(IBP(I),N,2) = 0.0
85: C   SMIC(IBP(I),N,3) = 0.0
86: C   SMIC(IBP(I),N,4) = 0.0
87: C   SMIC(IBP(I),N,5) = 0.0
88: C
89: C .... TOTAL .....
90: C
91: C LL = L1 + IEP(I)
92: C SMIC(IBP(I),N,1) = EXP(
93: C @ CXP(LL+1)+DE(I)*(CXP(LL) - CXP(LL+1)) )
94: C
95: C .... COHERENT SCATTERING .....
96: C
97: C if ( JGAMM.eq.0 ) then
98: C   if ( KLBP2(N,MT2,1).ne.0.and.IEP(I).ge.KLBP2(N,MT2,3)
99: C & .and.IEP(I).lt.KLBP2(N,MT2,4) ) then
100: C
101: C LL2 = L2 + IEP(I)
102: C SMIC(IBP(I),N,2) =
103: C & EXP(CXP(LL2+1)+DE(I)*(CXP(LL2)-CXP(LL2+1)))
104: C   end if
105: C end if
106: C
107: C .... INCOHERENT SCATTERING .....
108: C
109: C if ( KLBP2(N,MT3,1).ne.0.and.IEP(I).ge.KLBP2(N,MT3,3)
110: C & .and.IEP(I).lt.KLBP2(N,MT3,4) ) then
111: C LL3 = L3 + IEP(I)
112: C SMIC(IBP(I),N,3) =
113: C & EXP(CXP(LL3+1)+DE(I)*(CXP(LL3)-CXP(LL3+1)))
114: C   end if
115: C
116: C .... PAIR PRODUCTION .....
117: C (HAVING THRESHOLD AT 1.022 MEV
118: C
119: C IF( E(I) .GE. LOG(1.022D6) .AND.
120: C @ KLBP2(N,MT4,1).NE.0 .AND. E(I).GE.XLBP2(N,MT4,1)
121: C if ( KLBP2(N,MT4,1).ne.0.and.IEP(I).ge.KLBP2(N,MT4,3)
122: C & .and.IEP(I).lt.KLBP2(N,MT4,4) ) then
123: C
124: C LL4 = L4 + IEP(I)
125: C SMIC(IBP(I),N,4) =
126: C & EXP(CXP(LL4+1)+DE(I)*(CXP(LL4)-CXP(LL4+1)))
127: C   end if
128: C
129: C .... PHOTOELECTIC ABSORPTION .....
130: C

```

src/mvp/gmicp1.f

```
131: CM          IF(   KLBP2(N,MT5,1).NE.0 .AND. E(I).GE.XLBP2(N,MT5,1)
132:             if ( KLBP2(N,MT5,1).ne.0.and.IEP(I).ge.KLBP2(N,MT5,3)
133:             &    .and.IEP(I).lt.KLBP2(N,MT5,4) ) then
134:                 LL5      = L5 + IEP(I)
135:                 SMIC(IBP(I),N,5) =
136:             &    EXP(CXP(LL5+1)+DE(I)*(CXP(LL5)-CXP(LL5+1)))
137:             end if
138: C
139: C .... TOTAL .....
140: C
141:             SMIC(IBP(I),N,1) = SMIC(IBP(I),N,2) + SMIC(IBP(I),N,3)
142:             &    + SMIC(IBP(I),N,4) + SMIC(IBP(I),N,5)
143: C
144: 110 continue
145: C
146: return
147: end
```

src/mvp/gmicpn1.f

```

1:      subroutine GMICPN1(N,      NN,      IBP,      E,      NUC,      NUCPN,
2:      &      NUCPNI,NMTPN, NBANK, NUC_MAX, NSMICPN,
3:      &      SMICPN,CXPN, ICXPN, MCXPN, KLBPN1,KLBPN2,
4:      &      XLBPN1,EBOTLPN,      E2,      IEP,      IEP2, DE,
5:      &      SMCW, RNU )
6: C=<MVP>=====
7: C  PURPOSE:  get micro cross sections and put then in sigma-bank for
8: C            n'th nuclide. (photonuclear)
9: C  CALLED IN:  PHOTR, SOURCE, GTMACPN
10: C  CALLS:  BSVDEC
11: C=====
12: C
13: C  .... NN: NUMBER OF PARTICLES,      IBP: BANK POINTER,      E: energy .....
14: C
15: C      integer IBP(NN)
16: C      real E(NN)
17: C
18: C  .... BANK .....
19: C
20: C      real SMICPN(NBANK,NUCPN,NSMICPN)
21: C
22: C  .... RNU : nu-bar (1/2/3=total/delayed/prompt)
23: C
24: C      real RNU(NBANK,NUC_MAX,3)
25: C
26: C  .... CROSS SECTION & POINTERS .....
27: C
28: C      real CXPN(MCXPN), XLBPN1(NUCPNI,6), EBOTLPN
29: C      integer ICXPN(MCXPN), KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
30: C
31: C  .... WORKING ARRAY .....
32: C
33: C      real E2(NBANK), DE(NBANK)
34: C      integer IEP(NBANK), IEP2(NBANK)
35: C      real SMCW(NBANK,NSMICPN)
36: C
37: C      parameter( MT3 = 3, MT18 = 18 )
38: C
39: C-----
40: C      initialization
41: C-----
42: C
43: C      N2      = 0
44: C      do I = 1, NN
45: C          if ( E(I).lt.EBOTLPN ) N2 = N2 + 1
46: C      end do
47: C      if ( N2.eq.NN ) then
48: C          do K = 1, NSMICPN
49: C              do I = 1, NN
50: C                  SMICPN(IBP(I),N,K) = 0
51: C              end do
52: C          end do
53: C          go to 300
54: C      end if
55: C
56: C      NPTS      = KLBPN1(N,2)
57: C      LNU      = KLBPN1(N,9)
58: C      NNU      = KLBPN1(N,6)
59: C      LNUD      = KLBPN1(N,12)
60: C      NNUD      = KLBPN1(N,11)
61: C      N2      = LNU*NNU
62: C
63: C-----
64: C      ENERGY MESH POINT #
65: C-----

```

```

66: C
67: C      LEMESH      = KLBPN1(N,1)
68: C      call BSVDEC( CXPN(LEMESH), NPTS, E, IEP, NN )
69: C
70: C      *VOCL LOOP,NOVREC
71: C      do I = 1, NN
72: C          LE      = LEMESH + IEP(I)
73: C          DE(I)      = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
74: C      end do
75: C
76: C      ..... CLEAR ALL CROSS SECTIONS .....
77: C      do K = 1, NSMICPN
78: C          do I = 1, NN
79: C              SMCW(I,K) = 0
80: C          end do
81: C      end do
82: C      do K = 1, 3
83: C          do I = 1, NN
84: C              RNU(IBP(I),N,K) = 0
85: C          end do
86: C      end do
87: C
88: C-----
89: C      NONELASTIC (MT = 3)
90: C-----
91: C
92: C      L1      = KLBPN2(N,MT3,10) - KLBPN2(N,MT3,3)
93: C      *VOCL LOOP,NOVREC
94: C      do I = 1, NN
95: C          if ( IEP(I).ge.KLBPN2(N,MT3,3) .and.
96: C              &      IEP(I).lt.KLBPN2(N,MT3,4) ) then
97: C              LL      = L1 + IEP(I)
98: C              SMCW(I,1) = CXPN(LL+1) + DE(I)*(CXPN(LL)-CXPN(LL+1))
99: C          end if
100: C      end do
101: C
102: C-----
103: C      FISSION (MT = 18)
104: C-----
105: C
106: C      if ( KLBPN2(N,MT18,1).ne.0 ) then
107: C          L0      = KLBPN2(N,MT18,10) - KLBPN2(N,MT18,3)
108: C      *VOCL LOOP,NOVREC
109: C      do I = 1, NN
110: C          if ( IEP(I).ge.KLBPN2(N,MT18,3) .and.
111: C              &      IEP(I).lt.KLBPN2(N,MT18,4) ) then
112: C              LL      = L0 + IEP(I)
113: C              SMCW(I,2) = CXPN(LL+1) + DE(I)*(CXPN(LL)-CXPN(LL+1))
114: C          end if
115: C      end do
116: C      end if
117: C
118: C-----
119: C      NU * FISSION (NU = nu-total)
120: C-----
121: C
122: C      LSNU      = KLBPN1(N,10)
123: C
124: C      ..... COEFFICIENTS .....
125: C      if ( LNU.eq.1 ) then
126: C          do I = 1, NN
127: C              E2(I)      = CXPN(LSNU+NNU-1)
128: C          end do
129: C          do K = NNU-2, 0, -1
130: C              do I = 1, NN

```

src/mvp/gmicpn1.f

```

131:          E2(I) = E2(I)*E(I) + CXPN(LSNU+K)
132:        end do
133:      end do
134:      do I = 1, NN
135:        RNU(IBP(I),N,1) = E2(I)
136:      end do
137: *VOCL LOOP,NOVREC
138:      do I = 1, NN
139:        SMCW(I,2) = E2(I) * SMCW(I,2)
140:      end do
141: C
142: C ..... TABULATION .....
143: C else if ( LNU.eq.2 ) then
144: C   call BSVDEC( CXPN(LSNU), NNU, E, IEP2, NN )
145: *VOCL LOOP,NOVREC
146:   do I = 1, NN
147:     LE = LSNU + IEP2(I)
148:     LL = LE + NNU
149:     D0 = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
150:     SMCW(I,2) = (CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))) *
151: & SMCW(I,2)
152:     RNU(IBP(I),N,1) = CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))
153:   end do
154: end if
155: C
156: C -----
157: C Nu-delayed
158: C -----
159: C
160:   LSNUD = KLBPNI(N,14)
161: C
162: C ..... COEFFICIENTS .....
163: C if ( LNUD.eq.1 ) then
164: C   do I = 1, NN
165: C     E2(I) = CXPN(LSNUD+NNUD-1)
166: C   end do
167: C   do K = NNUD-2, 0, -1
168: C     do I = 1, NN
169: C       E2(I) = E2(I)*E(I) + CXPN(LSNUD+K)
170: C     end do
171: C   end do
172: C   do I = 1, NN
173: C     RNU(IBP(I),N,2) = E2(I)
174: C     RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
175: C   end do
176: C
177: C ..... TABULATION .....
178: C else if ( LNUD.eq.2 ) then
179: C   call BSVDEC( CXPN(LSNUD), NNUD, E, IEP2, NN )
180: *VOCL LOOP,NOVREC
181:   do I = 1, NN
182:     LE = LSNUD + IEP2(I)
183:     LL = LE + NNUD
184:     D0 = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
185:     RNU(IBP(I),N,2) = CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))
186:     RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
187:   end do
188: end if
189: C
190: C -----
191: C save SMCW in bank array
192: C -----
193: C
194:   do K = 1, NSMICPN
195:     do I = 1, NN

```

```

196:          SMICPN(IBP(I),N,K) = SMCW(I,K)
197:        end do
198:      end do
199: C
200:   300 continue
201:   return
202: end
203: C
204: C
205:   subroutine GMICPN2(N, NN, E, NUCPNI,NMTPN,
206: & CXPN, MCXPN, INCSTPN2, MTMPN, NMTMPN,
207: & KLBPNI,KLBPNI2,XLBPNI,EBOTLPN,
208: & IEP, DE, SMCN )
209: C=<MVP>=====
210: C PURPOSE: get the response micro cross sections, for n'th nuclide.
211: C (photonuclear)
212: C CALLED IN: STALNI
213: C CALLS: BSVDEC
214: C=====
215: C
216:   real E(NN)
217: C
218: C .... CROSS SECTION & POINTERS .....
219: C
220:   real CXPN(MCXPN), XLBPNI(NUCPNI,6), EBOTLPN
221:   integer KLBPNI(NUCPNI,16), KLBPNI2(NUCPNI,NMTPN,13)
222:   integer MTMPN(NMTMPN)
223: C
224: C .... WORKING ARRAY .....
225: C
226:   real DE(NN), SMCN(NN)
227:   integer IEP(NN)
228: C
229: C -----
230: C initialization
231: C -----
232: C
233:   N2 = 0
234:   do I = 1, NN
235:     if ( E(I).lt.EBOTLPN ) N2 = N2 + 1
236:   end do
237:   if ( N2.eq.NN ) then
238:     do I = 1, NN
239:       SMCN(I) = 0
240:     end do
241:     go to 300
242:   end if
243: C
244:   NPTS = KLBPNI(N,2)
245:   IMT = INCSTPN2 / 100
246:   ND = INCSTPN2 - IMT * 100
247: C
248: C -----
249: C ENERGY MESH POINT #
250: C -----
251: C
252:   LEMESH = KLBPNI(N,1)
253:   call BSVDEC( CXPN(LEMESH), NPTS, E, IEP, NN )
254: C
255: *VOCL LOOP,NOVREC
256:   do I = 1, NN
257:     LE = LEMESH + IEP(I)
258:     DE(I) = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
259:   end do
260: C

```

src/mvp/gmicpn1.f

```
261: C      ..... CLEAR ALL CROSS SECTIONS .....
262:      do I = 1, NN
263:          SMCN(I) = 0
264:      end do
265: C
266: C-----
267: C      response cross sections (any MTs)
268: C-----
269: C
270:      if ( ND.gt.0 ) then
271:          do M = 1, ND
272:              MTMM = MTMPN(IMT+M-1)
273:              if ( KLBN2(N,MTMM,1).ne.0 ) then
274:                  L0 = KLBN2(N,MTMM,10) - KLBN2(N,MTMM,3)
275: *VOCL LOOP,NOVREC
276:                  do I = 1, NN
277:                      if ( E(I).gt.KLBN1(N,3).and.
278:                          & (IEP(I).ge.KLBN2(N,MTMM,3) .and.
279:                          & IEP(I).lt.KLBN2(N,MTMM,4)) ) then
280:                          LL = L0 + IEP(I)
281:                          SMCN(I) = SMCN(I) + CXPN(LL+1) +
282:                          & DE(I) * (CXPN(LL)-CXPN(LL+1))
283:                      end if
284:                  end do
285:              end if
286:          end do
287:      end if
288: C
289:      300 continue
290:      return
291:      end
```

src/mvp/gmicpt.f

```

1:      subroutine GMICPT( NN,   IBP,   IRAND, JEIGN, NUC,   NMT,
2:      &                  NBANK, NSMIC, EEE,   SMIC, LMIC, KSPI, CX,
3:      &                  ICX,   MCX,   KLIB1, KLIB2, XLIB1,
4:      &                  SGTAL, CRES, KCRES, NSTAL, MCRES, E,   E2,
5:      &                  IEP,   IEP2, IBP2, DE,   IP1,   R, SMCW,
6:      &                  SMICP, CXPl, SMCWP, E2P,   DEP,   RP )
7:
8: C=<MVP>=====
9: C  PURPOSE: GET MICRO CROSS SECTIONS AND PUT THEM IN SIGMA-BANK.
10: C          (NEUTRON )
11: C  CALLED IN: NEUTR, SOURCE
12: C
13: C  UPDATE:
14: C  JAN 15 1992: CLEAR CROSS SECTIONS FIRST.
15: C          PURPOSE: 1. TO PREVENT PREPARING INVALID CROSS SECTION IN
16: C                     MIXED-PARTICLE PROBLEMS.
17: C                     2. BYPASS OPERATION ' ELSE SMIC(..) = 0.0 '
18: C          (M.SASAKI)
19: C 17 Mar 1994: B.M.C. SAMPLING OF PROBABILITY TABLE in unresolved energy
20: C               range may cause memory fault in nearest-value rounding
21: C               of floating point number.
22: C  9 Sep 1994: Increase second dimension of 'XLIB1' from 9 to 11.
23: C  FEB 15,1995: When "CX" in unresolved resonance region <= 0.0, set
24: C               ST=10**-5
25: C               SE=10**-5 * SE(PROBABILITY TABLE) / SE(PROBABILITY)
26: C               etc.
27: C 31 Aug 1995: make reaction MT numbers to parameter constannts.
28: C               (MT1,MT2,MT4,MT18,...)
29: C  6 Nov 1997: BMC sampling in unresolved resonance in ROUNDOP(NEAREST)
30: C               makes ridicurous index in a rare case;
31: C
32: C               K      = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
33: C
34: C               K is calculated as 2 + 2*KK when CX(LQ+KK)=0 and R(MM+I)=1
35: C               and KK=NUNR2-1, but K should be less than that.
36: C
37: C               K      = min(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
38: C
39: C               is safer.
40: C
41: C 12 Aug 1998: add function to treat LLSSF=1 data for unresolved
42: C               resonance in ENDF/B6 format.
43: C 28 Jan 1999: add new working array SMCW to aviod extensive indexed
44: C               access to SMIC.
45: C  1 Feb 1999: loop setting KSPI does not have NOVREC directive.
46: C=====
47: C
48: C   .... NN: NUMBER OF PARTICLES  IBP: POINTER  ....
49: C
50: C       integer IBP(NN)
51: C
52: C   .... BANK  ....
53: C
54: C       real      EEE(NBANK), SMIC(NBANK,NUC,NSMIC), SGTAL(NBANK,NSTAL)
55: C       integer   KSPI(NBANK,NUC), LMIC(8)
56: C+f.k 1999.9.29
57: C       real      SMICP(NBANK,NUC,NSMIC)
58: C-f.k 1999.9.29
59: C
60: C   .... CROSS SECTION & POINTERS  ....
61: C
62: C       real      CX(MCX), XLIB1(NUC,11)
63: C       integer   ICX(MCX), KLIB1(NUC,30), KLIB2(NUC,NMT,13)
64: C+f.k 1999.9.29
65: C       real      CXPl(MCX)

```

```

66: C-f.k 1999.9.29
67: C
68: C       real      CRES(MCRES)
69: C       integer   KCRES(NSTAL,4)
70: C
71: C   .... WORKING ARRAY  ....
72: C
73: C       real      E(NBANK), E2(NBANK), R(2*NBANK), DE(NBANK)
74: C       integer   IEP(NBANK), IEP2(NBANK), IBP2(NBANK), IP1(NBANK)
75: C       real      SMCW(NBANK,NSMIC)
76: C+f.k 1999.9.29
77: C       real      SMCWP(NBANK,NSMIC)
78: C       real      E2P(NBANK), RP(2*NBANK), DEP(NBANK)
79: C-f.k 1999.9.29
80: C
81: C       integer   MNEG
82: C       parameter( MWARN      = 20 )
83: C       parameter( MWARN      = 2000 )
84: C
85: C       parameter( MT1 = 1, MT2 = 2, MT4 = 4, MT18 = 18,
86: C               &      MT101 = 101, MT94 = 94, MT95=95, MT96=96 )
87: C
88: C       include ' ../shared/INC/_IOUNIT '
89: C
90: C       data MNEG /0/
91: C
92: C-----
93: C       GATHER ENERGY
94: C-----
95: C
96: C       do 100 I = 1, NN
97: C           E(I) = EEE(IBP(I))
98: C       100 continue
99: C
100: C-----
101: C--- LOOP FOR NUCLIDES -----
102: C-----
103: C
104: C       MT1      = 1
105: C       MT2      = 2
106: C
107: C       do 330 N = 1, NUC
108: C           NPTS  = KLIB1(N,2)
109: C           LNU   = KLIB1(N,14)
110: C           NNU   = KLIB1(N,9)
111: C           N2    = LNU*NNU
112: C
113: C-----
114: C       ENERGY MESH POINT #
115: C-----
116: C
117: C       LEMESH = KLIB1(N,1)
118: C       call BSVDEC( CX(LEMESH), NPTS, E, IEP, NN )
119: C
120: C *VOCL LOOP,NOVREC
121: C       do 110 I = 1, NN
122: C           KSPI(IBP(I),N) = IEP(I)
123: C           LE             = LEMESH + IEP(I)
124: C           DE(I)          = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
125: C           DEP(I)         = (CXPl(LE)-E(I)) / (CXPl(LE)-CXPl(LE-1))
126: C           IP1(I)         = (N-1)*NBANK + IBP(I)
127: C           CCCCCCCCCC DE1(I) = 1.0-DE(I)
128: C       110 continue
129: C
130: C       ..... CLEAR ALL CROSS SECTIONS .....

```

src/mvp/gmicpt.f

```

131: C      ( FROM JAN 15 1992 : TO PREVENT ERRONEOUS REACTION IN
132: C      MIXED PARTICLE MODE )
133: C
134: C      do 130 K = 1, NSMIC
135: C      do 120 I = 1, NN
136: CCC      SMIC(IBP(I),N,K)      = 0.0
137:      SMCW(I,K)      = 0.0
138:      SMCWP(I,K)      = 0.0
139:      120      continue
140:      130      continue
141:
142: C-----
143: C      TOTAL      (MT = 1) & ELASTIC (MT = 2)
144: C-----
145:
146:      LL      = KLIB2(N,MT1,11) - KLIB2(N,MT1,3)
147:      L2      = KLIB2(N,MT2,11) - KLIB2(N,MT2,3)
148: *VOCL LOOP,NOVREC
149: do 140 I = 1, NN
150:      LL      = L1 + IEP(I)
151: CCCCC      SMIC(IP1(I),1,LMIC(1)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
152:      SMCW(I,LMIC(1)) = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
153:      SMCWP(I,LMIC(1)) = CXPL(LL+1)+DEP(I)*(CXPL(LL)-CXPL(LL+1))
154:      if ( KLIB1(N,15).ne.1 .or. E(I).gt.XLIB1(N,4) ) then
155:      LL2      = L2 + IEP(I)
156: CCCCC      SMIC(IP1(I),1,LMIC(4)) = CX(LL2+1) + DE(I)*
157:      SMCW(I,LMIC(4)) = CX(LL2+1) + DE(I)*
158:      &      (CX(LL2)-CX(LL2+1))
159:      SMCWP(I,LMIC(4)) = CXPL(LL2+1) + DEP(I)*
160:      &      (CXPL(LL2)-CXPL(LL2+1))
161:      *****      ELSE
162:      *****      SMIC(IP1(I),1,LMIC(4)) = 0.0
163:      end if
164:
165:      140      continue
166:
167: C-----
168: C      INELASTIC (MT = 4)
169: C-----
170:
171:      if ( LMIC(6).ne.0 ) then
172:      MT      = 4
173:      if ( KLIB2(N,MT4,1).ne.0 ) then
174:      L0      = KLIB2(N,MT4,11) - KLIB2(N,MT4,3)
175: *VOCL LOOP,NOVREC
176: do 150 I = 1, NN
177:      if ( IEP(I).ge.KLIB2(N,MT4,3)
178:      &      .and.IEP(I).lt.KLIB2(N,MT4,4) ) then
179:      LL      = L0 + IEP(I)
180: CCCCC      SMIC(IP1(I),1,LMIC(6)) = CX(LL+1) + DE(I)*
181:      SMCW(I,LMIC(6)) = CX(LL+1) + DE(I)*
182:      &      (CX(LL)-CX(LL+1))
183:      SMCWP(I,LMIC(6)) = CXPL(LL+1) + DEP(I)*
184:      &      (CXPL(LL)-CXPL(LL+1))
185:      *****      ELSE
186:      *****      SMIC(IP1(I),1,LMIC(6))=0.0
187:      end if
188:      150      continue
189:      end if
190:      end if
191: C
192: C-----
193: C      FISSION (MT = 18)
194: C-----
195: C

```

```

196: Ccc      MT      = 18
197:
198:      if ( KLIB2(N,MT18,1).ne.0 ) then
199:
200: Ccccc      CALL      CXSINT(NN,IBP, N,MT18,
201: CccccX      SMIC(1,N,LMIC(3)), CX, MCX, KLIB2,NUC,NMT,
202: CccccW      IEP ,DE, DEL, NBANK)
203:
204:      L0      = KLIB2(N,MT18,11) - KLIB2(N,MT18,3)
205:
206: *VOCL LOOP,NOVREC
207: do 160 I = 1, NN
208:      if ( IEP(I).ge.KLIB2(N,MT18,3)
209:      &      .and.IEP(I).lt.KLIB2(N,MT18,4) ) then
210:      LL      = L0 + IEP(I)
211: CCCCC      SMIC(IP1(I),1,LMIC(3)) = CX(LL+1) + DE(I)*
212:      SMCW(I,LMIC(3)) = CX(LL+1) + DE(I)*
213:      &      (CX(LL)-CX(LL+1))
214:      SMCWP(I,LMIC(3)) = CXPL(LL+1) + DEP(I)*
215:      &      (CXPL(LL)-CXPL(LL+1))
216:      *****      ELSE
217:      *****      SMIC(IP1(I),1,LMIC(3)) = 0.0
218:      end if
219:      160      continue
220:      end if
221:
222: C-----
223: C      CAPTURE (MT = 101 )
224: C-----
225:
226: CCC      MT      = 101
227:
228:      if ( KLIB2(N,MT101,1).ne.0 ) then
229:      L0      = KLIB2(N,MT101,11) - KLIB2(N,MT101,3)
230: *VOCL LOOP,NOVREC
231: do 170 I = 1, NN
232:      if ( IEP(I).ge.KLIB2(N,MT101,3)
233:      &      .and.IEP(I).lt.KLIB2(N,MT101,4) ) then
234:      LL      = L0 + IEP(I)
235: CCCCC      SMIC(IP1(I),1,LMIC(5)) = CX(LL+1) + DE(I)*
236:      SMCW(I,LMIC(5)) = CX(LL+1) + DE(I)*
237:      &      (CX(LL)-CX(LL+1))
238:      SMCWP(I,LMIC(5)) = CXPL(LL+1) + DEP(I)*
239:      &      (CXPL(LL)-CXPL(LL+1))
240:      *****      ELSE
241:      *****      SMIC(IP1(I),1,LMIC(5)) = 0.0
242:      end if
243:      170      continue
244:      end if
245: C
246: C-----
247: C      UNRESOLVED RESONANCE (TOTAL , ELASTIC, FISSION )
248: C-----
249: C
250:      NUNR      = KLIB1(N,4)
251:      if ( NUNR.ne.0 ) then
252:      NUNR2      = KLIB1(N,5)
253:      LSUNR      = KLIB1(N,18)
254:      LLSSF      = KLIB1(N,30)
255:
256: C      ..... SELECT PARTICLES IN UNRESOLVED RESONANCE ENERGY RANGE ....
257:
258:      MM      = 0
259:      EL      = CX(LSUNR+NUNR-1)
260: *VOCL LOOP,NOVREC

```

src/mvp/gmicpt.f

```

261:      do 180 I = 1, NN
262:        if ( E(I).le.CX(LSUNR).and.E(I).ge.EL ) then
263:          MM = MM + 1
264:          CCCCC IBP2(MM) = IBP(I)
265:          IBP2(MM) = I
266:          E2(MM) = E(I)
267:        end if
268:      180 continue
269: C
270: C ..... DETERMINE TABLE POSITION
271: C
272: C      if ( MM.gt.0 ) then
273: C        call BSVDEC( CX(LSUNR), NUNR, E2, IEP2, MM )
274: C
275: C ..... SELECT TABLE .....
276: C
277: C      call RANU2( IRAND, R, MM, ICON )
278: C      LINT = LSUNR + NUNR
279: C
280: C      do 190 I = 1, MM
281: C        EE2 = CX(LSUNR+IEP2(I))
282: C        EE1 = CX(LSUNR+IEP2(I)-1)
283: C        INTUN = ICX(LINT+IEP2(I))
284: C        if ( INTUN.eq.5 .or. INTUN.eq.3 ) then
285: C          EE2 = LOG(EE2)
286: C          EE1 = LOG(EE1)
287: C          E2(I) = LOG(EE2(I))
288: C        end if
289: C        if ( R(I).gt.(E2(I)-EE2)/(EE1-EE2) )
290: C          & IEP2(I) = IEP2(I) + 1
291: C      190 continue
292: C
293: C      call RANU2( IRAND, R, 2*MM, ICON )
294: C
295: C      LS = LSUNR + 2*NUNR
296: C      CM1995-2-15 NNEG = 0
297: C
298: C      CM
299: C
300: C ..... CASE OF LLSSF=0 (BACKGROUND IS GIVEN) .....
301: C
302: C      if ( LLSSF.eq.0 ) then
303: C        *VOCL LOOP,NOVREC
304: C        do 200 I = 1, MM
305: C          LQ = LS + (IEP2(I)-1)*6*NUNR2
306: C          LIPOS = LQ + NUNR2
307: C
308: C          ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
309: C
310: C          C/#IF ROUNDOFF(NEAREST)
311: C          KK = MIN(NUNR2-1,INT(NUNR2*R(I)))
312: C          K = min(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
313: C          C/#ELSE
314: C          KK = NUNR2*R(I)
315: C          K = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
316: C          C/#ENDIF
317: C          K = ICX(LIPOS+K)
318: C
319: C          LCRX = LIPOS + 2*NUNR2 - 1
320: C
321: C          LCRX = LIPOS + 2*NUNR2 + 3*(K-1)
322: C
323: C          ..... TOTAL .....
324: C
325: C          CCCCC SMIC(IBP2(I),N,LMIC(1)) = SMIC(IBP2(I),N,LMIC(1))

```

```

326:      SMCW(IBP2(I),LMIC(1)) = SMCW(IBP2(I),LMIC(1))
327:      & + CX(LCRX)
328:      SMCWP(IBP2(I),LMIC(1)) = SMCWP(IBP2(I),LMIC(1))
329:      & + CXPl(LCRX)
330:
331: C ..... ELASTIC .....
332:
333: CCCCC SMIC(IBP2(I),N,LMIC(4)) = SMIC(IBP2(I),N,LMIC(4))
334: SMCW(IBP2(I),LMIC(4)) = SMCW(IBP2(I),LMIC(4))
335: & + CX(LCRX+1)
336: SMCWP(IBP2(I),LMIC(4)) = SMCWP(IBP2(I),LMIC(4))
337: & + CXPl(LCRX+1)
338:
339: C ..... FISSION .....
340:
341: CCCCC SMIC(IBP2(I),N,LMIC(3)) = SMIC(IBP2(I),N,LMIC(3))
342: SMCW(IBP2(I),LMIC(3)) = SMCW(IBP2(I),LMIC(3))
343: & + CX(LCRX+2)
344: SMCWP(IBP2(I),LMIC(3)) = SMCWP(IBP2(I),LMIC(3))
345: & + CXPl(LCRX+2)
346:
347: C ..... CAPTURE .....
348:
349: CCCCC SMIC(IBP2(I),N,LMIC(5)) = SMIC(IBP2(I),N,LMIC(5))
350: SMCW(IBP2(I),LMIC(5)) = SMCW(IBP2(I),LMIC(5))
351: & + CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
352: SMCWP(IBP2(I),LMIC(5)) = SMCWP(IBP2(I),LMIC(5))
353: & + CXPl(LCRX) - CXPl(LCRX+1) - CXPl(LCRX+2)
354: C
355: C .... check non-positive cross section ...
356: C
357: C      if ( SMIC(IBP2(I),N,LMIC(1)).le.0.0
358: C        .or. SMIC(IBP2(I),N,LMIC(4)).lt.0.0
359: C        .or. SMIC(IBP2(I),N,LMIC(3)).lt.0.0
360: C        .or. SMIC(IBP2(I),N,LMIC(5)).lt.0.0 ) then
361: C        if ( SMCW(IBP2(I),LMIC(1)).le.0.0
362: C          .or. SMCW(IBP2(I),LMIC(4)).lt.0.0
363: C          .or. SMCW(IBP2(I),LMIC(3)).lt.0.0
364: C          .or. SMCW(IBP2(I),LMIC(5)).lt.0.0 ) then
365: C          NNEG = NNEG + 1
366: C          CX0 = CX(LCRX)
367: C          CX1 = CX(LCRX+1)
368: C          CX2 = CX(LCRX+2)
369: C          CCCCC SMIC(IBP2(I),N,LMIC(1)) = 1.0E-5
370: C          CCCCC SMIC(IBP2(I),N,LMIC(4)) = 1.0E-5*CX1/CX0
371: C          CCCCC SMIC(IBP2(I),N,LMIC(3)) = 1.0E-5*CX2/CX0
372: C          CCCCC SMIC(IBP2(I),N,LMIC(5)) = 1.0E-5*(CX0-CX1-CX2)
373: C          CCCCC& /CX0
374: C          SMCW(IBP2(I),LMIC(1)) = 1.0E-5
375: C          SMCW(IBP2(I),LMIC(4)) = 1.0E-5*CX1/CX0
376: C          SMCW(IBP2(I),LMIC(3)) = 1.0E-5*CX2/CX0
377: C          SMCW(IBP2(I),LMIC(5)) = 1.0E-5*(CX0-CX1-CX2)
378: C          & /CX0
379: C        end if
380: C      200 continue
381: C
382: C ..... CASE OF LLSSF=1 (AVERAGED IS GIVEN) .....
383: C
384: C      else
385: C
386: C      *VOCL LOOP,NOVREC
387: C
388: C      do 205 I = 1, MM
389: C        LQ = LS + (IEP2(I)-1)*(6*NUNR2+3)
390: C        LIPOS = LQ + NUNR2
391: C        LSAV = LQ + 6*NUNR2

```


src/mvp/gmicpt.f

```

391: C
392: C ..... B.M.C. SAMPLING OF PROBABILITY TABLE .....
393: C
394: C/#IF ROUNDOFF(NEAREST)
395:         KK      = MIN(NUNR2-1,INT(NUNR2*R(I)))
396:         K        = min(1,INT(1.0-CX(LQ+KK)+R(MM+I))) + 2*KK
397: C/#ELSE
398:         KK      = NUNR2*R(I)
399:         K        = INT(1.0-CX(LQ+KK)+R(MM+I)) + 2*KK
400: C/#ENDIF
401:         K        = ICX(LIPOS+K)
402:
403:         LCRX     = LIPOS + 2*NUNR2 + 3*(K-1)
404:
405:         SIGC     = CX(LCRX) - CX(LCRX+1) - CX(LCRX+2)
406:         SIGCAV   = CX(LSAV) - CX(LSAV+1) - CX(LSAV+2)
407:         SIGCP    = CXP1(LCRX)-CXP1(LCRX+1)-CXP1(LCRX+2)
408:         SIGCVP   = CXP1(LSAV)-CXP1(LSAV+1)-CXP1(LSAV+2)
409: C
410: C .... check non-positive capture cross section ...
411: C     if it or averaged one is non-positive,
412: C     infinite dilution cross sections are used.
413: C
414:         if( SIGC.le.0.0.or. SIGCAV.le.0.0 ) then
415:             NNEG  = NNEG + 1
416:         else
417:
418: C ..... TOTAL .....
419:
420: CC      SMIC(IBP2(I),N,LMIC(1)) =
421: CC &      SMIC(IBP2(I),N,LMIC(1))/CX(LSAV) * CX(LCRX)
422:         SMCW(IBP2(I),LMIC(1)) =
423: &      SMCW(IBP2(I),LMIC(1))/CX(LSAV) * CX(LCRX)
424: *      SMCWP(IBP2(I),LMIC(1)) =
425: * &      SMCWP(IBP2(I),LMIC(1))/CXP1(LSAV) * CXP1(LCRX)
426:
427: C ..... ELASTIC .....
428:
429: CC      SMIC(IBP2(I),N,LMIC(4)) =
430: CC &      SMIC(IBP2(I),N,LMIC(4))/CX(LSAV+1)*CX(LCRX+1)
431:         SMCW(IBP2(I),LMIC(4)) =
432: &      SMCW(IBP2(I),LMIC(4))/CX(LSAV+1)*CX(LCRX+1)
433: *      SMCWP(IBP2(I),LMIC(4)) =
434: * &      SMCWP(IBP2(I),LMIC(4))/CXP1(LSAV+1)*CXP1(LCRX+1)
435:
436: C ..... FISSION .....
437:
438: CC      SMIC(IBP2(I),N,LMIC(3)) =
439: CC &      SMIC(IBP2(I),N,LMIC(3))/CX(LSAV+2)*CX(LCRX+2)
440:         SMCW(IBP2(I),LMIC(3)) =
441: &      SMCW(IBP2(I),LMIC(3))/CX(LSAV+2)*CX(LCRX+2)
442: *      SMCWP(IBP2(I),LMIC(3)) =
443: * &      SMCWP(IBP2(I),LMIC(3))/CXP1(LSAV+2)*CXP1(LCRX+2)
444:
445: C ..... CAPTURE .....
446:
447: CC      SMIC(IBP2(I),N,LMIC(5)) =
448: CC &      SMIC(IBP2(I),N,LMIC(5)) / SIGCAV * SIGC
449:         SMCW(IBP2(I),LMIC(5)) =
450: &      SMCW(IBP2(I),LMIC(5)) / SIGCAV * SIGC
451: *      SMCWP(IBP2(I),LMIC(5)) =
452: * &      SMCWP(IBP2(I),LMIC(5)) / SIGCVP * SIGCP
453:
454:         end if
455: 205      continue

```

```

456:         end if
457: C
458: C .... warning message for non-positive cross section fixup...
459: C
460:         if ( NNEG.gt.0 ) then
461:             if ( MNEG.lt.MWARN ) then
462:                 write(IOW,7050) NNEG, N
463: 7050      format(' !!! FOR',I5,' PARTICLES, CROSS SECTION',
464: &              'S ARE NEGATIVE IN UNRESOLVED RESONANCE REGION'/
465: &              '          FOR ',I4,'-TH NUCLIDE')
466:                 MNEG  = MNEG + NNEG
467:
468:             if ( MNEG.ge.MWARN ) then
469:                 write(IOW,*)
470: &              ' !!! NEGATIVE X-SEC MESSAGES ',
471: &              'WILL BE SUPPRESSED HEREAFTER.'
472:             end if
473:         end if
474:     end if
475: end if
476: end if
477: C-----
478: C      NU * FISSION
479: C-----
480:
481:         LSNU  = KLIB1(N,19)
482: C
483: C ..... COEFFICIENTS .....
484:
485:         if ( LNU.eq.1 ) then
486:             do 210 I = 1, NN
487:                 E2(I)  = CX(LSNU+NNU-1)
488:                 E2P(I) = CXP1(LSNU+NNU-1)
489: 210      continue
490:             do 230 K = NNU - 2, 0, -1
491:                 do 220 I = 1, NN
492:                     E2(I)  = E2(I)*E(I) + CX(LSNU+K)
493:                     E2P(I) = E2P(I)*E(I) + CXP1(LSNU+K)
494:                 continue
495:             230 continue
496: *VOCL LOOP,NOVREC
497:             do 240 I = 1, NN
498: CCC      SMIC(IP1(I),1,LMIC(2)) = E2(I)*SMIC(IP1(I),1,LMIC(3))
499:           SMCW(I,LMIC(2))  = E2(I)*SMCW(I,LMIC(3))
500:           SMCWP(I,LMIC(2)) = E2P(I)*SMCWP(I,LMIC(3))
501: 240      continue
502: C
503: C ..... TABULATION .....
504:
505:         else if ( LNU.eq.2 ) then
506:             call BSVDEC( CX(LSNU), NNU, E, IEP2, NN )
507: *VOCL LOOP,NOVREC
508:             do 250 I = 1, NN
509:                 LE  = LSNU + IEP2(I)
510:                 LL  = LE + NNU
511:                 D0  = (CX(LE)-E(I)) / (CX(LE)-CX(LE-1))
512:                 D0P = (CXP1(LE)-E(I)) / (CXP1(LE)-CXP1(LE-1))
513: CCC      SMIC(IP1(I),1,LMIC(2)) = (CX(LL)+D0*(CX(LL-1)-CX(LL)))*
514: CCC &      SMIC(IP1(I),1,LMIC(3))
515:           SMCW(I,LMIC(2)) = (CX(LL)+D0*(CX(LL-1)-CX(LL)))*
516:           SMCW(I,LMIC(3))
517:           SMCWP(I,LMIC(2)) = (CXP1(LL)+D0P*(CXP1(LL-1)-CXP1(LL)))*
518:           SMCWP(I,LMIC(3))
519:         &
520: 250      continue
521:     end if

```

src/mvp/gmicpt.f

```

521:
522: C-----
523: C   (N,2N)   (MT = 94 )
524: C-----
525:
526:       if ( LMIC(7).ne.0 ) then
527: Cccc      MT      = 94
528:       if ( KLIB2(N,MT94,1).ne.0 ) then
529: Ccccc      CALL    CXSINT(NN,IBP, N,MT94,
530: CccccX        SMIC(1,N,LMIC(7)), CX, MCX, KLIB2,NUC,NMT,
531: CccccW        IEP ,DE, DE1, NBANK)
532:       L0      = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)
533: *VOCL LOOP,NOVREC
534:       do 260 I = 1, NN
535:         if ( IEP(I).ge.KLIB2(N,MT94,3)
536:           & .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
537:           LL      = L0 + IEP(I)
538: CCCCC      SMIC(IP1(I),1,LMIC(7)) = CX(LL+1) + DE(I)*
539:           SMCW(I,LMIC(7)) = CX(LL+1) + DE(I)*
540:           & (CX(LL)-CX(LL+1))
541:           SMCWP(I,LMIC(7)) = CXPI(LL+1) + DEP(I)*
542:           & (CXPI(LL)-CXPI(LL+1))
543: C*****      ELSE
544: C*****      SMIC(IP1(I),1,LMIC(7)) = 0.0
545:           end if
546:       260      continue
547:           end if
548:       end if
549: C
550: C-----
551: C   LOSS : CAPTURE + FISSION - (N,2N) - 2*(N,3N) - 3*(N,4N)
552: C-----
553: C
554: C       IF(JEIGN.NE.0) THEN
555: C
556: C       if ( LMIC(8).ne.0 ) then
557: C
558: C       ..... (N,2N) HAS BEEN CALCULATED (NSMIC >= 7) .....
559: C
560: C       if ( LMIC(7).ne.0 ) then
561: *VOCL LOOP,NOVREC
562:       do 270 I = 1, NN
563: CCC        R(I)      = SMIC(IP1(I),1,LMIC(5))
564: CCC &          + SMIC(IP1(I),1,LMIC(3))
565: CCC &          - SMIC(IP1(I),1,LMIC(7))
566:       R(I)      = SMCW(I,LMIC(5))
567:       &          + SMCW(I,LMIC(3))
568:       &          - SMCW(I,LMIC(7))
569:       RP(I)     = SMCWP(I,LMIC(5))
570:       &          + SMCWP(I,LMIC(3))
571:       &          - SMCWP(I,LMIC(7))
572: 270      continue
573:       else
574: C
575: C       ..... (N,2N) IS CALCULATED HERE (NSMIC < 7) ....
576: C
577: Cccc      MT      = 94
578: C
579:       if ( KLIB2(N,MT94,1).ne.0 ) then
580: C
581: C          CALL    CXSINT(NN,IBP, N,MT94,
582: C X              R              , CX, MCX, KLIB2,NUC,NMT,
583: C W              IEP ,DE, DE1, NBANK)
584: C
585:       L0      = KLIB2(N,MT94,11) - KLIB2(N,MT94,3)

```

```

586: C
587: *VOCL LOOP,NOVREC
588:       do 280 I = 1, NN
589:         if ( IEP(I).ge.KLIB2(N,MT94,3)
590:           & .and.IEP(I).lt.KLIB2(N,MT94,4) ) then
591:           LL      = L0 + IEP(I)
592:           E2(I)   = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
593:           E2P(I)  = CXPI(LL+1)+DEP(I)*(CXPI(LL)-CXPI(LL+1))
594:         else
595:           E2(I)   = 0.0
596:           E2P(I)  = 0.0
597:         end if
598:
599: CCC        R(I)      = SMIC(IP1(I),1,LMIC(5))
600: CCC &          + SMIC(IP1(I),1,LMIC(3)) - E2(I)
601:       R(I)      = SMCW(I,LMIC(5))
602:       &          + SMCW(I,LMIC(3)) - E2(I)
603:       RP(I)     = SMCWP(I,LMIC(5))
604:       &          + SMCWP(I,LMIC(3)) - E2P(I)
605: 280      continue
606:       else
607: *VOCL LOOP,NOVREC
608:       do 290 I = 1, NN
609: CCC        R(I)      = SMIC(IP1(I),1,LMIC(5))
610: CCC &          + SMIC(IP1(I),1,LMIC(3))
611:       R(I)      = SMCW(I,LMIC(5))
612:       &          + SMCW(I,LMIC(3))
613:       RP(I)     = SMCWP(I,LMIC(5))
614:       &          + SMCWP(I,LMIC(3))
615: 290      continue
616:       end if
617:       end if
618: C
619: C-----
620: C   (N,3N) , (N,4N)
621: C-----
622: C
623: CCCC      MT      = 95
624:       if ( KLIB2(N,MT95,1).ne.0 ) then
625: C
626: Ccccc      CALL    CXSINT(NN,IBP, N,MT95,
627: CccccX        R              , CX, MCX, KLIB2,NUC,NMT,
628: CccccW        IEP ,DE, DE1, NBANK)
629: C
630:       L0      = KLIB2(N,MT95,11) - KLIB2(N,MT95,3)
631: *VOCL LOOP,NOVREC
632:       do 300 I = 1, NN
633:         if ( IEP(I).ge.KLIB2(N,MT95,3)
634:           & .and.IEP(I).lt.KLIB2(N,MT95,4) ) then
635:           LL      = L0 + IEP(I)
636:           E2(I)   = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
637:           E2P(I)  = CXPI(LL+1)+DEP(I)*(CXPI(LL)-CXPI(LL+1))
638:         else
639:           E2(I)   = 0.0
640:           E2P(I)  = 0.0
641:         end if
642:
643: CCC        R(I)      = R(I) - 2.0*E2(I)
644: CCC        RP(I)     = RP(I) - 2.0*E2P(I)
645: 300      continue
646:       end if
647: C
648: CCC      MT      = 96
649: C
650:       if ( KLIB2(N,MT96,1).ne.0 ) then

```

src/mvp/gmicpt.f

```

651:
652: Ccccc      CALL   CXSINT(NN,IBP, N,MT96,
653: CccccX      E2      , CX, MCX, KLIB2,NUC,NMT,
654: CccccW      IEP ,DE, DE1, NBANK)
655:
656:          L0      = KLIB2(N,MT96,11) - KLIB2(N,MT96,3)
657: *VOCL LOOP,NOVREC
658:      do 310 I = 1, NN
659:          if ( IEP(I).ge.KLIB2(N,MT96,3)
660:      &      .and.IEP(I).lt.KLIB2(N,MT96,4) ) then
661:              LL      = L0 + IEP(I)
662:              E2(I)    = CX(LL+1) + DE(I)*(CX(LL)-CX(LL+1))
663:              E2P(I)   = CXP1(LL+1)+DEP(I)*(CXP1(LL)-CXP1(LL+1))
664:          else
665:              E2(I)    = 0.0
666:              E2P(I)   = 0.0
667:          end if
668:              R(I)     = R(I) - 3.0*E2(I)
669:              RP(I)    = RP(I) - 3.0*E2P(I)
670:      310      continue
671:      end if
672: C
673: C-----
674: C      COMPLETE CALCULATION OF SIGMA LOSS
675: C-----
676: C
677: *VOCL LOOP,NOVREC
678:      do 320 I = 1, NN
679:      CCC      SMIC(IP1(I),1,LMIC(8)) = R(I)
680:      SMCW(I,LMIC(8)) = R(I)
681:      SMCWP(I,LMIC(8)) = RP(I)
682:      320      continue
683:      end if
684: C
685: C      ... save SMCW in bank array ...
686: C
687:      do 322 K=1,NSMIC
688:          do 324 I=1,NN
689:              SMIC(IBP(I),N,K) = SMCW(I,K)
690:              SMICP(IBP(I),N,K) = SMCWP(I,K)
691:          324      continue
692:          322      continue
693: C
694:      330 continue
695: C
696: C-----
697: C ... point-wise response : SGTAL
698: C      Energy in CRES is from low to high
699: C-----
700: C
701:      do 500 N = 1, NSTAL
702:          if ( KCRES(N,4).ne.0 ) then
703:      *VOCL LOOP,NOVREC
704:          do 510 I = 1, NN
705:              SGTAL(IBP(I),N) = 0.0
706:          510      continue
707:          if ( KCRES(N,3).eq.1 ) then
708:              NPTS      = KCRES(N,2)
709:              LEMESH     = KCRES(N,1)
710:              LEM        = LEMESH + NPTS - 1
711:              call BSVINC( CRES(LEMESH), NPTS, E, IEP, NN )
712: C
713:      *VOCL LOOP,NOVREC
714:          do 520 I = 1, NN
715:              if ( E(I).ge.CRES(LEMESH) .and.
716:      &          E(I).le.CRES(LEM) ) then
717:                  LE      = LEMESH + IEP(I)
718:                  LL      = LE + NPTS
719:                  SGTAL(IBP(I),N) = CRES(LL) +
720:                      (CRES(LE)-E(I)) / (CRES(LE)-CRES(LE-1))
721:                  &          *(CRES(LL-1)-CRES(LL))
722:              endif
723:          520      continue
724:          endif
725:      endif
726:      500 continue
727: C
728:      return
729:      end

```

src/mvp/gtdate.c

```
1: /*****
2:  * Output date and time of compilation on standard output
3:  * using ANSI C predefined macros if possible.
4:  *
5:  * 11 Jan 2001: written by Y.Nagaya
6:  *****/
7:
8: #include <stdio.h>
9: #include <string.h>
10:
11: #ifdef CRAY_C
12: #include <fortran.h>
13: #endif
14:
15: void get_cdate( char *cdate, int *cdatelen );
16: void get_ctime( char *ctime, int *ctimelen );
17:
18: void gtdate(cdate, cdatelen, ctime, ctimelen)
19:     char *cdate ;
20:     int *cdatelen ;
21:     char *ctime ;
22:     int *ctimelen ;
23: {
24:     get_cdate( cdate, cdatelen ) ;
25:     get_ctime( ctime, ctimelen ) ;
26: }
27: void gtdate_(cdate, cdatelen, ctime, ctimelen)
28:     char *cdate ;
29:     int *cdatelen ;
30:     char *ctime ;
31:     int *ctimelen ;
32: {
33:     get_cdate( cdate, cdatelen ) ;
34:     get_ctime( ctime, ctimelen ) ;
35: }
36:
37: #ifdef CRAY_C
38: void GTDATE(cdate, cdatelen, ctime, ctimelen)
39:     _fcd cdate ;
40:     int *cdatelen ;
41:     _fcd ctime ;
42:     int *ctimelen ;
43: {
44:     get_cdate( _fcdtscp(cdate), cdatelen ) ;
45:     get_ctime( _fcdtscp(ctime), ctimelen ) ;
46: }
47: #else
48: void GTDATE(cdate, cdatelen, ctime, ctimelen)
49:     char *cdate ;
50:     int *cdatelen ;
51:     char *ctime ;
52:     int *ctimelen ;
53: {
54:     get_cdate( cdate, cdatelen ) ;
55:     get_ctime( ctime, ctimelen ) ;
56: }
57: #endif
58:
59: void _GTDATE(cdate, cdatelen, ctime, ctimelen)
60:     char *cdate ;
61:     int *cdatelen ;
62:     char *ctime ;
63:     int *ctimelen ;
64: {
65:     get_cdate( cdate, cdatelen ) ;
```

```
66:     get_ctime( ctime, ctimelen ) ;
67: }
68:
69: void get_cdate( cdate, cdatelen )
70:     char *cdate ;
71:     int *cdatelen ;
72: {
73:     char *vp ;
74:     int i, nl ;
75:
76:     vp = "UNKNOWN" ;
77: #ifdef __DATE__
78:     vp = __DATE__ ;
79: #endif
80:
81:     nl = strlen(vp) ;
82:     if( nl <= *cdatelen )
83:     {
84:         memcpy( cdate, vp, nl ) ;
85:         for( i=nl; i< *cdatelen; ++i ) cdate[i] = ' ' ;
86:     }
87:     else
88:         memcpy( cdate, vp, *cdatelen ) ;
89: }
90:
91: void get_ctime( ctime, ctimelen )
92:     char *ctime ;
93:     int *ctimelen ;
94: {
95:     char *vp ;
96:     int i, nl ;
97:
98:     vp = "UNKNOWN" ;
99: #ifdef __TIME__
100:     vp = __TIME__ ;
101: #endif
102:
103:     nl = strlen(vp) ;
104:     if( nl <= *ctimelen )
105:     {
106:         memcpy( ctime, vp, nl ) ;
107:         for( i=nl; i< *ctimelen; ++i ) ctime[i] = ' ' ;
108:     }
109:     else
110:         memcpy( ctime, vp, *ctimelen ) ;
111: }
```

src/mvp/gtmacp.f

```

1:      subroutine GTMACP( IRAND, NN,      IBP,  MAT,  JAMXC, JDEBG,
2:      &                  NBANK, EEE,  MB,   NUC,   NSTAL, NSMAC, SMAC,
3:      &                  LMAC,  KMAC,  MMAC,  NSMIC, SMIC,  KSPI,
4:      &                  LPDNP, IATMT, DNSTP, NMAT, NPATOM, NMTP,  CXP,
5:      &                  MCXP,  KLBPl, KLBp2, XLBP2, IF3,  IFI,  R,
6:      &                  WK4,  WK5,  WK6,   STOT,  LMIC)
7: C=<MVP>=====
8: C  PURPOSE      : TO GET OR PREPARE MACRO CROSS SECTION FOR PHOTONS.
9: C                (CALCULATE TOTAL CROSS SECTION ONLY )
10: C
11: C  CALLED IN : FLIONE, FLIGHT
12: C  CALLS      : NONE
13: C=====
14:      integer JDEBG(*)
15: C
16: C .... NN : PARTICLE NUMBER  IBP : BANK POINTERS  MAT: MAT #
17: C
18:      integer NN, IBP(NBANK), MAT
19: C
20: C .... MATERIAL DATA .....
21: C
22:      integer LPDNP(NMAT+1), IATMT(*)
23:      real DNSTP(*)
24: C
25: C .... Cross section .....
26: C
27:      real CXP(MCXP)
28:      integer KLBPl(NPATOM,20), KLBp2(NPATOM,NMTP,6)
29:      real XLBP2(NPATOM,NMTP,2)
30: C
31: C .... BANK .....
32: C
33:      real EEE(NBANK)
34: C
35: C .... SIGMA BANK .....
36: C
37:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC)
38:      integer MB, KMAC(NBANK,MB,2), MMAC(NBANK,2), LMAC(8)
39:      integer KSPI(NBANK,NUC)
40:      integer LMIC(8)
41: C
42: C .... WORKING ARRAYS ...
43: C
44:      integer IF3(NBANK), IFI(NBANK)
45:      real R(NBANK), STOT(NBANK)
46:      real WK4(NBANK), WK5(NBANK), WK6(NBANK)
47: C
48:      include '../shared/INC/_IOUNIT'
49:      include 'INC/_IOUNIT2'
50: C
51: C=====
52: C  DETERMINE BANK NUMBER TO BE USED  ( MMAC(IBP(I),1) )
53: C=====
54: C
55:      N3      = 0
56:      do 100 I = 1, NN
57:        IF3(I) = 0
58:      100 continue
59: C-----
60: C  IF3 : STATUS INDICATOR      0 = TREATMENT UNDETERMINED YET
61: C                               1 = MACRO DATA MUST BE CALCULATED NEWLY
62: C                               -1 = MACRO DATA ALREADY PREPARED
63: C-----
64: C-----
65: C  COUNT THE NUMBER OF PARTICLES WHOSE DATA MUST BE CALCULATED NEWLY.

```

```

66: C  IFI : GATHERED BANK POINTERS
67: C-----
68:      N2      = 0
69: C
70:      do 120 K = 1, MB + 1
71:        if ( N3.eq.NN ) go to 130
72:      *VOCL LOOP,NOVREC
73:        do 110 I = 1, NN
74:          if ( IF3(I).eq.0 ) then
75: C-----
76: C      MACRO DATA ARE NOT IN BANK, MUST BE CALCULATED NEWLY!
77: C-----
78:          if ( MMAC(IBP(I),2).lt.K ) then
79:            IF3(I) = 1
80:            KMAC0 = MIN(MMAC(IBP(I),2)+1,MB)
81:            MMAC(IBP(I),2) = KMAC0
82:            MMAC(IBP(I),1) = KMAC0
83:            KMAC(IBP(I),KMAC0,2) = MAT
84:            N2      = N2 + 1
85:            IFI(N2) = IBP(I)
86: C-----
87: C      MACRO DATA FOR MATERIAL 'MAT' MAY EXIST IN BANK
88: C-----
89:          else
90:            if ( KMAC(IBP(I),K,2).eq.MAT ) then
91:              IF3(I) = -1
92:              MMAC(IBP(I),1) = K
93:            end if
94:          end if
95: C
96:          if ( IF3(I).ne.0 ) N3      = N3 + 1
97:          end if
98:          110 continue
99:          120 continue
100: C-----
101:          130 continue
102: C-----
103: C  N2 = 0 : NO NEED TO CALCULATE MACRO DATA !
104: C-----
105:      if ( N2.eq.0 ) return
106: C
107:      if ( JAMXC.ne.0 ) then
108:        do 150 N = LPDNP(MAT), LPDNP(MAT+1) - 1
109:          NI      = IATMT(N)
110:          NNN      = 0
111:          do 140 I = 1, N2
112:            if ( KSPI(IFI(I),NI).eq.0 ) then
113:              NNN      = NNN + 1
114:              IF3(NNN) = IFI(I)
115:              R(NNN)   = LOG(EEE(IFI(I)))
116:            end if
117:          140 continue
118:          if ( NNN.gt.0 ) then
119:            call GMICP1( NI, NNN, IF3, R, JGAMM, NUC, NMTP, NPATOM,
120:              &          NBANK, NSMIC, SMIC, KSPI, CXP, CXp, MCXP, KLBPl,
121:              &          KLBp2, XLBP2, WK4, WK5, WK6 )
122:          end if
123:          150 continue
124:        end if
125: C
126: C=====
127: C  CALCULATE MACRO DATA FOR SOME PARTICLES
128: C=====
129: C
130: C-----

```

src/mvp/gtmacp.f

```

131: C    CLEAR OR INITIALIZE TEMPORARY ARRAYS
132: C    (IF3 IS USED AS TEMPORARY STORAGE OF COLLIDING NUCLIDE NUMBER)
133: C-----
134:      NI      = IATMT(LPDNP(MAT))
135:      RHO     = DNSTP(LPDNP(MAT))
136:      do 160 I = 1, N2
137:        IF3(I) = NI
138:        STOT(I) = SMIC(IFI(I),NI,1)*RHO
139:      160 continue
140: C-----
141: C    CALCULATE MACRO CROSS SECTION BY SUMMATION OF MICRO CROSS SECTIONS
142: C-----
143:      if ( LPDNP(MAT+1)-LPDNP(MAT)).ge.2 ) then
144: C
145:        do 180 N = LPDNP(MAT) + 1, LPDNP(MAT+1) - 1
146:          call RANU2( IRAND, R, N2, ICON )
147:          NI      = IATMT(N)
148:          RHO     = DNSTP(N)
149:          do 170 I = 1, N2
150:            ST     = SMIC(IFI(I),NI,1)*RHO
151: C
152: C    ..... IF3 : ATOM TO COLLIDE .....
153: C
154:          STOT(I) = STOT(I) + ST
155:          if ( ST.gt.STOT(I)*R(I) ) IF3(I) = NI
156:      170    continue
157:      180    continue
158:    end if
159: C-----
160: C    PUT CALCULATED MACROSCOPIC DATA IN BANK
161: C-----
162: *VOCL LOOP,NOVREC
163:      do 190 I = 1, N2
164:        KMAC(IFI(I),MMAC(IFI(I),1),1) = IF3(I)
165: c##<2007/03/14:PN3:
166:        KMAC(IFI(I),MMAC(IFI(I),1),2) = MAT
167: c##>
168:        SMAC(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOT(I)
169:      190 continue
170: C-----
171: C    OTHER MACROSCOPIC CROSS SECTIONS IF NECESSARY
172: C-----
173:      do 420 K = 2, 5
174:        if(LMAC(K).gt.0) then
175:          do 430 I = 1, N2
176:            STOT(I) = 0.0
177:      430    continue
178:            do 440 N = LPDNP(MAT), LPDNP(MAT+1)-1
179:              NI = IATMT(N)
180:              RHO = DNSTP(N)
181:              do 450 I = 1, N2
182:                STOT(I) = STOT(I) + SMIC(IFI(I),NI,LMIC(K))*RHO
183:      450    continue
184:      440    continue
185: *VOCL LOOP,NOVREC
186:          do 460 I = 1, N2
187:            SMAC(IFI(I),MMAC(IFI(I),1),LMAC(K)) = STOT(I)
188:      460    continue
189:          end if
190:      420 continue
191: C-----
192: C    DATA CHECK ONLY FOR DEBUGGING (1990/3/15)
193: C-----
194:      if ( JDEBG(1).ne.0 ) then
195:        do 200 I = 1, N2

```

```

196:        if ( MMAC(IFI(I),1).lt.1
197:          &      .or. MMAC(IFI(I),1).gt.MMAC(IFI(I),2)
198:          &      .or. MMAC(IFI(I),2).lt.0 .or. MMAC(IFI(I),2).gt.MB ) then
199:          write(IOG,*) ' == INVALID MMAC DATA IN GTMACP ! == '
200:          write(IOG,*) '   POS = ', IFI(I), ' MMAC = ',
201:          &      MMAC(IFI(I),1), MMAC(IFI(I),2)
202:        end if
203:      200    continue
204:      do 220 M = 1, MB
205:        do 210 I = 1, N2
206:          if ( KMAC(IFI(I),M,1).lt.0 .or. KMAC(IFI(I),M,1).gt.NUC
207:            &      .or. KMAC(IFI(I),M,2).lt.0
208:            &      .or. KMAC(IFI(I),M,2).gt.NMAT ) then
209:            write(IOG,*) ' == INVALID KMAC DATA IN GTMACP ! == '
210:            write(IOG,*) '   POS = ', IFI(I), ' M= ', M,
211:            &      ' KMAC = ', KMAC(IFI(I),M,1),
212:            &      KMAC(IFI(I),M,2)
213:          end if
214:      210    continue
215:      220    continue
216:    end if
217: C
218:    return
219:  end

```

src/mvp/gtmacpn.f

```

1:      subroutine GTMACPN(IRAND, NN,      IBP,      MAT,      JAMXC, JDEBG,
2:      &      NBANK, EEE,      MB,      NUC,      NSTAL, NSMAC, SMAC,
3:      &      LMAC, KMAC, MMAC,      NSMIC, SMIC,      KSPI,
4:      &      LPDNP, IATMT, NATMT, DNSTP, NMAT,      NPATOM, NMTP,
5:      &      CXP,      MCXP, KLB1, KLB2, KLB3,
6:      &      NUCPN, NUCPNA, NUC_MAX,      NSMICPN,
7:      &      SMICPN, MNUCPN, IZTBN, LPIDPN, DNSTPN, EBOTLPN,
8:      &      CXPN,      MCXPN, KLBPN1, KLBPN2, KLBPN3, NUCPNI,
9:      &      NMTPN,
10:     &      RNU,      IF3,      IFI,      R,      WK4,      WK5,      WK6,
11:     &      STOT, SMCW )
12: C=<MVP>=====
13: C PURPOSE: to get or prepare macro cross section including photo-
14: C          nuclear for photons. (calculate total cross section only )
15: C CALLED IN: FLIONE
16: C CALLS: GMICP1, GMICPN1, RANU2
17: C
18: C=====
19: C
20:      include 'inc/shared/INC/_IOUNIT'
21:      include 'INC/_IOUNIT2'
22: C
23:      integer JDEBG(*)
24: C
25: C .... NN : PARTICLE NUMBER,      IBP : BANK POINTERS,      MAT : MAT # ....
26: C
27:      integer NN, IBP(NBANK), MAT
28: C
29: C .... MATERIAL DATA ...
30: C
31:      integer LPDNP(NMAT+1), IATMT(*), NATMT(NPATOM), MNUCPN(NMAT+1),
32:      &      IZTBN(NUCPNA), LPIDPN(NUCPNA)
33:      real DNSTP(*), DNSTPN(NUCPNA)
34: C
35: C .... Cross section ....
36: C
37:      real CXP(MCXP), KLB2(NPATOM, NMTP, 2)
38:      integer KLB1(NPATOM, 20), KLB2(NPATOM, NMTP, 6)
39:      real CXPN(MCXPN), KLBPN1(NUCPNI, 6), EBOTLPN
40:      integer KLBPN1(NUCPNI, 16), KLBPN2(NUCPNI, NMTPN, 13)
41: C
42: C .... BANK ....
43: C
44:      real EEE(NBANK)
45: C
46: C .... SIGMA BANK ....
47: C
48:      real SMAC(NBANK, MB, NSMAC), SMIC(NBANK, NUC, NSMIC),
49:      &      SMICPN(NBANK, NUCPN, NSMICPN)
50:      real RNU(NBANK, NUC_MAX, 3)
51:      integer MB, KMAC(NBANK, MB, 2), MMAC(NBANK, 2), LMAC(8)
52:      integer KSPI(NBANK, NUC)
53: C
54: C .... WORKING ARRAYS ....
55: C
56:      integer IF3(NBANK), IFI(NBANK)
57:      real R(NBANK), STOT(NBANK), WK4(NBANK), WK5(NBANK), WK6(NBANK)
58:      real SMCW(NBANK, NSMICPN)
59: C
60: C=====
61: C DETERMINE BANK NUMBER TO BE USED ( MMAC(IBP(I), 1) )
62: C=====
63: C
64:      N2      = 0
65:      N3      = 0

```

```

66:      do I = 1, NN
67:      IF3(I) = 0
68:      end do
69: C-----
70: C IF3 : STATUS INDICATOR      0 = TREATMENT UNDETERMINED YET
71: C      1 = MACRO DATA MUST BE CALCULATED NEWLY
72: C      -1 = MACRO DATA ALREADY PREPARED
73: C-----
74: C COUNT THE NUMBER OF PARTICLES WHOSE DATA MUST BE CALCULATED NEWLY.
75: C IFI : GATHERED BANK POINTERS
76: C-----
77:      do K = 1, MB + 1
78:      if ( N3.eq.NN ) go to 130
79: *VOCL LOOP, NOVREC
80:      do I = 1, NN
81:      if ( IF3(I).eq.0 ) then
82: C
83: C ..... MACRO DATA ARE NOT IN BANK, MUST BE CALCULATED NEWLY !
84:      if ( MMAC(IBP(I), 2).lt.K ) then
85:      IF3(I) = 1
86:      KMAC0 = MIN(MMAC(IBP(I), 2)+1, MB)
87:      MMAC(IBP(I), 2) = KMAC0
88:      MMAC(IBP(I), 1) = KMAC0
89:      KMAC(IBP(I), KMAC0, 2) = MAT
90:      N2      = N2 + 1
91:      IFI(N2) = IBP(I)
92: C
93: C ..... MACRO DATA FOR MATERIAL 'MAT' MAY EXIST IN BANK
94:      else
95:      if ( KMAC(IBP(I), K, 2).eq.MAT ) then
96:      IF3(I) = -1
97:      MMAC(IBP(I), 1) = K
98:      end if
99:      end if
100: C
101:      if ( IF3(I).ne.0 ) N3 = N3 + 1
102:      end if
103:      end do
104:      end do
105: C
106: 130 continue
107: C ..... N2 = 0 : NO NEED TO CALCULATE MACRO DATA !
108:      if ( N2.eq.0 ) return
109: C
110: C ..... Adaptive-Macro cross section
111:      if ( JAMXC.ne.0 ) then
112:      do N = LPDNP(MAT), LPDNP(MAT+1)-1
113:      NI      = IATMT(N)
114:      IZ      = NATMT(NI)
115:      NNN      = 0
116:      do I = 1, N2
117:      if ( KSPI(IFI(I), NI).eq.0 ) then
118:      NNN      = NNN + 1
119:      IF3(NNN) = IFI(I)
120:      R(NNN)   = log(EEE(IFI(I)))
121:      STOT(NNN) = EEE(IFI(I))
122:      end if
123:      end do
124:      if ( NNN.gt.0 ) then
125:      call GMICP1( NI, NNN, IF3, R, JGAMM, NUC, NMTP, NPATOM,
126:      &      NBANK, NSMIC, SMIC, KSPI, CXP, MCXP, KLB1,
127:      &      KLB2, KLB3, WK4, WK5, WK6 )
128:      do K = MNUCPN(MAT), MNUCPN(MAT+1)-1
129:      if ( IZTBN(K).eq.IZ ) then
130:      call GMICPN1( LPIDPN(K), NNN, IF3, STOT, NUC,

```

src/mvp/gtmacpn.f

```

131:      &          NUCPN, NUCPNI, NMTPN, NBANK, NUC_MAX,
132:      &          NSMICPN, SMICPN, CXPN, CXPN, MCXPN,
133:      &          KLBNP1, KLBNP2, XLBNP1, EBOTLPN, WK4, WK5,
134:      &          WK6, R, SMCW, RNU )
135:      &          end if
136:      &          end do
137:      &          end if
138:      &          end do
139:      &          end if
140: C
141: C-----
142: C  CALCULATE MACRO DATA FOR SOME PARTICLES
143: C-----
144: C-----
145: C  CLEAR OR INITIALIZE TEMPORARY ARRAYS
146: C  (IF3 IS USED AS TEMPORARY STORAGE OF COLLIDING NUCLIDE NUMBER)
147: C-----
148: C
149:      NI      = IATMT(LPDNP(MAT))
150:      RHO      = DNSTP(LPDNP(MAT))
151:      IZ      = NATMT(NI)
152:      do I = 1, N2
153:          IF3(I) = NI
154:          ST      = SMIC(IFI(I),NI,1) * RHO
155:          WK6(I) = ST
156:          do K = MNUCPN(MAT), MNUCPN(MAT+1)-1
157:              if ( IZTBN(K).eq.IZ )
158:              &      ST = ST + SMICPN(IFI(I),LPIDPN(K),1) * DNSTPN(K)
159:              end do
160:          STOT(I) = ST
161:      end do
162: C
163: C-----
164: C  CALCULATE MACRO CROSS SECTION BY SUMMATION OF MICRO CROSS SECTIONS
165: C-----
166: C
167:      if ( LPDNP(MAT+1)-LPDNP(MAT).ge.2 ) then
168:      do N = LPDNP(MAT)+1, LPDNP(MAT+1)-1
169:          call RANU2( IRAND, R, N2, ICON )
170:          NI      = IATMT(N)
171:          RHO      = DNSTP(N)
172:          IZ      = NATMT(NI)
173:          do I = 1, N2
174:              ST      = SMIC(IFI(I),NI,1) * RHO
175:              WK6(I) = WK6(I) + ST
176:              do K = MNUCPN(MAT), MNUCPN(MAT+1)-1
177:                  if ( IZTBN(K).eq.IZ )
178:                  &      ST = ST + SMICPN(IFI(I),LPIDPN(K),1) * DNSTPN(K)
179:                  end do
180:              STOT(I) = STOT(I) + ST
181:              if ( ST.gt.STOT(I)*R(I) ) IF3(I) = NI
182:          end do
183:      end do
184:      end if
185: C
186: C-----
187: C  PUT CALCULATED MACROSCOPIC DATA IN BANK
188: C-----
189: C
190: *VOCL LOOP,NOVREC
191:      do I = 1, N2
192:          KMAC(IFI(I),MMAC(IFI(I),1),1) = IF3(I)
193:          KMAC(IFI(I),MMAC(IFI(I),1),2) = MAT
194:          SMAC(IFI(I),MMAC(IFI(I),1),LMAC(1)) = STOT(I)
195:          SMAC(IFI(I),MMAC(IFI(I),1),LMAC(2)) = WK6(I)

```

```

196:      end do
197: C
198: C ..... DATA CHECK ONLY FOR DEBUGGING
199:      if ( JDEBG(1).ne.0 ) then
200:          do I = 1, N2
201:              if ( MMAC(IFI(I),1).lt.1.or.MMAC(IFI(I),1).gt.MMAC(IFI(I),2)
202:              &      .or.MMAC(IFI(I),2).lt.0.or.MMAC(IFI(I),2).gt.MB ) then
203:                  write(IOG,*) ' == INVALID MMAC DATA IN GTMACPN ! == '
204:                  write(IOG,*) '   POS = ', IFI(I), ' MMAC = ',
205:                  &      MMAC(IFI(I),1), MMAC(IFI(I),2)
206:              end if
207:          end do
208:          do M = 1, MB
209:              do I = 1, N2
210:                  if ( KMAC(IFI(I),M,1).lt.0.or.KMAC(IFI(I),M,1).gt.NUC.or.
211:                  &      KMAC(IFI(I),M,2).lt.0.or.KMAC(IFI(I),M,2).gt.NMAT )
212:                  &      then
213:                      write(IOG,*) ' == INVALID KMAC DATA IN GTMACPN ! == '
214:                      write(IOG,*) '   POS = ', IFI(I), ' M= ', M, ' KMAC = ',
215:                      &      KMAC(IFI(I),M,1), KMAC(IFI(I),M,2)
216:                  end if
217:              end do
218:          end do
219:      end if
220: C
221:      return
222:      end

```


src/mvp/gtmicp.f

```

1:      subroutine GTMICP( JGAMM, NN,      IBP,      NUC,      NMTP,      NPATOM,
2:      &                  NBANK, NSMIC, EEE,      SMIC,      KSPI,
3:      X                  CXP,      ICXP,      MCXP,      KLBP1, KLBP2,      XLBP2,
4:      W                  E,      IEP,      DE,      IP1 )
5:      C      X                  SGTAL, CRES, KCRES, NSTAL, MCRES,
6:      C=====
7:      C PURPOSE: GET MICRO CROSS SECTIONS AND PUT THEM IN SIGMA-BANK.
8:      C      FOR PHOTONS
9:      C CALLED IN:  PHOTR, SOURCE
10:     C=====
11:     C
12:     C .... NN: NUMBER OF PARTICLES      IBP: POINTER      ....
13:     C
14:     C      integer IBP(NN)
15:     C
16:     C .... BANK .....
17:     C
18:     C      real EEE(NBANK), SMIC(NBANK,NUC,NSMIC)
19:     C      real SGTAL(NBANK,NSTAL)
20:     C      integer KSPI(NBANK,NUC)
21:     C
22:     C .... CROSS SECTION & POINTERS .....
23:     C
24:     C      real CXP(MCXP)
25:     C      integer ICXP(MCXP), KLBP1(NPATOM,20), KLBP2(NPATOM,NMTP,6)
26:     C      real XLBP2(NPATOM,NMTP,2)
27:     C
28:     C      real CRES(MCRES)
29:     C      integer KCRES(NSTAL,4)
30:     C-----
31:     C      POSITIONING FOR REACTIONS
32:     C
33:     C      1 : TOTAL
34:     C      2 : COHERENT SCATTERING
35:     C      3 : INCOHERENT SCATTERING
36:     C      4 : PAIR PRODUCTION TOTAL
37:     C      5 : PHOTOELECTRIC
38:     C-----
39:     C
40:     C .... WORKING ARRAY .....
41:     C
42:     C      real E(NBANK), DE(NBANK)
43:     C      integer IEP(NBANK), IP1(NBANK)
44:     C
45:     C      parameter( MT1 = 1 )
46:     C      parameter( MT2 = 2 )
47:     C      parameter( MT3 = 4 )
48:     C      parameter( MT4 = 6 )
49:     C      parameter( MT5 = 8 )
50:     C
51:     C-----
52:     C      GATHER ENERGY ( -> LOG )
53:     C-----
54:     C
55:     C      do 100 I = 1, NN
56:     C          E(I) = LOG(EEE(IBP(I)))
57:     C      100 continue
58:     C
59:     C-----
60:     C--- LOOP FOR ATOM -----
61:     C-----
62:     C
63:     C      do 130 N = 1, NPATOM
64:     C          NPTS = KLBP1(N,2)
65:     C-----

```

```

66:     C      ENERGY MESH POINT #
67:     C-----
68:     C      LEMESH = KLBP1(N,1)
69:     C      call BSVDEC( CXP(LEMESH), NPTS, E, IEP, NN )
70:     C
71:     C      do 110 I = 1, NN
72:     C          KSPI(IBP(I),N) = IEP(I)
73:     C          LE = LEMESH + IEP(I)
74:     C          DE(I) = (CXP(LE)-E(I)) / (CXP(LE)-CXP(LE-1))
75:     C          cccccccccc IP1(I) = (N-1)*NBANK + IBP(I)
76:     C          110 continue
77:     C
78:     C-----
79:     C-----
80:     C
81:     C      ..... LSTF3 - ISTID .....
82:     C      ***** L1 = KLBP2(N,MT1,5) - KLBP2(N,MT1,3)
83:     C      L2 = KLBP2(N,MT2,5) - KLBP2(N,MT2,3)
84:     C      L3 = KLBP2(N,MT3,5) - KLBP2(N,MT3,3)
85:     C      L4 = KLBP2(N,MT4,5) - KLBP2(N,MT4,3)
86:     C      L5 = KLBP2(N,MT5,5) - KLBP2(N,MT5,3)
87:     C      ccc*VOCL LOOP,NOVREC(SMIC)
88:     C      *VOCL LOOP,NOVREC
89:     C      do 120 I = 1, NN
90:     C          SMIC(IBP(I),N,2) = 0.0
91:     C          SMIC(IBP(I),N,3) = 0.0
92:     C          SMIC(IBP(I),N,4) = 0.0
93:     C          SMIC(IBP(I),N,5) = 0.0
94:     C
95:     C      .... TOTAL .....
96:     C
97:     C      *      LL = L1 + IEP(I)
98:     C      *      SMIC(IBP(I),N,1) = EXP(
99:     C      *      @      CXP(LL+1)+DE(I)*(CXP(LL) - CXP(LL+1)) )
100:    C
101:    C      .... COHERENT SCATTERING .....
102:    C
103:    C      if ( JGAMM.eq.0 ) then
104:    C          if ( KLBP2(N,MT2,1).ne.0.and.IEP(I).ge.KLBP2(N,MT2,3)
105:    C          & .and.IEP(I).lt.KLBP2(N,MT2,4) ) then
106:    C
107:    C          LL2 = L2 + IEP(I)
108:    C          SMIC(IBP(I),N,2) =
109:    C          &      EXP(CXP(LL2+1)+DE(I)*(CXP(LL2)-CXP(LL2+1)))
110:    C          end if
111:    C      ***** ELSE
112:    C
113:    C      ... SET SIGMA-COHERENT TO ZERO FOR GAMMA-RAY MODEL ....
114:    C
115:    C      ***** SMIC(IBP(I),N,2) = 0.0
116:    C      end if
117:    C
118:    C      .... INCOHERENT SCATTERING .....
119:    C
120:    C      if ( KLBP2(N,MT3,1).ne.0.and.IEP(I).ge.KLBP2(N,MT3,3)
121:    C      & .and.IEP(I).lt.KLBP2(N,MT3,4) ) then
122:    C          LL3 = L3 + IEP(I)
123:    C          SMIC(IBP(I),N,3) =
124:    C          &      EXP(CXP(LL3+1)+DE(I)*(CXP(LL3)-CXP(LL3+1)))
125:    C          end if
126:    C
127:    C      .... PAIR PRODUCTION .....
128:    C      (HAVING THRESHOULD AT 1.022 MEV
129:    C
130:    CM      IF( E(I) .GE. LOG(1.022D6) .AND.

```

src/mvp/gtmicp.f

```
131: CM   @           KLBP2(N,MT4,1).NE.0 .AND. E(I).GE.XLBP2(N,MT4,1)          196:      return
132:      if ( KLBP2(N,MT4,1).ne.0.and.IEP(I).ge.KLBP2(N,MT4,3)          197:      end
133:      &           .and.IEP(I).lt.KLBP2(N,MT4,4) ) then
134: C
135:      LL4      = L4 + IEP(I)
136:      SMIC(IBP(I),N,4)      =
137:      &           EXP(CXP(LL4+1)+DE(I)*(CXP(LL4)-CXP(LL4+1)))
138:      end if
139: C
140: C   .... PHOTOELECTIC ABSORPTION .....
141: C
142: CM   IF( KLBP2(N,MT5,1).NE.0 .AND. E(I).GE.XLBP2(N,MT5,1)
143:      if ( KLBP2(N,MT5,1).ne.0.and.IEP(I).ge.KLBP2(N,MT5,3)
144:      &           .and.IEP(I).lt.KLBP2(N,MT5,4) ) then
145:      LL5      = L5 + IEP(I)
146:      SMIC(IBP(I),N,5)      =
147:      &           EXP(CXP(LL5+1)+DE(I)*(CXP(LL5)-CXP(LL5+1)))
148:      end if
149: C
150: C   .... TOTAL .....
151: C
152:      SMIC(IBP(I),N,1)      = SMIC(IBP(I),N,2) + SMIC(IBP(I),N,3)
153:      &           + SMIC(IBP(I),N,4) + SMIC(IBP(I),N,5)
154: C
155: 120      continue
156: 130      continue
157: C
158: C-----
159: C ... point-wise response : SGTAL
160: C   Energy in CRES is from low to high
161: C-----
162: C
163: C   if ( NSTAL.gt.0 ) then
164: C       do 140 I = 1, NN
165: C           E(I)      = EEE(IBP(I))
166: C 140      continue
167: C       do 170 N = 1, NSTAL
168: C           if ( KCRES(N,4).ne.0 ) then
169: C *VOCL LOOP,NOVREC
170: C           do 150 I = 1, NN
171: C               SGTAL(IBP(I),N) = 0.0
172: C           continue
173: C           if ( KCRES(N,3).eq.2 ) then
174: C               NPTS      = KCRES(N,2)
175: C               LEMESH     = KCRES(N,1)
176: C               LEM        = LEMESH + NPTS - 1
177: C               call BSVINC( CRES(LEMESH), NPTS, E, IEP, NN )
178: C
179: C *VOCL LOOP,NOVREC
180: C           do 160 I = 1, NN
181: C               if ( E(I).ge.CRES(LEMESH).and.E(I).le.CRES(LEM) )
182: C               &           then
183: C                   LE      = LEMESH + IEP(I)
184: C                   LL      = LE + NPTS
185: C                   SGTAL(IBP(I),N) = CRES(LL) + (CRES(LE)-E(I)) /
186: C                   &           (CRES(LE)-CRES(LE-1))*
187: C                   &           (CRES(LL-1)-CRES(LL))
188: C               end if
189: C 160      continue
190: C           end if
191: C       end if
192: C 170      continue
193: C
194: C   end if
195: C
```

src/mvp/gtmicpn.f

```

1:      subroutine GTMICPN(NN,   IBP,   NUC,   NUCPN, NUCPNI, NMTPN,
2:      &                      NBANK, NSMICPN,   NUC_MAX,   EEE,
3:      &                      SMICPN, CXPN,   ICXPN, MCXPN, KLBP1, KLBP2,
4:      &                      XLBP1, EBOTLPN, E,   E2,   IEP,   IEP2, DE,
5:      &                      SMCW,   RNU )
6: C=<MVP>=====
7: C  PURPOSE:  get micro cross sections and put then in sigma-bank.
8: C            (photo-nuclear)
9: C  CALLED IN:  NEUTR, PHOTR, SOURCE, PHOTNUC
10: C  CALLS:  BSVDEC
11: C=====
12: C
13: C  include '../shared/INC/_IOUNIT'
14: C
15: C  .... NN: NUMBER OF PARTICLES,   IBP: BANK POINTER .....
16: C
17: C  integer IBP(NN)
18: C
19: C  .... BANK .....
20: C
21: C  real EEE(NBANK), SMICPN(NBANK,NUCPN,NSMICPN)
22: C
23: C  .... RNU : nu-bar (1/2/3=total/delayed/prompt)
24: C
25: C  real RNU(NBANK,NUC_MAX,3)
26: C
27: C  .... CROSS SECTION & POINTERS .....
28: C
29: C  real CXPN(MCXPN), XLBP1(NUCPNI,6), EBOTLPN
30: C  integer ICXPN(MCXPN), KLBP1(NUCPNI,16), KLBP2(NUCPNI,NMTPN,13)
31: C
32: C  .... WORKING ARRAY .....
33: C
34: C  real E(NBANK), E2(NBANK), DE(NBANK)
35: C  integer IEP(NBANK), IEP2(NBANK)
36: C  real SMCW(NBANK,NSMICPN)
37: C
38: C  parameter( MT3 = 3, MT18 = 18 )
39: C
40: C-----
41: C  GATHER ENERGY
42: C-----
43: C
44: C  N2      = 0
45: C  do I = 1, NN
46: C    E(I)   = EEE(IBP(I))
47: C    if ( EEE(IBP(I)).lt.EBOTLPN ) N2 = N2 + 1
48: C  end do
49: C  if ( N2.eq.NN ) then
50: C    do K = 1, NSMICPN
51: C      do N = 1, NUCPN
52: C        do I = 1, NN
53: C          SMICPN(IBP(I),N,K) = 0
54: C        end do
55: C      end do
56: C    end do
57: C    go to 300
58: C  end if
59: C
60: C-----
61: C--- LOOP FOR NUCLIDES -----
62: C-----
63: C
64: C  do 200 N = 1, NUCPN
65: C    NPTS      = KLBP1(N,2)

```

```

66: C    LNU      = KLBP1(N,9)
67: C    NNU      = KLBP1(N,6)
68: C    LNUD     = KLBP1(N,12)
69: C    NNUD     = KLBP1(N,11)
70: C    N2       = LNU*NNU
71: C
72: C-----
73: C    ENERGY MESH POINT #
74: C-----
75: C
76: C    LEMESH   = KLBP1(N,1)
77: C    call BSVDEC( CXPN(LEMESH), NPTS, E, IEP, NN )
78: C
79: C  *VOCL LOOP,NOVREC
80: C    do I = 1, NN
81: C      if ( E(I).lt.XLBP1(N,3) ) then      ! energy is lower than thresho
82: C        IEP(I) = NPTS
83: C        DE(I)  = 0
84: C      else
85: C        LE     = LEMESH + IEP(I)
86: C        DE(I)  = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
87: C      end if
88: C    end do
89: C
90: C  .... CLEAR ALL CROSS SECTIONS .....
91: C    do K = 1, NSMICPN
92: C      do I = 1, NN
93: C        SMCW(I,K) = 0
94: C      end do
95: C    end do
96: C    do K = 1, 3
97: C      do I = 1, NN
98: C        RNU(IBP(I),N,K) = 0
99: C      end do
100: C    end do
101: C
102: C-----
103: C    NONELASTIC (MT = 3)
104: C-----
105: C
106: C    L1       = KLBP2(N,MT3,10) - KLBP2(N,MT3,3)
107: C  *VOCL LOOP,NOVREC
108: C    do I = 1, NN
109: C      if ( IEP(I).ge.KLBP2(N,MT3,3) .and.
110: C        & IEP(I).lt.KLBP2(N,MT3,4) ) then
111: C        LL    = L1 + IEP(I)
112: C        SMCW(I,1) = CXPN(LL+1) + DE(I)*(CXPN(LL)-CXPN(LL+1))
113: C      end if
114: C    end do
115: C
116: C-----
117: C    FISSION (MT = 18)
118: C-----
119: C
120: C    if ( KLBP2(N,MT18,1).ne.0 ) then
121: C      L0     = KLBP2(N,MT18,10) - KLBP2(N,MT18,3)
122: C  *VOCL LOOP,NOVREC
123: C    do I = 1, NN
124: C      if ( IEP(I).ge.KLBP2(N,MT18,3) .and.
125: C        & IEP(I).lt.KLBP2(N,MT18,4) ) then
126: C        LL    = L0 + IEP(I)
127: C        SMCW(I,2) = CXPN(LL+1) + DE(I)*(CXPN(LL)-CXPN(LL+1))
128: C      end if
129: C    end do

```

src/mvp/gtmicpn.f

```

130:         end if
131: C
132: C -----
133: C      NU * FISSION  (NU = nu-total)
134: C -----
135: C
136:         LSNU      = KLBP1(N,10)
137: C
138: C ..... COEFFICIENTS .....
139:         if ( LNU.eq.1 ) then
140:           do I = 1, NN
141:             E2(I)   = CXPN(LSNU+NNU-1)
142:           end do
143:           do K = NNU-2, 0, -1
144:             do I = 1, NN
145:               E2(I) = E2(I)*E(I) + CXPN(LSNU+K)
146:             end do
147:           end do
148:           do I = 1, NN
149:             RNU(IBP(I),N,1) = E2(I)
150:           end do
151: *VOCL LOOP,NOVREC
152:           do I = 1, NN
153:             SMCW(I,2) = E2(I) * SMCW(I,2)
154:           end do
155: C
156: C ..... TABULATION .....
157:         else if ( LNU.eq.2 ) then
158:           call BSVDEC( CXPN(LSNU), NNU, E, IEP2, NN )
159: *VOCL LOOP,NOVREC
160:           do I = 1, NN
161:             LE      = LSNU + IEP2(I)
162:             LL      = LE + NNU
163:             D0      = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
164:             SMCW(I,2) = (CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))) *
165:               & SMCW(I,2)
166:             RNU(IBP(I),N,1) = CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))
167:           end do
168:         end if
169: C
170: C -----
171: C      Nu-delayed
172: C -----
173: C
174:         LSNU      = KLBP1(N,14)
175: C
176: C ..... COEFFICIENTS .....
177:         if ( LNU.eq.1 ) then
178:           do I = 1, NN
179:             E2(I)   = CXPN(LSNU+NNUD-1)
180:           end do
181:           do K = NNUD-2, 0, -1
182:             do I = 1, NN
183:               E2(I) = E2(I)*E(I) + CXPN(LSNU+K)
184:             end do
185:           end do
186:           do I = 1, NN
187:             RNU(IBP(I),N,2) = E2(I)
188:             RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
189:           end do
190: C
191: C ..... TABULATION .....
192:         else if ( LNU.eq.2 ) then
193:           call BSVDEC( CXPN(LSNU), NNUD, E, IEP2, NN )
194: *VOCL LOOP,NOVREC

```

```

195:         do I = 1, NN
196:           LE      = LSNU + IEP2(I)
197:           LL      = LE + NNUD
198:           D0      = (CXPN(LE)-E(I)) / (CXPN(LE)-CXPN(LE-1))
199:           RNU(IBP(I),N,2) = CXPN(LL) + D0 * (CXPN(LL-1)-CXPN(LL))
200:           RNU(IBP(I),N,3) = RNU(IBP(I),N,1) - RNU(IBP(I),N,2)
201:         end do
202:       end if
203: C
204: C -----
205: C      save SMCW in bank array
206: C -----
207: C
208:         do K = 1, NSMICPN
209:           do I = 1, NN
210:             SMICPN(IBP(I),N,K) = SMCW(I,K)
211:           end do
212:         end do
213: C
214:         200 continue
215:         300 continue
216: C
217:         return
218:       end

```

src/mvp/gtsgtl.f

```

1:      subroutine GTSGTL( JNP,  NN,   IBP,  NBANK, EEE,  SGTL, CRES,
2:      &                  KCRES, NSTAL, MCRES, E,    IEP )
3: C=<MVP>=====
4: C PURPOSE: Get pointwise response SGTL.
5: C CALLED IN:  NEUTR,PHOTR, SOURCE
6: C JNP : 1/2=neutron/photon ( added in April 14, 2000)
7: C=====
8: C
9: C .... NN: NUMBER OF PARTICLES  IBP: POINTER .....
10: C
11:      integer IBP(NN)
12: C
13: C .... BANK .....
14: C
15:      real EEE(NBANK), SGTL(NBANK,NSTAL)
16: C
17: C .... CROSS SECTION & POINTERS .....
18: C
19:      real CRES(MCRES)
20:      integer KCRES(NSTAL,4)
21: C
22: C .... WORKING ARRAY .....
23: C
24:      real E(NBANK)
25:      integer IEP(NBANK)
26: C
27:      include '../shared/INC/_IOUNIT'
28: C
29: C-----
30: C
31:      if ( NSTAL.gt.0 ) then
32: *VOCL LOOP,NOVREC
33:          do 100 I = 1, NN
34:              E(I) = EEE(IBP(I))
35:          100 continue
36:          do 130 N = 1, NSTAL
37:              if ( KCRES(N,4).ne.0 ) then
38: *VOCL LOOP,NOVREC
39:                  do 110 I = 1, NN
40:                      SGTL(IBP(I),N) = 0.0
41:                  110 continue
42:                  if ( KCRES(N,3).eq.JNP ) then
43:                      NPTS = KCRES(N,2)
44:                      LEMESH = KCRES(N,1)
45:                      LEM = LEMESH + NPTS - 1
46: C
47: C ....      Energy in CRES is from low to high
48: C
49:                      call BSVINC( CRES(LEMESH), NPTS, E, IEP, NN )
50: *VOCL LOOP,NOVREC
51:                      do 120 I = 1, NN
52:                          if ( E(I).ge.CRES(LEMESH).and.E(I).le.CRES(LEM) )
53:                              &
54:                                  then
55:                                      LE = LEMESH + IEP(I)
56:                                      LL = LE + NPTS
57:                                      SGTL(IBP(I),N) = CRES(LL) + (CRES(LE)-E(I)) /
58:                                          &
59:                                              (CRES(LE)-CRES(LE-1))*
60:                                                  (CRES(LL-1)-CRES(LL))
61:                                  end if
62:                              &
63:                                  continue
64:                          end if
65:                      end if
66:                  120 continue
67:              end if
68:          130 continue
69:      end if

```

src/mvp/gtvval.f

```

1:      subroutine GTVVAL( VNAME, VALUE, IERR )
2: C=<MVP>=====
3: C PURPOSE: Return the value of an internal variable named VNAME:
4: C CALLED IN: DENTAK, PARA & OTHERS
5: C ARGUMENTS:
6: C     VNAME : variable name (not variable itself !)
7: C     VALUE : value (expanded to double precision if necessary)
8: C     IERR  : error code
9: C           0 : successful
10: C          1 : non-existent variable name or value fetching
11: C             not allowed.
12: C
13: C           2 : probably value undefined yet
14: C              (this case is not supported currently)
15: C
16: C CAUTION: This routine is called from utility routine DENTAK & PARAM
17: C           But code-dependent (MVP & GMVP use different source for
18: C           this routine.
19: C
20: C=====
21:      character*(*) VNAME
22:      real*8 VALUE
23:      integer IERR
24: C
25:      include 'INC/_KPIDS'
26:      include 'INC/_NGPS'
27: C
28:      include '../shared/INC/_SIZES'
29:      include 'INC/_SIZES2'
30:      include '../shared/INC/_PGEOM'
31:      include 'INC/_CXSEC'
32: C
33:      real*8 V0
34: C
35: C ..... VJ : A JUNK VALUE TO CONFIRM VALUE ASSIGNMENT .....
36: C
37:      real*8 VJ
38:      parameter( VJ = -9.9876543D38 )
39: C
40:      IERR = 0
41:      V0 = VJ
42: C
43: C-----
44: C
45: C---- COMMON /SIZES/ ----
46: C
47:      if ( VNAME(1:2).ge.'NA'.and.VNAME(1:2).le.'NF' ) then
48:          if ( VNAME.eq.'NBATCH' ) V0 = NBATCH
49:          if ( VNAME.eq.'NCELL' ) V0 = NCELL
50:          if ( VNAME.eq.'NEST' ) V0 = NEST
51:          if ( VNAME.eq.'NFBANK' ) V0 = NFBANK
52:          if ( VNAME.eq.'NFBNK0' ) V0 = NFBNK0
53:      else if ( VNAME(1:2).ge.'NG'.and.VNAME(1:2).le.'NI' ) then
54:          if ( VNAME(1:3).eq.'NGP'.or.VNAME(1:6).eq.'NGROUP' ) then
55:              KK = INDEX(VNAME,'.')
56:              if ( KK.ne.0.and.KK.lt.LEN(VNAME) ) then
57:                  call KPSYMB( VNAME(KK+1:), '>', IK, 0 )
58:                  if ( IK.ne.0 ) V0 = NGP(IK)
59:              else if ( KK.eq.0 ) then
60:                  if ( VNAME.eq.'NGP1' ) V0 = NGP(KPNEUT)
61:                  if ( VNAME.eq.'NGP2' ) V0 = NGP(KPPHOT)
62:                  if ( VNAME.eq.'NGROUP' ) V0 = NGROUP
63:              end if
64:          end if
65:          if ( VNAME.eq.'NHIST' ) V0 = NHIST

```

```

66:          if ( VNAME.eq.'NHSUB' ) V0 = NHSUB
67:          if ( VNAME.eq.'NINPZ' ) V0 = NINPZ
68:      else if ( VNAME(1:2).ge.'NJ'.and.VNAME(1:2).le.'NN' ) then
69:          if ( VNAME.eq.'NMAT' ) V0 = NMAT
70:          if ( VNAME.eq.'NMEMS' ) V0 = NMEMS
71:          if ( VNAME.eq.'NLATT' ) V0 = NLATT
72:          if ( VNAME.eq.'NLLEV' ) V0 = NLLEV
73:          if ( VNAME.eq.'NLBZ' ) V0 = NLBZ
74:      else if ( VNAME(1:2).ge.'NO'.and.VNAME(1:2).le.'NZ' ) then
75:          if ( VNAME.eq.'NPART' ) V0 = NPART
76:          if ( VNAME.eq.'NZONE' ) V0 = NZONE
77:          if ( VNAME.eq.'NREG' ) V0 = NREG
78:          if ( VNAME.eq.'NSOUR' ) V0 = NSOUR
79:          if ( VNAME.eq.'NRESP' ) V0 = NRESP
80:          if ( VNAME.eq.'NTHIST' ) V0 = NTHIST
81:          if ( VNAME.eq.'NTIME' ) V0 = NTIME
82:          if ( VNAME.eq.'NTREG' ) V0 = NTREG
83:          if ( VNAME.eq.'NPIC' ) V0 = NPIC
84:          if ( VNAME.eq.'NPKIND' ) V0 = NPKIND
85:      else
86:          if ( VNAME.eq.'BANKP' ) V0 = BANKP
87:          if ( VNAME.eq.'TCPU' ) V0 = TCPU
88:          if ( VNAME.eq.'ECUT' ) V0 = ECUT
89:      end if
90:      if ( V0.ne.VJ ) go to 100
91: C
92: C ..... COMMON /PGEOM/ .....
93: C
94:      if ( VNAME.eq.'DEPS' ) V0 = DEPS
95:      if ( VNAME.eq.'DINF' ) V0 = DINF
96:      if ( V0.ne.VJ ) go to 100
97: C
98: C ..... COMMON /CXSEC/ .....
99: C
100:      if ( VNAME.eq.'NUC' ) V0 = NUC
101:      if ( VNAME.eq.'NPATOM' ) V0 = NPATOM
102:      if ( VNAME(1:4).eq.'ETOP'.or.VNAME(1:4).eq.'EBOT' ) then
103:          KK = INDEX(VNAME,'.')
104:          if ( KK.ne.0.and.KK.lt.LEN(VNAME) ) then
105:              call KPSYMB( VNAME(KK+1:), '>', IK, 0 )
106:              if ( IK.ne.0 ) then
107:                  if ( VNAME(1:4).eq.'ETOP' ) V0 = ETOPX(IK)
108:                  if ( VNAME(1:4).eq.'EBOT' ) V0 = EBOTX(IK)
109:              end if
110:          else if ( KK.eq.0 ) then
111:              if ( VNAME.eq.'ETOP' ) V0 = ETOPX(KPNEUT)
112:              if ( VNAME.eq.'EBOT' ) V0 = EBOTX(KPNEUT)
113:              if ( VNAME.eq.'ETOPP' ) V0 = ETOPX(KPPHOT)
114:              if ( VNAME.eq.'EBOTP' ) V0 = EBOTX(KPPHOT)
115:          end if
116:      end if
117: CC if ( VNAME.eq.'ETOP' ) V0 = ETOP
118: CC if ( VNAME.eq.'EBOT' ) V0 = EBOT
119: CC if ( VNAME.eq.'ETOPP' ) V0 = ETOPP
120: CC if ( VNAME.eq.'EBOTP' ) V0 = EBOTP
121:      if ( V0.ne.VJ ) go to 100
122: C
123: C .... VERIFICATION ...
124: C
125:      100 continue
126:      if ( V0.ne.VJ ) then
127:          VALUE = V0
128:          IERR = 0
129:      else
130:          IERR = 1

```

src/mvp/gtvval.f

```
131:      end if
132:      return
133:  end
```

SAFE

src/mvp/igtflg.f

```
1:      function IGTFLG(NAME)
2: C==<MVP>=====
3: C  PURPOSE: GET VALUE OF OPTION PARAMETERS(FLAG) BY NAME
4: C  CALLED IN: ANYWHERE
5: C  CALLS      : (NONE)
6: C=====
7:      include 'INC/_FLAGS'
8:      include '../shared/INC/_IUNIT'
9: C
10:     character*(*) NAME
11: C
12:     if ( NAME.eq.'JDEBG' ) then
13:         IGTFLG = JDEBG(1)
14:     else if ( NAME.eq.'JMCHK' ) then
15:         IGTFLG = JMCHK
16:     else if ( NAME.eq.'JTLLT' ) then
17:         IGTFLG = JTLLT
18:     else if ( NAME.eq.'JLATT' ) then
19:         IGTFLG = JLATT
20:     else
21:         write(IMG,*) 'XXX(IGTFLG) CANNOT GET FLAG VALUE !' ('', NAME,
22: &      ' ' )
23:         stop 666
24:     end if
25:     return
26:     end
```


src/mvp/inpidx.f

```

1:      subroutine INPIDX( INDX, IDS,  FILE,  PATH,  LPATH, IOPR, IEND )
2: C=<MVP>=====
3: C  PURPOSE: GET A NUCID & FILE NAME DATA FROM CROSS SECTION INDEX FILE.
4: C  CALLED FROM:  FALOC, FALOCF
5: C-----
6: C  INDX : I/O UNIT OF INDEX FILE
7: C  IDS  : ID  FETCHED.
8: C  FILE : FILE NAME FETCHED.
9: C  PATH : CURRENT PATH.
10: C  LPATH : LENGTH OF CURRENT PATH-NAME.
11: C  IOPR : message printout I/O unit.
12: C  IEND : 0 = DATA FETCHED
13: C       1 = END OF INDEX-FILE
14: C=====
15: C      character*(*) IDS, FILE, PATH
16: C
17: C      character*144 LINE
18: C      character*256 cwrk
19: C      logical NMD
20: C
21: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
22: C
23: C      ... get path name of index file ...
24: C
25: C      if ( PATH.eq.' ' ) then
26: C          call GTPATH( INDX, PATH, LPATH )
27: C      end if
28: C
29: C/#ENDIF
30: C
31: C      check %%%%%%%%%
32: C      write(*,*) '%% INDX, IDS,  FILE,  PATH,  LPATH, IOPR, IEND ',
33: C      & INDX,':',IDS,':',FILE,':',PATH,':',LPATH,':',IOPR,':',IEND
34: C      %%%%%%%%%
35: C
36: C      100 LINE      = ' '
37: C      read(INDX,fmt ='(A)',end =110) LINE
38: C      if ( LINE.eq.' ' ) go to 100
39: C      if ( LINE(1:1).eq.'*' ) go to 100
40: C
41: C      .... GET ID CHARACTER OR 'PATH' STRING ....
42: C
43: C      IS      = 1
44: C      call NOBLNK( LINE, IS, IE, LEN(LINE) )
45: C      IE      = IS + MIN(LEN(IDS)-1,IE-IS)
46: C
47: C
48: C      .... PATH NAME ....
49: C
50: C      if ( LINE(IS:IE).eq.'PATH' ) then
51: C
52: C          .... GET PATH NAME ...
53: C
54: C          IS      = IE + 1
55: C          call NOBLNK( LINE, IS, IE, LEN(LINE) )
56: C          IE      = IS + MIN(LEN(PATH)-1,IE-IS)
57: C          if ( IE.ge.IS ) then
58: C              PATH  = LINE(IS:IE)
59: C              LPATH  = IE - IS + 1
60: C
61: C/#IF .NOT.NOGETENV
62: C
63: C      ... When PATH includes "$xxxx", expand it with environment variable
64: C      $xxxx . Here "xxxx" contains alphanumeric characters and "_".
65: C

```

```

66: C          KK = INDEX( PATH, '$' )
67: C          if( KK.ne.0 ) then
68: C              call ENVEXP( PATH, CWRK, IOPR, IERR )
69: C              if( IERR.ne.0 ) then
70: C                  write(IOPR,7200) PATH(:ICLEN2(PATH))
71: C                  format(1x,'!!! (INPIDX) Failed to expand $xxx string ',
72: C                      &
73: C                      ' in PATH line of library index file.'//
74: C                      1x,' <PATH ',a,'>')
75: C                  call CNTERR('WARNING')
76: C              else
77: C                  PATH = CWRK
78: C                  LPATH = ICLEN2(PATH)
79: C              end if
80: C/#ENDIF
81: C
82: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
83: C
84: C      ... when PATH is given by relative path ...
85: C
86: C      UNIX : not start with '/'
87: C      MS-DOS(Windows) : not have disk drive letter ( "[a-z]:" )
88: C      MacOS : beginning with ":"
89: C
90: C/#IF UNIX
91: C      if ( LPATH.gt.0.and.PATH(1:1).ne.'/' ) then
92: C/#ELSEIF FILENAME(DOS)
93: C      *      if ( lpath.gt.0.and.PATH(2:2).ne.':' ) then
94: C/#ELSEIF FILENAME(MACOS)
95: C      *      if ( lpath.gt.0.and.PATH(1:1).eq.':' ) then
96: C/#ENDIF
97: C
98: C      ... get absolute path of index file again ...
99: C      call GTPATH( INDX, FILE, LPP )
100: C
101: C      PATH      = FILE(:LPP) //PATH(:LPATH)
102: C
103: C      end if
104: C
105: C/#ENDIF
106: C      end if
107: C      go to 100
108: C      else
109: C
110: C      .... ID ....
111: C
112: C      IDS      = ' '
113: C      IDS      = LINE(IS:IE)
114: C
115: C      .... GET FILENAME
116: C
117: C      IS      = IE + 1
118: C      call NOBLNK( LINE, IS, IE, LEN(LINE) )
119: C      IE      = IS + MIN(LEN(FILE)-1,IE-IS)
120: C
121: C      FILE     = ' '
122: C
123: C      if ( LPATH.gt.0 ) then
124: C/#IF UNIX
125: C
126: C      ... filename with relative directory path ..
127: C      if ( LINE(IS:IS).ne.'/' ) then
128: C          FILE  = PATH(1:LPATH) //LINE(IS:IE)
129: C      else
130: C          FILE  = LINE(IS:IE)

```

src/mvp/inpidx.f

```
131:          end if
132: C
133: C/#ELSE
134: *          FILE      = PATH(1:LPATH) //LINE(IS:IE)
135: C/#ENDIF
136:          else
137:          FILE      = LINE(IS:IE)
138:          end if
139: C
140: C      .... SKIP DUMMY LINE (FOR FUTURE USE) ....
141: C
142: Ccccccccccccc READ(INDX, '()')
143: C      end if
144: C
145: C      IEND      = 0
146: C      return
147: C
148: C      ..... END OF INDEX FILE .....
149: C
150: C      110 IEND      = 1
151: C      return
152: C      end
153: C
154: C =====
155: C subroutine gtpath gets directory path name of file opened on I/O
156: C unit IOU
157: C
158: C      subroutine GTPATH( IOU, PATH, LPATH )
159: C
160: C      character*(*) PATH
161: C
162: C      ... local data ...
163: C
164: C      character*256 FILE
165: C      logical NMD
166: C
167: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
168: C
169: C      PATH      = ' '
170: C      LPATH      = 0
171: C      FILE      = ' '
172: C      inquire(IOU, name = FILE, named = NMD, iostat = IOS)
173: C
174: C      if ( IOS.eq.0 ) then
175: C          if ( NMD ) then
176: C              K      = 0
177: C              do 100 K = LEN(FILE), 1, -1
178: C/#IF FILENAME(DOS)
179: C
180: C      CHAR(92) is '\' in ASC-II character code.
181: C      '\' may be taken as "escape code" characters in character string
182: C      on some Fortran compilers. so such an expression is used.
183: C
184: C          if ( FILE(K:K).eq.char(92) ) go to 110
185: C/#ELSEIF FILENAME(MACOS)
186: C          if ( FILE(K:K).eq.':' ) go to 110
187: C/#ELSE
188: C          if ( FILE(K:K).eq.'/' ) go to 110
189: C/#ENDIF
190: C      100      continue
191: C              K      = 0
192: C      110      continue
193: C          if ( K.gt.0 ) then
194: C              PATH      = FILE(:K)
195: C              LPATH      = K
196: C          end if
197: C      end if
198: C      end if
199: C/#ENDIF
200: C      return
201: C      end
```

src/mvp/intmpn.f

```
1: c##<2007/03/14:PN3:
2: c## subroutine INTMPN( IUB, NCIDI, TEMPN, NUC, IPRT )
3:   subroutine INTMPN( IUB, NCIDI, TEMPN, NUC, IPRT, NCIDP )
4: c##>
5: C=====
6: C purpose: input nuclide temperature from temporary file IUB
7: C         (data may be written in MTDATA routine.)
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C i IUB : working file I/O unit
11: C o NCIDI : nuclide ID's as input
12: C o TEMPN : nuclide temperature
13: C i NUC : number of nuclides
14: C i IPRT : message printout I/O unit
15: c##<2007/03/14:PN3:
16: C o NCIDP : nuclide ID's of photonuclear as input
17: c##>
18: C=====
19:   character*16 NCIDI(NUC)
20:   real*8 TEMPN(NUC)
21: c##<2007/03/14:PN3:
22:   character*16 NCIDP(NUC)
23: c##>
24:
25:   do 100 I = 1, NUC
26: c##<2007/03/14:PN3:
27: c##       read(IUB,err =110,end =110) NCIDI(I), TEMPN(I)
28:       read(IUB,err =110,end =110) NCIDI(I), TEMPN(I), NCIDP(I)
29: c##>
30:   100 continue
31: C
32:   return
33: C
34: c##<2007/03/14:PN3:
35: c#110 write(IPRT,*) 'XXX(INTMPN) Error in reading nuclide temperature',
36:   110 write(IPRT,'(1X,A,A,I3,A)')
37:   & 'XXX(INTMPN) Error in reading nuclide temperature',
38: c##>
39:   & ' from temporary file (unit ', IUB, ' )'
40:   call PRSTOP( 1, 'Error in nuclide temperature hadling.' )
41:   stop 666
42: C
43:   end
```

src/mvp/intro2.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine INTRO2( A,      H,      CHA,      LIMIT, LIMITL, LIMITC,
3:     &                TITLE, MAXSF, JNP,      TCUTDM )
4: C=<MVP>=====
5: C PURPOSE: COMPLETE PREPARATION FOR RANDOM WALK AFTER DATA INPUT.
6: C CALLED IN: INTRO
7: C CALLS:
8: C   CKODR4 CNTERR CRVOL CRVOL1 CRVOLT DENTB2 EDITFL HEADER KEEP
9: C   KEPV LABEL MID2NM R4PRNT REFZON RESTI0 RESTRT SRCINP WRKARY
10: C
11: C-----
12: C
13: C arguments ( i = input, o = output, c =constant, w = work )
14: C
15: C c IPR : printout I/O unit
16: C
17: C io A(*) : dynamic memory array (task shared)
18: C io H(*) : dynamic memory array (task local)
19: C io CHA(*) : character dynamic memory array (task shared)
20: C i LIMIT : effective size of A(*)
21: C i LIMITL : effective size of H(*)
22: C i LIMITC : effective size of CHA(*)
23: C
24: C i TITLE : problem titile
25: C i MAXSF : maximum number of surface data to be used in geometry
26: C i JNP : 1 = neutron-photon coupled problem 0 = no
27: C o JNRUN : 1 = input is non-executable because of fatal errors
28: C           0 = no fatal error
29: C=====
30: C
31:   real A(*)
32:   real H(*)
33:   character*4 CHA(*)
34: C
35:   character TITLE(2)*72
36: C
37:   include '../shared/INC/_TASKDT'
38:   include '../shared/INC/_LISTOFF'
39: C
40:   include 'INC/_KPIDS'
41:   include 'INC/_KPSYMS'
42:   include 'INC/_NGPS'
43: C
44:   include 'INC/_FLAGS'
45:   include '../shared/INC/_SIZES'
46:   include 'INC/_SIZES2'
47:   include 'INC/_CXSEC'
48:   include 'INC/_PXSEC'
49:   include 'INC/_XBANK'
50:   include 'INC/_FBANK2'
51:   include 'INC/_SBANK'
52:   include 'INC/_STACK'
53:   include 'INC/_XWORK'
54:   include '../shared/INC/_PGEOM'
55:   include '../shared/INC/_PVRED'
56:   include 'INC/_PSOUR'
57:   include '../shared/INC/_PTALY0'
58:   include 'INC/_PTALY'
59:   include 'INC/_PTALY2'
60:   include '../shared/INC/_STALY'
61:   include '../shared/INC/_PTLSP'
62:   include '../shared/INC/_LISTON'
63:   include '../shared/INC/_WORDL'
64:   include '../shared/INC/_IOUNIT'
65:   include 'INC/_PERT' ! pert

```

```

66: C
67: C-----
68: C
69: C
70: C-----
71: C ... KEEP MEMORY AREAS FOR NON-INPUT PARAMETERS & GIVE VALUES TO THEM
72: C-----
73: C
74: C
75: C-----
76: C   .... /SIZES/ .....
77: C-----
78: CCCCC IF(NSCT.EQ.0) NSCT = NSCTX
79: C
80: CCCC call CHKNP( 'MVP', JNEUT, NGP1, JPHOT, NGP2, NGROUP )
81:       call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2, NGROUP )
82:       call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
83: C
84: C-----
85: C   .... /PGEOM/ .....
86: C-----
87: C
88: C-----
89: C   .... TRANSLATE MAT ID # TO MAT # .....
90: C-----
91: C
92:       call MID2NM( 'MVP', A(LKZMAT), A(LKMAT), A(LKINPZ), NINPZ, NZONE,
93:         &          JIMAG, A(LIDMAT), A(LDSMAT), NMAT )
94: C
95: C-----
96: C   .... NEXT-ZONE MEMORY ARRAY .....
97: C-----
98: C
99: C ... CHECK WHETHER NMEMO WAS INPUT OR NOT .....
100: C-----
101:       if ( NMEMO.le.0 ) then
102:         NMEMO = 5
103:         write(IMSG,7000) NMEMO
104:         call CNTERR( 'WARNING' )
105:       end if
106: C
107: 7000 format(/' !!!(intro2) Data "NMEMO" was set to a default value (',
108:   & i4,')'.)
109:   &/13x,' "NMEMO" is the size of memory of next-entering-zones.'
110:   &/13x,'For each zone, and it is desirable that "NMEMO" be greater',
111:   & ' than the maximum number of possible next-zones.'
112:   &/13x,'This parameter might be important to reduce time consumed',
113:   & ' for particle tracking in some cases.'/)
114: C
115:       call LKEPV( H(1), 'KMEMO', LKMEMO, NMEMO*NZONE, 'I4', LAST, 0,
116:         &          JDEBG )
117:       if ( NMEMOP.gt.0 ) then
118:         call LKEPV( H(1), 'MEMC', LMEMC, NMEMO*NZONE, 'I4', LAST, 0,
119:         &          JDEBG )
120:         call LKEPV( H(1), 'MEMZ', LMEMZ, NZONE, 'I4', LAST, 0, JDEBG )
121:       end if
122: C
123:       if ( LVOL.ne.0 ) then
124:         write(IPR,7020) (A(LVOL+I),I=0,NINPZ-1)
125:       end if
126: 7020 format(/1X,130('=')' VOLUMES OF EACH INPUT-ZONE '//1X,130('=')//
127:   &          (1X,2X,1P,10E12.5))
128: C
129: C-----
130: C   .... VARIANCE REDUCTION PARAMETER .....

```

src/mvp/intro2.f

```

131: C-----
132: C
133: C      =====
134: C      call HEADER( IPR, 'VARIANCE REDUCTION PARAMETERS' )
135: C      =====
136: C
137: C      if ( JIMPT.ne.0.and.LXIMP.eq.0 ) then
138: C          call KEPV( A(1), 'XIMP', LXIMP, NGROUP*NREG, 'R4', LAST, 1.0,
139: C          &          JDEBG )
140: C      end if
141: C      if ( JBRILT.ne.0 .or. JWWND.ne.0 ) then
142: C          if ( LWKIL.eq.0 ) then
143: C              call KEPV( A(1), 'WKIL', LWKIL, NGROUP*NREG, 'R4', LAST,
144: C              &          1.E-5, JDEBG )
145: C          end if
146: C          if ( LWSRV.eq.0 ) then
147: C              call KEPV( A(1), 'WSRV', LWSRV, NGROUP*NREG, 'R4', LAST,
148: C              &          1.E-4, JDEBG )
149: C          end if
150: C      end if
151: C
152: C      if ( LSTCK(0).lt.0 ) then
153: C          write(IMG,7140) 'variance reduction parameter printout.'
154: C          call PRSTOP( 1, 'MEMORY OVER.' )
155: C          stop 999
156: C      end if
157: C
158: C      if ( JIMPT.ne.0 ) then
159: C
160: C          *****
161: C          call LABEL( IPR, 'IMPORTANCE (GROUP & REGION-WISE)' )
162: C          *****
163: C
164: C          call R4PRNT( 'XIMP', A(LXIMP), NGROUP, NREG, 'GROUP', 'REGION',
165: C          &          IPR )
166: C      end if
167: C
168: C      if ( JRRILT.ne.0 .or. JWWND.ne.0 ) then
169: C          *****
170: C          call LABEL( IPR, 'WEIGHT THRESHOULD FOR RUSSIAN-ROULETTE' )
171: C          *****
172: C
173: C          call R4PRNT( 'WKIL', A(LWKIL), NGROUP, NREG, 'GROUP', 'REGION',
174: C          &          IPR )
175: C
176: C          *****
177: C          call LABEL( IPR, 'SURVIVAL WEIGHT IN RUSSIAN-ROULETTE' )
178: C          *****
179: C
180: C          call R4PRNT( 'WSRV', A(LWSRV), NGROUP, NREG, 'GROUP', 'REGION',
181: C          &          IPR )
182: C
183: C      ... check consistency of WSRV and WKIL
184: C
185: C          call CKWGHT( A(LWSRV), A(LWKIL), NGROUP, NREG )
186: C
187: C      end if
188: C
189: C      if ( JTIME.ne.0.and.LWTIME.ne.0 ) then
190: C          JWTIM = 1
191: C          *****
192: C          call LABEL( IPR, 'TIME BIN WEIGHTING FACTOR' )
193: C          *****
194: C
195: C          call R4PRNT( 'WTIME', A(LWTIME), NTIME, 1, 'TIME', ' ', IPR )

```

```

196: C
197: C      ... WTIME must be non positive ...
198: C      call CKVAL4( 'WTIME', A(LWTIME), NTIME, IFL, 1 )
199: C      if ( IFL.ne.0 ) then
200: C          write(IMG,'(1X,A,I6,A)')
201: C          &          'XXX(intro2) Time bin weight factor WTIME includes',
202: C          &          IFL, ' non positive value.'
203: C          call CNTERR( 'FATAL' )
204: C      end if
205: C      end if
206: C
207: C      if ( JPSTR.ne.0 ) then
208: C          *****
209: C          call LABEL( IPR, 'PATH STRETCHING PARAMETERS' )
210: C          *****
211: C          if ( JLATT.ne.0 ) then
212: C              write(IPR,*)
213: C              &          '!!! Both PATH-STRETCHING option and LATTICE option ',
214: C              &          ' are selected.',
215: C              &          ' Path stretching in lattice region may not be effective',
216: C              &          ' in current version.'
217: C              call CNTERR( 'WARNING' )
218: C          end if
219: C
220: C      ...path stretching with lattice geometry requires JFISX > 0
221: C      to get absolute coordinates of particles.
222: C
223: C      if ( JLATT.ne.0 ) JFISX = 1
224: C
225: C      if ( LPSXYZ.eq.0 ) then
226: C          write(IMG,'(1X,A,A)')
227: C          &          'XXX(intro2) PATH-STRETCHING option is selected but',
228: C          &          ' no target points PSXYZ(x y z) are given.'
229: C          call CNTERR( 'FATAL' )
230: C      else
231: C          write(IPR,7040)
232: C          7040      format(/1X,3X,'=== target point coordinates ===')
233: C          call PRXYZ( IPR, 'PSXYZ', A(LPSXYZ) )
234: C      end if
235: C
236: C      if ( LPSALP.eq.0 ) then
237: C          write(IMG,'(1X,A,A)')
238: C          &          '!!!(intro2) Path stretching factor (PSALP) is not',
239: C          &          ' input!! Set to 0.0.'
240: C          call CNTERR( 'WARNING' )
241: C          call KEPV( A(1), 'PSALP', LPSALP, NREG, 'R4', LAST, 0.0,
242: C          &          JDEBG )
243: C      end if
244: C          call R4PRNT( 'PSALP', A(LPSALP), NREG, 1, 'REGION', ' ', IPR )
245: C          call CKALP( A(LPSALP), NREG )
246: C      end if
247: C
248: C-----
249: C      .... /PXSEC/ ..... FISSION WEIGHT & FISSION SPECTRUM
250: C-----
251: C
252: C      if ( JFISS.ne.0 ) then
253: C
254: C          *****
255: C          call HEADER( IPR, 'DATA TO CONTROL FISSION REACTION' )
256: C          *****
257: C
258: C          if ( LWGTF.eq.0 ) then
259: C              write(IMG,'(1X,A,A)')
260: C              &          '!!!(intro2) Fission weights (WGTF) are not',

```

src/mvp/intro2.f

```

261:      &          ' input!! Set to 1.0.'
262:      call CNTERR( 'WARNING' )
263: c##<2007/03/14:PN3:
264: c##          call KEPV( A(1), 'WGTF', LWGTF, NREG, 'R4', LAST, 1.0, JDEBG
265: c## &          )
266:      call KEPV( A(1), 'WGTF', LWGTF, NREG*2, 'R4', LAST, 1.0,
267:      &          JDEBG )
268: c##>
269:      end if
270: C
271: C *****
272: C      call LABEL( IPR, 'FISSION WEIGHTS' )
273: C *****
274: C
275: c##<2007/03/14:PN3:
276: c##          call R4PRNT( 'WGTF', A(LWGTF), NREG, 1, 'REGION', ' ', IPR )
277: c##C
278: c##          call CKVAL4( 'WGTF', A(LWGTF), NREG, IFL, 1 )
279:      if ( JDLN.eq.0 ) then
280:          call R4PRNT( 'WGTF', A(LWGTF), NREG, 1, 'REGION', ' ', IPR )
281:      else
282:          call R4PRNT( 'WGTF', A(LWGTF), NREG, 2, 'REGION', 'PRM/DLN',
283:          &          IPR )
284:      end if
285:      call CKVAL4( 'WGTF', A(LWGTF), NREG*2, IFL, 1 )
286: c##>
287:      if ( IFL.ne.0 ) then
288:          write(IMG,'(1X,A,I6,A)')
289:          &          'XXX(intro2) Fission neutron weight WGTF includes ',
290:          &          IFL, ' non positive value.'
291:          call CNTERR( 'FATAL' )
292:      end if
293: C
294:      call KEEP( 'WGTFI', LWGTFI, NREG, 'R4', LAST, JDEBG )
295: C
296:      call ARYINV( A(LWGTF), A(LWGTFI), NREG )
297: C
298:      end if
299: C
300:      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
301: C
302: C =====
303: C      call HEADER( IPR, 'DATA FOR PHOTON GENERATION' )
304: C =====
305: C
306:      if ( LWGTP.eq.0 ) then
307:          write(IMG,'(1X,A,A)')
308:          &          '!!!(intro2) Photon weights (WGTP) are not input!!',
309:          &          ' Set to 1.0.'
310:          call CNTERR( 'WARNING' )
311:          call KEPV( A(1), 'WGTP', LWGTP, NREG, 'R4', LAST, 1.0, JDEBG
312:          &          )
313:      end if
314: C
315: C *****
316: C      call LABEL( IPR, 'PHOTON GENERATION WEIGHTS' )
317: C *****
318: C
319:      call R4PRNT( 'WGTP', A(LWGTP), NREG, 1, 'REGION', ' ', IPR )
320: C
321:      call CKVAL4( 'WGTP', A(LWGTP), NREG, IFL, 1 )
322:      if ( IFL.ne.0 ) then
323:          write(IMG,'(1X,A,I6,A)')
324:          &          'XXX(intro2) Photon generation weight WGTP includes ',
325:          &          IFL, ' non positive value.'
```

```

326:      call CNTERR( 'FATAL' )
327:      end if
328: C
329: c##<2007/03/14:PN3:
330:      if ( JPHNU.ne.0 ) then
331: C
332: C =====
333: C      call HEADER( IPR, 'DATA FOR PHOTO-NUCLEAR GENERATION' )
334: C =====
335: C
336:      if ( LWGTPN.eq.0 ) then
337:          write(IMG,907)
338:          call CNTERR( 'WARNING' )
339:          call KEPV( A(1), 'WGTPN', LWGTPN, NMTPN*NUCPN*NREG, 'R4',
340:          &          LAST, 0.001, JDEBG )
341:      end if
342:      if ( NMONPNW.gt.0 ) then          ! if monitoring of photonuclea
r reaction, WGTPN is forced to 0.
343:          do I = 1, NMTPN*NUCPN*NREG
344:              A(LWGTPN+I-1) = 0
345:          end do
346:      end if
347: C *****
348: C      call LABEL( IPR, 'PHOTO-NUCLEAR GENERATION WEIGHTS' )
349: C *****
350: C      call WGTPNVRT( A(LWGTPN), NMTPN, NUCPN, NREG, IPR )
351: C
352:      if ( LPPNBR.eq.0 ) then
353:          write(IMG,937)
354:          call CNTERR( 'WARNING' )
355:          call KEPV( A(1), 'PPNBR', LPPNBR, NUCPN, 'R4', LAST, 0.0,
356:          &          JDEBG )
357:      end if
358: C *****
359: C      call LABEL( IPR, 'FORCED PHOTO-NUCLEAR PROBABILITIES' )
360: C *****
361: C      call R4PRNT( 'PPNBR', A(LPPNBR), NUCPN, 1, 'NUCLIDE',
362: C      &          ' ', IPR )
363: C      &
364: C      call CKVAL4( 'PPNBR', A(LPPNBR), NUCPN, IFL, 0 )
365:      if ( IFL.ne.0 ) then
366:          write(IMG,938) IFL
367:          call CNTERR( 'FATAL' )
368: C
369:      if ( LPFCNR.eq.0 ) then
370:          write(IMG,939)
371:          call CNTERR( 'WARNING' )
372:          call KEPV( A(1), 'PFCN', LPFCNR, (NUCPN+1)*NREG, 'R4',
373:          &          LAST, 0.0, JDEBG )
374:      end if
375: C *****
376: C      call LABEL( IPR, 'FORCED COLLISION PROBABILITIES OF PHOTON' )
377: C *****
378: C      call R4I4PRNT( 'PFCN', A(LPFCNR), NUCPN+1, NREG, 'PROBAB.',
379: C      &          'NUCLIDE', 'REGION', IPR )
380: C      &
381: C      call CKVAL4( 'PFCN', A(LPFCNR), (NUCPN+1)*NREG, IFL, 0 )
382:      if ( IFL.ne.0 ) then
383:          write(IMG,940) IFL
384:          call CNTERR( 'FATAL' )
385: C
386:      if ( LPMT.eq.0 ) then
387:          write(IMG,941)
388:          call CNTERR( 'WARNING' )
389:          call KEPV( A(1), 'PMT', LPMT, NUCPN*NREG*2, 'R4', LAST,
```

src/mvp/intro2.f

```

390:      &                0.0, JDEBG )
391:      end if
392: C      *****
393:      call LABEL( IPR, 'FORCED PN REACTION PROBABILITIES' )
394: C      *****
395:      call PMTWRT( A(LPMT), NUCPN, NREG, IPR )
396: C
397:      if ( NMONPNW.gt.0 ) then
398: C      *****
399:      call LABEL( IPR, 'MONITORING OF PHOTO-NUCLEAR REACTION' )
400: C      *****
401:      call MONPNWRT( A(LMONPNW), NMONPNW, MONPNW, IPR )
402:      end if
403:      end if
404: 907 format(' !!!(intro2) Secondary particle weights (WGTPN) of',
405:      & ' photonuclear are not input !! Set to 0.001.')
406: 908 format(' XXX(intro2) Secondary particle weight WGTPN includes ',
407:      & ' i5,' negative values.')
408: 937 format(' !!!(intro2) Probabilities of forced photonuclear',
409:      & ' reaction (PPNBR) are not input !! Set to 0.')
410: 938 format(' XXX(intro2) Probabilities of forced photonuclear',
411:      & ' reaction PPNBR includes ',i5,' negative values.')
412: 939 format(' !!!(intro2) Probabilities of forced collision nuclide',
413:      & ' for photon (PFCN) are not input !! Set to 0.')
414: 940 format(' XXX(intro2) Probabilities of forced collision nuclide',
415:      & ' for photon PFCN includes ',i5,' negative values.')
416: 941 format(' !!!(intro2) Probabilities of forced photonuclear',
417:      & ' reaction type (PMT) are not input !! Set to 0.')
418: c##>
419:      end if
420: C-----
421: C      .... /PSOUR/ .....
422: C-----
423:
424: C/IF .NOT.SOURCE(NEW)
425:
426:      if ( LPENRG.eq.0 ) then
427:      if ( JFIXD.ne.0 ) then
428:      write(IMG,'(1X,A,A)')
429:      &      ' !!!(intro2) Source energy distribution is not',
430:      &      ' specified (PENRG). OK ?'
431:      call CNTERR( 'WARNING' )
432:      else
433:      call KEEP( 'PENRG', LPENRG, NGROUP*NSOUR, 'R4', LAST, JDEBG )
434:      &
435:      end if
436:      end if
437: C
438: C-----
439: C      ..... Change ISZON(2,NSOUR) from INPUT-ZONE to ZONE .....
440: C-----
441:      if ( LISZON.ne.0 ) then
442:      call CISZON( A(LISZON), NSOUR, A(LKINPZ), NZONE )
443:      else
444:      call KEPV( A(1), 'ISZON', LISZON, 2*NSOUR, 'I4', LAST, 0, JDEBG )
445:      &
446:      end if
447:      if ( NMEMS.eq.0 ) NMEMS = 5
448:      call LKEPV( H(1), 'MEMZN', LMEMZN, NMEMS*NSOUR, 'I4', LAST, 0,
449:      &      JDEBG )
450:      call KEEP( 'KENRG', LKENRG, 2*NGROUP*NSOUR, 'I4', LAST, JDEBG )
451:      call KEEP( 'INSRC', LINSRC, NSOUR, 'I4', LAST, JDEBG )
452: Ccccc call KEEP( 'XSOUR', LXSOUR, NSOUR, 'R4', LAST, JDEBG )
453: Ccccc call KEEP( 'XSOUR', LXSOUR, NSOUR, 'R8', LAST, JDEBG )
454: C-----

```

```

455: C ..... Printout of source data & B.M.C transformation .....
456: C-----
457:      if ( JEIGN.ne.0.and.LIFISM.eq.0 ) then
458:      write(IMG,'(1X,A,A)')
459:      &      ' !!!(intro2) Data "IFISM" is not specified. Set to 1.'
460:      call CNTERR( 'WARNING' )
461:      call KEPV( A(1), 'IFISM', LIFISM, NSOUR, 'I4', LAST, 1, JDEBG )
462:      end if
463: C
464:      call GTLAST( LTLAST )
465: C
466:      call KEEP( 'TEMP', LTEMP, NGROUP, 'I4', LTLAST, JDEBG )
467: C
468:      call PRSOUR( NSOUR, NGROUP, NMAT, JTIME, JEIGN, A(LKINPZ),
469:      &      A(LKSOUR), A(LISZON), A(LSOUR), A(LPSPAC), A(LPENRG),
470:      &      A(LKENRG), A(LNSTIM), A(LSTIM), A(LPSTIM), A(LISTIM),
471:      &      A(LNSANG), A(LSANG), A(LSAXIS), A(LPSANG), A(LISANG),
472:      &      A(LIFISM), A(LTEMP) )
473: C
474:      call STLAST( 'PRSOU', LTLAST, LAST )
475: C
476: C/#ENDIF
477: C
478: C
479: C-----
480: C      .... /PTALY/ .....
481: C-----
482: C
483: C      =====
484:      call HEADER( IPR, 'TALLY AND RESPONSE PARAMETERS' )
485: C      =====
486: C
487:      call GTLAST( LLS0 )
488:      call LGTLST( LLL0 )
489: C
490:      call LKEPV( H(1), 'WSUM', LWSUM, 1, 'R8', LAST, 0.0D0, JDEBG )
491:      call LKEPV( H(1), 'WLEK', LWLEK, 1, 'R8', LAST, 0.0D0, JDEBG )
492: CCCC call LKEPV( H(1), 'AAVT', LAAVT, 3, 'R4', LAST, 0.0D0, JDEBG )
493:      call LKEPV( H(1), 'AAVT', LAAVT, 3, 'R4', LAST, 0.0E0, JDEBG )
494:      call LKEPV( H(1), 'XAVT', LXAVT, 3, 'R4', LAST, 0.0E0, JDEBG )
495:      call LKEPV( H(1), 'EAVT', LEAVT, 1, 'R4', LAST, 0.0E0, JDEBG )
496:      call LKEPV( H(1), 'NLOST', LNLOST, 1, 'I4', LAST, 0, JDEBG )
497: C
498: C      NPKIND = 0
499:      if ( JNEUT.ne.0 ) NPKIND = NPKIND + 1
500:      if ( JPHOT.ne.0 ) NPKIND = NPKIND + 1
501: C
502:      if ( JRESP.ne.0.and.NRESP.ne.0.and.LRESP.eq.0 ) then
503:      write(IMG,'(1X,A,A)')
504:      &      ' !!!(intro2) No response function is specified. ',
505:      &      ' All assumed to be 1.0.'
506:      call CNTERR( 'WARNING' )
507:      call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4', LAST, 1.0,
508:      &      JDEBG )
509:      end if
510:      if ( JRESP.eq.0 ) then
511:      if ( NRESP2.eq.0 ) NRESP = 0
512:      if ( NSTAL2.eq.0 ) NSTAL = 0
513:      end if
514: C
515:      call LKEPV( H(1), 'FLTR', LFLTR, NGROUP*NREG, 'R8', LAST, 0D0,
516:      &      JDEBG )
517:      call LKEPV( H(1), 'SFLTR', LSFLTR, NGROUP*NTREG*2, 'R8', LAST,
518:      &      0D0, JDEBG )
519:      call LKEPV( H(1), 'FLCL', LFLCL, NGROUP*NREG, 'R8', LAST, 0D0,

```

src/mvp/intro2.f

```

520:      &      JDEBG )
521:      call LKEPV( H(1), 'SFLCL', LSFLCL, NGROUP*NTREG*2, 'R8', LAST,
522:      &      OD0, JDEBG )
523: C
524:      if ( JRESP.ne.0 ) then
525:      call LKEPV( H(1), 'RETR', LRETR, NREG*NRESP, 'R8', LAST, OD0,
526:      &      JDEBG )
527:      call LKEPV( H(1), 'SRETR', LSRETR, NTREG*NRESP*2, 'R8', LAST,
528:      &      OD0, JDEBG )
529:      call LKEPV( H(1), 'RECL', LRECL, NREG*NRESP, 'R8', LAST, OD0,
530:      &      JDEBG )
531:      call LKEPV( H(1), 'SRECL', LSRECL, NTREG*NRESP*2, 'R8', LAST,
532:      &      OD0, JDEBG )
533:      end if
534: C-----
535: C For weighted time of flight
536: C-----
537:      if ( JTIME.ne.0 ) then
538: CC      call LKEPV( H(1), 'TFLH', LTFLH, NPKIND, 'R8', LAST, OD0, JDEBG
539: CC      &
540: CC      call LKEPV( H(1), 'TFLHS', LTFLHS, NPKIND*2, 'R8', LAST, OD0,
541: CC      &      JDEBG )
542:      call LKEPV( H(1), 'TFLH', LTFLH, KPLIM, 'R8', LAST, OD0, JDEBG
543:      &
544:      call LKEPV( H(1), 'TFLHS', LTFLHS, KPLIM*2, 'R8', LAST, OD0,
545:      &      JDEBG )
546:      end if
547: C
548: C-----
549: C      .... FOR POINT DETECTOR ....
550: C-----
551: C
552:      if ( NPDET.eq.0 ) JPTDT = 0
553:      if ( JPTDT.eq.0 ) NPDET = 0
554: C
555:      if ( JPTDT.ne.0 ) then
556: C
557: C      .... RELOCATE XPDET ARRAY FOR USE IN RANDOM WORK ....
558: C      AND CALCULATE LATTICE-RELATED PARAMETER IF NECESSARY.
559: C
560:      NPLEN = 2 + (NEST+1)*3
561:      call KEPV(A(1), 'XPDET', LTEMP, NPLEN*NPDET, 'R8', LAST, OD0,
562:      &      JDEBG )
563:      call KEPV(A(1), 'IPDET', LIPDET, NPDET*2, 'I4', LAST, 0, JDEBG )
564:      if ( JLATT.ne.0 ) then
565:      call KEPV(A(1), 'IPDT2', LIPDT2, NEST*3*NPDET, 'I4', LAST, 0,
566:      &      JDEBG )
567:      end if
568:      if ( LJPUSED.eq.0 ) then
569:      call KEPV(A(1), 'JPUSED', LJPUSED, NPDET, 'I4', LAST, 1,
570:      &      JDEBG )
571:      end if
572:      if ( LJSPDT.eq.0 ) then
573:      if ( JEIGN.ne.0 ) then
574:      call KEPV(A(1), 'JSPDT', LJSPDT, NPDET, 'I4', LAST, 0,
575:      &      JDEBG )
576:      else
577:      call KEPV(A(1), 'JSPDT', LJSPDT, NPDET, 'I4', LAST, 1,
578:      &      JDEBG )
579:      end if
580:      end if
581: C
582:      if ( LSTCK(0).lt.0 ) then
583:      write(IPR,7140) 'point detector setting.'
584:      call PRSTOP( 1, 'MEMORY OVER.' )

```

```

585:      stop 999
586:      end if
587: C
588:      call RXPDET( A, JVMNT, JLATT, JSIMP, JHLAT, JEIGN,
589:      &      A(LXPDET), A(LTEMP), NPDET, NPLEN, NEST,
590:      &      A(LIPDET), A(LIPDT2), A(LJSPDT),
591:      &      A(LIPCEL), A(LKCELL), A(LKDALT), A(LKZMAT), A(LMLBZZ),
592:      &      A(LKZREG), A(LISUSP), A(LKSPSU), A(LKTCSP),
593:      &      NSDA, NZDA, NZONE, NCELL, NLBZ,
594:      &      NSPACE, NSUZON, NUNV, NKTCS )
595: C
596:      LXPDET = LTEMP
597: C
598: C      .... POINT DETECTOR TALLY ....
599: C
600:      call LKEPV(H(1), 'FLPD', LFLPD, NGROUP*NPDET, 'R8', LAST, OD0,
601:      &      JDEBG )
602:      call LKEPV(H(1), 'SFLPD', LSFLPD, NGROUP*NPDET*2, 'R8', LAST, OD0,
603:      &      JDEBG )
604:      if ( JRESP.ne.0 ) then
605:      call LKEPV(H(1), 'REPD', LREPD, NPDET*NRESP, 'R8', LAST, OD0,
606:      &      JDEBG )
607:      call LKEPV(H(1), 'SREPD', LSREPD, NPDET*NRESP*2, 'R8', LAST, OD0,
608:      &      JDEBG )
609:      end if
610: C
611: C
612:      end if
613: C
614: C-----
615: C      .... /PTALY2/ ..... (specific to MVP )
616: C-----
617:      call KEPV( A(1), 'LMAC', LLMAC, 8, 'I4', LAST, 0, JDEBG )
618:      call KEPV( A(1), 'LMIC', LLMIC, 8, 'I4', LAST, 0, JDEBG )
619:      call KEPV( A(1), 'LEMAC', LLEMAC, 16*2, 'I4', LAST, 0, JDEBG )
620:      call KEPV( A(1), 'LEMIC', LLEMIC, 16*2, 'I4', LAST, 0, JDEBG )
621:      if ( NSTAL.gt.0 ) then
622:      call KEPV( A(1), 'KCRES', LKCRES, NSTAL*4, 'I4', LAST, 0, JDEBG
623:      &
624:      end if
625:      if ( JPTDT.ne.0 ) then
626:      call KEPV(A(1), 'LMACI', LLMACI, 8, 'I4', LAST, 0, JDEBG )
627:      call KEPV(A(1), 'LMICI', LLMICI, 8, 'I4', LAST, 0, JDEBG )
628:      if ( NSTAL.gt.0 ) then
629:      call KEPV(A(1), 'KCRESI', LKCRSI, NSTAL*4, 'I4', LAST, 0,
630:      &      JDEBG )
631:      end if
632:      end if
633: C-----
634: C      .... Set values of LMIC, LMAC, LEMIC, LEMAC & etc.
635: C-----
636:      if ( LSTCK(0).lt.0 ) then
637:      write(IMG,7140) 'macro/micro reaction flags setting.'
638:      call PRSTOP( 1, 'MEMORY OVER.' )
639:      stop 999
640:      end if
641: C
642:      call EDITFL( JEIGN, JMICE, JMACE, A(LLMIC), A(LLMAC), NSMIC,
643:      &      NSMAC, A(LLEMIC), A(LLEMAC), NEMIC, NEMAC, JRESP,
644:      &      H(LIETAL), A(LKCRES), NIETAL, NETALY, NSTAL,
645:      &      c##<2007/03/14:PN3:
646:      &      JPTDT, A(LLMICI), A(LLMACI), NSMICI, NSMACI, A(LKCRSI) )
647:      &      JPTDT, A(LLMICI), A(LLMACI), NSMICI, NSMACI, A(LKCRSI),
648:      &      JPHNU, NSMICPN,
649:      &      c##>

```


src/mvp/intro2.f

```

650: c+beff1
651:      &          JBEFF )
652: c-beff1
653: C
654:      if ( NEMIC.gt.0 ) then
655:          NN      = (NGROUP+NPKIND)*NTREG
656:      else
657:          NN      = 0
658:      end if
659: call LKEPV( H(1), 'WCXTY', LWCXTY, NN, 'R8', LAST, OD0, JDEBG )
660: C
661:      NN      = NGROUP*NREG*NUC*NEMIC
662: call LKEPV( H(1), 'RMIC', LRMIC, NN, 'R8', LAST, OD0, JDEBG )
663:      NN      = NGROUP*NTREG*NUC*NEMIC*2
664: call LKEPV( H(1), 'SRMIC', LSRMIC, NN, 'R8', LAST, OD0, JDEBG )
665:      NN      = NPKIND*NTREG*NUC*NEMIC*2
666: call LKEPV( H(1), 'RMICR', LRMICR, NN, 'R8', LAST, OD0, JDEBG )
667:      NN      = (NGROUP+NPKIND)*NTREG*NUC*NEMIC*2
668: call LKEPV( H(1), 'XMIC', LXMIC, NN, 'R8', LAST, OD0, JDEBG )
669:
670:      if ( NEMIC.ne.0 ) then
671:          NN      = NGROUP*NREG*NUC
672:      else
673:          NN      = 0
674:      end if
675:
676: call LKEPV( H(1), 'DNFLX', LDNFLX, NN, 'R8', LAST, OD0, JDEBG )
677:      NN      = NGROUP*NTREG*NEMAC*2
678: call LKEPV( H(1), 'RMAC', LRMAC, NN, 'R8', LAST, OD0, JDEBG )
679:      NN      = NPKIND*NTREG*NEMAC*2
680: call LKEPV( H(1), 'RMACR', LRMACR, NN, 'R8', LAST, OD0, JDEBG )
681:      NN      = (NGROUP+NPKIND)*NTREG*NEMAC*2
682: call LKEPV( H(1), 'XMAC', LXMAC, NN, 'R8', LAST, OD0, JDEBG )
683: C
684:      if ( NEMAC.gt.0 ) then
685:          call LKEPV( H(1), 'DMAC', LDMAC, NGROUP*NTREG, 'R8', LAST, OD0,
686:      &          JDEBG )
687:      end if
688:      if ( JSCTM.ne.0 ) then
689:          KY = 0
690:          if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) KY = KY + 1
691:          if ( JMICE(6).ne.0.or.JMACE(6).ne.0 ) KY = KY + 1
692:          if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) KY = KY + 1
693:          NN = (NGP(KPNEUT)**2)*NREG*NUC*KY
694:          call LKEPV( H(1), 'DNFLXSM', LDNFLXSM, NN, 'R8', LAST, OD0,
695:      &          JDEBG )
696:          NN = (NGP(KPNEUT)**2)*NREG*NUC*KY*JSCTM
697:          call LKEPV( H(1), 'RMICSM', LRMICSM, NN, 'R8', LAST, OD0, JDEBG )
698:          NN = ((NGP(KPNEUT)+1)**2)*NTREG*NUC*KY*2*JSCTM
699:          call LKEPV( H(1), 'SRMICSM', LSRMICSM, NN, 'R8', LAST, OD0,
700:      &          JDEBG )
701:          LXMICSM = LAST + 1
702: c%c      call LKEPV( H(1), 'XMICSM', LXMICSM, NN, 'R8', LAST, OD0, JDEBG )
703:          KY = 0
704:          if ( JMACE(4).ne.0 ) KY = KY + 1
705:          if ( JMACE(6).ne.0 ) KY = KY + 1
706:          if ( JMACE(7).ne.0 ) KY = KY + 1
707:          if ( KY.gt.0 ) then
708:              NN = ((NGP(KPNEUT)+1)**2)*NTREG*KY*2*JSCTM
709:              call LKEPV( H(1), 'RMACSM', LRMACSM, NN, 'R8', LAST, OD0,
710:      &          JDEBG )
711:              LXMACSM = LAST + 1
712: c%c      call LKEPV( H(1), 'XMACSM', LXMACSM, NN, 'R8', LAST, OD0,
713: c%c      &          JDEBG )
714:          end if

```

```

715:      end if
716:      if ( JSCMU.ne.0 ) then
717:          KY = 0
718:          if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) KY = KY + 1
719:          if ( JMICE(6).ne.0.or.JMACE(6).ne.0 ) KY = KY + 1
720:          if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) KY = KY + 1
721:          NN = (NGP(KPNEUT)*(NGP(KPNEUT)+1))*NREG*NUC*KY
722:          call LKEPV( H(1), 'RMIMU', LRMIMU, NN, 'R8', LAST, OD0, JDEBG )
723:          call LKEPV( H(1), 'WMIMU', LWMIMU, NN, 'R8', LAST, OD0, JDEBG )
724:          NN = ((NGP(KPNEUT)+1)*(NGP(KPNEUT)+2))*NTREG*NUC*KY*3
725:          call LKEPV( H(1), 'SMIMU', LSMIMU, NN, 'R8', LAST, OD0, JDEBG )
726:          KY = 0
727:          if ( JMACE(4).ne.0 ) KY = KY + 1
728:          if ( JMACE(6).ne.0 ) KY = KY + 1
729:          if ( JMACE(7).ne.0 ) KY = KY + 1
730:          if ( KY.gt.0 ) then
731:              NN = ((NGP(KPNEUT)+1)*(NGP(KPNEUT)+2))*NTREG*KY*3
732:              call LKEPV( H(1), 'RMAMU', LRMAMU, NN, 'R8', LAST, OD0,
733:      &          JDEBG )
734:          end if
735:      end if
736: C
737: C
738: C-----
739: C ..... for speciall tally (when $TALLY block is specified )
740: C-----
741: C
742: C ... default tally with SGTAL bank is obsolete ...
743:      if ( JRESP.ne.0 ) then
744:          call LKEPV( H(1), 'RSTR', LRSTR, NREG*NSTAL, 'R8', LAST, OD0,
745:      &          JDEBG )
746:          call LKEPV( H(1), 'RSCL', LRSCCL, NREG*NSTAL, 'R8', LAST, OD0,
747:      &          JDEBG )
748:          call LKEPV( H(1), 'SRSTR', LSRSTR, NTREG*NSTAL*2, 'R8', LAST,
749:      &          OD0, JDEBG )
750:          call LKEPV( H(1), 'SRSCCL', LSRSCCL, NTREG*NSTAL*2, 'R8', LAST,
751:      &          OD0, JDEBG )
752:      end if
753: C
754:      if ( NDTALY.gt.0 ) then
755:          call LKEPV( H(1), 'DTALY', LDTALY, NLDTAL, 'R8', LAST, OD0,
756:      &          JDEBG )
757:      end if
758:      if ( NETALY.gt.0 ) then
759:          call LKEPV( H(1), 'ETALY', LETALY, NLETAL*2, 'R8', LAST, OD0,
760:      &          JDEBG )
761:      end if
762:      if ( LJTEVE.eq.0 ) then
763:          call LKEPV( A(1), 'JTEVE', LJTEVE, NTEVE, 'I4', LAST, 0, JDEBG )
764:      end if
765: C
766: C ..... Use free-lattice-frame mode for surface tally
767: C
768:      if ( JTSRF.ne.0.and.JLATT.ne.0 ) JFISX = 1
769: C
770: C-----
771: C .... regionwise monitoring ....
772: C-----
773: C
774:      if ( JMNTR.ne.0 ) then
775:          call LKEPV( H(1), 'NCLSN', LNCLSN, NGROUP*NREG, 'R8', LAST,
776:      &          OD0, JDEBG )
777:          call LKEPV( H(1), 'WCLSN', LWCLSN, NGROUP*NREG, 'R8', LAST,
778:      &          OD0, JDEBG )
779:          call LKEPV( H(1), 'NLEAK', LNLEAK, NGROUP*NREG, 'R8', LAST,

```

```
src/mvp/intro2.f
```

```

780:      &      OD0, JDEBG )
781:      call LKEPV( H(1), 'WLEAK', LWLEAK, NGROUP*NREG, 'R8', LAST,
782:      &      OD0, JDEBG )
783:      call LKEPV( H(1), 'NABSB', LNABSB, NGROUP*NREG, 'R8', LAST,
784:      &      OD0, JDEBG )
785:      call LKEPV( H(1), 'WABSB', LWABSB, NGROUP*NREG, 'R8', LAST,
786:      &      OD0, JDEBG )
787:      call LKEPV( H(1), 'NECUT', LNECUT, NREG, 'R8', LAST, OD0, JDEBG
788:      &      )
789:      call LKEPV( H(1), 'WECUT', LWECUT, NREG, 'R8', LAST, OD0, JDEBG
790:      &      )
791:      call LKEPV( H(1), 'NTCUT', LNTCUT, NGROUP*NREG, 'R8', LAST,
792:      &      OD0, JDEBG )
793:      call LKEPV( H(1), 'WTCUT', LWTCUT, NGROUP*NREG, 'R8', LAST,
794:      &      OD0, JDEBG )
795:      call LKEPV( H(1), 'NKILD', LNKILD, NGROUP*NREG, 'R8', LAST,
796:      &      OD0, JDEBG )
797:      call LKEPV( H(1), 'WKILD', LWKILD, NGROUP*NREG, 'R8', LAST,
798:      &      OD0, JDEBG )
799:      call LKEPV( H(1), 'NSURV', LNSURV, NGROUP*NREG, 'R8', LAST,
800:      &      OD0, JDEBG )
801:      call LKEPV( H(1), 'WSURV', LWSURV, NGROUP*NREG, 'R8', LAST,
802:      &      OD0, JDEBG )
803:      call LKEPV( H(1), 'NSPLT', LNSPLT, NGROUP*NREG, 'R8', LAST,
804:      &      OD0, JDEBG )
805:      call LKEPV( H(1), 'WSPLT', LWSPLT, NGROUP*NREG, 'R8', LAST,
806:      &      OD0, JDEBG )
807:      end if
808: C
809: C-----
810: C      .... event monitor ....
811: C-----
812: C
813: C##<2007/03/14:PN3:PN4:
814: C##      NEVENT = 30
815: C##      NEVENT = 34
816: C##>
817: C      call LKEPV( H(1), 'NCNTR', LNCNTR, NEVENT*2, 'R8', LAST, OD0,
818: C      &      JDEBG )
819: C      call LKEPV( H(1), 'WCNTR', LWCNTR, NEVENT*2, 'R8', LAST, OD0,
820: C      &      JDEBG )
821: C      call LKEPV( H(1), 'NCNTR', LNCNTR, NEVENT*KPLIM, 'R8', LAST, OD0,
822: C      &      JDEBG )
823: C      call LKEPV( H(1), 'WCNTR', LWCNTR, NEVENT*KPLIM, 'R8', LAST, OD0,
824: C      &      JDEBG )
825: C
826: CCC      NGROUP = NGP1 + NGP2
827: C+beff1
828: C      if( JBEFF.ne.0 ) then
829: C          NEBEF = 12
830: C          call LKEPV( H(1), 'WCBEF', LWCBEF, NEBEF, 'R8', LAST, OD0,
831: C          &      JDEBG )
832: C      end if
833: C-beff1
834: C##<2007/03/14:PN3:
835: C
836: C-----
837: C      .... independent monitoring ....
838: C-----
839: C
840: C      if ( JNEUT.ne.0.and.JPHOT.ne.0.and.JPHNU.ne.0.and.NMONPNW.gt.0 )
841: C      &      then
842: C          NN99 = MONPNW * 2
843: C          call LKEPV(H(1),'NMPNWGT',LNMPNWGT,MONPNW,'R8',LAST,OD0,JDEBG)
844: C          call LKEPV(H(1),'WMPNWGT',LWMPNWGT,NN99, 'R8',LAST,OD0,JDEBG)

```

```

845:      end if
846: c##>
847: C
848: C-----
849: C ..... Check energy range of boundary & ETOP, EBOT & ETHMAX..
850: C-----
851: C
852: C      (... do not pass IPR from Jan 2000)
853:       call EGYCHK( A, LIMIT, JKPAR, ETOPX(1), EBOTX(1), JNEUT, JPHOT,
854: &                JEIGN, JDEBG, NGP1, NGP2, NGROUP, ETOP, ETOPL, EBOT,
855: &                EBOTL, ETHMAX, ESABL, ETOPP, ETOPLP, EBOTP, EBOTLP,
856: &                LENGYB, LENGPB )
857: C
858:       EBOTE = EBOTP
859: C
860:       if ( JTIME.eq.0 ) then
861:         NTIME = 1
862:         call KEEP( 'TIMEB', LTIMEB, NTIME+1, 'R4', LAST, JDEBG )
863:       end if
864: C
865: C
866: C-----
867: C ..... Check and printout of energy & time boundary .....
868: C-----
869: C
870: C
871: C      *****
872:       call LABEL( IPR, 'ENERGY BOUNDARIES (EV)' )
873: C      *****
874: C
875:       J = 0
876:       do 100 IK = 1, KPLIM
877:         if ( JKPAR(IK).ne.0 ) then
878:           J = J + 1
879:           if ( J.gt.1 ) write(IPR,'(1H)')
880:             write(IPR,'( ' ** ',a,' **'/)')
881:           &          KPSYM(1,IK) (:ICLEN2(KPSYM(1,IK)))
882:           call R4PRNT( 'ENGYB', A(LENGYB+KENGPIK-1), NGP(IK)+1, 1,
883:           &          ' GROUP', ' ', IPR )
884:         end if
885:       100 continue
886: C
887:       if ( NGP1.gt.0 ) then
888:         write(IPR,'( ' ** NEUTRON **'/)')
889:         call R4PRNT( 'ENGYB', A(LENGYB), NGP1+1, 1, ' GROUP', ' ', IPR
890:         &          )
891:       end if
892: C
893:       if ( NGP2.gt.0 ) then
894:         write(IPR,'(/' ** PHOTON **'/)')
895:         call R4PRNT( 'ENGPB', A(LENGPB), NGP2+1, 1, ' GROUP', ' ', IPR
896:         &          )
897:       end if
898: C
899:       if ( JTIME.ne.0 ) then
900:         if ( LTIMEB.eq.0 ) then
901:           write(MSG,'(1X,A)')
902:           &          'XXX(intro2) No time bin boundary is specified.'
903:           call CNTERR( 'FATAL' )
904:         else
905: C
906: C      *****
907:       call LABEL( IPR, 'TIME BOUNDARIES (SEC)' )
908: C      *****
909: C

```

src/mvp/intro2.f

```

910:      call R4PRNT( 'TIMEB', A(LTIMEB), NTIME+1, 1, ' TIME', ' ',
911:      &      IPR )
912:      call CKODR4( A(LTIMEB), NTIME+1, 2, INVLD )
913:      if ( INVLD.ne.0 ) then
914:        write(IMG, '(//1X,A,A)')
915:        &      'XXX(intro2) Time bin boundary is not in',
916:        &      ' increasing order.'
917:      call CNTERR( 'FATAL' )
918:      end if
919: C
920:      if ( TCUT.eq.TCUTDM ) then
921:        TCUT = A(LTIMEB+NTIME)
922:        write(IPR, '(//1X,A,A)')
923:        &      '<<MESSAGE>> Cutoff time (TCUT) is set',
924:        &      ' to the highest time bin boundary.'
925:      call CNTERR( 'MESSAGE' )
926:      end if
927: C
928:      if ( JPTIM.ne.0 ) then
929:        if ( TCUT.ne.A(LTIMEB+NTIME) ) then
930:          TCUT = A(LTIMEB+NTIME)
931:          write(IPR, '(//1X,A,A)')
932:          &      '<<MESSAGE>> Cutoff time (TCUT) is set',
933:          &      ' to the highest time bin boundary.'
934:        call CNTERR( 'MESSAGE' )
935:        end if
936:        if ( A(LTIMEB).gt.0.0 ) then
937:          A(LTIMEB) = 0.0
938:          write(IPR, '(//1X,A,A)')
939:          &      '<<MESSAGE>> the lowest time bin boundary',
940:          &      ' is set to be 0.0.'
941:        call CNTERR( 'MESSAGE' )
942:        end if
943:      end if
944:    end if
945:  end if
946: C-----
947: C ..... Calulation & printout of REGION volume .....
948: C-----
949:      if ( LTRVOL.eq.0.and.LRVOL.eq.0.and.LVOL.eq.0 ) then
950:        write(IMG,7060)
951:        7060 format('// !!!(intro2) Neither tally-region volumes nor region',
952:        &      ' volumes are input, So all tally-region volumes are ',
953:        &      'set to 1.0//')
954:        call CNTERR( 'WARNING' )
955:        call KEPV( A(1), 'TRVOL', LTRVOL, NTREG, 'R4', LAST, 1.0, JDEBG
956:        &      )
957:      end if
958: C
959:      if ( LRVOL.eq.0 ) then
960:        if ( LVOL.eq.0 ) then
961:          write(IPR,7080)
962:          7080 format(/1X,<<MESSAGE>> Volumes of each input zone are ',
963:          &      'not input.'//10X,
964:          &      'Their volumes are set so that volumes of each',
965:          &      ' region are equal to 1.0//')
966:          call CNTERR( 'MESSAGE' )
967: C
968:          call KEPV( A(1), 'VOL', LVOL, NINPZ, 'R4', LAST, 0.0, JDEBG
969:          &      )
970:          call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 1.0, JDEBG
971:          &      )
972: C
973:          if ( LSTCK(0).lt.0 ) then
974:            write(IMG,7160) 'region volume setting.'
```

```

975:            call CNTERR( 'FATAL' )
976:          else
977:            call CRVOL1( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ,
978:            &      A(LKMAT) )
979:          end if
980:        else
981:          call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 0.0, JDEBG
982:          &      )
983:          write(IPR,7100)
984:          7100 format(/1X,<<MESSAGE>> Calculate region volumes from zone',
985:          &      ' volumes//')
986:          call CNTERR( 'MESSAGE' )
987: C
988:          if ( LSTCK(0).lt.0 ) then
989:            write(IMG,7160) 'region volume setting.'
990:            call CNTERR( 'FATAL' )
991:          else
992:            call CRVOL( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ )
993:          end if
994:        end if
995:      else
996:        call KEPV( A(1), 'VOL', LVOL, NINPZ, 'R4', LAST, 0.0, JDEBG )
997:        if ( LSTCK(0).lt.0 ) then
998:          write(IMG,7160) 'region volume setting.'
999:          call CNTERR( 'FATAL' )
1000:        else
1001:          call CRVOL1( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ,
1002:          &      A(LKMAT) )
1003:        end if
1004:      end if
1005:
1006:      if ( LTRVOL.eq.0 ) then
1007:        write(IMG,7120)
1008:        7120 format('// !!!(intro2) Tally-region volumes are calculated',
1009:        &      ' from those of composing regions.'//)
1010:        call CNTERR( 'WARNING' )
1011:        call KEEP( 'TRVOL', LTRVOL, NTREG, 'R4', LAST, JDEBG )
1012: C
1013:      if ( LSTCK(0).lt.0 ) then
1014:        write(IMG,7160) 'tally region volume setting.'
1015:        call CNTERR( 'FATAL' )
1016:      else
1017:        call CRVOL( A(LTRVOL), NTREG, A(LRVOL), NREG, A(LIPTRG),
1018:        &      A(LLPTRG) )
1019:      end if
1020:    end if
1021: C
1022:    if ( LSTCK(0).lt.0 ) then
1023:      write(IMG,7160) 'region volume printout.'
1024:      call CNTERR( 'FATAL' )
1025:    else
1026: C *****
1027:      call LABEL( IPR, 'VOLUMES OF EACH INPUT ZONE' )
1028: C *****
1029: C
1030:      call R4PRNT( 'VOL', A(LVOL), NINPZ, 1, 'INP.ZONE', ' ', IPR )
1031: C
1032: C *****
1033:      call LABEL( IPR, 'VOLUMES OF EACH REGION' )
1034: C *****
1035: C
1036:      call R4PRNT( 'RVOL', A(LRVOL), NREG, 1, 'REGION', ' ', IPR )
1037: C
1038: C *****
1039:      call LABEL( IPR, 'VOLUMES OF EACH TALLY-REGION' )
```

src/mvp/intro2.f

```

1040: C *****
1041: C
1042:       call R4PRNT( 'TRVOL', A(LTRVOL), NTREG, 1, 'T-REGION', ' ', IPR
1043: &                )
1044:       end if
1045: C
1046: C-----
1047: C Make number density table for each ZONE & REGION
1048: C-----
1049:       call KEPV( A(1), 'DNZON', LDNZON, NUC*NZONE*2, 'R4', LAST, 0.0,
1050: &                JDEBG )
1051:       call KEPV( A(1), 'DNREG', LDNREG, NUC*NREG*2, 'R4', LAST, 0.0,
1052: &                JDEBG )
1053: C
1054:       if ( LSTCK(0).lt.0 ) then
1055:         write(IMG,7140) 'number density table processing.'
1056:         call PRSTOP( 1, 'MEMORY OVER.' )
1057:         stop 999
1058:       end if
1059: C
1060:       call DENTB2( JNEUT, JPHOT, A(LDNZON), NUC, NZONE, A(LKZMAT),
1061: &                A(LINUCT), A(LDENST), A(LLPDEN), NMAT, CHA(LNUCID),
1062: &                A(LNATMT), NPATOM, A(LIATMT), A(LLPDNP), A(LDNSTP),
1063: &                A(LDNREG), A(LRVOL), A(LVOL), NREG, NINPZ, A(LKREG),
1064: &                A(LKMAT) )
1065: C-----
1066: C ..... PRINTOUT OF RESPONSE FUNCTION .....
1067: C-----
1068:       if ( JRESP.ne.0 .or. NRESP2.gt.0 .or. NSTAL2.gt.0 ) then
1069: C
1070: C *****
1071:       call LABEL( IPR,
1072: &                'RESPONSE FUNCTIONS FOR REACTION RATE CALCULATION' )
1073: C *****
1074: C
1075:       if ( NRESP.gt.0 ) then
1076:         call R4PRNT( 'RESP', A(LRESP), NGROUP, NRESP, 'GROUP',
1077: &                'RESPONSE', IPR )
1078:       end if
1079: C-----
1080: C ... read point-wise response functions
1081: C-----
1082:       if ( NSTAL.gt.0.and.NICRES.gt.0 ) then
1083: C
1084:       call KEEP( 'KCRES', LKCRES, NSTAL*4, 'I4', LAST, JDEBG )
1085:       call REMAIN( 'CRES', NLTEMP, 'R4', LAST )
1086:       if ( NLTEMP.le.0 ) then
1087:         write(IMG,'(1X,A,A)')
1088:         &                'XXX(intro2) No-more memory to store point-wise',
1089:         &                'responses !!!'
1090:         call CNTERR( 'FATAL' )
1091:       else
1092:         call KEEP( 'CRES', LCRES, NLTEMP, 'R4', LAST, JDEBG )
1093:         call XSECD( A(LICRES), A(LKCRES), A(LCRES), LCRES, LAST,
1094: &                LIMIT, NLTEMP, NSTAL, NICRES, MCRES )
1095: C
1096:       if ( JPTDT.ne.0 ) then
1097:         call STKCRS( A(LKCRES), A(LKCRSI), NSTAL )
1098: C ..... STKCRS in editfl.f
1099:       end if
1100: C
1101: C ... release of unused memory is carried out in XSECD ...
1102: C       call RESIZE( 'CRES', LCRES, MCRES, 'R4', LAST )
1103: C
1104:       end if

```

```

1105:
1106:       end if
1107:     end if
1108: C
1109: C
1110:       call GTLAST( LLS1 )
1111:       call LGTLST( LLL1 )
1112: C
1113: c##<2007/03/14:PN3:
1114: c## write(IPR,'(1X,A,I12,A,I12)')
1115: c## &                ' MEMORY FOR TALLY DATA (WORDS) : TASK SHARED ', LLS1
1116: c## &                ' - LLS0, ' TASK LOCAL ', LLL1 - LLL0
1117: c## write(IPR,'(1X,A,I12,A,I12,A)')
1118: c## &                ' MEMORY POSITION OF TALLY DATA: FROM A(' , LLS0,
1119: c## &                ' ) TO A(' , LLS1 - 1, ' )'
1120: c## write(IPR,'(1X,A,I12,A,I12,A)')
1121: c## &                ' MEMORY POSITION OF TALLY DATA: FROM H(' , LLL0,
1122: c## &                ' ) TO H(' , LLL1 - 1, ' )'
1123:       write(IPR,601) LLS1-LLS0, LLL1-LLL0, LLS0, LLS1-1, LLL0, LLL1-1
1124:       601 format(
1125: &/' memory for tally data (words) : task shared',i11,
1126: & ' : task local',i11
1127: &/' memory position of tally data : from A(' ,i11,') to A(' ,i11,')'
1128: &/' memory position of tally data : from H(' ,i11,') to H(' ,i11,')'
1129: & )
1130: c##>
1131: C
1132:       call MEMUSE( 'TALLY', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL, LIMITC
1133: &                )
1134: C
1135: C-----
1136: C ..... /PSOUR/ .....
1137: C-----
1138: C-----
1139: C
1140: C/#IF SOURCE(NEW)
1141: C
1142: C
1143: C ... lsrcsp = 0 : probably source input is in old-fashioned way.
1144: C       Convert to new type of source generation data
1145: C
1146:       if ( LSRCSP.eq.0 ) then
1147:         if ( JEIGN.ne.0.and.LIFISM.eq.0 ) then
1148:           write(IMG,'(1X,A,A)')
1149:           &                '!!!(intro2) Data "IFISM" is not specified.',
1150:           &                ' Set to 1.'
1151:           call CNTERR( 'WARNING' )
1152:           call KEPV( A(1), 'IFISM', LIFISM, NSOUR, 'I4', LAST, 1,
1153:           &                JDEBG )
1154:         end if
1155: C
1156: C       NSOUR may be increased when problem is neutron/photon
1157: C       coupled and PENRG(1:ngroup) has non zero elements for both
1158: C       of neutron and photon energy group.
1159: C
1160:       if ( LSTCK(0).lt.0 .or. LSTCKL(0).lt.0 ) then
1161:         write(IMG,7140) 'conversion of old type source input.'
1162:         call PRSTOP( 1, 'MEMORY OVER.' )
1163:         stop 999
1164:       end if
1165: C
1166:       call SRCINP( A, H, CHA, LIMIT, LIMITL, LIMITC, LSRCSP, NSRCSP,
1167: &                LLSOUR, LIDSRC, 'MVP', JKPAR, KPLIM, JFISS, JEIGN,
1168: &                JTIME, JNEUT, JPHOT, JUNDG, JDEBG, MWVEC, MVSTK, MXREJ,
1169: &                NERR, A(LIDMAT), CHA(LNUCID), NUC, NTGX, LKSOUR,

```

src/mvp/intro2.f

```

1170:      &          LISZON, LPSPAC, LPENRG, LIFISM, LFKAI, LENGYB, LENGPB,
1171:      &          EINCD )
1172: C
1173:      end if
1174: C
1175:      if ( NMEMS.eq.0 ) NMEMS = 5
1176:      call LKEPV( H(1), 'MEMZN', LMEMZN, NMEMS*NSOUR, 'I4', LAST, 0,
1177:      &          JDEBG )
1178:
1179: C/#ENDIF
1180:
1181: C-----
1182: C      .... INPUT RESTART FILE & DETERMINE SIZE OF KEFF-TALLY ARRAY
1183: C      Flag JREST may be fixed in auto-restart mode.
1184: C-----
1185: C
1186:      NTHIST = 0
1187:      NBATCH = 0
1188:      if ( JARST.ne.0 .or. JREST.ne.0 ) then
1189:      call NEST10( NTHIST, H(LWSUM), H(LWLEK), LRAND, NBATCH, NFISB,
1190:      &          NGROUP, NRESP, NREG, NTREG, NPKIND, NMOMO, NZONE,
1191:      &          NPART, NLATT, NEST, NUC, NEMIC, NEMAC, NSTAL, NETALY,
1192:      &          NETRV )
1193:      end if
1194: C
1195: C      =====
1196: C      call HEADER( IPR, 'Event stack, particle bank and working area' )
1197: C      =====
1198: C
1199: C-----
1200: C      check batch size and sub-batch size (added Feb 2000)
1201: C      NHSUB may be given an appropriate value if not sapecified.
1202: C-----
1203: C
1204:      NBANK1 = NBANK
1205:      NHSUB1 = NHSUB
1206:      NHIST1 = NHIST
1207:
1208:      if(JEXDP.eq.2.or.JEXDP.eq.3) then ! if exact reso-scat models are used
1209:      if(NBANK1.lt.2*NHIST1) then
1210:      NBANK = 2*NHIST
1211:      write(IMG,'(1X,A,A)')
1212:      &          '!!!(INTRO2) NBANK is small for exact resonance scattering'
1213:      &          ' models. NBANK is set to 2*NHIST.'
1214:      call CNTERR( 'WARNING' )
1215:      end if
1216:      end if
1217: C
1218:      call CKBATS( NHSUB, NHIST, NBANK )
1219: C
1220: C-----
1221: C      ... check and reset (if necessary) history control & bank parameters
1222: C      for multi tasking
1223: C-----
1224: C
1225:      NPART0 = NPART
1226:      if ( NTASK.gt.1 ) then
1227:      call TSKCTL( 'MVP', JEIGN, JWWND, JIMPT, JFISS, JREPR, NPART,
1228:      &          NTHIST, NBATCH, NHIST, NHSUB, NBANK, IMPMAX, NFBANK,
1229:      &          NFBANK0, NPART0, NHIST0, NHSUB0, NBANK0, NFBANK0 )
1230:      end if
1231: C
1232:      call KEEP( 'ITASK', LITASK, MNTASK*NTASK, 'I4', LAST, JDEBG )
1233:      call KEEP( 'IRANT', LIRANT, NTASK, 'I4', LAST, JDEBG )
1234:      call KEEP( 'IRSUT', LIRSUT, NTASK, 'I4', LAST, JDEBG )
1235:
1236:      call KEEP( 'IRSMT', LIRSMT, NTASK, 'I4', LAST, JDEBG )
1237: C-----
1238: C      .... /STACK/ /XBANK/.....
1239: C-----
1240: C
1241:      call GTLAST( LLS0 )
1242:      call LGTLST( LLL0 )
1243: C
1244:      if ( JNP.ne.0.and.(BANKP.le.0.0.or.BANKP.ge.1.0) ) then
1245:      BANKP = 0.5
1246:      write(IMG,'(1X,A,A,1P,E11.4,A)')
1247:      &          '!!!(intro2) "BANKP" (photon banking probability) is',
1248:      &          ' not specified or has an invalid value. ( BANKP = ',
1249:      &          BANKP, ' )'
1250:      write(IMG,'(1X,A,1P,E11.4)') ' SET: BANKP = ', BANKP
1251:      call CNTERR( 'WARNING' )
1252:      end if
1253: C
1254:      RRNB = 0.0
1255:      if ( NBANK1.gt.0.and.NHSUB1.gt.0 ) then
1256:      RRNB = MAX(1.0D0,DBLE(NBANK1)/NHSUB1)
1257:      end if
1258: C
1259:      if ( NBANK.eq.0 ) then
1260:      if ( RRNB.ne.0 ) then
1261:      NBANK = NINT(RRNB*NHSUB)
1262:      else if ( JNP.eq.0 ) then
1263:      CCCCC NBANK = NHIST
1264:      NBANK = NHSUB
1265:      else
1266:      CCCCC NBANK = NHIST/(1.0-BANKP)
1267:      NBANK = NHSUB/(1.0-BANKP)
1268:      end if
1269: C
1270:      if ( RRNB.eq.0.0 ) then
1271:      if ( JIMPT.ne.0.or.JWWND.ne.0.or.
1272:      &          (JFIXD.ne.0.and.JFISS.ne.0) ) then
1273:      NBANK = 2*NBANK
1274:      end if
1275:      end if
1276: C
1277:      write(IMG,'(1X,A,A,I10)')
1278:      &          '!!!(intor2) Bank length (NBANK) is not specified.',
1279:      &          ' Set to ', NBANK
1280:      call CNTERR( 'WARNING' )
1281: C
1282: C      else if ( JNP.ne.0.and.NBANK.le.NHIST/(1.0-BANKP) ) then
1283: C      else if ( JNP.ne.0.and.NBANK.le.NHSUB/(1.0-BANKP) ) then
1284: C
1285:      NBANK0 = NBANK
1286:      NBANK = NHIST / ( 1.0 - BANKP )
1287:      C
1288:      IF(JIMPT.NE.0.OR.JWWND.NE.0.OR.JFISS.NE.0) NBANK = 2*NBANK
1289:      C
1290:      C
1291:      C      WRITE(IPR,*) '!!! SPECIFIED BANK LENGTH MAY BE SMALL ',
1292:      C      &          'TO NEUTRON/PHOTON COUPLED PROBLEM. ',
1293:      C      &          ' CHANGED NBANK: ',NBANK0,' -> ',NBANK
1294:      call CNTERR( 'WARNING' )
1295:      end if
1296: C
1297: C      if ( NBANK.lt.NHIST ) then
1298: C      NBANK = NHIST
1299: C      if ( JIMPT.ne.0 .or. JWWND.ne.0 .or. JFISS.ne.0 ) then
1300: C      NBANK = 2*NHIST

```

src/mvp/intro2.f

```

1300: CC      end if
1301:      if ( NBANK.lt.NHSUB ) then
1302:          NBB0 = NBANK
1303:          NBANK = NHSUB
1304:          if ( RRNB.ne.0 ) then
1305:              NBANK = NINT(RRNB*NHSUB)
1306:          else if ( JIMPT.ne.0.or.JWWND.ne.0.or.
1307:      &          (JFIXD.ne.0.and.JFISS.ne.0) ) then
1308:              NBANK = 2*NHSUB
1309:          end if
1310:          write(IMG,'(1X,A,I9,A,I9,A,I9)')
1311:      &          '!!!intro2) Particle bank length (NBANK=', NBB0,
1312:      &          ') is smaller than sub-batch size (NHSUB=', NHSUB,
1313:      &          '). Set to ', NBANK
1314:          call CNTERR( 'WARNING' )
1315:      end if
1316: C
1317: C-----
1318: C ..... check NBNK2 if JUNDG is "ON" .....
1319: C-----
1320:      if ( JUNDG.ne.0 ) then
1321:          if ( NBNK2.eq.0 ) then
1322:              NBNK2 = MIN(500,NBANK)
1323:              write(IMG,'(1X,A,A,I7,A)')
1324:      &          '!!!intro2) Particle bank size for "underground".
1325:      &          ' modules (NBNK2) is not specified. Set it to ',
1326:      &          NBNK2, ' '.
1327:              call CNTERR( 'WARNING' )
1328:          end if
1329:      end if
1330:
1331: C-----
1332: C ..... IN THE CASE OF REFLECTIVE BOUNDARY .....
1333: C-----
1334:      if ( JREFL.ne.0 ) then
1335: C
1336:          call KEPV( A(1), 'KSREF', LKSREF, NZDA, 'I4', LAST, 0, JDEBG )
1337: C
1338: C ..... SEARCH REFLECTIVE BOUNDARIES .....
1339: C
1340:          call REMAIN( 'KKREF', NLTEMP, 'I4', LAST )
1341:          call KEEP( 'KKREF', LKKREF, NLTEMP, 'I4', LAST, JDEBG )
1342: C
1343:          if ( NLTEMP.le.0 ) then
1344:              write(IMG,'(1X,A,A)')
1345:      &          'XXX(intro2) No-more memory to store point-wise',
1346:      &          ' responses !!!'
1347:              call CNTERR( 'FATAL' )
1348:          else
1349: C
1350:              call REFZON( A(LSDA), A(LKZDA), A(LKZAA), A(LKZMAT),
1351:      &          A(LKSREF), NZDA, NZONE, A(LKKREF), NLTEMP, NREFS,
1352:      &          JDEBG )
1353: C
1354:          end if
1355: C
1356:          call RESIZE( 'KKREF', LKKREF, 2*NREFS, 'I4', LAST )
1357: C
1358: C .... reflection stack
1359: C
1360:          call LKEPV( H(1), 'NBREF', LNBREF, NREFS+1, 'I4', LAST, 0,
1361:      &          JDEBG )
1362:          call LKEPV( H(1), 'LSREF', LLSREF, NBANK, 'I4', LAST, 0, JDEBG
1363:      &          )
1364:          call LKEPV( H(1), 'IZREF', LIZREF, NBANK, 'I4', LAST, 0, JDEBG

```

```

1365:      &          )
1366:          call LKEPV( H(1), 'ISREF', LISREF, NBANK, 'I4', LAST, 0, JDEBG
1367:      &          )
1368: C
1369:      end if
1370: C
1371: C-----
1372: C KEEP MEMORY FOR EVENT STACKS
1373: C-----
1374: C
1375:          call LKEPV( H(1), 'LSFFL', LLSFFL, NBANK, 'I4', LAST, 0, JDEBG )
1376:          call LKEPV( H(1), 'NFFL', LNFFL, NZONE+1, 'I4', LAST, 0, JDEBG )
1377:          call LKEPV( H(1), 'IZFFL', LIZFFL, NBANK, 'I4', LAST, 0, JDEBG )
1378: C
1379:          if ( JNEUT.ne.0 ) then
1380:              call LKEPV( H(1), 'LSCOL', LLSCOL, NBANK, 'I4', LAST, 0, JDEBG
1381:      &          )
1382:              NCOLS = 0
1383:          end if
1384: C
1385:          if ( JPHOT.ne.0 ) then
1386:              call LKEPV( H(1), 'LSCLP', LLSCLP, NBANK, 'I4', LAST, 0, JDEBG
1387:      &          )
1388:              NCOLP = 0
1389:          end if
1390: C
1391:          call LKEPV( H(1), 'LSSRC', LLSSRC, NBANK, 'I4', LAST, 0, JDEBG )
1392:          call LKEPV( H(1), 'NNXT', LNNXT, NZONE+1, 'I4', LAST, 0, JDEBG )
1393:          call LKEPV( H(1), 'IZNXT', LIZNXT, NBANK, 'I4', LAST, 0, JDEBG )
1394:          IDEFER = 0
1395:          call LKEPV( H(1), 'LSDED', LLSDED, NBANK, 'I4', LAST, 0, JDEBG )
1396:          NDEAD = 0
1397: C
1398: C .... LATTICE SEARCH STACK .....
1399: C
1400:          if ( JLATT.ne.0 ) then
1401:              call LKEPV( H(1), 'LSLAT', LLSLAT, NBANK, 'I4', LAST, 0, JDEBG
1402:      &          )
1403:              call LKEPV( H(1), 'IZLAT', LIZLAT, NBANK, 'I4', LAST, 0, JDEBG
1404:      &          )
1405:              call LKEPV( H(1), 'NXLT', LNXLT, NLBZ+1, 'I4', LAST, 0, JDEBG )
1406:          end if
1407: C
1408: C-----
1409: C ..... IN THE CASE OF POINT DETECTOR .....
1410: C          ( STACK FOR IMAGINARY PARTICLES )
1411: C-----
1412: C
1413:          if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1414:              if ( IMPMAX.le.0 ) then
1415:                  write(IPR,'(1X,A,A,I10)')
1416:      &          '<<MESSAGE>> BANK LENGTH FOR IMAGINARY PARTICLES ',
1417:      &          'IS NOT SPECIFIED. SET TO 'NBANK' ! ', NBANK
1418:                  call CNTERR( 'MESSAGE' )
1419:                  IMPMAX = NBANK
1420:              end if
1421: C
1422:              NIMPT = 0
1423:              NDIMPT = IMPMAX
1424: C
1425:              call LKEPV(H(1),'IMSFL',LIMSFL, IMPMAX, 'I4', LAST, 0, JDEBG )
1426:              call LKEPV(H(1),'IMNFL',LIMNFL, NZONE+1, 'I4', LAST, 0, JDEBG )
1427:              call LKEPV(H(1),'IMZFL',LIMZFL, IMPMAX, 'I4', LAST, 0, JDEBG )
1428: C
1429:              call LKEPV(H(1),'IMSNX',LIMSNX, IMPMAX, 'I4', LAST, 0, JDEBG )

```

src/mvp/intro2.f

```

1430:      call LKEPV(H(1),'IMNNX',LIMNNX, NZONE+1, 'I4', LAST, 0, JDEBG )
1431:      call LKEPV(H(1),'IMZNX',LIMZNX, IMPMAX, 'I4', LAST, 0, JDEBG )
1432:      IMDEFR = 0
1433: C
1434:      if ( JLATT.ne.0 ) then
1435:          call LKEPV(H(1),'IMSLT',LIMSLT,IMPMAX,'I4', LAST, 0, JDEBG )
1436:          call LKEPV(H(1),'IMSLT',LIMNLT,NLBZ+1,'I4', LAST, 0, JDEBG )
1437:          call LKEPV(H(1),'IMZLT',LIMZLT,IMPMAX,'I4', LAST, 0, JDEBG )
1438:      end if
1439:      call LKEPV(H(1),'IMDED',LIMDED,IMPMAX,'I4',LAST, 0, JDEBG )
1440:  end if
1441: C
1442:      call GTLAST( LLS1 )
1443:      call LGTLST( LLL1 )
1444: C
1445: c##<2007/03/14:PN3:
1446: c## write(IPR, '(1X,A,I12,A,I12,)' )
1447: c## & ' MEMORY FOR STACK DATA (WORDS) : TASK SHARED ', LLS1
1448: c## & ' - LLS0, ' TASK LOCAL ', LLL1 - LLL0
1449: c## write(IPR, '(1X,A,I12,A,I12,A)' )
1450: c## & ' MEMORY POSITION OF STACK DATA: FROM A(' , LLS0,
1451: c## & ' ) TO A(' , LLS1 - 1, ' )'
1452: c## write(IPR, '(1X,A,I12,A,I12,A)' )
1453: c## & ' MEMORY POSITION OF STACK DATA: FROM H(' , LLL0,
1454: c## & ' ) TO H(' , LLL1 - 1, ' )'
1455:      write(IPR,602) LLS1-LLS0, LLL1-LLL0, LLS0, LLS1-1, LLL0, LLL1-1
1456: 602 format(
1457:  &' memory for stack data (words) : task shared',i11,
1458:  &' : task local',i11
1459:  &' memory position of stack data : from A(' ,i11,' ) to A(' ,i11,' )'
1460:  &' memory position of stack data : from H(' ,i11,' ) to H(' ,i11,' )'
1461:  & )
1462: c##>
1463:      call MEMUSE( 'EVENT STACK', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
1464:      &          LIMITC )
1465: C
1466: C-----
1467: C ..... /XBANK/ .....
1468: C-----
1469: C
1470:      call GTLAST( LLS0 )
1471:      call LGTLST( LLL0 )
1472: C
1473:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1474:          INBANK = NBANK + IMPMAX
1475:      else
1476:          INBANK = NBANK
1477:      end if
1478: C
1479:      call LKEPV( H(1), 'XXX', LXXX, INBANK, 'R8', LAST, OD0, JDEBG )
1480:      call LKEPV( H(1), 'YYY', LYXX, INBANK, 'R8', LAST, OD0, JDEBG )
1481:      call LKEPV( H(1), 'ZZZ', LZXX, INBANK, 'R8', LAST, OD0, JDEBG )
1482:      call LKEPV( H(1), 'AAA', LAAA, INBANK, 'R8', LAST, OD0, JDEBG )
1483:      call LKEPV( H(1), 'BBB', LBBB, INBANK, 'R8', LAST, OD0, JDEBG )
1484:      call LKEPV( H(1), 'CCC', LCCC, INBANK, 'R8', LAST, OD0, JDEBG )
1485:      call LKEPV( H(1), 'WWW', LWWW, INBANK, 'R4', LAST, OE0, JDEBG )
1486:      call LKEPV( H(1), 'EEE', LEEE, INBANK, 'R4', LAST, OE0, JDEBG )
1487:      call LKEPV( H(1), 'KKP', LKKP, INBANK, 'I4', LAST, 0, JDEBG )
1488:      call LKEPV( H(1), 'IZZ', LIZZ, INBANK, 'I4', LAST, 0, JDEBG )
1489:      call LKEPV( H(1), 'IGG', LIGG, INBANK, 'I4', LAST, 0, JDEBG )
1490:
1491:      call LKEPV( H(1), 'TTT', LTTT, INBANK, 'R8', LAST, OD0, JDEBG )
1492:      if ( JTIME.ne.0 ) then
1493:          call LKEPV( H(1), 'ITT', LITT, INBANK, 'I4', LAST, 0, JDEBG )
1494:      end if

1495: C
1496:      call LKEPV( H(1), 'KLSF', LKLSF, INBANK, 'I4', LAST, 0, JDEBG )
1497: C
1498:      if ( JIMPT.ne.0 ) then
1499:          call LKEPV( H(1), 'XIM', LXIM, NBANK, 'R4', LAST, OE0, JDEBG )
1500:      end if
1501:      if ( JPRT.ne.0 ) then
1502:          NGSP1 = NGSP + 1
1503: c+beff2
1504:          NOCS = (NOPSDS*NGSP+1+NORDDSD)*NPTDS
1505:          NTPT = NOCS+(NGSP+2)*NPTCS
1506:          if ( NPTBE.gt.0 ) NTPT = NTPT+(NGSP+2) ! beff with correlated samplin
1507: g
1508:          if ( NPTBE.gt.0 ) NTPT = NTPT+(1+NGSP) ! beff with diff.-oper. samp.
1509:          if ( NPTBE.gt.0 ) NTPT = NTPT+(1+NGSP) ! lambda with diff.-oper. samp
1509: c-beff2
1510:      call LKEPV( H(1), 'WWD ', LWWD , NBANK*NPTDS*NORDDSD, 'R8',
1511:      &          LAST, OD0, JDEBG )
1512:      call LKEPV( H(1), 'WWD2 ', LWWD2 , NBANK*NPTDS, 'R8',
1513:      &          LAST, OD0, JDEBG )
1514:      call LKEPV( H(1), 'WSD ', LWSL , NBANK*NGSP1*NPTDS*NOPSDS,
1515:      &          'R8', LAST, OD0, JDEBG )
1516:      call LKEPV( H(1), 'WWR ', LWWL , NBANK*NPTCS, 'R8',
1517:      &          LAST, OD0, JDEBG )
1518:      call LKEPV( H(1), 'WSC ', LWSL , NBANK*NGSP1*NPTCS, 'R8',
1519:      &          LAST, OD0, JDEBG )
1520:      call LKEPV( H(1), 'WWT ', LWWT , NBANK*NPTDS, 'R8',
1521:      &          LAST, OD0, JDEBG )
1522:      call LKEPV( H(1), 'WCTR', LWCTR, 7*NTPT*NUCPT, 'R8',
1523:      &          LAST, OD0, JDEBG )
1524: c+beff2
1525:      if ( NPTBE.gt.0 ) then
1526:          call LKEPV( H(1), 'WWB ', LWWB , NBANK , 'R8',
1527:      &          LAST, OD0, JDEBG )
1528:          call LKEPV( H(1), 'WSB ', LWSB , NBANK*NGSP1, 'R8',
1529:      &          LAST, OD0, JDEBG )
1530:          call LKEPV( H(1), 'WWBD ', LWWD , NBANK , 'R8',
1531:      &          LAST, OD0, JDEBG )
1532:          call LKEPV( H(1), 'WSBD ', LWSL , NBANK*NGSP, 'R8',
1533:      &          LAST, OD0, JDEBG )
1534:          call LKEPV( H(1), 'WWLD ', LWWD , NBANK , 'R8',
1535:      &          LAST, OD0, JDEBG )
1536:          call LKEPV( H(1), 'WSLD ', LWSL , NBANK*NGSP, 'R8',
1537:      &          LAST, OD0, JDEBG )
1538:      end if
1539: c-beff2
1540: C---+---1---+---2---+---3---+---4---+---5---+---6---+---7---
1541:      end if
1542: C-----
1543: C..... BANK DATA FOR LATTICE GEOMETRY .....
1544: C-----
1545:      if ( JLATT.ne.0 ) then
1546:          call LKEPV( H(1), 'LEVL', LLEVL, INBANK, 'I4', LAST, 0, JDEBG )
1547:          call LKEPV( H(1), 'LZZ', LLZZ, NBANK*NEST, 'I4', LAST, 0, JDEBG
1548:      &          )
1549:          call LKEPV( H(1), 'LPOS', LLPOS, NBANK*NEST, 'I4', LAST, 0,
1550:      &          JDEBG )
1551:          if ( JHLAT.ne.0 ) then
1552:              call LKEPV( H(1), 'LCRS', LLCRS, NBANK*NEST, 'I4', LAST, 0,
1553:      &          JDEBG )
1554:          end if
1555:          if ( NPDET.gt.0 ) then
1556:              call LKEPV(H(1),'LZZI',LLZZI,IMPMAX*NEST,'I4',LAST,0,JDEBG
1557:      &          )

```

src/mvp/intro2.f

```

1558:      call LKEPV(H(1),'LPOSI',LLPOSI, IMPMAX*NEST, 'I4', LAST, 0,
1559:      &          JDEBG )
1560:      if ( JHLAT.ne.0 ) then
1561:        call LKEPV(H(1),'LCRSI',LLCRSI,IMPMAX*NEST,'I4',LAST, 0,
1562:      &          JDEBG )
1563:      end if
1564:    end if
1565:  end if
1566:  if ( JTLLT.ne.0 ) then
1567:    N1 = NEST + 1
1568:    call LKEPV( H(1), 'IBSPC', LIBSPC, NBANK*N1, 'I4', LAST, 0,
1569:  &          JDEBG )
1570:  end if
1571: C
1572: C .. always keep IBREG (from Jan 2000)
1573: C for convenience to new features
1574: C
1575: call LKEPV( H(1), 'IBREG', LIBREG, NBANK, 'I4', LAST, 0, JDEBG )
1576: C
1577: C-----
1578: C ..... FISSION BANK .....
1579: C-----
1580: C
1581: if ( JEIGN.ne.0 ) then
1582:   if ( NFBANK.eq.0 ) then
1583:     NFBANK = NHIST
1584:     NFBNK0 = NFBANK
1585:     write(IPR,'(1X,A,A,I10)')
1586:   &     '<<MESSAGE>> FISSION BANK LENGTH (NFBANK) IS NOT',
1587:   &     ' SPECIFIED. SET TO ', NFBANK
1588:     call CNTERR( 'MESSAGE' )
1589:   else if ( NFBANK.lt.NHIST ) then
1590:     write(IMG,'(1X,A,I9,A,I9,A)')
1591:   &     ' !!!(intro2) Fission bank length (NFBANK=', NFBANK,
1592:   &     ') is smaller than batch size (NHIST=', NHIST, ') '
1593:     call CNTERR( 'WARNING' )
1594:   end if
1595:
1596:   if ( NFBNK0.eq.0 ) NFBNK0 = NFBANK
1597:
1598:   call LKEPV( H(1), 'XXXF', LXXXF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1599:   &          )
1600:   call LKEPV( H(1), 'YYF', LYYF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1601:   &          )
1602:   call LKEPV( H(1), 'ZZF', LZZF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1603:   &          )
1604:   call LKEPV( H(1), 'IZZF', LIZZF, NFBNK0, 'I4', LAST, 0, JDEBG )
1605:   call LKEPV( H(1), 'KLSFF', LKLSFF, NFBNK0, 'I4', LAST, 0, JDEBG
1606:   &          )
1607:   call LKEPV( H(1), 'EEEF', LEEEF, NFBNK0, 'R4', LAST, 0E0, JDEBG
1608:   &          )
1609:   call LKEPV( H(1), 'INUF', LINUF, NFBNK0, 'I4', LAST, 0, JDEBG )
1610:   call LKEPV( H(1), 'IMTF', LIMTF, NFBNK0, 'I4', LAST, 0, JDEBG )
1611:
1612:   if ( JLATT.ne.0 ) then
1613:     call LKEPV( H(1), 'LEVL', LLEVL, NFBNK0, 'I4', LAST, 0,
1614:   &          JDEBG )
1615:     call LKEPV( H(1), 'LZZF', LLZZF, NFBNK0*NEST, 'I4', LAST, 0,
1616:   &          JDEBG )
1617:     call LKEPV( H(1), 'LPOSF', LLPOSF, NFBNK0*NEST, 'I4', LAST,
1618:   &          0, JDEBG )
1619:     if ( JHLAT.ne.0 ) then
1620:       call LKEPV( H(1), 'LCRSF', LLCRSF, NFBNK0*NEST, 'I4',
1621:   &          LAST, 0, JDEBG )
1622:     end if

```

```

1623:   end if
1624:   if ( JTLLT.ne.0 ) then
1625:     call LKEPV( H(1), 'IBSPF', LIBSPF, NFBNK0*NEST, 'I4', LAST,
1626:   &          0, JDEBG )
1627:   end if
1628: C
1629: C ... always keep IBRGF ...
1630: C
1631: call LKEPV( H(1), 'IBRGF', LIBRGF, NFBNK0, 'I4', LAST, 0, JDEBG
1632: &          )
1633: C
1634: C ... IFISB(NHIST) : pointer to fission bank selected for
1635: C source of a batch.
1636: C Sources are selected at the beginning of each batch.
1637: C This is to manage sub-batches in eigenvalue calculation
1638: C
1639: call LKEPV( H(1), 'IFISB', LIFISB, NHIST, 'I4', LAST, 0, JDEBG
1640: &          )
1641: C
1642: C
1643: C ... when NHSUB < NHIST, keep memory to copy fission bank,
1644: C
1645: if ( NHSUB.lt.NHIST ) then
1646:   call LKEPV( H(1), 'XXXF2', LXXXF2, NHIST, 'R8', LAST, 0D0,
1647:   &          JDEBG )
1648:   call LKEPV( H(1), 'YYF2', LYYF2, NHIST, 'R8', LAST, 0D0,
1649:   &          JDEBG )
1650:   call LKEPV( H(1), 'ZZF2', LZZF2, NHIST, 'R8', LAST, 0D0,
1651:   &          JDEBG )
1652:   call LKEPV( H(1), 'IZZF2', LIZZF2, NHIST, 'I4', LAST, 0,
1653:   &          JDEBG )
1654:   call LKEPV( H(1), 'KLSFF2', LKLSFF2, NHIST, 'I4', LAST, 0,
1655:   &          JDEBG )
1656:   call LKEPV( H(1), 'EEEF2', LEEEF2, NHIST, 'R4', LAST, 0E0,
1657:   &          JDEBG )
1658:   call LKEPV( H(1), 'INUF2', LINUF2, NHIST, 'I4', LAST, 0,
1659:   &          JDEBG )
1660:   if ( JLATT.ne.0 ) then
1661:     call LKEPV( H(1), 'LEVL2', LLEVL2, NHIST, 'I4', LAST,
1662:   &          0, JDEBG )
1663:     call LKEPV( H(1), 'LZZF2', LLZZF2, NHIST*NEST, 'I4',
1664:   &          LAST, 0, JDEBG )
1665:     call LKEPV( H(1), 'LPOSF2', LLPOSF2, NHIST*NEST, 'I4',
1666:   &          LAST, 0, JDEBG )
1667:     if ( JHLAT.ne.0 ) then
1668:       call LKEPV( H(1), 'LCRSF2', LLCRSF2, NHIST*NEST, 'I4',
1669:   &          LAST, 0, JDEBG )
1670:     end if
1671:   end if
1672:   call LKEPV( H(1), 'IBRGF2', LIBRGF2, NHIST, 'I4', LAST, 0,
1673:   &          JDEBG )
1674:   if ( JTLLT.ne.0 ) then
1675:     call LKEPV( H(1), 'IBSPF2', LIBSPF2, NHIST*NEST, 'I4',
1676:   &          LAST, 0, JDEBG )
1677:   end if
1678: end if
1679: C
1680: C
1681: C
1682: if ( JEIGN.ne.0 .and. JPRT.ne.0 ) then
1683:   call LKEPV( H(1), 'WD0', 'LWD0', NFBNK0*NGSP1*NPTDS, 'R8',
1684:   &          LAST, 0D0, JDEBG )
1685:   call LKEPV( H(1), 'WD0', 'LWD0', NFBNK0*NGSP1*NPTDS*NOPSDS,
1686:   &          'R8', LAST, 0D0, JDEBG )
1687:   call LKEPV( H(1), 'WR0', 'LWR0', NFBNK0*NGSP1*NPTCS, 'R8',

```


src/mvp/intro2.f

```

1688:      &          LAST, OD0, JDEBG )
1689:      call LKEPV(H(1),'WTO ',LWTO ,NFBNK0,'R8', LAST, OD0, JDEBG )
1690:      call LKEPV(H(1),'WTOA ',LWTOA ,NFBNK0,'R8', LAST, OD0, JDEBG )
1691:      call LKEPV(H(1),'WTOB ',LWTOB ,NFBNK0,'R8', LAST, OD0, JDEBG )
1692: c+beff2
1693:      if ( NPTBE.gt.0 ) then
1694:          call LKEPV( H(1),'WB0 ',LWB0 ,NFBNK0*NGSP1,'R8',
1695:              &          LAST, OD0, JDEBG )
1696:          call LKEPV( H(1),'WBD0 ',LWBD0 ,NFBNK0*NGSP,'R8',
1697:              &          LAST, OD0, JDEBG )
1698:          call LKEPV( H(1),'WLD0 ',LWLD0 ,NFBNK0*NGSP,'R8',
1699:              &          LAST, OD0, JDEBG )
1700:      end if
1701: c-beff2
1702:      end if
1703: C-----1-----2-----3-----4-----5-----6-----7---
1704: C
1705:      if ( JEIGN.eq.0.and.NSKIP.ne.0 ) then
1706:          write(IMG,'(1X,A)')
1707:          &          '!!!(intro2) "NSKIP" is not zero in fixed source',
1708:          &          'problem. NSKIP is set to 0'
1709:          call CNTERR( 'WARNING' )
1710:          NSKIP = 0
1711:      end if
1712: C
1713: C
1714: C-----
1715: C..... optional bank data .....
1716: C-----
1717: C
1718: C ... setting of custom BANK data ...
1719: C      K[DI][BNK|FBK](11:16) is currently reserved for users.
1720: C
1721:      call ZZBANK( KDBNK, MDBNK, KIBNK, MIBNK, KDFBK, MDFBK, KIFBK,
1722:          &          MIFBK )
1723: C
1724: C..... floating point optional bank data .....
1725: C
1726: C ... save particle weight on birth
1727:      if ( JRWVR.ne.0 ) then
1728:          KDBNK(1) = 1
1729:      end if
1730: C
1731: C ... save distance to lattice frames (NEST+1 banks)
1732: C      and position on frame(to be reached) and direction
1733:      if ( JFISX.ne.0 ) then
1734:          KDBNK(4) = NEST + 1
1735:          KDBNK(6) = 6*NEST
1736:      end if
1737: C
1738: C ... save position of STG cell (3*NEST banks)
1739:      if ( JSTGR.ne.0 ) then
1740:          KDBNK(5) = 3*NEST
1741:      end if
1742: C
1743: C ... for surface tallies (distance and angle)
1744:      if ( JTSRF.ne.0 ) then
1745:          KDBNK(7) = 2*NTSRF
1746:      end if
1747: C
1748:      NNND = 0
1749:      do 110 K = 1, MDBNK
1750:          if ( KDBNK(K).ne.0 ) then
1751:              NNND1 = NNND
1752:              NNND = NNND + KDBNK(K)

```

```

1753:          KDBNK(K) = NNND1 + 1
1754:      end if
1755: 110 continue
1756:      KDBNK(0) = NNND
1757:      call LKEPV( H(1), 'DBNK', LDBNK, INBANK*NNND, 'R8', LAST, OD0,
1758:          &          JDEBG )
1759: C
1760: C..... integer optional bank data .....
1761: C
1762: C
1763: C ... save particle generation number
1764: C      (another case of non zero KIBNK(4) is GENERATION tally.
1765: C      see TALINP routine.)
1766: C
1767:      if ( MXPGEN.gt.0 ) then
1768:          KIBNK(4) = 1
1769:      end if
1770: C
1771:      if ( JFISX.ne.0 ) then
1772: C ... save flight count in a path
1773:          KIBNK(5) = 1
1774: C ... lattice level giving the smallest distance until the level
1775:          KIBNK(6) = NEST
1776: C ... required NND type
1777:          KIBNK(7) = 1
1778:      end if
1779: C
1780:      if ( JDEBG(6).gt.0 ) then
1781:          KIBNK(8) = JDEBG(6)
1782:      end if
1783: C
1784: C ... marker region passing flag
1785: C
1786:      if ( KIBNK(9).gt.0 ) then
1787:          KIBNK(9) = NMKREG
1788:      end if
1789: C
1790: C ... for surface tallies (flag and angle bin)
1791:      if ( JTSRF.ne.0 ) then
1792:          KIBNK(10) = 2*NTSRF
1793: C ... save flight count in a path
1794:          KIBNK(5) = 1
1795:      end if
1796: C
1797: c+beff1+beff2
1798:      if ( JBEFF.ne.0 .or. NPTBE.gt.0 ) then
1799:          KIBNK(20) = 1
1800:      end if
1801: c-beff1+beff2
1802: C
1803:      NNNI = 0
1804:      do 120 K = 1, MIBNK
1805:          if ( KIBNK(K).ne.0 ) then
1806:              NNNI1 = NNNI
1807:              NNNI = NNNI + KIBNK(K)
1808:              KIBNK(K) = NNNI1 + 1
1809:          end if
1810: 120 continue
1811:          KIBNK(0) = NNNI
1812:          call LKEPV( H(1), 'IBNK', LIBNK, INBANK*NNNI, 'I4', LAST, 0,
1813:              &          JDEBG )
1814: c##<2007/03/14:PN4:
1815: C
1816:      if ( JPHNU.ne.0 ) then
1817:          call LKEPV( H(1), 'KPNFG', LKPNFG, INBANK, 'I4', LAST, 0,

```

src/mvp/intro2.f

```

1818:      &      JDEBG )
1819:      end if
1820: c##>
1821: C
1822: C-----
1823: C ... optional fission bank data ...
1824: C-----
1825: C
1826:      if ( JEIGN.ne.0 ) then
1827: C
1828: C ... save position of STG cell (3*NEST banks)
1829: C
1830:      if ( JSTGR.ne.0 ) then
1831:      KDFBK(5) = 3*NEST
1832:      end if
1833: C
1834:      NNNDF = 0
1835:      do 130 K = 1, MDFBK
1836:      if ( KDFBK(K).ne.0 ) then
1837:      NNNDF = NNNDF
1838:      NNNDF = NNNDF + KDFBK(K)
1839: C990618      KDFBK(K) = NNNDF
1840:      KDFBK(K) = NNNDF + 1
1841:      end if
1842: 130      continue
1843:      KDFBK(0) = NNNDF
1844:      call LKEPV( H(1), 'DFBK', LDFBK, NFBNK0*NNNDF, 'R8', LAST, 0D0,
1845:      &      JDEBG )
1846: C
1847: C ... lattice level giving the smallest distance until the level
1848: C      if ( JFISX.ne.0 ) then
1849: C      KIFBK(6) = 1
1850: C      end if
1851: C
1852:      NNNIF = 0
1853:      do 140 K = 1, MIFBK
1854:      if ( KIFBK(K).ne.0 ) then
1855:      NNNIF = NNNIF
1856:      NNNIF = NNNIF + KIFBK(K)
1857:      KIFBK(K) = NNNIF + 1
1858:      end if
1859: 140      continue
1860:      KIFBK(0) = NNNIF
1861:      call LKEPV( H(1), 'IFBK', LIFBK, NFBNK0*NNNIF, 'I4', LAST, 0,
1862:      &      JDEBG )
1863: C
1864: C
1865:      if ( MDFBK2.ne.MDFBK .or. MIFBK.ne.MIFBK2 ) then
1866:      write(IMSG,'(1X,A,A)')
1867:      &      'XXX(intro2) MDFBK2.ne.MDFBK .or. MIFBK.ne.MIFBK2',
1868:      &      '!! Check parameter definition in header file.'
1869:      stop 666
1870:      end if
1871: C
1872:      if ( NHSUB.lt.NHIST ) then
1873:      do 150 K = 1, MDFBK
1874:      KDFBK2(K) = KDFBK(K)
1875: 150      continue
1876:      KDFBK2(0) = KDFBK(0)
1877:      call LKEPV( H(1), 'DFBK2', LDFBK2, NHIST*KDFBK2(0), 'R8',
1878:      &      LAST, 0D0, JDEBG )
1879:      do 160 K = 1, MIFBK
1880:      KIFBK2(K) = KIFBK(K)
1881: 160      continue
1882:      KIFBK2(0) = KIFBK(0)

```

```

1883:      call LKEPV( H(1), 'IFBK2', LIFBK2, NHIST*KIFBK2(0), 'I4',
1884:      &      LAST, 0, JDEBG )
1885:      end if
1886:      end if
1887: C
1888: C-----
1889: C..... BANK DATA FOR IMAGINARY PARTICLE FOR NEXT EVENT ESTIMATOR ...
1890: C-----
1891:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1892:      call KEEP2( 'XXXI', LXXX, NBANK, 'R8', LXXXI, JDEBG )
1893:      call KEEP2( 'YYYI', LYYY, NBANK, 'R8', LYYYI, JDEBG )
1894:      call KEEP2( 'ZZZI', LZZZ, NBANK, 'R8', LZZZI, JDEBG )
1895:      call KEEP2( 'AAAI', LAAA, NBANK, 'R8', LAAAI, JDEBG )
1896:      call KEEP2( 'BBBI', LBBB, NBANK, 'R8', LBBBI, JDEBG )
1897:      call KEEP2( 'CCCI', LCCC, NBANK, 'R8', LCCCI, JDEBG )
1898:      call KEEP2( 'WWWI', LWWW, NBANK, 'R4', LWWWI, JDEBG )
1899:      call KEEP2( 'EEEI', LEEE, NBANK, 'R4', LEEEI, JDEBG )
1900:      call KEEP2( 'IZZI', LIZZ, NBANK, 'I4', LIZZI, JDEBG )
1901:      call KEEP2( 'IGGI', LIGG, NBANK, 'I4', LIGGI, JDEBG )
1902:      if ( JTIME.ne.0 ) then
1903:      call KEEP2( 'ITTI', LITT, NBANK, 'I4', LITTI, JDEBG )
1904:      end if
1905:      call KEEP2( 'TTTI', LTTT, NBANK, 'R8', LTTTI, JDEBG )
1906:      call KEEP2( 'KLSFI', LKLSF, NBANK, 'I4', LKLSFI, JDEBG )
1907:      call KEEP2( 'DBNKI', LDBNK, NBANK*NNND, 'R8', LDBNKI, JDEBG )
1908:      call KEEP2( 'IBNKI', LIBNK, NBANK*NNNI, 'I4', LIBNKI, JDEBG )
1909:      if ( JLATT.ne.0 ) then
1910:      call KEEP2( 'LEVLI', LLEVL, NBANK, 'I4', LLEVLI, JDEBG )
1911:      end if
1912: C
1913:      call LKEPV(H(1),'KDETP',LKDETP,IMPMAX, 'I4', LAST, 0, JDEBG )
1914:      call LKEPV(H(1),'OPTI',LOPTI,IMPMAX, 'R8', LAST, 0.0D0, JDEBG )
1915:      call LKEPV(H(1),'PATH',LPATH,IMPMAX, 'R8', LAST, 0.0D0, JDEBG )
1916:      end if
1917: C
1918: C-----
1919: C..... SIGMA BANK /SBANK/ .....
1920: C-----
1921: C
1922:      if ( MB.eq.0 ) then
1923:      write(IPR,'(1X,A,A,1X,A,I8,A)')
1924:      &      '<<MESSAGE>> NUMBER OF MATERIALS STORED IN SIGMA BANK',
1925:      &      ' (MB) IS NOT INPUT.',
1926:      &      ' SET TO THE NUMBER OF MATERIALS (NMAT= ',
1927:      &      NMAT, ' )'
1928:      MB = NMAT
1929:      call CNTERR( 'MESSAGE' )
1930:      end if
1931: C
1932:      call LKEPV( H(1), 'SMAC', LSMAC, NBANK*MB*NSMAC, 'R4', LAST, 0E0,
1933:      &      JDEBG )
1934:      call LKEPV( H(1), 'KMAC', LKMAC, NBANK*MB*2, 'I4', LAST, 0, JDEBG
1935:      &      )
1936:      call LKEPV( H(1), 'MMAC', LMMAC, NBANK*2, 'I4', LAST, 0, JDEBG )
1937:      call LKEPV( H(1), 'SMIC', LSMIC, NBANK*NUC*NSMIC, 'R4', LAST, 0E0,
1938:      &      JDEBG )
1939:      call LKEPV( H(1), 'KSPI', LKSPI, NBANK*NUC, 'I4', LAST, 0, JDEBG )
1940:      call LKEPV( H(1), 'SGTAL', LSGTAL, NBANK*NSTAL, 'R4', LAST, 0E0,
1941:      &      JDEBG )
1942: C      if ( JFMUL.eq.1 ) then
1943:      c##<2007/03/14:PN3:
1944:      c##      call LKEPV( H(1), 'RNU', LRNU, NBANK*NUC*3, 'R4', LAST, 0E0,
1945:      c##      &      JDEBG )
1946:      NUC_MAX = max(NUC, NUCPN)
1947:      call LKEPV( H(1), 'RNU', LRNU, NBANK*NUC_MAX*3, 'R4', LAST, 0E0,

```

src/mvp/intro2.f

```

1948:      &          JDEBG )
1949:      if ( JPHNU.ne.0 ) then
1950:        call LKEPV( H(1), 'SMICPN', LSMICPN, NBANK*NUCPN*NSMICPN, 'R4',
1951:          &          LAST, OE0, JDEBG )
1952:      end if
1953: c##>
1954: C      end if
1955:      if ( JPERT.ne.0 ) then
1956:        call LKEPV( H(1), 'SMACP', LSMACP, NBANK*MB*NSMAC, 'R4', LAST,
1957:          &          OE0, JDEBG )
1958:        call LKEPV( H(1), 'SMICP', LSMICP, NBANK*NUC*NSMIC, 'R4', LAST,
1959:          &          OE0, JDEBG )
1960:      end if
1961:
1962: C-----
1963: C..... SIGMA BANK FOR IMAGINARY PARTICLE FOR NEXT EVENT ESTIMATOR ...
1964: C-----
1965:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1966:        call LKEPV(H(1), 'SMACI', LSMACI, IMPMAX*MB*NSMACI, 'R4',
1967:          &          LAST, OE0, JDEBG )
1968:        call LKEPV(H(1), 'KMACI', LKMACI, IMPMAX*MB, 'I4', LAST, 0,
1969:          &          JDEBG )
1970:        call LKEPV(H(1), 'MMACI', LMMACI, IMPMAX*2, 'I4', LAST, 0,
1971:          &          JDEBG )
1972:        call LKEPV(H(1), 'SMICI', LSMICI, IMPMAX*NUC*NSMICI, 'R4',
1973:          &          LAST, OE0, JDEBG )
1974:        call LKEPV(H(1), 'KSPII', LKSPII, IMPMAX*NUC, 'I4', LAST, 0,
1975:          &          JDEBG )
1976:        call LKEPV(H(1), 'SGTLI', LSGTLI, IMPMAX*NSTAL, 'R4', LAST,
1977:          &          OE0, JDEBG )
1978:      end if
1979: C
1980: C .....
1981: C
1982:      call GTLAST( LLS1 )
1983:      call LGTLST( LLL1 )
1984: C
1985: c##<2007/03/14:PN3:
1986: c##      write(IPR,'(/1X,A,I12,A,I12)')
1987: c## &      ' MEMORY FOR BANK DATA (WORDS) : TASK SHARED ', LLS1
1988: c## &      - LLS0, ' TASK LOCAL ', LLL1 - LLL0
1989: c##      write(IPR,'(1X,A,I12,A,I12,A)')
1990: c## &      ' MEMORY POSITION OF BANK DATA: FROM A(' , LLS0, ' ) TO A('
1991: c## &      LLS1 - 1, ' )'
1992: c##      write(IPR,'(1X,A,I12,A,I12,A)')
1993: c## &      ' MEMORY POSITION OF BANK DATA: FROM H(' , LLL0, ' ) TO H('
1994: c## &      LLL1 - 1, ' )'
1995: c##      write(IPR,603) LLS1-LLS0, LLL1-LLL0, LLS0, LLS1-1, LLL0, LLL1-1
1996: 603 format(
1997: &/' memory for bank data (words) : task shared',i11,
1998: &' : task local',i11
1999: &/' memory position of bank data : from A(' ,i11,') to A(' ,i11,')'
2000: &/' memory position of bank data : from H(' ,i11,') to H(' ,i11,')'
2001: & )
2002: c##>
2003:      call MEMUSE( 'PARTICLE BANK', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
2004:        &          LIMITC )
2005: C
2006: C .....
2007: C      write(IPR,'(/1X,A,I12,A)')
2008: C &      ' LAST POSITION OF MEMORY EXCEPT WORKING AREA = ', LAST,
2009: C &      ' (WORD)'
2010: C
2011: C-----
2012: C      .... INPUT RESTART FILE & DETERMINE SIZE OF KEFF-TALLY ARRAY

```

```

2013: C-----
2014: C
2015: C      if ( JREST.ne.0 ) then
2016: C        call RESTIO( NTHIST, H(LWSUM), H(LWLEK), IRAND, NBATCH, NFISB,
2017: C          &          NGROUP, NRESP, NREG, NTREG, NPKIND, NMEMO, NZONE,
2018: C          &          NPART, NLATT, NEST, NUC, NEMIC, NEMAC, NSTAL, NETALY )
2019: C      else
2020: C        NTHIST = 0
2021: C        NBATCH = 0
2022: C      end if
2023: C
2024: C      NPART > 0 : IS ABSOLUTE HISTORY NUMBER.
2025: C      NPART < 0 : RUN NTHIST + ABS(NPART) HISTORIES.
2026: C
2027: C      if ( NPART.gt.0 ) then
2028: C        if ( JEIGN.ne.0 ) then
2029: C          if ( NPART.le.NTHIST ) then
2030: C            NLENG = NBATCH
2031: C          else
2032: C            NLENG = NBATCH + (NPART-NTHIST-1) /NHIST + 1
2033: C          end if
2034: C        end if
2035: C      else if ( NPART.lt.0 ) then
2036: C        if ( JEIGN.ne.0 ) NLENG = NBATCH + (ABS(NPART)-1) /NHIST + 1
2037: C        NPART = NTHIST + ABS(NPART)
2038: C      else
2039: C        write(IMG,'(/A,A)')
2040: C        &        'XXX(intro2) TOTAL NUMBER OF HISTORIES (NPART) IS',
2041: C        &        ' NOT SPECIFIED OR INPUT AS ZERO.'
2042: C        call CNTERR( 'FATAL' )
2043: C      end if
2044: C
2045: C
2046: C      if ( JEIGN.ne.0.and.NLENG.lt.NSKIP+2 ) then
2047: C        write(IMG,'(/1X,A,I4,A,A)')
2048: C        &        '!!!intro2) Number of batches to be run (', NLENG,
2049: C        &        ' ) is too small ',
2050: C        &        ' to calculate standard deviations of some results.'
2051: C        write(IMG,'(1X,A)') ' It should be greater than "NSKIP"+2.'
2052: C        call CNTERR( 'WARNING' )
2053: C        if ( NSKIP.ne.0 ) then
2054: C          NSKIP2 = NLENG - 2
2055: C          if ( NSKIP2.ge.0 ) then
2056: C            write(IMG,'(1X,A,A,I4,A)')
2057: C            &            '!!!intro2) Number of skipped batch for tally',
2058: C            &            ' (NSKIP) is changed to ', NSKIP2,
2059: C            &            ' to enable calculation of standard deviation.'
2060: C            call CNTERR( 'WARNING' )
2061: C            NSKIP = NSKIP2
2062: C          end if
2063: C        end if
2064: C      end if
2065: C
2066: C
2067: C      if ( JEIGN.ne.0 ) then
2068: C        call LKEPV( H(1), 'XSOC', LXSOC, NLENG, 'R8', LAST, OD0, JDEBG
2069: C          &          )
2070: C        call LKEPV( H(1), 'XSXV', LXSXV, NLENG*3, 'R8', LAST, OD0,
2071: C          &          JDEBG )
2072: C      c##<2007/03/14:PN4:
2073: C      c##      call LKEPV( H(1), 'XKEF', LXKEF, 7*NLENG, 'R8', LAST, OD0,
2074: C      c## &          JDEBG )
2075: C      call LKEPV( H(1), 'XKEF', LXKEF, 10*NLENG, 'R8', LAST, OD0,
2076: C      &          JDEBG )
2077: C      c##>

```

src/mvp/intro2.f

```

2078:         if ( JPERT.ne.0 ) then
2079:             call LKEPV( H(1), 'XKEFP', LXKEFP, 7*NLENG*NTPT*NUCPT, 'R8',
2080:                 &         LAST, OD0, JDEBG )
2081:         end if
2082: c+beffl
2083:         if( JBEFF.ne.0 ) then
2084:             call LKEPV( H(1), 'XBEF', LXBEF, NEBEF*NLENG, 'R8', LAST,
2085:                 &         OD0, JDEBG )
2086:         end if
2087: c-beffl
2088:         end if
2089: C
2090: C === ETRV : special tally which requires "real variance" calculation.
2091: C
2092:         if ( NBTRV.gt.0 ) then
2093:             NBETRV = NLENG - NSKIP
2094:             if ( NBETRV.gt.0 ) then
2095:                 call LKEPV( H(1), 'ETRV', LETRV, METRV*(NBETRV+1), 'R8',
2096:                     &         LAST, OD0, JDEBG )
2097:             end if
2098:         end if
2099: C
2100:         if ( JREST.ne.0 ) then
2101:             call RESTR( NGROUP, NRESB, NREG, NTREG, NPKIND, NBATCH, NMEMO,
2102:                 &         NEVENT, NZONE, H(LKMEMO), H(LWCNTR), H(LNCNTR),
2103:                 &         H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
2104:                 &         H(LXAVT), H(LAAVT), H(LCAVT),
2105:                 &         NFISB, NFBANK, NFBNK0, NLENG, H(LXXXF), H(LYYYF),
2106:                 &         H(LZZZF), H(LLEEF), H(LIZZF), H(LINUF), H(LIBBGF),
2107:                 &         H(LIBSPF), H(LXSOC), H(LXSXV), H(LXKEF), NLATT, NEST,
2108:                 &         H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF), NUC, NEMIC,
2109:                 &         NEMAC, H(LSRMIC), H(LRMICR), H(LXMIC), H(LRMAC),
2110:                 &         H(LRMACR), H(LXMAC), H(LSRSTR), H(LSRACL), NSTAL,
2111:                 &         H(LWCXTY), H(LDNFLX), NETALY, H(LETALY), NLETALY )
2112:         end if
2113: C
2114: C .....
2115: C
2116:         call GTLAST( LLS1 )
2117:         call LGTLST( LLL1 )
2118: c##<2007/03/14:PN3:
2119: c## write(IPR, '(1X,A,I12,A,I12)')
2120: c## & ' USED MEMORY EXCEPT WORKING AREA (WORDS): TASK SHARED '
2121: c## & LSIZ(LLS1) - 1, ' TASK LOCAL ', LSIZ(LLL1) - 1
2122: c## write(IPR, '(1X,A,I12,A,I12,A)')
2123: c## & ' LAST POSITION OF MEMORY EXCEPT WORKING AREA: A(' , LLS1
2124: c## & - 1, ' ) H(' , LLL1 - 1, ' )'
2125:         write(IPR,604) LSIZ(LLS1)-1, LSIZ(LLL1)-1, LLS1, LLL1
2126:         604 format(
2127:             &/' used memory except working area (words) : task shared',i11,
2128:             &/' : task local',i11
2129:             &/' memory last position except working area : A(' ,i11,') and H(' ,
2130:             &i11,')')
2131: c##>
2132: C
2133: C-----
2134: C ..... /XWORK/ .....
2135: C-----
2136: C
2137:         call GTLAST( LLS0 )
2138:         call LGTLST( LLL0 )
2139: C
2140:         call KEEP( 'LXYZ', LLXYZ, 10, 'I4', LAST, JDEBG )
2141:         call KEEP( 'LPWRK', LLPWRK, MWVEC, 'I4', LAST, JDEBG )
2142:         call KEEP( 'LVSTK', LLVSTK, MVSTK, 'I4', LAST, JDEBG )

```

```

2143:
2144:         if ( LSTCK(0).lt.0 ) then
2145:             LLXYZ = 0
2146:             LLPWRK = 0
2147:             LLVSTK = 0
2148:         end if
2149: C
2150:         call WRKARY( A, H, LIMIT, LIMITL, MAXSF, MWVEC, MVSTK, LLXYZ,
2151:             &         LLPWRK, LLVSTK, A(LLXYZ), A(LLPWRK), A(LLVSTK), MAXWK )
2152: C
2153: C-----
2154: C ..... for CADENZ .....
2155: C-----
2156: C
2157: CCC call KEEPC( 'RNM', LRNM, NBINMX, 'C128', LAST, JDEBG )
2158: call KEEPC( 'RNM', LRNM, MXRGBN, 'C128', LAST, JDEBG )
2159: C
2160: C .....
2161: C
2162: C/#IF PARA( CRAY SX* )
2163:         call GTLAST( LAST )
2164: C/#ELSE
2165:             LAST = MAXWK
2166: C/#ENDIF
2167: C
2168: c##<2007/03/14:PN3:
2169: c## write(IPR, '(1X,A,I12,A)')
2170: c## & ' USED DYNAMIC MEMORY SIZE (TASK SHARED) = ', LSIZ(LAST) -
2171: c## & 1, ' (WORD)'
2172:         write(IPR,605) LSIZ(LAST)-1
2173:         605 format(
2174:             &/' used dynamic memory size (words) : task shared',i11)
2175: c##>
2176: C
2177:         if ( LSIZ(LAST)-1.gt.LIMIT ) then
2178:             write(IMG, '(1X,A,I11,A)')
2179:             & 'XXX(intro2) MEMORY OVER !! LIMIT =', LIMIT, ' (WORD)'
2180:             call PRSTOP( 1, 'MEMORY OVER.' )
2181:             stop 999
2182:         end if
2183: C
2184: C/#IF PARA( CRAY* SX* )
2185: C
2186: C         call LGTLST( LLAST )
2187: C ... local memory size is determined in wrkary routine : maxwk ...
2188: c##<2007/03/14:PN3:
2189: c## write(IPR, '(1X,A,I12,A)')
2190: c## & ' TASK LOCAL DYNAMIC MEMORY TO BE USED = ', LSIZ(LMAXWK)
2191: c## & - 1, ' (WORD)'
2192:         write(IPR,606) LSIZ(LMAXWK)-1
2193:         606 format(' task local dynamic memory to be used (words) =',i11
2194:             &/)
2195: c##>
2196:         write(IPR,*) ' '
2197:         if ( LSIZ(LMAXWK)-1.gt.LIMITL ) then
2198:             write(IMG, '(1X,A,I11,A)')
2199:             & 'XXX(intro2) LOCAL MEMORY OVER !! LIMIT =', LIMITL,
2200:             & ' (WORD)'
2201:             call PRSTOP( 1, 'MEMORY OVER.' )
2202:             stop 999
2203:         end if
2204: C
2205:         call MEMUSE( 'WORKING AREA', 0, MAXWK-LLL0, LIMIT, LIMITL, LIMITC
2206:             & )
2207:         call GTLAST( LLS1 )

```

src/mvp/intro2.f

```
2208:      call MEMUSE( ' ', LSIZ(LLS1)-1, LSIZL(MAXWK)-1, LIMIT, LIMITL,
2209:      &          LIMITC )
2210:
2211: C/#ELSE
2212:      call MEMUSE( 'WORKING AREA', LAST-LLS0, LAST-LLS0, LIMIT, LIMITL,
2213:      &          LIMITC )
2214:      call MEMUSE( ' ', LSIZ(LAST)-1, LSIZL(LAST)-1, LIMIT, LIMITL,
2215:      &          LIMITC )
2216: C/#ENDIF
2217: C
2218: C
2219: C-----
2220: C ..... PARAMETER PRESETTING FOR RANDOM WALK OR TALLY-ANALYSIS ...
2221: C-----
2222: C
2223: C
2224:      if ( JREST.ge.1 ) then
2225:      if ( NPART0.gt.0.and.NTHIST.ge.NPART0 ) then
2226:      JNRUN = 1
2227:      else
2228:      JNRUN = 0
2229:      end if
2230:      else
2231:      NTHIST = 0
2232:      NBATCH = 0
2233:      JNRUN = 0
2234:      end if
2235: C
2236: 7140 format(/1X,'XXX(INTRO2) Cannot proceed input processing any more',
2237:      &      ' because of memory shortage.'/1X,' * Stop before ',A)
2238: 7160 format(/1X,'XXX(INTRO2) Cannot perform input processing ',
2239:      &      ' because of memory shortage.'/1X,' * Skip ',A)
2240: C
2241:      return
2242:      end
```

src/mvp/intro.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine INTRO( TITLE, NTASK0, MLIMIT )
3: C=<MVP>=====
4: C PURPOSE:  DATA INPUT & PREPARATION FOR RANDOM WALK FOR MVP CODE
5: C CALLED IN: CENTER
6: C CALLS:  INTRO2,  DTLIST,OPTION,RESTRT,FREAD,GEOMIN,VALUE0,ARAYIN,ARYSUM,
7: C          DMREAD,  RESET,KEEP,KEPV,ATRANS,RESET,CISZON,CRVOL,CKVAL4,CKODR4
8: C=====
9:   include '../shared/INC/_ARRAY'
10: C
11:   include '../shared/INC/_LISTOFF'
12: C
13:   include 'INC/_KPIDS'
14:   include 'INC/_KPSYMS'
15:   include 'INC/_NGPS'
16: C
17:   include '../shared/INC/_SIZES'
18:   include 'INC/_FLAGS'
19:   include 'INC/_SIZES2'
20:   include 'INC/_CXSEC'
21:   include 'INC/_PXSEC'
22:   include 'INC/_XBANK'
23:   include 'INC/_SBANK'
24:   include 'INC/_STACK'
25:   include 'INC/_XWORK'
26:   include '../shared/INC/_PGEOM'
27:   include '../shared/INC/_PVRED'
28:   include 'INC/_PSOUR'
29:   include '../shared/INC/_PTALY0'
30:   include 'INC/_PTALY'
31:   include 'INC/_PTALY2'
32:   include '../shared/INC/_PTLSP'
33:   include '../shared/INC/_STALY'
34: C
35:   include '../shared/INC/_IOUNIT'
36:   include 'INC/_IOUNIT2'
37: C
38:   include '../shared/INC/_LISTON'
39: C
40:   include '../shared/INC/_WORDL'
41: C
42: C/#IF PARA(PVM)
43:   include '../shared/INC/_PVMPARA'
44: C/#ELSEIF PARA(MPI)
45: *   include 'mpif.h'
46: C/#ENDIF
47:   include '../shared/INC/_TASKDT'
48: C
49:   include '../shared/INC/_RAND'
50:   include 'INC/_PERT' ! pert
51: C
52:   character TITLE(2)*72
53: C
54:   character VNM*6, VNAME*72, VSUBF*16
55: C
56: C ... dummy value for cutoff time
57: C
58:   parameter( TCUTDM = -1.0E30 )
59: C
60: -----
61: C
62:   call CPUTM( T0 )
63: C
64: C ... check unopend scratch file
65: C

```

```

66:   call OPENSC( 'MVP' )
67: C
68: C ... initialize random number generator
69: C
70:   call RNINIT( 1 )
71: C
72: C ... DATA INPUT UNIT SETTING ....
73: C
74:   call RIUNIT( IOIN )
75: C
76: C .... pass version description to HEADER routine...
77: C
78: CC   call VERSTR
79: C
80: C/#IF .NOT.PARA( PVM MPI )
81: C
82: C
83: C .... INPUT DATA LISTING .....
84: C
85:   call INPLST( IPR, MSG, INP, IOIN, NTASK0, MLIMIT )
86: C
87: C/#ELSE IF PARA( PVM MPI )
88: C
89:   NTASK = NTASK0
90:   call TOKEI( T01, 0 )
91:   if ( IDTASK.eq.1 ) then
92:     call INPLST( IPR, MSG, INP, IOIN, NTASK0, MLIMIT )
93:     if ( NTASK0.gt.1 ) then
94:       call RWIND( IOIN )
95:       MSGNUM = 0
96:       IT0 = 1
97: 100   read(IOIN,'(A)',end =110) VNAME
98:       go to 120
99: 110   VNAME = '///END'
100: 120   MSGNUM = MSGNUM + 1
101:       call MVPCOM_BCAST_CH( IT0, MSGNUM, VNAME, LEN(VNAME), INFO )
102:       if ( VNAME.ne.'///END' ) go to 100
103:     end if
104:   else
105:     call RWIND( IOIN )
106:     MSGNUM = 0
107:     IT0 = 1
108: 130   MSGNUM = MSGNUM + 1
109:     VNAME = ' '
110:     call MVPCOM_BCAST_CH( IT0, MSGNUM, VNAME, LEN(VNAME), INFO )
111:     if ( VNAME.ne.'///END' ) then
112:       write(IOIN,'(A)') VNAME
113:       go to 130
114:     end if
115:     call RWIND( IOIN )
116:     call INPLST( IPR, MSG, IOIN, IOIN, NTASK0, MLIMIT )
117:   end if
118: C
119:   call RWIND( IOIN )
120: C
121:   call TOKEI( T02, 0 )
122: C
123: C .. use MSG for task time monitor
124:   write(MSG,'(1X,A,I4,A,1P,E12.5,A,A)') 'TASK ', IDTASK,
125:   &      ': Elapsed ', T02 - T01, ' SEC FOR ',
126:   &      'INPUT DATA PASSING TO OTHER TASKS.'
127: C
128: C/#ENDIF
129: C
130:   call HEADER( IPR, 'PROBLEM TITLE & OPTIONS' )

```

src/mvp/intro.f

```

131: C
132: C
133: C -----
134: C ..... READ TITLE ( 2 LINES )
135: C -----
136: C
137: C
138: C      read(IOIN,'(A)') (TITLE(I),I=1,2)
139: C      write(IPR,7000) (TITLE(I),I=1,2)
140: C      7000 format(1X,130('=')/1X,A/1X,A/1X,130('=')/)
141: C
142: C
143: C -----
144: C ..... READ OPTION PARAMETERS
145: C -----
146: C
147: C      KJPERT = 0 ! pert
148: C
149: C      NTASK = 0
150: C
151: C      call OPTION( MLIMIT )
152: C
153: C      .... MEMORY ALLOCATION CHECK ....
154: C      LSTT : starting position index in dynamic memory array A
155: C              in common /ARRAY/ .
156: C
157: C      call MEMALC( IPR, LSTT, LSTTL, IERR )
158: C
159: C      write(IPR,'()')
160: C
161: C      if ( NTASK.eq.0 ) NTASK = 1
162: C      if ( NTASK0.ne.0 ) NTASK = NTASK0
163: C      C/#IF PARA(PVM MPI)
164: C      * if(JTLST.ne.0) call OPEN_TLF(IDTASK, TLFN) ! time-list
165: C      C/#ENDIF
166: C
167: C -----
168: C ..... DEFAULT VALUES or initial setting of variables mainly in
169: C              common /SIZES/
170: C      (DEFAULT VALUES OF DEPS & DINF WILL BE GIVEN IN 'GEOMIN'.)
171: C -----
172: C      NMEMOP = 2
173: C      DEPS = 0.0D0
174: C      DINF = 0.0D0
175: C      if ( JTIME.ne.0 ) then
176: C          TCUT = TCUTDM
177: C      else
178: C          TCUT = 0.0
179: C      end if
180: C
181: C      NGROUP = 0
182: C      do 140 I = 1, KPLIM
183: C          NGP(I) = 0
184: C          KNGP(I) = 0
185: C          KENGP(I) = 0
186: C      140 continue
187: C          KNGP(KPLIM+1) = 0
188: C          KENGP(KPLIM+1) = 0
189: C
190: C      NZONE = 0
191: C      NINPZ = 0
192: C      NSURF = 0
193: C      NSDA = 0
194: C      NZDA = 0
195: C      NMEMO = 0

196: C      NMAT = 0
197: C      NREG = 0
198: C      NTIME = 0
199: C      NSOUR = 0
200: C      NMEMS = 0
201: C      NRESP = 0
202: C      NRESP2 = 0
203: C      NRSKIP = 0
204: C      NSKIP = 0
205: C      NGP1 = 0
206: C      NGP2 = 0
207: C      NSPACE = 0
208: C      NBODY = 0
209: C      NTREG = 0
210: C      NSTALY = 0
211: C      NETALY = 0
212: C      NDTALY = 0
213: C      NLETAL = 0
214: C      NLDTAL = 0
215: C      NBINMX = 0
216: C      NMKREG = 0
217: C
218: C      IRNSU = 0
219: C      IRNSM = 0
220: C      IRNSL = 0
221: C
222: C      NTSRF = 0
223: C      MTSRF = 0
224: C      NANGLE = 0
225: C
226: C      IRAND = 0
227: C      NBPINT = 1
228: C      NRSINT = 0
229: C      NTMINT = 10
230: C
231: C      NHIST = 0
232: C      NHSUB = 0
233: C      NPART = 0
234: C      NBANK = 0
235: C      NBNK2 = 0
236: C      IMPMAX = 0
237: C
238: C      NFBANK = 0
239: C      NFBNK0 = 0
240: C
241: C      NPDET = 0
242: C      NPFCN = 0 ! photo-nuc
243: C      NPPNBR = 0 ! photo-nuc
244: C      NPMT = 0 ! photo-nuc
245: C      NMONPNW = 0 ! photo-nuc
246: C      .... FOR CONTINUOUS ENERGY VERSION ....
247: C      MB = 0
248: C      ETOP = 2.0E+7
249: C      EBOT = 1.0E-5
250: C      ETHMAX = 4.5
251: C      AMLIM = 1000.0
252: C      EWCUT = 275.0
253: C      EINCD = 0.0253
254: C
255: C      ETOPP = 1.0E+8
256: C      EBOTP = 1.0E+3
257: C      BANKP = 0.5
258: C
259: C      do 150 IK = 1, KPLIM
260: C          ETOPX(IK) = 0.0

```

src/mvp/intro.f

```

261:          EBOTX(IK)  = 0.0
262: 150 continue
263:      ETOPIX(KPNEUT) = ETOP
264:      EBOTX(KPNEUT)  = EBOT
265:      ETOPIX(KPPHOT) = ETOPP
266:      EBOTX(KPPHOT)  = EBOTP
267: C
268:      NSTAL  = 0
269:      NSTAL2 = 0
270:      NICRES = 0
271: C
272:      NNCST  = 0
273:      NNCSI  = 0
274: C
275:      NXPGEN = 0
276: C
277:      NMMLAG = 10
278:      NEITER = 500
279: C
280:      WLLIM  = 1.0E-10
281: C
282:      ELOOP  = 0.2
283: C
284:      TCPU   = 0.0
285: C
286:      NPKIND = 0
287: C      if ( JNEUT.ne.0 ) NPKIND = NPKIND + 1
288: C      if ( JPHOT.ne.0 ) NPKIND = NPKIND + 1
289:      KKPID  = 0
290:      do 160 IK = 1, KPLIM
291:          if ( JKPAR(IK).ne.0 ) then
292:              KKPID  = IK
293:              NPKIND = NPKIND + 1
294:          end if
295: 160 continue
296: C
297: C      ... KKPID is the only effective particle species ID for
298: C      single particle problem.
299: C
300:      if ( NPKIND.gt.1 ) KKPID  = 0
301: C-----
302: C ..... CLEAR SOME POINTERS .....
303: C-----
304:      LXIMP  = 0
305:      LWKIL  = 0
306:      LWSRV  = 0
307:      LWTIME = 0
308:      LPSXYZ = 0
309:      LPSALP = 0
310:      LSRCSP = 0
311:      LSOUR  = 0
312:      LIDSRC = 0
313:      LKSOUR = 0
314:      LPENRG = 0
315:      LRESP  = 0
316:      LFKAI  = 0
317:      LIFISM = 0
318:      LWGTF  = 0
319:      LWGTFD = 0
320:      LWGTP  = 0
321:      LWGTPN = 0 ! photo-nuc
322:      LPFCNR = 0 ! photo-nuc
323:      LPPNBR = 0 ! photo-nuc
324:      LPMT   = 0 ! photo-nuc
325:      LMONPNW = 0 ! photo-nuc

326:      LISZON = 0
327:      LENGYB = 0
328:      LENGPB = 0
329:      LTIMEB = 0
330:      LVOL   = 0
331:      LRVOL  = 0
332:      LTRVOL = 0
333: C##      LKR   = 0 ! photo-nuc
334:      LNSANG = 0
335:      LSANG  = 0
336:      LPSANG = 0
337:      LSAXIS = 0
338:      LIDTAL = 0
339:      LIETAL = 0
340:      LKDTAL = 0
341:      LKETAL = 0
342:      LJTEVE = 0
343:      LLTEVE = 0
344:      LKTEVE = 0
345:      NTEVE  = 0
346: C
347:      LIDSRF = 0
348:      LITSRF = 0
349:      LKTSRF = 0
350:      LKTSFJ = 0
351:      LANGLB = 0
352: C
353:      LXPDET = 0
354:      LJPUSD = 0
355:      LJSPDT = 0
356: C
357: C ... initialize extra bank parameter index
358:      do 170 K = 0, MDBNK
359:          KDBNK(K) = 0
360: 170 continue
361:      do 180 K = 0, MIBNK
362:          KIBNK(K) = 0
363: 180 continue
364:      KPNPRD = 0 ! photo-nuc: flag for particle production: ../shared/INC/_ST
ALY
365:      LKPNFG = 0 ! photo-nuc
366: C
367: C ... initialize extra fission bank parameter index
368:      if ( JEIGN.ne.0 ) then
369:          do 190 K = 1, MDFBK
370:              KDFBK(K) = 0
371: 190 continue
372:          do 200 K = 1, MIFBK
373:              KIFBK(K) = 0
374: 200 continue
375:          end if
376: C
377: C ... local flags ...
378:      ICHK  = 0
379:      ICK   = 0
380: C
381: C ..... JNP = 1 : means neutron - photon coupled problem
382: C
383: C      JNP  = JNEUT*JPHOT
384: C      JNP  = 0
385: C      if ( JKPAR(KPNEUT).ne.0.and.JKPAR(KPPHOT).ne.0 ) then
386: C          JNP  = 1
387: C      end if
388: C
389: C ..... checker of duplicate blocks

```


src/mvp/intro.f

```

390: C
391:     KKKGEO = 0
392:     KKKXSC = 0
393:     KKKSRC = 0
394:     KKKTAL = 0
395:
396: C ... TEMPORARY WORKING ARRAY ....           ! photo-nuc
397:     NRRRO = MAX(NREG,NTREG)                   ! photo-nuc
398:     call KEEP( 'KR', LKR, NRRRO, 'I4', LAST, JDEBG ) ! photo-nuc
399: C
400: C-----
401: C ... READ OTHER DATA .....
402: C-----
403: C
404: C
405: C .... To set null address assignment checker,
406: C     and initial value of 'LAST' is set in this routine ...
407: C
408:     call CYNULO( A, LAST, LSTT )
409: C
410: C     .... NLEN : Number of non blank characters read as VNAME ...
411: C
412:     210 call FREADS( VNAME, NLEN, IEND )
413:     if ( IEND.ne.0 ) go to 220
414:     ICALL = 0
415: C
416: C
417: C-----
418: C ..... GEOMETRY .....
419: C-----
420: C
421: C
422:     if ( NLEN.ge.5.and.VNAME(1:5).eq.'$GEOM' ) then
423:
424:         if ( KKKGEO.ne.0 ) then
425:             write(IMG,'(1X,A)')
426:         & 'XXX(intro) $GEOMETRY block is found more than once.'
427:         & call CNTERR( 'FATAL' )
428:         end if
429:         KKKGEO = 1
430:
431:         call GTLAST( LLS0 )
432:         call LGTLST( LLL0 )
433:         call CPUTM( TT0 )
434:
435:         call GEOMIN( ICALL, A, LIMIT, CHA, LIMITC, LAST, MAXSF, JDEBG,
436:         & JHLAT, JLATT, JREFL, JSIMP, JTLLT, JFISX, JSTGR )
437:
438:         call CPUTM( TT1 )
439:
440:         write(IPR,'(/2x,a,1p,e12.5,a/)')
441:         & '==== CPU TIME FOR GEOMETRY PROCESSING ', TT1 - TT0,
442:         & ' SEC'
443: C
444:         call GTLAST( LLS1 )
445:         call LGTLST( LLL1 )
446:         call MEMUSE( 'GEOMETRY DATA', LLS1-LLS0, LLL1-LLL0, LIMIT,
447:         & LIMITL, LIMITC )
448: C
449:         go to 210
450:     end if
451: C
452: C-----
453: C ..... CROSS SECTION .....
454: C-----

```

```

455: C-----
456: C
457: C
458:     if ( NLEN.eq.5.and.VNAME(1:5).eq.'$XSEC' .or. NLEN.ge.5
459:     & .and.VNAME(1:5).eq.'$CROS' ) then
460: C
461:         if ( KKKXSC.ne.0 ) then
462:             write(IMG,'(1X,A,A)')
463:         & 'XXX(intro) $CROSS SECTION block is found more',
464:         & ' than once.'
465:         & call CNTERR( 'FATAL' )
466:         end if
467:         KKKXSC = 1
468:
469:         call CPUTM( TT0 )
470:         call TOKEI( TT0, 0 )
471: C
472:         call GTLAST( LLS0 )
473:         call LGTLST( LLL0 )
474: C
475: C ... perform cross section input on parent task only in PVM mode.
476: C
477:         call GTLAST( LAST )
478:         LSTART = LAST
479: C
480:         JXSKIP = 0
481:         call CTXSIN( A, A, CHA, LSTART, LAST, LIMIT, LIMITC, JXSKIP )
482:         call CTXSIN( A, A, CHA, LIMIT, LIMITC, JXSKIP )
483: C
484:         call STLAST( 'XSEC', LAST, LAST )
485:         call CPUTM( TT1 )
486:         call TOKEI( TT0, 0 )
487:
488:         write(IPR,'(/1x,a)')
489:         & '==== TIMES FOR CROSS SECTION PROCESSING ==== '
490:         write(IPR,'(6x,a,1p,e12.4,a)') 'CPU ', TT1 - TT0, ' SEC'
491:         write(IPR,'(6x,a,1p,e12.4,a)') 'ELAPSED ', TT0 - TT1, ' SEC'
492:         write(IPR,'(1h)')
493: C
494: C ..... RESET INPUT STATUS FOR FREAD ROUTINE ...
495: C
496:         call RESET
497: C
498:         call GTLAST( LLS1 )
499:         call LGTLST( LLL1 )
500:         call MEMUSE( 'X-SEC DATA', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
501:         & LIMITC )
502: C
503:         go to 210
504:     end if
505: C
506: C-----
507: C "REGION"-DEPENDENT DATA
508: C-----
509: C !REGION-NAME( VNAME(DATA) .... )
510: C OR VNAME( !REGION-NAME( DATA ) .... )
511: C OR VNAME( DATA .... )
512: C
513:     IPPP = INDEX(VNAME(:NLEN),'.') - 1
514:     if ( IPPP.lt.0 ) IPPP = NLEN
515:     if(VNAME(1:1).eq.'I' .or. VNAME(1:IPPP).eq.'XIMP' .or.
516:     & VNAME(1:IPPP).eq.'WKIL' .or. VNAME(1:IPPP).eq.'WSRV' .or.
517:     & VNAME(1:IPPP).eq.'WGTF' .or. VNAME(1:IPPP).eq.'WGTP' .or.
518:     & VNAME(1:IPPP).eq.'WGTPNR' .or. ! photo-nuc
519:     & VNAME(1:IPPP).eq.'RVOL' .or. VNAME(1:IPPP).eq.'TRVOL' .or.

```

src/mvp/intro.f

```

520:      &  VNAME(1:IPPP).eq.'PSALP') then
521: C
522: C      ... check number of energy groups.
523: C
524:      if ( VNAME(1:IPPP).eq.'XIMP' .or. VNAME(1:IPPP).eq.'WKIL' .or.
525:      &  VNAME(1:IPPP).eq.'WSRV' ) then
526:      call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
527:      &  NGROUP )
528:      call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
529:      end if
530: C
531:      call REGDAT( MSG, VNAME(:NLEN), VNM, VSUBF, A, A, LIMIT, LAST,
532:      &  JNEUT, JPHOT, JDEBG, NREG, NGP(1), JKPAR(1), KPSYM,
533:      &  KPLIM, KNGP(1), NGROUP, NGP1, NGP2, LXIMP, LWKIL,
534:      &  LWSRV, LWGTF, LWGTP, LRVOL, LTRVOL, LPSALP, JTLT,
535:      &  A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
536:      &  A(LISPM), A(LISOSP), A(LKSUZ), A(LIRGSP), NEST,
537:      &  NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAME$, A(LITRNM),
538:      &  NTREG, A(LLPTRG), A(LLPTRG), A(LKR), NRRRO,
539:      &  NUCPN, NMTPN, LWGTPN) ! photo-nuc
540: C
541:      if ( VNAME(1:IPPP).eq.'WGTPNR' .and. NUCPN.gt.1 ) then ! photo-nuc
542: C      ... rearrangement from (nreg) to (nmtpn,nucpn,nreg) ! photo-nuc
543:      do I = 1, NREG ! photo-nuc
544:      WGTPN0 = A(LWGTPN-I+NREG) ! photo-nuc
545:      I1 = LWGTPN + NMTPN * NUCPN * (NREG - I) - 1 ! photo-nuc
546:      do J = 1, NMTPN*NUCPN ! photo-nuc
547:      A(I1+J) = WGTPN0 ! photo-nuc
548:      end do ! photo-nuc
549:      end do ! photo-nuc
550:      end if ! photo-nuc
551: C
552: C      ICALL = 1
553:      go to 210
554: C
555: C
556:      end if
557: C
558: C
559: C----- SOURCE -----
560: C ..... SOURCE .....
561: C-----
562: C
563: C/#IF SOURCE(NEW)
564: C
565:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$SOUR' ) then
566: C
567: CCC      call CHKNGP( 'MVP',JNEUT, NGP1, JPHOT, NGP2, NGROUP )
568:      call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
569:      &  NGROUP )
570: C
571:      if ( KKKSRC.ne.0 ) then
572:      write(MSG,'(1X,A)')
573:      &  'XXX(intro) $SOURCE block is found more than once.'
574:      call CNTERR( 'FATAL' )
575:      end if
576:      KKKSRC = 1
577: C
578:      call GTLAST( LLS0 )
579:      call LGTLST( LLL0 )
580: C
581:      call GTLAST( LAST )
582:      LSTART = LAST
583: C
584:      NTGX0 = 0

```

```

585:
586:      call SRCINP( A, H, CHA, LIMIT, LIMITL, LIMITC, LSRCSP, NSRCSP,
587:      &  LSOUR, LIDSRC, 'MVP', JKPAR(1), KPLIM, JFISS, JEIGN,
588:      &  JTIME, JNEUT, JPHOT, JUNDG, JDEBG, MWVEC, MVSTK, MXREJ,
589:      &  NERR, A(LIDMAT), CHA(LNUCID), NUC, NTGX0, LKSOUR,
590:      &  LISZON, LPSPAC, LPENRG, LIFISM, LFKAI, LENGYB, LENGPB,
591:      &  EINCD )
592: C
593:      call GTLAST( LLS1 )
594:      call LGTLST( LLL1 )
595:      call MEMUSE( 'SOURCE DATA', LLS1-LLS0, LLL1-LLL0, LIMIT,
596:      &  LIMITL, LIMITC )
597: C
598:      go to 210
599:      end if
600: C
601: C/#ENDIF
602: C
603: C
604: C-----
605: C ..... TALLY .....
606: C-----
607: C
608: C
609:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$TALL' ) then
610: C
611: CCCCC      call CHKNGP( 'MVP',JNEUT, NGP1, JPHOT, NGP2, NGROUP )
612:      call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
613:      &  NGROUP )
614: C
615:      if ( KKKTAL.ne.0 ) then
616:      write(MSG,'(1X,A)')
617:      &  'XXX(intro) $TALLY block is found more than once.'
618:      call CNTERR( 'FATAL' )
619:      end if
620:      KKKTAL = 1
621: C
622:      call CPUTM( TT0 )
623:      call TALINP( 'MVP', ICALL, LICRES, NICRES, LWTIME, A, A, A,
624:      &  CHA, H, H, H, NGP(1), JKPAR(1), KPSYM, KNGP(1),
625:      &  KENGP(1), KPLIM, JNEUT, JPHOT, JTIME, JFISS, JEIGN,
626:      &  JRESP, JTLT, JTSRF, JPTDT, JDEBG, A(LIDSRC), LIDSRC,
627:      &  NUC, KDBNK, MDBNK, KIBNK, MIBNK,
628:      &  LKR, LIMIT, LXIMP, LWKIL, LWSRV, LWGTF, LWGTP,
629:      &  LPSALP,
630:      &  JPHNU, NPATOM, NUCPN, NMTPN, LWGTPN) ! photo-nuc
631:      call CPUTM( TT1 )
632: C
633:      write(IPR,'(/1X,a,1p,e12.5,a/)')
634:      &  '==== CPU TIME FOR TALLY BLOCK PROCESSING:', TT1 - TT0,
635:      &  ' SEC'
636:      go to 210
637:      end if
638: C
639: C----- ! pho
to-nuc
640: C ..... other variance reduction data ..... ! pho
to-nuc
641: C----- ! pho
to-nuc
642: C
643:      if ( NLEN.eq.4.and.VNAME(1:4).eq.'PFCN' ) then !+pho
to-nuc
644:      call PFCNINP( MSG, A, A, LIMIT, LAST, JDEBG, NMAT, NREG,
645:      &  NUCPN, NPFCN, LPFCNR, CHA(LNCIDPN), A(LMNUCPN),

```

src/mvp/intro.f

```

646: &      A(LLPIDPN), JTLT, A(LIDMAT), A(LKMAT), A(LKREG),
647: &      A(LKREGI), A(LKCELI), NINPZ, A(LISPNM), A(LISUSP),
648: &      A(LKSUZON), A(LIRGSP), NEST, NSPACE, NSUZON, NZONE,
649: &      CHA(LTNAMS), NNAMES, A(LITRNM), NTREG, A(LIPTRG),
650: &      A(LLPTRG), A(LKR), NRRRO )
651:   go to 210
652:   else if ( NLEN.eq.5.and.VNAME(1:5).eq.'PPNBR' ) then
653:     call PNBRINP( MSG, A, A, LIMIT, LAST, JDEBG, NUCPN, NPPNBR,
654: &      LPPNBR, CHA(LNCIDPN), A(LMNUCPN), A(LLPIDPN) )
655:     go to 210
656:   else if ( NLEN.eq.5.and.VNAME(1:5).eq.'WGTPN' ) then
657:     call WGTPNINP( MSG, A, A, LIMIT, LAST, JDEBG, NREG, NUCPN,
658: &      NMTPN, LWGTPN, CHA(LNCIDPN), A(LMNUCPN), A(LLPIDPN),
659: &      JTLT, A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
660: &      A(LISPNM), A(LISUSP), A(LKSUZON), A(LIRGSP), NEST,
661: &      NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAMES, A(LITRNM),
662: &      NTREG, A(LIPTRG), A(LLPTRG), A(LKR), NRRRO )
663:     go to 210
664:   else if ( NLEN.eq.3.and.VNAME(1:3).eq.'PMT' ) then
665:     call PMTINP( MSG, A, A, LIMIT, LAST, JDEBG, NREG, NUCPN,
666: &      LPMT, NPMT, NMTPN, CHA(LNCIDPN), A(LMNUCPN),
667: &      A(LLPIDPN), JTLT, A(LKMAT), A(LKREG), A(LKREGI),
668: &      A(LKCELI), NINPZ, A(LISPNM), A(LISUSP), A(LKSUZON),
669: &      A(LIRGSP), NEST, NSPACE, NSUZON, NZONE, CHA(LTNAMS),
670: &      NNAMES, A(LITRNM), NTREG, A(LIPTRG), A(LLPTRG), A(LKR),
671: &      NRRRO )
672:     go to 210
673:   else if ( NLEN.eq.5.and.VNAME(1:5).eq.'WGTFD' ) then
674:     call WGTFDINP( MSG, A, A, LIMIT, LAST, JDEBG, NREG, LWGTF,
675: &      JTLT, A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
676: &      A(LISPNM), A(LISUSP), A(LKSUZON), A(LIRGSP), NEST,
677: &      NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAMES, A(LITRNM),
678: &      NTREG, A(LIPTRG), A(LLPTRG), A(LKR), NRRRO )
679:     LWGTFD = 1
680:     go to 210
681:   else if ( NLEN.eq.8.and.VNAME(1:8).eq.'MONPNWGT' ) then
682:     call MONPNWINP( MSG, A, A, LIMIT, LAST, JDEBG, NREG, NUCPN,
683: &      LMONPNW, NMONPNW, NMTPN, CHA(LNCIDPN), A(LMNUCPN),
684: &      A(LLPIDPN), JTLT, A(LKMAT), A(LKREG), A(LKREGI),
685: &      A(LKCELI), NINPZ, A(LISPNM), A(LISUSP), A(LKSUZON),
686: &      A(LIRGSP), NEST, NSPACE, NSUZON, NZONE, CHA(LTNAMS),
687: &      NNAMES, A(LITRNM), NTREG, A(LIPTRG), A(LLPTRG), NUCPNI,
688: &      A(LKLBPNI), A(LKLBPNI2), A(LKR), NRRRO )
689:     go to 210
690:   end if
to-nuc
691: C
692: C
693: C-----
694: C ..... PERTURBATION .....
695: C-----
696: C
697: C
698:   if ( NLEN.ge.5 .and. VNAME(1:5).eq.'$PERT' ) then
699:     call CPUTM( TT0 )
700:     KJPRT = 1
701:     call PTINP( 'MVP', 6, ICALL, A, A, A, CHA, H, H, H,
702: &      CHA(LNUCID), IUB )
703:     call CPUTM( TT1 )
704:     write(IPR, '(1x,a,1p,e12.5,a)')
705:     &      '==== CPU TIME FOR PERTURBATION BLOCK PROCESSING:',
706:     &      TT1 - TT0, ' SEC'
707:     go to 210
708:   end if
709: C

```

```

710: C-----
711: C ..... EXTRACT 'VARIABLE NAME' FROM VNAME .....
712: C-----
713: C
714:   VNM = ' '
715:   VSUBF = ' '
716:   IPER = INDEX(VNAME, '.')
717:   IKPID = 0
718: C
719: C ..... VNAME with particle species subfield .....
720: C
721:   if ( IPER.ne.0 ) then
722:     VNM(1:IPER-1) = VNAME(1:IPER-1)
723:     VSUBF = VNAME(IPER+1:NLEN)
724: C
725: C   IPR2 = INDEX('NP',VSUBF(1:1))
726: C   call KPSYMB( VSUBF(1:ICLEN2(VSUBF)), '>', IKPID, 0 )
727: C
728:   if ( IKPID.eq.0 ) then
729:     write(MSG, '(1x,a,a,a,a,a)')
730:     &      'XXX(intro) Data name suffix( ',
731:     &      VSUBF(1:ICLEN2(VSUBF)), ' ) of ',
732:     &      VNAME(1:ICLEN2(VNAME)), ' is incorrect.'
733:     write(MSG,*) ' No matching particle species.'
734:     call CNTERR( 'FATAL' )
735:     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
736:     if ( IRET.ne.0 ) go to 240
737:     go to 210
738: C
739: C ... supported particle species but not used in this problem
740: C
741:   else if ( JKPAR(IKPID).eq.0 ) then
742:     write(MSG, '(1x,a,a,a,a,a)')
743:     &      '!!! (intro) ', VNAME(1:ICLEN2(VNAME)),
744:     &      ' has no meanings because particle species <',
745:     &      VSUBF(1:ICLEN2(VSUBF)), '> is not treated',
746:     &      ' in current problem.'
747:     call CNTERR( 'WARNING' )
748:     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
749:     if ( IRET.ne.0 ) go to 240
750:     go to 210
751:   end if
752: C
753: C ..... VNAME WITHOUT SUBFIELD .....
754: C
755:   else
756:     NLLL = MIN(NLEN, LEN(VNM))
757:     VNM(1:NLLL) = VNAME(1:NLLL)
758:   end if
759: C
760: C   if ( VSUBF.eq.'P'.and.JPHOT.eq.0 ) then
761: C     write(IPR, '(1x,a,a,a)') '!!! ', VNAME,
762: C     &      ' has no meanings for no-photon problem. Ignored !!'
763: C     call CNTERR( 'WARNING' )
764: C     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
765: C     if ( IRET.ne.0 ) go to 210
766: C     go to 180
767: C   end if
768: C   if ( VSUBF.eq.'N'.and.JNEUT.eq.0 ) then
769: C     write(IPR, '(1x,a,a,a)') '!!! ', VNAME,
770: C     &      ' has no meaning for no-neutron problem. Ignored !!'
771: C     call CNTERR( 'WARNING' )
772: C     call DMREAD( VNAME, IDUMMY, IDUMMY, IRET )
773: C     if ( IRET.ne.0 ) go to 210
774: C     go to 180

```

!-pho

src/mvp/intro.f

```

775: C      end if
776: C
777: C
778: C -----
779: C ..... OTHERS (SIZE PARAMETER ETC.) .....
780: C -----
781: C
782: C
783: C      if ( VNM.eq.'DEPS' ) call VALUE0( VNAME, ICALL, 'R8', DEPS )
784: C      if ( VNM.eq.'DINF' ) call VALUE0( VNAME, ICALL, 'R8', DINF )
785: C      if ( VNM.eq.'NPART' ) call VALUE0( VNAME, ICALL, 'I4', NPART )
786: C      if ( VNM.eq.'NPART' ) call VALUE0( VNAME, ICALL, 'I8', NPART )
787: C      if ( VNM.eq.'NHIST' ) call VALUE0( VNAME, ICALL, 'I4', NHIST )
788: C      if ( VNM.eq.'NHSUB' ) call VALUE0( VNAME, ICALL, 'I4', NHSUB )
789: C      if ( VNM.eq.'NBANK' ) call VALUE0( VNAME, ICALL, 'I4', NBANK )
790: C      if ( VNM.eq.'NBANK2' ) call VALUE0( VNAME, ICALL, 'I4', NBANK2 )
791: C      if ( VNM.eq.'NBPINT' ) call VALUE0( VNAME, ICALL, 'I4', NBPINT )
792: C      if ( VNM.eq.'NRSINT' ) call VALUE0( VNAME, ICALL, 'I4', NRSINT )
793: C      if ( VNM.eq.'NTMINT' ) call VALUE0( VNAME, ICALL, 'I4', NTMINT )
794: C      if ( VNM.eq.'BANKP' ) call VALUE0( VNAME, ICALL, 'R4', BANKP )
795: C
796: C      if ( VNM.eq.'NGROUP' ) then
797: C      if ( VSUBF.eq.' ' .or. JNP.eq.0 ) then
798: C      call VALUE0( VNAME, ICALL, 'I4', NGROUP )
799: C      if ( JNEUT.ne.0 ) NGP1 = NGROUP
800: C      if ( JPHOT.ne.0 ) NGP2 = NGROUP
801: C      if ( VSUBF.eq.' ' ) then
802: C      call VALUE0( VNAME, ICALL, 'I4', NGROUP )
803: C      if ( JNP.eq.0 ) then
804: C      if ( JNEUT.ne.0 ) NGP1 = NGROUP
805: C      if ( JPHOT.ne.0 ) NGP2 = NGROUP
806: C      else
807: C      write(IMG,*) '!!! data "NGROUP()" is ambiguous for ',
808: C      &      'multi particle coupled problem (N+P etc.)'
809: C      call CNTERR('WARNING')
810: C      if ( JNEUT.ne.0 ) then
811: C      write(IPR,*)
812: C      &      ' "NGROUP()" is taken as number of NEUTRON energy bin'
813: C      NGP1 = NGROUP
814: C      end if
815: C      end if
816: C      else if ( VSUBF.eq.'N' ) then
817: C      call VALUE0( VNAME, ICALL, 'I4', NGP1 )
818: C      if ( JPHOT.eq.0 ) NGROUP = NGP1
819: C      if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
820: C      else if ( VSUBF.eq.'P' ) then
821: C      call VALUE0( VNAME, ICALL, 'I4', NGP2 )
822: C      if ( JNEUT.eq.0 ) NGROUP = NGP2
823: C      if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
824: C      end if
825: C      end if
826: C
827: C      if ( VNM.eq.'NGROUP' .or. VNM.eq.'NGP' ) then
828: C      call VALUE0( VNAME, ICALL, 'I4', NNNNG )
829: C      if ( IKPID.eq.0 ) then
830: C      if ( KKPID.ne.0 ) then
831: C      NGP(KKPID) = NNNNG
832: C      NGROUP = NNNNG
833: C      if ( KKPID.eq.KPNEUT ) NGP1 = NNNNG
834: C      if ( KKPID.eq.KPPHOT ) NGP2 = NNNNG
835: C      else
836: C      write(IMG, '(1x,a,a)')
837: C      &      '!!!(intro) Data "NGROUP()" is ambiguous for ',
838: C      &      'multi particle coupled problem (N+P etc.)'
839: C      call CNTERR( 'WARNING' )

```

```

840: C      if ( JKPAR(KPNEUT).ne.0 ) then
841: C      write(IPR,*)
842: C      &      ' "NGROUP()" is taken as number of NEUTRON energy bin'
843: C      NGP(KPNEUT) = NNNNG
844: C      NGP1 = NNNNG
845: C      end if
846: C      end if
847: C      else
848: C      NGP(IKPID) = NNNNG
849: C      if ( IKPID.eq.KPNEUT ) NGP1 = NNNNG
850: C      if ( IKPID.eq.KPPHOT ) NGP2 = NNNNG
851: C      end if
852: C      end if
853: C
854: C      ... NGP1 and NGP2 will be obsolete and they should be replaced
855: C      with NGP(KPNEUT) and NGP(KPPHOT).
856: C      The following lines are only for backward compatibility.
857: C
858: C      if ( VNM.eq.'NGP1' ) then
859: C      call VALUE0( VNAME, ICALL, 'I4', NGP1 )
860: C      NGP(KPNEUT) = NGP1
861: C      end if
862: C      if ( VNM.eq.'NGP2' ) then
863: C      call VALUE0( VNAME, ICALL, 'I4', NGP2 )
864: C      NGP(KPPHOT) = NGP2
865: C      end if
866: C
867: C      if ( VNM.eq.'NGP1'.or.VNM.eq.'NGP2' ) then
868: C      if ( NGP1.gt.0.and.NGP2.gt.0 ) then
869: C      NGROUP = NGP1 + NGP2
870: C      end if
871: C      end if
872: C
873: C      ... Are following statements fully compatible with the obsoleted
874: C      statements above ???
875: C
876: C      if ( NPKIND.eq.2.and.JKPAR(KPNEUT).ne.0.and.JKPAR(KPPHOT).ne.0
877: C      &      .and.NGP1.gt.0.and.NGP2.gt.0 ) then
878: C      NGROUP = NGP1 + NGP2
879: C      end if
880: C
881: C      if ( VNM.eq.'NMEMO' ) call VALUE0( VNAME, ICALL, 'I4', NMEMO )
882: C
883: C      if ( VNM.eq.'NREG' ) then
884: C      write(IPR, '(1x,a)')
885: C      &      '!!! Direct input for number of REGIONS (NREG) has no meaning.'
886: C      write(IPR, '(1x,2a)')
887: C      &      ' (use "%NREG" parameter for REGION-number-',
888: C      &      'bounded parameters if necessary.)'
889: C      end if
890: C      if ( VNM.eq.'NTIME' ) call VALUE0( VNAME, ICALL, 'I4', NTIME )
891: C      if ( VNM.eq.'NSOUR' ) call VALUE0( VNAME, ICALL, 'I4', NSOUR )
892: C      if ( VNM.eq.'NRESP' ) call VALUE0( VNAME, ICALL, 'I4', NRESP )
893: C      if ( VNM.eq.'TCPU' ) call VALUE0( VNAME, ICALL, 'R4', TCPU )
894: C      if ( VNM.eq.'IRAND' ) call VALUE0( VNAME, ICALL, 'I4', IRAND )
895: C      if ( VNM.eq.'IRAND' ) then
896: C      call VALUE0( VNAME, ICALL, 'I4', IRAND )
897: C      IRNSU = int(IRAND/2.d0**42)
898: C      IRNSM = int((IRAND - IRNSU*2.d0**42)/2.d0**21)
899: C      IRNSL = nint(IRAND - IRNSU*2.d0**42 - IRNSM*2.d0**21)
900: C      end if
901: C
902: C      if ( VNM.eq.'MXPGEN' ) call VALUE0( VNAME, ICALL, 'I4', MXPGEN )
903: C      IF(VNM.EQ. 'ECUT' ) CALL VALUE0(VNAME,ICALL, 'R4', ECUT )
904: C      if ( VNM.eq.'TCUT' ) call VALUE0( VNAME, ICALL, 'R4', TCUT )

```

src/mvp/intro.f

```

905:      if ( VNM.eq.'SUPPLY' ) call VALUE0( VNAME, ICALL, 'R4', SUPPLY )
906:      if ( VNM.eq.'WLLIM' ) call VALUE0( VNAME, ICALL, 'R4', WLLIM )
907:      if ( VNM.eq.'ELOOP' ) call VALUE0( VNAME, ICALL, 'R4', ELOOP )
908:
909:      if ( VNM.eq.'NFBANK' ) call VALUE0( VNAME, ICALL, 'I4', NFBANK )
910:      if ( VNM.eq.'NSKIP' ) call VALUE0( VNAME, ICALL, 'I4', NSKIP )
911:      if ( VNM.eq.'NMXLAG' ) call VALUE0( VNAME, ICALL, 'I4', NMXLAG )
912:      if ( VNM.eq.'NEITER' ) call VALUE0( VNAME, ICALL, 'I4', NEITER )
913:
914:      if ( VNM.eq.'NPICT' ) call VALUE0( VNAME, ICALL, 'I4', NPICT )
915:      if ( VNM.eq.'NMEMS' ) call VALUE0( VNAME, ICALL, 'I4', NMEMS )
916:      if ( VNM.eq.'NMEMOP' ) call VALUE0( VNAME, ICALL, 'I4', NMEMOP )
917:      if ( VNM.eq.'NRSKIP' ) call VALUE0( VNAME, ICALL, 'I4', NRSKIP )
918: C
919:      if ( VNM.eq.'NPDET' ) call VALUE0( VNAME, ICALL, 'I4', NPDET )
920:      if ( VNM.eq.'IMPMAX' ) call VALUE0( VNAME, ICALL, 'I4', IMPMAX )
921: C
922: C
923: C ..... for continuous x-seg data ....
924: C
925: C
926:      if ( VNM.eq.'ETOP' ) then
927:      if ( IKPID.ne.0 ) then
928:          call VALUE0( VNAME, ICALL, 'R8', ETOPIX(IKPID) )
929:          if ( IKPID.eq.KPNEUT ) ETOPI = ETOPIX(IKPID)
930:          if ( IKPID.eq.KPPHOT ) ETOPI = ETOPIX(IKPID)
931:      else
932:          call VALUE0( VNAME, ICALL, 'R8', ETOPIX(KPNEUT) )
933:          ETOPI = ETOPIX(KPNEUT)
934:      end if
935:
936: C      if ( IPER.eq.0 .or. VSUBF.eq.'N' ) then
937: C          call VALUE0( VNAME, ICALL, 'R4', ETOPI )
938: C      else if ( VSUBF.eq.'P' ) then
939: C          call VALUE0( VNAME, ICALL, 'R4', ETOPI )
940: C      end if
941:
942:      end if
943: C
944:      if ( VNM.eq.'EBOT' ) then
945:      if ( IKPID.ne.0 ) then
946:          call VALUE0( VNAME, ICALL, 'R8', EBOTX(IKPID) )
947:          if ( IKPID.eq.KPNEUT ) EBOT = EBOTX(IKPID)
948:          if ( IKPID.eq.KPPHOT ) EBOT = EBOTX(IKPID)
949:      else
950:          call VALUE0( VNAME, ICALL, 'R8', EBOTX(KPNEUT) )
951:          EBOT = EBOTX(KPNEUT)
952:      end if
953:
954: C      if ( IPER.eq.0 .or. VSUBF.eq.'N' ) then
955: C          call VALUE0( VNAME, ICALL, 'R4', EBOT )
956: C      else if ( VSUBF.eq.'P' ) then
957: C          call VALUE0( VNAME, ICALL, 'R4', EBOT )
958: C      end if
959:
960:      end if
961: C
962:      if ( VNM.eq.'ETOPP' ) then
963:          call VALUE0( VNAME, ICALL, 'R8', ETOPIX(KPPHOT) )
964:          ETOPI = ETOPIX(KPPHOT)
965:      end if
966:      if ( VNM.eq.'EBOTP' ) then
967:          call VALUE0( VNAME, ICALL, 'R8', EBOTX(KPPHOT) )
968:          EBOT = EBOTX(KPPHOT)
969:      end if

```

```

970: C
971:      if ( VNM.eq.'ETHMAX' ) call VALUE0( VNAME, ICALL, 'R4', ETHMAX )
972:      if ( VNM.eq.'AMLIM' ) call VALUE0( VNAME, ICALL, 'R4', AMLIM )
973:      if ( VNM.eq.'EWCUT' ) call VALUE0( VNAME, ICALL, 'R4', EWCUT )
974:      if ( VNM.eq.'NSTAL' ) call VALUE0( VNAME, ICALL, 'R4', NSTAL )
975:      if ( VNM.eq.'EINCD' ) call VALUE0( VNAME, ICALL, 'R4', EINCD )
976:      if ( VNM.eq.'MB' ) call VALUE0( VNAME, ICALL, 'I4', MB )
977: C
978: C
979: C-----
980: C ..... OTHERS (ARRAYS)
981: C      (SIGN CHECK (CALL CKVAL4 OR CKVAL8) ADDED 1/17/1990)
982: C-----
983: C
984: C-----
985: C ...../PVRED/.....
986: C-----
987: C
988: C
989: C-----
990: C ...../PGEOM/.....
991: C-----
992: C
993:      if ( VNM.eq.'KMAT' .or. VNM.eq.'KREG' ) then
994:          write(IMG,7020)
995:          call CNTERR( 'FATAL' )
996:          NDUM2 = NINPZ
997:          call DMREAD( VNM, NDUM1, NDUM2, IRET )
998:          if ( IRET.ne.0 ) go to 240
999:      end if
1000: 7020 format(/' XXX(intro) Old-fashioned input for mat # & region #'
1001: &/' Specifying "KMAT" & "KREG" separately from the zone data ',
1002: &'block of geometry input is not allowed in this version.')
1003: C
1004:      if ( VNM.eq.'VOL' ) then
1005:          call ARAYIN( VNAME, A, ICALL, 'R4', LVOL, ICK, JDEBG, LAST,
1006: &          NINPZ, 'NINPZ' )
1007:          call CKVAL4( VNAME, A(LVOL), NINPZ, IFL, 1 )
1008:          if ( IFL.ne.0 ) then
1009:              ICHK = ICHK + 1
1010:              write(IMG,7080) VNAME(:NLEN)
1011:              call CNTERR( 'FATAL' )
1012:          end if
1013:      end if
1014: C
1015:      if ( VNM.eq.'PAPER' ) then
1016:          call ARAYIN( VNAME, A, ICALL, 'R4', LPAPER, ICK, JDEBG, LAST,
1017: &          12*NPICT, ' ' )
1018:      end if
1019: C
1020: C-----
1021: C ...../PSOUR/.....
1022: C-----
1023: C
1024:      if ( VNM.eq.'KSOUR' ) then
1025:          call ARAYIN( VNAME, A, ICALL, 'I4', LKSOUR, ICK, JDEBG, LAST,
1026: &          NSOUR, 'NSOUR' )
1027:      end if
1028:      if ( VNM.eq.'ISZON' ) then
1029:          call ARAYIN( VNAME, A, ICALL, 'I4', LISZON, ICK, JDEBG, LAST, 2
1030: &          *NSOUR, ' ' )
1031:      end if
1032:      if ( VNM.eq.'SOUR' ) then
1033:          call ARAYIN( VNAME, A, ICALL, 'R8', LSOUR, ICK, JDEBG, LAST,
1034: &          NSOUR, 'NSOUR' )

```

src/mvp/intro.f

```

1035:      call CKVAL8( VNAME, A(LSOUR), NSOUR, IFL, 0 )
1036:      if ( IFL.ne.0 ) then
1037:        ICHK = ICHK + 1
1038:        write(IMG,7080) VNAME(:NLN)
1039:        call CNTERR( 'FATAL' )
1040:      end if
1041:    end if
1042:    if ( VNM.eq.'PSPAC' ) then
1043:      call ARAYIN( VNAME, A, ICALL, 'R4', LPSPAC, ICK, JDEBG, LAST,
1044: &      10*NSOUR, ' ' )
1045:    end if
1046:
1047:    if ( VNM.eq.'PENRG' ) then
1048:      call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
1049: &      NGROUP )
1050:      call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
1051:
1052: C      if ( JNP.eq.0 .or. VSUBF.eq.' ' ) then
1053: C        call ARAYIN( VNAME, A, ICALL, 'R4', LPENRG, ICK, JDEBG,
1054: C &      LAST, NGROUP*NSOUR, ' ' )
1055: C(TMP) 6.MAR.92. .... CALL CKVAL4(VNM,A(LPENRG),NGROUP*NSOUR,ICK,0)
1056: C      else
1057:
1058:        if ( IKPID.eq.0 .or. KKPID.ne.0 ) then
1059:
1060:          call ARAYIN( VNAME, A, ICALL, 'R4', LPENRG, ICK, JDEBG,
1061: &          LAST, NGROUP*NSOUR, ' ' )
1062:
1063:        else if ( IKPID.ne.0 ) then
1064:
1065:          if ( LPENRG.eq.0 ) then
1066:            call KEPV( A(1), 'PENRG', LPENRG, NGROUP*NSOUR, 'R4',
1067: &            LAST, 0.0, JDEBG )
1068:          end if
1069:
1070: CCCCC if ( VSUBF.eq.'N' ) then
1071: C      .... input data temporary from A(LTMPRY)
1072: C
1073: C      call GTLAST( LASTTM )
1074: C      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1075: C &      LAST, NGP(IKPID)*NSOUR, ' ' )
1076: C
1077: C      .... scatter data a(ltmpy) onto a(lpenrg) ...
1078: C
1079: C      call ASCTR4( VNAME, A(LPENRG), A(LTMPRY), NGROUP, NSOUR,
1080: C &      NGP(IKPID), KNGP(IKPID), NGP(IKPID) )
1081: C
1082: C      call STLAST( 'PENRG', LASTTM, LAST )
1083: C
1084: C      else if ( VSUBF.eq.'P' ) then
1085: C
1086: C      .... input data temporary from A(LTMPRY)
1087: C
1088: C      call GTLAST( LASTTM )
1089: C      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1090: C &      LAST, NGP2*NSOUR, ' ' )
1091: C
1092: C      .... scatter data a(ltmpy) onto a(lpenrg) ...
1093: C
1094: C      call ASCTR4( VNAME, A(LPENRG), A(LTMPRY), NGROUP, NSOUR,
1095: C &      NGP2, NGP1+1, NGP2 )
1096: C
1097: C      call STLAST( 'PENRG', LASTTM, LAST )
1098: C
1099: C      end if

```

```

1100:      end if
1101:    end if
1102: C
1103:    if ( VNM.eq.'NSTIM' ) then
1104:      call ARAYIN( VNAME, A, ICALL, 'I4', LNSTIM, ICK, JDEBG, LAST,
1105: &      NSOUR, 'NSOUR' )
1106:    end if
1107:
1108:    if ( VNM.eq.'STIM' ) then
1109: C      .... MNSTIM IS SUM OF NSTIM ....
1110: C      call ARYSUM( A(LNSTIM), NSOUR, MNSTIM )
1111: C      call ARAYIN( VNAME, A, ICALL, 'R4', LSTIM, ICK, JDEBG, LAST,
1112: C &      MNSTIM, ' ' )
1113: C
1114: C      end if
1115: C      if ( VNM.eq.'PSTIM' ) then
1116: C        call ARYSUM( A(LPSTIM), NSOUR, MNSTIM )
1117: C        call ARAYIN( VNAME, A, ICALL, 'R4', LPSTIM, ICK, JDEBG, LAST,
1118: C &      MNSTIM, ' ' )
1119: C      end if
1120: C      if ( VNM.eq.'NSANG' ) then
1121: C        call ARAYIN( VNAME, A, ICALL, 'I4', LNSANG, ICK, JDEBG, LAST,
1122: C &      NSOUR, 'NSOUR' )
1123: C      end if
1124: C      if ( VNM.eq.'SANG' ) then
1125: C
1126: C      .... MNSANG IS SUM OF NSANG ....
1127: C      call ARYSUM( A(LNSANG), NSOUR, MNSANG )
1128: C      call ARAYIN( VNAME, A, ICALL, 'R4', LSANG, ICK, JDEBG, LAST,
1129: C &      MNSANG, ' ' )
1130: C      end if
1131: C      if ( VNM.eq.'PSANG' ) then
1132: C        call ARYSUM( A(LNSANG), NSOUR, MNSANG )
1133: C        call ARAYIN( VNAME, A, ICALL, 'R4', LPSANG, ICK, JDEBG, LAST,
1134: C &      MNSANG, ' ' )
1135: C      end if
1136: C      if ( VNM.eq.'IFISM' ) then
1137: C        call FISMIN( IPR, LIFISM, NSOUR, A, LAST, CHA(LNUCID), NUC,
1138: C &      JDEBG )
1139: C      ICALL = 1
1140: C      end if
1141: C
1142: C-----
1143: C ...../PTALY/.....
1144: C-----
1145: C
1146:    if ( VNM.eq.'RESP' ) then
1147:      call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
1148: &      NGROUP )
1149:      call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
1150: CCCCC if ( JNP.eq.0 .or. VSUBF.eq.' ' ) then
1151: C      if ( IKPID.eq.0 .or. KKPID.ne.0 ) then
1152: C        call ARAYIN( VNAME, A, ICALL, 'R4', LRESP, ICK, JDEBG, LAST,
1153: C &      NGROUP*NRESP, ' ' )
1154: C        call CKVAL4( VNM, A(LRESP), NGROUP*NRESP, IFL, 0 )
1155: C        if ( IFL.ne.0 ) then
1156: C          ICHK = ICHK + 1
1157: C          write(IMG,7080) VNAME(:NLN)
1158: C          call CNTERR( 'FATAL' )
1159: C        end if
1160: C      else
1161: C        if ( LRESP.eq.0 ) then
1162: C          call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4', LAST,
1163: C &          0.0, JDEBG )
1164: C        end if

```

src/mvp/intro.f

```

1165:         if ( IKPID.ne.0 ) then
1166:             call GTLAST( LASTTM )
1167:             call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1168: &                 LAST, NGP(IKPID)*NRESP, ' ' )
1169: C
1170: C             .... scatter data a(ltmpry) onto a(lresp) ...
1171: C
1172: C             call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
1173: &                 NGP(IKPID), KNGP(IKPID), NGP(IKPID) )
1174: C
1175: C             call STLAST( 'RESP', LASTTM, LAST )
1176: C             end if
1177: C
1178: C             if ( VSUBF.eq.'N' ) then
1179: C
1180: C                 .... input data temporary from A(LTMPRY)
1181: C
1182: C                 call GTLAST( LASTTM )
1183: C                 call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1184: &                     LAST, NGP1*NRESP, ' ' )
1185: C
1186: C                 .... scatter data a(ltmpry) onto a(lpenrg) ...
1187: C
1188: C                 call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
1189: &                     NGP1, 1, NGP1 )
1190: C
1191: C                 call STLAST( 'RESP', LASTTM, LAST )
1192: C             else if ( VSUBF.eq.'P' ) then
1193: C
1194: C                 .... input data temporary from A(LTMPRY)
1195: C
1196: C                 call GTLAST( LASTTM )
1197: C                 call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1198: &                     LAST, NGP2*NRESP, ' ' )
1199: C
1200: C                 .... scatter data a(ltmpry) onto a(lpenrg) ...
1201: C
1202: C                 call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
1203: &                     NGP2, NGP1+1, NGP2 )
1204: C
1205: C                 call STLAST( 'RESP', LASTTM, LAST )
1206: C             end if
1207: C             end if
1208: C             end if
1209: C
1210: C             if ( VNM.eq.'ENGYB' .or. VNM.eq.'ENGPB' ) then
1211: C
1212: C                 call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
1213: &                     NGROUP )
1214: C                 call KGPSSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
1215: C
1216: C                 if ( LENGYB.eq.0 ) then
1217: C                     call KEPV( A(1), 'ENGYB', LENGYB, KENGP(KPLIM+1)-1, 'R4',
1218: &                         LAST, 0.0, JDEBG )
1219: C                 end if
1220: C
1221: C                 KP = 0
1222: C                 if ( VNM.eq.'ENGPB' ) then
1223: C                     KP = KPPHOT
1224: C                     if ( JKPAR(KPPHOT).eq.0 ) then
1225: C                         KP = 0
1226: C                     end if
1227: C                 else if ( IKPID.eq.0.and.JKPAR(KPNEUT).ne.0 ) then
1228: C                     KP = KPNEUT
1229: C                 else if ( IKPID.ne.0 ) then

```

```

1230:             KP = IKPID
1231: C             end if
1232: C             if ( KP.eq.0 ) then
1233: C                 write(IMG,*) '!!! Energy boundary data <', VNAME(:NLEN),
1234: &                     '> is not effective for current input.'
1235: C                 call DMREAD( VNAME(:NLEN), IDUMMY, IDUMMY, IRET )
1236: C                 if ( IRET.ne.0 ) go to 240
1237: C                 go to 210
1238: C             end if
1239: C
1240: C             NB = NGP(KP) + 1
1241: C             call R4READ( VNAME(:NLEN), A(LENGYB+KENGP(KP)-1), NA, -NB, IRET
1242: &                 )
1243: C             if ( NA.ne.NB ) then
1244: C                 write(IMG,'(1x,a,a,i5,a,i5,a)')
1245: C                 &                 'XXX(intro) Number of energy boundary data <',
1246: C                 &                 VNAME(:NLEN), '> (= ', NA,
1247: C                 &                 ' )i is different from an expected one (= ', NB, ').'
1248: C                 ICHK = ICHK + 1
1249: C                 call CNTERR( 'FATAL' )
1250: C             end if
1251: C
1252: C             call CKVAL4( VNM, A(LENGYB+KENGP(KP)-1), NGP(KP)+1, IFL, 1 )
1253: C             if ( IFL.ne.0 ) then
1254: C                 ICHK = ICHK + 1
1255: C                 write(IMG,7080) VNAME(:NLEN)
1256: C                 call CNTERR( 'FATAL' )
1257: C             end if
1258: C             if ( KP.eq.KPPHOT ) then
1259: C                 if ( LENGPB.eq.0 ) then
1260: C                     call KEPV( A(1), 'ENGPB', LENGPB, NGP(KP)+1, 'R4', LAST,
1261: &                         0.0, JDEBG )
1262: C                 end if
1263: C                 call ATRANS( A(LENGYB+KENGP(KP)-1), A(LENGPB), NGP(KP)+1 )
1264: C                 end if
1265: C                 ICALL = 1
1266: C             end if
1267: C
1268: C             ... obsoleted energy boundary input procedure(from Apr 2000)
1269: C             scheme to determine number of energy group from
1270: C             number of energy boundary data is no longer effective.
1271: C             (backward compatibility may be lost by this ???)
1272: C
1273: C             if ( VSUBF.eq.'N' .or. VSUBF.eq.' ' ) then
1274: C                 call ARAYIN( VNAME, A, ICALL, 'R4', LENGYB, ICK, JDEBG,
1275: &                     LAST, NGP1+1, ' ' )
1276: C                 call SIZCHK( VNAME, 'NGP1', NGP1, NFIMP(0)-1, ICK )
1277: C
1278: C                 call CKVAL4( VNM, A(LENGYB), NGP1+1, IFL, 1 )
1279: C                 if ( IFL.ne.0 ) then
1280: C                     ICHK = ICHK + 1
1281: C                     write(IMG,7080) VNAME(:NLEN)
1282: C                     call CNTERR( 'FATAL' )
1283: C                 end if
1284: C             else if ( VSUBF.eq.'P' ) then
1285: C                 call ARAYIN( VNAME, A, ICALL, 'R4', LENGPB, ICK, JDEBG,
1286: &                     LAST, NGP2+1, ' ' )
1287: C                 call SIZCHK( VNAME, 'NGP2', NGP2, NFIMP(0)-1, ICK )
1288: C                 call CKVAL4( VNAME, A(LENGPB), NGP2+1, IFL, 1 )
1289: C                 if ( IFL.ne.0 ) then
1290: C                     ICHK = ICHK + 1
1291: C                     write(IMG,7080) VNAME(:NLEN)
1292: C                     call CNTERR( 'FATAL' )
1293: C                 end if
1294: C             end if

```

src/mvp/intro.f

```

1295: C      end if
1296: C
1297: C      if ( VNM.eq.'ENGPB' ) then
1298: C          call ARAYIN( VNAME, A, ICALL, 'R4', LENGPB, ICK, JDEBG, LAST,
1299: C      &      NGP2+1, ' ' )
1300: C          call SI2CHK( VNAME, 'NGP2', NGP2, NFIMP(0)-1, ICK )
1301: C          call CKVAL4( VNM, A(LENGPB), NGP2+1, IFL, 1 )
1302: C          if ( IFL.ne.0 ) then
1303: C              ICHK = ICHK + 1
1304: C              write(IMG,7080) VNAME(:NLEN)
1305: C              call CNTERR( 'FATAL' )
1306: C          end if
1307: C      end if
1308: C
1309: C      if ( VNM.eq.'TIMEB' ) then
1310: C          call ARAYIN( VNAME, A, ICALL, 'R4', LTIMEB, ICK, JDEBG, LAST,
1311: C      &      NTIME+1, ' ' )
1312: C          call SI2CHK( VNAME, 'NTIME', NTIME, NFIMP(0)-1, ICK )
1313: C      end if
1314: C
1315: C      if ( VNM.eq.'WTIME' ) then
1316: C          call ARAYIN( VNAME, A, ICALL, 'R4', LWTIME, ICK, JDEBG, LAST,
1317: C      &      NTIME, ' ' )
1318: C      end if
1319: C
1320: C-----
1321: C      .... special tally (using dosimetry file).....
1322: C-----
1323: C
1324: C      if ( VNM.eq.'STAL' .or. IMATCH('CRESP',VNM).ne.0 ) then
1325: C          call CRSINP( VNAME, A, IA, LICRES, NICRES, NSTAL, JDEBG )
1326: C          ICALL = 1
1327: C      end if
1328: C
1329: C-----
1330: C      .... target position of path stretching .....
1331: C-----
1332: C
1333: C      if ( VNM.eq.'PSXYZ' ) then
1334: C
1335: C          call ARAYIN( VNAME, A, ICALL, 'R8', LPSXYZ, ICK, JDEBG, LAST,
1336: C      &      3, ' ' )
1337: C
1338: C          if ( MOD(NFIMP(0),3).ne.0 ) then
1339: C              write(IMG,7040) NFIMP(0)
1340: C              call CNTERR( 'FATAL' )
1341: C          else if ( NFIMP(0).eq.0 ) then
1342: C              call PUTVD( A(LPSXYZ), 3, 0.0D0 )
1343: C          end if
1344: C      end if
1345: C      7040 format('// XXX(intro) Number of input data PSXYZ() is ',I4,
1346: C      & ' while a multiple of 3 is required.')
1347: C
1348: C-----
1349: C      .... POINT DETECTOR : INPUT ONLY 5 PARAMETERS HERE. ....
1350: C-----
1351: C
1352: C      if ( VNM.eq.'XPDET' ) then
1353: C          call ARAYIN( VNM, A, ICALL, 'R8', LXPDET, ICK, JDEBG, LAST,
1354: C      &      NPDET*5, ' ' )
1355: C      end if
1356: C      if ( VNM.eq.'SPDET' ) then
1357: C          call ARAYIN( VNM, A, ICALL, 'I4', LJSPDT, ICK, JDEBG, LAST,
1358: C      &      NPDET, ' ' )
1359: C      end if

```

```

1360: C
1361: C-----
1362: C      .... INPUT FOR CUSTOMIZED VERSION .....
1363: C-----
1364: C
1365: C      $CUSTOM
1366: C      ....
1367: C      $END CUSTOM
1368: C
1369: C      ( ignored in the standard version )
1370: C
1371: C      if ( VNAME.eq.'$CUSTOM' ) then
1372: C          call CUSTOM( IOIN, IPR, A, A, CHA )
1373: C          ICALL = 1
1374: C          go to 210
1375: C      end if
1376: C
1377: C-----
1378: C      .... TRIED TO INPUT INVALID ARRAY OR VARIABLES !!!! .....
1379: C-----
1380: C
1381: C      if ( ICALL.eq.0 ) then
1382: C          write(IMG,7060) VNAME(:ICLEN(VNAME))
1383: C          call CNTERR( 'WARNING' )
1384: C          call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
1385: C          if ( IRET.ne.0 ) go to 240
1386: C      end if
1387: C      go to 210
1388: C      7060 format(' !!!(intro) Input data whose name is <',A,
1389: C      & '> does not exist or direct input is not allowed !!!')
1390: C
1391: C
1392: C=====
1393: C
1394: C
1395: C      INPUT END .....
1396: C
1397: C      220 continue
1398: C
1399: C      ... check perturbation block
1400: C
1401: C      if(JPERT.eq.1 .and. KJPERT.eq.0) then
1402: C          write(IMG,'(1X,A,A)')
1403: C      &      'XXX(intro) PERTURBATION block does not exist in ',
1404: C      &      'INPUT DECK for PERTURBATION-CALCULATION.'
1405: C          ICHK = ICHK + 1
1406: C          call CNTERR('FATAL')
1407: C      end if
1408: C      if(JPERT.eq.0 .and. KJPERT.eq.1) then
1409: C          write(IMG,'(1X,A,A,A,A)')
1410: C      &      '!!!(intro) PERTURBATION block exists in ',
1411: C      &      'INPUT DECK. But PERTURBATION option is',
1412: C      &      ' not specified.',
1413: C      &      ' PERTURBATION-CALCULATION is skipped.'
1414: C          call CNTERR('WARNING')
1415: C      end if
1416: C      call PTINCK(IMG, NREG, A(LWGTF))
1417: C
1418: C      if ( LWGTF.ne.0 .and. JDLYN.ne.0 .and. JFISS.ne.0 ) then
1419: C          if ( JEIGN.ne.0 .or. LWGTFD.eq.0 ) then
1420: C              do I = 1, NREG
1421: C                  A(LWGTF+I-1+NREG) = A(LWGTF+I-1)
1422: C              end do
1423: C              write(IMG,'(1X,A,A)')
1424: C      &      '!!!(INTRO) Delayed fission neutron weights are ',

```


src/mvp/intro.f

```

1425:      &          'set to be the same as prompt ones,'
1426:      if ( JEIGN.ne.0 ) then
1427:        write(IMG, '(1X,A,A)')
1428:      &          'because the same weights are assumed ',
1429:      &          'in an eigenvalue problem.'
1430:      else
1431:        write(IMG, '(1X,A)')
1432:      &          'because no WGTFD data are input.'
1433:      end if
1434:      call CNTERR( 'WARNING' )
1435:    end if
1436:  end if
1437:
1438: C-----
1439: C .... read point-wise response functions (obsolete)
1440: C-----
1441: C if ( NSTAL.gt.0 .and. NICRES.gt.0 ) then
1442: C   call KEEP( 'CRES', LCRES, NSTAL*3, 'I4', LAST, JDEBG )
1443: C   call REMAIN( 'CRES', NLTEMP, 'R4', LAST )
1444: C   if ( NLTEMP.le.0 ) then
1445: C     write(IPR,*)
1446: C   &     'XXX no-more memory to store point-wise responses !!!'
1447: C     call CNTERR( 'FATAL' )
1448: C   else
1449: C     call KEEP( 'CRES', LCRES, NLTEMP, 'R4', LAST, JDEBG )
1450: C     call XSECD( A(LICRES), A(LKRES), A(LCRES), LCRES, LAST,
1451: C   &     LIMIT, NLTEMP, NSTAL, NICRES, MCRES )
1452: C   ... release of unused memory is carried out in XSECD ...
1453: C   call RESIZE( 'CRES', LCRES, MCRES, 'R4', LAST )
1454: C   end if
1455: C   end if
1456: C   end if
1457: C   end if
1458: C   end if
1459: C   end if
1460: C
1461: 7080 format(// 'XXX(intro) Data <',A,'> has a value with invalid sign.'
1462: &/)
1463: C
1464: if ( ICHK.ne.0 ) then
1465:   write(IMG,7100)
1466: 7100 format('1'///1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX')/1X,
1467: &          'Your input includes data of invalid sign !!'//1X,
1468: &          'Please check input data or error-messages !!!'//
1469: &          1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX'))
1470:   call CNTERR( 'FATAL' )
1471: end if
1472: if ( ICK.ne.0 ) then
1473:   write(IMG,7120)
1474: 7120 format('1'///1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX')/1X,
1475: &          'Your input includes data of invalid length !!'//
1476: &          1X, 'Please check input data or error messages !!!'
1477: &          //1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX'))
1478:   call CNTERR( 'FATAL' )
1479: end if
1480: C7100 format(//
1481: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1482: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1483: C &' '
1484: C &' Your input includes data of invalid sign !!'
1485: C &' Please check input data or error messages !!!'
1486: C &' '
1487: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1488: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1489: C &)
1490: C7120 format(//
1491: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1492: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1493: C &' '
1494: C &' Your input includes data of invalid length !!'
1495: C &' Please check input data or error messages !!!'
1496: C &' '
1497: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1498: C &' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
1499: C &)
1500: C
1501: C-----
1502: C PREPARE NON-INPUT PARAMETERS
1503: C-----
1504: C
1505: JNRUN = 0
1506: C
1507: call INTRO2( A, H, CHA, LIMIT, LIMITL, LIMITC, TITLE, MAXSF, JNP,
1508: & TCUTDM )
1509: C
1510: if ( KPNPRD.ne.0 ) then
1511:   if ( JFIXD.ne.0 .and. KIBNK(17)+KIBNK(18)+KIBNK(19).le.0 ) ! photo-nu
1512:   & KPNPRD = 0 ! photo-nu
1513: end if ! photo-nu
1514: C
1515: C ( IF JNRUN > 0 , NO RANDOM WALK NECESSARY )
1516: C
1517: call CPUTM( T1 )
1518: write(IPR, '(1X,A,1P,E12.4,A)')
1519: & '==== CPU TIME FOR DATA PREPARATION:', T1 - T0, ' SEC'
1520: C
1521: C-----
1522: C ..... DRAW PICTURES
1523: C-----
1524: C
1525: if ( JPICT.ne.0 ) then
1526:   call PABLO( A, LIMIT, LWORK, NPICT, NZONE, NSDA, NZDA, NCELL,
1527: & JLATT, JTLT, JVMNT, JSIMP, JDEBG, A(LIDCEL),
1528: & A(LIPCEL), A(LKCELL), A(LSDA), A(LKINPZ), A(LKMAT),
1529: & A(LKZREG), A(LKZDA), A(LKZAA), A(LPAPER) )
1530: end if
1531: C
1532: C/#IF PARA(PVM MPI)
1533: C
1534: C ... skip random number for sub tasks & reset random number parameters
1535: C
1536: do 230 I = 0, IDTASK - 1
1537:   call RANU2( IRAND, RTEMP, 1, ICON )
1538: 230 continue
1539: C
1540: NRSTEP = NTASK
1541: if ( MOD(NTASK,2).eq.0 ) NRSTEP = NRSTEP + 1
1542: call RNINIT( NRSTEP )
1543: C/#ENDIF
1544: C
1545: C-----
1546: C ..... PRINT CONTENTS OF COMMON /SIZES/ AND OTHER NON-ARRAY DATA
1547: C-----
1548: C
1549: call PRSIZE( IPR )
1550: C

```

src/mvp/intro.f

```
1551: C-----
1552: C ..... DEBUG PRINT OF COMMONS .....
1553: C-----
1554: C
1555:       if ( JDEBG(1).ne.0 ) call CHECK( ' === END OF INTRO === ' )
1556: C
1557: C-----
1558: C ..... CHECK INPUT OR PROCESSING ERROR .....
1559: C-----
1560: C
1561:       call PBNERR( IPR, 'AFTER DATA INPUT' )
1562:       call CHKERR( 'WARNING', KK )
1563:       if ( KK.gt.0 ) then
1564:         write(IPR,7140)
1565: 7140       format(// ' *** SOME WARNING MESSAGES HAVE BEEN ISSUED. ****'//
1566: & ' I recommend you checking your input data',
1567: & ' once more.'// ' Warning message is printed',
1568: & ' with "!!!" symbol on the head of printed line.'// )
1569:       end if
1570: C
1571:       JSTOP = 0
1572:       call CHKERR( 'FATAL', KK )
1573: C
1574:       if ( KK.gt.0 ) then
1575:         JSTOP = 1
1576:         write(IPR,7160)
1577: 7160       format('1',' *** FATAL ERRORS HAVE BEEN DETECTED',
1578: & ' IN INPUT-PHASE. STOP EXECUTION. ***'//
1579: & ' Message for fatal error is printed',
1580: & ' with "XXX" symbol on the head of printed line.'//
1581: & ' You must check your input data carefully !! ' )
1582: C/#IF UNIX
1583:       call FLUSHSTD( )
1584: C/#ENDIF
1585:       stop 888
1586:       end if
1587: C
1588: C-----
1589: C       IF JNRUN > 0, NO RANDOM WALK WILL BE DONE.
1590: C-----
1591: C
1592: CCC   if ( JNRUN.ne.0 ) JSTOP = 1
1593: C
1594: C-----
1595: C       Check flag JRUNM :
1596: C-----
1597: C
1598:       if ( MOD(JRUNM,10).ne.0 .or. MOD(JRUNM/10,10).ne.0 ) then
1599:         JSTOP = 1
1600:         write(IPR,7180) JRUNM
1601: 7180       format('1'/1X,'=== Stopped before random walk with ',
1602: & ' "RUN-MODE" control option : RUN-MODE = ',I9)
1603:       end if
1604:       return
1605: C
1606: C-----
1607: C       Error in 'DMREAD':
1608: C-----
1609: C
1610:       240 write(IMG,7200)
1611: 7200       format(' XXX(intro) Unexpected end of data or data read error',
1612: & ' during skipping unnecessary/invalid data.')
1613:       call PRSTOP( 1, 'ERROR DURING SKIPPING INPUT DATA.' )
1614:       stop 888
1615: C
```

1616: end

src/mvp/isnds.f

```
1:      logical function ISNDS(IDNUC)
2:      c
3:      c=====
4:      c Purpose: ISNDS (=IS_Nuclide_of_Doppler_Scattering) returns whether
5:      c           the input nuclide ID is addressed with Doppler scattering
6:      c           or not.
7:      c-----
8:      c arguments (i=input)
9:      c i NUCID : nuclide ID to be checked.
10:     c=====
11:     integer IDNUC, IRETV
12:     include '../shared/INC/_ARRAY'
13:     include 'INC/_KPIDS'
14:     include 'INC/_CXSEC'
15:
16:     call JUDGE_NUC_DOP_SCAT(IDNUC, IRETV, NUC, A(LINTDS))
17:     if (IRETV.eq.0) then
18:       ISNDS = .false.
19:     else
20:       ISNDS = .true.
21:     end if
22:     return
23:     end
24:
25:     subroutine JUDGE_NUC_DOP_SCAT(INUC, IRETV, NUC, INTDS)
26:     integer INUC, IRETV
27:     integer INTDS(NUC)
28:     IRETV = INTDS(INUC)
29:     return
30:     end
31:
32:     subroutine MKINTDS(NUC, NMT, KLIB2, INTDS)
33:     c=====
34:     c Purpose: MKINTDS (=MaKe_Nuclide_Table_for_Doppler_Scattering)
35:     c           makes a table of nuclides for which Doppler scattering is
36:     c           considered.
37:     c-----
38:     c arguments (i=input, o=output)
39:     c i NUC : # of nuclides.
40:     c i NMT : # of MT's.
41:     c i KLIB2 : Pointers to #2 record data in MVP library.
42:     c o INTDS : Nuclide table for Doppler scattering.
43:     c=====
44:     integer NUC, NMT
45:     integer KLIB2(NUC,NMT,21), INTDS(NUC)
46:     do INUC = 1, NUC
47:       MTINFO = KLIB2(INUC,120,1)
48:       if (MTINFO.gt.0) then
49:         INTDS(INUC) = 1
50:       end if
51:     end do
52:     return
53:     end
```

src/mvp/keff0.f

```

1:      subroutine KEFF0( NBATCH,IPBT,  NSKIP, JPR,   XSOC,  XKEF,
2:      &                  XKAV1, XKER1, XKAV2, XKER2, XKAV3, XKER3,
3:      &                  XKEF1, XSOCB, XKB,
4:      &                  JPHNU ) ! photo-nuc
5: C=====
6: C  PURPOSE: Statistical analysis of k-effective (called after some
7: C           batches for monitoring).
8: C           Calculate average form first batch to batch IPBT
9: C           and M.L.E.'s from batch NSKIP+1 to batch IPBT.
10: C  CALLED IN: TALSUM
11: C  CALL      : MTXINV
12: C-----
13: C  arguments (i=input, o=output, w=work).
14: C
15: C i NBATCH : number of batches (probably calculated until now)
16: C i IPBT   : batch # for which tally analysis is performed.
17: C i NSKIP  : number of skipped batch for k-eff estimation
18: C i JPR    : printout if non zero
19: C i XSOC(I) : neutron source weight sum until batch I
20: C i XKEF(I) : neutron production (fission) sum until batch I
21: C o XKAV1(6),XKER1(6) : averaged K-eff and % error of 6 way
22: C                       from batch 1 to IPBT.
23: C                       ( production(track/collision/analog)
24: C                       and balance(track/collision/analog) )
25: C o XKAV2(6),XKER2(6) : averaged K-eff and % error of 6 way
26: C                       from batch NSKIP+1 to IPBT.
27: C                       ( production(track/collision/analog)
28: C                       and balance(track/collision/analog) )
29: C o XKAV3(6),XKER3(6) : averaged M.L.E. K-eff and % error of 6 way
30: C                       from batch NSKIP+1 to IPBT.
31: C                       ( track/collision/analog
32: C                       and production/n-balance/all )
33: C w XKEF1(7,NBATCH),XSOCB(NBATCH),XKB(7,NBATCH): working array
34: C=====
35:      implicit real*8(A-H,O-Z)
36:      parameter (MAXXK = 10) ! photo-nuc
37: C
38:      include '../shared/INC/_IUNIT'
39:      include 'INC/_IUNIT2'
40: C
41:      real*8 XSOC(NBATCH)
42:      real*8 XKEF(MAXXK,NBATCH)
43: C
44: C ... until IPBT'th batch
45:      real*8 XKAV1(MAXXK)
46:      real*8 XKER1(MAXXK)
47: C ... after NSKIP'th batch
48:      real*8 XKAV2(MAXXK)
49:      real*8 XKER2(MAXXK)
50: C ... M.L.E. after NSKIP'th batch
51:      real*8 XKAV3(MAXXK)
52:      real*8 XKER3(MAXXK)
53: C
54: C ... working array
55: C
56:      real*8 XKEF1(MAXXK,NBATCH)
57:      real*8 XSOCB(NBATCH)
58:      real*8 XKB(MAXXK,NBATCH)
59: C
60: C ... criterion to reject too mach correlated combination of
61: C     estimation in maximum likelihood estimation.
62: C
63:      parameter( DECORR = 1.0D-5 )
64: C
65: C ... local data

```

```

66: C
67:      real*8 X1, X2, X3, X4, X5, X6, S1
68:      character*32 TAG
69:      character*48 ESTCMB
70: C
71:      real*8 COVXK(6,6)
72:      real*8 CORR(6,6)
73:      real*8 XKSD(6)
74:      integer IWK1(6)
75: C
76:      integer JJCC(6)
77:      integer JJCC0(6)
78: C
79:      real*8 AVG(16), SIGA(16), SIGR(16)
80:      character*48 ESTKEF(16)
81: C
82: C ... mixing factor of M.L.E.'s ...
83: C
84: C       XKAV1(J) = Sum[I] ( XKB(I)*FMIX(I,J) )
85: C
86:      real*8 FMIX(6,6)
87: C
88: C-----
89: C-----
90: C-----
91: C**** STATISTICAL ANALYSIS OF K-EFFECTIVE
92: C-----
93: C
94:      XSOCB(1) = XSOC(1)
95: C
96:      XKEF1(7,1) = XKEF(7,1)
97:      XKEF1(1,1) = XKEF(1,1)
98:      XKEF1(2,1) = XKEF(2,1)
99:      XKEF1(3,1) = XKEF(3,1)
100:      XKEF1(4,1) = XKEF(4,1) + XKEF1(7,1)
101:      XKEF1(5,1) = XKEF(5,1) + XKEF1(7,1)
102:      XKEF1(6,1) = XKEF(6,1) + XKEF1(7,1)
103:      if(JPHNU.ne.0) then ! photo-nuc
104:          XKEF1(8,1) = XKEF(8,1) ! photo-nuc
105:          XKEF1(9,1) = XKEF(9,1) ! photo-nuc
106:          XKEF1(10,1) = XKEF(10,1) ! photo-nuc
107:      end if ! photo-nuc
108: C
109:      XKB(1,1) = XKEF1(1,1) /XSOC(1)
110:      XKB(2,1) = XKEF1(2,1) /XSOC(1)
111:      XKB(3,1) = XKEF1(3,1) /XSOC(1)
112:      XKB(4,1) = XKEF1(1,1) /XKEF1(4,1)
113:      XKB(5,1) = XKEF1(2,1) /XKEF1(5,1)
114:      XKB(6,1) = XKEF1(3,1) /XKEF1(6,1)
115:      if(JPHNU.ne.0) then ! photo-nuc
116:          XKB(8,1) = XKEF1(8,1) /XSOC(1) ! photo-nuc
117:          XKB(9,1) = XKEF1(9,1) /XSOC(1) ! photo-nuc
118:          XKB(10,1) = XKEF1(10,1)/XSOC(1) ! photo-nuc
119:      end if ! photo-nuc
120: CC
121: CC XSOCB : SOURCE (EACH BATCH)
122: CC XKB : ESTIMATOR OF KEFF (EACH BATCH)
123: CC XKAV1 : CUMULATIVE ESTIMATOR OF KEFF TO THIS BATCH
124: CC 1: TRACK LENGTH ESTIMATOR (PRODUCTION RATE)
125: CC 2: COLLISION ESTIMATOR (PRODUCTION RATE)
126: CC 3: ANALOG ESTIMATOR (PRODUCTION RATE)
127: CC 4: TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
128: CC 5: COLLISION ESTIMATOR (NEUTRON BALANCE)
129: CC 6: ANALOG ESTIMATOR (NEUTRON BALANCE)
130: CC 8: TRACK LENGTH ESTIMATOR (photoneutron production rate)

```

src/mvp/keff0.f

```

131: CC          9: COLLISION ESTIMATOR      (photoneutron production rate)
132: CC          10: ANALOG ESTIMATOR        (photoneutron production rate)
133: CC
134:      do 100 I = 2, IPBT
135:
136:          XSOCB(I) = XSOC(I) - XSOC(I-1)
137:
138:          XKEF1(7,I) = XKEF(7,I)
139:          XKEF1(1,I) = XKEF(1,I)
140:          XKEF1(2,I) = XKEF(2,I)
141:          XKEF1(3,I) = XKEF(3,I)
142:          XKEF1(4,I) = XKEF(4,I) + XKEF1(7,I)
143:          XKEF1(5,I) = XKEF(5,I) + XKEF1(7,I)
144:          XKEF1(6,I) = XKEF(6,I) + XKEF1(7,I)
145:          if(JPHNU.ne.0) then ! photo-nuc
146:              XKEF1(8,I) = XKEF(8,I) ! photo-nuc
147:              XKEF1(9,I) = XKEF(9,I) ! photo-nuc
148:              XKEF1(10,I) = XKEF(10,I) ! photo-nuc
149:          end if ! photo-nuc
150:
151:          XKB(1,I) = (XKEF1(1,I)-XKEF1(1,I-1)) /XSOCB(I)
152:          XKB(2,I) = (XKEF1(2,I)-XKEF1(2,I-1)) /XSOCB(I)
153:          XKB(3,I) = (XKEF1(3,I)-XKEF1(3,I-1)) /XSOCB(I)
154:          XKB(4,I) = (XKEF1(1,I)-XKEF1(1,I-1)) /
155:          & (XKEF1(4,I)-XKEF1(4,I-1))
156:          XKB(5,I) = (XKEF1(2,I)-XKEF1(2,I-1)) /
157:          & (XKEF1(5,I)-XKEF1(5,I-1))
158:          XKB(6,I) = (XKEF1(3,I)-XKEF1(3,I-1)) /
159:          & (XKEF1(6,I)-XKEF1(6,I-1))
160:          if(JPHNU.ne.0) then ! photo-nuc
161:              XKB(8,I) = (XKEF1(8,I)-XKEF1(8,I-1)) /XSOCB(I) ! photo-nuc
162:              XKB(9,I) = (XKEF1(9,I)-XKEF1(9,I-1)) /XSOCB(I) ! photo-nuc
163:              XKB(10,I) = (XKEF1(10,I)-XKEF1(10,I-1)) /XSOCB(I) ! photo-nuc
164:          end if ! photo-nuc
165:      100 continue
166:
167:          XKAV1(1) = XKEF1(1,IPBT) /XSOC(IPBT)
168:          XKAV1(2) = XKEF1(2,IPBT) /XSOC(IPBT)
169:          XKAV1(3) = XKEF1(3,IPBT) /XSOC(IPBT)
170: c ... source weighted "balance" estimators
171:          XKAV1(4) = 0.0D0
172:          XKAV1(5) = 0.0D0
173:          XKAV1(6) = 0.0D0
174:          do 110 J = 1, IPBT
175:              XKAV1(4) = XKAV1(4) + XKB(4,J)*XSOCB(J)
176:              XKAV1(5) = XKAV1(5) + XKB(5,J)*XSOCB(J)
177:              XKAV1(6) = XKAV1(6) + XKB(6,J)*XSOCB(J)
178:      110 continue
179:          XKAV1(4) = XKAV1(4) /XSOC(IPBT)
180:          XKAV1(5) = XKAV1(5) /XSOC(IPBT)
181:          XKAV1(6) = XKAV1(6) /XSOC(IPBT)
182:          if(JPHNU.ne.0) then ! photo-nuc
183:              XKAV1(8) = XKEF1(8,IPBT) /XSOC(IPBT) ! photo-nuc
184:              XKAV1(9) = XKEF1(9,IPBT) /XSOC(IPBT) ! photo-nuc
185:              XKAV1(10) = XKEF1(10,IPBT)/XSOC(IPBT) ! photo-nuc
186:          end if ! photo-nuc
187: c
188:          do 130 L = 1, 6
189:              XKER1(L) = 0.0D0
190:              if ( IPBT.gt.1 ) then
191:                  do 120 J = 1, IPBT
192:                      XKER1(L) = XKER1(L) + (XKB(L,J)-XKAV1(L))*2
193:      120          continue
194:                      XKER1(L) = SQRT(XKER1(L)/dble(IPBT)/(IPBT-1))
195:          & / XKAV1(L)*100.0D0
196:
197:          end if
198:      130 continue
199:          !+photo-nuc
200:          if(JPHNU.ne.0) then
201:              do L = 8, 10
202:                  XKER1(L) = 0
203:                  if ( IPBT.gt.1 ) then
204:                      do J = 1, IPBT
205:                          XKER1(L) = XKER1(L) + (XKB(L,J)-XKAV1(L))*2
206:                      end do
207:                      XKER1(L) = sqrt(XKER1(L)/dble(IPBT)/(IPBT-1)) /
208:                      & XKAV1(L)*100
209:                  end if
210:              end do
211:          !-photo-nuc
212:
213:          if ( JPR.ne.0 ) then
214:              write(IOT,7001)
215:      7001          format(1x,'+',6('+++++'))
216:              write(IOT,7000) IPBT
217:              write(IOT,7040) (XKAV1(L),XKER1(L),L=1,6)
218:      7000          format(1x,'+ * K-EFFECTIVE AVERAGE TO BATCH ',i8,' *'
219:              & /1X,'+',
220:              & ' TRACK LENGTH(PROD.) ',
221:              & ' COLLISION(PROD.) ',
222:              & ' ANALOG(PROD.) ',
223:              & ' TRACK LENGTH(BAL.) ',
224:              & ' COLLISION(BAL.) ',
225:              & ' ANALOG(BAL.) ' )
226:          if(JPHNU.ne.0) then ! photo-n
227:              write(IOT,7002) ! photo-n
228:              write(IOT,7040) (XKAV1(L),XKER1(L),L=8,10) ! photo-n
229:          end if ! photo-n
230:      7002 format(1X,'+', ' TRACK LENGTH(PN) ', ' COLLISION(PN) ', ! photo-n
231:          & ' ANALOG(PN) ' ) ! photo-n
232:          end if
233: c
234:          IS = NSKIP + 1
235: c
236:          if ( IPBT.gt.IS+3 ) then
237:              CCC
238:              CCC XSOC(I) : SOURCE. CUMULATIVE TO I'TH BATCH
239:              CCC
240:              CCC SIMILAR TO XKEF(6,I) KEFF (PRODUCTION & LOSS)
241:              CCC
242:              XSS = XSOC(IPBT)
243:              if ( NSKIP.gt.0 ) then
244:                  XSS = XSOC(IPBT) - XSOC(IS-1)
245:                  XKAV2(1) = (XKEF1(1,IPBT)-XKEF1(1,IS-1)) /XSS
246:                  XKAV2(2) = (XKEF1(2,IPBT)-XKEF1(2,IS-1)) /XSS
247:                  XKAV2(3) = (XKEF1(3,IPBT)-XKEF1(3,IS-1)) /XSS
248:              else
249:                  XSS = XSOC(IPBT)
250:                  XKAV2(1) = XKEF1(1,IPBT) /XSS
251:                  XKAV2(2) = XKEF1(2,IPBT) /XSS
252:                  XKAV2(3) = XKEF1(3,IPBT) /XSS
253:              end if
254: c

```

src/mvp/keff0.f

```

255: c ... source weighted "balance" estimators
256: c
257:       XKAV2(4)   = 0.0D0
258:       XKAV2(5)   = 0.0D0
259:       XKAV2(6)   = 0.0D0
260:       do 140 J = IS, IPBT
261:         XKAV2(4)   = XKAV2(4) + XKB(4,J)*XSOCB(J)
262:         XKAV2(5)   = XKAV2(5) + XKB(5,J)*XSOCB(J)
263:         XKAV2(6)   = XKAV2(6) + XKB(6,J)*XSOCB(J)
264:       140 continue
265:       XKAV2(4)   = XKAV2(4) / XSS
266:       XKAV2(5)   = XKAV2(5) / XSS
267:       XKAV2(6)   = XKAV2(6) / XSS
268: C
269: C ... COVXK : COVARIANCE MATRIX
270: C
271:       do 160 K = 1, 6
272:         do 150 L = 1, 6
273:           COVXK(L,K) = 0.0
274:         150 continue
275:       160 continue
276:
277:       XSS = XSOC(IPBT) - XSOC(IS-1)
278:       do 190 K = 1, 6
279:         do 180 L = 1, 6
280:           if ( L.ge.K ) then
281:             do 170 J = IS, IPBT
282:               X1 = XSOCB(J) / (XSS-XSOCB(J))
283:               COVXK(K,L) = COVXK(K,L) + X1*(XKB(K,J)-XKAV2(K))*
284:                 & (XKB(L,J)-XKAV2(L))
285:             170 continue
286:           else
287:             COVXK(K,L) = COVXK(L,K)
288:           end if
289:         180 continue
290:       190 continue
291: C
292:       XX = 1.D0/(NBATCH-IS+1)
293:       do 210 K = 1, 6
294:         do 200 L = 1, 6
295:           COVXK(K,L) = COVXK(K,L)*XX
296:         200 continue
297:       210 continue
298: C
299: C..... ESTIMATION OF KEFF BY THE PRINCIPLE OF MAXIMUM LIKELYHOOD
300: C       XKAV3 : KEFF
301: C       XKERR : STANDARD DEVIATION OF COMBINED ESTIMATION (%)
302: C       1 : TRACK LENGTH (PRODUCTION + BALANCE)
303: C       2 : COLLISION (PRODUCTION + BALANCE)
304: C       3 : ANALOG (PRODUCTION + BALANCE)
305: C       4 : PRODUCTION (TRK+COL+ANALOG)
306: C       5 : BALANCE (TRK+COL+ANALOG)
307: C       6 : ALL ESTIMATORS
308: C..... ESTIMATION OF KEFF BY EACH ESTIMATOR
309: C       XKEF : KEFF
310: C       XKSD : STANDARD DEVIATION (%)
311: C       1 : TRACK LENGTH (PRODUCTION)
312: C       2 : COLLISION (PRODUCTION)
313: C       3 : ANALOG (PRODUCTION)
314: C       4 : TRACK LENGTH (BALANCE)
315: C       5 : COLLISION (BALANCE)
316: C       6 : ANALOG (BALANCE)
317: C
318:       do 220 J = 1, 6
319:         XKER2(J) = SQRT(ABS(COVXK(J,J))) / XKAV2(J)*100.

```

```

320:       220 continue
321: C
322:       if ( JPR.ne.0 ) then
323:         write(IOW,7020) IS, IPBT
324:         write(IOW,7040) (XKAV2(J),XKER2(J),J=1,6)
325:       7020 format(1x,'+' /1x,'+ * K-EFFECTIVE AVERAGE ',
326:         & 'FROM BATCH ',i4,' TO ',i8,' *'
327:         & /1x,'+' ,
328:         & ' TRACK LENGTH(PROD.) ',
329:         & ' COLLISION(PROD.) ',
330:         & ' ANALOG(PROD.) ',
331:         & ' TRACK LENGTH(BAL.) ',
332:         & ' COLLISION(BAL.) ',
333:         & ' ANALOG(BAL.) ')
334:       7040 format(1x,'+' ,6(1P,E13.5,'(' ,0P,F5.3,'%)'))
335:       end if
336: C
337: C.... clear mixing factor of estimators (used in "real variance"
338: C calculation.
339: C
340:       do 240 J = 1, 6
341:         do 230 K = 1, 6
342:           FMIX(K,J) = 0.0D0
343:         230 continue
344:       240 continue
345: C
346: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 3 ESTIMATORS
347: C
348:       JCORR1 = 0
349:
350:       do 260 J = 4, 5
351:         K = 3*J - 11
352:         L = 3*J - 10
353:         M = 3*J - 9
354:
355:         V11V22 = COVXK(K,K)*COVXK(L,L)
356:         V22V33 = COVXK(L,L)*COVXK(M,M)
357:         V33V11 = COVXK(M,M)*COVXK(K,K)
358:         V12V12 = COVXK(K,L)**2
359:         V23V23 = COVXK(L,M)**2
360:         V31V31 = COVXK(M,K)**2
361:         V12V33 = COVXK(K,L)*COVXK(M,M)
362:         V23V11 = COVXK(L,M)*COVXK(K,K)
363:         V31V22 = COVXK(M,K)*COVXK(L,L)
364:         V12V23 = COVXK(K,L)*COVXK(L,M)
365:         V23V31 = COVXK(L,M)*COVXK(M,K)
366:         V31V12 = COVXK(M,K)*COVXK(K,L)
367:         A = V11V22*COVXK(M,M) + 2.*V12V23*COVXK(K,M)
368:         & - V23V23*COVXK(K,K) - V31V31*COVXK(L,L)
369:         & - V12V12*COVXK(M,M)
370:         T11 = V22V33 - V23V23
371:         T12 = V23V31 - V12V33
372:         T13 = V12V23 - V31V22
373:         T22 = V33V11 - V31V31
374:         T23 = V31V12 - V23V11
375:         T33 = V11V22 - V12V12
376:         U1 = T11 + T12 + T13
377:         U2 = T12 + T22 + T23
378:         U3 = T13 + T23 + T33
379:         USUM = U1 + U2 + U3
380:
381:       if ( USUM.le.0.0D0 ) then
382:
383:         if ( JCORR1.eq.0 ) then
384:           JCORR1 = 1

```

src/mvp/keff0.f

```

385: C      write(IOT,(''1''/lx,a,a,a))
386: C      &      '!!! Some M.L.E. triplets cannot be evaluated ',
387: C      &      'correctly because of strong correlation or ',
388: C      &      'some other reason.'
389: C      write(IOT,('(/a/))')
390: C      &      ' Dummy values -1.0 will be output.'
391: C      end if
392: C      write(IOT,('lx,a,i4,a,a/4x,a,3E14.5'))
393: C      &      'Cumulative after batch ', I, ' type : ',
394: C      &      ESTCMB, 'weights : ', U1, U2, U3
395: C
396: C      XKAV3(J) = -1.0D0
397: C      XKER3(J) = 0.0
398: C      else
399: C      USUM = 1.0D0/USUM
400: C      XKAV3(J) = (U1*XKAV2(K)+U2*XKAV2(L)+U3*XKAV2(M))*USUM
401: C      XKER3(J) = USUM*A
402: C      XKER3(J) = SQRT(ABS(XKER3(J))) /XKAV3(J)*100.
403: C
404: C      if ( I.eq.IS ) then
405: C      FMIX(K,J) = U1*USUM
406: C      FMIX(L,J) = U2*USUM
407: C      FMIX(M,J) = U3*USUM
408: C      end if
409: C      end if
410: C      250 continue
411: C      260 continue
412: C
413: C      C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 2 ESTIMATORS
414: C
415: C      JCORR1 = 0
416: C      do 280 J = 1, 3
417: C      K = J
418: C      L = J + 3
419: C
420: C      CORR1 = COVXK(K,L) /SQRT(COVXK(K,K)*COVXK(L,L))
421: C
422: C      if ( ABS(CORR1-1.0D0).lt.DECORR ) then
423: C      if ( JCORR1.eq.0 ) then
424: C      JCORR1 = 1
425: C      write(IOT,(''1''/lx,a,a/lx,a,E12.5))
426: C      &      '!!! Too strongly correlated estimator pairs ',
427: C      &      ', are found in M.L.E. calculation.',
428: C      &      ' (criterion : abs(correlation coeff.-1) < ',
429: C      &      DECORR
430: C      write(IOT,('(/a/))')
431: C      &      ' Dummy values of -1.0 will be output.'
432: C      end if
433: C
434: C      write(IOT,('lx,a,i4,a,a,a,E14.7'))
435: C      &      ' Cumulative after batch ', I, ' type : ',
436: C      &      ESTCMB, ' : correlation =', CORR1
437: C      XKAV3(J) = -1.0D0
438: C      XKER3(J) = 0.0D0
439: C      else
440: C      X1 = COVXK(K,K) - COVXK(K,L)
441: C      X2 = COVXK(L,L) - COVXK(K,L)
442: C      XKAV3(J) = (X1*XKAV2(L)+X2*XKAV2(K)) / (X1+X2)
443: C      XKER3(J) =
444: C      &      (COVXK(K,K)*COVXK(L,L)-COVXK(K,L)*COVXK(K,L)) /
445: C      &      (X1+X2)
446: C      XKER3(J) = SQRT(ABS(XKER3(J))) /XKAV3(J)*100.0
447: C
448: C      FMIX(L,J) = X1/(X1+X2)
449: C      FMIX(K,J) = X2/(X1+X2)
450: C
451: C      end if
452: C      270 continue
453: C      280 continue
454: C
455: C      C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF ALL 6 ESTIMATORS
456: C      GET AN INVERSE MATRIX OF COVXK INTO COVXK
457: C
458: C      XKER3(6) = 0.0
459: C      XKAV3(6) = -1.0
460: C
461: C      ... omit one estimator from too strongly correlated pair ...
462: C      ( non-diagonal elements of COVXK for omitted estimator is
463: C      cleared to 0.0 )
464: C
465: C      do 320 L = 6, 1, -1
466: C      JJCC(L) = 0
467: C      do 300 K = 1, L - 1
468: C
469: C      CORR1 = COVXK(K,L) /SQRT(COVXK(K,K)*COVXK(L,L))
470: C
471: C      if ( ABS(CORR1-1.0D0).lt.DECORR ) then
472: C      JJCC(L) = 1
473: C      COVXK(L,L) = 1.0D0
474: C      do 290 KK = 1, 6
475: C      if ( KK.ne.L ) then
476: C      COVXK(KK,L) = 0.0D0
477: C      COVXK(L,KK) = 0.0D0
478: C      end if
479: C      290 continue
480: C      go to 310
481: C      end if
482: C      300 continue
483: C
484: C      310 continue
485: C      320 continue
486: C
487: C      do 330 L = 1, 6
488: C      JJCC0(L) = JJCC(L)
489: C      330 continue
490: C
491: C      ... invert covariance matrix ...
492: C
493: C      call MTXINV( COVXK(1,1), 6, IWK1, CORR(1,1), 1.0D-16, ICON )
494: C
495: C      if ( ICON.ne.0 ) then
496: C      write(IOT,('lx,a,a'))
497: C      &      '!!! CANNOT INVERT COVARIANCE MATRIX ',
498: C      &      ' --> MAX.LIKELYHOOD ESTIMATOR PRINTED AS -1'
499: C
500: C
501: C      XKER3(6) = 0.0
502: C      XKAV3(6) = -1.0
503: C      else
504: C      XKER3(6) = 0.0
505: C      XKAV3(6) = 0.0
506: C      end if
507: C
508: C      if ( XKAV3(6).ne.-1.0 ) then
509: C      do 350 K = 1, 6
510: C      do 340 L = 1, 6
511: C      if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
512: C      XKER3(6) = XKER3(6) + COVXK(K,L)
513: C      XKAV3(6) = XKAV3(6) + COVXK(K,L)*XKAV2(L)
514: C      end if

```

src/mvp/keff0.f

```
515: C
516: C   ... weight for "ALL" estimator (NSKIP+1 'th batch)
517: C
518:   340      continue
519:   350      continue
520:
521:       XKER3(6)   = 1./XKER3(6)
522:       XKAV3(6)   = XKAV3(6)*XKER3(6)
523:       XKER3(6)   = SQRT(ABS(XKER3(6))) /XKAV3(6)*100.
524:   end if
525:
526: C
527:   if ( JPR.ne.0 ) then
528:     write(IOW,7060) IS, IPBT
529:     write(IOT,7040) (XKAV3(J),XKER3(J),J=1,6)
530: C
531:   7060      format(1x,'+
532:   &          /1x,'+ * M.L.E. K-EFFECTIVE BY EACH ESTIMATOR ',
533:   &          'FROM BATCH ',i4,' TO ',i8,' */1x,'+',
534:   &          ' TRACK LENGTH(MLE) ',
535:   &          ' COLLISION(MLE) ',
536:   &          ' ANALOG(MLE) ',
537:   &          ' PRODUCTION(MLE) ',
538:   &          ' BALANCE(MLE) ',
539:   &          ' ALL(MLE) ')
540:   end if
541: C
542:   end if
543: C
544:   if ( JPR.ne.0 ) write(IOT,7001)
545: C
546:   return
547: end
```


src/mvp/keff.f

```

1:      subroutine KEFF( TITLE, JPRTS, NBATCH,NSKIP, NMXLG,NEITER,WLEK,
2:      &                WCNTR, XSOC,  XKEF,  XSOCB, XKB,  COVXK, XKMLE,
3:      &                XKERR, CORR, IWK1,  XKSD,  XKBB,  CA,    CR,
4:      &                JPHNU ) ! photo-nuc
5: C=====
6: C  PURPOSE: Statistical analysis of k-effective.
7: C      Write the results on file (I/O unit 30)
8: C  CALLED IN: TALLYO
9: C  CALL      : MTXINV.
10: C=====
11:
12:      implicit real*8(A-H,O-Z)
13:      parameter(MAXXK = 10) ! photo-nuc
14:
15:      include '../shared/INC/_IOUNIT'
16:      include 'INC/_IOUNIT2'
17:
18:      character*72 TITLE(2)
19:      integer JPRTS(*)
20:      real*8 WLEK, WCNTR(*)
21:      real*8 XSOC(NBATCH)
22:      real*8 XKEF(MAXXK,NBATCH)
23: C
24:      real*8 XSOCB(NBATCH)
25:      real*8 XKB(MAXXK,NBATCH)
26:      real*8 COVXK(MAXXK,MAXXK,NBATCH)
27:      real*8 XKMLE(MAXXK,NBATCH)
28:      real*8 XKERR(MAXXK,NBATCH)
29:      real*8 CORR(MAXXK,MAXXK,*)
30:      integer IWK1(MAXXK)
31:      real*8 COVVK(6,6)
32: C
33:      real*8 XKSD(MAXXK,NBATCH)
34:      real*8 XKBB(NBATCH), CA(NMXLG), CR(NMXLG)
35: C
36:      real*8 X1, X2, X3, X4, X5, X6, S1
37:      character*32 TAG
38: C
39: C ... criterion to reject too mach correlated combination of
40: C   estimation in maximum likelihood estimation.
41: C
42:      parameter( DECORR  = 1.0D-5 )
43: C
44: C ... local data
45: C
46:      character*48 ESTCMB
47: C
48:      integer JJCC(6)
49:      integer JJCC0(6)
50:      real*8 WTALL(6)
51: C
52:      real*8 AVG(16), SIGA(16), SIGR(16)
53:      character*48 ESTKEF(16)
54: C
55: C ... mixing factor of M.L.E.'s ....
56: C
57: C      XKMLE(J) = Sum[I] ( XKB(I)*FMIX(I,J) )
58: C
59:      real*8 FMIX(MAXXK,MAXXK)
60: C
61: C-----
62: C
63: C
64: C-----
65: C**** STATISTICAL ANALYSIS OF K-EFFECTIVE

```

```

66: C-----
67: C
68:      call HEADER( IOT, '      RESULT OF EIGENVALUE CALCULATION      ' )
69:
70:      write(IOT,7000)
71:      7000 format(/IX,' << EIGEN-VALUE ( K-EFFECTIVE ) >>')
72: C
73: C.... SET WCNTR(7) = TOTAL LEAKAGE IN THIS RUN
74: C
75:      WCNTR(7)  = WLEK
76: C
77:      XSOCB(1)  = XSOC(1)
78:      XKEF(4,1) = XKEF(4,1) + XKEF(7,1)
79:      XKEF(5,1) = XKEF(5,1) + XKEF(7,1)
80:      XKEF(6,1) = XKEF(6,1) + XKEF(7,1)
81:      XKB(1,1)  = XKEF(1,1) /XSOC(1)
82:      XKB(2,1)  = XKEF(2,1) /XSOC(1)
83:      XKB(3,1)  = XKEF(3,1) /XSOC(1)
84:      XKB(4,1)  = XKEF(1,1) /XKEF(4,1)
85:      XKB(5,1)  = XKEF(2,1) /XKEF(5,1)
86:      XKB(6,1)  = XKEF(3,1) /XKEF(6,1)
87:      if(JPHNU.ne.0) then ! photo-nuc
88:          XKB(8,1) = XKEF(8,1) /XSOC(1) ! photo-nuc
89:          XKB(9,1) = XKEF(9,1) /XSOC(1) ! photo-nuc
90:          XKB(10,1) = XKEF(10,1)/XSOC(1) ! photo-nuc
91:      end if ! photo-nuc
92:      XKMLE(1,1) = XKB(1,1)
93:      XKMLE(2,1) = XKB(2,1)
94:      XKMLE(3,1) = XKB(3,1)
95:      XKMLE(4,1) = XKB(4,1)
96:      XKMLE(5,1) = XKB(5,1)
97:      XKMLE(6,1) = XKB(6,1)
98:      if(JPHNU.ne.0) then ! photo-nuc
99:          XKMLE(8,1) = XKB(8,1) ! photo-nuc
100:          XKMLE(9,1) = XKB(9,1) ! photo-nuc
101:          XKMLE(10,1) = XKB(10,1) ! photo-nuc
102:      end if ! photo-nuc
103: CC
104: CC XSOCB : SOURCE (EACH BATCH)
105: CC XKB : ESTIMATOR OF KEFF (EACH BATCH)
106: CC XKMLE : CUMULATIVE ESTIMATOR OF KEFF TO THIS BATCH
107: CC 1: TRACK LENGTH ESTIMATOR (PRODUCTION RATE)
108: CC 2: COLLISION ESTIMATOR (PRODUCTION RATE)
109: CC 3: ANALOG ESTIMATOR (PRODUCTION RATE)
110: CC 4: TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
111: CC 5: COLLISION ESTIMATOR (NEUTRON BALANCE)
112: CC 6: ANALOG ESTIMATOR (NEUTRON BALANCE)
113: CC
114:      do 110 I = 2, NBATCH
115:          XKEF(4,I) = XKEF(4,I) + XKEF(7,I)
116:          XKEF(5,I) = XKEF(5,I) + XKEF(7,I)
117:          XKEF(6,I) = XKEF(6,I) + XKEF(7,I)
118:          XKMLE(1,I) = XKEF(1,I) /XSOC(I)
119:          XKMLE(2,I) = XKEF(2,I) /XSOC(I)
120:          XKMLE(3,I) = XKEF(3,I) /XSOC(I)
121: C980809 XKMLE(4,I) = XKEF(1,I) /XKEF(4,I)
122: C980809 XKMLE(5,I) = XKEF(2,I) /XKEF(5,I)
123: C980809 XKMLE(6,I) = XKEF(3,I) /XKEF(6,I)
124:          XSOCB(I) = XSOC(I) - XSOC(I-1)
125:          XKB(1,I) = (XKEF(1,I)-XKEF(1,I-1)) /XSOCB(I)
126:          XKB(2,I) = (XKEF(2,I)-XKEF(2,I-1)) /XSOCB(I)
127:          XKB(3,I) = (XKEF(3,I)-XKEF(3,I-1)) /XSOCB(I)
128:          XKB(4,I) = (XKEF(1,I)-XKEF(1,I-1)) /((XKEF(4,I)-XKEF(4,I-1))
129:          XKB(5,I) = (XKEF(2,I)-XKEF(2,I-1)) /((XKEF(5,I)-XKEF(5,I-1))
130:          XKB(6,I) = (XKEF(3,I)-XKEF(3,I-1)) /((XKEF(6,I)-XKEF(6,I-1))

```

src/mvp/keff.f

```

131:         if(JPHNU.ne.0) then                                ! photo-nuc
132:             XKMLE(8,I) = XKEF(8,I) /XSOC(I)                ! photo-nuc
133:             XKMLE(9,I) = XKEF(9,I) /XSOC(I)                ! photo-nuc
134:             XKMLE(10,I) = XKEF(10,I)/XSOC(I)                ! photo-nuc
135:             XKB(8,I) = (XKEF(8,I)-XKEF(8,I-1)) /XSOCB(I)    ! photo-nuc
136:             XKB(9,I) = (XKEF(9,I)-XKEF(9,I-1)) /XSOCB(I)    ! photo-nuc
137:             XKB(10,I) = (XKEF(10,I)-XKEF(10,I-1)) /XSOCB(I) ! photo-nuc
138:         end if                                              ! photo-nuc
139: C980809
140: c ... new: source weighted "balance" estimators
141: c
142:     XKMLE(4,I) = 0.0D0
143:     XKMLE(5,I) = 0.0D0
144:     XKMLE(6,I) = 0.0D0
145:     do 100 J = 1, I
146:         XKMLE(4,I) = XKMLE(4,I) + XKB(4,J)*XSOCB(J)
147:         XKMLE(5,I) = XKMLE(5,I) + XKB(5,J)*XSOCB(J)
148:         XKMLE(6,I) = XKMLE(6,I) + XKB(6,J)*XSOCB(J)
149:     100 continue
150:     XKMLE(4,I) = XKMLE(4,I) /XSOC(I)
151:     XKMLE(5,I) = XKMLE(5,I) /XSOC(I)
152:     XKMLE(6,I) = XKMLE(6,I) /XSOC(I)
153: 110 continue
154:
155:     if(JPHNU.ne.0) then
156:         write(IOT,7021) '=== K-EFFECTIVE ( EACH BATCH ) ===' ! photo-nuc
157:     else
158:         write(IOT,7020) '=== K-EFFECTIVE ( EACH BATCH ) ==='
159:     end if
160: 7021 format(/1X,18X,A/
161: &/1X,'          PRODUCTION ',
162: &'          NEUTRON BALANCE ',
163: &'          PHOTONEUTRON' /
164: &/1X,' BATCH TRACK L. COLLISION ANALOG ',
165: &'          TRACK L. COLLISION ANALOG ',
166: &'          TRACK L. COLLISION'
167: &/1X,103(' '))
168: 7020 format(/1X,18X,A/1X,'          PRODUCTION ',
169: &'          NEUTRON BALANCE '//1X,
170: &'          BATCH TRACK L. COLLISION ANALOG ',
171: &'          TRACK L. COLLISION ANALOG' /1X,
172: &'          '
173: &'          '
174:
175:     if(JPHNU.ne.0) then
176:         write(IOT,7041) (I,(XKB(J,I),J=1,6),(XKB(J,I),J=8,9), ! photo-nuc
177: & I=1,NBATCH) ! photo-nuc
178:     else
179:         write(IOT,7040) (I,(XKB(J,I),J=1,6),I=1,NBATCH)
180:     end if
181: 7041 format(1X,I7,2X,1P,8E12.5)
182: 7040 format(1X,I7,2X,1P,6E12.5)
183:
184:     if(JPHNU.ne.0) then
185:         write(IOT,7021) ! photo-nu
186:         & '=== K-EFFECTIVE ( CUMULATIVE TO THIS BATCH ) ===' ! photo-nu
187:         write(IOT,7041) (I,(XKMLE(J,I),J=1,6),(XKMLE(J,I),J=8,9), ! photo-nu
188: & I=1,NBATCH) ! photo-nu
189:     else
190:         write(IOT,7020)
191:         & '=== K-EFFECTIVE ( CUMULATIVE TO THIS BATCH ) ==='
192:         write(IOT,7040) (I,(XKMLE(J,I),J=1,6),I=1,NBATCH)
193:     end if
194:
195: C-----
196: C .... OUTPUT TO BINARY FILE (IV) EIGEN VALUES
197: C-----
198: C
199: C -----0---*---1---*---2---*---3---
200: TAG = 'K-EFF: BATCH & CUMULATIVE '
201: C
202: C
203: write(IORS) TAG, ((XKB(J,I),I=1,NBATCH),J=1,6),
204: & ((XKMLE(J,I),I=1,NBATCH),J=1,6)
205: C
206: X1 = XKEF(1,NBATCH)
207: X2 = XKEF(2,NBATCH)
208: X3 = XKEF(3,NBATCH)
209: C980809
210: c X4 = XKEF(4,NBATCH)
211: c X5 = XKEF(5,NBATCH)
212: c X6 = XKEF(6,NBATCH)
213: if(JPHNU.ne.0) then ! photo-nuc
214:     X8 = XKEF(8,NBATCH) ! photo-nuc
215:     X9 = XKEF(9,NBATCH) ! photo-nuc
216:     X10 = XKEF(10,NBATCH) ! photo-nuc
217: end if ! photo-nuc
218: CCC
219: CCC XSOC(I) : SOURCE. CUMULATIVE TO I'TH BATCH
220: CCC ----> SUM OF I'TH BATCH TO THE LAST BATCH.
221: CCC
222: CCC SIMILAR TO XKEF(6,I) KEFF (PRODUCTION & LOSS)
223: CCC
224: do 120 J = NBATCH,2,-1
225:     XSS = XSOC(NBATCH) - XSOC(J-1)
226:     XKEF(1,J) = (X1-XKEF(1,J-1)) /XSS
227:     XKEF(2,J) = (X2-XKEF(2,J-1)) /XSS
228:     XKEF(3,J) = (X3-XKEF(3,J-1)) /XSS
229: C980809
230: c XKEF(4,J) = (X1-XKEF(1,J-1)) /(X4-XKEF(4,J-1))
231: c XKEF(5,J) = (X2-XKEF(2,J-1)) /(X5-XKEF(5,J-1))
232: c XKEF(6,J) = (X3-XKEF(3,J-1)) /(X6-XKEF(6,J-1))
233: if(JPHNU.ne.0) then ! photo-nuc
234:     XKEF(8,J) = (X8-XKEF(8,J-1)) /XSS ! photo-nuc
235:     XKEF(9,J) = (X9-XKEF(9,J-1)) /XSS ! photo-nuc
236:     XKEF(10,J) = (X10-XKEF(10,J-1)) /XSS ! photo-nuc
237: end if ! photo-nuc
238: 120 continue
239:
240: XKEF(1,1) = X1/XSOC(NBATCH)
241: XKEF(2,1) = X2/XSOC(NBATCH)
242: XKEF(3,1) = X3/XSOC(NBATCH)
243: C980809
244: c XKEF(4,1) = X1/X4
245: c XKEF(5,1) = X2/X5
246: c XKEF(6,1) = X3/X6
247: C980809
248: if(JPHNU.ne.0) then ! photo-nuc
249:     XKEF(8,1) = X8/XSOC(NBATCH) ! photo-nuc
250:     XKEF(9,1) = X9/XSOC(NBATCH) ! photo-nuc
251:     XKEF(10,1) = X10/XSOC(NBATCH) ! photo-nuc
252: end if ! photo-nuc
253:
254: c ... new: source weighted "balance" estimators
255: c
256: do 140 I = 1, NBATCH

```

src/mvp/keff.f

```

257:      XKEF(4,I) = 0.0D0
258:      XKEF(5,I) = 0.0D0
259:      XKEF(6,I) = 0.0D0
260:      do 130 J = I, NBATCH
261:          XKEF(4,I) = XKEF(4,I) + XKB(4,J)*XSOCB(J)
262:          XKEF(5,I) = XKEF(5,I) + XKB(5,J)*XSOCB(J)
263:          XKEF(6,I) = XKEF(6,I) + XKB(6,J)*XSOCB(J)
264: 130      continue
265:      if (I.eq.1) then
266:          XSS = XSOC(NBATCH)
267:      else
268:          XSS = XSOC(NBATCH) - XSOC(I-1)
269:      end if
270:      XKEF(4,I) = XKEF(4,I) /XSS
271:      XKEF(5,I) = XKEF(5,I) /XSS
272:      XKEF(6,I) = XKEF(6,I) /XSS
273: 140      continue
274:
275: C..... COVXK : COVARIANCE MATRIX
276:
277:      do K = 1, MAXXK
278:          do L = 1, MAXXK
279:              do I = 1, NBATCH
280:                  COVXK(K,L,I) = 0.d0
281:              end do
282:          end do
283:      end do
284:
285:      do 190 I = 1, NBATCH - 1
286:          if (I.eq.1) then
287:              XSS = XSOC(NBATCH)
288:          else
289:              XSS = XSOC(NBATCH) - XSOC(I-1)
290:          end if
291:          do 180 K = 1, 6
292:              do 170 L = 1, 6
293:                  if ( L.ge.K ) then
294:
295:                      do 160 J = I, NBATCH
296:                          X1 = XSOCB(J) / (XSS-XSOCB(J))
297:                          COVXK(K,L,I) = COVXK(K,L,I)
298:                          + X1*(XKB(K,J)-XKEF(K,I))*
299:                          (XKB(L,J)-XKEF(L,I))
300: 160                      continue
301:
302:                      else
303:                          COVXK(K,L,I) = COVXK(L,K,I)
304:                      end if
305: 170                  continue
306: 180              continue
307:                  !+photo-nuc
308:                  if(JPHNU.ne.0) then
309:                      do K = 8, 10
310:                          do L = 8, 10
311:                              if ( L.ge.K ) then
312:                                  do J = 1, NBATCH
313:                                      X1 = XSOCB(J) / (XSS - XSOCB(J))
314:                                      COVXK(K,L,I) = COVXK(K,L,I) +
315:                                      & X1*(XKB(K,J)-XKEF(K,I))*(XKB(L,J)-XKEF(L,I))
316:                                  end do
317:                              else
318:                                  COVXK(K,L,I) = COVXK(L,K,I)
319:                              end if
320:                          end do
321:                  end do

```

```

322:      end if
323:      !-photo-nuc
324: 190      continue
325:
326:      do 220 I = 1, NBATCH - 1
327:          XX = 1.D0/(NBATCH-I+1)
328:          do 210 K = 1, 6
329:              do 200 L = 1, 6
330:                  COVXK(K,L,I) = COVXK(K,L,I)*XX
331: 200              continue
332: 210          continue
333:
334:          if(JPHNU.ne.0) then
335:              do K = 8, 10
336:                  do L = 8, 10
337:                      COVXK(K,L,I) = COVXK(K,L,I)*XX
338:                  end do
339:              end do
340:          end if
341: 220      continue
342: C
343: C..... KEEP CORRELATION MATRIX FOR ESTIMATION AFTER (NSKIP+1)TH BATCH
344: C
345:      NSKIPl = NSKIP + 1
346:
347:      do 240 K = 1, 6
348:          do 230 L = 1, 6
349:              CORR(K,L,1) = COVXK(K,L,NSKIPl) /
350:              & Sqrt(COVXK(K,K,NSKIPl)*COVXK(L,L,NSKIPl))
351: 230          continue
352: 240      continue
353:
354:      if(JPHNU.ne.0) then
355:          do K = 8, 10
356:              do L = 8, 10
357:                  CORR(K,L,1) = COVXK(K,L,NSKIPl) /
358:                  & sqrt(COVXK(K,K,NSKIPl)*COVXK(L,L,NSKIPl))
359:              end do
360:          end do
361:      end if
362: C
363: C..... ESTIMATION OF KEFF BY THE PRINCIPLE OF MAXIMUM LIKELYHOOD
364: C      XKMLE : KEFF
365: C      XKERR : STANDARD DEVIATION OF COMBINED ESTIMATION (%)
366: C      1 : TRACK LENGTH (PRODUCTION + BALANCE)
367: C      2 : COLLISION (PRODUCTION + BALANCE)
368: C      3 : ANALOG (PRODUCTION + BALANCE)
369: C      4 : PRODUCTION (TRK+COL+ANALOG)
370: C      5 : BALANCE (TRK+COL+ANALOG)
371: C      6 : ALL ESTIMATORS
372: C..... ESTIMATION OF KEFF BY EACH ESTIMATOR
373: C      XKEF : KEFF
374: C      XKSD : STANDARD DEVIATION (%)
375: C      1 : TRACK LENGTH (PRODUCTION)
376: C      2 : COLLISION (PRODUCTION)
377: C      3 : ANALOG (PRODUCTION)
378: C      4 : TRACK LENGTH (BALANCE)
379: C      5 : COLLISION (BALANCE)
380: C      6 : ANALOG (BALANCE)
381: C      8 : TRACK LENGTH (PHOTONEUTRON) ! photo-nuc
382: C      9 : COLLISION (PHOTONEUTRON) ! photo-nuc
383: C      10 : ANALOG (PHOTONEUTRON) ! photo-nuc
384: C
385:      do 260 J = 1, 6
386:          do 250 I = 1, NBATCH - 1

```

src/mvp/keff.f

```

387:      XKSD(J,I) = SQRT(ABS(COVXK(J,J,I))) / XKEF(J,I)*100.
388:      250 continue
389:      260 continue
390:
391:      if(JPHNU.ne.0) then
392:          do J = 8, 10
393:              do I = 1, NBATCH - 1
394:                  XKSD(J,I) = sqrt(abs(COVXK(J,J,I))) / XKEF(J,I) * 100.d0
395:              end do
396:          end do
397:      end if
398: C
399: C
400:      write(IOT,7060)
401: 7060 format('1',' << K-EFFECTIVE BY EACH ESTIMATOR ',
402: & ' ( CUMULATIVE AFTER THIS BATCH ) >>'/1X,
403: & ' === PRODUCTION ===',
404: & ' === NEUTRON BALANCE ===',
405: & //1X,' BATCH TRACK LENGTH EST. COLLISION EST. ANALOG EST.',
406: & ' TRACK LENGTH EST. COLLISION EST. ANALOG EST. '/1X,
407: & '-----',
408: & '-----',
409: & '-----',
410: & '-----',
411: & '-----',
412: & '-----')
413:
414:      do 270 I = 1, NBATCH - 1
415:          write(IOT,7080) I, (XKEF(J,I),XKSD(J,I),J=1,6)
416:      270 continue
417:
418: 7080 format(1X,I7,6(1P,E13.5,'(','0P,F5.3,'%'))
419:
420:      !+photo-nuc
421:      if(JPHNU.ne.0) then
422:          write(IOT,7061)
423:          do I = 1, NBATCH - 1
424:              write(IOT,7081) I, (XKEF(J,I),XKSD(J,I),J=8,10)
425:          end do
426:      end if
427: 7061 format(
428: & /28x,'=== PHOTONEUTRON ===' /
429: & /' BATCH TRACK LENGTH EST. COLLISION EST. ANALOG EST.'
430: & /' -----'
431: & /' -----')
432: 7081 format(1X,I7,3(1P,E13.5,'(','0P,F6.3,'%'))
433:      !-photo-nuc
434: C
435: C.... clear mixing factor of estimators (used in "real variance"
436: C calculation.
437: C
438:      do 290 J = 1, 6
439:          do 280 K = 1, 6
440:              FMIX(K,J) = 0.0D0
441:          280 continue
442:      290 continue
443: C
444: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 3 ESTIMATORS

```

```

445: C
446: JCORR1 = 0
447:
448: do 310 J = 4, 5
449:     K = 3*J - 11
450:     L = 3*J - 10
451:     M = 3*J - 9
452:
453:     if ( J.eq.4 ) then
454:         ESTCMB = 'production (track length/collision/analog)'
455:     else if ( J.eq.5 ) then
456:         ESTCMB = 'balance (track length/collision/analog)'
457:     end if
458:
459:     do 300 I = 1, NBATCH - 2
460:         V11V22 = COVXK(K,K,I)*COVXK(L,L,I)
461:         V22V33 = COVXK(L,L,I)*COVXK(M,M,I)
462:         V33V11 = COVXK(M,M,I)*COVXK(K,K,I)
463:         V12V12 = COVXK(K,L,I)**2
464:         V23V23 = COVXK(L,M,I)**2
465:         V31V31 = COVXK(M,K,I)**2
466:         V12V33 = COVXK(K,L,I)*COVXK(M,M,I)
467:         V23V11 = COVXK(L,M,I)*COVXK(K,K,I)
468:         V31V22 = COVXK(M,K,I)*COVXK(L,L,I)
469:         V12V23 = COVXK(K,L,I)*COVXK(L,M,I)
470:         V23V31 = COVXK(L,M,I)*COVXK(M,K,I)
471:         V31V12 = COVXK(M,K,I)*COVXK(K,L,I)
472:         A = V11V22*COVXK(M,M,I) + 2.*V12V23*COVXK(K,M,I)
473:         & - V23V23*COVXK(K,K,I) - V31V31*COVXK(L,L,I)
474:         & - V12V12*COVXK(M,M,I)
475:         T11 = V22V33 - V23V23
476:         T12 = V23V31 - V12V33
477:         T13 = V12V23 - V31V22
478:         T22 = V33V11 - V31V31
479:         T23 = V31V12 - V23V11
480:         T33 = V11V22 - V12V12
481:         U1 = T11 + T12 + T13
482:         U2 = T12 + T22 + T23
483:         U3 = T13 + T23 + T33
484:         USUM = U1 + U2 + U3
485:
486:         if ( USUM.le.0.0D0 ) then
487:
488:             if ( JCORR1.eq.0 ) then
489:                 JCORR1 = 1
490:                 write(IOT,('(1'/1X,a,a,a)')
491:                 & '!!! Some M.L.E. triplets cannot be evaluated ',
492:                 & 'correctly because of strong correlation or ',
493:                 & 'some other reason.'
494:                 write(IOT,('(a/))')
495:                 & ' Dummy values -1.0 will be output.'
496:             end if
497:             write(IOT,('(1X,a,i4,a,a/4X,a,3E14.5)')
498:             & 'Cumulative after batch ', I, ' type : ',
499:             & ESTCMB, 'weights : ', U1, U2, U3
500:
501:             XKMLE(J,I) = -1.0D0
502:             XKERR(J,I) = 0.0
503:         else
504:             USUM = 1.0D0/USUM
505:             XKMLE(J,I) = (U1*XKEF(K,I)+U2*XKEF(L,I)+U3*XKEF(M,I))*
506:             & USUM
507:             XKERR(J,I) = USUM*A
508:             XKERR(J,I) = SQRT(ABS(XKERR(J,I))) / XKMLE(J,I)*100.
509:

```

src/mvp/keff.f

```

510:         if ( I.eq.NSKIP1 ) then
511:             FMIX(K,J) = U1*USUM
512:             FMIX(L,J) = U2*USUM
513:             FMIX(M,J) = U3*USUM
514:         end if
515:     end if
516: 300 continue
517: 310 continue
518: !+photo-nuc
519: if(JPHNU.ne.0) then
520:     JCORR1 = 0
521:     J = 8
522:     K = 8
523:     L = 9
524:     M = 10
525:     do I = 1, NBATCH - 2
526:         V11V22 = COVXK(K,K,I)*COVXK(L,L,I)
527:         V22V33 = COVXK(L,L,I)*COVXK(M,M,I)
528:         V33V11 = COVXK(M,M,I)*COVXK(K,K,I)
529:         V12V12 = COVXK(K,L,I)**2
530:         V23V23 = COVXK(L,M,I)**2
531:         V31V31 = COVXK(M,K,I)**2
532:         V12V33 = COVXK(K,L,I)*COVXK(M,M,I)
533:         V23V11 = COVXK(L,M,I)*COVXK(K,K,I)
534:         V31V22 = COVXK(M,K,I)*COVXK(L,L,I)
535:         V12V23 = COVXK(K,L,I)*COVXK(L,M,I)
536:         V23V31 = COVXK(L,M,I)*COVXK(M,K,I)
537:         V31V12 = COVXK(M,K,I)*COVXK(K,L,I)
538:         A = V11V22*COVXK(M,M,I) + 2*V12V23*COVXK(K,M,I) -
539:             & V23V23*COVXK(K,K,I) - V31V31*COVXK(L,L,I) -
540:             & V12V12*COVXK(M,M,I)
541:         T11 = V22V33 - V23V23
542:         T12 = V23V31 - V12V33
543:         T13 = V12V23 - V31V22
544:         T22 = V33V11 - V31V31
545:         T23 = V31V12 - V23V11
546:         T33 = V11V22 - V12V12
547:         U1 = T11 + T12 + T13
548:         U2 = T12 + T22 + T23
549:         U3 = T13 + T23 + T33
550:         USUM = U1 + U2 + U3
551:         if ( USUM.le.0.d0 ) then
552:             if ( JCORR1.eq.0 ) then
553:                 JCORR1 = 1
554:                 write(IOT,911)
555:             end if
556:             write(IOT,912) I, U1, U2, U3
557:             XKMLE(J,I) = -1.d0
558:             XKERR(J,I) = 0.d0
559:         else
560:             USUM = 1.d0 / USUM
561:             XKMLE(J,I) = (U1*XKEF(K,I)+U2*XKEF(L,I)+U3*XKEF(M,I))*
562:                 & USUM
563:             XKERR(J,I) = USUM*A
564:             XKERR(J,I) = sqrt(abs(XKERR(J,I))) / XKMLE(J,I) * 100.d0
565:         end if
566:     end do
567: end if
568: 911 format(/' !!! Some M.L.E. triplets cannot be evaluated correctly',
569: & ' because of strong correlation or some other reason.'
570: & /' Dummy values -1.0 will be output.')
571: 912 format(/' Cumulative after batch ',i4,' type = photoneutron ('
572: & 'track-length/collision/analog); weight=',lp,3e13.5)
573: !-photo-nuc
574: C

```

```

575: C
576: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 2 ESTIMATORS
577: C
578: C
579:     JCORR1 = 0
580:     do 330 J = 1, 3
581:         K = J
582:         L = J + 3
583:
584:         if ( J.eq.1 ) then
585:             ESTCMB = 'track length (production & balance)'
586:         else if ( J.eq.2 ) then
587:             ESTCMB = 'collision (production & balance)'
588:         else if ( J.eq.3 ) then
589:             ESTCMB = 'analog (production & balance)'
590:         end if
591: C
592:     do 320 I = 1, NBATCH - 2
593:         CORR1 = COVXK(K,L,I) /SQRT(COVXK(K,K,I)*COVXK(L,L,I))
594:
595:         if ( ABS(CORR1-1.0D0).lt.DECORR ) then
596:             if ( JCORR1.eq.0 ) then
597:                 JCORR1 = 1
598:                 write(IOT,('(1'/lx,a,a/lx,a,E12.5/))
599:                 & '!!! Too strongly correlated estimator pairs '
600:                 & ', 'are found in M.L.E. calculation.',
601:                 & ' (criterion : abs(correlation coeff.-1) < ',
602:                 & DECORR
603:                 write(IOT,('(a/))')
604:                 & ' Dummy values of -1.0 will be output.'
605:             end if
606:
607:             write(IOT,('(lx,a,i4,a,a,a,E14.7/))
608:             & ' Cumulative after batch ', I, ' type : ',
609:             & ESTCMB, ' : correlation =', CORR1
610:             XKMLE(J,I) = -1.0D0
611:             XKERR(J,I) = 0.0D0
612:         else
613:             X1 = COVXK(K,K,I) - COVXK(K,L,I)
614:             X2 = COVXK(L,L,I) - COVXK(K,L,I)
615:             XKMLE(J,I) = (X1*XKEF(L,I)+X2*XKEF(K,I)) / (X1+X2)
616:             XKERR(J,I) =
617:                 & (COVXK(K,K,I)*COVXK(L,L,I)-COVXK(K,L,I)*
618:                 & COVXK(K,L,I)) / (X1+X2)
619:             XKERR(J,I) = SQRT(ABS(XKERR(J,I))) / XKMLE(J,I)*100.0
620:             if ( I.eq.NSKIP1 ) then
621:                 FMIX(L,J) = X1/(X1+X2)
622:                 FMIX(K,J) = X2/(X1+X2)
623:             end if
624:         end if
625:     320 continue
626: 330 continue
627: C
628: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF ALL 6 ESTIMATORS
629: C GET AN INVERSE MATRIX OF COVXK INTO COVVK
630: C
631: C
632: C ... covariance matrix has (NEST+1)*NEST/2 degree of freedom.
633: C (where NEST is number of combined estimators. here NEST=6)
634: C
635: C To keep the degree of freedom, required number of
636: C batches are calculated as;
637: C
638: C NEST* NB - NEST >= (NEST+1)*NEST/2 --> NB >= (NEST+3)/2
639: C

```

src/mvp/keff.f

```

640: C      ( In the equation above, "-NEST" is necessary because
641: C      expected values of each estimator are approximated by
642: C      ensemble means. )
643: C
644: C      NB00      = 3
645: C
646: C      do 340 I = 1, NBATCH - 2
647: C          XKERR(6,I) = 0.0
648: C          XKMLE(6,I) = -1.0
649: C      340 continue
650: C
651: C      do 430 I = 1, NBATCH - 2 - NB00
652: C
653: C      ... omit one estimator from too strongly correlated pair ...
654: C      ( non-diagonal elements of COVXK for omitted estimator is
655: C      cleared to 0.0 )
656: C
657: C      do 380 L = 6, 1, -1
658: C          JJCC(L) = 0
659: C          do 360 K = 1, L - 1
660: C
661: C              CORR1 = COVXK(K,L,I) / SQRT(COVXK(K,K,I)*COVXK(L,L,I))
662: C
663: C              if ( ABS(CORR1-1.0D0).lt.DECORR ) then
664: C                  JJCC(L) = 1
665: C                  COVXK(L,L,I) = 1.0D0
666: C                  do 350 KK = 1, 6
667: C                      if ( KK.ne.L ) then
668: C                          COVXK(KK,L,I) = 0.0D0
669: C                          COVXK(L,KK,I) = 0.0D0
670: C                      end if
671: C                  350 continue
672: C                  go to 370
673: C              end if
674: C          360 continue
675: C
676: C          370 continue
677: C          380 continue
678: C          if ( I.eq.NSKIP1 ) then
679: C              do 390 L = 1, 6
680: C                  JJCC(L) = JJCC(L)
681: C                  WTALL(L) = 0.0D0
682: C          390 continue
683: C          end if
684: C
685: C
686: C      ... invert covariance matrix ...
687: C
688: C      c+photo-nuc
689: C      ... A temporary array COVWK is necessary since COVXK is extended
690: C      to COVXK(10,10,NBATCH) for photonuclear treatment.
691: C      This algorithm should be modified in future.
692: C
693: C      call MTXINV( COVXK(1,1,I), 6, IWK1, CORR(1,1,2), 1.0D-16, ICON
694: C      &
695: C
696: C      do K = 1, 6
697: C          do L = 1, 6
698: C              COVWK(K,L) = COVXK(K,L,I)
699: C          end do
700: C      end do
701: C
702: C      call MTXINV( COVWK, 6, IWK1, CORR(1,1,2), 1.0D-16, ICON )
703: C
704: C      do K = 1, 6

```

```

705: C      do L = 1, 6
706: C          COVXK(K,L,I) = COVWK(K,L)
707: C      end do
708: C      end do
709: C      c-photo-nuc
710: C
711: C      if ( ICON.ne.0 ) then
712: C          write(IOT,'(1x,a,i8,a)')
713: C          & '!!! CANNOT INVERT COVARIANCE MATRIX (BATCH ', I,
714: C          & ' ) --> MAX.LIKELYHODD ESTIMATOR PRINTED AS -1'
715: C          XKERR(6,I) = 0.0
716: C          XKMLE(6,I) = -1.0
717: C      else
718: C          XKERR(6,I) = 0.0
719: C          XKMLE(6,I) = 0.0
720: C      end if
721: C
722: C      if ( XKMLE(6,I).ne.-1.0 ) then
723: C          do 410 K = 1, 6
724: C              do 400 L = 1, 6
725: C                  if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
726: C                      XKERR(6,I) = XKERR(6,I) + COVXK(K,L,I)
727: C                      XKMLE(6,I) = XKMLE(6,I) + COVXK(K,L,I)*XKEF(L,I)
728: C                      if ( I.eq.NSKIP1 ) then
729: C                          WTALL(L) = WTALL(L) + COVXK(K,L,I)
730: C                      end if
731: C                  end if
732: C
733: C      ... weight for "ALL" estimator (NSKIP+1 'th batch)
734: C      400 continue
735: C      410 continue
736: C
737: C      if ( I.eq.NSKIP1 ) then
738: C          do 420 L = 1, 6
739: C              if ( JJCC(L).eq.0 ) then
740: C                  WTALL(L) = WTALL(L) / XKERR(6,I)
741: C                  FMIX(L,6) = WTALL(L)
742: C              end if
743: C          420 continue
744: C      end if
745: C
746: C      XKERR(6,I) = 1./XKERR(6,I)
747: C      XKMLE(6,I) = XKMLE(6,I)*XKERR(6,I)
748: C      XKERR(6,I) = SQRT(ABS(XKERR(6,I))) / XKMLE(6,I)*100.
749: C      end if
750: C      430 continue
751: C
752: C      write(IOT,7100)
753: C      7100 format('1', ' << K-EFFECTIVE BY THE PRINCIPLE OF',
754: C      & ' MAXIMUM LIKELIHOOD',
755: C      & ' ( CUMULATIVE AFTER THIS BATCH ) >>'//1X,
756: C      & ' BATCH TRACK LENGTH EST. COLLISION EST.',
757: C      & ' ANALOG EST. ',
758: C      & ' PRODUCTION NEUTRON BALANCE',
759: C      & ' ALL ESTIMATORS'/1X,
760: C      & ' -----',
761: C      & ' -----',
762: C      & ' -----',
763: C      & ' -----')
764: C
765: C      do 440 I = 1, NBATCH - 2 - NB00
766: C          write(IOT,7080) I, (XKMLE(J,I),XKERR(J,I),J=1,6)
767: C      440 continue
768: C
769: C      do 450 I = NBATCH - 2 - NB00 + 1, NBATCH - 2

```

src/mvp/keff.f

```

770:      write(IOT,7120) I, (XKMLE(J,I),XKERR(J,I),J=1,5)
771:      450 continue
772:
773:      7120 format(1X,I7,5(1P,E13.5,'(' ,0P,F5.3,'%')',5X,15(' - '))
774:
775:      if ( JPHNU.ne.0 ) then
776:         write(IOT,7101)
777:         do I = 1, NBATCH - 2
778:            write(IOT,7081) I, XKMLE(8,I), XKERR(8,I)
779:         end do
780:      end if
781:      7101 format(// BATCH PHOTONEUTRON
782:      &      // -----)
783:
784:      write(IOT,7140) NSKIPL, NBATCH
785:      7140 format('1', ' << SUMMARY OF K-EFFECTIVE >>',
786:      &      '== RESULT OF BATCHES FROM ',I2,' TO ',I4,' ==//
787:      &      ' 1 : TRACK LENGTH ESTIMATOR (PRODUCTION)//
788:      &      ' 2 : COLLISION ESTIMATOR (PRODUCTION)//
789:      &      ' 3 : ANALOG ESTIMATOR (PRODUCTION)//
790:      &      ' 4 : TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)//
791:      &      ' 5 : COLLISION ESTIMATOR (NEUTRON BALANCE)//
792:      &      ' 6 : ANALOG ESTIMATOR (NEUTRON BALANCE)//
793: C
794: C ... printout correlation table ....
795: C
796:      write(IOT,'(3X,90(' - '))')
797:      write(IOT,'(1X,a)')
798:      &      ' KEFF ERROR(%) CORRELATION
799:      write(IOT,'(3X,90(' - '))')
800:
801:      do 460 J = 1, 6
802:         write(IOT,7160) J, XKEF(J,NSKIPL), XKSD(J,NSKIPL),
803:         &      (CORR(K,J,1),K=1,J)
804:      460 continue
805:
806:      7160 format(1X,I5,1P,E15.5,0P,F10.4,0P,6F9.3)
807:
808:      write(IOT,'(3X,90(' - '))')
809: C
810: C ... printout mixing table for M.L.E. ....
811: C
812:      write(IOT,7180) ((FMIX(K,J),K=1,6),J=1,6)
813: C
814:      7180 format(//1X,4X,'MIXING RATIO FOR MAXIMUM LIKELIHOOD ESTIMATORS'//
815:      &      4X,'-----',6('-----')/4X,
816:      &      ' M.L.E. ',13X,' PRODUCTION ',13X,13X,
817:      &      ' BALANCE ',4X,' ',2
818:      &      ('TRACK LENGTH ', ' COLLISION ', ' ANALOG ') /4X,
819:      &      '-----',6('-----')/4X,
820:      &      ' TRACK LENGTH ',6(F11.5,2X)/4X,' COLLISION ',6
821:      &      (F11.5,2X)/4X,' ANALOG ',6(F11.5,2X)/4X,
822:      &      ' PRODUCTION ',6(F11.5,2X)/4X,' NEUTRON BALANCE',6
823:      &      (F11.5,2X)/4X,' ALL ',6(F11.5,2X)/4X,
824:      &      '-----',6('-----'))
825: C
826: C
827:      JJCC1 = JJCC0(1) + JJCC0(2) + JJCC0(3) + JJCC0(4) + JJCC0(5)
828:      &      + JJCC0(6)
829:
830:      if ( XKMLE(6,NSKIPL).eq.-1.0 ) then
831:         write(IOT,'((4X,A/4X,A,A,A))')
832:      &      ' !!! The "ALL" estimator above is not valid !!!,
833:      &      ' Probably "Some estimations are too strongly ',
834:      &      ' correlated each other to get a ',

```

```

835:      &      'maximum likelihood estimation.'
836:
837:      else if ( JJCC1.gt.0 ) then
838:
839:         write(IOT,'((4X,A/4X,A,A))')
840:      &      ' !!! The "ALL" estimator above is not a whole combination',
841:      &      ' of the 6 estimators,',
842:      &      ' Because some estimators are too strongly correlated.'
843:         write(IOT,'(/4X,A/4X,6E12.5)') ' Weight of each estimator : ',
844:      &      (WTALL(L),L=1,6)
845:
846:      end if
847:
848:      write(IOT,7200) (XKMLE(J,NSKIPL),XKERR(J,NSKIPL),J=1,6)
849:      7200 format(//1X,4X,
850:      &      ' RESULTS BY THE PRINCIPLE OF MAXIMUM LIKELIHOOD'//
851:      &      4X,
852:      &      '+-----+'//
853:      &      4X,'| TRACK LENGTH --> KEFF=',1P,E12.5,' (' ,0P,F7.4,
854:      &      '%' )|/4X,'| COLLISION --> KEFF=',1P,E12.5,' (' ,
855:      &      0P,F7.4,'%')|/4X,'| ANALOG --> KEFF=',1P,
856:      &      E12.5,' (' ,0P,F7.4,'%')|/4X,
857:      &      '| PRODUCTION --> KEFF=',1P,E12.5,' (' ,0P,F7.4,
858:      &      '%' )|/4X,'| NEUTRON BALANCE--> KEFF=',1P,E12.5,' (' ,
859:      &      0P,F7.4,'%')|/4X,'| ALL --> KEFF=',1P,
860:      &      E12.5,' (' ,0P,F7.4,'%')|/4X,
861:      &      '+-----+'//
862:
863: C
864:      do 470 M = 1, 3
865:         write(IOT,7220) M, (XKMLE(J,NSKIPL)*
866:         &      (1.0-M*XKERR(J,NSKIPL)/100.0),XKMLE(J,NSKIPL)*
867:         &      (1.0+M*XKERR(J,NSKIPL)/100.0),J=1,6)
868:      470 continue
869: C
870:      7220 format(/1X,4X,' RANGE IN ',I1,' STANDARD DEVIATION'//4X,
871:      &      '+-----+'//
872:      &      4X,'| TRACK LENGTH --> ',F12.9,' < K < ',F12.9,' '|/4X,
873:      &      '| COLLISION --> ',F12.9,' < K < ',F12.9,' '|/4X,
874:      &      '| ANALOG --> ',F12.9,' < K < ',F12.9,' '|/4X,
875:      &      '| PRODUCTION --> ',F12.9,' < K < ',F12.9,' '|/4X,
876:      &      '| NEUTRON BALANCE--> ',F12.9,' < K < ',F12.9,' '|/4X,
877:      &      '| ALL --> ',F12.9,' < K < ',F12.9,' '|/4X,
878:      &      '+-----+'//
879: C
880: C-----
881: C Real variance estimation
882: C-----
883: C
884:      if ( NBATCH-NSKIPL.gt.NMXLAG ) then
885: C
886:      KK = 0
887:      do 490 J = 1, 4
888: C
889:      KK = KK + 1
890: C
891:      if ( J.eq.1 ) then
892:         ESTKEF(KK) = 'Track length(PRODUCTION)'
893:      else if ( J.eq.2 ) then
894:         ESTKEF(KK) = 'Collision (PRODUCTION)'
895:      else if ( J.eq.3 ) then
896:         ESTKEF(KK) = 'Analog (PRODUCTION)'
897:      else if ( J.eq.4 ) then
898:         ESTKEF(KK) = 'Track length(BALANCE)'
899:      else if ( J.eq.5 ) then

```

src/mvp/keff.f

```

900:      ESTKEF(KK) = 'Collision (BALANCE)'
901:      else if ( J.eq.6 ) then
902:        ESTKEF(KK) = 'Analog (BALANCE)'
903:      end if
904: C
905: C      ... calculate batchwise k-eff with weighting factor
906: C
907:      do 480 I = NSKIP + 1, NBATCH
908:        if ( NSKIPL.gt.1 ) then
909:          XSS1 = XSOC(NBATCH)-XSOC(NSKIPL-1)
910:        else
911:          XSS1 = XSOC(NBATCH)
912:        end if
913:        XKBB(I) = KKB(J,I)*XSOC(I)/XSS1*(NBATCH-NSKIP)
914: 480      continue
915: C
916:      EPS      = 1.0D-10
917:      JPR      = -1
918:      IDEBG    = 1
919: C
920:      ESTCMB = 'K-eff by '//ESTKEF(KK)
921:      call REALVR( ESTCMB, XKBB, 1, 1, NSKIP+1, NBATCH, AVG(KK)
922:      &          SIGA(KK), SIGR(KK), CA, CR, NMXLG, EPS, NEITER,
923:      &          JPR, IDEBG, IERR )
924: C
925: 490      continue
926: C
927:      do 520 J = 1, 6
928:        KK      = KK + 1
929: C
930:        if ( J.eq.1 ) then
931:          ESTKEF(KK) = 'M.L.E. of TRACK LENGTH estimators'
932:        else if ( J.eq.2 ) then
933:          ESTKEF(KK) = 'M.L.E. of COLLISION estimators'
934:        else if ( J.eq.3 ) then
935:          ESTKEF(KK) = 'M.L.E. of ANALOG estimators'
936:        else if ( J.eq.4 ) then
937:          ESTKEF(KK) = 'M.L.E. of PRODUCTION estimators'
938:        else if ( J.eq.5 ) then
939:          ESTKEF(KK) = 'M.L.E. of NEUTRON BALANCE estimators'
940:        else if ( J.eq.6 ) then
941:          ESTKEF(KK) = 'M.L.E. of ALL estimators'
942:        end if
943: C
944:      do 510 I = NSKIPL, NBATCH
945:        XKBB(I) = 0.0D0
946:        do 500 K = 1, 6
947:          XKBB(I) = XKBB(I) + KKB(K,I)*FMIX(K,J)
948: 500      continue
949:        if ( NSKIPL.gt.1 ) then
950:          XSS1 = XSOC(NBATCH)-XSOC(NSKIPL-1)
951:        else
952:          XSS1 = XSOC(NBATCH)
953:        end if
954:        XKBB(I) = XKBB(I)*XSOC(I)/XSS1*(NBATCH-NSKIP)
955: 510      continue
956: C
957:      EPS      = 1.0D-10
958:      JPR      = -1
959:      IDEBG    = 1
960: C
961:      ESTCMB = 'K-eff '//ESTKEF(KK)
962:      call REALVR( ESTCMB, XKBB, 1, 1, NSKIPL, NBATCH, AVG(KK),
963:      &          SIGA(KK), SIGR(KK), CA, CR, NMXLG, EPS, NEITER,
964:      &          JPR, IDEBG, IERR )

```

```

965: C
966: 520      continue
967: C
968:      NWARN    = 0
969:      write(IOT,7230)
970: 7230      format(1X,'*****',7('*****'))
971:      write(IOT,*) ' * Real variance is estimated by the method',
972:      &          ' of Nuc.Sci.Eng., 125,1(1997). *'
973: C
974: 7240      format(1X,' K-eff Estimator',24X,2X,' K-eff ',4X,
975:      &          ' SigA(%) ',1X,' SigR(%) ',1X,' SigR/SigA')
976: 7240      format(1X,' K-eff Estimator',19X,2X,' K-eff ',2X,
977:      &          ' SigA(%) ',1X,' SigR(%) ',1X,' SigR/SigA')
978: C
979: 7260      format(1X,'---',9('-----'))
980: 7260      format(1X,'-----',7('-----'))
981: 7280      format(1X,A40,2X,1P,E12.5,1X,0P,F11.5,1X,F11.5,1X,F11.5)
982: 7280      format(1X,A35,2X,1P,E12.5,1X,0P,F9.4,1X,F9.4,1X,F9.4)
983:      do 530 K = 1, KK
984:        RRR      = SQRT( SIGR(K) / (SIGA(K)+1.0D-20) )
985:        if ( RRR.gt.1.5 ) then
986:          NWARN    = NWARN + 1
987:          if ( NWARN.eq.1 ) then
988:            write(IOT,7240)
989:            write(IOT,7260)
990:          end if
991:          write(IOT,7280) ESTKEF(K), AVG(K),
992:          &          SQRT(SIGA(K)) / (AVG(K)+1.0D-20)*100D0,
993:          &          SQRT(SIGR(K)) / (AVG(K)+1.0D-20)*100D0,
994:          &          SQRT( SIGR(K) / (SIGA(K)+1.0D-20) )
995:        end if
996: 530      continue
997:      if ( NWARN.gt.0 ) then
998:        write(IOT,7260)
999:        write(IOT,*) ' * Results for other estimators are ',
1000:        &          'reasonable.'
1001:      else
1002:        write(IOT,*) ' * Standard deviation given above is ',
1003:        &          'reasonable.'
1004:      end if
1005:      write(IOT,7230)
1006:
1007:    end if
1008: C-----
1009: C ..... OUTPUT TO BINARY FILE (V)      EIGEN VALUES (AFTER EACH BATCH)
1010: C-----
1011: C      -----0-----1-----2-----3--
1012:      TAG      = 'K-EFF: AFTER EACH BATCH'
1013: C
1014: C
1015:      write(IORS) TAG, (XKEF(1,I),I=1,NBATCH-1),
1016:      &          (XKSD(1,I),I=1,NBATCH-1)
1017:      write(IORS) TAG, (XKEF(2,I),I=1,NBATCH-1),
1018:      &          (XKSD(2,I),I=1,NBATCH-1)
1019:      write(IORS) TAG, (XKEF(3,I),I=1,NBATCH-1),
1020:      &          (XKSD(3,I),I=1,NBATCH-1)
1021:      write(IORS) TAG, (XKEF(4,I),I=1,NBATCH-1),
1022:      &          (XKSD(4,I),I=1,NBATCH-1)
1023:      write(IORS) TAG, (XKEF(5,I),I=1,NBATCH-1),
1024:      &          (XKSD(5,I),I=1,NBATCH-1)
1025:      write(IORS) TAG, (XKEF(6,I),I=1,NBATCH-1),
1026:      &          (XKSD(6,I),I=1,NBATCH-1)
1027: C
1028: C      -----0-----1-----2-----3--
1029:      TAG      = 'K-EFF: MAXIMUM LIKELYHOOD'

```


src/mvp/keff.f

```
1030: C      -----
1031: C
1032:      write(IORS) TAG, (XKMLE(1,I),I=1,NBATCH-1),
1033:      &      (XKERR(1,I),I=1,NBATCH-1)
1034:      write(IORS) TAG, (XKMLE(2,I),I=1,NBATCH-1),
1035:      &      (XKERR(2,I),I=1,NBATCH-1)
1036:      write(IORS) TAG, (XKMLE(3,I),I=1,NBATCH-1),
1037:      &      (XKERR(3,I),I=1,NBATCH-1)
1038:      write(IORS) TAG, (XKMLE(4,I),I=1,NBATCH-1),
1039:      &      (XKERR(4,I),I=1,NBATCH-1)
1040:      write(IORS) TAG, (XKMLE(5,I),I=1,NBATCH-1),
1041:      &      (XKERR(5,I),I=1,NBATCH-1)
1042:      write(IORS) TAG, (XKMLE(6,I),I=1,NBATCH-1),
1043:      &      (XKERR(6,I),I=1,NBATCH-1)
1044: C
1045:      return
1046:      end
```

src/mvp/keffpt.f

```

1:      subroutine KEFFPT(TITLE, JPRTS, NBATCH, NSKIP, NPTDS, NPTCS,MAXDA,
2:      &                XSOC, XKEFP, XSOCB, XKB, XKPb,
3:      &                XKPb, XKPML, DELA, NUCPT, NUC, JPTNU,
4:      &                NUCID, JPRT, JPDEN, NTPT, NGSP, NOCS, NORDDS, NPTBE,
5:      &                MXPTOP, JPTOP)
6: C=<MVP/GMVP>=====
7: C PURPOSE   : Statistical analysis of delta-k
8: C CALLED IN : CADENZ
9: C CALL      :
10: C-----
11:      implicit real*8(A-H,O-Z)
12: c
13:      include '../shared/INC/_IOUNIT'
14:      include 'INC/_IOUNIT2'
15: c
16:      character*72 TITLE(2)
17:      integer JPRTS(*)
18:      real*8 DELA(MAXDA,NPTDS+NPTCS)
19:      real*8 XSOC(NBATCH)
20:      real*8 XKEFP(NUCPT,NTPT,7,NBATCH)
21:
22:      real*8 XSOCB(NBATCH)
23:      parameter (MAXKK = 10)
24:      real*8 XKB(MAXKK,NBATCH)
25:      real*8 XKPb(NUCPT,NTPT,7,NBATCH)
26:      real*8 XKPML(NUCPT,NTPT,6,NBATCH)
27:      real*8 XKPb(NUCPT,NTPT,6,NBATCH)
28:
29: c      real*8 COVXX(6,6,NBATCH)
30: c      real*8 XKERR(6,NBATCH)
31: c      real*8 CORR(6,6,*)
32: c      integer IKW1(6)
33: c
34: c      real*8 XKBB(NBATCH), CA(NMXLAG), CR(NMXLAG)
35: c
36:      integer JPTNU(NUC,NPTDS)
37:      character*16 NUCID(*)
38:      integer JPTOP(MXPTOP,NPTDS+NPTCS)
39:
40:      real*8 X1, X2, X3, X4, X5, X6, S1, XSS
41:      character*32 TAG
42: C
43: C ... criterion to reject too much correlated combination of
44: C estimation in maximum likelihood estimation.
45: C
46:      parameter( DECORR = 1.0D-5 )
47: C
48: C ... local data
49: C
50:      character*48 ESTCMB
51: C
52:      integer JJCC(6)
53:      integer JJCC0(6)
54:      real*8 WTALL(6)
55: C
56:      real*8 AVG(16), SIGA(16), SIGR(16)
57:      character*48 ESTKEF(16)
58: C
59: C ... mixing factor of M.L.E.'s ....
60: C
61: C      XKMLE(J) = Sum[I] ( XKB(I)*FMIX(I,J) )
62: C
63: C      real*8 FMIX(6,6)
64: C
65: C-----

```

```

66: C
67: C-----
68: C*** STATISTICAL ANALYSIS OF dk/da
69: C-----
70: C
71:      call HEADER( IOT, '      RESULT OF PERTURBATION CALCULATION      ' )
72:
73: C
74: C.... SET WCNTR(7) = TOTAL LEAKAGE IN THIS RUN
75: C
76: c      WCNTR(7) = WLEK
77: C
78:
79: c ... XSOC & XSOCB have been already evaluated in KEFF ...
80: c      XSOCB(1) = XSOC(1)
81: c      XSOC(1) = XSOCB(1)
82: c      do 104 INC = 1, NUCPT
83: c        do 105 IPT = 1, NTPT
84: c          XKPb(INC,IPT,4,1) = 0.0
85: c          XKPb(INC,IPT,5,1) = 0.0
86: c          XKPb(INC,IPT,6,1) = 0.0
87: c      XKEF(4,1) = XKEF(4,1) + XKEF(7,1)
88: c      XKEF(5,1) = XKEF(5,1) + XKEF(7,1)
89: c      XKEF(6,1) = XKEF(6,1) + XKEF(7,1)
90: c          XKPb(INC,IPT,1,1) = XKEFP(INC,IPT,1,1) /XSOC(1)
91: c          XKPb(INC,IPT,2,1) = XKEFP(INC,IPT,2,1) /XSOC(1)
92: c          XKPb(INC,IPT,3,1) = XKEFP(INC,IPT,3,1) /XSOC(1)
93: c      XKB(4,1) = XKEF(1,1) /XKEF(4,1)
94: c      XKB(5,1) = XKEF(2,1) /XKEF(5,1)
95: c      XKB(6,1) = XKEF(3,1) /XKEF(6,1)
96: c          XKPML(INC,IPT,1,1) = XKPb(INC,IPT,1,1)
97: c          XKPML(INC,IPT,2,1) = XKPb(INC,IPT,2,1)
98: c          XKPML(INC,IPT,3,1) = XKPb(INC,IPT,3,1)
99: c          XKPML(INC,IPT,4,1) = XKPb(INC,IPT,4,1)
100: c          XKPML(INC,IPT,5,1) = XKPb(INC,IPT,5,1)
101: c          XKPML(INC,IPT,6,1) = XKPb(INC,IPT,6,1)
102: c      105 continue
103: c      104 continue
104: C
105: C      XSOCB : SOURCE (EACH BATCH)
106: C      XKB : ESTIMATOR OF KEFF (EACH BATCH)
107: C      XKMLE : CUMULATIVE ESTIMATOR OF KEFF TO THIS BATCH
108: C      1: TRACK LENGTH ESTIMATOR (PRODUCTION RATE)
109: C      2: COLLISION ESTIMATOR (PRODUCTION RATE)
110: C      3: ANALOG ESTIMATOR (PRODUCTION RATE)
111: C      4: TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
112: C      5: COLLISION ESTIMATOR (NEUTRON BALANCE)
113: C      6: ANALOG ESTIMATOR (NEUTRON BALANCE)
114: C
115: c ... XSOC has been already evaluated in KEFF. redefined here.
116: c      do 114 I = 2, NBATCH
117: c        XSOC(I) = XSOC(I-1) + XSOCB(I)
118: c      114 continue
119: c
120: c      do 113 INC = 1, NUCPT
121: c        do 115 IPT = 1, NTPT
122: c          write(*,*) ' IPT = ',IPT
123: c-----
124: c      ICT = 0
125: c      do 116 IK = 1, NUC
126: c        if( IPT.le.NPTDS .and. JPTNU(IK,IPT).ne.0 ) ICT = ICT + 1
127: c        if( ICT.eq.INC ) then
128: c          write(*,*) ' Nuclide ID :',IK,NUCID(IK)
129: c          write(*,7051) IK,NUCID(IK)
130: c        end if

```

src/mvp/keffpt.f

```

131: 116 continue
132: c -----
133: do 110 I = 2, NBATCH
134: c   XKEF(4,I) = XKEF(4,I) + XKEF(7,I)
135: c   XKEF(5,I) = XKEF(5,I) + XKEF(7,I)
136: c   XKEF(6,I) = XKEF(6,I) + XKEF(7,I)
137: c   XKPML(INC,IPT,1,I) = XKEFP(INC,IPT,1,I) /XSOC(I)
138: c   XKPML(INC,IPT,2,I) = XKEFP(INC,IPT,2,I) /XSOC(I)
139: c   XKPML(INC,IPT,3,I) = XKEFP(INC,IPT,3,I) /XSOC(I)
140: c ... XSOC & XSOCB have been already evaluated in KEFF ...
141: c   XSOCB(I) = XSOC(I) - XSOC(I-1)
142:
143: c   XKPB(INC,IPT,1,I) = (XKEFP(INC,IPT,1,I)-XKEFP(INC,IPT,1,I-1))
144: c   & /XSOCB(I)
145: c   XKPB(INC,IPT,2,I) = (XKEFP(INC,IPT,2,I)-XKEFP(INC,IPT,2,I-1))
146: c   & /XSOCB(I)
147: c   XKPB(INC,IPT,3,I) = (XKEFP(INC,IPT,3,I)-XKEFP(INC,IPT,3,I-1))
148: c   & /XSOCB(I)
149: c ... dummy value ...
150: c   XKPB(INC,IPT,4,I) = 0.0
151: c   XKPB(INC,IPT,5,I) = 0.0
152: c   XKPB(INC,IPT,6,I) = 0.0
153: c   XKB(4,I) = (XKEF(1,I)-XKEF(1,I-1)) / (XKEF(4,I)-XKEF(4,I-1))
154: c   XKB(5,I) = (XKEF(2,I)-XKEF(2,I-1)) / (XKEF(5,I)-XKEF(5,I-1))
155: c   XKB(6,I) = (XKEF(3,I)-XKEF(3,I-1)) / (XKEF(6,I)-XKEF(6,I-1))
156: C980809
157: c ... new: source weighted "balance" estimators
158: c
159: c   XKMLE(4,I) = 0.0D0
160: c   XKMLE(5,I) = 0.0D0
161: c   XKMLE(6,I) = 0.0D0
162: c   do 100 J = 1, I
163: c     XKMLE(4,I) = XKMLE(4,I) + XKB(4,J)*XSOCB(J)
164: c     XKMLE(5,I) = XKMLE(5,I) + XKB(5,J)*XSOCB(J)
165: c     XKMLE(6,I) = XKMLE(6,I) + XKB(6,J)*XSOCB(J)
166: c 100 continue
167: c ... dummy value ...
168: c   XKPML(INC,IPT,4,I) = 0.0
169: c   XKPML(INC,IPT,5,I) = 0.0
170: c   XKPML(INC,IPT,6,I) = 0.0
171: c   XKMLE(4,I) = XKMLE(4,I) /XSOC(I)
172: c   XKMLE(5,I) = XKMLE(5,I) /XSOC(I)
173: c   XKMLE(6,I) = XKMLE(6,I) /XSOC(I)
174: 110 continue
175:
176: cyn write(IOT,7020) '=== dk/da ( EACH BATCH ) ==='
177: 7020 format(/1X,18X,A//1X,' PRODUCTION
178: & ' NEUTRON BALANCE ',/1X,
179: & ' BATCH TRACK L. COLLISION ANALOG ',
180: & ' TRACK L. COLLISION ANALOG'/1X,
181: & '-----',
182: & '-----')
183:
184: cyn write(IOT,7040) (I,(XKPB(INC,IPT,J,I),J=1,6),I=1,NBATCH)
185:
186: 7040 format(1X,I5,2X,1P,6E12.5)
187:
188: cyn write(IOT,7020)
189: cyn & '=== dk/da ( CUMULATIVE TO THIS BATCH ) ==='
190: cyn write(IOT,7040) (I,(XKPML(INC,IPT,J,I),J=1,6),I=1,NBATCH)
191:
192: 115 continue
193: 113 continue
194:
195: C-----

```

```

196: C ..... OUTPUT TO BINARY FILE (IV) EIGEN VALUES
197: C-----
198: C
199: C -----0-----1-----2-----3---
200: TAG = 'DIFFERENTIAL OF K: BATCH '
201: C -----
202: C
203: write(IORS) TAG, NUCPT, NTPT
204: do INC = 1, NUCPT
205: do IPT = 1, NTPT
206: write(IORS) ((XKPB(INC,IPT,J,I),I=1,NBATCH),J=1,6)
207: end do
208: end do
209: C
210: do INC = 1, NUCPT
211: do IPT = 1, NTPT
212: X1 = XKEFP(INC,IPT,1,NBATCH)
213: X2 = XKEFP(INC,IPT,2,NBATCH)
214: X3 = XKEFP(INC,IPT,3,NBATCH)
215: do J = NBATCH, 2, -1
216: XSS = XSOC(NBATCH) - XSOC(J-1)
217: XKEFP(INC,IPT,1,J) = (X1-XKEFP(INC,IPT,1,J-1)) /XSS
218: XKEFP(INC,IPT,2,J) = (X2-XKEFP(INC,IPT,2,J-1)) /XSS
219: XKEFP(INC,IPT,3,J) = (X3-XKEFP(INC,IPT,3,J-1)) /XSS
220: end do
221: XKEFP(INC,IPT,1,1) = X1/XSOC(NBATCH)
222: XKEFP(INC,IPT,2,1) = X2/XSOC(NBATCH)
223: XKEFP(INC,IPT,3,1) = X3/XSOC(NBATCH)
224: end do
225: end do
226:
227: C..... COVXX : COVARIANCE MATRIX
228:
229: c do 150 K = 1, (NBATCH-1)*6*6
230: c   COVXX(K,1,1) = 0.0
231: c 150 continue
232:
233: c do 190 I = 1, NBATCH - 1
234: c   do 180 K = 1, 6
235: c     do 170 L = 1, 6
236: c       if ( L.ge.K ) then
237: c
238: c         do 160 J = I, NBATCH
239: c           X1 = XSOCB(J) / (XSOC(I)-XSOCB(J))
240: c           COVXX(K,L,I) = COVXX(K,L,I)
241: c           & + X1*(XKB(K,J)-XKEF(K,I))*
242: c           & (XKB(L,J)-XKEF(L,I))
243: c 160 continue
244: c
245: c       else
246: c         COVXX(K,L,I) = COVXX(L,K,I)
247: c       end if
248: c 170 continue
249: c 180 continue
250: c 190 continue
251: C
252: c do 220 I = 1, NBATCH - 1
253: c   XX = 1.D0/(NBATCH-I+1)
254: c   do 210 K = 1, 6
255: c     do 200 L = 1, 6
256: c       COVXX(K,L,I) = COVXX(K,L,I)*XX
257: c 200 continue
258: c 210 continue
259: c 220 continue
260: C

```

src/mvp/keffpt.f

```

261: C..... KEEP CORRELATION MATRIX FOR ESTIMATION AFTER (NSKIP+1)TH BATCH
262: C
263:       NSKIP1 = NSKIP + 1
264: c
265: c       do 240 K = 1, 6
266: c           do 230 L = 1, 6
267: c               CORR(K,L,1) = COVXK(K,L,NSKIP1) /
268: c               &          SQRT(COVXK(K,K,NSKIP1)*COVXK(L,L,NSKIP1))
269: c       230 continue
270: c 240 continue
271: C
272: C..... ESTIMATION OF KEFF BY THE PRINCIPLE OF MAXIMUM LIKELYHOOD
273: C       XKMLP : KEFF
274: C       XKERR : STANDARD DEVIATION OF COMBINED ESTIMATION (%)
275: C           1 : TRACK LENGTH (PRODUCTION + BALANCE)
276: C           2 : COLLISION (PRODUCTION + BALANCE)
277: C           3 : ANALOG (PRODUCTION + BALANCE)
278: C           4 : PRODUCTION (TRK+COL+ANALOG)
279: C           5 : BALANCE (TRK+COL+ANALOG)
280: C           6 : ALL ESTIMATORS
281: C..... ESTIMATION OF KEFF BY EACH ESTIMATOR
282: C       XKEF : KEFF
283: C       XKSD : STANDARD DEVIATION (%)
284: C           1 : TRACK LENGTH (PRODUCTION)
285: C           2 : COLLISION (PRODUCTION)
286: C           3 : ANALOG (PRODUCTION)
287: C           4 : TRACK LENGTH (BALANCE)
288: C           5 : COLLISION (BALANCE)
289: C           6 : ANALOG (BALANCE)
290: C
291: c       do 260 J = 1, 6
292: c           do 250 I = 1, NBATCH - 1
293: c               XKSD(J,I) = SQRT(ABS(COVXK(J,J,I))) /XKEF(J,I)*100.
294: c 250 continue
295: c 260 continue
296: c
297: c       do 254 INC = 1, NUCPT
298: c           do 251 IPT = 1, NTPT
299: c               do 252 J = 1, 6
300: c                   XKPSD(INC,IPT,J,1) = XKPB(INC,IPT,J,1)*XKPB(INC,IPT,J,1)
301: c                   do 253 I = 2, NBATCH
302: c                       XKPSD(INC,IPT,J,I) = XKPSD(INC,IPT,J,I-1)
303: c                   &          + XKPB(INC,IPT,J,I)*XKPB(INC,IPT,J,I)
304: c 253 continue
305: c 252 continue
306: c 251 continue
307: c 254 continue
308: c
309: c       do 264 INC = 1, NUCPT
310: c           do 261 IPT = 1, NTPT
311: c               do 262 J = 1, 6
312: c                   XXKP = XKPSD(INC,IPT,J,NBATCH)
313: c                   do 263 I = 2, NBATCH
314: c                       K = NBATCH - I + 2
315: c                       XKPSD(INC,IPT,J,K) = XXKP - XKPSD(INC,IPT,J,K-1)
316: c 263 continue
317: c                   XKPSD(INC,IPT,J,1) = XXKP
318: c 262 continue
319: c 261 continue
320: c 264 continue
321: c
322: c       do 274 INC = 1, NUCPT
323: c           do 271 IPT = 1, NTPT
324: c               do 272 J = 1, 6
325: c                   XKPSD(INC,IPT,J,NBATCH) = 0.0

```

```

326:       do 273 I = 2, NBATCH
327:       K = NBATCH - I + 1
328:       XKPSD(INC,IPT,J,K) = ( XKPSD(INC,IPT,J,K)
329:       &          -I*XKEFP(INC,IPT,J,K)*XKEFP(INC,IPT,J,K) )/I/(I-1)
330:       XKPSD(INC,IPT,J,K) = sqrt(XKPSD(INC,IPT,J,K))
331: 273 continue
332: 272 continue
333: 271 continue
334: 274 continue
335: c
336: C       do 275 INC = 1, NUCPT
337: C       do 265 IPT = 1, NTPT
338: C       Ccyn write(IOT,7050) IPT
339: C       Cc7050 format('1',' PERTURBATION ID : ',i3)
340: C       Cc-----
341: C       ICT = 0
342: C       do 276 IK = 1, NUC
343: C           if( IPT.le.NPTDS .and. JPTNU(IK,IPT).ne.0 ) ICT = ICT + 1
344: C           if( ICT.eq.INC ) then
345: C               write(*,*) ' Nuclide ID : ',IK,NUCID(IK)
346: C               Ccyn write(IOT,7051) IK,NUCID(IK)
347: C           end if
348: C 276 continue
349: C 7051 format('1',' Nuclide ID : ',i3,2x,'(',a16,')')
350: C       Cc-----
351: C 265 continue
352: C 275 continue
353: C
354: C ... final output for perturbation calculation ...
355: C
356: IDS = 0
357: ICS = 0
358: do IPT = 1, NPTDS+NPTCS
359:     write(IOT,('' PERTURBATION ID:'' ,i4)) JPTOP(1,IPT)
360:     IMTHD = JPTOP(2,IPT) ! method = DOS or CS
361:     if(IMTHD.eq.1) then
362:         write(IOT,('' METHOD: DIFFERENTIAL OPERATOR SAMPLING''))
363:         IDS = IDS + 1
364:         call OUTPDS(IDS, IOT, NUCPT, NTPT, NBATCH, NORDDS, NPTDS,
365:         &          NPTCS, NSKIPL, NGSP, MAXDA, DELA, XKEFP, XKPSD)
366:     else if(IMTHD.eq.2) then
367:         write(IOT,('' METHOD: CORRELATED SAMPLING''))
368:         ICS = ICS + 1
369:         call OUTPCS(ICS, IOT, NUCPT, NTPT, NBATCH, NORDDS, NPTDS,
370:         &          NPTCS, NSKIPL, NGSP, MAXDA, DELA, XKEFP, XKPSD, NOCS)
371:     else
372:         write(IOT,('' XXX FATAL ERROR. IMTHD='',i4)) IMTHD
373:         stop
374:     end if
375:     write(IOT,('()'))
376: end do
377:
378: if (NPTBE.ne.0) then
379:     call OUTPKP(ICS, IOT, NUCPT, NTPT, NBATCH, NPTCS, NSKIPL,
380:     &          NGSP, XKEFP, XKPSD, NOCS, MAXXX, XKPB, XKPB)
381: end if
382:
383: return
384: end
385:
386: subroutine OUTPDS(IPT, IOT, NUCPT, NTPT, NBATCH, NORDDS, NPTDS,
387: &          NPTCS, NSKIPL, NGSP, MAXDA, DELA, XKEFP, XKPSD)
388:
389: C ... Output results for differential operator sampling
390:

```

src/mvp/keffpt.f

```

391:      real DELA(MAXDA,NPTDS+NPTCS)
392:      real*8 XKEFP(NUCPT,NTPT,7,NBATCH)
393:      real*8 XKPSD(NUCPT,NTPT,6,NBATCH)
394:
395:      INC = 1 ! currently INC must be 1.
396: C ... output suppressed up to 2nd order though NORDDS=8 ...
397:      NOUTO = 2 ! order up to which results are output.
398:
399:      write(IOT,'(a)')
400: & ' Diff. coef. of k-effective w/o pertrubed source effect'
401: write(IOT,'(a)') ' Order   Diff. Coef.      1 sigma   1 sigma(%)'
402: do IOD = 1, NOUTO
403:   IP = (IOD-1)*NPTDS+IPT
404:   DKEFF = XKEFP(INC,IP,2,NSKIPL)
405:   SDKEF = XKPSD(INC,IP,2,NSKIPL)
406:   write(IOT,'(1x,i3,3x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')
407: &   IOD, DKEFF, SDKEF, abs(SDKEF/DKEFF)*100.d0
408: end do
409:
410:   ISP = NGSP
411:   write(IOT,'(a,i2)')
412: & ' Pertrubed source effect (PSE) : generation length = ',ISP
413: write(IOT,'(a)') ' Order   Diff. Coef.      1 sigma   1 sigma(%)'
414: do IOD = 1, NOUTO
415:   IP = (NORDDS+(IOD-1)*NGSP+ISP)*NPTDS+IPT
416:   DKSP = XKEFP(INC,IP,2,NSKIPL)
417:   SDKSP = XKPSD(INC,IP,2,NSKIPL)
418:   write(IOT,'(1x,i3,3x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')
419: &   IOD, DKSP, SDKSP, abs(SDKSP/DKSP)*100.d0
420: end do
421:
422:   write(IOT,'(/' Perturbed source effect as a function of',
423: & ' ' generation length.'')')
424: write(IOT,'(' Output to check convergence of PSE.''))
425: do IOD = 1, NOUTO
426:   write(IOT,'(/' Order = ',i3)') IOD
427:   write(IOT,'(a)')
428: &   ' Gen.   Diff. Coef.      1 sigma   1 sigma(%)'
429: do ISP = 1, NGSP
430:   IP = (NORDDS+(IOD-1)*NGSP+ISP)*NPTDS+IPT
431:   DKSP = XKEFP(INC,IP,2,NSKIPL)
432:   SDKSP = XKPSD(INC,IP,2,NSKIPL)
433:   write(IOT,'(1x,i3,3x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')
434: &   ISP, DKSP, SDKSP, abs(SDKSP/DKSP)*100.d0
435: end do
436: end do
437:
438: C ... DELTA K ...
439:
440:   write(IOT,'(/(a,a)')
441: & ' Change in k-effective (delta k) for specified fractional',
442: & ' change in perturbation paramter'
443:
444: do IDA = 1, MAXDA
445:   if(DELA(IDA,IPT).ne.0.) then
446:     write(IOT,'(/' Fractional change = ',e13.5)')
447: &     DELA(IDA,IPT)
448:     DELAI = dble(DELA(IDA,IPT))
449:
450:     write(IOT,'(a,a)')
451: &     ' Delta k w/o pertrubed source effect',
452: &     ' (cumulative to the order)'
453:     write(IOT,'(a)')
454: &     ' Order   Delta k      1 sigma   1 sigma(%)'
455:     DELKI = 0.d0 ! delta k

```

```

456:     VRDKI = 0.d0 ! variance of delta k
457:     FACTR = 1 ! factorial for Taylor coef.
458:     do IOD = 1, NOUTO
459:       IP = (IOD-1)*NPTDS+IPT
460:       DKEFFI = XKEFP(INC,IP,2,NSKIPL)
461:       SDKEFI = XKPSD(INC,IP,2,NSKIPL)
462:       FACTR = FACTR*dble(IOD)
463:       DELKI = DELKI + DKEFFI*DELA**IOD/FACTR
464:       VRDKI = VRDKI + (SDKEFI*DELA**IOD/FACTR)**2
465:       SDDKI = sqrt(VRDKI)
466:       FSDDK = abs(SDDKI/DELKI)*100.d0
467:       write(IOT,'(1x,i3,3x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')
468: &       IOD, DELKI, SDDKI, FSDDK
469:     end do
470:
471:     ISP = NGSP ! generation length = NGSP = 10
472:     write(IOT,'(a,a,i2)')
473: &     ' Delta k with pertrubed source effect',
474: &     ' (cumulative to the order): generation length = ',ISP
475:     write(IOT,'(a)')
476: &     ' Order   Delta k      1 sigma   1 sigma(%)'
477:     DKSP = 0.d0 ! delta k
478:     VDKSP = 0.d0 ! variance of delta k
479:     FACTR = 1 ! factorial for Taylor coef.
480:     do IOD = 1, NOUTO
481:       IP1 = (IOD-1)*NPTDS+IPT
482:       DCK = XKEFP(INC,IP1,2,NSKIPL) ! diff. coef. of k
483:       SDCK = XKPSD(INC,IP1,2,NSKIPL) ! std. dev. of k
484:       IP2 = (NORDDS+(IOD-1)*NGSP+ISP)*NPTDS+IPT
485:       DCSP = XKEFP(INC,IP2,2,NSKIPL) ! diff. coef. of PSE
486:       SDCSP = XKPSD(INC,IP2,2,NSKIPL) ! std. dev. of PSE
487:       FACTR = FACTR*dble(IOD)
488:       DKSP = DKSP + (DCK+DCSP)*DELA**IOD/FACTR
489:       VDKSP = VDKSP + ((SDCK+SDCSP)*DELA**IOD/FACTR)**2
490:       SDKSP = sqrt(VDKSP)
491:       FDKSP = abs(SDKSP/DKSP)*100.d0
492:       write(IOT,'(1x,i3,3x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')
493: &       IOD, DKSP, SDKSP, FDKSP
494:     end do
495:   end if
496: end do
497: return
498: end
499:
500:   subroutine OUTPCS(IPT, IOT, NUCPT, NTPT, NBATCH, NORDDS, NPTDS,
501: & NPTCS, NSKIPL, NGSP, MAXDA, DELA, XKEFP, XKPSD, NOCS)
502:
503: C ... Output results for differential operator sampling
504:
505:   real DELA(MAXDA,NPTDS+NPTCS)
506:   real*8 XKEFP(NUCPT,NTPT,7,NBATCH)
507:   real*8 XKPSD(NUCPT,NTPT,6,NBATCH)
508:
509:   INC = 1 ! currently INC must be 1.
510:
511:   write(IOT,'(/' Fractional Change = ',e13.5)')
512: &   DELA(1,NPTDS+IPT)
513:
514:   write(IOT,'(a)')
515: &   ' Delta k w/o pertrubed source effect'
516:   write(IOT,'(a)')
517: &   ' Delta k      1 sigma   1 sigma(%)'
518:   DELK0 = XKEFP(INC,NOCS+IPT,2,NSKIPL)
519:   SDDK0 = XKPSD(INC,NOCS+IPT,2,NSKIPL)
520:   write(IOT,'(1x,6x,1pe13.5,2x,1pe11.3,2x,0pf7.2)')

```

src/mvp/keffpt.f

```

521:      & DELK0, SDDK0, abs(SDDK0/DELK0)*100.d0
522:      ISP = NGSP ! NGSP = 10 (default)
523:      IP = NOCS+(ISP+1)*NPTCS+IPT
524:      DELKS = XKEFP(INC,IP,2,NSKIP1)
525:      SDDKS = XKPSD(INC,IP,2,NSKIP1)
526:      DELK = DELK0 + DELKS
527:      SDDK = sqrt(SDDK0**2 + SDDKS**2)
528:      write(IOT,'(a)')
529:      & ' Delta k with pertrubed source effect'
530:      write(IOT,'(a)')
531:      & ' Delta k 1 sigma 1 sigma(%)'
532:      write(IOT,'(1x,6x,1p13.5,2x,1p11.3,2x,0pf7.2)')
533:      & DELK, SDDK, abs(SDDK/DELK)*100.d0
534:
535:      write(IOT,'(/' Perturbed source effect as a function of',
536:      & ' generation length.'')')
537:      write(IOT,'(/' Output to check convergence of PSE.'')')
538:      write(IOT,'(a)')
539:      & ' Gen. Delta k 1 sigma 1 sigma(%)'
540:      do ISP = 1, NGSP
541:        IP = NOCS+(ISP+1)*NPTCS+IPT
542:        DELKS = XKEFP(INC,IP,2,NSKIP1)
543:        SDDKS = XKPSD(INC,IP,2,NSKIP1)
544:        write(IOT,'(1x,i3,3x,1p13.5,2x,1p11.3,2x,0pf7.2)')
545:        & ISP, DELKS, SDDKS, abs(SDDKS/DELKS)*100.d0
546:      end do
547:
548:      return
549:      end
550:
551:      subroutine OUTPKP(IPT, IOT, NUCPT, NTPT, NBATCH, NPTCS, NSKIP,
552:      & NGSP, XKEFP, XKPSD, NOCS, MAXXK, XKB, XKPBS)
553: C=====
554: C Output kinetics parameters
555: C=====
556:      implicit real*8(A-H,O-Z)
557:
558:      real*8 XKB(MAXXK,NBATCH)
559:      real*8 XKPBS(NUCPT,NTPT,7,NBATCH)
560:      real*8 XKEFP(NUCPT,NTPT,7,NBATCH)
561:      real*8 XKPSD(NUCPT,NTPT,6,NBATCH)
562:
563: C ... local work array ...
564:      real*8 XKPBS, XBB, AVEB, VARB
565:      real*8 PSEB, AVEPSE, VARPSE
566:      allocatable XKPBS(:, :) ! sum of dk/da + PSE for beff
567:      allocatable XBB(:, :) ! beff in each batch
568:      allocatable AVEB(:) ! average of beff
569:      allocatable VARB(:) ! variance of beff
570:      allocatable PSEB(:, :) ! PSE in each batch
571:      allocatable AVEPSE(:) ! average of PSE
572:      allocatable VARPSE(:) ! variance of PSE
573:
574: C ... allocate memory ...
575:      NPTMAX = NGSP+3 ! # of c.s. tally for beff
576:      NPTMAX = NPTMAX+NGSP+1 ! # of d.o.s. tally for beff
577:      NPTMAX = NPTMAX+NGSP+1 ! # of d.o.s. tally for lambda
578:      allocate(XKPBS(NPTMAX,NBATCH))
579:      allocate(XBB(NPTMAX,NBATCH))
580:      allocate(AVEB(NPTMAX))
581:      allocate(VARB(NPTMAX))
582:      allocate(PSEB(NPTMAX,NBATCH))
583:      allocate(AVEPSE(NPTMAX))
584:      allocate(VARPSE(NPTMAX))
585:

```

```

586: C -----
587:      INC = 1 ! currently INC must be 1.
588:      NSKIP1 = NSKIP+1
589:
590: c ... calculate beff in each batch
591:
592:      IPBEF = NGSP+3 ! pointer to beff tally
593:      IPLAM = IPBEF+NGSP+1 ! pointer to lambda tally
594:      IPBOL = IPLAM+NGSP+1 ! pointer to beff/lambda tally
595:      do I = 1, NBATCH
596:        XKPBS(1,I) = XKPBS(1,1,2,I) ! beta : correlated
597:        XBB(1,I) = -XKPBS(1,1,2,I)/XKB(2,I)
598:        do J = 2, NGSP+2 ! loop for adding PSE
599:          XKPBS(J,I) = XKPBS(1,1,2,I)+XKPBS(1,J,2,I)
600:          XBB(J,I) = -XKPBS(J,I)/XKB(2,I)
601:          PSEB(J,I) = -XKPBS(1,J,2,I)/XKB(2,I)
602:        end do
603:
604:        XKPBS(IPBEF,I) = XKPBS(1,IPBEF,2,I) ! beta : differential
605:        XBB(IPBEF,I) = XKPBS(1,IPBEF,2,I)/XKB(2,I)
606:        do J = IPBEF+1, IPBEF+NGSP ! loop for adding PSE
607:          XKPBS(J,I) = XKPBS(1,IPBEF,2,I)+XKPBS(1,J,2,I)
608:          XBB(J,I) = XKPBS(J,I)/XKB(2,I)
609:          PSEB(J,I) = XKPBS(1,J,2,I)/XKB(2,I)
610:        end do
611:
612:        XKPBS(IPLAM,I) = XKPBS(1,IPLAM,2,I) ! lambda : differential
613:        XBB(IPLAM,I) = -XKPBS(1,IPLAM,2,I)/XKB(2,I)**2
614:        do J = IPLAM+1, IPLAM+NGSP ! loop for adding PSE
615:          XKPBS(J,I) = XKPBS(1,IPLAM,2,I)+XKPBS(1,J,2,I)
616:          XBB(J,I) = -XKPBS(J,I)/XKB(2,I)**2
617:          PSEB(J,I) = -XKPBS(1,J,2,I)/XKB(2,I)**2
618:        end do
619:      end do
620:
621: C ... average
622:
623:      RN = dble(NBATCH-NSKIP)
624:      SUMK = 0.d0
625:      SUMK2 = 0.d0
626:      do I = NSKIP+1, NBATCH
627:        SUMK = SUMK + XKB(2,I)
628:        SUMK2 = SUMK2 + XKB(2,I)**2
629:      end do
630:      AVEK = SUMK/RN
631:      VARK = (SUMK2/RN-AVEK**2)/(RN-1.d0)
632:
633:      do J = 1, NPTMAX
634:        AVEB(J) = 0.d0
635:        VARB(J) = 0.d0
636:        AVEPSE(J) = 0.d0
637:        VARPSE(J) = 0.d0
638:        do I = NSKIP+1, NBATCH
639:          AVEB(J) = AVEB(J) + XBB(J,I)
640:          VARB(J) = VARB(J) + XBB(J,I)**2
641:          AVEPSE(J) = AVEPSE(J) + PSEB(J,I)
642:          VARPSE(J) = VARPSE(J) + PSEB(J,I)**2
643:        end do
644:        AVEB(J) = AVEB(J)/RN
645:        VARB(J) = (VARB(J)/RN-AVEB(J)**2)/(RN-1.d0)
646:        AVEPSE(J) = AVEPSE(J)/RN
647:        VARPSE(J) = (VARPSE(J)/RN-AVEPSE(J)**2)/(RN-1.d0)
648:      end do
649:
650: c ... output results

```

src/mvp/keffpt.f

```

651:
652:     write(IOT,'(lx,a)')
653:     & 'Results with correlated sampling'
654:     write(IOT,'(lx,a)')
655:     & 'Beta-effective results w/o perturbed source effect'
656:     write(IOT,'(lx,a)')
657:     & '      b-eff      1 sigma      1 sigma(%)'
658:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
659:     & AVEB(1), sqrt(VARB(1)), sqrt(VARB(1))/AVEB(1)*100.d0
660:     write(IOT,'(lx,a)')
661:     & 'Beta-effective results with perturbed source effect'
662:     write(IOT,'(lx,a)')
663:     & '      b-eff      1 sigma      1 sigma(%)'
664:     IP = NGSP+2
665:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
666:     & AVEB(IP), sqrt(VARB(IP)), sqrt(VARB(IP))/AVEB(IP)*100.d0
667:
668:     write(IOT,'('' Perturbed source effect as a function of'',
669:     & '' generation length.'')')
670:     write(IOT,'('' Output to check convergence of PSE.'')')
671:     write(IOT,'(lx,a)')
672:     & 'Gen.      b-eff      1 sigma      1 sigma(%)'
673:     do ISP = 1, NGSP
674:         write(IOT,'(lx,i3,3x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
675:         & ISP, AVEPSE(IP+2), sqrt(VARPSE(IP+2)),
676:         & sqrt(VARPSE(IP+2))/abs(AVEPSE(IP+2))*100.d0
677:     end do
678:
679:     write(IOT,'(/lx,a)')
680:     & 'Results with differential operator sampling'
681:     write(IOT,'(lx,a)')
682:     & 'Beta-effective results w/o perturbed source effect'
683:     write(IOT,'(lx,a)')
684:     & '      b-eff      1 sigma      1 sigma(%)'
685:     IP = IPBEF
686:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
687:     & AVEB(IP), sqrt(VARB(IP)), sqrt(VARB(IP))/AVEB(IP)*100.d0
688:     write(IOT,'(lx,a)')
689:     & 'Beta-effective results with perturbed source effect'
690:     write(IOT,'(lx,a)')
691:     & '      b-eff      1 sigma      1 sigma(%)'
692:     IP = IPBEF+NGSP
693:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
694:     & AVEB(IP), sqrt(VARB(IP)), sqrt(VARB(IP))/AVEB(IP)*100.d0
695:
696:     write(IOT,'('' Perturbed source effect as a function of'',
697:     & '' generation length.'')')
698:     write(IOT,'('' Output to check convergence of PSE.'')')
699:     write(IOT,'(lx,a)')
700:     & 'Gen.      b-eff      1 sigma      1 sigma(%)'
701:     do ISP = 1, NGSP
702:         IP = IPBEF+ISP
703:         write(IOT,'(lx,i3,3x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
704:         & ISP, AVEPSE(IP), sqrt(VARPSE(IP)),
705:         & sqrt(VARPSE(IP))/abs(AVEPSE(IP))*100.d0
706:     end do
707:
708:     write(IOT,'(/lx,a)')
709:     & 'Results with differential operator sampling'
710:     write(IOT,'(lx,a)')
711:     & 'Lambda results w/o perturbed source effect'
712:     write(IOT,'(lx,a)')
713:     & '      lambda      1 sigma      1 sigma(%)'
714:     IP = IPLAM
715:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
716:
717:     & AVEB(IP), sqrt(VARB(IP)), sqrt(VARB(IP))/AVEB(IP)*100.d0
718:     write(IOT,'(lx,a)')
719:     & 'Lambda results with perturbed source effect'
720:     write(IOT,'(lx,a)')
721:     & '      lambda      1 sigma      1 sigma(%)'
722:     IP = IPLAM+NGSP
723:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
724:     & AVEB(IP), sqrt(VARB(IP)), sqrt(VARB(IP))/AVEB(IP)*100.d0
725:
726:     write(IOT,'('' Perturbed source effect as a function of'',
727:     & '' generation length.'')')
728:     write(IOT,'('' Output to check convergence of PSE.'')')
729:     write(IOT,'(lx,a)')
730:     & 'Gen.      lambda      1 sigma      1 sigma(%)'
731:     do ISP = 1, NGSP
732:         IP = IPLAM+ISP
733:         write(IOT,'(lx,i3,3x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
734:         & ISP, AVEPSE(IP), sqrt(VARPSE(IP)),
735:         & sqrt(VARPSE(IP))/abs(AVEPSE(IP))*100.d0
736:     end do
737:
738:     write(IOT,'(/lx,a)')
739:     & 'Results with differential operator sampling'
740:     write(IOT,'(lx,a)')
741:     & 'Beta-effective/Lambda results w/o perturbed source effect'
742:     D1 = abs(AVEB(IPBEF))
743:     E1 = sqrt(VARB(IPBEF))
744:     D2 = abs(AVEB(IPLAM))
745:     E2 = sqrt(VARB(IPLAM))
746:     VVW = D1/D2*sqrt((E1/D1)**2+(E2/D2)**2)
747:     write(IOT,'(lx,a)')
748:     & '      beff/lam      1 sigma      1 sigma(%)'
749:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
750:     & D1/D2, VVW, VVW/D1*D2*100.d0
751:     write(IOT,'(lx,a)')
752:     & 'Beta-effective/Lambda results with perturbed source effect'
753:     D1 = abs(AVEB(IPBEF+NGSP))
754:     E1 = sqrt(VARB(IPBEF+NGSP))
755:     D2 = abs(AVEB(IPLAM+NGSP))
756:     E2 = sqrt(VARB(IPLAM+NGSP))
757:     VVW = D1/D2*sqrt((E1/D1)**2+(E2/D2)**2)
758:     write(IOT,'(lx,a)')
759:     & '      beff/lam      1 sigma      1 sigma(%)'
760:     write(IOT,'(lx,6x,lpe13.5,2x,lpe11.3,2x,0pf7.2)')
761:     & D1/D2, VVW, VVW/D1*D2*100.d0
762: C ... release dynamic memory
763:     deallocate(XKPBS, XBB, AVEB, VARB)
764:     deallocate(PSEB, AVEPSE, VARPSE)
765:
766:     return
767: end

```

src/mvp/kpinit.f

```
1:      subroutine KPINIT
2: C=====
3: C purpose: initialize particle symbol strings for MVP
4: C called in: MAIN
5: C calls : none
6: C-----
7: C initialization in this routine must synchronize with contents of
8: C included files _KPIDS and _KPSYMS
9: C=====
10:      include '../shared/INC/_IOUNIT'
11: C
12:      include 'INC/_KPIDS'
13:      include 'INC/_KPSYMS'
14: c##<2007/03/14:PN3:
15:      include 'INC/_KPMASS'
16: c##>
17: C
18:      include 'INC/_FLAGS'
19: C
20: C
21: C ... number of particles possible to treat in this code
22: C (this should be maximum of possible KP*)
23: C
24: C parameter ( KPLIM = ? )
25: C
26: C
27: C ... particle symbol string and their alias (or short form)
28: C
29: C character*16 KPSYM
30: C common /CKPSYM/ KPSYM(2,KPLIM)
31: C
32: C
33: C-----
34: C
35:      if ( MKPAR.lt.KPLIM ) then
36: c##<2007/03/14:PN3:
37: c##      write(IMG,*) 'XXX(KPINIT) Array size of JKPAR(*) must be',
38: c##      &      'greater than KPLIM. Program error. Check your source!!!'
39:      write(IMG,'(1X,A,A)')
40:      &      'XXX(KPINIT) Array size of JKPAR(*) must be greater',
41:      &      ' than KPLIM. Program error. Check your source!!!'
42: c##>
43:      stop 666
44:      end if
45: C
46: C-----
47: C
48: c##<2007/03/14:PN3:
49: c##C      === possible initialization
50: C      <<< particle symbol string >>>
51: c##>
52: C
53: C ... neutron and photon
54:      KPSYM(1,KPNEUT) = 'NEUTRON'
55:      KPSYM(2,KPNEUT) = 'N'
56:      KPSYM(1,KPPHOT) = 'PHOTON'
57:      KPSYM(2,KPPHOT) = 'P'
58: C ... electron and proton
59:      KPSYM(1,KPELEC) = 'ELECTRON'
60:      KPSYM(2,KPELEC) = 'EL'
61: C      KPSYM(1,KPPROT) = 'PROTON'
62: C      KPSYM(2,KPPROT) = 'PR'
63: C ... pi mesons (pi-0 pi+ pi- )
64: C      KPSYM(1,KPIO) = 'PIO'
65: C      KPSYM(2,KPIO) = 'PIO'
```

```
66: C      KPSYM(1,KPIIP) = 'PI-PLUS'
67: C      KPSYM(2,KPIIP) = 'PIP'
68: C      KPSYM(1,KPPIM) = 'PI-MINUS'
69: C      KPSYM(2,KPPIM) = 'PIM'
70: C      ... mu mesons (mu+ mu- )
71: C      KPSYM(1,KPMUP) = 'MU-PLUS'
72: C      KPSYM(2,KPMUP) = 'MUP'
73: C      KPSYM(1,KPMUM) = 'MU-MINUS'
74: C      KPSYM(2,KPMUM) = 'MUM'
75: C
76: c##<2007/03/14:PN3:
77: C      <<< mass in n.m.u. >>>
78: C
79:      PMASS(KPNEUT) = 1.0D+0
80:      PMASS(KPPHOT) = 0.0D+0
81:      PMASS(KPELEC) = 9.1093897D-31 / 1.6749286D-27
82: C      PMASS(KPPROT) = 0.998622D0
83: C      PMASS(5) = 0.998622D0*(2.0141018D0/1.0078250D0) ! deuteron
84: C      PMASS(6) = 0.998622D0*(3.01604929D0/1.0078250D0) ! triton
85: C      PMASS(7) = 0.998622D0*(3.0160293D0/1.0078250D0) ! He-3
86: C      PMASS(8) = 0.998622D0*(4.0026032D0/1.0078250D0) ! alpha
87: C
88: c##>
89:      return
90:      end
```


src/mvp/kpsymb.f

```
1:      subroutine KPSYMB( PSYM, DIR,  PID,  MODE )
2: C=====
3: C purpose: conversion to/from particle symbol from/to particle ID
4: C         for CGVIEW code
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C
8: C i/o PSYM : particle symbol string
9: C i DIR : a character to specify direction of conversion
10: C      '>' : from symbol to ID
11: C          returns PID=0 if no matching symbol is found.
12: C      '<' : from ID to symbol
13: C          returns PSYM=' ' if no matching ID is found.
14: C o/i PID : particle ID number
15: C i MODE : returned symbol string when DIR='<';
16: C         MODE=0 : full symbol name
17: C         MODE=1(<>0) : short symbol name
18: C=====
19:      include '../shared/INC/_IOUNIT'
20: C
21:      include 'INC/_KPLIDS'
22:      include 'INC/_KPSYMS'
23: C
24:      character*(*) PSYM, DIR
25:      integer PID, MODE
26: C
27: C-----
28: C
29:      if ( DIR.eq.'>' ) then
30:          PID = 0
31: C
32: C      .... check full name first ....
33: C
34:          do 100 I = 1, KPLIM
35:              if ( KPSYM(1,I).eq.PSYM ) then
36:                  PID = I
37:                  go to 110
38:              end if
39:          100 continue
40:          110 continue
41:          if ( PID.gt.0 ) return
42: C
43: C      .... check short name (alias) ....
44: C
45:          do 120 I = 1, KPLIM
46:              if ( KPSYM(2,I).eq.PSYM ) then
47:                  PID = I
48:                  go to 130
49:              end if
50:          120 continue
51:          130 continue
52:          return
53: C
54:      else if ( DIR.eq.'<' ) then
55: C
56:          PSYM = ' '
57:          if ( PID.lt.1 .or. PID.gt.KPLIM ) return
58: C
59:          if ( MODE.eq.0 ) then
60:              PSYM = KPSYM(1,PID)
61:          else
62:              PSYM = KPSYM(2,PID)
63:          end if
64: C
65:      else
```

```
66:          write(IMGMSG,*) 'XXX(KPSYMB) Program error:',
67:      &          ' invalid conversion direction key ',
68:      &          '<', DIR,'> is given (must be ">" or "<").'
69:          stop 666
70:      end if
71: C
72:      return
73:      end
```

src/mvp/main.f

```
1: *VOCL TOTAL,SCALAR
2: C
3: C   JAEA VECTORIZED MONTE CARLO TRANSPORT CODE. MAIN ROUTINE
4: C
5: C
6: C
7: C
8: C
9: C
10: C
11: C
12: C
13: C
14: C
15: C   The continuous energy neutron/photon transport Monte Carlo
16: C   simulation code of Japan Atomic Research Agency (JAEA).
17: C
18: C -----
19: C   History of development
20: C -----
21: C
22: C 1989:
23: C   First internal version is made as continuous energy version of
24: C   the multi-group neutron Monte Carlo code GMVP, which is developed
25: C   also in the Reactor System Laboratory of JAERI, in 1989.
26: C 1991:
27: C   Photon reaction treatment was implemented.
28: C 1993:
29: C   Modified as a multi-platform code which runs on many UNIX systems
30: C   as well as the FACOM VP supercomputer on which this code had been
31: C   developed.
32: C 1994:
33: C   Opened for public use as MVP version 94.1.
34: C 1995:
35: C   Support HI/OSF - SYSTEM(HIOSF) (Hitachi S-series)
36: C   Support SGI (IRIX 5.2)
37: C 1996:
38: C   Support CRAY MP-series, LINUX, FreeBSD & 32bit MS-Windows
39: C
40: C -----
41: C   Development staff
42: C -----
43: C
44: C Version 3:
45: C   Research Group for Reactor Physics and Standard Nuclear
46: C   Code System of JAEA
47: C   Dr. Y.Nagaya, K.Okumura, Dr. T.Mori
48: C
49: C Version 2:
50: C   Reactor Physics Group of JAERI:
51: C   Y.Nagaya, K.Okumura, Dr. T.Mori, Dr. M.Nakagawa
52: C   supported by
53: C   M.Sasaki & K.Kaneko
54: C
55: C Version 1:
56: C   Reactor System Laboratory of JAERI:
57: C   Dr. T.Mori, Dr. M.Nakagawa
58: C   supported by
59: C   The Japan Research Institute, Limited:
60: C   M.Sasaki (coding,debugging etc.) & K.Kaneko (x-sec library)
61: C   and members of Reactor System Laboratory.
62: C
63: C=====
64: C
65: C/#IF CMAIN
```

```
66: *      subroutine FTMAIN
67: C/#ELSE
68: *      program MVP
69: C/#ENDIF
70: C
71: C -----
72: C
73: C/#IF MS_VISUAL
74: C ... enable to use iargc, getarg for Visual Fortran.
75: *      use dflib
76: *      use dfport
77: C/#ENDIF
78: C
79: C/#IF PARA(VPP)
80: *      include '../shared/INC/_VPPPARA'
81: C/#ENDIF
82: *      include 'INC/_ATPOOL'
83: C
84: *      character*256 CSTRNG
85: C
86: *      character*128 TSKINF
87: C
88: C/#IF PARA(VPP)
89: *      real*8 TTT0, TTT1
90: *      !xocl processor p(mpe)
91: C/#ENDIF
92: C
93: *      include '../shared/INC/_IOUNIT'
94: C
95: C -----
96: C
97: C/#IF UNBUFFER
98: C
99: C ... FUNBUF: set standard output unbuffered
100: C
101: C      call FUNBUF()
102: C/#ENDIF
103: C
104: C ... initialize some I/O unit variables.
105: C
106: C      call IIOSSET
107: C
108: C ... check environment variables if possible ...
109: C
110: C (FUNBUF may be called by checking environment variable)
111: C
112: *      ATPOOL = ' '
113: C/#IF .NOT.NOGETENV
114: *      call CHKENV( 'MVP' )
115: C/#ENDIF
116: C
117: C/#IF PARA(VPP)
118: C
119: C      call CHROUND(3)
120: C
121: *      call GETTOD(TTT0)
122: C/#ENDIF
123: C
124: C/# IF SYSTEM(FACOMVPP)
125: C      call CHROUND(3)
126: C/# ENDIF
127: C
128: C ... common area data initialization (do not use BLOCK DATA)
129: C
130: *      call WDBL
```

src/mvp/main.f

```

131:      call BPROGV
132:      call FLAGIN
133:      call ERINIT
134:      call FRINIT(INP,IPR,10)
135:      call INDROT
136:      call KPINIT
137:      call BLKISO ! photonuc
138: C
139: C ... data initialization for custom version.
140: C
141:      call ZZINIT
142: C
143:      NTASK0 = 0
144:      TSKINF = ' '
145:      MLIMIT = 0
146:      IPREIN = 0
147:      call S1STFN( 'NOFILENAME' )
148: C
149: C-----
150: C Program control information from command line arguments.
151: C-----
152: C
153: C ... ARGNONE : FAT flag indicating no means to get command argument.
154: C
155: C/#IF ARGV
156: C
157: C/# IF ARGV( GETARG2 )
158: *      KJ = IARGC() - 1
159: C/# ELSEIF ARGV( MSF )
160: *      KJ = NARGS() - 1
161: C/# ELSE
162:      KJ = IARGC()
163: C/# ENDIF
164:
165:      do 100 I = 1, KJ
166:
167:          IPREIN = IPREIN + 1
168: C/# IF SYSTEM( CRAY )
169: *          call PXFGETARG(I, CSTRNG, LLLLL, ierr )
170: C/# ELSEIF ARGV( IGETARG )
171: *          LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
172: C/# ELSE
173: C/# IF ARGV( GETARG2 )
174: *          call GETARG(I+1, CSTRNG )
175: C/# ELSEIF ARGV( MSF )
176: C ... LL2 should be integer*2
177: *          call GETARG(I, CSTRNG, LL2 )
178: C/# ELSE
179:          call GETARG( I, CSTRNG )
180: C/# ENDIF
181:          LLLLL = MIN( INDEX( CSTRNG, ' ' ) - 1, LEN( CSTRNG ) )
182:          if ( LLLLL.eq.0 ) LLLLL = LEN( CSTRNG )
183: C/# ENDIF
184:
185: C
186: C ... quit command line check if '@@' is encountered ...
187: C
188:
189:      if ( CSTRNG(1:LLLLL) .eq. '@@' ) goto 105
190:
191:      call PREIN0( IPREIN, CSTRNG(1:LLLLL), NTASK0, TSKINF, MLIMIT )
192:
193: 100 continue
194: C
195: 105 continue

```

```

196: C
197:      if ( KJ.gt.0 ) write(6,(''1''))
198:
199: C ... END OF C/#IF ARGV ...
200: C/#ENDIF
201: C
202: C-----
203: C ... pre-setting of SIGINT event handler that flushes standard
204: C      output.
205: C-----
206: C
207: CCCCCCCC/# IF SYSTEM(HP*)
208: C/#IF FLOATHANDLER( ONSTMT )
209: C
210: C == Floating point error handler specification by "ON" statement
211: C (Specific to HP fortran)
212: C ---- TO PREVENT UNDERFLOW ABORT in trapping mode (+T option)---
213: C
214: *      ON REAL*4 UNDERFLOW CALL UNDERF4
215: *      ON REAL*8 UNDERFLOW CALL UNDERF8
216: C/#ENDIF
217:
218: C/#IF UNIX
219: C
220: C ... some implementation of MPI uses SIGUSR, SIGINT
221: C      so MVP/GMVP do not use it for them
222: C
223: C/#IF .NOT.PARA(MPI)
224:      call FSIGSET( )
225:      call FSIGUSR( )
226: C/#ENDIF
227: C/#ENDIF
228: C
229:      call AAALOC
230:      call ACALOC
231: C
232: C-----
233: C Processing Starts
234: C-----
235: C
236: C/#IF PARA
237: *      call CENTER( NTASK0, TSKINF, MLIMIT )
238: C/#ELSE
239: C/# IF .NOT.DYNAMIC
240: *      call CENTER
241: C/# ELSE
242:      call CENTER( MLIMIT )
243: C/# ENDIF
244: C/#ENDIF
245: C
246: C
247: C-----
248: C timing information output for VPP
249: C-----
250: C/#IF PARA(VPP)
251:      call GETTOD( TTT1 )
252:      TMTOTL = TTT1 - TTT0
253:      write(6,*) ' '
254:      write(6,7000) '##### VPP exec-time information'
255:      write(6,7000) '#####'
256:      write(6,7020) '##### time(intro) :', TMINTR/1.0D06,
257:      &      '[sec]'
258:      write(6,7020) '##### time(para-region) :', TMPARA/1.0D06,
259:      &      '[sec]'
260: C

```

src/mvp/main.f

```
261:      write(6,7000) '##### time(restart-input)'
262:      do 110 I = 1, MPE
263:          write(6,7040) '#####', 'PE', I, ':', TMINPP(I) /1.0D06,
264:          & '[sec]'
265:      110 continue
266: C
267:      write(6,7000) '##### time(action)'
268:      do 120 I = 1, MPE
269:          write(6,7040) '#####', 'PE', I, ':', TMACT(I) /1.0D06,
270:          & '[sec]'
271:      120 continue
272: C
273:      write(6,7000) '##### time(PE-sum)'
274:      do 130 I = 1, MPE
275:          write(6,7040) '#####', 'PE', I, ':', TMSUM(I) /1.0D06,
276:          & '[sec]'
277:      130 continue
278: C
279:      write(6,7000) '##### time(file-output)'
280:      do 140 I = 1, MPE
281:          write(6,7040) '#####', 'PE', I, ':', TMOUPT(I) /1.0D06,
282:          & '[sec]'
283:      140 continue
284: C
285:      write(6,7020) '##### time(gather print):', TMPRINT/1.0D06,
286:      & '[sec]'
287:      write(6,7020) '##### time(cadenz)      :', TMCNDZ/1.0D06,
288:      & '[sec]'
289:      write(6,7020) '##### time(total)       :', TMTOTL/1.0D06,
290:      & '[sec]'
291:
292:      7000 format(A)
293:      7020 format(A,1X,F8.3,1X,A)
294:      7040 format(A,15X,A,I2,2X,A,1X,F8.3,1X,A)
295:
296: C/#ENDIF
297: C
298:      stop
299:      end
300: C
301: C === Floating point error handler specified by "ON" statement
302: C      (Specific to HP fortran)
303: C
304: C
305: C ... underflow handling ...
306: C
307:      subroutine UNDERF4( R4 )
308:      real R4
309:      R4      = 0.0
310:      return
311:      end
312: C
313: C
314:      subroutine UNDERF8( R8 )
315:      real*8 R8
316:      R8      = 0.0D0
317:      return
318:      end
```

src/mvp/mmchk.f

```
1: C
2:      SUBROUTINE MMCHK( IFLAG )
3: CC  ---- CHECK FOR INVALID ASSIGNMENT ----
4: *    PARAMETER (MAXCHK = 1024 , AM1 = -0.987E21, AM2= 1.23456E-30 )
5: *    COMMON /CHKMEM/ NCHK, LCHK(MAXCHK)
6: *    COMMON /CHKMM2/ VCHK(MAXCHK)
7: *    CHARACTER*8 VCHK
8: *    -INCLUDE $ARRAY
9: *    -INCLUDE $FLAGS
10: *    IF( JMCHK .EQ. 0 ) RETURN
11: C
12: *    IFLAG = 0
13: *    DO 100 I=1,NCHK
14: *        IF( A(LCHK(I)).NE.AM1 .OR. A(LCHK(I)+1).NE.AM2 ) THEN
15: *            WRITE(6,*) 'XXX MEMORY ERROR: LCHK(' ,I,') ',LCHK(I),
16: *            @      / < ',VCHK(I), '> ',
17: *            @      A(LCHK(I)),A(LCHK(I)+1)
18: *            IFLAG = 1
19: *        ENDF
20: * 100 CONTINUE
21:      RETURN
22:      END
```

src/mvp/mntcpu.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine MNTCPU
3: C=====
4: C PURPOSE : CHECK CPU TIME TO CALL MONITOR ROUINTES (--> VMNTR0 )
5: C CALLED IN: ACTION
6: C CALLS : VMNTR0,VMNTR1,VMNTR2,VMNTRC,VMNTRG
7: C
8: C=====
9:   include '../shared/INC/_IOUNIT'
10:  include 'INC/_NMON'
11:
12: C/#IF SYSTEM(ACOS SX*)
13: *   real*8 T0,T1,T3,T4
14: C/#ENDIF
15: C
16:   parameter( NN = 100 )
17: C
18: C-----
19:   write(IPR,*) '== CPU time monitoring overhead measurement'
20: C
21:   T00 = 0.0
22:   do 100 I = 1, NN
23: C/#IF SYSTEM(FACOM*)
24: *   call CLOCK(T0,0,1)
25: *   call CLOCK(T1,0,1)
26: C/#ELSEIF SYSTEM(ACOS SX*)
27: *   call CLOCK(T0)
28: *   call CLOCK(T1)
29: C/#ELSEIF SYSTEM(CRAY*)
30: *   call SECOND(T0)
31: *   call SECOND(T1)
32: CC/#ELSEIF SYSTEM(HP* SUN* DECOSF* MIPS* NECEWS* SGI*)
33: C   call cputime(t0)
34: C   call cputime(t1)
35: C/#ELSE
36: Ccc   t0 = 0.0
37: Ccc   t1 = 0.0
38: *   call cputm(t0)
39: *   call cputm(t1)
40: C/#ENDIF
41:   T00 = T00 + T1 - T0
42:   100 continue
43:   T00 = T00/NN
44:   write(IPR,'(1X,A,1P,E11.4,A)') ' (MNTCPU) ... T00 = ', T00,
45:   & ' sec'
46: C-----
47:   do 150 NS = 1, NMON
48: C-----
49: C
50:   do 120 NU = 1, NMON
51: C
52:   if ( NU.ne.NS ) then
53:   do 110 I = 1, NN
54:   call VMNTR0( NS, NU )
55:   call VMNTR2( NS, NU )
56:   110 continue
57:   call VMNTRT( T01, NS, NU )
58:   T2 = T01/NN
59:   call VMNTRG( T2, NS, NU )
60: CCC   write(IPR,'(1X,a,1p,e11.4,a,i3,i3,a)')
61: CCC & ' (MNTCPU) SCPU = ', T2, ' SEC (', NS, NU, ' )'
62:   end if
63:   120 continue
64: C
65:   call VMNTRC

```

```

66:   do 130 I = 1, NN
67:   call VMNTR0( NS, IPR )
68:   call VMNTR2( NS )
69:   130 continue
70:   call VMNTRT( T01, NS, NS )
71:   T2 = T01/NN
72:   call VMNTRG( T2, NS, NS )
73: C
74:   T02 = 0.0
75:   do 140 I = 1, NN
76: C/#IF SYSTEM(FACOM*)
77: *   call CLOCK(T3,0,1)
78: C/#ELSEIF SYSTEM(ACOS SX*)
79: *   call CLOCK(T3)
80: C/#ELSEIF SYSTEM(CRAY*)
81: *   call SECOND(T3)
82: CC/#ELSEIF SYSTEM(HP* SUN* DECOSF* MIPS* NECEWS* SGI*)
83: C   call cputime(t3)
84: C/#ELSE
85: *   call CPUTM(T3)
86: C/#ENDIF
87:   call VMNTR1( NS, 9999 )
88: C/#IF SYSTEM(FACOM*)
89: *   call CLOCK(T4,0,1)
90: C/#ELSEIF SYSTEM(ACOS SX*)
91: *   call CLOCK(T4)
92: C/#ELSEIF SYSTEM(CRAY*)
93: *   call SECOND(T4)
94: CC/#ELSEIF SYSTEM(HP* SUN* DECOSF* MIPS* NECEWS* SGI*)
95: C   call cputime(t4)
96: C/#ELSE
97: *   call CPUTM(T4)
98: C/#ENDIF
99:   T02 = T02 + T4 - T3
100:   140 continue
101:   T02 = T02/NN - T00
102:   call VMNTRH( T02, NS )
103: C
104:   write(IPR,'(1X,a,1p,e11.4,a,i3,a)') ' (MNTCPU) ... SCPU2 = ',
105:   & T02, ' SEC (SUBROUTINE# ', NS, ' )'
106: C
107:   150 continue
108: C
109:   call VMNTRC
110:   return
111: end

```

src/mvp/monpnwinp.f

```

1:      subroutine MONPNWINP(MSG, A, IA, LIMIT, LAST, JDEBG, NREG, NUCPN,
2:      &                      LMONPNW, NMONPNW, NMTPN, NCIDPN, MNUCPN, LPIDPN,
3:      &                      JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM,
4:      &                      ISUSP, KSUZN, IRGSP, NEST, NSPACE, NSUZON,
5:      &                      NZONE, TNAMS, NNAMES, ITRNM, NTREG, IPTRG,
6:      &                      LPTRG, NUCPNI, KLBPN1, KLBPN2, KR, NRRR0 )
7: C=====
8: C  purpose:  input array data for monitoring a particle weight from a
9: C            specified reaction type (mt) by region and nuclide in
10: C           photonuclear reaction.
11: C  called in:  INTRO
12: C  callis:
13: C=====
14:      include '../shared/INC/_LNAM'
15:      include 'INC/_KPIDS'
16: C
17:      real A(LIMIT)
18:      integer IA(LIMIT), JDEBG(*)
19: C
20: C      .... data for photonuclear ....
21:      integer MNUCPN(*), LPIDPN(*)
22:      integer KLBPN1(NUCPNI,*), KLBPN2(NUCPNI,NMTPN,*)
23:      character NCIDPN(*)*16
24: C
25: C      .... data for region name matching ....
26:      integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ),
27:      &          ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
28:      &          KSUZN(NINPZ,2), IRGSP(2,NREG)
29:      integer ITRNM(NTREG), IPTRG(NTREG+1), LPTRG(*), KR(NRRR0)
30:      character TNAMS(NNAMES)*(LNAM)
31: C
32: C      .... local variable ....
33:      integer KN(200), KM(135)
34:      character LINE*72, CHAR*4, NM*5
35: C
36: C-----
37: C
38: C      .... check and initialize ....
39: C      if ( NUCPN.le.0 ) go to 901
40: C      if ( NREG.le.0 ) go to 903
41: C      if ( NMTPN.le.0.or.NMTPN.gt.135 ) go to 925
42: C      NMONPNW = 0
43: C      JSKIP = 0
44: C      MO = 0
45: C      NN = 0
46: C      NN1 = 12000
47: C      NN1 = max(4*NUCPN*NREG, NN1)
48: C      if ( LMONPNW.eq.0 )
49: C      & call KEPV( IA(1), 'MONPNWGT', LMONPNW, NN1, 'I4', LAST,0,JDEBG)
50: C
51: C=====
52: C  region-wise form: MONPNWGT( !region( ( nuclide, reaction ) ... ) ... )
53: C                      MONPNWGT( !region( nuclide, reaction ) ... )
54: C                      MONPNWGT( !reg1+!reg2+...( nucl1+nucl2+..., react1+react2+
55: C                      MONPNWGT( !region( nuclide ) ... )
56: C                      MONPNWGT( !region( reaction ) ... )
57: C                      MONPNWGT( !region ... )
58: C=====
59: C  MONPNWGT(i,j) array ...
60: C      i = 1 : given region number
61: C      2 : given nuclide number of photonuclear
62: C      3 : given reaction number (MT) of photonuclear
63: C      4 : monitoring number for a set of region, nuclide and reaction
64: C          (sequential number started from 1)
65: C      j      : number of given sets for monitoring
66: C=====
67: C
68:      100 call FPROBE( IPP, ' (', LINE )
69:      if ( IPP.eq.0 ) then
70:      call GTLINE( IEND )
71:      if ( IEND.ne.0 ) go to 905
72:      go to 100
73:      end if
74:      CHAR=' '
75:      CHAR(1:2) = '()'
76:      if ( LINE(1:1).ne.'!' ) go to 937
77:      IITERM = 0
78: C
79: C      >>> !region-name( ... )
80: C
81:      110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
82: C
83:      if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.' ' ) then
84:      IITERM = 1
85:      if ( NLEN.gt.0 ) go to 115
86:      go to 500
87:      end if
88:      if ( NLEN.eq.0 ) then
89:      if ( IEND.eq.0 ) go to 110
90:      go to 907
91:      end if
92:      115 NR = 0
93: C      .... check the multiple regions ....
94:      NL1 = 1
95:      120 NL2 = NLEN
96:      IKR = 0
97:      do I = NL1, NL2
98:      if ( LINE(I:I).eq.'+' ) then
99:      NL2 = I - 1
100:      IKR = I + 1
101:      if ( IKR.gt.NLEN ) IKR = 0
102:      go to 125
103:      end if
104:      end do
105:      125 IK = INDEX(LINE(NL1:NL2),'@')
106:      if ( IK.gt.0 ) then
107: C
108: C      .... @NAME without wildcard character ....
109: C      if ( INDEX(LINE(NL1:NL2),'?').eq.0 .and.
110: C      & INDEX(LINE(NL1:NL2),'*').eq.0 ) then
111: C      call CBSINC( TNAMS, NNAMES, LINE(IK:NL2), IPOS )
112: C      if ( IPOS.eq.0 ) then
113: C      write(MSG,910) LINE(IK:NL2)
114: C      call CNTERR( 'FATAL' )
115: CM2016
116:      if ( IKR.eq.0 ) then
117:      call DMREAD( ' ', IDM1, N1, IIER )
118:      if ( IIER.ne.0 ) go to 911
119:      go to 110
120:      else
121:      NL1 = IKR
122:      go to 120
123:      end if
124:      end if
125:      do I = 1, NTREG

```

src/mvp/monpnwinp.f

```

126:         if ( ITRNM(I).lt.0.and.IPOS.eq.abs(ITRNM(I)) ) go to 130
127:     end do
128:     go to 913
129: 130     IRD      = I
130: CM2016     if ( JSKIP.ne.0 ) then
131: CM2016         call DMREAD( ' ', IDM1, N1, IIER )
132: CM2016         if ( IIER.ne.0 ) go to 911
133: CM2016         go to 110
134: CM2016     end if
135:         do K = IPTRG(IRD), IPTRG(IRD+1)-1
136:             NR      = NR + 1
137:             KR(NR) = LPTRG(K)
138:         end do
139: C
140: C         ..... @NAME with wildCard character .....
141:         else
142:             do J = 1, NIREG
143:                 if ( ITRNM(J).lt.0 ) then
144: CM2016                     if ( IMATCH(LINE(IK:NLEN),TNAMS(abs(ITRNM(J)))) .ne.0 )
145:                         if ( IMATCH(LINE(IK:NL2),TNAMS(abs(ITRNM(J)))) .ne.0 )
146: &                             then
147:                             do 140 I = IPTRG(J), IPTRG(J+1)-1
148:                                 do K = 1, NR
149:                                     if ( KR(K).eq.LPTRG(I) ) go to 140
150:                                 end do
151:                                 NR      = NR + 1
152:                                 KR(NR) = LPTRG(I)
153: 140                             continue
154:                             end if
155:                         end if
156:                     end do
157:                 end if
158: C
159: C         ..... regular region name .....
160:         else if ( IK.eq.0 ) then
161:             IBEGIN = 0
162:             JEND    = 0
163: 150         continue
164:         call RGFIND( MSG, LINE(NL1:NL2), IBEGIN, JEND, IREG, IERR,
165: &                 JTLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISENM, ISUSP,
166: &                 KSUZ, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
167: &                 NNAMES )
168:         if ( IERR.ne.0 ) then
169:             write(MSG,916) LINE(NL1:NL2)
170:             call CNTERR( 'FATAL' )
171: CM2016
172:             if ( IKR.eq.0 ) then
173:                 call DMREAD( ' ', IDM1, N1, IIER )
174:                 if ( IIER.ne.0 ) go to 911
175:                 go to 110
176:             else
177:                 NL1      = IKR
178:                 go to 120
179:             end if
180:         end if
181:         if ( IREG.ne.0 ) then
182:             do K = 1, NR
183:                 if ( KR(K).eq.IREG ) go to 150
184:             end do
185:             NR      = NR + 1
186:             KR(NR) = IREG
187:         end if
188:         if ( JEND.eq.0 ) go to 150
189:     end if
190:     if ( IKR.gt.0 ) then

```

```

191:         NL1      = IKR
192:         go to 120
193:     end if
194: C
195:     if ( ITERM.eq.0.or.IITEM.eq.1 ) then      ! only region name
196:         MO      = MO + 1                      ! monitoring number
197:         do K = 1, NR
198:             do J = 1, NUCPN
199:                 NMT2G = KLBPN1(J,4)           ! number of reactions
200:             producing secondary products
201:             if ( NN+NMT2G.gt.NN1 ) go to 931
202:             do I = 1, NMT2G
203:                 NN      = NN + 1
204:                 I1      = 4 * (NN - 1) - 1
205:                 IA(LMONPNW+I1+1) = KR(K)      ! region number
206:                 IA(LMONPNW+I1+2) = J          ! photonuclear nuclide
207:                 IA(LMONPNW+I1+3) = KLBPN2(J,I,9) ! photonuclear reactio
208:                 IA(LMONPNW+I1+4) = MO          ! monitoring number
209:                 if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1 ! deletion of d
210:             end do
211:         end do
212:         if ( IITEM.eq.1 ) go to 500
213:         go to 110
214:     end if
215: C
216: C         ..... read the top of data block .....
217: 160 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
218:         if ( IEND.ne.0 ) go to 917
219:         if ( NLEN.le.0.and.ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.'') )
220: &         go to 110
221: C
222: C         ..... many data blocks : ((nuclide,reaction)...) .....
223:         if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.'(' ) then
224: C             ..... read the nuclide name .....
225:             call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
226:             if ( IEND.ne.0 ) go to 919
227:             if ( NLEN.le.0 ) go to 929
228:             KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
229:             if ( KNAME0.le.0 ) then                ! undefined nuclide
230:                 do I = 1, NUCPN
231:                     KN(I) = I
232:                 end do
233:                 IINUCPN = NUCPN
234:                 go to 200
235:             end if
236:             IINUCPN = 0
237:             NL1      = 1
238: 170         continue                                ! return here, if multiple nucl
239:         if ( ITERM.ne.0 ) then
240:             NL2      = NLEN
241:             IKN      = 0
242:             do I = NL1, NL2
243:                 if ( LINE(I:I).eq.'+' ) then
244:                     NL2      = I - 1
245:                     IKN      = I + 1
246:                     go to 175
247:                 end if
248:             end do
249:             175     NL0      = NL2 - NL1 + 1
250:                     if ( NL0.eq.5 ) then
251:                         NM      = LINE(NL1:NL2)

```


src/mvp/monpnwinp.f

```

251:      else if ( NL0.eq.3 ) then
252:          NM      = LINE(NL1:NL2) // ' '
253:      else if ( NL0.eq.2 ) then
254:          NM      = LINE(NL1:NL2) // ' '
255:      else if ( NL0.eq.1 ) then
256:          NM      = LINE(NL1:NL2) // '0 '
257:      else
258:          go to 929
259:      end if
260:      IKW      = 0
261:      if ( NM(3:5).eq.'000' ) then
262:          IKW = 1
263:      else if ( NM(3:3).eq.'1' ) then
264:          IKW = 1
265:      else if ( NM(2:2).eq.'*' ) then
266:          IKW = 1
267:          NM(2:2) = '0'
268:      else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
269:          IKW = 1
270:      end if
271:      if ( IKW.eq.0 ) then
272:          do I = 1, NUCPN
273:              if ( NM(1:NL0).eq.NCIDPN(I)(1:NL0) ) go to 180
274:          end do
275:          go to 923
276:      180      IINUCPN = IINUCPN + 1
277:              KN(IINUCPN) = I
278:      else if ( IKW.eq.1 ) then
279:          II      = 0
280:      185      do I = II+1, NUCPN
281:                  if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 190
282:              end do
283:              if ( IINUCPN.le.0 ) go to 923
284:              go to 195
285:      190      IINUCPN = IINUCPN + 1
286:              KN(IINUCPN) = I
287:              if ( I.lt.NUCPN ) then
288:                  II      = I
289:                  go to 185
290:              end if
291:      195      continue
292:      end if
293:      if ( IKN.gt.0 ) then
294:          NL1      = IKN
295:          go to 170
296:      end if
297:      if ( ITERM.ne.0 ) then
298:          if ( IINUCPN.le.0 ) go to 935
299:          go to 220
300:      end if
301:      C      ..... read the reaction type (mt) .....
302:      call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
303:      if ( IEND.ne.0 ) go to 919
304:      if ( NLEN.le.0 ) go to 929
305:      200      continue
306:      IIMTPN = 0
307:      NL1      = 1
308:      210      continue
309:      NL2      = NLEN
310:      IKR      = 0
311:      do I = NL1, NL2
312:          if ( LINE(I:I).eq.'+' ) then
313:              NL2      = I - 1
314:              IKR      = I + 1
315:          go to 215
316:      end if
317:      end do
318:      215      IIMTPN = IIMTPN + 1
319:      read(LINE(NL1:NL2),*) MTPN0
320:      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
321:      KM(IIMTPN) = MTPN0
322:      if ( IKR.gt.0 ) then
323:          NL1      = IKR
324:          go to 210
325:      end if
326:      C      ..... store the given data .....
327:      MO      = MO + 1
328:      do K = 1, NR
329:          do J = 1, IINUCPN
330:              do I = 1, IIMTPN
331:                  if ( KNAME0.le.0 ) then
332:                      do I2 = 1, KLBPNI(KN(J),4)
333:                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
334:                      end do
335:                      go to 218
336:      217      continue
337:                  end if
338:                  NN      = NN + 1
339:                  I1      = 4 * (NN - 1) - 1
340:                  IA(LMONPNW+I1+1) = KR(K)
341:                  IA(LMONPNW+I1+2) = KN(J)
342:                  IA(LMONPNW+I1+3) = KM(I)
343:                  IA(LMONPNW+I1+4) = MO
344:      218      continue
345:              end do
346:          end do
347:          go to 225
348:      220      MO      = MO + 1
349:          do K = 1, NR
350:              do J = 1, IINUCPN
351:                  NMT2G      = KLBPN1(KN(J),4)
352:                  producing secondary products
353:                  do I = 1, NMT2G
354:                      NN      = NN + 1
355:                      I1      = 4 * (NN - 1) - 1
356:                      IA(LMONPNW+I1+1) = KR(K)
357:                      IA(LMONPNW+I1+2) = KN(J)
358:                      IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
359:                      IA(LMONPNW+I1+4) = MO
360:                      if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
361:                  end do
362:              end do
363:          end do
364:      225      continue
365:      C      ..... single data block : (nuclide,reaction) .....
366:      C      else
367:      C      ..... check and read the nuclide name .....
368:      C      if ( NLEN.le.0 ) go to 929
369:      C      KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
370:      C      if ( KNAME0.gt.0 ) then
371:      C          IINUCPN = 0
372:      C          NL1      = 1
373:      C          NL2      = NLEN
374:      C          IKR      = 0
375:      C          do I = NL1, NL2
376:      C              if ( LINE(I:I).eq.'+' ) then
377:      C                  NL2      = I - 1
378:      C                  IKR      = I + 1
379:      C              go to 215
380:      C          end if
381:      C          end do
382:      C          215      IIMTPN = IIMTPN + 1
383:      C          read(LINE(NL1:NL2),*) MTPN0
384:      C          if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
385:      C          KM(IIMTPN) = MTPN0
386:      C          if ( IKR.gt.0 ) then
387:      C              NL1      = IKR
388:      C              go to 210
389:      C          end if
390:      C          ..... store the given data .....
391:      C          MO      = MO + 1
392:      C          do K = 1, NR
393:      C              do J = 1, IINUCPN
394:      C                  do I = 1, IIMTPN
395:      C                      if ( KNAME0.le.0 ) then
396:      C                          do I2 = 1, KLBPNI(KN(J),4)
397:      C                              if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
398:      C                          end do
399:      C                          go to 218
400:      C                      217      continue
401:      C                      end if
402:      C                      NN      = NN + 1
403:      C                      I1      = 4 * (NN - 1) - 1
404:      C                      IA(LMONPNW+I1+1) = KR(K)
405:      C                      IA(LMONPNW+I1+2) = KN(J)
406:      C                      IA(LMONPNW+I1+3) = KM(I)
407:      C                      IA(LMONPNW+I1+4) = MO
408:      C                      218      continue
409:      C                  end do
410:      C              end do
411:      C              go to 225
412:      C          220      MO      = MO + 1
413:      C              do K = 1, NR
414:      C                  do J = 1, IINUCPN
415:      C                      NMT2G      = KLBPN1(KN(J),4)
416:      C                      producing secondary products
417:      C                      do I = 1, NMT2G
418:      C                          NN      = NN + 1
419:      C                          I1      = 4 * (NN - 1) - 1
420:      C                          IA(LMONPNW+I1+1) = KR(K)
421:      C                          IA(LMONPNW+I1+2) = KN(J)
422:      C                          IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
423:      C                          IA(LMONPNW+I1+4) = MO
424:      C                          if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
425:      C                      end do
426:      C                  end do
427:      C              end do
428:      C          225      continue
429:      C          ..... single data block : (nuclide,reaction) .....
430:      C          else
431:      C          ..... check and read the nuclide name .....
432:      C          if ( NLEN.le.0 ) go to 929
433:      C          KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
434:      C          if ( KNAME0.gt.0 ) then
435:      C              IINUCPN = 0
436:      C              NL1      = 1
437:      C              NL2      = NLEN
438:      C              IKR      = 0
439:      C              do I = NL1, NL2
440:      C                  if ( LINE(I:I).eq.'+' ) then
441:      C                      NL2      = I - 1
442:      C                      IKR      = I + 1
443:      C                  go to 215
444:      C              end if
445:      C              end do
446:      C              215      IIMTPN = IIMTPN + 1
447:      C              read(LINE(NL1:NL2),*) MTPN0
448:      C              if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
449:      C              KM(IIMTPN) = MTPN0
450:      C              if ( IKR.gt.0 ) then
451:      C                  NL1      = IKR
452:      C                  go to 210
453:      C              end if
454:      C              ..... store the given data .....
455:      C              MO      = MO + 1
456:      C              do K = 1, NR
457:      C                  do J = 1, IINUCPN
458:      C                      do I = 1, IIMTPN
459:      C                          if ( KNAME0.le.0 ) then
460:      C                              do I2 = 1, KLBPNI(KN(J),4)
461:      C                                  if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
462:      C                              end do
463:      C                              go to 218
464:      C                          217      continue
465:      C                          end if
466:      C                          NN      = NN + 1
467:      C                          I1      = 4 * (NN - 1) - 1
468:      C                          IA(LMONPNW+I1+1) = KR(K)
469:      C                          IA(LMONPNW+I1+2) = KN(J)
470:      C                          IA(LMONPNW+I1+3) = KM(I)
471:      C                          IA(LMONPNW+I1+4) = MO
472:      C                          218      continue
473:      C                      end do
474:      C                  end do
475:      C                  go to 225
476:      C              220      MO      = MO + 1
477:      C                  do K = 1, NR
478:      C                      do J = 1, IINUCPN
479:      C                          NMT2G      = KLBPN1(KN(J),4)
480:      C                          producing secondary products
481:      C                          do I = 1, NMT2G
482:      C                              NN      = NN + 1
483:      C                              I1      = 4 * (NN - 1) - 1
484:      C                              IA(LMONPNW+I1+1) = KR(K)
485:      C                              IA(LMONPNW+I1+2) = KN(J)
486:      C                              IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
487:      C                              IA(LMONPNW+I1+4) = MO
488:      C                              if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
489:      C                          end do
490:      C                      end do
491:      C                  end do
492:      C              225      continue
493:      C              ..... single data block : (nuclide,reaction) .....
494:      C              else
495:      C              ..... check and read the nuclide name .....
496:      C              if ( NLEN.le.0 ) go to 929
497:      C              KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
498:      C              if ( KNAME0.gt.0 ) then
499:      C                  IINUCPN = 0
500:      C                  NL1      = 1
501:      C                  NL2      = NLEN
502:      C                  IKR      = 0
503:      C                  do I = NL1, NL2
504:      C                      if ( LINE(I:I).eq.'+' ) then
505:      C                          NL2      = I - 1
506:      C                          IKR      = I + 1
507:      C                      go to 215
508:      C                  end if
509:      C                  end do
510:      C                  215      IIMTPN = IIMTPN + 1
511:      C                  read(LINE(NL1:NL2),*) MTPN0
512:      C                  if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
513:      C                  KM(IIMTPN) = MTPN0
514:      C                  if ( IKR.gt.0 ) then
515:      C                      NL1      = IKR
516:      C                      go to 210
517:      C                  end if
518:      C                  ..... store the given data .....
519:      C                  MO      = MO + 1
520:      C                  do K = 1, NR
521:      C                      do J = 1, IINUCPN
522:      C                          do I = 1, IIMTPN
523:      C                              if ( KNAME0.le.0 ) then
524:      C                                  do I2 = 1, KLBPNI(KN(J),4)
525:      C                                      if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
526:      C                                  end do
527:      C                                  go to 218
528:      C                              217      continue
529:      C                              end if
530:      C                              NN      = NN + 1
531:      C                              I1      = 4 * (NN - 1) - 1
532:      C                              IA(LMONPNW+I1+1) = KR(K)
533:      C                              IA(LMONPNW+I1+2) = KN(J)
534:      C                              IA(LMONPNW+I1+3) = KM(I)
535:      C                              IA(LMONPNW+I1+4) = MO
536:      C                              218      continue
537:      C                          end do
538:      C                      end do
539:      C                      go to 225
540:      C                  220      MO      = MO + 1
541:      C                      do K = 1, NR
542:      C                          do J = 1, IINUCPN
543:      C                              NMT2G      = KLBPN1(KN(J),4)
544:      C                              producing secondary products
545:      C                              do I = 1, NMT2G
546:      C                                  NN      = NN + 1
547:      C                                  I1      = 4 * (NN - 1) - 1
548:      C                                  IA(LMONPNW+I1+1) = KR(K)
549:      C                                  IA(LMONPNW+I1+2) = KN(J)
550:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
551:      C                                  IA(LMONPNW+I1+4) = MO
552:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
553:      C                              end do
554:      C                          end do
555:      C                      end do
556:      C                  225      continue
557:      C                  ..... single data block : (nuclide,reaction) .....
558:      C                  else
559:      C                  ..... check and read the nuclide name .....
560:      C                  if ( NLEN.le.0 ) go to 929
561:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
562:      C                  if ( KNAME0.gt.0 ) then
563:      C                      IINUCPN = 0
564:      C                      NL1      = 1
565:      C                      NL2      = NLEN
566:      C                      IKR      = 0
567:      C                      do I = NL1, NL2
568:      C                          if ( LINE(I:I).eq.'+' ) then
569:      C                              NL2      = I - 1
570:      C                              IKR      = I + 1
571:      C                          go to 215
572:      C                      end if
573:      C                      end do
574:      C                      215      IIMTPN = IIMTPN + 1
575:      C                      read(LINE(NL1:NL2),*) MTPN0
576:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
577:      C                      KM(IIMTPN) = MTPN0
578:      C                      if ( IKR.gt.0 ) then
579:      C                          NL1      = IKR
580:      C                          go to 210
581:      C                      end if
582:      C                      ..... store the given data .....
583:      C                      MO      = MO + 1
584:      C                      do K = 1, NR
585:      C                          do J = 1, IINUCPN
586:      C                              do I = 1, IIMTPN
587:      C                                  if ( KNAME0.le.0 ) then
588:      C                                      do I2 = 1, KLBPNI(KN(J),4)
589:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
590:      C                                      end do
591:      C                                      go to 218
592:      C                                  217      continue
593:      C                                  end if
594:      C                                  NN      = NN + 1
595:      C                                  I1      = 4 * (NN - 1) - 1
596:      C                                  IA(LMONPNW+I1+1) = KR(K)
597:      C                                  IA(LMONPNW+I1+2) = KN(J)
598:      C                                  IA(LMONPNW+I1+3) = KM(I)
599:      C                                  IA(LMONPNW+I1+4) = MO
600:      C                                  218      continue
601:      C                              end do
602:      C                          end do
603:      C                      go to 225
604:      C                  220      MO      = MO + 1
605:      C                      do K = 1, NR
606:      C                          do J = 1, IINUCPN
607:      C                              NMT2G      = KLBPN1(KN(J),4)
608:      C                              producing secondary products
609:      C                              do I = 1, NMT2G
610:      C                                  NN      = NN + 1
611:      C                                  I1      = 4 * (NN - 1) - 1
612:      C                                  IA(LMONPNW+I1+1) = KR(K)
613:      C                                  IA(LMONPNW+I1+2) = KN(J)
614:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
615:      C                                  IA(LMONPNW+I1+4) = MO
616:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
617:      C                              end do
618:      C                          end do
619:      C                      end do
620:      C                  225      continue
621:      C                  ..... single data block : (nuclide,reaction) .....
622:      C                  else
623:      C                  ..... check and read the nuclide name .....
624:      C                  if ( NLEN.le.0 ) go to 929
625:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
626:      C                  if ( KNAME0.gt.0 ) then
627:      C                      IINUCPN = 0
628:      C                      NL1      = 1
629:      C                      NL2      = NLEN
630:      C                      IKR      = 0
631:      C                      do I = NL1, NL2
632:      C                          if ( LINE(I:I).eq.'+' ) then
633:      C                              NL2      = I - 1
634:      C                              IKR      = I + 1
635:      C                          go to 215
636:      C                      end if
637:      C                      end do
638:      C                      215      IIMTPN = IIMTPN + 1
639:      C                      read(LINE(NL1:NL2),*) MTPN0
640:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
641:      C                      KM(IIMTPN) = MTPN0
642:      C                      if ( IKR.gt.0 ) then
643:      C                          NL1      = IKR
644:      C                          go to 210
645:      C                      end if
646:      C                      ..... store the given data .....
647:      C                      MO      = MO + 1
648:      C                      do K = 1, NR
649:      C                          do J = 1, IINUCPN
650:      C                              do I = 1, IIMTPN
651:      C                                  if ( KNAME0.le.0 ) then
652:      C                                      do I2 = 1, KLBPNI(KN(J),4)
653:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
654:      C                                      end do
655:      C                                      go to 218
656:      C                                  217      continue
657:      C                                  end if
658:      C                                  NN      = NN + 1
659:      C                                  I1      = 4 * (NN - 1) - 1
660:      C                                  IA(LMONPNW+I1+1) = KR(K)
661:      C                                  IA(LMONPNW+I1+2) = KN(J)
662:      C                                  IA(LMONPNW+I1+3) = KM(I)
663:      C                                  IA(LMONPNW+I1+4) = MO
664:      C                                  218      continue
665:      C                              end do
666:      C                          end do
667:      C                      go to 225
668:      C                  220      MO      = MO + 1
669:      C                      do K = 1, NR
670:      C                          do J = 1, IINUCPN
671:      C                              NMT2G      = KLBPN1(KN(J),4)
672:      C                              producing secondary products
673:      C                              do I = 1, NMT2G
674:      C                                  NN      = NN + 1
675:      C                                  I1      = 4 * (NN - 1) - 1
676:      C                                  IA(LMONPNW+I1+1) = KR(K)
677:      C                                  IA(LMONPNW+I1+2) = KN(J)
678:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
679:      C                                  IA(LMONPNW+I1+4) = MO
680:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
681:      C                              end do
682:      C                          end do
683:      C                      end do
684:      C                  225      continue
685:      C                  ..... single data block : (nuclide,reaction) .....
686:      C                  else
687:      C                  ..... check and read the nuclide name .....
688:      C                  if ( NLEN.le.0 ) go to 929
689:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
690:      C                  if ( KNAME0.gt.0 ) then
691:      C                      IINUCPN = 0
692:      C                      NL1      = 1
693:      C                      NL2      = NLEN
694:      C                      IKR      = 0
695:      C                      do I = NL1, NL2
696:      C                          if ( LINE(I:I).eq.'+' ) then
697:      C                              NL2      = I - 1
698:      C                              IKR      = I + 1
699:      C                          go to 215
700:      C                      end if
701:      C                      end do
702:      C                      215      IIMTPN = IIMTPN + 1
703:      C                      read(LINE(NL1:NL2),*) MTPN0
704:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
705:      C                      KM(IIMTPN) = MTPN0
706:      C                      if ( IKR.gt.0 ) then
707:      C                          NL1      = IKR
708:      C                          go to 210
709:      C                      end if
710:      C                      ..... store the given data .....
711:      C                      MO      = MO + 1
712:      C                      do K = 1, NR
713:      C                          do J = 1, IINUCPN
714:      C                              do I = 1, IIMTPN
715:      C                                  if ( KNAME0.le.0 ) then
716:      C                                      do I2 = 1, KLBPNI(KN(J),4)
717:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
718:      C                                      end do
719:      C                                      go to 218
720:      C                                  217      continue
721:      C                                  end if
722:      C                                  NN      = NN + 1
723:      C                                  I1      = 4 * (NN - 1) - 1
724:      C                                  IA(LMONPNW+I1+1) = KR(K)
725:      C                                  IA(LMONPNW+I1+2) = KN(J)
726:      C                                  IA(LMONPNW+I1+3) = KM(I)
727:      C                                  IA(LMONPNW+I1+4) = MO
728:      C                                  218      continue
729:      C                              end do
730:      C                          end do
731:      C                      go to 225
732:      C                  220      MO      = MO + 1
733:      C                      do K = 1, NR
734:      C                          do J = 1, IINUCPN
735:      C                              NMT2G      = KLBPN1(KN(J),4)
736:      C                              producing secondary products
737:      C                              do I = 1, NMT2G
738:      C                                  NN      = NN + 1
739:      C                                  I1      = 4 * (NN - 1) - 1
740:      C                                  IA(LMONPNW+I1+1) = KR(K)
741:      C                                  IA(LMONPNW+I1+2) = KN(J)
742:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
743:      C                                  IA(LMONPNW+I1+4) = MO
744:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
745:      C                              end do
746:      C                          end do
747:      C                      end do
748:      C                  225      continue
749:      C                  ..... single data block : (nuclide,reaction) .....
750:      C                  else
751:      C                  ..... check and read the nuclide name .....
752:      C                  if ( NLEN.le.0 ) go to 929
753:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
754:      C                  if ( KNAME0.gt.0 ) then
755:      C                      IINUCPN = 0
756:      C                      NL1      = 1
757:      C                      NL2      = NLEN
758:      C                      IKR      = 0
759:      C                      do I = NL1, NL2
760:      C                          if ( LINE(I:I).eq.'+' ) then
761:      C                              NL2      = I - 1
762:      C                              IKR      = I + 1
763:      C                          go to 215
764:      C                      end if
765:      C                      end do
766:      C                      215      IIMTPN = IIMTPN + 1
767:      C                      read(LINE(NL1:NL2),*) MTPN0
768:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
769:      C                      KM(IIMTPN) = MTPN0
770:      C                      if ( IKR.gt.0 ) then
771:      C                          NL1      = IKR
772:      C                          go to 210
773:      C                      end if
774:      C                      ..... store the given data .....
775:      C                      MO      = MO + 1
776:      C                      do K = 1, NR
777:      C                          do J = 1, IINUCPN
778:      C                              do I = 1, IIMTPN
779:      C                                  if ( KNAME0.le.0 ) then
780:      C                                      do I2 = 1, KLBPNI(KN(J),4)
781:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
782:      C                                      end do
783:      C                                      go to 218
784:      C                                  217      continue
785:      C                                  end if
786:      C                                  NN      = NN + 1
787:      C                                  I1      = 4 * (NN - 1) - 1
788:      C                                  IA(LMONPNW+I1+1) = KR(K)
789:      C                                  IA(LMONPNW+I1+2) = KN(J)
790:      C                                  IA(LMONPNW+I1+3) = KM(I)
791:      C                                  IA(LMONPNW+I1+4) = MO
792:      C                                  218      continue
793:      C                              end do
794:      C                          end do
795:      C                      go to 225
796:      C                  220      MO      = MO + 1
797:      C                      do K = 1, NR
798:      C                          do J = 1, IINUCPN
799:      C                              NMT2G      = KLBPN1(KN(J),4)
800:      C                              producing secondary products
801:      C                              do I = 1, NMT2G
802:      C                                  NN      = NN + 1
803:      C                                  I1      = 4 * (NN - 1) - 1
804:      C                                  IA(LMONPNW+I1+1) = KR(K)
805:      C                                  IA(LMONPNW+I1+2) = KN(J)
806:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
807:      C                                  IA(LMONPNW+I1+4) = MO
808:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
809:      C                              end do
810:      C                          end do
811:      C                      end do
812:      C                  225      continue
813:      C                  ..... single data block : (nuclide,reaction) .....
814:      C                  else
815:      C                  ..... check and read the nuclide name .....
816:      C                  if ( NLEN.le.0 ) go to 929
817:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
818:      C                  if ( KNAME0.gt.0 ) then
819:      C                      IINUCPN = 0
820:      C                      NL1      = 1
821:      C                      NL2      = NLEN
822:      C                      IKR      = 0
823:      C                      do I = NL1, NL2
824:      C                          if ( LINE(I:I).eq.'+' ) then
825:      C                              NL2      = I - 1
826:      C                              IKR      = I + 1
827:      C                          go to 215
828:      C                      end if
829:      C                      end do
830:      C                      215      IIMTPN = IIMTPN + 1
831:      C                      read(LINE(NL1:NL2),*) MTPN0
832:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
833:      C                      KM(IIMTPN) = MTPN0
834:      C                      if ( IKR.gt.0 ) then
835:      C                          NL1      = IKR
836:      C                          go to 210
837:      C                      end if
838:      C                      ..... store the given data .....
839:      C                      MO      = MO + 1
840:      C                      do K = 1, NR
841:      C                          do J = 1, IINUCPN
842:      C                              do I = 1, IIMTPN
843:      C                                  if ( KNAME0.le.0 ) then
844:      C                                      do I2 = 1, KLBPNI(KN(J),4)
845:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
846:      C                                      end do
847:      C                                      go to 218
848:      C                                  217      continue
849:      C                                  end if
850:      C                                  NN      = NN + 1
851:      C                                  I1      = 4 * (NN - 1) - 1
852:      C                                  IA(LMONPNW+I1+1) = KR(K)
853:      C                                  IA(LMONPNW+I1+2) = KN(J)
854:      C                                  IA(LMONPNW+I1+3) = KM(I)
855:      C                                  IA(LMONPNW+I1+4) = MO
856:      C                                  218      continue
857:      C                              end do
858:      C                          end do
859:      C                      go to 225
860:      C                  220      MO      = MO + 1
861:      C                      do K = 1, NR
862:      C                          do J = 1, IINUCPN
863:      C                              NMT2G      = KLBPN1(KN(J),4)
864:      C                              producing secondary products
865:      C                              do I = 1, NMT2G
866:      C                                  NN      = NN + 1
867:      C                                  I1      = 4 * (NN - 1) - 1
868:      C                                  IA(LMONPNW+I1+1) = KR(K)
869:      C                                  IA(LMONPNW+I1+2) = KN(J)
870:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
871:      C                                  IA(LMONPNW+I1+4) = MO
872:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
873:      C                              end do
874:      C                          end do
875:      C                      end do
876:      C                  225      continue
877:      C                  ..... single data block : (nuclide,reaction) .....
878:      C                  else
879:      C                  ..... check and read the nuclide name .....
880:      C                  if ( NLEN.le.0 ) go to 929
881:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
882:      C                  if ( KNAME0.gt.0 ) then
883:      C                      IINUCPN = 0
884:      C                      NL1      = 1
885:      C                      NL2      = NLEN
886:      C                      IKR      = 0
887:      C                      do I = NL1, NL2
888:      C                          if ( LINE(I:I).eq.'+' ) then
889:      C                              NL2      = I - 1
890:      C                              IKR      = I + 1
891:      C                          go to 215
892:      C                      end if
893:      C                      end do
894:      C                      215      IIMTPN = IIMTPN + 1
895:      C                      read(LINE(NL1:NL2),*) MTPN0
896:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
897:      C                      KM(IIMTPN) = MTPN0
898:      C                      if ( IKR.gt.0 ) then
899:      C                          NL1      = IKR
900:      C                          go to 210
901:      C                      end if
902:      C                      ..... store the given data .....
903:      C                      MO      = MO + 1
904:      C                      do K = 1, NR
905:      C                          do J = 1, IINUCPN
906:      C                              do I = 1, IIMTPN
907:      C                                  if ( KNAME0.le.0 ) then
908:      C                                      do I2 = 1, KLBPNI(KN(J),4)
909:      C                                          if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 217
910:      C                                      end do
911:      C                                      go to 218
912:      C                                  217      continue
913:      C                                  end if
914:      C                                  NN      = NN + 1
915:      C                                  I1      = 4 * (NN - 1) - 1
916:      C                                  IA(LMONPNW+I1+1) = KR(K)
917:      C                                  IA(LMONPNW+I1+2) = KN(J)
918:      C                                  IA(LMONPNW+I1+3) = KM(I)
919:      C                                  IA(LMONPNW+I1+4) = MO
920:      C                                  218      continue
921:      C                              end do
922:      C                          end do
923:      C                      go to 225
924:      C                  220      MO      = MO + 1
925:      C                      do K = 1, NR
926:      C                          do J = 1, IINUCPN
927:      C                              NMT2G      = KLBPN1(KN(J),4)
928:      C                              producing secondary products
929:      C                              do I = 1, NMT2G
930:      C                                  NN      = NN + 1
931:      C                                  I1      = 4 * (NN - 1) - 1
932:      C                                  IA(LMONPNW+I1+1) = KR(K)
933:      C                                  IA(LMONPNW+I1+2) = KN(J)
934:      C                                  IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9)
935:      C                                  IA(LMONPNW+I1+4) = MO
936:      C                                  if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1
937:      C                              end do
938:      C                          end do
939:      C                      end do
940:      C                  225      continue
941:      C                  ..... single data block : (nuclide,reaction) .....
942:      C                  else
943:      C                  ..... check and read the nuclide name .....
944:      C                  if ( NLEN.le.0 ) go to 929
945:      C                  KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
946:      C                  if ( KNAME0.gt.0 ) then
947:      C                      IINUCPN = 0
948:      C                      NL1      = 1
949:      C                      NL2      = NLEN
950:      C                      IKR      = 0
951:      C                      do I = NL1, NL2
952:      C                          if ( LINE(I:I).eq.'+' ) then
953:      C                              NL2      = I - 1
954:      C                              IKR      = I + 1
955:      C                          go to 215
956:      C                      end if
957:      C                      end do
958:      C                      215      IIMTPN = IIMTPN + 1
959:      C                      read(LINE(NL1:NL2),*) MTPN0
960:      C                      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
961:      C                      KM(IIMTPN) = MTPN0
962:      C                      if ( IKR.gt.0 ) then
96
```

src/mvp/monpnwinp.f

```

374: 230      continue      ! return here, if multiple nucl in the region
ides are given
375:      NL2      = NLEN
376:      IKN      = 0
377:      do I = NL1, NL2
378:          if ( LINE(I:I).eq.'+' ) then
379:              NL2      = I - 1
380:              IKN      = I + 1
381:              go to 235
382:          end if
383:      end do
235      NL0      = NL2 - NL1 + 1
384:      if ( NL0.eq.5 ) then
385:          NM      = LINE(NL1:NL2)
386:      else if ( NL0.eq.3 ) then
387:          NM      = LINE(NL1:NL2) // ' '
388:      else if ( NL0.eq.2 ) then
389:          NM      = LINE(NL1:NL2) // ' '
390:      else if ( NL0.eq.1 ) then
391:          NM      = LINE(NL1:NL2) // '0 '
392:      else
393:          go to 929
394:      end if
395:      IKW      = 0
396:      if ( NM(3:5).eq.'000' ) then
397:          IKW      = 1
398:      else if ( NM(3:3).eq.'*' ) then
399:          IKW      = 1
400:      else if ( NM(2:2).eq.'*' ) then
401:          IKW      = 1
402:          NM(2:2) = '0'
403:      else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
404:          IKW      = 1
405:      end if
406:      if ( IKW.eq.0 ) then      ! single isotope
407:          do I = 1, NUCPN
408:              if ( NM(1:NL0).eq.NCIDPN(I)(1:NL0) ) go to 240
409:          end do
410:          go to 923
411:      IINUCPN = IINUCPN + 1
412:      KN(IINUCPN) = I
413:      240      if ( IKW.eq.1 ) then      ! element
414:          II      = 0
415:          245      do I = II+1, NUCPN
416:              if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 250
417:          end do
418:          if ( IINUCPN.le.0 ) go to 923
419:          go to 255
420:      IINUCPN = IINUCPN + 1
421:      250      KN(IINUCPN) = I
422:      if ( I.lt.NUCPN ) then
423:          II      = I
424:          go to 245
425:      end if
426:      255      continue
427:      end if
428:      if ( IKN.gt.0 ) then
429:          NL1      = IKN
430:          go to 230
431:      end if
432:      if ( ITERM.ne.0 ) then      ! undefined reaction type
433:          if ( IINUCPN.le.0 ) go to 935
434:          go to 280
435:      end if
436:      else
437:          ! nuclide name is undefined. --> all nuclides

438:      IINUCPN = NUCPN
439:      do I = 1, NUCPN
440:          KN(I) = I
441:      end do
442:      go to 260
443:  end if
444:  C      ..... read the reaction type (mt) .....
445:  call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
446:  if ( IEND.ne.0 ) go to 919
447:  if ( NLEN.le.0 ) go to 929
448:  260      continue
449:  IIMTPN = 0
450:  NL1      = 1
451:  270      continue      ! return here, if multiple reac
tions are given
452:  NL2      = NLEN
453:  IKR      = 0
454:  do I = NL1, NL2
455:      if ( LINE(I:I).eq.'+' ) then
456:          NL2      = I - 1
457:          IKR      = I + 1
458:          go to 275
459:      end if
460:  end do
461:  275      IIMTPN = IIMTPN + 1
462:      read(LINE(NL1:NL2),*) MTPN0
463:      if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
464:      KM(IIMTPN) = MTPN0
465:      if ( IKR.gt.0 ) then
466:          NL1      = IKR
467:          go to 270
468:      end if
469:  C      ..... store the given data .....
470:  MO      = MO + 1      ! monitoring number
471:  do K = 1, NR
472:      do J = 1, IINUCPN
473:          do I = 1, IIMTPN
474:              if ( KNAME0.le.0 ) then
475:                  do I2 = 1, KLBPNI(KN(J),4)
476:                      if ( KM(I).eq.KLBPN2(KN(J),I2,9) ) go to 277
477:                  end do
478:                  go to 278
479:                  277      continue
480:              end if
481:              NN      = NN + 1
482:              I1      = 4 * (NN - 1) - 1
483:              IA(LMONPNW+I1+1) = KR(K)      ! region number
484:              IA(LMONPNW+I1+2) = KN(J)      ! photonuclear nuclide
485:              IA(LMONPNW+I1+3) = KM(I)      ! photonuclear reactio
n number
486:              IA(LMONPNW+I1+4) = MO      ! monitoring number
487:          end do
488:      end do
489:  end do
490:  go to 285
491:  280      MO      = MO + 1      ! monitoring number
492:      do K = 1, NR
493:          do J = 1, IINUCPN
494:              NMT2G = KLBPNI(KN(J),4)      ! number of reactions
495:          producing secondary products
496:          do I = 1, NMT2G
497:              NN      = NN + 1

```

src/mvp/monpnwinp.f

```

498:          I1      = 4 * (NN - 1) - 1
499:          IA(LMONPNW+I1+1) = KR(K)          ! region number
500:          IA(LMONPNW+I1+2) = KN(J)          ! photonuclear nuclide
number
501:          IA(LMONPNW+I1+3) = KLBPN2(KN(J),I,9) ! photonuclear reactio
n number
502:          IA(LMONPNW+I1+4) = MO              ! monitoring number
503:          if ( KLBPN2(KN(J),I,9).eq.98 ) NN = NN - 1 ! deletion of d
elayed neutron
504:          end do
505:          end do
506:          end do
507: 285      continue
508:          if ( ITERM.gt.0 ) go to 110
509: 290      call CHREAD( ' ', LINE, ' )' NLEN, ITERM, IEND )
510:          if ( IEND.ne.0 ) go to 919
511:          if ( ITERM.eq.0 ) go to 290
512:          go to 110
513:          end if
514:          go to 160
515: C
516: C      >>> end of input
517: C
518: 500      continue
519:          NMONPNW = NN
520:          NN2     = 4 * NN
521:          if ( NN1.ne.NN2 ) call RESIZE( 'MONPNWGT', LMONPNW, NN2,'I4',LAST)
522:          return
523: C
524: C      ..... error process .....
525: 901      write(IMG,902)
526:          call PRSTOP( 1, 'Problem in order of input data.' )
527:          go to 999
528: 903      write(IMG,904)
529:          call PRSTOP( 1, 'Problem in order of input data.' )
530:          go to 999
531: 905      write(IMG,906)
532:          call PRSTOP( 1, 'Unexpected end of input file.' )
533:          go to 999
534: 907      write(IMG,908)
535:          call PRSTOP( 1, 'Unexpected end of input file.' )
536:          go to 999
537: 911      write(IMG,912)
538:          call PRSTOP( 1, 'Error in region dependent data input.' )
539:          go to 999
540: 913      write(IMG,914) (ITRNM(I),I=1,NTREG)
541:          call PRSTOP( 1, 'Problem in region dependent data input.' )
542:          go to 999
543: 917      write(IMG,918)
544:          call PRSTOP( 1, 'Unexpected end of input file.' )
545:          go to 999
546: 919      write(IMG,920)
547:          call PRSTOP( 1, 'Unexpected end of input file.' )
548:          go to 999
549: 921      write(IMG,922)
550:          call PRSTOP( 1, 'Incomplete data structure of input file.' )
551:          go to 999
552: 923      write(IMG,924) NM, NLEN
553:          call PRSTOP( 1, 'Unknown nuclide name in input file.' )
554:          go to 999
555: 925      write(IMG,926)
556:          call PRSTOP( 1, 'Problem in order of input data.' )
557:          go to 999
558: 927      write(IMG,928) MTPN0
559:          call PRSTOP( 1, 'Incomplete data structure of input file.' )

```

```

560:          go to 999
561: 929      write(IMG,930) NLEN
562:          call PRSTOP( 1, 'Incomplete data structure of input file.' )
563:          go to 999
564: 931      write(IMG,932) NN1
565:          call PRSTOP( 1, 'Lack of storage size of input data.' )
566:          go to 999
567: 935      write(IMG,936)
568:          call PRSTOP( 1, 'Incomplete data structure of input file.' )
569:          go to 999
570: 937      write(IMG,938)
571:          call PRSTOP( 1, 'Unknown data in input file.' )
572:          stop 888
573: 902      format(/' XXX(monpnwinp) Number of "photonuclear nuclide" (NUCPN',
574:          & ' ) is not determined before photonuclear dependent data input.'
575:          & /15x,'Your input for <MONPNWGT> must be placed after',
576:          & ' $XSEC block.')
577: 904      format(/' XXX(monpnwinp) Number of "region" (NREG) is not determ',
578:          & ' ined before REGION dependent data input.'
579:          & /15x,'Your input for <MONPNWGT> must be placed after',
580:          & ' $GEOMETRY block.')
581: 906      format(/' XXX(monpnwinp) End of file is encountered for',
582:          & ' <MONPNWGT>.')
583: 908      format(/' XXX(monpnwinp) End of file is encountered during',
584:          & ' "!region-name(...)" search for <MONPNWGT>.')
585: 910      format(/' XXX(monpnwinp) Tally region <',a,'> does not exist.'
586:          & /15x,'Data of <MONPNWGT> for this region are meaningless.')
587: 912      format(/' XXX(monpnwinp) Unexpected end of data or input error',
588:          & ' during skipping unnecessary or invalid data.'
589:          & /' (REGION dependent data input)')
590: 914      format(/' XXX(monpnwinp) Fatal error for tally-region name',
591:          & ' (ITRNM).')
592:          & /(i6,i10))
593: 916      format(/' XXX(monpnwinp) Region <',a,'> is not defined or',
594:          & ' invalid name.'
595:          & /15x,'Data of <MONPNWGT> for this region are meaningless.')
596: 918      format(/' XXX(monpnwinp) End of file is encountered during data',
597:          & ' in "!region-name(nuclide,reaction)..." or'
598:          & /15x,'"!region-name(nuclide,reaction)" for <MONPNWGT>.')
599: 920      format(/' XXX(monpnwinp) End of file is encountered during data',
600:          & ' in "(nuclide,reaction)" for <MONPNWGT>.')
601: 922      format(/' XXX(monpnwinp) Structure of data block incomplete in',
602:          & ' "(nuclide,reaction)" for <MONPNWGT>.')
603: 924      format(/' XXX(monpnwinp) Nuclide name is not existed in nuclide',
604:          & ' name list for <MONPNWGT>.')
605:          & /15x,'name=',a,' length=',i3)
606: 926      format(/' XXX(monpnwinp) Number of reactions on "photonuclear',
607:          & ' nuclide" (NMPN) is not determined before photonuclear',
608:          & ' dependent data input.'
609:          & /15x,'Your input for <MONPNWGT> must be placed after',
610:          & ' $XSEC block.')
611: 928      format(/' XXX(monpnwinp) Reaction type for photonuclear is out',
612:          & ' of range (1 to 135) for <MONPNWGT>.')
613:          & /15x,'MTPN0=',i5)
614: 930      format(/' XXX(monpnwinp) Length of data in "(nuclide,reaction)',
615:          & ' " is illegal for <MONPNWGT>.')
616:          & /15x,'NLEN=',i5)
617: 932      format(/' XXX(monpnwinp) Storage size of data is lack for',
618:          & ' <MONPNWGT>.')
619:          & /15x,'NN1=',i7)
620: 936      format(/' XXX(monpnwinp) Nuclide name is not recognized for',
621:          & ' <MONPNWGT>.')
622: 938      format(/' XXX(monpnwinp) First data in each block is must be a',
623:          & ' region name for <MONPNWGT>.')
624:          end

```

src/mvp/monpnwinp.f

```

625: c##
626:      subroutine MONPNWWT( MONPNWGT, NMONPNW, MONPNW, IPR )
627: C=<MVP>=====
628: C  purpose:  print array data for monitoring a particle weight from a
629: C            specified reaction type (mt) by region and nuclide in
630: C            photonuclear reaction.
631: C  called in:  INTRO2
632: C  calls:
633: C=====
634: C
635: C      .... data for photonuclear ....
636: C      integer MONPNWGT(4,NMONPNW)
637: C
638: C      .... local variable ....
639: C      character HL0(200)*10
640: C
641: C-----
642: C
643: C      ..... initialize .....
644: C      NM      = MONPNWGT(4,NMONPNW)
645: C      N1      = 1
646: C      N2      = NMONPNW
647: C      MONPNW  = NM
648: C
649: C      ..... print the reading data and check .....
650: C      write(IPR,600) NMONPNW
651: C      do M = 1, NM
652: C        write(IPR,601) M
653: C        do I = N1, N2
654: C          if ( MONPNWGT(4,I).ne.M ) then
655: C            N2      = I - 1
656: C            go to 110
657: C          end if
658: C        end do
659: C      110  continue
660: C        write(HL0(1),501) MONPNWGT(2,N1), MONPNWGT(3,N1)
661: C        IR      = MONPNWGT(1,N1)
662: C        I3      = 1
663: C        N3      = N1
664: C      120  continue
665: C        N3      = N3 + 1
666: C        if ( N3.gt.N2 ) go to 130
667: C        if ( IR.eq.MONPNWGT(1,N3) ) then
668: C          I3      = I3 + 1
669: C          write(HL0(I3),501) MONPNWGT(2,N3), MONPNWGT(3,N3)
670: C        else
671: C          if ( I3.gt.0 ) write(IPR,602) IR, (HL0(I),I=1,I3)
672: C          I3      = 1
673: C          IR      = MONPNWGT(1,N3)
674: C        end if
675: C        go to 120
676: C      130  if ( I3.gt.0 ) write(IPR,602) IR, (HL0(I),I=1,I3)
677: C        if ( M.lt.NM ) then
678: C          N1      = N2 + 1
679: C          N2      = NMONPNW
680: C        end if
681: C      end do
682: C
683: C      return
684: C
685: C      501 format(' (',i3,',',i3,')')
686: C      600 format('/' <<< ARRAY MONPNWGT(1:4,I) (I=1,',i4,') >>> .....',
687: C              & ' form: region (nuclide, reaction)')
688: C      601 format('/' monitoring set number ',i4
689: C              & /' -----')

```

```

690:      602 format(' REG.',i4,' | ',12a10: /(10x,' | ',12a10))
691:      end

```

src/mvp/mtdata.f

```

1:      subroutine MTDATA( MPK,   IDMAT, MNUC, LPDEN, NMAT,  INUCT,
2:      &                  DENST, TPRMIN,TMPMAX,TMPMIN,DSMAT, NUCID, NUC,
3:      &                  MAXNC, MLEN,  LAST,  LIMIT, LIMITA, JPHNU )
4: C=====
5: C PURPOSE: TO GIVE VALUES TO MATERIAL DATA &
6: C          GET TOTAL NUCLIDE NUMBER AND ID TABLE
7: C CALLED IN: CTXSIN
8: C CALLS: pctlb, unpknd, unpkcs
9: C=====
10:     integer MPK(*)
11:     integer IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1)
12:     integer INUCT(MLEN)
13:     real DENST(MLEN)
14:     real*8 TPRMIN, TMPMAX, TMPMIN
15:     real DSMAT(NMAT)
16:     character*16 NUCID(MAXNC)
17: C
18:     include '../shared/INC/_IUNIT'
19:     include '../shared/INC/_WORDL'
20: C
21: C     .... local data ...
22:     character CBUF*8, TNUCID*16, NUCID0*16
23: c##<2007/03/14:PN3:
24:     character*16 NUCIDP(200), TNUCIDP
25: c##>
26:     real*8 TEMPS, TEMPS0, TPRES
27:     character*64 CWRK
28:     integer NDIGIT
29:     character*16 FMT
30: C-----
31: C
32: C     .... check length of nuclide temperature suffix for nuclide ID
33: C     if any positive temperature is input.
34: C
35:     NTMPDG = 0
36:     if ( TMPMAX.gt.0 ) then
37:         NTMPDG = INT(LOG10(TMPMAX+1.0D-5)) + 1
38:     end if
39: C
40:     NUC      = 0
41:     LPDEN(1) = 1
42:     do 180 M = 1, NMAT
43: C
44: C     ... label ...
45: C
46:         NDIGIT = int(log10(real(M)))+1
47:         write(FMT,'(a6,i1,a1)') '('&'',I',NDIGIT,')'
48:         write(CBUF,FMT) M
49:         call PCTLB(MPK, 'MATERIAL HEADER', CBUF(1:NDIGIT+1), IRET)
50:
51:         if ( IRET.ne.0 ) then
52: c##<2007/03/14:PN3:
53: c##         write(IMG*,*) 'XXX MISSING DATA FOR ', M, 'TH' MATERIAL.'
54:         write(IMG,'(1X,A,I5,A)') 'XXX(mtdata) MISSING DATA FOR ',
55:             & M, 'TH' MATERIAL.'
56: c##>
57:             call CNTERR( 'FATAL' )
58:             return
59:         end if
60: C
61: C
62:         call PCTLB( MPK, 'NUMBER OF NUCLIDE', 'MNUC', IRET )
63:         call UNPKND( MPK, 'NUMBER OF NUCLIDE', 'I4', MNUC(M), 1, ND,
64:             & IRET )
65:         LPDEN(M+1) = LPDEN(M) + MNUC(M)

```

```

66: C
67:     call PCTLB(MPK, 'MATERIAL HEADER', CBUF(1:NDIGIT+1), IRET)
68:     call PCTLB( MPK, 'MATERIAL ID', 'IDMAT', IRET1 )
69:     call UNPKND( MPK, 'MATERIAL ID', 'I4', IDMAT(M), 1, ND, IRET2 )
70:     if ( IRET1.ne.0 .or. IRET2.ne.0 ) then
71: c##<2007/03/14:PN3:
72: c##         write(IMG*,*) 'XXX MISSING ID FOR ', M, 'TH' MATERIAL.',
73: c## & IRET1, IRET2
74:         write(IMG,'(1X,A,I5,A,2I6)') 'XXX(mtdata) MISSING ID FOR ',
75:             & M, 'TH' MATERIAL. ', IRET1, IRET2
76: c##>
77:         call CNTERR( 'FATAL' )
78:         return
79:     end if
80: C-----
81: C     CHECK duplicate definition of material ID
82: C-----
83:     if ( IDMAT(M).gt.0 ) then
84:         do 100 MM = 1, M - 1
85:             if ( IDMAT(MM).eq.IDMAT(M) ) then
86:                 write(IMG,'(1X,a,i3,a,i5,a,i3,a)')
87:                     & 'XXX(mtdata) ID OF ', M, 'th material (',
88:                     & IDMAT(M), ' ) is same as that of ', MM,
89:                     & 'th material.'
90:                 call CNTERR( 'FATAL' )
91:             end if
92:         100 continue
93:     end if
94: C-----
95: C     CHECK NUCLIDE ID & densities
96: C-----
97:     DSMAT(M) = 0.0
98:     do 170 N = 1, MNUC(M)
99:         TNUCID = ' '
100: C
101: C     .... unpack nuclide-ID & number density ....
102: C
103:         call UNPKCS( MPK, 'NUCLIDE ID', TNUCID, NL, IRET1 )
104: c##<2007/03/14:PN3:
105:         call UNPKCS( MPK, 'PN-NUCLIDE ID', TNUCIDP, NLP, IRET5 )
106: c##>
107:         call UNPKND( MPK, 'NUMBER DENSITY', 'R4', RDEN, 1, ND, IRET2 )
108:         &
109:         call UNPKND( MPK, 'TEMPERATURE', 'R8', TEMPS0, 1, ND, IRET3 )
110:         &
111:         call UNPKND( MPK, 'TPRES', 'R8', TPRES, 1, ND, IRET4 )
112: c##<2007/03/14:PN3:
113: c##         if ( IRET1.ne.0 .or. IRET2.ne.0 .or. IRET3.ne.0
114: c## & .or. IRET4.ne.0 ) then
115: c##             write(IMG*,*) 'XXX(MTDATA) invalid data in data packet ',
116: c## & IRET1, IRET2, IRET3, IRET4, M, N
117: c##         if ( IRET1.ne.0 .or. IRET2.ne.0 .or. IRET3.ne.0 .or.
118: c## & IRET4.ne.0 .or. IRET5.ne.0 ) then
119: c##             write(IMG,'(1X,A,5I6,2I5)')
120: c## & 'XXX(MTDATA) invalid data in data packet ',
121: c## & IRET1, IRET2, IRET3, IRET4, IRET5, M, N
122: c##>
123:             call CNTERR( 'FATAL' )
124:             return
125:         end if
126: C
127: C     ... create nuclide ID with temperature if positive temperature
128: C     is given.
129: C
130: c##<2007/03/14:PN3:

```

src/mvp/mtdata.f

```

131:         if ( JPHNU.ne.0 ) then
132:             NUCID0 = ' '
133:             NUCID0 = TNUCIDP(:NLP)
134:             TNUCIDP = NUCID0
135:         end if
136:         NUCID0 = ' '
137: c##>
138:         NUCID0 = TNUCID(:NL)
139:         TEMPS = TEMPS0
140:         if ( TEMPS0.gt.0.0D0 ) then
141:             TEMPS = NINT(TEMPS0/TPRECS)*TPRECS
142:             if ( TEMPS.ne.TEMPS0 ) then
143:                 write(IPR,7000) TNUCID(:NL), TEMPS0, TEMPS
144:             end if
145: C
146:             CWRK = ' '
147:             write(CWRK,'(f20.6)') TEMPS
148: C
149: C ... cut unnecessary zeros after decimal point
150:             K = INDEX(CWRK,'.')
151:             do 110 II = LEN(CWRK), K + 1, -1
152:                 if ( INDEX('123456789',CWRK(II:II)).ne.0 ) go to 120
153:             110 continue
154:             II = K - 1
155:             120 if( II.le.len(CWRK)) CWRK(II+1:) = ' '
156:             call CCOMP( CWRK, LEN(CWRK), CWRK, IFL )
157:             LL= ICLEN(CWRK)
158:             if ( NL+LL.gt.LEN(TNUCID) ) then
159: c##<2007/03/14:PN3:
160: c##             write(IMG,*)
161: c## &             'XXX(MTDATA) Tried to attach temperature string <',
162: c## &             CWRK(:LL), '> to nuclide ID <', TNUCID(:NL),
163: c## &             '> but',
164: c## &             ' resulting string length is too long.'
165: c##             write(IMG,*)
166: c## &             ' Probably you required too high precision for',
167: c## &             ' nuclide temperature. (TPRECS =', TPRECS,
168: c## &             ') '
169:             write(IMG,192) CWRK(:LL), TNUCID(:NL), TPRECS
170: c##>
171:             call CNTERR( 'FATAL' )
172:         end if
173:         TNUCID(NL+1:) = CWRK(:LL)
174:         NL = ICLEN(TNUCID)
175:     end if
176: C
177: 7000 format(1X,' >>> Temperature of nuclide <',A,> is rounded',
178: &             ' from ',F14.6,' to ',F14.6)
179: C
180: c##<2007/03/14:PN3:
181: c##         do 130 K = 1, NUC
182: c##             if ( NUCID(K).eq.TNUCID(:NL) ) go to 140
183: c#130         continue
184:             if ( NUC.gt.0 ) then
185:                 if ( JPHNU.eq.0 ) then
186:                     do K = 1, NUC
187:                         if ( NUCID(K).eq.TNUCID(:NL) ) go to 140
188:                     end do
189:                 else
190:                     do K = 1, NUC
191:                         if ( NUCID(K).eq.TNUCID(:NL).and.
192: &                             NUCIDP(K).eq.TNUCIDP(:NLP) ) go to 140
193:                     &                     end do
194:                 end if
195:             end if

```

```

196: c##>
197:         NUC = NUC + 1
198: C
199:         if ( NUC.gt.MAXNC ) go to 190
200:         NUCID(NUC) = TNUCID(:NL)
201: c##<2007/03/14:PN3:
202:         if ( JPHNU.ne.0 ) then
203:             if ( NUC.gt.200 ) go to 191
204:             NUCIDP(NUC) = TNUCIDP(:NLP)
205:         end if
206: c##>
207:         K = NUC
208: C
209: C ... save nuclide ID as input and temperature on work file.
210: c##<2007/03/14:PN3:
211: c##             write(IUB) NUCID0,TEMPS
212:             write(IUB) NUCID0, TEMPS, TNUCIDP
213: c##>
214: C
215: 140         continue
216: C
217: c##<2007/03/14:PN3:
218: c##C             ... set number density table .... ID & densyty
219: C             ... set number density table .... ID & density
220: c##>
221: C
222:             INUCT(LPDEN(M)+N-1) = K
223:             DENST(LPDEN(M)+N-1) = RDEN
224: C
225: C
226:             if ( RDEN.ge.0.0 ) then
227:                 DSMAT(M) = DSMAT(M) + RDEN
228:             end if
229: C
230: c##<2007/03/14:PN3:
231: c##C             ... check whether the same nuc-ID is used more than once
232: C             ... check whether the same nuclide-ID is used more than once
233: c##>
234: C             for a material.
235: C
236:             do 150 KM = 1, N - 1
237:                 if ( INUCT(LPDEN(M)+KM-1).eq.K ) then
238:                     write(IMG,'(/1x,a,a,a,i5)') '!!! nuclide <',
239: &                         NUCID(K):(ICLEN2(NUCID(K))),
240: &                         '> is used more than once for material ',
241: &                         IDMAT(M)
242:                     call CNTERR( 'WARNING' )
243:                     go to 160
244:                 end if
245:             150 continue
246:             160 continue
247:             170 continue
248: C
249:             if ( DSMAT(M).eq.0.0 ) then
250:                 write(IMG,'(/1x,a,i5,a/3x,a,a/)')
251: &                 '!!! all number densities for material ', IDMAT(M),
252: &                 ' are zero.',
253: &                 ' Material # of zones including this material will be',
254: &                 ' replaced by inner-void material (0).'
255:                 call CNTERR( 'WARNING' )
256:             end if
257:             180 continue
258: C-----
259: c##<2007/03/14:PN3:
260: c##C             PRINT NUCLDE ID'S

```

src/mvp/mtdata.f

```
261: C PRINT NUCLIDE ID'S
262: c##>
263: C-----
264: c##<2007/03/14:PN3:
265: c## write(IPR,7020) NUC, (N,NUCID(N),N=1,NUC)
266: if ( JPHNU.eq.0 ) then
267: write(IPR,7020) NUC, (N,NUCID(N),N=1,NUC)
268: else
269: write(IPR,7021) NUC, (N,NUCID(N),NUCIDP(N),N=1,NUC)
270: end if
271: c##>
272: 7020 format(///' *****'/
273: & ' NUCLIDES : TOTAL NUMBER = ',I4/
274: & ' *****'/
275: & '(1X,5X,5(I3,2X,'<',A,'>',2X))
276: c##<2007/03/14:PN3:
277: 7021 format(///' *****'/
278: & ' NUCLIDE SET : TOTAL NUMBER = ',I4/
279: & ' *****'/
280: & '(1X,5X,3(I3,1X,'<',A,' - ',A,'>',2X))
281: c##>
282: return
283: C -----
284: C 170 write(IPR,*) 'XXX MEMORY INSUFFICIENT TO STORE NUCLIDE ID'S'
285: C write(IPR,*) ' INCREASE MEMORY LIMIT ( = ', LIMIT, ' )'
286: C call PRSTOP( 1, 'MEMORY OVER.' )
287:
288: c##<2007/03/14:PN3:
289: c#190 write(IMG,*) 'XXX Character memory is insufficient to store '
290: c## & 'nuclide ID's.'
291: 190 write(IMG,(1X,A,A)) 'XXX(mtdata) Character memory is',
292: & ' insufficient to store nuclide ID's.'
293: go to 900
294: 191 write(IMG,(1X,A,A)) 'XXX(mtdata) Character memory is',
295: & ' insufficient to store photonuclear nuclide ID's.'
296: 192 format(' XXX(mtdata) Tried to attach temperature string <', A,
297: & '> to nuclide ID <', A, '> but resulting string length is too',
298: & ' long.'
299: &/' Probably you required too high precision for nuclide',
300: & ' temperature. (TPRECS=',1P,E12.5,')')
301: 900 continue
302: c##>
303: call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
304: stop 999
305: end
```

src/mvp/mtdatp.f

```

1:      subroutine MTDATP( IDMAT, MNUC, LPDEN, NMAT, NUCID, NUC,
2:      &                  INUCT, DENST, MLEN, NPATOM, NATMT, MPATM,
3:      &                  LPDNP, IATMT, DNSTP )
4: C=<MVP>=====
5: C  PURPOSE:
6: C      GET TOTAL ATOM NUMBER AND ATOM NUMBER TABLE FOR PHOTON REACTION.
7: C  CALLED IN: CTXSIN   CALLS: NATOMZ, CHSYMB
8: C=====
9: C
10: C  .... MATERIAL DATA FOR NUCLIDES ....
11: C
12: C      integer IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1)
13: C      integer INUCT(MLEN)
14: C      real    DENST(MLEN)
15: C      character*16 NUCID(NUC)
16: C
17: C
18: C  .... MATERIAL DATA FOR ATOMS .....
19: C
20: C      MPATM(NMAT) : NUMBER OF ATOM INCLUDED IN EACH MATERIAL.
21: C      LPDNP(NMAT) : POSITION POINTER FOR TABLES
22: C      NPATOM      : NUMBER OF ATOMIC ELEMENTS USE IN PROBLEM
23: C      NATMT(NPATOM) : ATOMIC NUMBER TABLE USE IN PROBLEM
24: C      IATMT(MLEN)  : ATOMIC NUMBER TABLE (POSITION IN NATMT(NMAT))
25: C      DNSTP(MLEN)  : NUMBER DENSITY TABLE
26: C
27: C
28: C      integer MPATM(NMAT), LPDNP(NMAT+1)
29: C      integer NATMT(NUC), IATMT(MLEN)
30: C      real    DNSTP(MLEN)
31: C
32: C      include '../shared/INC/_IOUNIT'
33: C
34: C      character*2 CHSYMB
35: C      external CHSYMB
36: C
37: C-----
38: C
39: C      NPATOM = 0
40: C
41: C      do 120 M = 1, NMAT
42: C-----
43: C      CHECK NUCLIDE ID
44: C-----
45: C      do 110 N = 1, MNUC(M)
46: C          L = LPDEN(M) + N - 1
47: C
48: C      .... GET ATOMIC NUMBER FROM NUCLIDE ID .....
49: C
50: C          NA = NATOMZ(NUCID(INUCT(L)))
51: C
52: C          if ( NA.eq.0 ) then
53: C              write(IMSG,7000) NUCID(INUCT(L))
54: C              format(/1X,'XXX CANNOT DETERMINE ATOMIC NUMBER ',
55: C                  &      'FOR NUCLIDE <',A,'> XXX STOP ! ')
56: C              call PRSTOP( 1, 'INVALID NUCLIDE SYSBOL.' )
57: C              stop 888
58: C          end if
59: C
60: C      do 100 K = 1, NPATOM
61: C          if ( NATMT(K).eq.NA ) go to 110
62: C      100 continue
63: C      NPATOM = NPATOM + 1
64: C      NATMT(NPATOM) = NA
65: C

```

```

66: 110 continue
67: 120 continue
68: C
69: C      .... SORT ATOMIC NUMBERS ....
70: C
71: C      do 140 N = NPATOM - 1, 1, -1
72: C          IFF = 0
73: C          do 130 M = 1, N
74: C              if ( NATMT(M).gt.NATMT(M+1) ) then
75: C                  NA1 = NATMT(M)
76: C                  NATMT(M) = NATMT(M+1)
77: C                  NATMT(M+1) = NA1
78: C                  IFF = 1
79: C              end if
80: C          130 continue
81: C          if ( IFF.eq.0 ) go to 150
82: C      140 continue
83: C      150 continue
84: C
85: C
86: C      *****
87: C      call LABEL( IPR, 'ATOMIC NUMBERS' )
88: C      *****
89: C
90: C      write(IPR,'(/1X,' * NUMBER OF ATOM IN PHOTON REACTION: ',I4)')
91: C      &      NPATOM
92: C      write(IPR,'(/1X,' * ATOMIC NUMBERS : ''')')
93: C      write(IPR,'(/1X,9X,10I5)') (NATMT(I),I=1,NPATOM)
94: C
95: C      LPDNP(1) = 1
96: C      do 210 M = 1, NMAT
97: C-----
98: C      COUNT NUMBER OF ATOM FOR EACH MATERIAL
99: C-----
100: C          LP = LPDNP(M)
101: C          MPATM(M) = 0
102: C          do 200 N = 1, MNUC(M)
103: C              L = LPDEN(M) + N - 1
104: C
105: C      .... GET ATOMIC NUMBER FROM NUCLIDE ID ....
106: C
107: C          NA = NATOMZ(NUCID(INUCT(L)))
108: C          do 160 NAA = 1, NPATOM
109: C              if ( NATMT(NAA).eq.NA ) go to 170
110: C          160 continue
111: C          170 continue
112: C
113: C          do 180 K = 1, MPATM(M)
114: C              if ( IATMT(LP+K-1).eq.NAA ) go to 190
115: C          180 continue
116: C
117: C          MPATM(M) = MPATM(M) + 1
118: C          IATMT(LP+MPATM(M)-1) = NAA
119: C          DNSTP(LP+MPATM(M)-1) = DENST(L)
120: C          go to 200
121: C
122: C      190 DNSTP(LP+K-1) = DNSTP(LP+K-1) + DENST(L)
123: C
124: C      200 continue
125: C      LPDNP(M+1) = LPDNP(M) + MPATM(M)
126: C      210 continue
127: C
128: C
129: C      *****
130: C      call LABEL( IPR, 'NUMBER DENSITIES FOR EACH ATOMIC ELEMENT' )

```


src/mvp/mtdatp.f

```
131: C      *****
132: C
133:      write(IPR,7020)
134: 7020 format(/1X,5X,' MAT-ID  ATOM #   SYMBOL   DENSITY (/CM3*BARN)'/1X,
135:      &      5X,'-----  -----  -----  -----'/)
136:      do 230 M = 1, NMAT
137:          LP      = LPDNP(M)
138:          write(IPR,'(1X ,5X,I7)') IDMAT(M)
139:          do 220 N = 1, MPATM(M)
140:              NA      = NATMT(IATMT(LP+N-1))
141:              write(IPR,7040) NA, CHSYMB(NA), DNSTP(LP+N-1)
142: 220      continue
143: 7040      format(1X,5X,7X,3X,I3,6X,A2,7X,1PE12.5)
144: 230      continue
145: C
146: C
147:      return
148:      end
```

src/mvp/mtsums.f

```

1: *VOCL TOTAL,SCALAR
2: C/#IF PARA(CRAY* SX*)
3: *      subroutine MTSUMS(TITLE, H, IDTASK1, IAINFL1, LIMITL1,
4: *      &      IRANT, NRSTEP, NBATCH1, NPART1,NHIST1, NBANK1, IMPMAX1,
5: *      &      NTHIST1, NFBANK1, NFBNK01, SRCSP,
6: *      &      WSUM, WLEK, XAVT, AAVT, EAVT,
7: *      &      WCNTR, NCNTR, NLOST,
8: *      &      SFLTR,      SFLCL,
9: *      &      SRETR,      SRECL,
10: *      &      XSOC, XSXV, XKEF,
11: *      &      SRMIC, RMICR, WCXTY, XMIC, RMAC, RMACR, XMAC,
12: *      &      SRSTR, SRSCCL,
13: *      &      IETAL, ETALY, TPLHS,
14: *      &      ETRV,
15: *      &      NCLSN, WCLSN, NLEAK, WLEAK,
16: *      &      NABSB, WABSB, NECUT, WECUT,
17: *      &      NTOUT, WTOUT, NKILD, WKILD,
18: *      &      NSURV, WSURV, NSPLT, WSPLT,
19: *      &      XKEFP,
20: *      &      SRMICS, RMACSM,
21: *      &      WCBEF, XBEF )
22: C/#ELSE
23:      subroutine MTSUMS( TITLE, H )
24: C/#ENDIF
25: C=<MVP>=====
26: C PURPOSE: Take summation of data after calculation in multitask mode.
27: C
28: C      In VPP-Fortran, this routine must be called in
29: C      "!xocl parallel region ... !xocl end parallel" block
30: C
31: C CALLED IN: CENTER
32: C CALLS: ACTMPP, ACTVPP, ACTSMP
33: C CALLS: TSKSM*
34: C=====
35:      character TITLE(2)*72
36: CCCC include '../shared/INC/_ARRAY'
37:      real H(*)
38: CCC
39: C
40:      include 'INC/_KPIDS'
41:      include 'INC/_NGPS'
42: C
43:      include 'INC/_FLAGS'
44:      include '../shared/INC/_SIZES'
45:      include 'INC/_SIZES2'
46:      include 'INC/_XBANK'
47:      include 'INC/_SBANK'
48:      include 'INC/_STACK'
49:      include 'INC/_XWORK'
50:      include '../shared/INC/_PGEOM'
51:      include 'INC/_CXSEC'
52:      include 'INC/_PXSEC'
53:      include '../shared/INC/_PVRED'
54:      include 'INC/_PSOUR'
55:      include '../shared/INC/_PTALY0'
56:      include 'INC/_PTALY'
57:      include 'INC/_PTALY2'
58:      include '../shared/INC/_PTLSP'
59:      include '../shared/INC/_STALY'
60: C
61:      include '../shared/INC/_COUNTS'
62:
63: C/#IF PARA(PVM)
64:      include '../shared/INC/_PVMPARA'
65: C/#ELSEIF PARA(MPI)

```

```

66: *      include 'mpif.h'
67: C/#ELSEIF PARA(VPP)
68:      include '../shared/INC/_VPPPARA'
69: C/#ENDIF
70:      include '../shared/INC/_TASKDT'
71:      include '../shared/INC/_TIMEDT'
72:      include '../shared/INC/_IOUNIT'
73:      include 'INC/_IOUNIT2'
74:
75:      include 'INC/_PERT'
76: C
77:      logical      JMEM
78: C
79: C/#IF PARA(CRAY* SX*)
80: *      integer SRCSP(NSRCSP)
81: *      real*8 WSUM,WLEK
82: *      real      XAVT(3),AAVT(3),EAVT
83: *      integer IAINFL1(3)
84: C/# IF INTEGER8
85: *      integer*8 NPART1, NTHIST1
86: C/# ELSE
87: *      integer      NPART1, NTHIST1
88: C/# ENDIF
89: C
90: *      real*8 DT
91: C/#ENDIF
92: C
93: C-----
94: C
95: C/#IF PARA
96: C
97: C/#IF PARA(SX* CRAY*)
98: *      call MVPSYNC_BARRIER( 1 )
99: C/#ENDIF
100: C
101: C/# IF PARA(CRAY* SX*)
102: *      if( IDTASK.EQ.ITASK0 ) goto 678
103: C/# ENDIF
104: C
105: C/#IF PARA(SX* CRAY*)
106: *      call MVPSYNC_LOCK(1)
107: C/#ENDIF
108: C
109: C -----
110: C second argument of TSKSM* is effective only in shared memory
111: C multitask mode ( local data of master task/thread )
112: C -----
113: C
114:      if( JEIGN.eq.0 ) then
115:          call TSKSMI('NBATCH', NBATCH1, NBATCH, 1 )
116:      end if
117: C
118: C/#IF INTEGER8
119: *      call TSKSMI8('NTHIST', NTHIST1, NTHIST, 1 )
120: C/#ELSE
121:      call TSKSMI('NTHIST', NTHIST1, NTHIST, 1 )
122: C/#ENDIF
123: C
124:      call TSKSMD('WSUM', WSUM, H(LWSUM), 1 )
125:      call TSKSMD('WLEK', WLEK, H(LWLEK), 1 )
126:      call TSKSMR('XAVT', XAVT, H(LXAVT), 3 )
127:      call TSKSMR('AAVT', AAVT, H(LAAVT), 3 )
128:      call TSKSMR('EAVT', EAVT, H(LEAVT), 1 )
129:      call TSKSMI('NLOST', NLOST, H(LNLOST), 1 )
130: CC      call TSKSMD('WCNTR', WCNTR, H(LWCNTR), NEVENT*2 )

```

src/mvp/mtsums.f

```

131: CC      call TSKSMD('NCNTR', NCNTR, H(LNCNTR), NEVENT*2 )
132:      call TSKSMD('WCNTR', WCNTR, H(LWCNTR), NEVENT*KPLIM )
133:      call TSKSMD('NCNTR', NCNTR, H(LNCNTR), NEVENT*KPLIM )
134:      call TSKSMD('SFLTR', SFLTR, H(LSFLTR), NGROUP*NTREG*2 )
135:      call TSKSMD('SFLCL', SFLCL, H(LSFLCL), NGROUP*NTREG*2 )
136:      call TSKSMD('SRETR', SRETR, H(LSRETR), NTREG*NRESP*2 )
137:      call TSKSMD('SRECL', SRECL, H(LSRECL), NTREG*NRESP*2 )
138: C
139:      if( NEMIC.ne.0 ) then
140:          call TSKSMD('SRMIC',SRMIC,H(LSRMIC), NGROUP*NTREG*NUC*NEMIC*2)
141:          call TSKSMD('RMICR',RMICR,H(LRMICR), NPKIND*NTREG*NUC*NEMIC*2)
142: C
143:          call TSKSMD('WCXTY',WCXTY,H(LWCXTY), (NGROUP+NPKIND)*NTREG )
144:          call TSKSMD('XMIC', XMIC,H(LXMIC),
145:      &              (NGROUP+NPKIND)*NTREG*NUC*NEMIC*2 )
146:      end if
147: C
148:      if ( NEMAC.gt.0 ) then
149:          call TSKSMD('RMAC', RMAC, H(LRMAC), NGROUP*NTREG*NEMAC*2 )
150:          call TSKSMD('RMACR', RMACR, H(LRMACR), NPKIND*NTREG*NEMAC*2 )
151:          call TSKSMD('XMAC', XMAC, H(LXMAC),
152:      &              (NGROUP+NPKIND)*NTREG*NEMAC*2)
153:      end if
154:
155:      if ( JRESP.ne.0.and.NSTAL.gt.0 ) then
156:          call TSKSMD('SRSTR', SRSTR, H(LSRSTR), NTREG*NSTAL*2 )
157:          call TSKSMD('SRACL', SRACL, H(LSRACL), NTREG*NSTAL*2 )
158:      end if
159: C
160:      if( NSTALY.gt.0 ) then
161:          call TSKSMD('ETALY',ETALY,H(LETALY), NLETAL*2)
162: C
163:          if ( NETRV.gt.0 ) then
164:              call TSKSMD('ETRV', ETRV, H(LETRV), METRV*(NBATCH-NSKIP) )
165:          end if
166:      end if
167: C
168:      if( JTIME.gt.0 ) then
169:      CCCCC      call TSKSMD('TFLHS',TFLHS,H(LTFLHS), NPKIND*2)
170:          call TSKSMD('TFLHS',TFLHS,H(LTFLHS), KPLIM*2)
171:      end if
172: C
173:      if( JEIGN.ne.0 ) then
174:          call TSKSMD('XSOC', XSOC, H(LXSOC), NBATCH )
175:          call TSKSMD('XSXV', XSXV, H(LXSXV), NLENG*3 )
176:      c##<2007/03/14:PN4:
177:      c##      call TSKSMD('XKEF', XKEF, H(LXKEF), 7*NBATCH )
178:          call TSKSMD('XKEF', XKEF, H(LXKEF), 10*NBATCH )
179:      c##>
180:      end if
181: C
182:      if( JMNTR.ne.0 ) then
183:          call TSKSMD('NCLSN', NCLSN, H(LNCLSN), NGROUP*NREG )
184:          call TSKSMD('WCLSN', WCLSN, H(LWCLSN), NGROUP*NREG )
185:          call TSKSMD('NLEAK', NLEAK, H(LNLEAK), NGROUP*NREG )
186:          call TSKSMD('WLEAK', WLEAK, H(LWLEAK), NGROUP*NREG )
187:          call TSKSMD('NABSB', NABSB, H(LNABSB), NGROUP*NREG )
188:          call TSKSMD('WABSB', WABSB, H(LWABSB), NGROUP*NREG )
189:          call TSKSMD('NECUT', NECUT, H(LNECUT), NREG )
190:          call TSKSMD('WECUT', WECUT, H(LWECUT), NREG )
191:          call TSKSMD('NTCUT', NTCUT, H(LNTCUT), NGROUP*NREG )
192:          call TSKSMD('WTCUT', WTCUT, H(LWTCUT), NGROUP*NREG )
193:          call TSKSMD('NKILD', NKILD, H(LNKILD), NGROUP*NREG )
194:          call TSKSMD('WKILD', WKILD, H(LWKILD), NGROUP*NREG )
195:          call TSKSMD('NSURV', NSURV, H(LNSURV), NGROUP*NREG )
196:
197:          call TSKSMD('WSURV', WSURV, H(LWSURV), NGROUP*NREG )
198:          call TSKSMD('NSPLT', NSPLT, H(LNSPLT), NGROUP*NREG )
199:          call TSKSMD('WSPLT', WSPLT, H(LWSPLT), NGROUP*NREG )
200:      end if
201:      if( JEIGN.ne.0 .and. JPERT.ne.0 ) then
202:          call TSKSMD('XKEFP', XKEFP, H(LXKEFP), 7*NBATCH*NTPT*NUCPT )
203:      end if
204:      c+beffl
205:      if( JEIGN.ne.0 .and. JBEFF.ne.0 ) then
206:          call TSKSMD('WCBEF', WCBEF, H(LWCBEF), NEBEF )
207:          call TSKSMD('XBEF', XBEF, H(LXBEF), 7*NBATCH )
208:      end if
209:      c-beffl
210:
211:      if ( JSCTM.ne.0 ) then
212:          if ( NEMIC.ne.0 ) then
213:              KY = 0
214:              if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) KY = KY + 1
215:              if ( JMICE(6).ne.0.or.JMACE(6).ne.0 ) KY = KY + 1
216:              if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) KY = KY + 1
217:              call TSKSMD('SRMICSM', SRMICSM, H(LSRMICSM),
218:      &                  ((NGP(KPNEUT)+1)**2)*NTREG*NUC*KY*2 )
219:          end if
220:          if ( NEMAC.gt.0 ) then
221:              KY = 0
222:              if ( JMACE(4).ne.0 ) KY = KY + 1
223:              if ( JMACE(6).ne.0 ) KY = KY + 1
224:              if ( JMACE(7).ne.0 ) KY = KY + 1
225:              call TSKSMD('RMACSM', RMACSM, H(LRMACSM),
226:      &                  ((NGP(KPNEUT)+1)**2)*NTREG*KY*2 )
227:          end if
228:      end if
229: C
230: C
231: C/IF PARA(SX* CRAY*)
232: *      call MVPSYNC_UNLOCK(1)
233: C/#ENDIF
234: C
235:      678 continue
236: C/IF PARA(PVM MPI)
237: *      call TOKEI(TE2,0)
238: *      write(IOW,'(/lx,a,i5,a)')
239: *      & 'Task:',IDTASK,' ELAPSED TIME IN "MTSUMS" =='
240: *      write(IOW,'(lx,a,e12.5)') ' BEFORE TASK SUM ',TE1-TE0
241: *      write(IOW,'(lx,a,e12.5)') ' TASK SUM ',TE2-TE1
242: *      write(IOW,'(lx,a,e12.5)') ' TOTAL ',TE2-TE0
243: C/#ENDIF
244: C
245: C/# ENDIF
246: C
247:      return
248:      end

```

src/mvp/mttask.f

```

1:      subroutine MTTASK( TITLE, A, H, CHA, IAINFL, LIMITL,
2:      &
3:      &
4:      &
5:      C/#!/IF PARA( CRAY* SX* )
6:
7: C=====
8: C purpose: to control multi processing of 'ACTION' routine in MVP code.
9: C           (for shared memory machine such as CRAY MP series or SX3 )
10: C called in : CENTER
11: C=====
12:      integer ITASK(MNTASK,NTASK)
13:      integer IRANT(NTASK)
14:      integer ILOCK(*)
15:      integer IBARR(*)
16:
17:      integer IRSUT(NTASK), IRSMT(NTASK), IRSLT(NTASK)
18: C
19:      character*72 TITLE(2)
20: C
21: CCCC include '../shared/INC/_ARRAY'
22:      integer IAINFL(3), LIMITL
23:      real A(*)
24:      real H(*)
25:      character*4 CHA(*)
26: C
27:      include '../shared/INC/_SIZES'
28:      include 'INC/_FLAGS'
29:      include 'INC/_PSOUR'
30:      include '../shared/INC/_IOUNIT'
31:      include '../shared/INC/_PTALY0'
32:      include 'INC/_PTALY'
33:      include 'INC/_PTALY2'
34:      include '../shared/INC/_STALY'
35:
36:      include '../shared/INC/_RAND'
37:
38:      include 'INC/_PERT'
39: C
40:      external ACTSMP
41: C
42: C
43: C
44:      NRSTEP = NTASK
45:      if( mod(ntask,2) .eq. 0 ) NRSTEP = NRSTEP + 1
46: C
47: C
48: C
49: C-----
50: C assign lock and barrier
51: C-----
52: C
53:
54:      CALL MVPSYNC_ASSIGN_LOCK( 1, IERR )
55:      CALL MVPSYNC_ASSIGN_LOCK( 2, IERR )
56:      if( JREPR.eq.0 ) CALL MVPSYNC_ASSIGN_LOCK( 3, IERR )
57:
58:      call MVPSYNC_ASSIGN_BARRIER( 1, IERR )
59:
60:      do 11 I=1,NTASK
61:          call MVPSYNC_ASSIGN_EVENT( I, IERR )
62:      11 continue
63: C
64:      call CPUTM( TC0 )
65:      call TOKEI( TE0, 0 )

```

```

66: C
67: C-----
68: C ... Fork sub tasks .....
69: C-----
70: C
71:      ITASK0 = 1
72:      IRAN0 = IRAND
73:      IRSU0 = IRNSU
74:      IRSM0 = IRNSM
75:      IRSLO = IRNSL
76:
77:      do 110 N = 2, NTASK
78: C
79: C ... skip random number
80: C
81:          call RANU2( IRAND, RTEMP, 1, ICOD)
82:
83:          IRANT(N) = IRAND
84:          IRSUT(N) = IRNSU
85:          IRSMT(N) = IRNSM
86:          IRSLT(N) = IRNSL
87:      110 continue
88:
89:      do 100 N = 2, NTASK
90: C
91: C ... Pass address of tally data of main task as arguments.
92: C
93: C
94: C/#!/IF PARA( SX* )
95:      IK = 2
96:      ITASK(IK,N) = N
97:      call PTFORK( ITASK(1,N), ITASK(2,N),
98: C/#!/ELSEIF PARA( CRAY* )
99:      *
100:      * IK = 3
101:      * ITASK(1,N) = 3
102:      * ITASK(IK,N) = N
103:      * call TSKSTART( ITASK(1,N),
104: C/#!/ENDIF
105:      &
106:      & ACTSMP, TITLE,
107:      & ITASK(IK,N), IAINFL, LIMITL,
108:      & IRANT(N), NRSTEP, NBATCH, NPART, NHIST, NHSUB, NBANK,
109:      & IMPMAX, NTHIST, NFBANK,NFBNK0, H(LSRCSP),
110:      & H(LWSUM), H(LWLEK), H(LXAVT),H(LAAVT),H(LEAVT),
111:      & H(LWCNTR), H(LNCNTR), H(LNLOST),
112:      & H(LSFLTR), H(LSFLCL),
113:      & H(LSRETR), H(LSRECL),
114:      & H(LXSOC), H(LXSXV), H(LXKEF),
115:      & H(LSRMIC), H(LRMICR), H(LWCXTY), H(LXMIC),
116:      & H(LRMAC), H(LRMACR), H(LXMAC),
117:      & H(LSRSTR), H(LSRSCCL),
118:      & H(LIETAL), H(LETALY), H(LTFLHS), H(LETRV),
119:      & h(lNCLSN), h(lWCLSN), h(lNLEAK), h(lWLEAK),
120:      & h(lNABSB), h(lWABSB), h(lNECUT), h(lWECUT),
121:      & h(lNLCUT), h(lWLCUT), h(lNKILD), h(lWKILD),
122:      & h(lNSURV), h(lWSURV), h(lNSPLT), h(lWSPLT),
123:      & IRSUT(N), IRSMT(N), IRSLT(N),
124:      & H(LXKEFP),
125:      & H(LSRMICSM), H(LRMACSM),
126:      & H(LWCBEF), H(LXBEB) )
127: C
128: C-----
129: C ... start main task
130: C-----

```

src/mvp/mttask.f

```
131: C
132:       N = 1
133: C/#!/IF PARA( SX* )
134:       ITASK(2,N) = ITASK0
135: C/#!/ELSEIF PARA( CRAY* )
136:       ITASK(3,N) = ITASK0
137: C/#!/ENDIF
138:       IRANT(N) = IRANO
139:       IRSUT(N) = IRSU0
140:       IRSMT(N) = IRSM0
141:       IRSLT(N) = IRSLO
142: C
143:       call ACTSMP( TITLE,
144: &       ITASK0, IAINFL, LIMITL,
145: &       IRANT(N), NRSTEP, NBATCH, NPART, NHIST, NHSUB, NBANK,
146: &       IMPMAX, NTHIST, NFBANK, NFBNK0, H(LSRCSP),
147: &       H(LWSUM), H(LWLEK), H(LXAVT), H(LAAVT), H(LEAVT),
148: &       H(LWCNTR), H(LNCNTR), H(LNLOST),
149: &       H(LSFLTR), H(LSFLCL),
150: &       H(LSRETR), H(LSRECL),
151: &       H(LXSOC), H(LXSXV), H(LXKEF),
152: &       H(LSRMIC), H(LRMICR), H(LWCXTY), H(LXMIC),
153: &       H(LRMAC), H(LRMACR), H(LXMAC),
154: &       H(LSRSTR), H(LSRACL),
155: &       H(LIETAL), H(LETALY), H(LTFLHS), H(LETRV),
156: &       h(1NCLSN), h(1WCLSN), h(1NLEAK), h(1WLEAK),
157: &       h(1NABSB), h(1WABSB), h(1NECUT), h(1WECUT),
158: &       h(1NTCUT), h(1WTCUT), h(1NKILD), h(1WKILD),
159: &       h(1NSURV), h(1WSURV), h(1NSPLT), h(1WSPLT),
160: &       IRSUT(N), IRSMT(N), IRSLT(N),
161: &       H(LXKEFP),
162: &       H(LSRMICSM), H(LRMACSM),
163: &       H(LWCBEF), H(LXBEF) )
164: C
165: C-----
166: C      ... wait termination of subtask
167: C-----
168: C
169:       do 200 N=2,NTASK
170: C/#!/IF PARA( SX* )
171:       call PTJOIN( ITASK(1,N) )
172: C/#!/ELSEIF PARA( CRAY* )
173:       call TSKWAIT( ITASK(1,N) )
174: C/#!/ENDIF
175:       200 continue
176: C
177: C
178:       call CPUTM( TC1 )
179:       call TOKEI( TE1, 0 )
180: C
181: C
182:       write(IOW,7200) TE1-TE0, TC1-TC0
183: c##<2007/03/14:PN3:
184: c7200 format(/1x,' ==== TIME MONITOR OF MULITTASKING ==='/
185: 7200 format(/1x,' ==== TIME MONITOR OF MULITTASKING ===',1p/
186: c##>
187: &       1x,'      TOTAL CPU TIME      ',e12.5,' (SEC)'/
188: &       1x,'      TOTAL ELAPSED TIME   ',e12.5,' (SEC)'/)
189: C/#!/ENDIF
190:       return
191:       end
```

src/mvp/natomz.f

```

1:      function NATOMZ(NUCID)
2: C=<MVP>=====
3: C  PURPOSE :  RETURN ATOMIC NUMBER FOR A NUCLIDE-ID
4: C  CALLED IN:
5: C=====
6:      character*(*) NUCID
7: C
8: C  ---- CHEMICAL SYMBOLS IN ORDER OF ATOMIMC NUMBER ----
9: C
10:     parameter( MATOM      = 103 )
11:     character*2 CSYMBL(MATOM)
12: C
13:     data (CSYMBL(I),I=1,MATOM) /
14:     1 'H ',
15:     2 'LI','BE',
16:     3 'NA','MG',
17:     4 'K ', 'CA', 'SC', 'TI', 'V ', 'CR', 'MN', 'FE', 'CO', 'NI', 'CU', 'ZN',
18:     &      'GA', 'GE', 'AS', 'SE', 'BR', 'KR',
19:     5 'RB', 'SR', 'Y ', 'ZR', 'NB', 'MO', 'TC', 'RU', 'RH', 'PD', 'AG', 'CD',
20:     &      'IN', 'SN', 'SB', 'TE', 'I ', 'XE',
21:     6 'CS', 'BA',
22:     7 'LA', 'CE', 'PR', 'ND', 'PM', 'SM', 'EU', 'GD', 'TB', 'DY', 'HO',
23:     8 'ER', 'TM', 'YB', 'LU',
24:     &      'HF', 'TA', 'W ', 'RE', 'OS', 'IR', 'PT', 'AU', 'HG',
25:     &      'TL', 'PB', 'BI', 'PO', 'AT', 'RN',
26:     9 'FR', 'RA',
27:     A  'AC', 'TH', 'PA', 'U ', 'NP', 'PU', 'AM', 'CM', 'BK', 'CF', 'ES',
28:     A  'FM', 'MD', 'NO', 'ER' /
29: C
30: C
31:     K      = 0
32:     if( len(NUCID).gt.1 ) then
33:         K      = INDEX('ABCDEFGHIJKLMNOPQRSTUVWXYZ',NUCID(2:2))
34:         K2     = INDEX('abcdefghijklmnopqrstuvwxy',NUCID(2:2))
35:         K      = max(K,K2)
36:     end if
37: C
38: C  .... 2 CHARACTER SYMBOLS ....
39: C
40:     if ( K.gt.0 ) then
41:         do 100 I = 1, MATOM
42:             if ( CSYMBL(I)(1:2).eq.NUCID(1:2) ) then
43:                 NATOMZ = I
44:                 return
45:             end if
46:         100 continue
47: C
48: C  .... 1 CHARACTER SYMBOLS ....
49: C
50:     else
51: CCCCCC if ( NUCID(1:1).eq.'H' .or. NUCID(2:2).eq.'D' ) then
52:         if ( NUCID(1:1).eq.'H' .or. NUCID(1:1).eq.'D' .or.
53:             &      NUCID(1:1).eq.'T' ) then
54:             NATOMZ = 1
55:             return
56:         end if
57:         do 110 I = 1, MATOM
58:             if ( CSYMBL(I)(2:2).eq.' ' .and. CSYMBL(I)(1:1).eq.NUCID(1:1)
59:                 &      ) then
60:                 NATOMZ = I
61:                 return
62:             end if
63:         110 continue
64:     end if
65: C
66: C  .... INVALID NAME ? ....
67: C
68:     NATOMZ = 0
69:     return
70:     end

```

src/mvp/neesel.f

```

1:      subroutine NEESEL( MODE, IEVENT,MACT, MZONE, NIMPT, NDIMPT,
2:      &                IMNFL, IMNNX, IMDEFR,IMNLT )
3: C=====
4: C  PURPOSE: SELECTION OF EVENT FOR NEXT EVENT ESTIMATOR.
5: C  CALLED IN: NXTEE
6: C  CALLS:
7: C
8: C=====
9: C
10: C  IEVENT : COUNTER OF EVENT PROCESSING IN A CALL OF THE NXTEE ROUTINE.
11: C  MACT : ACTION NUMBER
12: C  MZONE : SELECTED ZONE # (FOR ZONE-SELECTION TASKS)
13: C
14: C  OPERATION MODE :
15: C    MODE = 1 : PROCESS ALL IMAGINARY PARTICLES
16: C              UNTIL FINISHED.
17: C    = 2 : PROCESS IMAGINARY PARTICLES UNTIL
18: C          ENTRY OF BANK IS LESS THAN 'NTHIN' PARTICLES.
19: C    = -N      : PROCESS N EVENTS.
20: C=====
21:      include '../shared/INC/_SIZES'
22:      include 'INC/_SIZES2'
23:      include 'INC/_FLAGS'
24: C
25:      integer IMNFL(NZONE+1), IMNNX(NZONE+1), IMNLT(NLBZ+1)
26: C
27:      IEVENT = IEVENT + 1
28: C
29: C    .... LIMITED COUNT OF OPERATIONS ( MODE < 0, -MODE OPERATIONS )
30: C
31:      if ( MODE.lt.0.and.IEVENT.gt.-MODE ) then
32:        MACT = 99
33:        return
34:      end if
35: C
36: C    .... DEFER NEXT EVENT ESTIMATION IF NUMBER OF IMAGINARY PARTICLES
37: C          FALLS BELOW A LIMIT. ( IMPMAX * SUPPLY NOW )
38: C
39:      if ( MODE.eq.2.and.NIMPT.le.SUPPLY*IMPMAX ) then
40:        MACT = 99
41:        return
42:      end if
43: C
44: C    .... FINISH IF NO PARTICLES IN BANK ....
45: C
46:      if ( NIMPT.eq.0 ) then
47:        MACT = 99
48:        return
49:      end if
50: C
51: C    .... SELECT NEXT ACTION ....
52: C
53: C    ===== FREE FLIGHT =====
54:      MAX1 = 0
55:      MZONE1 = 0
56:      if ( IMNFL(NZONE+1).gt.0 ) then
57:        do 100 N = 1, NZONE
58:          if ( IMNFL(N).gt.MAX1 ) then
59:            MAX1 = IMNFL(N)
60:            MZONE1 = N
61:          end if
62: 100      continue
63:      end if
64: C
65: C    ===== NEXT-ZONE SEARCH =====
66: C
67:      MAX2 = 0
68:      NZONE2 = 0
69:      if ( IMNNX(NZONE+1).gt.0 ) then
70:        do 110 N = 1, NZONE
71:          if ( IMNNX(N).gt.MAX2 ) then
72:            MAX2 = IMNNX(N)
73:            MZONE2 = N
74:          end if
75: 110      continue
76:      end if
77:      MAX3 = IMDEFR
78: C
79: C
80: C
81:      if ( JLATT.ne.0 ) then
82:        MAX6 = 0
83:        if ( IMNLT(NLBZ+1).gt.(IMPMAX
84:      &      -NDIMPT)*0.5 .or. MAX1.eq.0.and.MAX2.eq.0 ) then
85:          MAX6 = IMNLT(NLBZ+1)
86:        end if
87:        if ( MAX6.gt.MAX(MAX1,MAX2,MAX3) ) then
88:          MACT = 6
89:          MZONE = 0
90:          return
91:        end if
92:      end if
93: C
94: C    .... SELECT FLIGHT .....
95: C
96:      MMAX = MAX(MAX1,MAX2,MAX3)
97:      if ( MAX1.eq.MMAX ) then
98:        MACT = 1
99:        MZONE = MZONE1
100: C
101: C    .... SELECT SEARCH .....
102: C
103:      else if ( MAX2.eq.MMAX ) then
104:        MACT = 2
105:        MZONE = MZONE2
106: C
107: C    .... SELECT DEFERRED PARTICLE PROCESSING FOR SEARCH ....
108: C
109:      else if ( MAX3.eq.MMAX ) then
110:        MACT = 2
111:        MZONE = -1
112:      end if
113:      return
114: C
115:      end

```

src/mvp/neutr.f

```

1: C/#IF ARGSAVE
2: * subroutine NEUTRO( IOW, NPKIND,NGROUP,NZONE, NMAT, NREG,NTIME,
3: * N NRESP, NBANK,NFBANK,NFBNK0,BANKP,
4: * N NEVENT, NEST,NUC,NNK,NMT,MB,NSMAC,NSMIC,NSTAL,MCX,
5: * G KZMAT,KZREG,
6: * 6 XIMP,WKIL,WSRV,WGTF,WGTFI,WGTP,WTIME,WLLIM,MXPGEN,RESP,
7: * 7 ETOP, EBOT, ETHMAX, AMLIM, EWCUT, ENGYB, NNCST, INCST,
8: * 8 CX, KLIB1, XLIB1, KLIB2, KLB2S, XLIB2, SMAC, LMAC,
9: * 9 KMAC, MMAC, SMIC, LMIC, KSPI, SGTAL, IRAND,
10: * P NPATOM, NMTP, ETOPP, EBOTP,
11: * P CXP, MCXP, KLB1, KLB2, XLBP1, XLBP2,
12: c##<2007/03/14:PN3:PN4:
13: c##* B DNZON, LEMIC, NEMIC, CRES, KCRES, MCRES, NMKREG, MKREG
14: * B DNZON, LEMIC, NEMIC, CRES, KCRES, MCRES, NMKREG, MKREG,
15: * & NUC_MAX, TCUT, TIMEB, NNCSTPN, INCSTPN, NMTPN, MTMPN, NSTALY,
16: * & NUCPN, NUCPNI, NMTPN, NSMICPN, MCXPN, CXP, KLBPN1, KLBPN2,
17: * & XLBP1, SMICPN, EBOTPN, KPNPRD, KPNFG
18: c##>
19: * B )
20: C/#ELSE
21: subroutine NEUTR ( IOW, NPKIND,NGROUP,NZONE, NMAT, NREG,NTIME,
22: N NRESP, NBANK,NFBANK,NFBNK0,BANKP,
23: N NEVENT, NEST,NUC,NNK,NMT,MB,NSMAC,NSMIC,NSTAL,MCX,
24: G KZMAT,KZREG,
25: 6 XIMP,WKIL,WSRV,WGTF,WGTFI,WGTP,WTIME,WLLIM,MXPGEN,RESP,
26: 7 ETOP, EBOT, ETHMAX, AMLIM, EWCUT, ENGYB, NNCST, INCST,
27: 8 CX, KLIB1, XLIB1, KLIB2, KLB2S, XLIB2, SMAC, LMAC,
28: 9 KMAC, MMAC, SMIC, LMIC, KSPI, SGTAL, IRAND,
29: P NPATOM, NMTP, ETOPP, EBOTP,
30: P CXP, MCXP, KLB1, KLB2, XLBP1, XLBP2,
31: B DNZON, LEMIC, NEMIC, CRES, KCRES, MCRES, NMKREG, MKREG,
32: & LSFFL, NFFL, IZFFL, LSCOL, NCOLS, LSEDED, NDEAD, NFRIB,
33: 3 XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE,
34: 4 TTT,ITT, XIM, KLSF,KKP,IZZ, IGG, LEVL, LZZ, LPOS, LCRS,
35: 5 XXXF, YYF, ZZZF, EEFF, INUF, IZZF, LEVLF, LZZF
36: 6 LPOSF, LCRSF, IBREG, IBSPC, IBRGF, IBSPF,
37: B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
38: B DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
39: & FLCL, RECL, NCLSN, WCLSN, NABSB, WABSB,
40: 7 NECUT, WECUT, NKILD, WKILD, NSURV, WSURV, NSPLT, WSPLT,
41: 8 NCNTR, WCNTR, RMIC, DNFLX, RSCL,
42: & KYPOS, DNFLXSM, RMICSM, RMIMU, WMIMU, ! scat-mtx
43: T JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY, RNU,
44: W IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
45: W IWK7, IWK8, IWK9, IWK10, IWK11, IWK12,
46: W IWK13, IWK14, IWK15, IWK16, IWK17, IWK18, IWK19,
47: W DWK1, DWK2, WK3, WK4, WK5, R, SMCW,
48: & SMCWP, E2P, DEP, RP, ! pert
49: & CXPI, SMICP, SMACP, ! pert
50: & NPSTD, WWD, WD0, WCTRP, WD0A, WD0B, NBATCH, NSKIP, ! pert
51: & JPTTR, JPTNU, WWD2, NPTCS, WWR, WR0, WR0A, MAXDA, DELA, ! pert
52: & WWT, WT0, WTA, WT0B, ! pert
53: & NUCPT, MNUC, LPDEN, INUCT, DENST, WKPT, WKPD, WKPN, ! pert
54: & NGSP, WSD, WSC, NOCS, NTPT, NORDD, NOPS, ! pert
55: & IMTF, NEBEF, WCBF, WK6, WK7, WK8, ! beff
56: & NUC_MAX, TCUT, TIMEB, NNCSTPN, INCSTPN, NMTPN, MTMPN, NSTALY, ! phot
o-nuc
57: & NUCPN, NUCPNI, NMTPN, NSMICPN, MCXPN, CXP, KLBPN1, KLBPN2, ! phot
o-nuc
58: & XLBP1, SMICPN, EBOTPN, KPNPRD, KPNFG, DWK3, ! phot
o-nuc
59: & WWB, WB0, WSB, NPTBE, WWBD, WBD0, WSD, WLD0, WSLD, ! beff
60: & IOTL) ! time-list
61: C/#ENDIF
62: C=====

```

```

63: C PURPOSE: CONTROL OF COLLISION ANALYSIS
64: C (ENTRY NEUTR)
65: C CALLED IN: ACTION
66: C CALLS: TALLY,RANU2,FISGEN,GAMGEN,NTHSCT,THSCT,FREGAS,PANLGF,SPLITB,
67: C GETMIC,GTMICP,GTMICPN,GMICN1,GMICP1,GMICPN1,WWDPBS
68: C
69: C=====
70: implicit real*8(A-H,O-Z)
71: C
72: include 'INC/_KPIDS'
73: include 'INC/_NGPS'
74: include 'INC/_FLAGS'
75: C
76: C**** GEOMETRY ARRAY DATA
77: C
78: integer KZMAT(NZONE), KZREG(NZONE)
79: C
80: C**** VARIANCE REDUCTION DATA
81: C
82: real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
83: c##<2007/03/14:PN3:
84: c## real WGTF(NREG), WGTFI(NREG)
85: real WGTF(NREG,2), WGTFI(NREG), TIMEB(NTIME+1), TCUT
86: c##>
87: real WTIME(NTIME)
88: real WLLIM
89: C
90: C**** CROSS SECTION DATA IN CX ARRAY
91: C
92: real ETOP, EBOT, ETHMAX, AMLIM, EWCUT, ETOPP, EBOTP
93: c##<2007/03/14:PN3:
94: real EBOTLPN
95: c##>
96: real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
97: c##<2007/03/14:PN3:
98: c## integer KLIB1(NUC,27), KLIB2(NUC,NMT,21), KLB2S(NUC,NNK,NMT,3)
99: integer KLIB1(NUC,31), KLIB2(NUC,NMT,21), KLB2S(NUC,NNK,NMT,3)
100: c##>
101: C
102: real CRES(MCRES)
103: integer KCRES(NSTAL,4)
104: C
105: C**** SIGMA BANK
106: C
107: real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
108: & SGTAL(NBANK,NSTAL)
109: integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8),
110: & LMAC(8), LEMIC(16)
111: c##<2007/03/14:PN3:
112: c## real RNU(NBANK,NUC,3)
113: real RNU(NBANK,NUC_MAX,3)
114: c##>
115: C
116: C**** STACK
117: C
118: integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE+1), LSCOL(NBANK),
119: & LSEDED(NBANK)
120: C
121: C**** PARTICLE BANK
122: C
123: real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
124: real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
125: integer KKP(NBANK)
126: integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
127: & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)

```


src/mvp/neutr.f

```

128:      real EEE(NBANK), WWW(NBANK), XIM(NBANK)
129:      real*8 TTT(NBANK)
130:      integer ITT(NBANK)
131:      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
132: C
133:      real*8 DBNK(NBANK,*)
134:      integer KDBNK(0:MDBNK)
135:      integer IBNK(NBANK,*)
136:      integer KIBNK(0:MIBNK)
137:      integer KPNFG(NBANK) ! photonuc
138: C
139: C**** Fission bank
140: C
141:      real*8 XXXF(NFBNK0), YYF(NFBNK0), ZZZF(NFBNK0)
142:      integer IZZF(NFBNK0), INUF(NFBNK0), LEVLF(NFBNK0),
143: &      LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST)
144:      real EEEF(NFBNK0)
145:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
146:      integer IMTF(NFBNK0) ! beff
147: C
148:      real*8 DFBK(NFBNK0,*)
149:      integer KDFBK(0:MDFBK)
150:      integer IFBK(NFBNK0,*)
151:      integer KIFBK(0:MIFBK)
152: C
153: C**** TALLY ARRAY
154: C
155:      real ENGYB(*)
156: CCC      real ENGPB(*)
157: C
158:      integer INCST(NUC,*)
159: c##<2007/03/14:PN3:
160:      integer INCSTPN(NSTALY,2), MTMPN(NMTMPN)
161: c##>
162: C
163:      integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
164:      integer IDTAL(*)
165:      real*8 DTALY(*)
166: C
167:      real RESP(NGROUP,NRESP), DNZON(NUC,NZONE,2)
168:      real*8 FLCL(NGROUP,NREG), RECL(NREG,NRESP),
169: &      RMIC(NGROUP,NREG,NUC,NEMIC), RSCL(NREG,NSTAL),
170: &      DNFLX(NGROUP,NREG,NUC)
171: CCC      real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
172:      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
173:      real*8 WCLSN(NGROUP,NREG), WABSB(NGROUP,NREG), WECUT(NREG),
174: &      WKILD(NGROUP,NREG), WSURV(NGROUP,NREG),
175: &      WSPLT(NGROUP,NREG), NCLSN(NGROUP,NREG),
176: &      NABSB(NGROUP,NREG), NECUT(NREG), NKILD(NGROUP,NREG),
177: &      NSURV(NGROUP,NREG), NSPLT(NGROUP,NREG)
178:      integer MKREG(NREG,2)
179: cc%      real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*),
180: cc% &      RMICSM(NGP1,NGP1,NREG,NUC,*)
181:      real*8 DNFLXSM(*),RMICSM(*)
182:      integer KYPOS(3)
183:      real*8 RMIMU(*), WMIMU(*) ! scat-mtx
184: C
185: C**** WORKING AREA
186: C
187:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
188: &      IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
189: &      IWK9(NBANK), IWK10(NBANK), IWK11(NBANK), IWK12(NBANK),
190: &      IWK13(NBANK), IWK14(NBANK), IWK15(NBANK), IWK16(NBANK),
191: &      IWK17(NBANK), IWK18(NBANK), IWK19(NBANK)
192:      real*8 DWK1(NBANK), DWK2(NBANK)
193:      real*8 DWK3(NBANK) ! photonuc, time-list
194: CM2014real WK3(NBANK), WK4(NBANK), WK5(NBANK), R(8*NBANK)
195:      real WK3(NBANK), WK4(NBANK), WK5(NBANK), R(9*NBANK)
196:      real WK6(NBANK), WK7(NBANK), WK8(NBANK) ! beff
197:      real SMCW(NBANK,NSMIC)
198: C
199: C**** PERTURBATION ARRAYS ***
200: C
201: c ... cross section data in CX array
202: c
203:      real CXPl(MCX)
204: c
205: c ... sigma bank
206: c
207:      real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
208: c
209: c ... working area
210: c
211:      real SMCWP(NBANK,NSMIC)
212:      real*8 E2P(NBANK)
213:      real RP(8*NBANK)
214:      integer DEP(NBANK)
215: C
216: C ... variables for perturbation calculation ...
217: C
218:      real*8 WWD(NBANK,NPTDS,NORDDS), WWD2(NBANK,NPTDS)
219:      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
220:      real*8 WWR(NBANK,NPTCS)
221:      real*8 WSC(NBANK,0:NGSP,NPTCS)
222:      real*8 WD0(NFBNK0,0:NGSP,NPTDS,NOPSDS)
223:      real*8 WCTRP(NUCPT,7,NTPT)
224:      real*8 WR0(NFBNK0,0:NGSP,NPTCS)
225:      real*8 WR0A(NFBNK0)
226:      integer JPTTR(NREG,NPTDS+NPTCS),JPTNU(NUC,NPTDS)
227:      real DELA(MAXDA,NPTDS+NPTCS)
228:      real*8 WWT(NBANK,NPTDS),WT0(NFBNK0)
229:      real*8 WTA0(NFBNK0),WTOB(NFBNK0)
230: C
231:      real*8 WWB(NBANK) ! beff
232:      real*8 WB0(NFBNK0,0:NGSP) ! beff
233:      real*8 WSB(NBANK,0:NGSP) ! beff
234:      real*8 WWBD(NBANK) ! beff
235:      real*8 WBD0(NFBNK0,NGSP) ! beff
236:      real*8 WSBD(NBANK,NGSP) ! beff
237:      real*8 WWLD(NBANK) ! beff
238:      real*8 WLD0(NFBNK0,NGSP) ! beff
239:      real*8 WSLD(NBANK,NGSP) ! beff
240: c ... working area ...
241:      real*8 WKPT(NBANK,NPTDS)
242:      real*8 WKPD(NBANK,NPTDS)
243:      real*8 WKPN(NUCPT,NBANK,NPTDS)
244: C
245: C ... material data
246: C
247:      real DENST(*)
248:      integer MNUC(NMAT), INUCT(*), LPDEN(NMAT+1)
249: C
250: C**** END OF PERTURBATION ARRAYS ***
251: C
252:      real*8 WCBEF(NEBEF) ! beff
253: C
254: C
255: C
256:      parameter( M102 = 102, M114 = 114 )
257:      parameter( PI = 3.1415926535897932D+00 )

```

src/mvp/neutr.f

```

258:      parameter( PI2 = 2.0D0*PI )
259:      parameter( PIH = 0.5D0*PI )
260: C
261: C=====
262: C
263: C  ETOP      R4    UPPER ENERGY LIMIT (EV)
264: C  EBOT      R4    LOWER ENERGY LIMIT (EV)
265: C  ETHMAX    R4    UPPER ENERGY LIMIT FOR THERMAL SCATTERING (EV)
266: C  AMLIM     R4    UPPER MASS LIMIT FOR FREE-GAS TREATMENT
267: C  EWCUT     R4    THRESHOULD FOR ANALOG ASSORPTION (EV)
268: C          E < EWCUT : ANALOG ABSORPTION
269: C          E >= EWCUT : WEIGHT REDUCTION
270: C  ETOPL     R4    UPPER ENERGY LIMIT IN LIBRARY ( MINIMUM OF EHI )
271: C  EBOPL     R4    LOWER ENERGY LIMIT IN LIBRARY ( MAXIMUM OF ELOW )
272: C  ESABL     R4    UPPER ENERGY LIMIT FOR S(A,B) ( MAXIMUM OF EAB )
273: C
274: C  KLIB1(NUC,21)      I4    RECORD #1 (SPECIFICATION RECORD)
275: C
276: C      1:  STARTING POSITION OF DATA FOR A NUCLIDE
277: C      2:  NPIS      3:  NDATA      4:  NUNR      5:  NUNR2
278: C      6:  NLEV      7:  NBINA      8:  NBINE      9:  NNU      10:  NMT
279: C      11:  NNK      12:  IGFLG      13:  LFI      14:  LNU      15:  ITERM
280: C      16:  ISTU      17:  IENDU      18:  LUNK
281: C      19:  LSNU (START POSITION OF NU-VALUE)
282: C      20:  NBINA1 = NBINA + 1      21:  NBINE1 = NBINE + 1
283: C
284: C      ( NMT = 120,  NNK = 10      :  FIXED )
285: C
286: C  XLIB1(NUC,11)      R4    NUCLIDE PARAMETERS
287: C
288: C      1:  ATW      2:  TLAB      3:  ETHEM      4:  ESAB      5:  ELOW
289: C      6:  EHI      7:  1/(ATW+1)**2      8:  1/(ATW+1)      9:  ATW/(K*TLAB)
290: C      10:  TEFF      11:  ATW/(K*TEFF)
291: C
292: C  KLIB2(NUC,NMT,13)  I4    POINTERS TO #2 RECORD DATA      (1)
293: C
294: C      1:  MTINFO      2:  MTPAR      3:  ISTMT      4:  IENDMT      5:  NEUMT
295: C      6:  LCTMT      7:  NEANG      8:  NKF5      9:  NEF5      10:  MTHR
296: C      11:  LPS      12:  LANG      13:  LED
297: C
298: C  KLB2S(NUC,NNK,NMT,3) I4    POINTERS TO #2 RECORD DATA      (2)
299: C
300: C      1:  NEF5S      2:  LFF5S      3:  LSTF5S
301: C
302: C  XLIB2(NUC,NMT,2)      R4    Q-VALUE
303: C
304: C      1:  QVAL      2:  (ATW+1)/ATW*QVAL
305: C
306: C
307: C  CX(*)      R4    CROSS SECTION DATA
308: C  MCX      I4    LENGTH OF CX ARRAY
309: C
310: C=====
311: C
312: C /#IF ARGSAVE
313: C      return
314: C
315: C
316: C      entry NEUTR
317: C      2 ( LSFFL, NFFL , IZFFL, LSCOL, NCOLS, LSDED, NDEAD, NFISB,
318: C      XXX , YYY , ZZZ , AAA , BBB , CCC , WWW , EEE ,
319: C      4 TTT,ITT, XIM,KLSF,KKP,IZZ,IGG, LEVL, LZZ, LPOS, LCRS,
320: C      5 XXXF , YYYYF , ZZZF , EEEF , INUF , IZZF , LEVLF, LZZF ,
321: C      6 LPOSF, LCRSF, IBREG, IBSPC, IBRGF, IBSPF,
322: C      B DBNK,KDBNK,MBNK, IBNK, KIBNK,MIBNK,

```

```

323: *      B      DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
324: *      &      FLCL , RECL , NCLSN, WCLSN, NABSB, WABSB,
325: *      7      NECUT, WECUT, NKILD, WKILD, NSURV, WSURV, NSPLT, WSPLT,
326: *      8      NCNTR, WCNTR, RMIC , DNFLX,      RSCL,
327: *      *c%&  KYPOS, DNFLXSM, RMICSM,
328: *      &      KYPOS, DNFLXSM, RMICSM, RMIMU, WMIMU,
329: *      T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY, RNU,
330: *      W      IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
331: *      W      IWK7 , IWK8 , IWK9 , IWK10, IWK11, IWK12,
332: *      W      IWK13, IWK14, IWK15, IWK16, IWK17, IWK18, IWK19,
333: C##<2007/03/14:PN3:
334: C##*      W      DWK1 , DWK2 , WK3 , WK4 , WK5 , R, SMCW      )
335: *      W      DWK1 , DWK2 , WK3 , WK4 , WK5 , R, SMCW, DWK3 )
336: C##>
337: C /#ENDIF
338: C
339: C=====
340: C
341: C      IF THERE ARE ANY FISSION-PARTICLES IN NON-EIGENVALUE PROBLEMS,
342: C      THE FOLLOWING DATA FLOW OCCURS BY FISGEN AND GAMGEN
343: C
344: C      DEAD PARTICLE STACK ---->  FREE-FLIGHT STACK
345: C
346: C      === BANK DATA TO BE UPDATED ===
347: C
348: C      (XXX,YYY,ZZZ),(AAA,BBB,CCC)      : POSITION,DIRECTION
349: C      EEE,IGG : ENERGY AND ITS GROUP.      WWW : WEIGHT
350: C
351: C      MMAC(,2) : NO. OF CALCULATED MACRO CROSS SECTIONS ----> 0
352: C      SMIC, KSPI : MICRO CROSS SECTIONS
353: C
354: C      === BANK DATA ADDED NEWLY (SPLITTING, FISSION, (N,MN) REACTION) ===
355: C
356: C      (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #
357: C      EEE, IGG, WWW      : ENERGY GROUP & WEIGHT
358: C      LEVL,LZZ,LPOS,LCRS      : LATTICE-PARAMETER
359: C
360: C      === FISSION BANK (TO BE CREATED IN FISGEN) ===
361: C
362: C      (XXXF,YYYF,ZZZF),IZZF      : POSITION & ZONE #
363: C      EEEF      : ENERGY
364: C      INUF      : COLLIDING NUCLIDE
365: C      LEVLF,LZZF,LPOSF,LCRSF      : LATTICE-PARAMETER
366: C
367: C=====
368: C      if ( JVMNT.ne.0 ) call VMNTRI( 5, NCOLS )
369: C
370: C**** TAKE MONITOR
371: C
372: C      if ( JMNTR.ne.0 ) then
373: C      do 100 I = 1, NCOLS
374: C          IGT = IGG(LSCOL(I))
375: C          if ( JTLLT.eq.0 ) then
376: C              IRT = KZREG(IZZ(LSCOL(I)))
377: C          else
378: C              IRT = IBREG(LSCOL(I))
379: C          end if
380: C          NCLSN(IGT,IRT) = NCLSN(IGT,IRT) + 1
381: C          WCLSN(IGT,IRT) = WCLSN(IGT,IRT) + WWW(LSCOL(I))
382: C      100 continue
383: C      end if
384: C
385: C**** Gather MB-value, IREG, collided nuclide, IGG, weight/SGTOT
386: C      ----> IWK1, IWK2, IWK3, IWK4, DWK1
387: C****VOCL LOOP,NOVREC

```

src/mvp/neutr.f

```

388:      do 110 I = 1, NCOLS
389: C
390: C ... WCNTR(4,KPNEUT) : total collision weight
391:      WCNTR(4,KPNEUT) = WCNTR(4,KPNEUT) + WWW(LSCOL(I))
392:      IWK1(I) = MMAC(LSCOL(I),1)
393:      DWK1(I) = WWW(LSCOL(I)) /SMAC(LSCOL(I),IWK1(I),LMAC(1))
394:      IWK5(I) = IZZ(LSCOL(I))
395:      if ( JTLT.eq.0 ) IWK2(I) = KZREG(IWK5(I))
396:      if ( JTLT.ne.0 ) IWK2(I) = IBREG(LSCOL(I))
397:      IWK3(I) = KMAC(LSCOL(I),IWK1(I),1)
398:      IWK4(I) = IGG(LSCOL(I))
399: 110 continue
400:      NCNTR(4,KPNEUT) = NCNTR(4,KPNEUT) + NCOLS
401: C
402: C
403: C**** Take "special" tally by neutron collision estimator
404: C
405: C (STALN1 is called before TALLY, because TALLY routine may change
406: C IWK4 (energy grou #).)
407: C
408:      if ( NTEVE.gt.0.and.JTEVE(10).gt.0 ) then
409:      do 120 I = 1, NCOLS
410:      IWK6(I) = MMAC(LSCOL(I),1)
411:      if ( JTIME.ne.0 ) then
412:      IWK7(I) = ITT(LSCOL(I))
413:      DWK2(I) = TTT(LSCOL(I))
414:      DWK3(I) = 0 ! photonuc
415:      end if
416: 120 continue
417:      do 150 J = 0, JTEVE(10) - 1
418: C
419: C ... pointer to direct tally (idt) & d-tally # (it) ...
420:      IDT = KTEVE(LTEVE(10)+J) - 1
421:      IT = IDTAL(IDT+9)
422:      if ( IDTAL(IDT+7).eq.1003 .or. IDTAL(IDT+7).eq.1004 )
423:      & go to 150 ! photonuc
424: C
425: C ... skip if current region does not need this tally ...
426: C
427:      do 130 I = 1, NCOLS
428:      if ( JDTRG(IWK2(I),IT).ne.0 ) go to 140
429: 130 continue
430:      go to 150
431: C
432: 140 continue
433: C
434:      JPTIM2 = 0
435:      MZONE0 = 0
436: c##<2007/03/14:PN3:
437:      NUCPNA0 = 0 ! dummy argument
438:      MNUCPN0 = 0 ! dummy argument
439:      LPIDPN0 = 0 ! dummy argument
440:      MNEUTPN0 = 0 ! dummy argument
441:      DNSTPN0 = 0 ! dummy argument
442: c##>
443: C
444: C << comment on Feb 23 2000 >>
445: c##<2007/03/14:PN3:
446: c##C ... TCUT00 is used for TCUT not passed to this routine,
447: c##C and it should not be used in this call.
448: c##C ... TIMEB0 is also a spacer for real TMIBE(*)
449: c##>
450: C ... Currently treatment for JPTIM > 0 in STALN1 may be
451: C incorrect for collision estimator
452: C

```

```

453: c##<2007/03/14:PN3:
454: c## call STALN1( IOW, JTIME, JPTIM, MZONE0, IDTAL(IDT+1), DWK1,
455: c## & NCOLS, IWK5, IWK4, IWK2, LSCOL, IWK6, IWK7, DWK2,
456: c## & TDUMY0, NGROUP, NREG, NRESP, NUC, NBANK, NSMIC,
457: c## & NSMAC, NSTAL, MB, DNZON, NZONE, NTIME, NEMIC, RESP,
458: c## & TIMEB0, NMKREG, MKREG, DTALY, SMIC, LMIC, SMAC,
459: c## & LMAC, SGTAL, TCUT00, DBNK, KDBNK, MDBNK, IBNK,
460: c## & KIBNK, MIBNK, R, IWK8, IWK9, JPTIM2 )
461: call STALN1( IOW, JTIME, JPTIM, MZONE0, IDTAL(IDT+1), DWK1,
462: & NCOLS, IWK5, IWK4, IWK2, LSCOL, IWK6, IWK7, DWK2,
463: & DWK3, NGROUP, NREG, NRESP, NUC, NBANK, NSMIC,
464: & NSMAC, NSTAL, MB, DNZON, NZONE, NTIME, NEMIC, RESP,
465: & TIMEB, NMKREG, MKREG, DTALY, SMIC, LMIC, SMAC,
466: & LMAC, SGTAL, TCUT, DBNK, KDBNK, MDBNK, IBNK,
467: & KIBNK, MIBNK, R, IWK8, IWK9, JPTIM2,
468: & EEE, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, CXPNI,
469: & MCXPN, INCSTPN, MTMPN, NMTPN, KLBPN1, KLBPN2,
470: & XLBPN1, EBOTLPN, NUCPNA0, MNEUTPN0, DNSTPN0,
471: & MNUCPN0, LPIDPN0, SMCW(1,1), SMCW(1,2), SMCW(1,3),
472: & SMCW(1,4))
473: c##>
474: C
475: 150 continue
476: end if
477: C
478: C
479: C**** TAKE TALLY BY COLLISION ESTIMATOR
480: C
481: C
482:      if ( JVMNT.ne.0 ) call VMNT00( 10, 5 )
483:      if ( JVMNT.ne.0 ) call VMNTRI( 10, NCOLS )
484: C
485:      JREGN = -1
486:      JZONE = 1
487: C
488:      call TALLY( IOW, JEIGN, JRTCL, JREGN, JZONE, JRESP, DWK1, NCOLS,
489: & IWK4, IWK2, LSCOL, IWK1, DWK2, NGROUP, NGP(KPNEUT), NREG,
490: & NRESP, NUC, NBANK, NSMIC, NSMAC, NSTAL, MB, IWK5, DNZON,
491: & NZONE, LEMIC, NEMIC, RESP, SMIC, LMIC, SMAC, LMAC, SGTAL,
492: & FLCL, RECL, RMIC, DNFLX, RSCL, WCNTR(14,KPNEUT),
493: & WCNTR(19,KPNEUT),
494: c+beff1
495: & JBEFF, RNU, IBNK, KIBNK, MIBNK, WCBF(2), WCBF(5),
496: & WCBF(8), WCBF(11),
497: c-beff1
498: c##<2007/03/14:PN4:
499: & JPHNU, KPNUFG, WCNTR(31,KPNEUT), NUC_MAX )
500: c##>
501: C
502:      if ( JPRT.ne.0 ) then
503:      call TALKP( IOW, JEIGN, JRTCL, JREGN, JZONE, JRESP,DWK1, NCOLS,
504: & IWK4, IWK2, LSCOL, IWK1, DWK2, NGROUP, NGP(KPNEUT), NREG,
505: & NRESP, NUC, NBANK, NSMIC, NSMAC, NSTAL, MB, IWK5, DNZON,
506: & NZONE, LEMIC, NEMIC, RESP, SMIC, LMIC, SMAC, LMAC, SGTAL,
507: & FLCL, RECL, RMIC, DNFLX, RSCL,
508: & DUMMY1, DUMMY2 ,
509: & NPTDS, WWD, WCTRP, JPTTR, JPTNU, KZREG, WWD2,
510: & NPTCS, WWR, MAXDA, DELA ,
511: & SMACP,
512: & KZMAT, NUCPT, NMAT, MNUC, LPDEN, INUCT, DENST, WKPN,
513: & WWT, WKPT, WKPD, JPRT, JPTMP, JPDEN, NGSP, WSD,
514: & WSC, NOCS, NTPT, NORDD, NOPS,
515: c+beff2
516: & JBEFF, WWD, WSB, RNU, NPTBE,
517: & WWBD, WSD, WWLD, WSLD)

```

src/mvp/neutr.f

```

518: c-beff2
519:   end if
520: C
521:   if ( JVMNT.ne.0 ) call VMNT22( 10, 5 )
522: C
523: C**** Pure analog treatment of fission for noise analysis
524: C
525:   if ( JANLF.eq.1.and.JEIGN.eq.0 ) then
526: C
527: C <<<< CAUTION !!!!! >>>>
528: C
529: C -----
530: C   output from PANLGF routine
531: C   IMTTP : number of neutrons generating photon by absorption
532: C   IWK5  : bank pointer of fission neutrons. DO NOT DESTROY!
533: C   IWK11 : MT-number to be determined (initially= 0)
534: C   IWK12 : Bank pointer
535: C   IWK13 : Nuclide #
536: C   WK4   : Weight of paraent neutron ( absorption )
537: C   WK5   : Weight of paraent neutron ( scattering )
538: C
539: C -----
540: C   IWK2(region no.) is input to PANLGF routine
541: C
542: C -----
543: C   IWK3(colliding nuclide) is input/output of PANLGF routine
544: C   and contents of LSCOL may be changed.
545: C -----
546: C   call PANLGF( IOW, IWK5, IMTTP, IWK2, IWK3, IWK11, IWK12, IWK13,
547: C   &      WK4, WK5, NGROUP, NZONE, NMAT, NREG, NTIME, NBANK,
548: C   &      NFBANK, NFBANK0, NEST, NUC, NMT, MB, NSMIC, MMAC, MXPGEN,
549: C   &      ETOP, EBOT, ETHMAX, CX, MCX, KLIB1, XLIB1, KLIB2, XLIB2,
550: C   &      SMIC, LMIC, KSPI, LSFFL, NFFL, IZFFL, LSCOL, NCOLS,
551: C   &      LSDED, NDEAD, NFISB, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW,
552: C   &      EEE, TTT, ITT, XIM, KKP, IZZ, IGG, LEVL, LZZ, LPOS,
553: C   &      LCRS, KLSF, IBREG, IBSPC, XXXF, YYYF, ZZZF, EEEF, INUF,
554: C   &      IZZF, LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF, DBNK,
555: C   &      KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NECUT, WECUT, NCNTR,
556: C   &      WCNTR, IRAND, JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE,
557: C   &      DTALY, RNU, IWK1, IWK4, IWK6, IWK7, IWK8, IWK9, IWK10,
558: C   &      IWK14, IWK15, DWK1, DWK2, WK3, IWK16, IWK17, R,
559: C   &      NUC_MAX, ! photo-nuc
560: C   &      DWK3, IOTL) ! time-list
561: C
562: C   IMTTP0 = IMTTP
563: C   NCASE2 = NCOLS
564: C
565: C   go to 290
566: C   end if
567: C
568: C
569: C**** FISSION ANALYSIS
570: C
571: C
572: C
573:   if ( JFISS.ne.0 ) then
574: C
575: C <<<< CAUTION !!!!! >>>>
576: C
577: C -----
578: C   IWK5 is output from FISGEN routine (bank pointer of
579: C   fission neutron in fixed source case). DO NOT DESTROY!
580: C -----
581: C   IWK2(region no.) is input to FISGEN routine
582: C -----

```

```

583: C   IWK3(colliding nuclide) is input/output of FISGEN routine
584: C -----
585: C   and contents of LSCOL may be changed.
586: C -----
587: C
588: C   call FISGEN( IOW, IWK2, IWK3, IWK5, NGROUP, NZONE, NMAT, NREG,
589: C   &      NTIME, NBANK, NFBANK, NFBANK0, NEST, NUC, NNK, NMT, MB,
590: C   &      NSMIC, MMAC, WGTF, WGTFI, WTIME, WLLIM, MXPGEN, ETOP,
591: C   &      EBOT, ETHMAX, CX, MCX, KLIB1, XLIB1, KLIB2, XLIB2S,
592: C   &      XLIB2, SMIC, LMIC, LSFFL, NFFL, IZFFL, LSCOL, NCOLS,
593: C   &      LSDED, NDEAD, NFISB, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW,
594: C   &      EEE, TTT, ITT, XIM, KKP, IZZ, IGG, LEVL, LZZ, LPOS,
595: C   &      LCRS, KLSF, IBREG, IBSPC, XXXF, YYYF, ZZZF, EEEF, INUF,
596: C   &      IZZF, LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF, DBNK,
597: C   &      KDBNK, MDBNK, IBNK, KIBNK, MIBNK, DFBK, KDFBK, MDFBK,
598: C   &      IFBK, KIFBK, MIFBK, NECUT, WECUT, NCNTR, WCNTR, IRAND,
599: C   &      IWK1, IWK4, IWK6, IWK7, IWK8, IWK9, IWK10, IWK11,
600: C   &      IWK12, IWK13, DWK1, DWK2, WK3, WK4, WK5, R,
601: C   &      SMAC, LMAC, NSMAC, SMACP, SMICP,
602: C   &      NPTDS, WWD, WD0, WD0A, WD0B, NBATCH, NSKIP,
603: C   &      JPTTR, JPTNU, KZREG, NPTCS, WWR, WR0, WR0A, MAXDA,
604: C   &      DELA,
605: C   &      WWT, WT0, WT0A, WT0B,
606: C   &      NUCPT, WKPT, WKPD, WKPN,
607: C   &      NGSP, WSD, WSC, NORDDS, NOPSDDS,
608: C   &      RNU, IWK14, IWK15, IWK16, IMTF,
609: C   c-beff1
610: C   &      NEBEF, WCBEF, WK7, WK8,
611: C   c-beff1
612: C   c##<2007/03/14:PN3:PN4:
613: C   &      TCUT, TIMEB, KPNPRD, NUC_MAX, KPNFG,
614: C   c##>
615: C   c-beff2
616: C   &      WWB, WB0, WSB, NPTBE,
617: C   &      WWBD, WBD0, WSD, WWLD, WLD0, WSLD)
618: C   c-beff2
619: C
620: C   ( Working arrays
621: C   &      IWK1, IWK2, IWK4, IWK6, IWK7, IWK8,
622: C   &      IWK9, IWK10, IWK11, IWK12,
623: C   &      DWK1, DWK2, WK3, WK4, WK5,
624: C   &      WK6, WK7, WK8,
625: C   &      & R can be used freely hereafter )
626: C
627: C   end if
628: C
629: C
630: C -----
631: C**** Treatment of absorption reaction (neutron only calculation)
632: C -----
633: C
634: C   c##<2007/03/14:PN3:
635: C   &      NCASE2 = 0
636: C   c##>
637: C
638:   if ( JPHOT.eq.0 ) then
639: C
640: C.... Weight reduction only
641: C
642:   if ( EWCUT.le.EBOT ) then
643: C*VOCL LOOP,NOVREC
644: C   do 160 I = 1, NCOLS
645: C   IP = LSCOL(I)
646: C   WW =
647: C   &      (SMIC(IP,IWK3(I),LMIC(3))

```

src/mvp/neutr.f

```

648:      &          +SMIC(IP,IWK3(I),LMIC(5))) /
649:      &          SMIC(IP,IWK3(I),LMIC(1))
650:      WWW(IP) = WWW(IP)*(1.-WW)
651:      160      continue
652: C
653: C.... Analog absorption only
654: C
655:      else if ( EWCUT.gt.ETOP ) then
656:      call RANU2( IRAND, R, NCOLS, ICON )
657:      NN      = 0
658: *VOCL LOOP,NOVREC
659:      do 170 I = 1, NCOLS
660:      IP      = LSCOL(I)
661:      WW      =
662:      &          (SMIC(IP,IWK3(I),LMIC(3))
663:      &          +SMIC(IP,IWK3(I),LMIC(5))) /
664:      &          SMIC(IP,IWK3(I),LMIC(1))
665:      if ( R(I).lt.WW ) then
666:      NCNTR(23,KPNEUT) = NCNTR(23,KPNEUT) + 1
667:      WCNTR(23,KPNEUT) = WCNTR(23,KPNEUT) + WWW(IP)
668:      NDEAD      = NDEAD + 1
669:      LSDDED(NDEAD) = IP
670: c##<2007/03/14:PN3:
671:      if ( KIBNK(17).ne.0 ) IBNK(IP,KIBNK(17)) = 0
672:      if ( KIBNK(18).ne.0 ) IBNK(IP,KIBNK(18)) = 0
673:      if ( KIBNK(19).ne.0 ) IBNK(IP,KIBNK(19)) = 0
674: c##>
675: c##<2007/03/14:PN4:
676:      if ( JPHNU.ne.0 ) KPNFG(IP) = 0
677: c##>
678:      else
679:      NN      = NN + 1
680:      LSCOL(NN) = IP
681:      end if
682:      170      continue
683:      NCOLS   = NN
684: C
685: C... Mixed mode of weight reduction and analog absorption
686: C
687:      else
688:      call RANU2( IRAND, R, NCOLS, ICON )
689:      NN      = 0
690: *VOCL LOOP,NOVREC
691:      do 180 I = 1, NCOLS
692:      IP      = LSCOL(I)
693:      WW      =
694:      &          (SMIC(IP,IWK3(I),LMIC(3))
695:      &          +SMIC(IP,IWK3(I),LMIC(5))) /
696:      &          SMIC(IP,IWK3(I),LMIC(1))
697:      ID      = 0
698:      if ( EEE(IP).ge.EWCUT ) then
699:      WWW(IP) = WWW(IP)*(1.-WW)
700:      else if ( R(I).lt.WW ) then
701:      ID      = 1
702:      NCNTR(23,KPNEUT) = NCNTR(23,KPNEUT) + 1
703:      WCNTR(23,KPNEUT) = WCNTR(23,KPNEUT) + WWW(IP)
704:      NDEAD      = NDEAD + 1
705:      LSDDED(NDEAD) = IP
706:      end if
707: C
708:      if ( ID.eq.0 ) then
709:      NN      = NN + 1
710:      LSCOL(NN) = IP
711:      end if
712:      180      continue

```

```

713:      NCOLS   = NN
714:      end if
715: C
716: C
717: C-----
718: C**** Treatment of absorption reaction (neutron/photon calculation)
719: C-----
720: C
721: C
722:      else if ( JPHOT.ne.0 ) then
723:      call RANU2( IRAND, R, NCOLS, ICON )
724: C
725: C
726: C .....
727: C << STORED DATA FOR PHOTON GENERATION >>
728: C
729: C      Data for assumed-gamma-generation neutrons
730: C      ( effective until calling of 'GAMGEN' )
731: C
732: C      IMTTP      : Number of assumed-gamma-generation-neutrons
733: C      IWK11(I)   : Reaction MT number ( not MTTP !! )
734: C      IWK12(I)   : Bank pointer for assumed-gamma-generation neutrons
735: C                  (negative if parent neutron is not alive.)
736: C      IWK13(I)   : Colliding nuclide ( <- IWK3 )
737: C      WK4(I)     : Weight of parent neutron
738: C .....
739: C
740: C
741: C      IMTTP      = 0
742: C
743: C      .... NAF: Number of assumed-photon-generation from fission
744: C
745: C      NCASE1     = 0
746: C      NCASE2     = 0
747: C      NAF        = 0
748: C      NAF1       = 0
749: C
750: *VOCL LOOP,NOVREC
751:      do 190 I = 1, NCOLS
752:      IP      = LSCOL(I)
753:      ICASE    = 2
754: C
755: C      .... WF : Weight reduction factor ( reset to 0 or 1 for analog )
756: C                  (non absorption probability)
757: C
758: C      1.0 - ( FISSION + CAPTURE )/ TOTAL
759: C
760: C      WS        = 1.0
761: C      &          -
762: C      &          (SMIC(IP,IWK3(I),LMIC(3))+SMIC(IP,IWK3(I),LMIC(5)))
763: C      &          /SMIC(IP,IWK3(I),LMIC(1))
764: C      WF        = WS
765: C
766: C      .... PFIS : NON-ABSORPTION + FISSION PROBABILITY
767: C
768: C      ( to be used to determine photon generation from absorption
769: C      is from fission (MT=18) or capture (MT=102-114) )
770: C
771: C      PFIS      = 1.0 - SMIC(IP,IWK3(I),LMIC(5)) /
772: C      &          SMIC(IP,IWK3(I),LMIC(1))
773: C
774: C -----
775: C      .... Non-analog absorption .....
776: C -----
777:      if ( EEE(IP).ge.EWCUT ) then

```

src/mvp/neutr.f

```

778:         if ( R(I).gt.WS ) ICASE = 1
779: C
780: C -----
781: C .... ANALOG ABSORPTION .....
782: C -----
783:         else
784:         if ( R(I).le.WS ) then
785: C
786: C ..... Neutron is scattered ....
787: C***** ICASE = 2 ***** INITIAL SETTING ****
788:         WF = 1.0
789: C
790: C ..... Neutron is absorbed ....
791: C
792:         else
793:         ICASE = 0
794:         end if
795:         end if
796: C-----
797: C Collect data for gamma generation (from absorption reaction)
798: C-----
799:         if ( ICASE.eq.0 ) then
800:             IMTTP = IMTTP + 1
801:             IWK12(IMTTP) = IP
802: C ..... nuclide #
803:             IWK13(IMTTP) = IWK3(I)
804:             WK4(IMTTP) = WWW(IP)
805: C
806: C ..... Easily determined photon-generation MT (18) ..
807: C
808:             IWK11(IMTTP) = 0
809:             if ( WS.ne.PFIS.and.R(I).le.PFIS ) then
810:                 IWK11(IMTTP) = 18
811:                 NAF = NAF + 1
812:             end if
813:             end if
814: C
815: C-----
816: C Compress collision stack for non-absorbed neutrons
817: C
818: C ICASE = 1 : Photon generates thru. absorption reaction
819: C ICASE = 2 : Photon generates thru. scattering reaction
820: C ( WK5(LSCOL(I)) : Weight before reduction )
821: C
822: C ( ICASE = 0 : Analog absorption )
823: C-----
824: C
825:         if ( ICASE.eq.2 ) then
826:             NCASE2 = NCASE2 + 1
827:             LSCOL(NCASE2) = IP
828:             WK5(IP) = WWW(IP)
829:             WWW(IP) = WWW(IP)*WF
830:         else if ( ICASE.eq.1 ) then
831:             NCASE1 = NCASE1 + 1
832:             IWK6(NCASE1) = IP
833:             IWK1(NCASE1) = IWK3(I)
834:             IWK4(NCASE1) = 0
835:             if ( WS.ne.PFIS.and.R(I).le.PFIS ) then
836:                 IWK4(NCASE1) = 18
837:                 NAF1 = NAF1 + 1
838:             end if
839:             WK3(NCASE1) = WWW(IP)
840:             WWW(IP) = WWW(IP)*WF
841:         end if
842: 190 continue

```

```

843: C
844: C
845: C .... Add scattering neutron of casel to LSCOL ....
846: C .... Add gamma-production neutrons of casel to IWK12 ....
847: C
848: C
849:         NCOLS = NCASE2 + NCASE1
850:         NAF = NAF + NAF1
851:         IMTTP0 = IMTTP
852:         IMTTP = IMTTP0 + NCASE1
853: C
854:         do 200 I = 1, NCASE1
855:             LSCOL(I+NCASE2) = IWK6(I)
856: C
857:             IWK12(I+IMTTP0) = IWK6(I)
858:             IWK11(I+IMTTP0) = IWK4(I)
859:             IWK13(I+IMTTP0) = IWK1(I)
860:             WK4(I+IMTTP0) = WK3(I)
861: 200 continue
862: C
863: C-----
864: C Determine MT-number for assumed photon-generation from capture
865: C if necessary.
866: C IWK11 : MT-number to be determined (initially= 0)
867: C IWK12 : Bank pointer
868: C IWK13 : Nuclide #
869: C-----
870: C
871:         if ( NAF.lt.IMTTP ) then
872:             IDET = NAF
873: C
874: C ..... IWK4 : Cross section pointer ....
875: C ..... R(1:IMTTP) : Cross section summation ....
876: C ..... R(IMTTP+1:2*IMTTP) : Total capture cross section ....
877: C
878:             MXCAPG = 0
879:             do 210 I = 1, IMTTP
880:                 if ( IWK11(I).eq.0 ) then
881:                     IP = IWK12(I)
882: C
883: C ..... KSPI: Energy point #
884: C
885:                     IWK6(I) = KSPI(IP,IWK13(I))
886:                     IWK4(I) = KLIB1(IWK13(I),1) + IWK6(I)
887: C
888: C ..... NCAPG ....
889: C
890:                     IWK7(I) = KLIB1(IWK13(I),27)
891:                     MXCAPG = MAX(IWK7(I),MXCAPG)
892:                     R(I) = 0.0
893:                     R(I+IMTTP) = SMIC(IP,IWK13(I),LMIC(5))
894:                 end if
895: 210 continue
896: C
897:             call RANU2( IRAND, R(2*IMTTP+1), IMTTP, ICON )
898: C
899:             do 230 MMT = 1, MXCAPG
900:                 if ( IDET.eq.IMTTP ) go to 240
901: C
902:             *VOCL LOOP,NOVREC
903:             do 220 I = 1, IMTTP
904:                 if ( MMT.le.IWK7(I).and.IWK11(I).eq.0 ) then
905: C
906: C .... MTCAPG -> MT ....
907: C

```

src/mvp/neutr.f

```

908:          MT      = KLIB2(IWK13(I),MMT,21)
909:          IP      = IWK12(I)
910: C
911: C          .... Cross section for MT ....
912: C
913:          E1      = CX(IWK4(I))
914:          TI      = (EEE(IP)-E1)/(CX(IWK4(I))-E1)
915:          L1      = KLIB2(IWK13(I),MT,11)
916:          &      - KLIB2(IWK13(I),MT,3) + IWK6(I)
917: C
918: C          ..... SC: Capture cross section ...
919: C
920:          if ( IWK6(I).ge.KLIB2(IWK13(I),MT,3)
921:          & .and.IWK6(I).lt.KLIB2(IWK13(I),MT,4) ) then
922:              SC    = CX(L1)*TI + CX(L1+1)*(1.0-TI)
923:          else
924:              SC    = 0.0
925:          end if
926: C
927: C          ... Check ...
928: C
929:          R(I)     = R(I) + SC
930: C
931: C          (Partial sum of cap. xsec)/(Total cap. xsec) > Random.1
932: C          ----> MT determined
933: C
934:          if ( R(IMTTP+1)*R(I+2*IMTTP).le.R(I) ) then
935:              IWK11(I) = MT
936:              IDET     = IDET + 1
937:          end if
938:          end if
939:          220      continue
940:          230      continue
941: C
942:          240      if ( IDET.lt.IMTTP ) then
943:              do 250 I = 1, IMTTP
944: C          .... MTCAPG(NCAPG) -> MT IF MTTOG(MTCAPG)=3 ...
945:              if ( IWK11(I).eq.0.and.IWK7(I).ne.0 ) then
946:                  MTCAPG = KLIB2(IWK13(I),IWK7(I),21)
947:              if ( KLIB2(IWK13(I),MTCAPG,15).eq.3 ) then
948:                  IWK11(I) = MTCAPG
949:              end if
950:          end if
951:          250      continue
952:          end if
953: C
954:          end if
955: CM
956:          if ( NAF.gt.0 ) then
957:              do 260 I = 1, IMTTP
958:                  if ( IWK11(I).eq.18.and.KLIB2(IWK13(I),18,15).eq.0 ) then
959: C          ..... remove particle with mtog=0 by fission
960:                  IWK11(I) = 0
961:              end if
962:          260      continue
963:          end if
964: C
965:          NN      = 0
966:          NN0     = NDEAD
967: C
968:          *VOCL LOOP,NOVREC
969:          do 270 I = 1, IMTTP0
970:              WCNTR(23,KPNEUT) = WCNTR(23,KPNEUT) + WK4(I)
971:              if ( IWK11(I).eq.0 ) then
972:                  NDEAD = NDEAD + 1

```

```

973:          LSDED(NDEAD) = IWK12(I)
974: C##<2007/03/14:PN3:
975:          if ( KIBNK(17).ne.0 ) IBNK(IWK12(I),KIBNK(17)) = 0
976:          if ( KIBNK(18).ne.0 ) IBNK(IWK12(I),KIBNK(18)) = 0
977:          if ( KIBNK(19).ne.0 ) IBNK(IWK12(I),KIBNK(19)) = 0
978: C##>
979: C##<2007/03/14:PN4:
980:          if ( JPHNU.ne.0 ) KPNFG(IWK12(I)) = 0
981: C##>
982:          end if
983:          270      continue
984:          NCNTR(23,KPNEUT) = NCNTR(23,KPNEUT) + IMTTP0
985:          IMTTP0 = IMTTP0 - NDEAD + NN0
986:          *VOCL LOOP,NOVREC
987:          do 280 I = 1, IMTTP
988:              if ( IWK11(I).ne.0 ) then
989:                  NN = NN + 1
990:                  IWK11(NN) = IWK11(I)
991:                  IWK12(NN) = IWK12(I)
992:                  IWK13(NN) = IWK13(I)
993:                  WK4(NN) = WK4(I)
994:              end if
995:          280      continue
996:          IMTTP = NN
997: C
998:          end if
999: C
1000: C          ( IWK7 can be used freely hereafter )
1001: C
1002: C
1003:          290 continue
1004: C
1005: C =====
1006: C          Neutron scattering reaction
1007: C =====
1008: C
1009: C -----
1010: C          C.... Determination of reaction type
1011: C          LSCOL(I),NCOLS ----> LSCOL, NCOLS, IWK1, IWK2 (non-thermal)
1012: C          IWK3 , NCOLF, IWK4 (free-gas)
1013: C          IWK6 , NCOLT, IWK7 (thermal)
1014: C          STACK NO. NUC MT
1015: C -----
1016: C -----
1017: C
1018: C
1019:          NN      = 0
1020:          NCOLF   = 0
1021:          NCOLT   = 0
1022:          NTHR    = 0
1023:          NCS2    = 0
1024: CM2014
1025:          NRS      = 0
1026: C
1027:          *VOCL LOOP,NOVREC
1028:          do 300 I = 1, NCOLS
1029:              IP      = LSCOL(I)
1030:              NBMAT   = MMAT(IP,1)
1031:              KN      = KMAC(IP,NBMAT,1)
1032: C##<2007/03/14:PN3:
1033: C
1034:          if ( KPNPRD.eq.0 ) then
1035:              if ( KIBNK(17).ne.0 ) then
1036:                  if ( JTLTL.eq.0 ) then
1037:                      IBNK(IP,KIBNK(17)) = - KZREG(IZZ(IP))      ! produce-regi

```

src/mvp/neutr.f

```

on
1038:      else
1039:      IBNK(IP,KIBNK(17)) = - IBREG(IP)      ! produce-regi
on
1040:      end if
1041:      end if
1042:      if ( KIBNK(18).ne.0 ) IBNK(IP,KIBNK(18)) = KN      ! produce-nucl
ide
1043: c##<2007/03/14:PN4:
1044:      if ( JPHNU.ne.0 ) KPNFG(IP) = 0
1045: c##>
1046:      end if
1047: c##>
1048:
1049: C -----
1050: C      Non-thermal reaction   ( E > ETHMAX )
1051: C -----
1052:
1053:      if ( EEE(IP).gt.ETHMAX ) then
1054:      NN      = NN + 1
1055:      LSCOL(NN) = IP
1056:      if ( JPHOT.ne.0.and.I.le.NCASE2 ) NCS2 = NCS2 + 1
1057: C
1058: C      .... COLLIDING NUCLIDE # .....
1059: C
1060:      IWK1(NN) = KN
1061:
1062: C -----
1063: C      Free gas      ( E > ESAB and E > ETHERM )
1064: C -----
1065:
1066:      else if ( KLIB1(KN,15).eq.0 .or.
1067:      &      ( EEE(IP).gt.XLIB1(KN,4) .and.
1068:      &      EEE(IP).gt.XLIB1(KN,3) ) ) then
1069:      NCOLF = NCOLF + 1
1070:      IWK3(NCOLF) = IP
1071:      IWK4(NCOLF) = KN
1072:
1073: C -----
1074: C      Thermal reaction ..... S(A,B)   ( E <= ESAB or E<=ETHERM)
1075: C -----
1076:
1077:      else
1078:      NCOLT = NCOLT + 1
1079:      IWK6(NCOLT) = IP
1080:      IWK7(NCOLT) = KN
1081:      end if
1082:
1083:      if ( JPERT.ne.0 ) then
1084:
1085: C -----
1086: c DENSITY & NUMBER DENSITY
1087: C -----
1088:      if( JPDEN.ne.0 ) then
1089:
1090:      if ( JTLT.eq.0 ) then
1091:      IREG = KZREG(IZZ(IP))
1092:      else
1093:      IREG = IBREG(IP)
1094:      end if
1095:
1096:      do IPT = 1, NPTDS
1097: c ... density perturbation ...
1098:      WWD(IP,IPT,1) = WWD(IP,IPT,1)
1099:      &      + dble(JPTTR(IREG,IPT))

```

```

1100:      &      * dble(JPTNU(KN,IPT))
1101:      WWD(IP,IPT,2) = WWD(IP,IPT,2)
1102:      &      - dble(JPTTR(IREG,IPT))
1103:      &      * dble(JPTNU(KN,IPT))
1104:      WWD(IP,IPT,3) = WWD(IP,IPT,3)
1105:      &      + dble(JPTTR(IREG,IPT))
1106:      &      * dble(JPTNU(KN,IPT))*2.d0
1107:      WWD(IP,IPT,4) = WWD(IP,IPT,4)
1108:      &      - dble(JPTTR(IREG,IPT))
1109:      &      * dble(JPTNU(KN,IPT))*6.d0
1110:      WWD(IP,IPT,5) = WWD(IP,IPT,5)
1111:      &      + dble(JPTTR(IREG,IPT))
1112:      &      * dble(JPTNU(KN,IPT))*24.d0
1113:      WWD(IP,IPT,6) = WWD(IP,IPT,6)
1114:      &      - dble(JPTTR(IREG,IPT))
1115:      &      * dble(JPTNU(KN,IPT))*120.d0
1116:      WWD(IP,IPT,7) = WWD(IP,IPT,7)
1117:      &      + dble(JPTTR(IREG,IPT))
1118:      &      * dble(JPTNU(KN,IPT))*720.d0
1119:      WWD(IP,IPT,8) = WWD(IP,IPT,8)
1120:      &      - dble(JPTTR(IREG,IPT))
1121:      &      * dble(JPTNU(KN,IPT))*5040.d0
1122:
1123:      WWD2(IP,IPT) = WWD2(IP,IPT)
1124:      &      - dble(JPTTR(IREG,IPT))
1125:      &      * dble(JPTNU(KN,IPT))
1126:
1127:      end do
1128:
1129:      do 235 IPT = 1, NPTCS
1130:      DLT01 = dble(JPTTR(IREG,NPTDS+IPT))
1131:      DELA1 = dble(DELA(1,NPTDS+IPT))
1132:      WWR(IP,IPT) = WWR(IP,IPT)*(1.0d0+DELA1*DLT01)
1133:      do 236 ISP = 0, NGSP
1134:      WSC(IP,ISP,IPT) =
1135:      &      WSC(IP,ISP,IPT)*(1.0d0+DELA1*DLT01)
1136:      236 continue
1137:      235 continue
1138:
1139:      end if
1140:
1141:      end if
1142:      300 continue
1143:      NCOLS = NN
1144: C
1145:      if ( NCOLS.gt.0 ) then
1146: C
1147: C-----
1148: C.... Determination of threshold reaction (MT --> IWK2)
1149: C-----
1150: C
1151:      call RANU2( IRAND, R, NCOLS, ICON )
1152:      do 310 I = 1, NCOLS
1153: C
1154: C      .... X1 : elastic   X2 : (Total - Absorption)*(Random number)
1155: C
1156:      X1 = SMIC(LSCOL(I),IWK1(I),LMIC(4))
1157:      X2 = ( SMIC(LSCOL(I),IWK1(I),LMIC(1))
1158:      &      -SMIC(LSCOL(I),IWK1(I),LMIC(3))
1159:      &      -SMIC(LSCOL(I),IWK1(I),LMIC(5))) *R(I)
1160: C
1161: C      ... MT = 2 : elastic scattering
1162:      IWK2(I) = 2
1163: C
1164: C -----

```


src/mvp/neutr.f

```

1165: C      X1 < X2 and E > ETH      ==> Threshold reaction
1166: C      -----
1167: C
1168: C      MTTHR = KLIB2(IWK1(I),1,10)
1169: C      if ( MTTHR.gt.0.and.X1.lt.X2
1170: C      & .and.XLIB2(IWK1(I),MTTHR,2).lt.EEE(LSCOL(I)) ) then
1171: C      NTHR = NTHR + 1
1172: C      IWK8(NTHR) = I
1173: C      end if
1174: C 310 continue
1175: C
1176: C      if ( NTHR.gt.0 ) then
1177: C      call RANU2( IRAND, R, NTHR, ICON )
1178: C      MA = 0
1179: C      NN = 0
1180: C      *VOCL LOOP,NOVREC
1181: C      do 320 J = 1, NTHR
1182: C      I = IWK8(J)
1183: C      IP = LSCOL(I)
1184: C      KN = IWK1(I)
1185: C      -----
1186: C      MT # of total threshold reaction
1187: C      If only one threshold reaction is possible,
1188: C      this is the reaction number selected.
1189: C      -----
1190: C      MT = KLIB2(KN,1,10)
1191: C 311 MT = MAX( KLIB2(KN,1,10), 2 )
1192: C      IWK2(I) = MT
1193: C
1194: C      ..... Number of threshold reaction (NLEV) .....
1195: C
1196: C      IWK9(I) = KLIB1(KN,6)
1197: C      if ( IWK9(I).ge.2 ) then
1198: C      NN = NN + 1
1199: C      IWK10(NN) = I
1200: C
1201: C      ..... Maximum number of threshold reaction ....
1202: C
1203: C      MA = MAX(MA,IWK9(I))
1204: C
1205: C 312 Energy mesh interval (pointer)
1206: C
1207: C      L2 = KLIB1(KN,1) + KSPI(IP,KN)
1208: C      DWK1(I) = (EEE(IP)-CX(L2)) / (CX(L2-1)-CX(L2))
1209: C      DWK2(I) = 1. - DWK1(I)
1210: C      L1 = KLIB2(KN,MT,11) - KLIB2(KN,MT,3)
1211: C      & + KSPI(IP,KN)
1212: C
1213: C 313 Threshold total cross section * random number
1214: C
1215: C      WK3(I) = (CX(L1)*DWK1(I)+CX(L1+1)*DWK2(I))*R(J)
1216: C      end if
1217: C 320 continue
1218: C
1219: C      -----
1220: C      Select a threshold reaction if more than two threshold reactions
1221: C      are possible. (MA > 1)
1222: C      -----
1223: C
1224: C      do 340 N = 2, MA
1225: C      NTHR2 = NN
1226: C      NN = 0
1227: C      *VOCL LOOP,NOVREC
1228: C      do 330 J = 1, NTHR2
1229: C      I = IWK10(J)

```

```

1230: C      KN = IWK1(I)
1231: C      MT = KLIB2(KN,N,10)
1232: C      IP = LSCOL(I)
1233: C      -----
1234: C      Pick up cross section if energy point (KSPI) is within energy
1235: C      point-range of MT=KLIB2(KN,N,10), else gives zero x-sec.
1236: C      -----
1237: C      X1 = 0.
1238: C      if ( KLIB2(KN,MT,3).le.KSPI(IP,KN)
1239: C      & .and.KLIB2(KN,MT,4).gt.KSPI(IP,KN) ) then
1240: C      L1 = KLIB2(KN,MT,11) - KLIB2(KN,MT,3)
1241: C      & + KSPI(IP,KN)
1242: C      X1 = CX(L1)*DWK1(I) + CX(L1+1)*DWK2(I)
1243: C      end if
1244: C      -----
1245: C      X1 >= (total threshold cross section)*(random number)
1246: C      ==> This reaction is selected
1247: C      -----
1248: C      if ( X1.ge.WK3(I) ) then
1249: C      IWK2(I) = MT
1250: C      if ( N.lt.IWK9(I) ) then
1251: C      NN = NN + 1
1252: C      IWK10(NN) = I
1253: C      end if
1254: C      end if
1255: C 330 continue
1256: C      if ( NN.le.0 ) go to 350
1257: C 340 continue
1258: C 350 continue
1259: C      end if
1260: C
1261: C      ..... Store data for gamma generation .....
1262: C
1263: C      if ( JPHOT.ne.0 ) then
1264: C      NCASE2 = NCS2
1265: C      do 360 I = 1, NCASE2
1266: C      ... Select particles whose mttog > 0.
1267: C
1268: C      if ( KLIB2(IWK1(I),IWK2(I),15).ne.0 ) then
1269: C      IMTTP = IMTTP + 1
1270: C
1271: C      .... MT # .....
1272: C
1273: C      IWK11(IMTTP) = IWK2(I)
1274: C
1275: C      .... Bank pointer .....
1276: C
1277: C      IWK12(IMTTP) = LSCOL(I)
1278: C
1279: C      .... Colliding nuclide .....
1280: C
1281: C      IWK13(IMTTP) = IWK1(I)
1282: C
1283: C      .... Weight ( before weight reduction ) ....
1284: C
1285: C      WK4(IMTTP) = WK5(LSCOL(I))
1286: C      end if
1287: C
1288: C      continue
1289: C 360
1290: C
1291: C      ( Working area WK5 can be used freely hereafter )
1292: C
1293: C
1294: C

```

src/mvp/neutr.f

```

1295:         end if
1296: C
1297:         end if
1298: C
1299: C
1300: C =====
1301: C   GAMMA GENERATION TREATMENT HERE !!
1302: C =====
1303: C   UNUSABLE WORK ARRAY   IWK1,2,3,4,5,6,7,8,19
1304: C   USABLE   IWK9,10, DWK1, DWK2, WK3, WK5
1305: C
1306: C
1307: C   .... OUTPUT OF PHTGEN ROUTINE ....
1308: C
1309: C   NPGEN : NUMBER OF GENERATED PHOTON
1310: C   IWK19 : BANK POINTERS OF GENERATED PHOTON
1311: C   NKILN : KILLED NEUTRON BECAUSE OF BANK-OVERFLOW !!
1312: C
1313: C   .... INPUT/OUTPUT
1314: C
1315: C   IWK19 : POSITION IN LSCOL ARRAY FOR EACH PARTICLE
1316: C   (IWK19=0 : NOT IN LSCOL )
1317: C
1318: C   if ( NDEAD.lt.IMTTP ) then
1319: C     do 370 I = 1, NCOLS
1320: C       IWK19(LSCOL(I)) = I
1321: C     370 continue
1322: C
1323: C     if ( NCOLF.gt.0 ) then
1324: C       do 380 I = 1, NCOLF
1325: C         IWK19(IWK3(I)) = NCOLS + I
1326: C       380 continue
1327: C     end if
1328: C     if ( NCOLT.gt.0 ) then
1329: C       do 390 I = 1, NCOLT
1330: C         IWK19(IWK6(I)) = NCOLS + NCOLF + I
1331: C       390 continue
1332: C     end if
1333: C
1334: C   end if
1335: C
1336: C   NPGEN = 0
1337: C   NKILN = 0
1338: C
1339: C   if ( JPHOT.ne.0.and.IMTTP.gt.0 ) then
1340: C     call PHTGEN( IMTTP, IWK11, IWK12, IWK13, WK4, IWK18, NPGEN,
1341: C       &         IWK19, NKILN, IOW, IRAND, JTIME, JIMPT, JWTIM, JRWR,
1342: C       &         JMNTR, JDEBG, JLATT, JHLAT, JTLLT, NZONE, NREG, NTIME,
1343: C       &         NBANK, BANKP, NEST, NUC, NMT, ETOPP, EBOTP, ENGYB, CX,
1344: C       &         CX, MCX, KLIB1, KLIB2, XLIB2, WGTP, WTIME, WLLIM,
1345: C       &         MXPGEN, KZREG, LSDED, NDEAD, LSCOL, IMTTP0, XXX, YYY,
1346: C       &         ZZZ, AAA, BBB, CCC, WWW, EEE, TTT, ITT, XIM, KKP, IZZ,
1347: C       &         IGG, LEVL, LZZ, LPOS, LCRS, KLSF, IBREG, IBSPC, DBNK,
1348: C       &         KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NCNTR, WCNTR, NEVENT,
1349: C       &         R, R(1), R(NBANK+1), R(2*NBANK+1), R(3*NBANK+1),
1350: C       &         R(4*NBANK+1), R(5*NBANK+1), R(6*NBANK+1), R(7*NBANK+1),
1351: C       &         IWK14, IWK15, IWK16, IWK17, IWK9, IWK10, DWK1, DWK2,
1352: C     c##<2007/03/14:PN3:PN4:
1353: C     c## &         WK3, WK5 )
1354: C     &         WK3, WK5, KPNPRD, KPNFG, JPHNU )
1355: C   c##>
1356: C
1357: C   < IWK10,DWK1,DWK2,WK3 & WK5
1358: C   ARE FREE TO USE HEREFTER >
1359: C

```

```

1360: C   ...SOME PARENT NEUTRONS ARE KILLED. I HOPE SUCH A CASE BE RARE. ....
1361: C
1362: C
1363: C   if ( NKILN.gt.0 ) then
1364: C     do 400 I = 1, NCOLS
1365: C       IWK14(I) = 0
1366: C     400 continue
1367: C     do 410 J = 1, NTHR
1368: C       IWK14(IWK8(J)) = 1
1369: C     410 continue
1370: C
1371: C     NN = 0
1372: C   *VOCL LOOP,NOVREC
1373: C     do 420 I = 1, NCOLS
1374: C       if ( LSCOL(I).gt.0 ) then
1375: C         NN = NN + 1
1376: C         LSCOL(NN) = LSCOL(I)
1377: C
1378: C       .... IWK1 & IWK2 IS WORKING ARRAY FOR (N,MN) TREATMET ...
1379: C
1380: C         IWK1(NN) = IWK1(I)
1381: C         IWK2(NN) = IWK2(I)
1382: C         IWK14(NN) = IWK14(I)
1383: C       end if
1384: C     420 continue
1385: C
1386: C     NN1 = 0
1387: C   *VOCL LOOP,NOVREC
1388: C     do 430 I = 1, NCOLF
1389: C       if ( LSCOL(I+NCOLS).gt.0 ) then
1390: C         NN1 = NN1 + 1
1391: C         IWK3(NN1) = IWK3(I)
1392: C         IWK4(NN1) = IWK4(I)
1393: C       end if
1394: C     430 continue
1395: C
1396: C     NN2 = 0
1397: C   *VOCL LOOP,NOVREC
1398: C     do 440 I = 1, NCOLT
1399: C       if ( LSCOL(I+NCOLS+NCOLF).gt.0 ) then
1400: C         NN2 = NN2 + 1
1401: C         IWK6(NN2) = IWK6(I)
1402: C         IWK7(NN2) = IWK7(I)
1403: C       end if
1404: C     440 continue
1405: C
1406: C     NCOLF = NN1
1407: C     NCOLT = NN2
1408: C
1409: C     ( IWK19 IS FREE TO USE HEREFTER ) ...
1410: C
1411: C     NCOLS = NN
1412: C     NTHR = 0
1413: C     do 450 I = 1, NCOLS
1414: C       if ( IWK14(I).eq.1 ) then
1415: C         NTHR = NTHR + 1
1416: C         IWK8(NTHR) = I
1417: C       end if
1418: C     450 continue
1419: C   end if
1420: C
1421: C   end if
1422: C
1423: C
1424: C-----

```

```

      IWK9(J) = 0
      IWK2(I) = -IWK2(I)
    end if
    NSPLTT = NSPLTT + IWK9(J)
    continue

    if ( NSPLTT.gt.0 ) then
      if ( NSPLTT.le.NDEAD ) then
        MA = 0
        do 470 J = 1, NTHR
          MA = MAX(MA,IWK9(J))
        continue
        N = 0
        do 490 K = 1, MA
          do 480 J = 1, NTHR
            if ( IWK9(J).ge.K ) then
              N = N + 1
              IWK10(N) = IWK8(J)
            end if
          continue
        continue
      end if
    end if

    .. INSUFFICIENT BANK SIZE FOR (N,MN) REACTIONS

    else
      NSPLTT = 0
      N = 0
      J1 = 0
      do 510 J = 1, NTHR
        K = IWK9(J)
        NSPLTT = NSPLTT + K
        if ( NSPLTT.gt.NDEAD ) go to 520
        J1 = J
        do 500 II = 1, K
          N = N + 1
          IWK10(N) = IWK8(J)
        continue
      continue
      go to 540
    end if
  end do
end sub

```

```

1490: C          WCNTR(6) = WCNTR(6) + WWW( IP )
1491: C
1492:          WCNTR(25,KPNEUT) = WCNTR(25,KPNEUT)
1493:      &          + WWW(IP)*IWK9(J)
1494: C
1495:          WWW(IP) = WWW(IP)*(IWK9(J)+1)
1496: C          if ( IWK2(I).le.9.and.R(J
1497: C      &          -J1).gt.0.5 ) IWK2(I) = IWK2(I) + 40
1498:          if ( IWK2(I).ge.6.and.IWK2(I).le.9
1499:      &          .and.R(J-J1).gt.0.5 ) IWK2(I) = IWK2(I)
1500:      &          + 40
1501:          end if
1502:      530      continue
1503:          end if
1504: C
1505: C
1506:      540      continue
1507:          end if
1508: C
1509:          NDEAD = NDEAD - NSPLTT
1510: *VOCL LOOP,NOVREC
1511:      do 550 I = 1, NSPLTT
1512:          I0 = IWK10(I)
1513:          IWK8(I) = LSCOL(I0)
1514:          IWK9(I) = LSDED(NDEAD+I)
1515:          NN = NCOLS + I
1516:          LSCOL(NN) = IWK9(I)
1517:          IWK1(NN) = IWK1(I0)
1518:          IWK2(NN) = KLIB2(IWK1(I0),IWK2(I0),2)
1519:          AAA(IWK9(I)) = AAA(IWK8(I))
1520:          BBB(IWK9(I)) = BBB(IWK8(I))
1521:          CCC(IWK9(I)) = CCC(IWK8(I))
1522:          XXX(IWK9(I)) = XXX(IWK8(I))
1523:          YYY(IWK9(I)) = YYY(IWK8(I))
1524:          ZZZ(IWK9(I)) = ZZZ(IWK8(I))
1525:          WWW(IWK9(I)) = WWW(IWK8(I))
1526:          EEE(IWK9(I)) = EEE(IWK8(I))
1527:          IGG(IWK9(I)) = IGG(IWK8(I))
1528:          KKP(IWK9(I)) = KKP(IWK8(I))
1529:          IZZ(IWK9(I)) = IZZ(IWK8(I))
1530:          TTT(IWK9(I)) = TTT(IWK8(I))
1531:          KLSF(IWK9(I)) = 0
1532:
1533:          WCNTR(25,KPNEUT) = WCNTR(25,KPNEUT) + WWW(IWK9(I))
1534:
1535:          if ( JLATT.ne.0 ) LEVL(IWK9(I)) = LEVL(IWK8(I))
1536:          if ( JIMPT.ne.0 ) XIM(IWK9(I)) = XIM(IWK8(I))
1537: CCCCCC          if ( JTLT.ne.0 ) IBREG(IWK9(I)) = IBREG(IWK8(I))
1538:          IBREG(IWK9(I)) = IBREG(IWK8(I))
1539:          if ( JTIME.ne.0 ) ITT(IWK9(I)) = ITT(IWK8(I))
1540:
1541:          if ( JPERT.ne.0 ) then
1542:
1543:              do IPT = 1, NPSTDs
1544:                  do IOD = 1, NORDDs
1545:                      WWD(IWK9(I),IPT,IOD) = WWD(IWK8(I),IPT,IOD)
1546:                  end do
1547:                  WWD2(IWK9(I),IPT) = WWD2(IWK8(I),IPT)
1548:                  do ISP = 0, NGSP
1549:                      do IOP = 1, NOPSDs
1550:                          WSD(IWK9(I),ISP,IPT,IOP) =
1551:      &                          WSD(IWK8(I),ISP,IPT,IOP)
1552:                      end do
1553:                  end do
1554:              end do

```

```

1490: C          WCNTR(6) = WCNTR(6) + WWW( IP )
1491: C
1492:          WCNTR(25,KPNEUT) = WCNTR(25,KPNEUT)
1493:      &          + WWW(IP)*IWK9(J)
1494: C
1495:          WWW(IP) = WWW(IP)*(IWK9(J)+1)
1496: C          if ( IWK2(I).le.9.and.R(J
1497: C      &          -J1).gt.0.5 ) IWK2(I) = IWK2(I) + 40
1498:          if ( IWK2(I).ge.6.and.IWK2(I).le.9
1499:      &          .and.R(J-J1).gt.0.5 ) IWK2(I) = IWK2(I)
1500:      &          + 40
1501:          end if
1502:      530      continue
1503:          end if
1504: C
1505: C
1506:      540      continue
1507:          end if
1508: C
1509:          NDEAD = NDEAD - NSPLTT
1510: *VOCL LOOP,NOVREC
1511:      do 550 I = 1, NSPLTT
1512:          I0 = IWK10(I)
1513:          IWK8(I) = LSCOL(I0)
1514:          IWK9(I) = LSDED(NDEAD+I)
1515:          NN = NCOLS + I
1516:          LSCOL(NN) = IWK9(I)
1517:          IWK1(NN) = IWK1(I0)
1518:          IWK2(NN) = KLIB2(IWK1(I0),IWK2(I0),2)
1519:          AAA(IWK9(I)) = AAA(IWK8(I))
1520:          BBB(IWK9(I)) = BBB(IWK8(I))
1521:          CCC(IWK9(I)) = CCC(IWK8(I))
1522:          XXX(IWK9(I)) = XXX(IWK8(I))
1523:          YYY(IWK9(I)) = YYY(IWK8(I))
1524:          ZZZ(IWK9(I)) = ZZZ(IWK8(I))
1525:          WWW(IWK9(I)) = WWW(IWK8(I))
1526:          EEE(IWK9(I)) = EEE(IWK8(I))
1527:          IGG(IWK9(I)) = IGG(IWK8(I))
1528:          KKP(IWK9(I)) = KKP(IWK8(I))
1529:          IZZ(IWK9(I)) = IZZ(IWK8(I))
1530:          TTT(IWK9(I)) = TTT(IWK8(I))
1531:          KLSF(IWK9(I)) = 0
1532:
1533:          WCNTR(25,KPNEUT) = WCNTR(25,KPNEUT) + WWW(IWK9(I))
1534:
1535:          if ( JLATT.ne.0 ) LEVL(IWK9(I)) = LEVL(IWK8(I))
1536:          if ( JIMPT.ne.0 ) XIM(IWK9(I)) = XIM(IWK8(I))
1537: CCCCCC          if ( JTLT.ne.0 ) IBREG(IWK9(I)) = IBREG(IWK8(I))
1538:          IBREG(IWK9(I)) = IBREG(IWK8(I))
1539:          if ( JTIME.ne.0 ) ITT(IWK9(I)) = ITT(IWK8(I))
1540:
1541:          if ( JPERT.ne.0 ) then
1542:
1543:              do IPT = 1, NPSTDs
1544:                  do IOD = 1, NORDDs
1545:                      WWD(IWK9(I),IPT,IOD) = WWD(IWK8(I),IPT,IOD)
1546:                  end do
1547:                  WWD2(IWK9(I),IPT) = WWD2(IWK8(I),IPT)
1548:                  do ISP = 0, NGSP
1549:                      do IOP = 1, NOPSDs
1550:                          WSD(IWK9(I),ISP,IPT,IOP) =
1551:      &                          WSD(IWK8(I),ISP,IPT,IOP)
1552:                      end do
1553:                  end do
1554:              end do

```

src/mvp/neutr.f

```

1555:
1556:      do IPT = 1, NPTCS
1557:        WWR(IWK9(I),IPT) = WWR(IWK8(I),IPT)
1558:      do ISP = 0, NGSP
1559:        WSC(IWK9(I),ISP,IPT) = WSC(IWK8(I),ISP,IPT)
1560:      end do
1561:    end do
1562:  c+beff2
1563:
1564:    if (NPTBE.gt.0) then
1565:      WWB(IWK9(I)) = WWB(IWK8(I))
1566:      do ISP = 0, NGSP
1567:        WSB(IWK9(I),ISP) = WSB(IWK8(I),ISP)
1568:      end do
1569:      WWBD(IWK9(I)) = WWBD(IWK8(I))
1570:      WWLD(IWK9(I)) = WWLD(IWK8(I))
1571:      do ISP = 1, NGSP
1572:        WSBD(IWK9(I),ISP) = WSBD(IWK8(I),ISP)
1573:        WSLD(IWK9(I),ISP) = WSLD(IWK8(I),ISP)
1574:      end do
1575:    end if
1576:  c-beff2
1577:    550    continue
1578:
1579:    NCNTR(25,KPNEUT) = NCNTR(25,KPNEUT) + NSPLTT
1580:  C
1581:    if ( JLATT.ne.0.and.NSPLTT.gt.0 ) then
1582:      do 570 K = 1, NEST
1583:        *VOCL LOOP,NOVREC
1584:
1585:        do 560 I = 1, NSPLTT
1586:          LZZ(IWK9(I),K) = LZZ(IWK8(I),K)
1587:          LPOS(IWK9(I),K) = LPOS(IWK8(I),K)
1588:          if ( JHLAT.ne.0 ) LCRS(IWK9(I),K) =
1589:            &          LCRS(IWK8(I),K)
1590:        continue
1591:      end if
1592:    if ( JTLTT.ne.0.and.NSPLTT.gt.0 ) then
1593:      do 590 K = 0, NEST
1594:        *VOCL LOOP,NOVREC
1595:
1596:        do 580 I = 1, NSPLTT
1597:          IBSPC(IWK9(I),K) = IBSPC(IWK8(I),K)
1598:        continue
1599:      end if
1600:  C
1601:  C .... optional bank parameters
1602:  C
1603:      do 610 K = 1, KDBNK(0)
1604:        *VOCL LOOP,NOVREC
1605:
1606:        do 600 I = 1, NSPLTT
1607:          DBNK(IWK9(I),K) = DBNK(IWK8(I),K)
1608:        continue
1609:      do 630 K = 1, KIBNK(0)
1610:        *VOCL LOOP,NOVREC
1611:
1612:        do 620 I = 1, NSPLTT
1613:          IBNK(IWK9(I),K) = IBNK(IWK8(I),K)
1614:        continue
1615:      do 630
1616:        NCOLS = NCOLS + NSPLTT
1617:      end if
1618:    end if
1619:  c##<2007/03/14:PN3:

```

```

1620:      if ( KPNPRD.eq.0 ) then
1621:        if ( KIBNK(19).ne.0 ) then
1622:          do I = 1, NCOLS
1623:            IP = LSCOL(I)
1624:            IBNK(IP,KIBNK(19)) = IWK2(I)      ! produce-reaction
1625:          end do
1626:        end if
1627:      end if
1628:  c##>
1629:  c-----
1630:  c TEMPERATURE
1631:  c-----
1632:    if( JPERT.ne.0 .and. JPTMP.ne.0 ) then
1633:      do 249 I = 1, NCOLS
1634:        if( IWK2(I).eq.2 .and.
1635:          &          SMIC(LSCOL(I),IWK1(I),LMIC(4)).ne.0.0) then
1636:
1637:          DSMIC = SMICP(LSCOL(I),IWK1(I),LMIC(4)) /
1638:            &          SMIC(LSCOL(I),IWK1(I),LMIC(4))
1639:          WWT(LSCOL(I),1) = WWT(LSCOL(I),1) + DSMIC
1640:          WWT(LSCOL(I),3) = WWT(LSCOL(I),3) + DSMIC
1641:          WWT(LSCOL(I),5) = WWT(LSCOL(I),5) + DSMIC
1642:          WWT(LSCOL(I),7) = WWT(LSCOL(I),7) + DSMIC
1643:        end if
1644:      249    continue
1645:    end if
1646:  C
1647:  C
1648:  C**** ANALYSIS OF NON-THERMAL SCATTERING REACTION
1649:  C... LSCOL(I) : STACK, IWK1(I) : COLLIDING NUCLEIDE,
1650:  C      IWK2(I) : REACTION TYPE (MT), NCOLS : NUMBER OF PARTICLES
1651:  C      -----> WK3(I) : SCATTERING ANGLE COSINE IN LAB SYSTEM,
1652:  C      IWK14(J): Stack pointer for doppler resonance scatt.
1653:  C      IWK15(IP): Bank for Weight before correction for DPRS
1654:  C      (WK6 in NTHSCT )
1655:  C      NCOLS IS MODIFIED BY ENERGY CUTOFF
1656:  C
1657:  C
1658:  C      call NTHSCT( IOW, IWK1, IWK2, WK3, JMNTR, JDEBG, JVMNT, JTLTT,
1659:  C      &          NGROUP, NZONE, NREG, NBANK, NUC, NNK, NMT, KZREG, NGP1,
1660:  C      &          ETOP, EBOT, ETHMAX, CX, CX, MCX, KLIB1, XLIB1, KLIB2,
1661:  C      &          KLB2S, XLIB2, LSCOL, NCOLS, LSDED, NDEAD, WWW, EEE,
1662:  C      &          IZZ, AAA, BBB, CCC, IBREG, NECUT, WECUT, NCNTR, WCNTR,
1663:  C      &          IRAND, IWK8, IWK9, IWK10, IWK11, IWK12, IWK13, IWK14,
1664:  C      &          IWK15, IWK16, IWK17, IWK19, R, DWK1, DWK2, WK4, WK5,
1665:  C      &          R(NBANK+1) )
1666:  C      call NTHSCT( IOW, IWK1, IWK2, WK3, JMNTR, JDEBG, JVMNT, JTLTT,
1667:  C      &          NGROUP, NZONE, NREG, NBANK, NUC, NNK, NMT, KZREG,
1668:  C      &          NEVENT, ETOP, EBOT, ETHMAX, EWCUT, CX, CX, MCX, KLIB1,
1669:  C      &          XLIB1, KLIB2, XLIB2, LSCOL, NCOLS, LSDED, NDEAD, WWW,
1670:  C      &          EEE, KKP, IZZ, AAA, BBB, CCC, IBREG, NECUT, WECUT,
1671:  C      &          NCNTR, WCNTR, JLATT, JHLAT, JIMPT, JTIME, NEST, XXX,
1672:  C      &          YYY, ZZZ, TTT, ITT, XIM, KLSF, LEVL, LZZ, LPOS, LCRS,
1673:  C      &          IBSPC, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, IRAND,
1674:  C      &          JSCTM, JRTCL, DNZON, IGG, ENGYB(KENGP(KPNEUT))),
1675:  C      &          NGP(KPNEUT), KSPI, SMAC, MB, NSMAC, LMAC, MMAC, SMIC,
1676:  C      &          NSMIC, LMIC, KYPOS, DNFLXSM, RMICSM, JSCMU, RMIMU,
1677:  C      &          WMIMU,
1678:  C      &          IWK8, IWK9, IWK10, IWK11, IWK12, IWK13, IWK14, IWK15,
1679:  C      &          IWK16, IWK17, IWK19, R, DWK1, DWK2, WK4, WK5,
1680:  C      &          R(NBANK+1),
1681:  C      &          JPERT, NPTDS, WWD, WWD2, NPTCS, WWR,
1682:  C      &          NGSP, WSD, WSC, NORDDS, NOPSDS,
1683:  C      c##<2007/03/14:PN4:
1684:  C      &          KPNFG, JPHNU,

```

src/mvp/neutr.f

```

1685: c##>
1686: c+beff2
1687:      &          JBEFF, WWB, WSB, NPTBE,
1688:      &          WWBD, WSD, WWLD, WSLD,
1689: c-beff2
1690:      &          JEXDP, NRS)
1691: C
1692:      end if
1693: CM2014
1694: C      Hereinafter, IWK14-->IWK16, IWK15-->IWK17 to keep IWK14&IWK15
1695: C
1696: C**** ANALYSIS OF THERMAL SCATTERING REACTION (SAB AND THERMAL ELASTIC)
1697: C.... IWK6 : STACK, IWK7 : COLLIDING NUCLIDE,
1698: C      NCOLT : NUMBER OF PARTICLES
1699: C      -----> WK3(NCOLS+1) : SCATTERING ANGLE COSINE IN LAB SYSTEM,
1700: C
1701: C      if ( NCOLT.gt.0 ) then
1702: c##<2007/03/14:PN3:
1703: CM2014 IWK14(1) = KIBNK(19)
1704: IWK16(1) = KIBNK(19)
1705: c##>
1706:      N1      = NCOLS + 1
1707:      call THSCT( IOW, IWK6, IWK7, WK3(N1), JMNTR, JDEBG, JVMNT,
1708:      &          NBANK, NUC, NNK, NMT, NSMIC, NCOLT, ETOP, EBOT, ETHMAX,
1709:      &          CX, CX, MCX, KL1B1, XL1B1, KL1B2, KLB2S, XL1B2, SMIC,
1710:      &          LMIC, KSPI, LSDED, NDEAD, WWW, EEE, NCNTR, WCNTR,
1711:      &          IRAND,
1712:      &          JSCTM, JRTCL, JTLT, JMICE, JMACE, ENGYB(KENGP(KPNEUT)),
1713:      &          NGP(KPNEUT), NGROUP, NZONE, NREG, IGG, IZZ, DNZON,
1714:      &          KZREG, IBREG, SMAC, MB, NSMAC, LMIC, MMAC, KYPOS,
1715:      &          DNFLXSM, RMICSM, JSCMU, RMIMU, WMIMU,
1716: CM      &          IWK14, IWK2, IWK8, IWK9, IWK10, IWK11, IWK12,
1717:      &          IWK16, IWK2, IWK8, IWK9, IWK10, IWK11, IWK12,
1718:      &          DWK1, DWK2, WK4, WK5, R )
1719: C
1720: c##<2007/03/14:PN3:
1721:      if ( KPNPRD.eq.0 ) then
1722:      if ( KIBNK(19).ne.0 ) then
1723: CM2014      if ( IWK14(1).gt.0 ) then
1724:      if ( IWK16(1).gt.0 ) then
1725: *VOCL LOOP,NOVREC
1726:      do I = 1, IWK16(1)
1727:      IBNK(IWK16(I+2),KIBNK(19)) = 93      ! thermal elastic reac
tion
1728:      end do
1729:      J      = IWK16(1)
1730:      else
1731:      J      = 0
1732:      end if
1733:      if ( IWK16(2).gt.0 ) then
1734: *VOCL LOOP,NOVREC
1735:      do I = 1, IWK16(2)
1736:      IBNK(IWK16(I+2+J),KIBNK(19)) = 92      ! thermal inelastic re
action
1737:      end do
1738:      end if
1739:      end if
1740:      end if
1741: c##>
1742:      end if
1743: C
1744: C
1745: C
1746: C**** ANALYSIS OF SCATTERING BY FREE-GAS
1747: C.... IWK3 : STACK, IWK4 : COLLIDING NUCL,
1748: C      NCOLF : NUMBER OF PARTICLES
1749: C      ----> IWK11(1) : ISOTROPIC SCATTERING IN CM SYTEM(ATW>AMLIM)
1750: C      NWOEC = NUMBER OF THESE NEUTRONS
1751: C      WK3(N1) : SCATTERING ANGLE COSINE
1752: C      DIRECTION COSINES ARE NOT CHANGED FOR THESE NEUTRONS
1753: C      ----> IWK12(1) : NEUTRON ANALYZED BY FREE GAS MODEL
1754: C      NCOLF = NUMBER OF THESE NEUTRONS
1755: C      (CHANGED IN FREGAS)
1756: C      DIRECTION COSINES ARE CALCULATED BY FREGAS
1757: C
1758: C
1759: C      NWOEC = 0
1760: C
1761: C      if ( NCOLF.gt.0 ) then
1762: C
1763: C      N1      = NCOLS + NCOLT + 1
1764: C
1765: C      call FREGAS( IOW, IWK3, IWK4, WK3(N1), IWK11, IWK12, JMNTR,
1766: C      &          JDEBG, JVMNT, NGROUP, NZONE, NREG, NBANK, NUC, NNK,
1767: C      &          NMT, KZREG, ETOP, EBOT, ETHMAX, AMLIM, XL1B1, NCOLF,
1768: C      &          NWOEC, LSDED, NDEAD, AAA, BBB, CCC, WWW, EEE, IZZ,
1769: C      &          IRAND,
1770: C      &          JSCTM, JRTCL, JTLT, JMICE, JMACE, ENGYB(KENGP(KPNEUT)),
1771: C      &          NGP(KPNEUT), IGG, DNZON, IBREG, SMAC, MB, NSMAC, LMIC,
1772: C      &          MMAC, SMIC, NSMIC, LMIC, KYPOS, DNFLXSM, RMICSM,
1773: C      &          JSCMU, RMIMU, WMIMU,
1774: CM      &          IWK15, IWK2, IWK7, IWK8, IWK9, IWK10, IWK13, IWK14,
1775: C      &          IWK17, IWK2, IWK7, IWK8, IWK9, IWK10, IWK13, IWK16,
1776: C      &          DWK1, DWK2, WK4, WK5, R )
1777: C
1778: c##<2007/03/14:PN3:
1779:      if ( KPNPRD.eq.0 ) then
1780:      if ( KIBNK(19).ne.0 ) then
1781: *VOCL LOOP,NOVREC
1782:      do I = 1, NCOLF
1783:      IBNK(IWK3(I),KIBNK(19)) = 0      ! free-gas isn't estimated.
1784:      end do
1785:      end if
1786:      end if
1787: c##>
1788:      end if
1789: C
1790: C-----
1791: C      CALCULATE OUTGOING DIRECTION FOR SCATTERED NEUTRONS
1792: C-----
1793: C
1794: C      WK3 : COSINES OF SCATTERING ANGLES ARE STORED
1795: C
1796: C.... ADD THERMAL SCATTERING (BY THSCT) TO COLLISION STACK
1797: *VOCL LOOP,NOVREC
1798:      do 640 I = 1, NCOLT
1799:      NCOLS      = NCOLS + 1
1800:      LSCOL(NCOLS)      = IWK6(I)
1801:      640 continue
1802: C
1803: C.... ADD ISOTROPIC SCATTERING (BY FREGAS) TO COLLISION STACK
1804: C
1805: *VOCL LOOP,NOVREC
1806:      do 650 I = 1, NWOEC
1807:      NCOLS      = NCOLS + 1
1808:      LSCOL(NCOLS)      = IWK11(I)
1809:      650 continue
1810: C
1811:      call RANU2( IRAND, R, NCOLS, ICON )
1812: C

```

src/mvp/neutr.f

```

1813: *VOCL LOOP,NOVREC
1814:   do 660 I = 1, NCOLS
1815:     IP = LSCOL(I)
1816:     AAAAA0 = AAA(IP)
1817:     BBBB0 = BBB(IP)
1818:     CCCCC0 = CCC(IP)
1819:     SINPSI = SQRT(1.-WK3(I)*WK3(I))
1820:     ETA = PI2*R(I)
1821:     SINETA = SIN(ETA)
1822:     COSETA = COS(ETA)
1823:     STHETA = 1. - AAAAA0*AAAAA0
1824:   *VOCL STMT,IF(100)
1825:     if ( STHETA.gt.0.0 ) then
1826:       STHETA = SQRT(STHETA)
1827:       COSPHI = BBBB0/STHETA
1828:       SINPHI = CCCCC0/STHETA
1829:     else
1830:       STHETA = 0.0
1831:       COSPHI = 1.0
1832:       SINPHI = 0.0
1833:     end if
1834:     BBBB0 = BBBB0*WK3(I) + AAAAA0*COSPHI*COSETA*SINPSI
1835:     & - SINPHI*SINPSI*SINETA
1836:     CCCCC0 = CCCCC0*WK3(I) + AAAAA0*SINPHI*COSETA*SINPSI
1837:     & + COSPHI*SINPSI*SINETA
1838:     AAAAA0 = AAAAA0*WK3(I) - COSETA*SINPSI*STHETA
1839:     S = 1./SQRT(AAAAA0*AAAAA0+BBBBB0*BBBBB0+CCCCC0*CCCCC0)
1840:     AAA(IP) = AAAAA0*S
1841:     BBB(IP) = BBBB0*S
1842:     CCC(IP) = CCCCC0*S
1843:   660 continue
1844: C
1845: C
1846: C
1847: C-----
1848: C**** Weight Window for Doppler Resonance Elastic Scattering
1849: C   IWK14(J): collision stack pointer (ICSP in WWDPRS)
1850: C   IP=LSCOL(IWK14(J))
1851: C   IWK14(J) < IWK14(J+1)
1852: C   IWK15(IP): weight before correction for DPRS (WBG in WWDPRS)
1853: C   NRS : number of neutrons processed
1854: C   ----> LSCOL & NCOLS will be changed in WWDPRS
1855: C-----
1856: C
1857:   if ( NRS.gt.0 ) then
1858:     call WWDPRS( IWK14, NRS, IWK15, IRAND,
1859:   S     LSCOL, NCOLS, LSDED, NDEAD,
1860:   &     NBANK, NZONE, NGROUP, NREG, NTIME, NEST,
1861:   V     WLLIM, KZREG, NCNTR, WCNTR, NEVENT,
1862:   C     NKILD, WKILD, NSURV, WSURV, NSPLT, WSPLT,
1863:   B     XXX, YYY, ZZZ, AAA, BBB, CCC,
1864:   B     TTT, WWW, EEE, XIM,
1865:   B     KKP, IZZ, IGG, ITT, LEVL, LZZ,
1866:   B     LPOS, LCRS, KLSF, IBREG, IBSPC,
1867:   c     B     NSMAC, NSMIC, MB, NUC, NSTAL,
1868:   c     B     SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL,
1869:   B     DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
1870:   W     R, WK3, WK4, WK5, IWK8, IWK9,
1871:   c     &     SMICP, SMACP,
1872:   &     NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD, WSC,
1873:   &     NORDDS, NOPSDS,
1874:   c##<2007/03/14:PN3:
1875:   c   &     NUCPN, NSMICPN, SMICPN,
1876:   c##>
1877:   c+beff2

```

```

1878:   &     WWB, WSB, NPTBE,
1879:   &     WWBD, WSBD, WWLD, WSLD
1880:   c-beff2
1881:   &     )
1882:   end if
1883: C
1884: C
1885: C-----
1886: C**** DETERMINATION OF NEUTRON ENERGY GROUP
1887: C-----
1888: C
1889: C
1890: C.... ADD PARTICLES SCATTERED BY FREE-GAS ATOM
1891: C
1892:   do 670 I = 1, NCOLF
1893:     NCOLS = NCOLS + 1
1894:     LSCOL(NCOLS) = IWK12(I)
1895:   670 continue
1896: CM
1897: C.... NCOLS0 : NUMBER OF PARTICLES FOR SPLITB
1898:   NCOLS0 = NCOLS
1899: C
1900: C.... ADD FISSION NEUTRONS to LSCOL (JEIGN = 0)
1901: C   (no more collsion and sent to LSFFL)
1902: C
1903:   if ( JEIGN.eq.0.and.NFISB.gt.0 ) then
1904:     do 680 I = 1, NFISB
1905:       LSCOL(NCOLS+I) = IWK5(I)
1906:   680 continue
1907:     NCOLS = NCOLS + NFISB
1908:   end if
1909: C
1910: C
1911: C.... DETERMINE ENERGY GROUP (NEUTRON) ....
1912: C
1913: C
1914:   do 690 I = 1, NCOLS
1915:     WK3(I) = EEE(LSCOL(I))
1916:   690 continue
1917: C
1918:   call BSVDEC( ENGYB(KENGP(KPNEUT)), NGP(KPNEUT)+1, WK3, IWK1, NCOLS
1919:   &     )
1920: C
1921: C
1922: C.... MODIFY IGG ARRAY
1923: C
1924: C
1925:   NGG = KNGP(KPNEUT) - 1
1926:   do 700 I = 1, NCOLS
1927:     IGG(LSCOL(I)) = NGG + IWK1(I)
1928:   700 continue
1929: C
1930: C
1931: C-----
1932: C   CALCULATION OF MICROSCOPIC CROSS SECTION (SMIC ARRAY)
1933: C-----
1934: C
1935: C
1936: C.... add photons to LSCOL (JEIGN = 0) (sent to LSFFL immediatly)
1937: C
1938:   if ( JPHOT.ne.0.and.NPGEN.ne.0 ) then
1939:     do 710 I = 1, NPGEN
1940:       LSCOL(NCOLS+I) = IWK18(I)
1941:   710 continue
1942:   end if

```

src/mvp/neutr.f

```

1943: C
1944:   if ( JAMXC.eq.0 ) then
1945:     if ( JNEUT.ne.0.and.NCOLS.ne.0 ) then
1946:       if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
1947:       if ( JVMNT.ne.0 ) call VMNTR1( 8, NCOLS )
1948:
1949:     if ( JPRT.ne.0 .and. JPTMP.ne.0 ) then
1950:       call GMICPT( NCOLS, LSCOL, IRAND, JEIGN, NUC, NMT, NBANK,
1951:         & NSMIC, EEE, SMIC, LMIC, KSPI, CX, CX, MCX, KL1B1,
1952:         & KL1B2, XL1B1, SGTAL, CRES, KCRES, NSTAL, MCRES, WK1,
1953:         & WK2, IWK1, IWK2, IWK3, IWK4, IWK5, R, SMCW,
1954:         & SMICP, CX1P, SMC1P, E2P, DEP, RP )
1955:     else
1956:       call GETMIC( NCOLS, LSCOL, IRAND, NUC, NMT, NBANK,
1957:         & NSMIC, EEE, SMIC, LMIC, KSPI, CX, CX, MCX, KL1B1,
1958:         & KL1B2, XL1B1, DWK1, DWK2, IWK1, IWK2, IWK3, IWK4,
1959: c##<2007/03/14:PN3:
1960: c## & IWK5, R, SMCW, RNU )
1961: c## & IWK5, R, SMCW, RNU, NUC_MAX )
1962: c##>
1963:   end if
1964:   if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
1965:   end if
1966: C
1967:   if ( JPHOT.ne.0.and.NPGEN.ne.0 ) then
1968: C
1969:     if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
1970:     if ( JVMNT.ne.0 ) call VMNTR1( 8, NPGEN )
1971: C
1972:     call GTMICP( JGAMM, NPGEN, LSCOL(NCOLS+1), NUC, NMTP,
1973:       & NPATOM, NBANK, NSMIC, EEE, SMIC, KSPI, CXP, CXP,
1974:       & MCXP, KLB1P, KLB2P, XLBP2, DWK1, IWK1, DWK2, IWK2 )
1975:
1976: c   call GTMICP( JGAMM, NPGEN, IWK18, NUC, NMTP, NPATOM, NBANK,
1977: c   & NSMIC, EEE, SMIC, KSPI, CXP, CXP, MCXP, KLB1P, KLB2P,
1978: c   & XLBP2,
1979: c   & DWK1, IWK1,
1980: c   & DWK2, IWK2 )
1981:
1982: C   & SGTAL, CRES, KCRES, NSTAL, MCRES,
1983: C
1984: c##<2007/03/14:PN3:
1985:   if ( JPHNU.ne.0 ) then
1986:     call GTMICPN( NPGEN, LSCOL(NCOLS+1), NUC, NUCPN, NUCPN1,
1987:       & NMTPN, NBANK, NSMICPN, NUC_MAX, EEE, SMICPN,
1988:       & CXPN, CXPN, MCXPN, KLBPN1, KLBPN2, XLBPN1,
1989:       & EBOTLPN, DWK1, DWK2, IWK1, IWK2, WK3, SMCW, RNU )
1990:   end if
1991: c##>
1992:   if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
1993:   end if
1994: C
1995: C   ... ADAPTIVE-MICRO-CALCULATION mode
1996: C   calculate micro for micro reaction rate tally here
1997: C
1998:   else
1999: C
2000: C   ... KSPI is used to judge that micro for the particle has
2001: C   been calculated or not
2002: C
2003:   do 730 N = 1, NUC
2004:     do 720 I = 1, NCOLS + NPGEN
2005:       KSPI(LSCOL(I),N) = 0
2006:       720 continue
2007:     730 continue

```

```

2008: C
2009:   if ( NNCST.gt.0 ) then
2010: C
2011:     if ( JNEUT.ne.0.and.NCOLS.gt.0 ) then
2012:       if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
2013:       if ( JVMNT.ne.0 ) call VMNTR1( 8, NCOLS )
2014:       do 740 I = 1, NCOLS
2015:         WK3(I) = EEE(LSCOL(I))
2016:       740 continue
2017:       do 750 NC = 1, NNCST
2018:         IN = INCST(NC,1)
2019:         if ( IN.ne.0 ) then
2020:           call GMICN1( IN, NCOLS, LSCOL, WK3, IRAND,
2021:             & NUC, NMT, NBANK, NSMIC, SMIC, LMIC, KSPI,
2022:             & CX, CX, MCX, KL1B1, KL1B2, XL1B1, DWK2,
2023:             & IWK1, IWK2, IWK3, IWK4, R, SMCW,
2024:             & RNU, NUC_MAX)
2025:         end if
2026:       750 continue
2027:       if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
2028:     end if
2029: C
2030:     if ( JPHOT.ne.0.and.NPGEN.gt.0 ) then
2031:       if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
2032:       if ( JVMNT.ne.0 ) call VMNTR1( 8, NPGEN )
2033:
2034:       do 760 I = NCOLS + 1, NCOLS + NPGEN
2035:         WK3(I) = LOG(EEE(LSCOL(I)))
2036: c##<2007/03/14:PN3:
2037:         WK5(I) = EEE(LSCOL(I))
2038: c##>
2039:       760 continue
2040:
2041:       do 770 NC = 1, NNCST
2042:         IN = INCST(NC,2)
2043:         if ( IN.ne.0 ) then
2044:           call GMICP1( IN, NPGEN, LSCOL(NCOLS+1),
2045:             & WK3(NCOLS+1), JGAMM, NUC, NMTP, NPATOM,
2046:             & NBANK, NSMIC, SMIC, KSPI, CXP, CXP, MCXP,
2047:             & KLB1P, KLB2P, XLBP2, IWK1, DWK2, IWK2 )
2048:         end if
2049:       770 continue
2050: c##<2007/03/14:PN3:
2051: C
2052:       if ( JPHNU.ne.0 ) then
2053:         do NC = 1, NUCPN
2054:           call GMICPN1( NC, NPGEN, LSCOL(NCOLS+1),
2055:             & WK5(NCOLS+1), NUC, NUCPN, NUCPN1, NMTPN,
2056:             & NBANK, NUC_MAX, NSMICPN, SMICPN, CXPN,
2057:             & CXPN, MCXPN, KLBPN1, KLBPN2, XLBPN1,
2058:             & EBOTLPN, DWK2, IWK1, IWK2, IWK3,
2059:             & SMCW, RNU )
2060:         end do
2061:       end if
2062: c##>
2063:
2064:       if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
2065:     end if
2066:   end if
2067: end if
2068: C
2069: C   ... pointwise response function ...
2070: C
2071:   if ( NSTAL.gt.0 ) then
2072:     if ( NCOLS.gt.0 ) then

```

src/mvp/neutr.f

```

2073:      JNP      = 1
2074:      call GTSCTL( JNP,  NCOLS, LSCOL, NBANK, EEE,
2075:      &          SGTAL, CRES,  KCRES, NSTAL, MCRES, DWK1, IWK1 )
2076:      end if
2077:      if ( NPGEN.gt.0 ) then
2078:      JNP      = 2
2079:      call GTSCTL( JNP,  NPGEN, LSCOL(NCOLS+1), NBANK, EEE,
2080:      &          SGTAL, CRES,  KCRES, NSTAL, MCRES, DWK1, IWK1 )
2081:      end if
2082:      end if
2083: C
2084:      NCOLS     = NCOLS + NPGEN
2085: C
2086: C-----
2087: C  TRANSFER PARTICLES TO FREE-FLIGHT STACK
2088: C-----
2089: C
2090: C
2091: C**** SET MMAC( ,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
2092: C          FOR THIS PARTICLE
2093: C
2094:      do 780 I = 1, NCOLS
2095:      MMAC(LSCOL(I),2)      = 0
2096:      780 continue
2097: C
2098: C .... SEND PARTICLES TO FREE-FLIGHT STACK ....
2099: C  EXCEPT FISSION NEUTRONS AND SECONDARY PHOTON
2100: C
2101:      NN      = NFFL(NZONE+1)
2102:      do 790 I = 1, NCOLS0
2103:      IP      = LSCOL(I)
2104:      LSFFL(NN+I) = IP
2105:      IZFFL(NN+I) = IZZ(IP)
2106:      790 continue
2107: C
2108: C ... reset flight path count to zero ...
2109:      if ( KIBNK(5).ne.0 ) then
2110:      do 800 I = 1, NCOLS0
2111:      IP      = LSCOL(I)
2112:      IBNK(IP,KIBNK(5)) = 0
2113:      800 continue
2114:      end if
2115: C
2116: *VOCL LOOP, SCALAR
2117:      do 810 I = 1, NCOLS0
2118:      NFFL(IZFFL(NN+I)) = NFFL(IZFFL(NN+I)) + 1
2119:      810 continue
2120: C
2121:      NFFL(NZONE+1) = NN + NCOLS0
2122: C
2123: C-----
2124: C  RUSSIAN ROULETTE KILL & SPLITTING BASED ON IMPORTANCE, WEIGHT
2125: C-----
2126: C
2127:      if ( JRRLT+JIMPT+JWWND.ne.0 ) then
2128:      ISF0     = NN + 1
2129:      NVARI    = NCOLS0
2130: c##<2007/03/14:PN3:
2131:      JJCOL    = 1
2132: c##>
2133: C
2134: c##<2007/03/14:PN3:
2135: c##      call SPLITB( 1, KPNEUT, ISF0, NVARI, IRAND, NPKIND, NBANK,
2136:      call SPLITB( JJCOL, KPNEUT, ISF0, NVARI, IRAND, NPKIND, NBANK,
2137: c##>

```

```

2138:      &          NZONE, NGROUP, NREG, NTIME, NEST, LSFFL, IZFFL, NFFL,
2139:      &          LSDED, NDEAD, NSMAC, NSMIC, XIMP, WKIL, WSRV, WTIME,
2140:      &          WLLIM, KZREG, NCNTR, WCNTR, NEVENT, XXX, YYY, ZZZ, AAA,
2141:      &          BBB, CCC, WWW, KKP, IZZ, IGG, TTT, ITT, LEVL, LZZ,
2142:      &          LPOS, LCRS, XIM, KLSF, EEE, IBREG, IBSPC, MB, NUC,
2143:      &          NSTAL, SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL, DBNK,
2144:      &          KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NKILD, WKILD, NSURV,
2145:      &          WSRV, NSPLT, WSPLT, R, DWK1, DWK2, WK3, IWK1, IWK2,
2146:      &          IWK3,
2147:      &          SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD,
2148:      &          WSC, NORDDS, NOPSDS,
2149: c##<2007/03/14:PN3:
2150:      &          NUCPN, NSMICPN, SMICPN,
2151: c##>
2152: c+beff
2153:      &          WWB, WSB, NPTBE,
2154:      &          WWBD, WSBD, WWLD, WSLD)
2155: c-beff
2156:      end if
2157: C
2158: C
2159: C .... SEND PARTICLES TO FREE-FLIGHT STACK ....
2160: C  FISSION NEUTRONS AND SECONDARY PHOTONS
2161: C
2162:      if ( NCOLS.gt.NCOLS0 ) then
2163:      N      = NCOLS - NCOLS0
2164:      NN      = NFFL(NZONE+1)
2165:      do 820 I = 1, N
2166:      IP      = LSCOL(I+NCOLS0)
2167:      LSFFL(NN+I) = IP
2168:      IZFFL(NN+I) = IZZ(IP)
2169:      820 continue
2170: C
2171: C ... reset flight path count to zero ...
2172:      if ( KIBNK(5).ne.0 ) then
2173:      do 830 I = 1, N
2174:      IP      = LSCOL(I+NCOLS0)
2175:      IBNK(IP,KIBNK(5)) = 0
2176:      830 continue
2177:      end if
2178: C
2179: *VOCL LOOP, SCALAR
2180:      do 840 I = 1, N
2181:      NFFL(IZFFL(NN+I)) = NFFL(IZFFL(NN+I)) + 1
2182:      840 continue
2183: C
2184:      NFFL(NZONE+1) = NN + N
2185: C
2186:      end if
2187: C
2188:      NCOLS     = 0
2189:      return
2190:      end

```


src/mvp/nid2nn.f

```

1:      subroutine NID2NN( NAME, IPART, N,      A,      CHA )
2: C=<MVP>=====
3: C purpose: determine nuclide or atom problem number of given
4: C      nuclide/atom name.
5: C      ( GMVP has dummy routine of the same name)
6: C      Returns number 0 if specified nuclide or atom is not used.
7: C called in: talin1
8: C=====
9: C
10: C i name : nuclide ID or atomic symbol
11: C i ipart : 1 = nuclide ID for neutron.
12: C      2 = Atomic symbol for photon/electron.
13: c##<2007/03/14:PN3:
14: C      3 = nuclide ID for photonuclear.
15: c##>
16: C o n      : nuclide/atom # in problem.
17: C i o a : task shared dynamic data area used as "REAL" type.
18: C i o cha : task shared dynamic data area used as "CHARACTER*4" type.
19: C-----
20:      character*(*) NAME
21: C
22:      real A(*)
23:      character*4 CHA(*)
24: C
25: CCCC include '../shared/INC/_ARRAY'
26: C
27:      include 'INC/_KPIDS'
28:      include 'INC/_NGPS'
29:      include 'INC/_CXSEC'
30: C
31:      N      = 0
32:      if ( IPART.eq.1 ) then
33:          call NID2N1( NAME, CHA(LNUCID), NUC, N )
34:      else if ( IPART.eq.2 ) then
35:          call NID2N2( NAME, A(LNATMT), NPATOM, N )
36: c##<2007/03/14:PN3:
37:      else if ( IPART.eq.3 ) then
38:          call NID2N3( NAME, CHA(LNCIDPN), NUCPNI, N )
39: c##>
40:      end if
41:      return
42:      end
43: C
44: C-----
45: C
46:      subroutine NID2N1( NAME, NUCID, NUC,      N )
47: C
48:      character*(*) NAME
49:      character*16 NUCID(NUC)
50: C
51:      do 100 I = 1, NUC
52:          if ( NAME.eq.NUCID(I) ) then
53:              N      = I
54:              return
55:          end if
56:      100 continue
57:      N      = 0
58:      return
59:      end
60: C
61: C-----
62: C
63:      subroutine NID2N2( NAME, NATMT, NPATOM, N )
64: C
65:      character*(*) NAME

```

```

66:      integer NATMT(NPATOM)
67: C
68:      external NATOMZ
69: C
70:      N      = 0
71:      M      = NATOMZ(NAME)
72: C
73:      if ( M.ne.0 ) then
74:          do 100 I = 1, NPATOM
75:              if ( NATMT(I).eq.M ) then
76:                  N      = I
77:                  return
78:              end if
79:          100 continue
80:      end if
81:      return
82:      end
83: c##<2007/03/14:PN3:
84: C
85: C=====
86: C
87:      subroutine NID2N3( NAME, NCIDPN,NUCPNI,N )
88: C
89:      character*(*) NAME
90:      character*16 NCIDPN(NUCPNI)
91: C
92:      do 100 I = 1, NUCPNI
93:          if ( NAME.eq.NCIDPN(I) ) then
94:              N      = I
95:              return
96:          end if
97:      100 continue
98:      N      = 0
99:      return
100:      end
101: c##>

```

src/mvp/nthsct.f

```

1:      subroutine NTHSCT( IOW, ICLN, IMT, CTH, JMNTR, JDEBG,
2:      & JMVMT, JTLLT, NGROUP, NZONE, NREG, NBANK, NUC,
3:      & NNK, NMT, KZREG, NEVENT, ETOP, EBOT,
4:      & ETHMAX, EWCUT,
5:      & CX, ICX, MCX, KLIB1, XLIB1,
6:      & KLIB2, XLIB2, LSCOL, NCOLS, LSDED, NDEAD,
7:      & WWW, EEE, KKP, IZZ, AAA, BBB, CCC,
8:      & IBREG, NECUT, WECUT, NCNTR, WCNTR,
9:      & JLATT, JHLAT, JIMPT, JTIME, NEST, XXX, YYY, ZZZ, TTT,
10:     & ITT, XIM, KLSF, LEVL, LZZ, LPOS, LCRS, IBSPC,
11:     & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
12:     & IRAND,
13:     & JSCTM, JRTCL, DNZON, IGG, ENGYB, NGP1, KSPI, SMAC, MB, ! scat
-mtx
14:     & NSMAC, LMAC, MMAC, SMIC, NSMIC, LMIC, KYPOS, DNFLXSM, ! scat
-mtx
15:     & RMICSM, JSCMU, RMIMU, WMIMU, ! scat
-mtx
16:     W IWK8, IWK9, IWK10, IWK11, IWK12, IWK13, IWK14, WK6,
17:     W WK7, WK8, WK9, WK10, WK4, WK5, WK3, WK3,
18:     W R,
19:     & JPRT, NPTDS, WWD, WWD2, NPTCS, WWR, ! pert
20:     & NGSP, WSD, WSC, NORDDS, NOPSDS, ! pert
21:     W KPNFG, JPHNU, ! photonuc
22:     & JBEFF, WWB, WSB, NPTBE, ! beff
23:     & WWBD, WSBD, WWLD, WSLD, ! beff
24:     & JEXDP, NRS) ! dopp-scat
25: C=====
26: C PURPOSE: ANALYSIS OF NON-THERMAL SCATTERING (NEUTRON)
27: C LSCOL(I) : STACK, ICLN(I) : COLLIDING NUCLIDE,
28: C IMT(I) : REACTION TYPE (MT), NCOLS : NUMBER OF PARTICLES
29: C -----> CTH(I) : SCATTERING ANGLE COSINE IN LAB SYSTEM,
30: C NCOLS IS MODIFIED BY ENERGY CUTOFF
31: C
32: C=====
33: C BANK DATA TO BE MODIFIED
34: C EEE : ENERGY
35: C=====
36: C CALLED IN: NEUTR
37: C CALL : RANU2, SEF5N, BSDEC3
38: C=====
39:     implicit real*8(A-H,O-Z)
40: C
41:     include 'INC/_KPIDS'
42: C
43:     integer JDEBG(*)
44: C
45: C**** INPUT
46: C
47:     integer ICLN(NBANK), IMT(NBANK)
48: C
49: C**** OUTPUT
50: C
51:     real CTH(NBANK)
52: C
53: C**** GEOMETRY ARRAY DATA
54: C
55:     integer KZREG(NZONE)
56: C
57: C**** CROSS SECTION DATA IN CX ARRAY
58: C
59:     real ETOP, EBOT, ETHMAX, EWCUT
60:     real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
61:     integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
62:     real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC) ! scat-mtx
63:     integer MMAC(NBANK,2), LMAC(8), LMIC(8), KSPI(NBANK,NUC) ! scat-mtx
64: C
65: C**** STACK
66: C
67:     integer LSCOL(NBANK), LSDED(NBANK)
68: C
69: C**** PARTICLE BANK
70: C
71:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
72:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
73:     real EEE(NBANK), WWW(NBANK), XIM(NBANK)
74:     integer KKP(NBANK)
75:     integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
76:     & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
77:     real*8 TTT(NBANK)
78:     integer ITT(NBANK)
79:     integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
80:     integer IGG(NBANK) ! scat-mtx
81: C
82:     real*8 DBNK(NBANK,MDBNK)
83:     integer KDBNK(0:MDBNK)
84:     integer IBNK(NBANK,MIBNK)
85:     integer KIBNK(0:MIBNK)
86:     integer KPNFG(NBANK) ! photonuc
87: C
88: C**** TALLY ARRAY
89: C
90:     real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
91: C
92:     real*8 NECUT(NREG), WECUT(NREG)
93:     real*8 WWD(NBANK,NPTDS,NORDDS), WWD2(NBANK,NPTDS) ! pert
94:     real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS) ! pert
95:     real*8 WWR(NBANK,NPTCS) ! pert
96:     real*8 WSC(NBANK,0:NGSP,NPTCS) ! pert
97:     real*8 WWB(NBANK) ! beff
98:     real*8 WSB(NBANK,0:NGSP) ! beff
99:     real*8 WWBD(NBANK) ! beff
100:    real*8 WSBD(NBANK,NGSP) ! beff
101:    real*8 WWLD(NBANK) ! beff
102:    real*8 WSLD(NBANK,NGSP) ! beff
103: C
104:    real DNZON(NUC,NZONE,2), ENGYB(*) ! scat-mtx
105:    real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*), ! scat-mtx
106:    & RMICSM(NGP1,NGP1,NREG,NUC,JSCTM,*) ! scat-mtx
107:    integer KYPOS(3) ! scat-mtx
108:    real*8 RMIMU(NGP1,NGP1+1,NREG,NUC,*), WMIMU(NGP1,NGP1+1,NREG,NUC,*) ! sca
t-mtx
109: C
110:     logical ISNDS ! dopp-scat
111: C
112: C**** WORK AREA
113: C
114:     integer IWK8(NBANK), IWK9(NBANK), IWK10(NBANK), IWK11(NBANK),
115:     & IWK12(NBANK), IWK13(NBANK), IWK14(NBANK)
116:     real WK2(NBANK), WK3(NBANK), WK4(NBANK)
117:     real WK6(NBANK), WK7(NBANK), WK8(NBANK)
118:     real WK9(NBANK), WK10(NBANK)
119: CM2014real R(7*NBANK)
120:     real R(8*NBANK)
121:     real*8 WK5(NBANK)
122: C
123: C real*8 ERFD ! dopp-scat with g-factor
124: C
125:     parameter( PI = 3.141592653589793200D+00 )
126:     parameter( PI2 = 2.0D0*PI )

```

src/mvp/nthsct.f

```

127:      parameter( PIH = 0.5D0*PI )
128: C
129:      DZERO = 0 ! scat-mtx
130: C
131: C**** SAMPLING OF SCATTERING ANGLE COSINE
132: C.... GATHER
133:      NNN = 0
134:      NN = 0
135: C
136: *VOCL LOOP,NOVREC
137:      do 100 I = 1, NCOLS
138:          IP = LSCOL(I)
139:          WK2(I) = EEE(IP)
140:          IMTI = ABS(IMT(I))
141:          NEANG = KLIB2(ICLN(I),IMTI,7)
142:          if ( NEANG.gt.0 ) then
143:              NNN = NNN + 1
144:              WK3(NNN) = EEE(IP)
145:              IWK8(NNN) = KLIB2(ICLN(I),IMTI,12)
146:              IWK9(NNN) = NEANG
147:              IWK13(NNN) = I
148:          else
149:              NN = NN + 1
150:              IWK14(NN) = I
151:          endif
152:      100 continue
153: C
154:      NF60 = NN
155: C
156: C.... SELECTION OF USED TABLE (INTERVAL) ----> IWK10
157: C      IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH A.D. ARE GIVEN
158: C      IWK9 : NUMBER OF INCIDENT ENERGY
159: C      WK3 : INCIDENT NEUTRON ENERGY
160: C
161:      call BSDEC3( CX, IWK9, WK5, IWK8, WK3, IWK10, NNN )
162: C
163:      call RANU2( IRAND, R, NNN*3, ICON )
164:      N = 0
165:      NF5 = 0
166:      N2 = NNN*2
167: C
168: *VOCL LOOP,NOVREC
169:      do 110 J = 1, NNN
170:          I = IWK13(J)
171: C
172:          if ( IWK10(J).eq.0 ) IWK10(J) = 1
173:          L2 = IWK8(J) + IWK10(J)
174:          L1 = L2 - 1
175:          INTE = MOD(ICX(L2+IWK9(J)),10)
176:          if ( INTE.eq.5 .or. INTE.eq.3 ) then
177: C..... LOG INTERPOLATION
178:              X1 = LOG(CX(L1)/WK3(J)) /LOG(CX(L1)/CX(L2))
179:          else
180: C..... LINEAR INTERPOLATION
181:              X1 = (CX(L1)-WK3(J)) / (CX(L1)-CX(L2))
182:          end if
183:          if ( X1.lt.0.0 ) X1 = 0.0
184:          if ( X1.gt.1.0 ) X1 = 1.0
185:
186: C/#IF ROUND OFF(NEAREST)
187: C97/11/07 IT = INT(R(J)+X1) + IWK10(J)
188:          IT = min(1,INT(R(J)+X1)) + IWK10(J)
189:          L3 =
190:          & MIN(INT(KLIB1(ICLN(I),7)*R(NNN+J))+1,KLIB1(ICLN(I),7))
191:          & + IWK8(J) + IWK9(J)*2 + KLIB1(ICLN(I),20)*(IT-1)

```

```

192:          XMU = (CX(L3-1)-CX(L3))*R(N2+J) + CX(L3)
193: C/#ELSE
194:      *      R1 = R(J) + X1
195:      *      IT = IWK10(J) + R1
196:      *      R2 = R(NNN+J)*KLIB1(ICLN(I),7)
197:      *      L3 = IWK8(J)+IWK9(J)*2
198:      *      + KLIB1(ICLN(I),20)*(IT-1) + R2
199:      *      XMU = (CX(L3+1)-CX(L3))*R(N2+J) + CX(L3)
200: C/#ENDIF
201:
202:          IP = LSCOL(I)
203:          ATW = XLIB1(ICLN(I),1)
204: C
205:          EE = WK3(J)
206:          INUC = ICLN(I)
207:          IMTI = ABS(IMT(I))
208: C
209: C..... BY USING REACTION KINEMATICS ----> EEE(LSCOL(I)), CTH(I)
210:          LCTMT = KLIB2(INUC,IMTI,6)
211: C+dopp
212: C
213: C..... Make a stack pointer for doppler scattering
214: C
215: C      if (JEXDP.ne.0 .and. IMTI.eq.2) then
216: C      if ( IMTI.eq.2 .and.
217: C      & ( JEXDP.eq.1 .or. JEXDP.eq.3 .or.
218: C      & ( JEXDP.eq.2 .and. ISNDS(INUC) ) ) ) then
219: C          N = N + 1
220: C          IWK10(N) = I
221: C          WK4(I) = ATW
222: C          WK5(I) = XMU
223: C-dopp
224: C      else if ( LCTMT.eq.2 ) then
225: C          GE = 1. - XLIB2(INUC,IMTI,2) /EE
226: C          if ( GE.lt.0.0 ) GE = 0.0
227: C          GE = ATW*SQRT(GE)
228: C          GM = GE*XMU
229: C          X2 = 1. + 2.*GM + GE*GE
230: C          SX2 = SQRT(X2)
231: C          CTH(I) = (1.+GM) /SX2
232: C          EEE(IP) = EE*XLIB1(INUC,7)*X2
233: C
234: C..... MAKE A STACK POINTER FOR SAMPLING EEE FROM FILE 5 DATA
235: C
236: C      else if ( LCTMT.eq.1 ) then
237: C          CTH(I) = XMU
238: C          NF5 = NF5 + 1
239: C          IWK13(NF5) = I
240: C
241: C..... MAKE A STACK POINTER FOR SAMPLING EEE FROM FILE 6 DATA
242: C
243: C      else
244: C          CTH(I) = XMU
245: C          NN = NN + 1
246: C          IWK14(NN) = I
247: C      end if
248: C
249: C..... MAKE A STACK POINTER FOR EXCEPTIONAL TREATMENT FOR SCATTERING
250: C      BY HYDROGEN (SIMPLIFIED FREE GAS MODEL IN VIM)
251: *VOCL STMT,IF(0)
252:      if ( EEE(IP).lt.ETHMAX.and.ATW.le.1.5 ) then
253:          N = N + 1
254:          IWK10(N) = I
255:          WK4(I) = ATW
256: CMM      WK5(I) = XMU

```

src/mvp/nthsct.f

```

257:          WK5(I) = 0.5*((ATW+1.)*(ATW+1.)*ETHMAX/EE-1.)/ATW-ATW)
258:      &          + 1.0
259:      end if
260: C
261: 110 continue
262: C
263: C
264: C..... MAKE A TEMPORARY STACK FOR SAMPLING FROM FILE 5 & 6 DATA
265: C      CONDENSE ICLN AND IMT ARRAYS
266: C      NF6 = NN
267: C      NSEF5 = NF6 + NF5
268: C
269: C      do 120 J = 1, NF6
270: C          IWK13(NF5+J) = IWK14(NF6-J+1)
271: C 120 continue
272: C      do 130 J = 1, NSEF5
273: C          IWK12(J) = ICLN(IWK13(J))
274: C          IWK11(J) = IMT(IWK13(J))
275: C          WK3(J) = WK2(IWK13(J))
276: C 130 continue
277: C
278: C
279: C**** EXCEPTIONAL TREATMENT FOR SCATTERING BY HYDROGEN
280: C      or Doppler scattering in resonance region
281: C
282: C      IWK10(N) : STACK POINTER LSCOL(IWK10)
283: C      ICLN (I) : COLLIDING NUCLIDE
284: C      WK2 (I) : INCIDENT ENERGY
285: C      WK4 (I) : ATW
286: C      WK5 (I) : SCATTERING ANGLE COSINE IN CM SYSTEM
287: C
288: CM2014
289:      NRS = 0
290: C
291:      if ( N.gt.0 ) then
292: CM2014
293:          if ( JEXDP.eq.2 .or. JEXDP.eq.3 ) then
294:              NRS = N
295:              do J = 1, NRS
296:                  IWK9(J) = IWK10(J)
297:              end do
298:          end if
299: CM2014
300: C
301: 140      NHS = N
302: C
303:          do 150 J = 1, NHS
304:              IWK14(J) = IWK10(J)
305: 150      continue
306: C
307: 160      continue
308: C
309:          call RANU2( IRAND, R, N*7, ICON )
310:          N2 = N*2
311:          N3 = N*3
312:          N4 = N*4
313:          N5 = N*5
314:          N6 = N*6
315: C
316:          NN = 0
317: C
318: *VOCL LOOP,NOVREC
319:          do 170 J = 1, N
320:              I = IWK10(J)
321:              IP = LSCOL(I)

```

```

322: C
323: c+dopp
324: CM2014
325: c      if (JEXDP.ne.0) then
326: c      &      if ( JEXDP.eq.1 .or. JEXDP.eq.3 .or.
327: c          ( JEXDP.eq.2 .and. ISNDS(ICLN(I)) ) ) then
328: c          AEKT = XLIB1(ICLN(I),9)*WK2(I)
329: c      else
330: c          if (WK2(I).le.5.) then
331: c              AEKT = XLIB1(ICLN(I),11)*WK2(I)
332: c          else
333: c              AEKT = XLIB1(ICLN(I),9)*WK2(I)
334: c          end if
335: c      end if
336: c      if (WK2(I).le.5.) then
337: c          AEKT = XLIB1(ICLN(I),11)*WK2(I)
338: c      else
339: c          AEKT = XLIB1(ICLN(I),9)*WK2(I)
340: c      end if
341: C
342:      if ( (4.*R(J)*R(J)).gt.(PI*AEKT*(1.-R(J))*(1.-R(J))) ) then
343:          X2 = -LOG(R(N+J)*R(N2+J)) /AEKT
344:      else
345:          COS2 = COS(PIH*R(N3+J))
346:          COS2 = COS2*COS2
347:          X2 = -(LOG(R(N+J))+LOG(R(N2+J))*COS2) /AEKT
348:      end if
349:      WK10(I) = SQRT(X2)
350: C..... WK10 = XT : (V-TARGET)/(V-NEUTRON)
351: C      (WK6,WK7,WK8):
352: C.....(AT,BT,CT) : DIRECTION OF TARGET NUCLIDE( ISOTROPIC )
353: C      WK6(I) = 2.*R(N4+J) - 1.0
354: C      SINT = SQRT(1.-WK6(I)*WK6(I))
355: C      FAI = PI2*R(N5+J)
356: C      WK7(I) = SINT*COS(FAI)
357: C      WK8(I) = SINT*SIN(FAI)
358: C      XMU = AAA(IP)*WK6(I) + BBB(IP)*WK7(I) + CCC(IP)*WK8(I)
359: C      WK9(I) = SQRT(1.+X2-2.*XMU*WK10(I))
360: C
361:      if ( WK9(I).lt.R(N6+J)*(1.+WK10(I)) ) then
362:          NN = NN + 1
363:          IWK10(NN) = I
364:      end if
365: 170      continue
366: C
367: C..... GO TO 160 : RESAMPLING OF XT AND XMU
368: C
369:      if ( NN.gt.0 ) then
370:          N = NN
371:          go to 160
372:      end if
373: C
374: C..... CALCULATE ENERGY AND DIRECTION AFTER COLLISION
375: C
376:          call RANU2( IRAND, R, NHS*3, ICON )
377: C
378: *VOCL LOOP,NOVREC
379:          do 180 J = 1, NHS
380:              I = IWK14(J)
381:              IP = LSCOL(I)
382: C
383:              A0 = AAA(IP)
384:              B0 = BBB(IP)
385:              C0 = CCC(IP)
386: C

```

src/mvp/nthsct.f

```

387: c+dopp
388: c      XA      = WK10(I)*WK4(I)
389: c      XMU     = WK5(I)*R(NHS*2+J) - 1.0
390: CM2014  if (JEXDP.ne.0) then
391:         if ( JEXDP.eq.1 .or. JEXDP.eq.3 .or.
392:           & ( JEXDP.eq.2 .and. ISNDS(ICLN(I)) ) ) then
393: C..... doppler scattering
394:         XA      = -WK10(I)
395:         XMU     = WK5(I)
396:         IHS     = 0
397:       else
398: C..... exceptional treatment for hydrogen
399:         XA      = WK10(I)*WK4(I)
400:         XMU     = WK5(I)*R(NHS*2+J) - 1.0
401:         IHS     = 1
402:       end if
403: c-dopp
404: C..... (A00,B00,C00) : MOVING DIRECTION OF CENTER OF MASS (H)
405: C      : Moving direction of neutron in CM system (dopp)
406:         A00     = A0 + WK6(I)*XA
407:         B00     = B0 + WK7(I)*XA
408:         C00     = C0 + WK8(I)*XA
409:         S       = 1./SQRT(A00*A00+B00*B00+C00*C00)
410:         A00     = A00*S
411:         B00     = B00*S
412:         C00     = C00*S
413: C
414:         S       = A00*A00 + B00*B00
415:         FAI     = PI2*R(J)
416: C
417: *VOCL STMT,IF(100)
418:       if ( S.gt.0. ) then
419:         SQT     = SQRT((1.-XMU*XMU)/S)
420:         COSP    = COS(FAI)*SQT
421:         SINP    = SIN(FAI)*SQT
422:         SS      = COSP*C00 + XMU
423: C.....(U1,U2,U3) : DIRECTION OF NEUTRON IN CM SYSTEM AFTER COLLISION
424: C
425:         U1      = A00*SS - B00*SINP
426:         U2      = B00*SS + A00*SINP
427:         U3      = XMU*C00 - S*COSP
428:       else
429:         SQT     = SQRT(1.-XMU*XMU)
430:         U1      = COS(FAI)*SQT
431:         U2      = SIN(FAI)*SQT
432:         U3      = XMU*C00
433:       end if
434:         SS      = 1./SQRT(U1*U1+U2*U2+U3*U3)
435:         U1      = U1*SS
436:         U2      = U2*SS
437:         U3      = U3*SS
438: C
439:         O1      = (WK9(I)*U1+WK10(I)*WK6(I))*WK4(I)
440:         O2      = (WK9(I)*U2+WK10(I)*WK7(I))*WK4(I)
441:         O3      = (WK9(I)*U3+WK10(I)*WK8(I))*WK4(I)
442: C..... (X1,X2,X3) : VECTOR V-NEW!/V-OLD! * (ATW+1) OF SCATTERED NEUTRON
443:         X1      = O1 + A0
444:         X2      = O2 + B0
445:         X3      = O3 + C0
446:         XX2     = X1*X1 + X2*X2 + X3*X3
447: C
448:         RATIO   = XX2*XLIB1(ICLN(I),7)
449:         ENEW    = WK2(I)*RATIO
450: C
451: c+dopp

```

```

452:       if ( WK2(I).gt.5. ) then
453:         C1      = 1.
454:       else
455:         AEKTEF   = XLIB1(ICLN(I),11)*WK2(I)
456:         AEKTL    = XLIB1(ICLN(I),9)*WK2(I)
457:         C1       = (AEKTEF-AEKTL)*(RATIO-1.D0)
458:       end if
459: CM2014  if (JEXDP.ne.0) then
460: c      if ( JEXDP.eq.1 .or. JEXDP.eq.3 .or.
461:   & ( JEXDP.eq.2 .and. ISNDS(ICLN(I)) ) ) then
462: C..... Doppler Scattering
463: c      XNOLM    = 1./SQRT(XX2)
464: c      AAA(IP)  = X1*XNOLM
465: c      BBB(IP)  = X2*XNOLM
466: c      CCC(IP)  = X3*XNOLM
467: c      EEE(IP)  = ENEW
468: c      if( EEE(IP).gt.ETOP) EEE(IP) = ETOP
469: C
470: c      if ( JSCMU.ne.0 .or. JSCTM.gt.1 ) then
471: c          XMU   = A0*AAA(IP) + B0*BBB(IP) + C0*CCC(IP)
472: c          CTH(I) = XMU + 3.
473: c      else
474: c          CTH(I) = 1.
475: c      end if
476: c      else
477: C ..... Exceptional Hydrogen Treatment
478: c      if ( WK2(I).gt.5. ) then
479: c          C1      = 1.
480: c      else
481: c          AEKTEF   = XLIB1(ICLN(I),11)*WK2(I)
482: c          AEKTL    = XLIB1(ICLN(I),9)*WK2(I)
483: c          C1       = (AEKTEF-AEKTL)*(RATIO-1.D0)
484: c      end if
485: C
486: C .... Rejected by short collision approximation
487: C      or when E>=ENEW>=ETHMAX in exceptional treatment of Hydrogen
488: CMMMM  IF( C1.LT. LOG(R(NHS+J))*WK4(I) ) THEN
489: c      if ( C1.lt.LOG(R(NHS+J))*WK4(I) .or.
490:   & ( ENEW.ge.ETHMAX.and.RATIO.le.1.0.and.IHS.eq.1 ) ) then
491: CM2014  if ( C1.lt.LOG(R(NHS+J))*WK4(I)
492: CM & .or. ( ENEW.ge.ETHMAX.and.RATIO.le.1.0 ) ) then
493:         NN      = NN + 1
494:         IWK10(NN) = I
495: C
496: C .... Accepted
497: C
498: C      ENEW      : WK2(Energy before collision) * RATIO
499: C      EEE(IP)   : neutron energy after collision ( ENEW)
500: C      (AAA,BBB,CCC) : modified
501: C      Scattering angle cosine in CM system CTH(I) = 1.0
502: C .... In exceptional treatment of hydrogen scattering (IHS=1)
503: C      When ENEW > INCIDENT ENERGY, ---> NO COLLISION
504: C      EEE(IP)   = INCIDENT ENERGY
505: C      SCATTERING ANGLE COSINE = 1.0
506: C
507: CM2014  else if ( RATIO.gt.1. ) then
508:         else if ( RATIO.gt.1. .and. IHS.eq.1 ) then
509:           EEE(IP) = WK2(I)
510:           CTH(I) = 1.0
511: CM2014  else if ( ENEW.lt.ETHMAX ) then
512:         else
513:           XNOLM   = 1./SQRT(XX2)
514:           AAA(IP) = X1*XNOLM
515:           BBB(IP) = X2*XNOLM
516:           CCC(IP) = X3*XNOLM

```

src/mvp/nthsct.f

```

517:      EEE(IP) = ENEW
518:      if( EEE(IP).gt.ETOP) EEE(IP) = ETOP
519:      if ( JSCMU.ne.0 .or. JSCTM.gt.1 ) then
520:        if ( IHS.eq.1 ) then
521:          X9 = 1 + 2*XMU*WK4(I) + WK4(I)**2
522:          XMU = (1 + XMU*WK4(I)) / SQRT(X9)
523:        else
524:          XMU = A0*AAA(IP) + B0*BBB(IP) + C0*CCC(IP)
525:        end if
526: C
527:        CTH(I) = XMU + 3.
528:      else
529:        CTH(I) = 1.
530:      end if
531: C      XNOLM = 1./SQRT(X9)
532: C      AAA(IP) = X1*XNOLM
533: C      BBB(IP) = X2*XNOLM
534: C      CCC(IP) = X3*XNOLM
535: C      end if
536: C      end if
537: C-dopp
538: 180 continue
539: C
540:      if ( NN.gt.0 ) then
541:        N = NN
542:        go to 140
543:      end if
544: C
545: C..... Weight Correction
546: C
547: CM2014 if (JEXDP.eq.2) then
548:      if (JEXDP.eq.2 .or. JEXDP.eq.3 ) then
549:        J0 = 0
550: CM2014 do J = 1, NHS
551:        do J = 1, NRS
552:          I = IWK14(J)
553:          I = IWK9(J)
554: C
555:          if (ISNDS(ICLN(I))) then
556: C..... only for nuclides with 0K elastic cross sections
557:            J0 = J0 + 1
558:            WK10(J0) = WK2(I)*WK9(I)*WK9(I)
559:            IWK9(J0) = KLIB1(ICLN(I),1)
560:            IWK8(J0) = KLIB1(ICLN(I),2)
561: CM2014            IWK14(J0) = IWK14(J)
562:            IWK14(J0) = I
563:          end if
564:        end do
565: C.....
566: CM2014 NHS = J0
567:        NRS = J0
568:
569: CM2014 call BSDEC3( CX, IWK8, WK5, IWK9, WK10, IWK10, NHS )
570:        call BSDEC3( CX, IWK8, WK5, IWK9, WK10, IWK10, NRS )
571:
572: CM2014 do J = 1, NHS
573:        do J = 1, NRS
574:          I = IWK14(J)
575:          KN = ICLN(I)
576:          IP = LSCOL(I)
577:          L1 = IWK9(J) + IWK10(J)
578:          D1 = (WK10(J)-CX(L1)) / (CX(L1-1)-CX(L1))
579:          D2 = 1 - D1
580:          L2 = KLIB2(KN,120,11) - KLIB2(KN,120,3)
581:          & + IWK10(J)

```

```

582: CM2014
583:      WK6(IP) = WWW(IP)
584: C
585:      if ( KLIB1(KN,4).eq.0 .or.
586:        & (KLIB1(KN,4).ne.0 .and.
587:        & (IWK10(J).lt.KLIB1(KN,16).or.
588:        & IWK10(J).ge.KLIB1(KN,17))
589:        & .and.
590:        & (KSPI(IP,KN).lt.KLIB1(KN,16).or.
591:        & KSPI(IP,KN).ge.KLIB1(KN,17)) )
592:        & ) then
593: C.... The cross section at 0 K
594:        XEL0K = CX(L2)*D1 + CX(L2+1)*D2
595:        if ( XEL0K.gt.0. .and. SMIC(IP,KN,LMIC(4)).gt.0. )
596:          & then
597:            CF = XEL0K / SMIC(IP,KN,LMIC(4))
598:            WWW(IP)= WWW(IP)*CF
599: C<<< dopp-scat with g-factor
600: C-      WWW(IP)= WWW(IP)*CF
601: C+..... B2= AE/kT
602: C+      if (WK2(I).le.5.) then
603: C+        B2 = XLIB1(KN,11)*WK2(I)
604: C+      else
605: C+        B2 = XLIB1(KN,9)*WK2(I)
606: C+      end if
607: C+      B2R = SQRT( B2 )
608: C+      GF = ( (B2+0.5)*ERFD(B2R) + SQRT(1./PI)*B2R*
609: C+      & EXP(-B2) ) / B2
610: C+      WWW(IP)= WWW(IP)*CF*GF
611: C>>> dopp-scat with g-factor
612:      end if
613:    end if
614:  end do
615: end if
616: end if
617: C
618: C**** SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA
619: C      IWK13(I) : PARTICLE STACK POINTER: LSCOL(IWK13(I))
620: C      IWK12(I) : COLLIDING NUCLIDE
621: C      IWK11(I) : REACTION TYPE (MT)
622: C      WK3(I) : INCIDENT NEUTRON ENERGY
623: C
624: C      IWK13,NSEF5,LSCOL,NCOLS ARE MODIFIED TO ADD (N,MN) NEUTRONS
625: C      IN SEF5N
626: C
627:      if ( NSEF5.gt.0 ) then
628: C      call SEF5N( IOW, IWK13, IWK12, IWK11, NSEF5, WK3, EEE, CX, CX,
629: C      & MCX, KLIB1, XLIB1, KLIB2, XLIB2, NBANK, NUC, NMT,
630: C      & IRAND, WK6, WK7, WK8, WK9, WK10, R, WK2, WK4, WK5,
631: C      & R(NBANK+1) )
632: C      call SEF5N( IOW, IWK13, IWK12, IWK11, NSEF5, WK3,
633: C      & LSCOL, NCOLS, NF60, NF6, NF5, CTH, EEE, CX, CX,
634: C      & MCX, KLIB1, XLIB1, KLIB2, XLIB2, NBANK, NUC, NMT,
635: C      & IRAND, JDEBG,
636: C      & JLATT, JHLAT, JTLLT, JIMPT, JTIME, NEST, LSDED, NDEAD,
637: C      & XXX,YYY,ZZZ, AAA,BBB,CCC, WWW, TTT, ITT, KKP,IZZ, XIM,
638: C      & KLSF, LEVL, LZZ, LPOS, LCRS, IBREG, IBSPC,
639: C      & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
640: C      & NCNTR, WCNTR, EWCUT,
641: CM201& IWK10, IWK14, WK6, WK7, WK8, WK9, WK10, R, IWK9, WK4,
642: CM      & WK5, R(NBANK+1),
643: C      & IWK8, IWK9, IWK10, WK7, WK8, WK9, WK10, R, R(NBANK+1),
644: C      & WK4, WK5, R(2*NBANK+1),
645: C      & JPRT, NPTDS, WWD, WWD2, NPTCS, WWR,
646: C      & NGSP, WSD, WSC, NORDDS, NOPSDS,

```

src/mvp/nthsct.f

```

647: c##<2007/03/14:PN4:
648:   &      KPNFG, JPHNU,
649: c##>
650: c+beff2
651:   &      JBEFF, WWB, WSB, NPTBE,
652:   &      WWBD, WSBD, WWLD, WSLD)
653: c-beff2
654: C
655: C**** MAKE ENERGY < ETOP
656: *VOCL LOOP,NOVREC
657:   do 190 J = 1, NSEF5
658:     IP = LSCOL(IWK13(J))
659:     EEE(IP) = MIN(EEE(IP),ETOP)
660:   190 continue
661:   end if
662: C
663: CM2014 .n. Replace IWK14 with IWK8 to keep IWK14 for later use
664:   if ( JSCTM.ne.0.or.(JSCTM.ne.0.and.JRTCL.ne.0) ) then
665:     do I = 1, NCOLS
666:       WK10(I) = EEE(LSCOL(I))
667:     end do
668: CM2014   call BSVDEC( ENGYB, NGP1+1, WK10, IWK14, NCOLS )
669:           call BSVDEC( ENGYB, NGP1+1, WK10, IWK8, NCOLS )
670: *VOCL LOOP,NOVREC
671:   do I = 1, NCOLS
672:     if ( WK10(I).lt.EBOT.or.WK10(I).gt.ETOP ) then
673:       if ( CTH(I).gt.1.0 ) CTH(I) = 1
674:       go to 195
675:     end if
676:     N = ICLN(I)
677:     IP = LSCOL(I)
678:     IMTI = abs(IMT(I))
679:     IMTJ = 0
680:     if ( IMTI.eq.2 ) then
681:       IMTJ = 1
682:     else if ( IMTI.ge. 6.and.IMTI.le. 9 ) then
683:       IMTJ = 3
684:     else if ( IMTI.eq.11 ) then
685:       IMTJ = 3
686:     else if ( IMTI.eq.16 ) then
687:       IMTJ = 3
688:     else if ( IMTI.ge.22.and.IMTI.le.23 ) then
689:       IMTJ = 2
690:     else if ( IMTI.eq.24 ) then
691:       IMTJ = 3
692:     else if ( IMTI.ge.28.and.IMTI.le.29 ) then
693:       IMTJ = 2
694:     else if ( IMTI.eq.30 ) then
695:       IMTJ = 3
696:     else if ( IMTI.ge.32.and.IMTI.le.36 ) then
697:       IMTJ = 2
698:     else if ( IMTI.eq.41 ) then
699:       IMTJ = 3
700:     else if ( IMTI.ge.44.and.IMTI.le.45 ) then
701:       IMTJ = 2
702:     else if ( IMTI.ge.51.and.IMTI.le.91 ) then
703:       IMTJ = 2
704:     end if
705:     if ( IMTJ.gt.0.and.KYPOS(IMTJ).gt.0 ) then
706:       if ( JSCTM.ne.0 ) then
707:         IE = IGG(IP)
708:         JE = IWK8(I)
709:         IR = KZREG(IZZ(IP))
710:         if ( JTLLT.ne.0 ) IR = IBREG(IP)
711:         XMU = CTH(I)

```

```

712:   if ( XMU.gt.1.0 ) then
713:     XMU = XMU - 3
714:     CTH(I) = 1
715:   end if
716:   XMU = XMU + 1
717:   WMIMU(IE,JE,IR,N,KYPOS(IMTJ)) =
718:   &      WMIMU(IE,JE,IR,N,KYPOS(IMTJ)) + WWW(IP)
719:   WMIMU(IE,NGP1+1,IR,N,KYPOS(IMTJ)) =
720:   &      WMIMU(IE,NGP1+1,IR,N,KYPOS(IMTJ)) + WWW(IP)
721:   RMIMU(IE,JE,IR,N,KYPOS(IMTJ)) =
722:   &      RMIMU(IE,JE,IR,N,KYPOS(IMTJ)) + WWW(IP)*XMU
723:   RMIMU(IE,NGP1+1,IR,N,KYPOS(IMTJ)) =
724:   &      RMIMU(IE,NGP1+1,IR,N,KYPOS(IMTJ)) + WWW(IP)*XMU
725: end if
726: if ( JSCTM.ne.0.and.JRTCL.ne.0 ) then
727:   DN = DNZON(N,IZZ(IP),1)
728:   if ( DN.gt.DZERO ) then
729:     if ( IMTJ.eq.1 ) then
730:       XSSJ = SMIC(IP,N,LMIC(4))
731:     else if ( IMTJ.eq.2 ) then
732:       L1 = KLIB1(N,1) + KSPI(IP,N)
733:       D1 = (WK2(I)-CX(L1))/(CX(L1-1)-CX(L1))
734:       D2 = 1 - D1
735:       L2 = KLIB2(N,IMTI,11)-KLIB2(N,IMTI,3)+KSPI(IP,N)
736:       XSSJ = CX(L2)*D1 + CX(L2+1)*D2
737:     else if ( IMTJ.eq.3 ) then
738:       XSSJ = SMIC(IP,N,LMIC(7))
739:     end if
740:     IE = IGG(IP)
741:     JE = IWK8(I)
742:     IR = KZREG(IZZ(IP))
743:     if ( JTLLT.ne.0 ) IR = IBREG(IP)
744:     DNF = DN*WWW(IP)/SMAC(IP,MMAC(IP,1),LMAC(1))
745:     DNFLXSM(IE,JE,IR,N,KYPOS(IMTJ)) =
746:     &      DNFLXSM(IE,JE,IR,N,KYPOS(IMTJ)) + DNF
747:     RMICSM(IE,JE,IR,N,KYPOS(IMTJ)) =
748:     c &      RMICSM(IE,JE,IR,N,KYPOS(IMTJ)) + XSSJ*DNF
749:     RMICSM(IE,JE,IR,N,1,KYPOS(IMTJ)) =
750:     &      RMICSM(IE,JE,IR,N,1,KYPOS(IMTJ)) + WWW(IP)
751:   if ( JSCTM.gt.1 ) then
752:     XMU = CTH(I)
753:     do J = 2, JSCTM
754:       XMUN = XMU**(J-1)*WWW(IP)
755:       RMICSM(IE,JE,IR,N,J,KYPOS(IMTJ)) =
756:       &      RMICSM(IE,JE,IR,N,J,KYPOS(IMTJ)) + XMUN
757:     end do
758:   end if
759: end if
760: end if
761: end if
762: 195 continue
763: end do
764: end if
765: C
766: CM2014
767:   if ( NRS.gt.0 ) then
768:     do I = 1, NCOLS
769:       IWK9(I) = 0
770:     end do
771:     do J = 1, NRS
772:       I = IWK14(J)
773:       c IWK9(I) = LSCOL(I)
774:       IWK9(I) = I
775:     end do
776:     NRS0 = 0

```

src/mvp/nthsct.f

```
777:      end if
778: C
779:      if ( ETHMAX.ge.EBOT ) then
780: *VOCL LOOP,NOVREC
781:      do 200 I = 1, NCOLS
782:          EEE(LSCOL(I)) = MAX(EEE(LSCOL(I)),EBOT)
783: CM2014
784:          if ( NRS.gt.0 .and. IWK9(I).gt.0 ) then
785:              NRS0 = NRS0 + 1
786: C          IWK14(NRS0) = LSCOL(I)
787:          IWK14(NRS0) = I
788:      end if
789: 200      continue
790: C
791:      else
792: C***** ENERGY CUT OFF
793: C
794:      NN = 0
795: C
796:      if ( JMNTR.ne.0 ) then
797: *VOCL LOOP,SCALAR
798:      do 210 I = 1, NCOLS
799:          J = LSCOL(I)
800:          if ( EEE(J).lt.EBOT ) then
801:              IZ = IZZ(J)
802:              if ( JTLL4.leq.0 ) then
803:                  IREG = KZREG(IZ)
804:              else
805:                  IREG = IBREG(J)
806:              end if
807:              NECUT(IREG) = NECUT(IREG) + 1
808:              WECUT(IREG) = WECUT(IREG) + WWW(J)
809:          end if
810: 210      continue
811:      end if
812: C
813: *VOCL LOOP,NOVREC
814:      do 220 I = 1, NCOLS
815:          LSC = LSCOL(I)
816:          if ( EEE(LSC).ge.EBOT ) then
817:              NN = NN + 1
818:              LSCOL(NN) = LSC
819:              CTH(NN) = CTH(I)
820:          if ( NRS.gt.0 .and. IWK9(I).gt.0 ) then
821:              NRS0 = NRS0 + 1
822: C          IWK14(NRS0) = LSC
823:          IWK14(NRS0) = NN
824:          end if
825:      else
826:          NDEAD = NDEAD + 1
827:          LSDED(NDEAD) = LSC
828:          NCNTR(8,KPNEUT) = NCNTR(8,KPNEUT) + 1
829:          WCNTR(8,KPNEUT) = WCNTR(8,KPNEUT) + WWW(LSC)
830: C##<2007/03/14:PN3:
831:          if ( KIBNK(17).ne.0 ) IBNK(LSC,KIBNK(17)) = 0
832:          if ( KIBNK(18).ne.0 ) IBNK(LSC,KIBNK(18)) = 0
833:          if ( KIBNK(19).ne.0 ) IBNK(LSC,KIBNK(19)) = 0
834: C##>
835: C##<2007/03/14:PN4:
836:          if ( JPHNU.ne.0 ) KPNFG(LSC) = 0
837: C##>
838:      end if
839: 220      continue
840: C
841:      NCOLS = NN

842: C
843:      end if
844: CM2014
845:      if ( NRS.gt.0 ) NRS = NRS0
846: C
847: C
848:      return
849:      end
```


src/mvp/nucfis.f

```
1:      subroutine NUCFIS( A, H, E, NS, INUC, EINCDD, IRAND, IERR, RWK )
2: C=<MVP>=====
3: C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA MT = 18,
4: C CALLED IN : SYSSRC, FSSAMP
5: C CALLS : SEF5N0
6: C
7: C <caution> this routine shares working arrays with SOURCE routine.
8: C-----
9: C
10: C arguments ( i= input, o = output, w = work )
11: C
12: C io A(*) : dynamic memory array (task shared)
13: C io H(*) : dynamic memory array (task local)
14: C o e(ns) : energy sampled
15: C i ns : number of particles whose energy E is sampled
16: C i inuc : nuclide #
17: C i eincdd: incident neutron energy
18: C o ierr : error code #
19: C w rwk(ns,5) : working array
20: C=====
21:      real A(*)
22:      real H(*)
23: C
24:      real*8 E(NS)
25: C
26:      real EINCDD
27:      real RWK(NS,5)
28: C
29: C ... working array pointers etc. ...
30: C
31: CCCC include '../shared/INC/_ARRAY'
32:      include 'INC/_WKSOU'
33:      include 'INC/_PXSEC'
34: C
35:      include 'INC/_KPIDS'
36:      include 'INC/_NGPS'
37: C
38:      include 'INC/_CXSEC'
39:      include '../shared/INC/_IOUNIT'
40:      include 'INC/_FLAGS'
41: C
42:      IERR = 0
43: C
44: C ... caution : take care for working arrays which can be used
45: C               in the following routine.
46: C At cuurent state ( July 1994: IWK1,IWK2, NNSRC & XSOUR are
47: C not allowed to use here. pointer: P1(5),P1(6),P1(12),P1(13))
48: C
49:      call SEF5N0( IPR, INUC, NS, EINCDD, E, A(LCX), A(LCX), MCX,
50:      & A(LKLIB1), A(LXLIB1), A(LKLIB2), A(LXLIB2), NUC, NMT,
51:      & IRAND, JDEBG,
52:      W H(P1(4)), H(P1(7)), H(P1(8)), H(P1(9)), H(P1(10)),
53:      W H(P1(11)), RWK(1,1) )
54: Cccc w h(p1(6)),h(p1(7)),h(p1(8)),h(p1(9)),h(p1(10)),h(p1(11)),
55: C
56:      return
57:      end
```

src/mvp/nuclst.f

```

1:      subroutine NUCLST( NAME,  INUCS, NN,      NNNUCS,A,      CHA,
2:      &                  IERR )
3: C=====
4: C purpose: make nuclide # list for a given nuclide name.
5: C called in: TALINI
6: C=====
7: C arguments (i=input, o=output, w=work)
8: C
9: C i NAME : nuclide name or combination of them jointed by "+" or removed
10: C      by "-".
11: C /example/
12: C ALL      : all nuclides given in the cross section block.
13: C FEN      : natural element of iron (photonuclear includes the
14: C isotopes.)
15: C FE6      : isotope of iron (Fe-56)
16: C FE*      : natural element and isotopes of iron (Fe-nat., 54, 56,
17: C 57, and 58)
18: C FEN+NN   : natural elements of iron and nickel
19: C FE*-FE8  : basically, isotopes of iron (Fe-54, 56, 57 except of Fe-58)
20: C o INUCS(NN) : array to store nuclide #'s
21: C      (a nuclide does not appear more than once in the list.)
22: C i NN      : size limit of INUCS(*)
23: C o NNNUCS : number of nuclide #'s stored in INUCS.
24: C i o A     : task shared dynamic data area used as "REAL" type.
25: C i o CHA   : task shared dynamic data area used as "CHARACTER*4" type.
26: C o IERR    : error code
27: C      .eq.0 : successful
28: C      .ne.0 : no nuclides found or more than NN matching nuclides
29: C      found.
30: C-----
31:      include 'INC/_KPIDS'
32:      include 'INC/_CXSEC'
33: C
34:      character NAME*(*), CHA(*)*4
35:      integer INUCS(NN)
36:      real A(*)
37: C
38: C-----
39: C
40: C ... actual procedure is in NCLST0
41: C
42:      call NCLST0( NAME, INUCS, NN, NNNUCS, IERR, NUC, NPATOM, NUCPN,
43:      &            CHA(LNUCID), A(LNATMT), CHA(LNCIDPN) )
44: C
45:      if ( NNNUCS.le.0.or.NNNUCS.gt.NN ) IERR = 2
46: C
47:      return
48:      end
49: C
50: C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
51: C
52:      subroutine NCLST0( NAME, INUCS, NN,      NNNUCS,IERR, NUC,
53:      &                  NPATOM,NUCPN, NUCID, NATMT, NCIDPN )
54: C=====
55: C purpose: working body of NUCLST routine.
56: C      (Make nuclide # list for a given nuclide name.)
57: C called in: NUCLST
58: C=====
59: C
60:      parameter (MAXTNUC=1000)
61:      include '../shared/INC/_IOUNIT'
62: C
63:      external CHSYMB
64:      character NAME*(*), NM*5, CHSYMB*2, NM2*2, NUCID(NUC)*16,
65:      &            NCIDPN(*)*16

```

```

66:      integer INUCS(NN), NATMT(NPATOM), KNUCS(MAXTNUC)
67: C
68: C-----
69: C
70: C ..... initialize
71:      NNNUCS = 0
72:      IERR = 0
73:      ISUBNM = 0
74:      ISS = 1
75:      ISSEND = LEN( NAME )
76:      KKNUCS = 0
77:      do K = 1, MAXTNUC
78:          KNUCS(K) = 0
79:      end do
80:      IKK0 = 0
81: C
82: C ..... start of new sub pattern
83:      100 if ( ISS.gt.ISSEND ) go to 150
84:      N2 = 0
85: C
86: C ..... nuclide name pattern may include more than one pattern string
87: C      separated by "+". -----> pattern1+pattern2+... or
88: C      separated by "-". -----> pattern1-pattern2-... or
89:      IKK1 = INDEX(NAME(ISS:), '+')
90:      IKK2 = INDEX(NAME(ISS:), '-')
91:      IKK3 = IKK0
92:      if ( IKK1.le.0.and.IKK2.le.0 ) then
93:          IKK = 0
94:          IKK0 = 0
95:      else if ( IKK1.gt.0.and.IKK2.gt.0 ) then
96:          IKK = min(IKK1,IKK2)
97:          IKK0 = 1
98:          if ( IKK.eq.IKK2 ) IKK0 = 2
99:      else if ( IKK1.gt.0 ) then
100:          IKK = IKK1
101:          IKK0 = 1
102:      else
103:          IKK = IKK2
104:          IKK0 = 2
105:      end if
106:      IEE = ISSEND
107:      if ( IKK.eq.1 ) then
108:          write(IMG,7000) ISS, NAME, (' ',I=1,ISS-1), '*'
109:          call CNTERR( 'WARNING' )
110:          ISS = ISS + 1
111:          go to 100
112:      else if ( IKK.gt.1 ) then
113:          IEE = ISS + IKK - 2
114:      end if
115: C
116: C ..... check the length and type of a nuclide name
117:      ISUBNM = ISUBNM + 1
118:      N = IEE - ISS + 1
119:      IVERF = 30
120:      if ( N.eq.5 ) then          ! normality in version 3.1 format
121:          IVERF = 31
122:          NM = NAME(ISS:IEE)
123:      else if ( N.eq.3 ) then      ! normality in version 3 format
124:          NM = NAME(ISS:IEE)//' '
125:      else if ( N.eq.2 ) then      ! abnormality, element
126:          NM = NAME(ISS:IEE)//'* '
127:      else if ( N.eq.1 ) then      ! abnormality, element
128:          NM = NAME(ISS:IEE)//'0* '
129:      else
130:          write(IMG,7001) N, NAME(ISS:IEE), ISUBNM

```

src/mvp/nuc1st.f

```

131:      call CNTERR( 'FATAL' )
132:      IERR      = 1
133:    end if
134:    IKW      = 0
135:    if ( NM(3:3).eq.'*' ) then          ! using a wildcard character
136:      IKW      = 1
137:    else if ( IVERF.eq.31.and.NM(3:5).eq.'000' ) then
138:      IKW      = 1
139:    else if ( NM(1:3).eq.'ALL' ) then    ! all nuclides in this calculation
140:      if ( ISS.eq.1 ) then
141:        IKW      = 2
142:      else
143:        write(MSG,7003) N, NAME(ISS:IEE), ISUBNM, ISS, IEE
144:        call CNTERR( 'FATAL' )
145:        IERR      = 4
146:      end if
147:    end if
148:  C
149:  C ..... neutron index
150:  if ( NUC.gt.0 ) then
151:    if ( IKW.eq.0 ) then
152:      do I = 1, NUC
153:        if ( ( IVERF.eq.30.and.NM(1:3).eq.NUCID(I)(1:3)).or.
154:          & ( IVERF.eq.31.and.NM(1:5).eq.NUCID(I)(1:5)) ) then
155:          KKNUCS = KKNUCS + 1
156:          if ( IKK3.eq.2 ) then
157:            KNUCS(KKNUCS) = -I
158:          else
159:            KNUCS(KKNUCS) = I
160:          end if
161:        end if
162:      end do
163:    else if ( IKW.eq.1 ) then
164:      do I = 1, NUC
165:        if ( NM(1:2).eq.NUCID(I)(1:2) ) then
166:          KKNUCS = KKNUCS + 1
167:          if ( IKK3.eq.2 ) then
168:            KNUCS(KKNUCS) = -I
169:          else
170:            KNUCS(KKNUCS) = I
171:          end if
172:        end if
173:      end do
174:    else if ( IKW.eq.2 ) then
175:      do I = 1, NUC
176:        KNUCS(I) = I
177:      end do
178:      KKNUCS = NUC
179:    end if
180:    N2      = N2 + NUC
181:  end if
182:  C
183:  C ..... photon index
184:  if ( NPATOM.gt.0 ) then
185:    if ( IKW.eq.2 ) then
186:      do I = 1, NPATOM
187:        KNUCS(N2+I) = N2 + I
188:      end do
189:      KKNUCS = KKNUCS + NPATOM
190:    else
191:      do I = 1, NPATOM
192:        NM2      = CHSYMB(NATMT(I))
193:        if ( NM2(2:2).eq.' ' ) NM2(2:2) = '0'
194:        if ( NM(1:2).eq.NM2 ) then
195:          KKNUCS = KKNUCS + 1

```

```

196:          if ( IKK3.eq.2 ) then
197:            KNUCS(KKNUCS) = - (N2 + I)
198:          else
199:            KNUCS(KKNUCS) = N2 + I
200:          end if
201:        end if
202:      end do
203:    end if
204:    N2      = N2 + NPATOM
205:  end if
206:  C
207:  C ..... photonuclear index
208:  if ( NUCPN.gt.0 ) then
209:    if ( NM(3:3).eq.'N' ) IKW = 1
210:    if ( IKW.eq.0 ) then
211:      do I = 1, NUCPN
212:        if ( ( IVERF.eq.30.and.NM(1:3).eq.NCIDPN(I)(1:3)).or.
213:          & ( IVERF.eq.31.and.NM(1:5).eq.NCIDPN(I)(1:5)) ) then
214:          KKNUCS = KKNUCS + 1
215:          if ( IKK3.eq.2 ) then
216:            KNUCS(KKNUCS) = - (N2 + I)
217:          else
218:            KNUCS(KKNUCS) = N2 + I
219:          end if
220:        end if
221:      end do
222:    else if ( IKW.eq.1 ) then
223:      do I = 1, NUCPN
224:        if ( NM(1:2).eq.NCIDPN(I)(1:2) ) then
225:          KKNUCS = KKNUCS + 1
226:          if ( IKK3.eq.2 ) then
227:            KNUCS(KKNUCS) = - (N2 + I)
228:          else
229:            KNUCS(KKNUCS) = N2 + I
230:          end if
231:        end if
232:      end do
233:    else if ( IKW.eq.2 ) then
234:      do I = 1, NUCPN
235:        KNUCS(N2+I) = N2 + I
236:      end do
237:      KKNUCS = KKNUCS + NUCPN
238:    end if
239:    N2      = N2 + NUCPN
240:  end if
241:  C
242:  if ( MAXTNUC.lt.KKNUCS ) then
243:    write(MSG,7002) MAXTNUC
244:    call CNTERR( 'FATAL' )
245:    IERR      = 3
246:    go to 180
247:  end if
248:  C
249:  C ..... proceed to next sub pattern
250:  ISS      = IEE + 2
251:  go to 100
252:  C
253:  150 continue
254:  N3      = NUC + NPATOM
255:  if ( NUC+NUCPN.gt.0 ) then
256:    do K = 1, KKNUCS
257:      K1      = KNUCS(K)
258:      if ( K1.lt.0 ) then
259:        K2      = abs(K1)
260:        if ( K2.le.NUC.or.K2.gt.N3 ) then

```

src/mvp/nuclst.f

```

261:      do I = 1, KKNUCS
262:        if ( K2.eq.KNUCS(I) ) KNUCS(I) = 0
263:      end do
264:      KNUCS(K) = 0
265:      end if
266:    end if
267:  end do
268:  do K = 1, KKNUCS
269:    K1 = KNUCS(K)
270:    if ( K1.lt.0 ) then
271:      K2 = abs(K1)
272:      if ( K2.gt.NUC.and.K2.le.N3 ) then
273:        do I = 1, KKNUCS
274:          if ( K2.eq.KNUCS(I) ) then
275:            NM2 = CHSYMB(NATMT(K2-NUC))
276:            if ( NM2(2:2).eq.' ' ) NM2(2:2) = '0'
277:            if ( NUC.gt.0 ) then
278:              do J = 1, KKNUCS
279:                K3 = KNUCS(J)
280:                if ( K3.gt.0.and.K3.le.NUC ) then
281:                  if ( NUCID(K3)(1:2).eq.NM2 ) go to 160
282:                end if
283:              end do
284:            end if
285:            if ( NUCPN.gt.0 ) then
286:              do J = 1, KKNUCS
287:                K3 = KNUCS(J)
288:                if ( K3.gt.N3 ) then
289:                  if ( NCIDPN(K3)(1:2).eq.NM2 ) go to 160
290:                end if
291:              end do
292:            end if
293:            KNUCS(I) = 0
294:          end if
295:        end do
296:        continue
297:      160 KNUCS(K) = 0
298:    end if
299:  end do
300:  else
301:    ! only photon interaction
302:    do K = 1, KKNUCS
303:      K1 = KNUCS(K)
304:      if ( K1.lt.0 ) then
305:        K2 = abs(K1)
306:        do I = 1, KKNUCS
307:          if ( K2.eq.KNUCS(I) ) KNUCS(I) = 0
308:        end do
309:        KNUCS(K) = 0
310:      end if
311:    end do
312:  end if
313: C
314:  do K = 1, KKNUCS
315:    if ( KNUCS(K).gt.0 ) then
316:      NNNUCS = NNNUCS + 1
317:      INUCS(NNNUCS) = KNUCS(K)
318:    end if
319:  end do
320: C
321:  180 continue
322:  return
323: C
324:  7000 format(/' !!!(NUCLST) The following nuclide name pattern',
325:    & ' has excessive "+" or "-" concatenator. ISS=',I4,' !!!'

```

```

326:    &/' Name: ',A
327:    &/' >> : ',I28(A:))
328:  7001 format(' XXX(NUCLST) Length of a nuclide name is greater than 3.'
329:    &/14X, 'N=', I3, ' NAME=', A, ' ISUBNM=', I3)
330:  7002 format(' XXX(NUCLST) Size of temporary array is less than total',
331:    & ' number of nuclide specifications (n,p,pn). MAXTNUC=',i6)
332:  7003 format(' XXX(NUCLST) "ALL" meaning the all nuclides in this',
333:    & ' calculation must be first position.'
334:    &/14X, 'N=', I3, ' NAME=', A, ' ISUBNM=', I3, ' ISS=',i5,
335:    & ' IEND=',i5)
336:  end
337: C
338: C #####
339: C
340:  subroutine CKPNMT( IWRK, ND, II, IA, IERR )
341: C=====
342: C purpose: check mt number with photonuclear cross section library.
343: C called in: TALINI
344: C=====
345: C arguments (i=input, o=output, w=work)
346: C
347: C i IWRK : integer array stored mt numbers of input.
348: C i ND : number of mt stored in IWRK.
349: C i II : location of checking photonuclear nuclide.
350: C i IA : task shared dynamic data area used as "REAL"/"INTEGER" type.
351: C o IERR : error code
352: C .eq.0 : all mt exist.
353: C .ne.0 : no
354: C-----
355:  include 'INC/_KPIDS'
356:  include 'INC/_CXSEC'
357: C
358:  integer IWRK(ND), IA(*)
359: C
360: C-----
361: C
362: C ... actual procedure is in CKPNMT0
363: C
364:  call CKPNMT0( IWRK, ND, II, IA(LKLBPN2), NUCPNI, NMTPN, IERR )
365: C
366:  return
367:  end
368: C
369: C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
370: C
371:  subroutine CKPNMT0( IWRK, ND, II, KLBPN2, NUCPNI, NMTPN, IERR )
372: C=====
373: C purpose: working body of CKPNMT routine.
374: C (Check mt with photonuclear cross section library.)
375: C called in: CKPNMT
376: C=====
377: C
378:  include '../shared/INC/_IOUNIT'
379:  integer IWRK(ND), KLBPN2(NUCPNI,NMTPN,13), JWRK(135)
380: C
381: C-----
382: C
383:  IERR = 0
384:  do I = 1, ND
385:    MT = IWRK(I)
386:    MTINFO = KLBPN2(II,MT,1)
387:    if ( MTINFO.le.0.or.MT.le.0.or.MT.gt.NMTPN ) then
388:      write(IMG,901) MT
389:      call CNTERR( 'FATAL' )
390:      IERR = 1

```

src/mvp/nuclst.f

```
391:          J          = 0
392:          do K = 1, NMTPN
393:             if ( KLBP2(II,K,1).gt.0 ) then
394:                J          = J + 1
395:                JWRK(J) = K
396:             end if
397:          end do
398:          write(IMG,902) J, (JWRK(K),K=1,J)
399:        end if
400:      end do
401:      return
402: C
403: 901 format('// XXX(ckpnmt) MT in MICROPN was not existed in photo',
404:           &'nuclear library.'
405:           &14x,'MT=',i5)
406: 902 format(14x,'allowable MT for this nuclide; number=',i3
407:           &/((14x,20i5))
408:           end
```

src/mvp/nxphr2.f

```

1:      subroutine NXPHR2(   IOW, A, H,  LSCLP, NIGEN,
2:      B      EEE , WWW ,
3:      B      DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
4:      B      NDBNK, NIBNK,
5:      X      CXP , ICXP , KLBP1, XLBP1, KLBP2, XLBP2,
6:      X      XLBE1, EEL , PBT , RKT , EBT , EBT ,
7:      X      IEBTP, WWAG , EEDG , EEEK , XK0 , XPIR2,
8:      X      KMAC , MMAC , SMIC , LMIC , KSPI ,
9:      B      IBP , ICLA , ISTB , NTBL , NRTYP,
10:     B      EIN , W , WE ,
11:     W      IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6,
12:     W      WW , R )
13: C-----
14: C
15: C  PURPOSE: Selection of photon reaction for next event estimator
16: C  CALLED IN: NXPHR
17: C  CALLS: RANU2
18: C  Updates:
19: C
20: C-----
21: C
22: C  === INTER-STACK DATA TREATED ===
23: C
24: C  COLLISION STACK ----->
25: C  (LSCLP,NCOLP)
26: C  (NOT REMOVED)
27: C  IBP(I) : bank pointer
28: C  ICLA(I) : collision atom
29: C  EIN(I) : incident energy
30: C  W(I) : imaginary photon weights
31: C  WE(I) : electron weights
32: C  NRTYP(10) : cummulative number of particles
33: C  1 : coherent scattering
34: C  2 : incoherent scattering
35: C  3 : pair creation
36: C  4 : photo electric
37: C  5 : electron by incoherent scattering in JGAMM=1
38: C  (NRTYP(2) >= NRTYP(5))
39: C  6 : electron by photo electric in JGAMM=1
40: C
41: C-----
42: C  implicit real*8 (A-H,O-Z)
43: C
44: C  real A(*)
45: C  real H(*)
46: C
47: C  include '../shared/INC/_SIZES'
48: C  include 'INC/_SIZES2'
49: C  include 'INC/_FLAGS'
50: C  include 'INC/_STACK'
51: C  include 'INC/_KPIDS'
52: C  include 'INC/_CXSEC'
53: C  include 'INC/_PXSEC'
54: C  include 'INC/_SBANK'
55: C  include '../shared/INC/_PTALY0'
56: C  include 'INC/_PTALY'
57: C  include 'INC/_PTALY2'
58: C
59: C**** CROSS SECTION DATA IN CXP ARRAY
60: C
61: C  real CXP(MCXP), XLBP1(NPATOM,10), XLBP2(NPATOM,NMTP,4)
62: C  integer ICXP(MCXP), KLBP1(NPATOM,20), KLBP2(NPATOM,NMTP,4)
63: C
64: C**** CROSS SECTION DATA FOR BREMSSTRAHLUNG OF ELECTRON
65: C

```

```

66:      real XLBE1(NPATOM,6)
67:      real EEL(NEE), RKT(MTOP), PBT(NEE,NMAT), EBT(MTOP,NEE,NMAT),
68:      &      EBT(MTOP-1,NEE,NMAT), WWAG(NMAT), EEDG(NMAT), EEEK(NMAT)
69:      integer IEBTP(2*(MTOP-1),NEE,NMAT)
70: C
71: C
72: C**** SIGMA BANK
73: C
74: C  real SMIC(NBANK,NUC,NSMIC)
75: C  integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8)
76: C
77: C**** STACK FOR REAL PARTICLE
78: C
79: C  integer LSCLP(NBANK)
80: C
81: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
82: C
83: C  real EEE(NBANK,2), WWW(NBANK,2)
84: C
85: C  real*8 DBNK(NBANK,NDBNK,2)
86: C  integer KDBNK(0:MDBNK)
87: C  integer IBNK(NBANK,NIBNK,2)
88: C  integer KIBNK(0:MIBNK)
89: C
90: C**** TALLY (MONITOR) ARRAY
91: C
92: C  real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
93: C
94: C  integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
95: C  &      IWK5(NBANK), IWK6(NBANK)
96: C  real R(3*NBANK)
97: C  real WW(NBANK)
98: C
99: C**** Prepared stack and bank
100: C
101: C  integer IBP(NBANK), ICLA(NBANK), ISTB(NBANK), NTBL(NBANK)
102: C  integer NRTYP(10)
103: C  real W(NBANK), EIN(NBANK), WE(NBANK)
104: C
105: C****
106: C
107: C
108: C  parameter( PI = 3.1415926535897932D+00 )
109: C  parameter( PI2 = 2.0D0*PI )
110: C  parameter( PIH = 0.5D0*PI )
111: C  parameter( PI4I = 1.0D0/(4.0D0*PI ) )
112: C
113: C  parameter( ONE = 0.999999D0 )
114: C  .... ENERGY OF ELECTRON .....
115: C  parameter( DM0C2 = 0.511D6 )
116: C
117: C
118: C-----
119: C  if ( JVMNT.ne.0 ) call VMNTR1( 5, NCOLP )
120: C-----
121: C
122: C-----
123: C  selection of reaction type generating imaginary particles.
124: C  Bank pointer, collision atom, weight
125: C  LSCLP ---> IBP(I), ICLA(I) W(I)
126: C  in order of coherent, incoherent, pair creation, photo-electric
127: C-----
128: C-----
129: C  POSITIONING FOR REACTIONS IN SMIC
130: C  1 : TOTAL

```

src/mvp/nxphr2.f

```

131: C      2 : COHERENT SCATTERING
132: C      3 : INCOHERENT SCATTERING
133: C      4 : PAIR PRODUCTION TOTAL
134: C      5 : PHOTOELECTRIC
135: C-----
136:
137: C
138: C      .... NF: Number of fissions, NS: Number of scattering
139: C
140: C      NCOH      = 0
141: C      NINC      = 0
142: C      NPPR      = 0
143: C      NPHE      = 0
144: C
145: C      ... gamma-ray reaction model .....
146: C
147: C      if ( JGAMM.eq.0 ) then
148: C      if ( JBREM.eq.0 ) then
149: C      call RANU2( IRAND, R, NCOLP, ICON )
150: C *VOCL LOOP,NOVREC
151: C      do 100 I = 1, NCOLP
152: C      MAT      = MMAC(LSCLP(I),1)
153: C      KN       = KMAC(LSCLP(I),MAT,1)
154: C      IWK6(LSCLP(I)) = KN
155: C      .... PS : scattering probability ...
156: C      photo electric reaction is treated as absorption
157: C      SIGT      = SMIC(LSCLP(I),KN,1)
158: C      SIGS      = SIGT - SMIC(LSCLP(I),KN,5)
159: C      PS        = SIGS / SIGT
160: C      .... WW = WWW * (scattering probability)
161: C      WW(LSCLP(I)) = WWW(LSCLP(I),1)*PS
162: C
163: C      P         = SIGS * R(I)
164: C      SIGCH      = SMIC(LSCLP(I),KN,2)
165: C      SIGIN      = SMIC(LSCLP(I),KN,3)
166: C      SIGPR      = SMIC(LSCLP(I),KN,4) + SIGIN
167: C      if ( P.lt.SIGCH ) then
168: C      NINC       = NINC + 1
169: C      IBP(NINC)  = LSCLP(I)
170: C      else
171: C      NPPR       = NPPR + 1
172: C      IWK3(NPPR) = LSCLP(I)
173: C      end if
174: C      100      continue
175: C
176: C      else
177: C      call RANU2( IRAND, R, NCOLP*2, ICON )
178: C *VOCL LOOP,NOVREC
179: C      do 110 I = 1, NCOLP
180: C      MAT      = MMAC(LSCLP(I),1)
181: C      KN       = KMAC(LSCLP(I),MAT,1)
182: C      IWK6(LSCLP(I)) = KN
183: C      .... PS : scattering probability ...
184: C      photo electric reaction is treated as absorption
185: C      SIGT      = SMIC(LSCLP(I),KN,1)
186: C      SIGS      = SIGT - SMIC(LSCLP(I),KN,5)
187: C      PS        = SIGS / SIGT
188: C      .... WW = WWW * (scattering probability)
189: C      WW(LSCLP(I)) = WWW(LSCLP(I),1)*PS
190: C
191: C      P         = SIGS * R(I)
192: C      SIGIN      = SMIC(LSCLP(I),KN,3)
193: C      if ( P.lt.SIGIN ) then
194: C      NINC       = NINC + 1
195: C      IBP(NINC)  = LSCLP(I)
196: C      else
197: C      NPPR       = NPPR + 1
198: C      IWK3(NPPR) = LSCLP(I)
199: C      end if
200: C      if ( R(NCOLP+I).le.PS ) then
201: C      IWK5(LSCLP(I)) = 1
202: C      .... election is created by photo electric
203: C      else
204: C      IWK5(LSCLP(I)) = 0
205: C      end if
206: C      110      continue
207: C      end if
208: C
209: C      .... generalized photon reaction model .....
210: C
211: C      else
212: C      call RANU2( IRAND, R, NCOLP, ICON )
213: C
214: C *VOCL LOOP,NOVREC
215: C      do 120 I = 1, NCOLP
216: C      MAT      = MMAC(LSCLP(I),1)
217: C      KN       = KMAC(LSCLP(I),MAT,1)
218: C      IWK6(LSCLP(I)) = KN
219: C      WW(LSCLP(I)) = WWW(LSCLP(I),1)
220: C
221: C      P         = SMIC(LSCLP(I),KN,1) * R(I)
222: C      SIGCH      = SMIC(LSCLP(I),KN,2)
223: C      SIGIN      = SMIC(LSCLP(I),KN,3) + SIGCH
224: C      SIGPR      = SMIC(LSCLP(I),KN,4) + SIGIN
225: C      if ( P.lt.SIGCH ) then
226: C      NCOH       = NCOH + 1
227: C      IBP(NCOH)  = LSCLP(I)
228: C      else if ( P.lt.SIGIN ) then
229: C      NINC       = NINC + 1
230: C      IWK2(NINC) = LSCLP(I)
231: C      else if ( P.lt.SIGPR ) then
232: C      NPPR       = NPPR + 1
233: C      IWK3(NPPR) = LSCLP(I)
234: C      else
235: C      NPHE       = NPHE + 1
236: C      IWK4(NPHE) = LSCLP(I)
237: C      end if
238: C      120      continue
239: C
240: C      if ( NINC.gt.0 ) then
241: C      do 130 I = 1, NINC
242: C      IBP(NCOH+I) = IWK2(I)
243: C      130      continue
244: C      end if
245: C      NINC      = NINC + NCOH
246: C      end if
247: C
248: C ... energy cut off for pair creation photons
249: C      if ( DM0C2.lt.EBOTP.and.JBREM.eq.0 ) then
250: C      NPPR      = 0
251: C      end if
252: C      if ( NPPR.gt.0 ) then
253: C      do 140 I = 1, NPPR
254: C      IBP(NINC+I) = IWK3(I)
255: C      140      continue
256: C      end if
257: C
258: C      NPPR      = NPPR + NINC
259: C
260: C      if ( NPHE.gt.0 ) then

```

src/mvp/nxphr2.f

```

261:      do 150 I = 1, NPHE
262:        IBP(NPPR+I) = IWK4(I)
263:      150 continue
264:      end if
265:      NPHE = NPHE + NPPR
266: C
267:      if ( JGAMM.ne.0.and.JBREM.ne.0 ) then
268:        NEPE = 0
269:        NEIN = 0
270: *VOCL LOOP,NOVREC
271:      do 160 I = 1, NINC
272:        IP = IBP(I)
273:        if ( IWK5(IP).eq.0 ) then
274:          NEPE = NEPE + 1
275:          IWK3(NEPE) = IP
276:        else
277:          NEIN = NEIN + 1
278:          IBP(NEIN) = IP
279:        end if
280:      160 continue
281: C .... electron production for imaginary photon by incoherent sc.
282: C 1 --- NEIN : by incoherent scattering
283: C NEIN+1 --- NINC : by photoelectric
284:      do 170 I = 1, NEPE
285:        IBP(NEIN+I) = IWK3(I)
286:      170 continue
287: C
288:      NEPP = 0
289: *VOCL LOOP,NOVREC
290:      do 180 I = NINC+1, NPPR
291:        IP = IBP(I)
292:        if ( IWK5(IP).eq.0 ) then
293:          NEPE = NEPE + 1
294:          IBP(NEIN+NEPE) = IP
295:        else
296:          NEPP = NEPP + 1
297:          IWK3(NEPP) = IP
298:        end if
299:      180 continue
300: C
301:      NEPE = NEIN + NEPE
302: C
303:      do 190 I = 1, NEPP
304:        IBP(NEPE+I) = IWK3(I)
305:      190 continue
306:      else
307:        NEIN = NINC
308:        NEPE = NINC
309:      end if
310: C
311: C
312: C=====
313: C gather particle data, and set pointer for cross sections
314: C=====
315: C
316:      do 200 I = 1, NPHE
317:        ICLA(I) = IWK6(IBP(I))
318:        W(I) = WW(IBP(I))
319:        EIN(I) = EEE(IBP(I),1)
320:      200 continue
321: C
322:      if ( JBREM.ne.0 ) then
323:        do 210 I = 1, NPHE
324:          WE(I) = WWW(IBP(I),1)
325:        210 continue

```

```

326:      end if
327: C
328:      NRTYP(1) = NCOH
329:      NRTYP(2) = NINC
330:      NRTYP(3) = NPPR
331:      NRTYP(4) = NPHE
332:      NRTYP(5) = NEIN
333:      NRTYP(6) = NEPE
334: C
335:      NIGEN = NPHE
336: C
337:      XK0 = XLBP1(1,4)
338:      XPIR2 = XLBP1(1,6)
339: C
340:      return
341:      end

```


src/mvp/nxphr3.f

```

1:      subroutine NXPHR3(   IOW, A, H, JAUGE,
2:      X      CXP , ICXP , KLB1, XLBP1, KLB2, XLBP2,
3:      X      XLBE1, EEL , PBT , RKT , EBT , EBT ,
4:      X      IEBTP, WWAG , EEDG , EEK ,
5:      D      KZMAT, IZZ ,
6:      B      NIGEN, IBP , ICLA , ISTB , NTBL , NRTYP,
7:      B      EIN , W , WE , EELE , EBRM ,
8:      B      A0 , B0 , C0 , AAE , BBE , CCE ,
9:      W      IWK1 , IWK2 , WK1 , WK2 , R )
10: C=====
11: C
12: C PURPOSE: Set some parameters and electron generation for next event
13: C estimator
14: C CALLED IN: NXPHR
15: C CALLS: RANU2, ELGEN2
16: C Updates:
17: C
18: C=====
19: C      IBP(I) : bank pointer
20: C      ICLA(I) : collision atom
21: C      EIN(I) : incident energy ( coherent, incoherent scatt.)
22: C      secondary photon energy ( photoe, pair creation)
23: C      EELE(I) : generated electron energy
24: C      EBRM(I) : generated bremsstrahlung photon energy
25: C      W(I) : particle weights including multiplicity
26: C      WE(I) : particle weights including multiplicity
27: C      NRTYP(10) : cumulative number of particles
28: C      1 : coherent scattering
29: C      2 : incoherent scattering
30: C      3 : pair creation
31: C      4 : photo electric
32: C      5 : electron by incoherent scattering in JGAMM=1
33: C      (NRTYP(2) >= NRTYP(5))
34: C      6 : electron by photo electric in JGAMM=1
35: C
36: C=====
37: C      implicit real*8 (A-H,O-Z)
38: C
39: C      real A(*)
40: C      real H(*)
41: C
42: C      include '../shared/INC/_SIZES'
43: C      include 'INC/_SIZES2'
44: C      include 'INC/_FLAGS'
45: C      include 'INC/_STACK'
46: C      include 'INC/_KPID5'
47: C      include 'INC/_CXSEC'
48: C      include 'INC/_PXSEC'
49: C      include 'INC/_SBANK'
50: C      include '../shared/INC/_PTALY0'
51: C      include 'INC/_PTALY'
52: C      include 'INC/_PTALY2'
53: C
54: C**** CROSS SECTION DATA IN CXP ARRAY
55: C
56: C      real CXP(MCXP), XLBP1(NPATOM,10), XLBP2(NPATOM,NMTP,4)
57: C      integer ICXP(MCXP), KLB1(NPATOM,20), KLB2(NPATOM,NMTP,4)
58: C
59: C**** CROSS SECTION DATA FOR BREMSSTRAHLUNG OF ELECTRON
60: C
61: C      real XLBE1(NPATOM,6)
62: C      real EEL(NEE), RKT(MTOP), PBT(NEE,NMAT), EBT(MTOP,NEE,NMAT),
63: C      & EBT(MTOP-1,NEE,NMAT), WWAG(NMAT), EEDG(NMAT), EEK(NMAT)
64: C      integer IEBTP(2*(MTOP-1),NEE,NMAT)
65: C

```

```

66: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
67: C
68: C      integer IZZ(NBANK,2)
69: C**** GEOMETRY
70: C      integer KZMAT(NZONE,2)
71: C
72: C**** working array
73: C
74: C      integer IWK1(NBANK), IWK2(NBANK)
75: C      real R(4*NBANK)
76: C      real WK1(NBANK), WK2(NBANK)
77: C
78: C**** Prepared stack and bank
79: C
80: C      integer IBP(NBANK), ICLA(NBANK), ISTB(NBANK), NTBL(NBANK)
81: C      integer NRTYP(10)
82: C      real W(NBANK), EIN(NBANK), WE(NBANK), EELE(NBANK), EBRM(NBANK)
83: C      real*8 A0(NBANK), B0(NBANK), C0(NBANK)
84: C      real*8 AAE(NBANK), BBE(NBANK), CCE(NBANK)
85: C
86: C      parameter( PI = 3.1415926535897932D+00 )
87: C      parameter( PI2 = 2.0D0*PI )
88: C      parameter( PIH = 0.5D0*PI )
89: C      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
90: C
91: C      parameter( ONE = 0.999999D0 )
92: C      ... ENERGY OF ELECTRON .....
93: C      parameter( DMOC2 = 0.511D6 )
94: C
95: C
96: C=====
97: C      if ( JVMNT.ne.0 ) call VMNTRI( 5, NCOLP )
98: C=====
99: C
100: C      ... AUGER ELECTRONS ARE TAKEN INTO ACCOUNT WHEN JAUGE = 1
101: C      Now set JAUGE=0 in NXPHR
102: C
103: C      NCOH = NRTYP(1)
104: C      NINC = NRTYP(2)
105: C      NPPR = NRTYP(3)
106: C      NPHE = NRTYP(4)
107: C      NEIN = NRTYP(5)
108: C      NEPE = NRTYP(6)
109: C      check
110: C      write(6,*) ' nxphr3, ncoh,ninc,nppr,nphe,nein,nepe ',
111: C      & ncoh,ninc,nppr,nphe,nein,nepe
112: C      write(6,*) ' ein=',(ein(i),i=1,nphe)
113: C
114: C=====
115: C      set parameters
116: C=====
117: C
118: C      if ( NCOH.gt.0 ) then
119: C      ..... SELECT FORM FACTOR TABLE
120: C      ..... NFFE ..... NFFE IS ALWAYS 1 .....
121: C
122: C      NFFE = 1
123: C      *VOCL LOOP,NOVREC
124: C      do 100 I = 1, NCOH
125: C      ..... NFF .....
126: C      NTBL(I) = KLB1(ICLA(I),7)
127: C      ..... C-TABLE ADDRESS .....
128: C      ISTB(I) = KLB1(ICLA(I),10) + NFFE
129: C      W(I) = W(I) * PI4I
130: C      100 continue

```

src/mvp/nxphr3.f

```

131:      end if
132: C
133:      if ( NINC.gt.NCOH ) then
134:        do 110 I = NCOH+1, NINC
135:          NTBL(I) = KLBP1(ICLA(I),6)
136:          ISTB(I) = KLBP1(ICLA(I),9)
137:          W(I) = W(I) * PI4I
138: 110      continue
139:      end if
140: C
141: C ... energy and weight of annihilation gamma are set
142: C below electron generation
143: C
144: C
145:      if ( NPHE.gt.NPPR ) then
146: C
147:        NEDGE = 2
148: C
149:        NPE = NPHE - NPPR
150: C
151:        if ( JBREM.eq.0 ) then
152:          call RANU2( IRAND, R, 2*NPE, ICON )
153: C
154: *VOCL LOOP,NOVREC
155:        do 140 J = 1, NPE
156:          I = NPPR + J
157:          LSP1 = KLBP1(ICLA(I),12) + 1
158:          EDGE1 = CXP(LSP1)
159:          EDGE2 = CXP(LSP1+1)
160: C
161:          .... OUTGOING ENERGY STORAGE ....
162:          EINI = EIN(I)
163:          EIN(I) = 0.0
164: C
165:          if ( EINI.gt.EDGE2 ) then
166:            ND1 = ICXP(LSP1+NEDGE)
167:            ND2 = ICXP(LSP1+NEDGE+1)
168:            if ( EINI.ge.EDGE1 ) then
169:              ND = ND1
170:              LS = LSP1 + 3*NEDGE + 1
171:              LF = LS + 3*(ND1+ND2)
172:            else
173:              ND = ND2
174:              LS = LSP1 + 3*NEDGE + 1 + 3*ND1
175:              LF = LS + 3*ND2
176:            end if
177: C/IF ROUND OFF(NEAREST)
178:            L = MIN(INT(ND*R(J)),ND-1)
179:            K = MIN(1,INT(1+R(J+NPE)-CXP(LS
180:              &
181:              +L))) + LS + ND + 2*L
182: C/ELSE
183:            L = ND*R(J)
184:            R1 = R(J+NPE) - CXP(LS+L)
185:            K = LS + ND + 2*L + 1 + R1
186: C/ENDIF
187:            EIN(I) = CXP(ICXP(K)+LF-1)
188:          end if
189: 140      continue
190: C
191:      else
192:        call RANU2( IRAND, R, 3*NPE, ICON )
193: C
194: *VOCL LOOP,NOVREC
195:        do 150 J = 1, NPE

```

```

196:          I = NPPR + J
197:          LSP1 = KLBP1(ICLA(I),12) + 1
198:          EDGE1 = CXP(LSP1)
199:          EDGE2 = CXP(LSP1+1)
200: C
201:          .... OUTGOING ENERGY STORAGE ....
202:          EINI = EIN(I)
203:          EIN(I) = 0.0
204: C
205:          MK = KZMAT(IZZ(1BP(I),1),1)
206:          EG = EEDG(MK)
207:          ED = 0.0
208:          EAUGER = 0.0
209: C
210:          if ( EINI.gt.EDGE2 ) then
211:            .... AUGER ELECTRON ....
212:            if ( R(NPE*2+J).gt.WWAG(MK).and.JAUGE.eq.1 ) then
213:              EAUGER = EEEK(MK)
214:            else
215:              ND1 = ICXP(LSP1+NEDGE)
216:              ND2 = ICXP(LSP1+NEDGE+1)
217:              if ( EINI.ge.EDGE1 ) then
218:                ND = ND1
219:                LS = LSP1 + 3*NEDGE + 1
220:                LF = LS + 3*(ND1+ND2)
221:                EG = EDGE1
222:              else
223:                ND = ND2
224:                LS = LSP1 + 3*NEDGE + 1 + 3*ND1
225:                LF = LS + 3*ND2
226:                EG = EDGE2
227:              end if
228: C/IF ROUND OFF(NEAREST)
229:              L = MIN(INT(ND*R(J)),ND-1)
230:              K = MIN(1,INT(1+R(J+NPE)-CXP(LS
231:                &
232:                +L))) + LS + ND + 2*L
233: C/ELSE
234:              L = ND*R(J)
235:              R1 = R(J+NPE) - CXP(LS+L)
236:              K = LS + ND + 2*L + 1 + R1
237: C/ENDIF
238:              EIN(I) = CXP(ICXP(K)+LF-1)
239:            end if
240:          end if
241: C
242:          EELE(I) = EINI - EG - ED
243:          EBRM(I) = EINI
244: 150      continue
245:      end if
246: C
247:      do 160 I = NPPR+1, NPHE
248:        if ( EIN(I).ge.EBOTP ) then
249:          W(I) = W(I) * PI4I
250:        else
251:          W(I) = 0.0
252:        end if
253: 160      continue
254:      end if
255: C
256: C=====
257: C generation of electrons
258: C=====
259: C
260:      if ( JBREM.ne.0 ) then

```

src/mvp/nxphr3.f

```

261:      JSTK = 0
262:
263:      if ( JGAMM.eq.1 ) then
264:        LSPE = NEIN + 1
265:        NPE = NEPE - NEIN
266:        LSPP = NEPE + 1
267:        NPP = NPPR - NEPE
268:      else
269:        LSPP = NINC + 1
270:        NPP = NPPR - NINC
271:        LSPE = NPPR + 1
272:        NPE = NPHE - NPPR
273:      end if
274: C
275:      if ( NPE.gt.0 ) then
276:        if ( JGAMM.ne.0 ) then
277:          call RANU2( IRAND, R, NPE, ICON )
278: C
279: *VOCL LOOP,NOVREC
280:       do 200 J = 1, NPE
281:         I = LSPE + J - 1
282:         LSP1 = KLBPl(ICLA(I),12) + 1
283:         EDGE1 = CXP(LSP1)
284:         EDGE2 = CXP(LSP1+1)
285: C      .... OUTGOING ENERGY STORAGE ....
286: C
287:         MK = KZMAT(IZZ(IBP(I),1),1)
288:         EG = EEDG(MK)
289:         ED = 0.0
290:         EAUGER = 0.0
291: C
292:         if ( EIN(I).gt.EDGE2 ) then
293: C      .... AUGER ELECTRON ....
294:           if ( R(J).gt.WWAG(MK).and.JAUGE.eq.1 ) then
295:             EAUGER = EEEK(MK)
296:           else
297:             if ( EIN(I).ge.EDGE1 ) then
298:               EG = EDGE1
299:             else
300:               EG = EDGE2
301:             end if
302:           end if
303:         end if
304: C
305:         EELE(I) = EIN(I) - EG - ED
306:         EBRM(I) = EIN(I)
307:       200 continue
308:     end if
309: C
310:     ITYP = 1
311:     call ELGEN2 ( IOW, ITYP, JSTK , IRAND,
312: S      ISEL , NPE , NBANK,
313: B      A0(LSPE) , B0(LSPE) , C0(LSPE) ,
314: B      AAE(LSPE) , BBE(LSPE) , CCE(LSPE) ,
315: B      EBRM(LSPE) , EELE(LSPE) , WE(LSPE) ,
316: B      COSL , EOUT , AA , BB , CC ,
317: W      R , WK1 , IWK2 , WK2 )
318:   end if
319: C
320:   if ( NPP.gt.0 ) then
321:     ITYP = 3
322: C
323:     call ELGEN2 ( IOW, ITYP, JSTK , IRAND,
324: S      ISEL , NPP , NBANK,
325: B      A0(LSPP) , B0(LSPP) , C0(LSPP) ,

```

```

326:   B      AAE(LSPP) , BBE(LSPP) , CCE(LSPP) ,
327:   B      EIN(LSPP) , EELE(LSPP) , WE(LSPP) ,
328:   B      COSL , EOUT , AA , BB , CC ,
329:   W      R , WK1 , IWK2 , WK2 )
330:   end if
331: C
332: C ... bremsstrahlung photon by photo electric and pair creation
333: C
334: C
335:       NELC = NPHE - NEIN
336: C
337:       if ( NELC.gt.0 ) then
338:         call BSVDEC( EEL, NEE, EELE(NEIN+1), IWK1(NEIN+1), NELC )
339: C
340:         call RANU2( IRAND, R, NELC*4, ICON )
341:         N2 = NELC*2
342:         N3 = NELC*3
343:         MTOP1 = MTOP - 1
344: C
345: *VOCL LOOP,NOVREC
346:       do 210 J = 1, NELC
347:         I = NEIN + J
348:         R1 = (EEL(IWK1(I))-EELE(I)) /
349: &           (EEL(IWK1(I))-EEL(IWK1(I)+1))
350:         if ( R1.gt.1.0 ) R1 = 1.0
351:         IMED = KZMAT(IZZ(IBP(I),1),1)
352:         YIELD = (1.0-R1)*PBT(IWK1(I),IMED)
353: &           + R1*PBT(IWK1(I)+1,IMED)
354: C
355:         RR = R1 + R(J)
356:         M = IWK1(I) + RR
357:         LS = MTOP1*R(NELC+J) + 1
358: C/#IF ROUNDOff(NEAREST)
359:         M = MIN(M,IWK1(I)+1)
360:         LS = MIN(LS,MTOP1)
361: C/#ENDIF
362:         LS2 = LS*2
363:         LS2 = min(LS2,int(LS2 + R(N2+J) - EBTPLS,M,IMED))
364:         LS3 = IEBTPLS2,M,IMED)
365: C
366:         EBTX : PROBABILITY FOR K/T RATIO SEARCH.
367:         RKT1 : K/T RATIO.
368: C
369:         EBT1 = EBT(LS3,M,IMED)
370:         EBT2 = EBT(LS3+1,M,IMED)
371:         EBTX = EBT2 + (EBT1-EBT2)*R(N3+J)
372:         EBTA = (EBT1+EBT2) / 2.0
373:         if ( LS3.eq.1 .or. EBTX.gt.EBTA ) then
374:           L = LS3
375:           EBT3 = EBT(L+2,M,IMED)
376:         else
377:           L = LS3 - 1
378:           EBT3 = EBT2
379:           EBT2 = EBT1
380:           EBT1 = EBT(L,M,IMED)
381:         end if
382:         RKT1 = (EBTX-EBT2)*(EBTX-EBT3)*RKT(L) /
383: &           ((EBT1-EBT2)*(EBT1
384: &           -EBT3)) + (EBTX-EBT1)*(EBTX-EBT3)*RKT(L+1) /
385: &           ((EBT1-EBT2)*(EBT3
386: &           -EBT2)) + (EBTX-EBT1)*(EBTX-EBT2)*RKT(L+2) /
387: &           ((EBT3-EBT1)*(EBT3-EBT2))
388:         EBRM(I) = EELE(I)*RKT1
389: C
390: C.... energy cut off

```

src/mvp/nxphr3.f

```
391:          if ( EBRM(I).lt.EBOTP ) then
392:             WE(I)  = 0.0
393:          else
394:             WE(I)  = WE(I) * YIELD * PI4I
395: C ... change electron energy
396:          EELE(I)  = SQRT(EELE(I)*(EELE(I)+2.*DMOC2))
397:          &        / (EELE(I)+DMOC2)
398:          end if
399: C
400: 210      continue
401: C
402:      end if
403: C
404:      do 220 I = NCOH+1, NEIN
405:          WE(I)  = WE(I) * PI4I
406: 220      continue
407: C
408:      end if
409: C
410: C .... set energy and weight of annihilation gamma
411: C
412:      if ( NPPR.gt.NINC ) then
413:          if ( EBOTP.le.DMOC2 ) then
414:              do 300 I = NINC+1, NPPR
415:                  W(I)  = W(I) * PI4I * 2.0
416:                  EIN(I) = DMOC2
417:              300      continue
418:          else
419:              do 310 I = NINC+1, NPPR
420:                  W(I)  = 0.0
421:              310      continue
422:          end if
423:      end if
424: C
425:      return
426:      end
```

src/mvp/nxphr4.f

```

1:      subroutine NXPHR4(   IOW, A, H, XK0, XPIR2,
2:      X      CXP   , ICXP  , KLBP1, XLBP1, KLBP2, XLBP2,
3:      X      XLBE1, EEL   , PBT  , RKT  , EBT  , EBT  ,
4:      X      IEBTP, WWAG  , EEDG  , EEEK  ,
5:      D      KZMAT, IZZ   , SMIC  ,
6:      B      IBP   , ICLA  , ISTB  , NTBL  , NRTYP,
7:      B      EIN   , W     , WE    , EELE  , EBRM  ,
8:      B      A0    , B0    , C0    , AAE   , BBE  , CCE  ,
9:      B      ISEL  , NSEL  , AA    , BB    , CC    , COSL ,
10:     B      WW    , EEEJ  , WWE   ,
11:     W      IWK1  , IWK2  , IWK3  , IWK4  , WK1   , WK2   ,
12:     W      R     )
13: C=====
14: C
15: C  PURPOSE: calculation of probabilities for next event estimator
16: C  CALLED IN: NXPHR
17: C  CALLS: RANU2, ELGEN2
18: C  Updates:
19: C
20: C=====
21: C      IBP(I)   : bank pointer
22: C      ICLA(I)  : collision atom
23: C      EIN(I)   : incident energy
24: C      EELE(I)  : generated electron energy
25: C      EBRM(I)  : generated bremsstrahlung photon energy
26: C      W(I)     : particle weights including multiplicity
27: C      WE(I)    : particle weights including multiplicity
28: C      NRTYP(10): cummulative number of particles
29: C      1 : coherent scattering
30: C      2 : incoherent scattering
31: C      3 : pair creation
32: C      4 : photo electric
33: C      5 : electron by incoherent scattering in JGAMM=1
34: C      (NRTYP(2) >= NRTYP(5))
35: C      6 : electron by photo electric in JGAMM=1
36: C
37: C=====
38: C      implicit real*8 (A-H,O-Z)
39: C
40: C      real A(*)
41: C      real H(*)
42: C
43: C      include '../shared/INC/_SIZES'
44: C      include 'INC/_SIZES2'
45: C      include 'INC/_FLAGS'
46: C      include 'INC/_STACK'
47: C      include 'INC/_KPIDS'
48: C      include 'INC/_CXSEC'
49: C      include 'INC/_PXSEC'
50: C      include 'INC/_SBANK'
51: C      include '../shared/INC/_PTALY0'
52: C      include 'INC/_PTALY'
53: C      include 'INC/_PTALY2'
54: C
55: C**** CROSS SECTION DATA IN CXP ARRAY
56: C
57: C      real CXP(MCXP), XLBP1(NPATOM,10), XLBP2(NPATOM,NMTP,4)
58: C      integer ICXP(MCXP), KLBP1(NPATOM,20), KLBP2(NPATOM,NMTP,4)
59: C
60: C**** CROSS SECTION DATA FOR BREMSSTRAHLUNG OF ELECTRON
61: C
62: C      real XLBE1(NPATOM,6)
63: C      real EEL(NEE), RKT(MTOP), PBT(NEE,NMAT), EBT(MTOP,NEE,NMAT),
64: C      &      EBT(MTOP-1,NEE,NMAT), WWAG(NMAT), EEDG(NMAT), EEEK(NMAT)
65: C      integer IEBTP(2*(MTOP-1),NEE,NMAT)

```

```

66: C
67: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
68: C
69: C      integer IZZ(NBANK,2)
70: C**** GEOMETRY
71: C      integer KZMAT(NZONE,2)
72: C**** cross section
73: C      real SMIC(NBANK,NUC,NSMIC)
74: C
75: C**** working array
76: C
77: C      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK)
78: C      real COSL(NBANK)
79: C      real WK1(NBANK), WK2(NBANK), R(NBANK*4)
80: C
81: C**** Prepared stack and bank
82: C
83: C      integer IBP(NBANK), ISEL(NBANK), ICLA(NBANK), ISTB(NBANK),
84: C      &      NTBL(NBANK)
85: C      integer NRTYP(10)
86: C      real W(NBANK), EIN(NBANK), WW(NBANK), EEEJ(NBANK),
87: C      &      WE(NBANK), EELE(NBANK), EBRM(NBANK), WWE(NBANK)
88: C      real*8 A0(NBANK), B0(NBANK), C0(NBANK),
89: C      &      AA(NBANK), BB(NBANK), CC(NBANK),
90: C      &      AAE(NBANK), BBE(NBANK), CCE(NBANK)
91: C
92: C      parameter( PI = 3.1415926535897932D+00 )
93: C      parameter( PI2 = 2.0D0*PI )
94: C      parameter( PIH = 0.5D0*PI )
95: C      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
96: C
97: C      parameter( ONE = 0.999999D0 )
98: C      ... ENERGY OF ELECTRON .....
99: C      parameter( DMOC2 = 0.511D6 )
100: C
101: C
102: C=====
103: C      if ( JVMNT.ne.0 ) call VMNTRL( 5, NCOLP )
104: C=====
105: C
106: C      NCOH = 0
107: C      NINC = 0
108: C      NPPR = 0
109: C      NPHE = 0
110: C      do 100 J = 1, NSEL
111: C      I = ISEL(J)
112: C      ... photon energy
113: C      EEEJ(J) = EIN(I)
114: C      if ( I.le.NRTYP(1) ) then
115: C      NCOH = NCOH + 1
116: C      else if ( I.le.NRTYP(2) ) then
117: C      NINC = NINC + 1
118: C      else if ( I.le.NRTYP(3) ) then
119: C      NPPR = NPPR + 1
120: C      end if
121: C 100 continue
122: C      NINC = NCOH + NINC
123: C      NPPR = NINC + NPPR
124: C      NPHE = NSEL
125: C
126: C      NEIN = NINC
127: C      if ( JBREM.ne.0.and.JGAMM.ne.0 ) then
128: C      do 110 J = 1, NINC
129: C      if ( ISEL(J).le.NRTYP(5) ) NEIN = J
130: C      if ( ISEL(J).le.NRTYP(6) ) NEPE = J

```

src/mvp/nxphr4.f

```

131: 110 continue
132: end if
133: C
134: C ..... CALCULATION OF SCATTERING ANGLE
135: C ASSUME COSL=0.0 FOR PARTICLES WITH DI(J)=0.0
136: C
137: if ( JBREM.eq.0 ) then
138: do 120 J = 1, NINC
139: I = ISEL(J)
140: COSL(J) = MIN(1.0D0, MAX(-1.0D0,
141: & A0(I)*AA(J)+B0(I)*BB(J)+C0(I)*CC(J) ))
142: 120 continue
143: else
144: do 130 J = 1, NPHE
145: I = ISEL(J)
146: COSL(J) = MIN(1.0D0, MAX(-1.0D0,
147: & A0(I)*AA(J)+B0(I)*BB(J)+C0(I)*CC(J) ))
148: 130 continue
149: end if
150: C
151: C=====
152: C photon reactions
153: C=====
154: C
155: if ( NINC.gt.0 ) then
156: XK = XK0*1.414213562D0
157: do 140 J = 1, NINC
158: I = ISEL(J)
159: IWK2(J) = ISTB(I)
160: IWK1(J) = NTBL(I)
161: ALFA = EEEJ(J) / DM0C2
162: if ( J.le.NCOH ) then
163: WK1(J) = XK*ALFA*SQRT(1.-COSL(J))
164: else
165: WK2(J) = 1./(1.+ALFA*(1.-COSL(J)))
166: WK1(J) = XK0*ALFA*SQRT(1.+WK2(J)*(WK2(J)-2.*COSL(J)))
167: end if
168: 140 continue
169: C
170: call BSINC3( CXP, IWK1, IWK4, IWK2, WK1, IWK3, NINC )
171: C
172: if ( NCOH.gt.0 ) then
173: do 150 J = 1, NCOH
174: LY = IWK2(J) + IWK3(J)
175: L1 = IWK2(J) + IWK3(J) + IWK1(J)*3
176: TI = (CXP(LY)-WK1(J)) / (CXP(LY)-CXP(LY-1))
177: if ( TI.lt.0. ) then
178: WW(J) = 0.0
179: else
180: I = ISEL(J)
181: CFF = CXP(L1) + (CXP(L1-1)-CXP(L1))*TI
182: WW(J) = WW(J)* 2.*XPIR2 *(1.+COSL(J)**2) *CFF**2
183: & /SMIC(IBP(I),ICLA(I),2)
184: end if
185: 150 continue
186: endif
187: C
188: if ( NINC.gt.NCOH ) then
189: do 160 J = NCOH+1, NINC
190: EEEJ(J) = EEEJ(J) * WK2(J)
191: LY = IWK2(J) + IWK3(J)
192: L1 = IWK2(J) + IWK3(J) + IWK1(J)
193: TI = (CXP(LY)-WK1(J)) / (CXP(LY)-CXP(LY-1))
194: I = ISEL(J)
195: if ( TI.ge.0. ) then

```

```

196: CSF = CXP(L1) + (CXP(L1-1)-CXP(L1))*TI
197: WW(J) = WW(J)* CSF
198: else
199: CSF0 = XLBP1(ICLA(I),2)
200: WW(J) = WW(J)* CSF0
201: end if
202: WW(J) = WW(J)* 2.*XPIR2 *WK2(J)**2
203: & *(WK2(J)+1./WK2(J)+COSL(J)**2-1.)
204: & /SMIC(IBP(I),ICLA(I),3)
205: 160 continue
206: endif
207: C
208: end if
209: C
210: C=====
211: C bremsstrahlung photon
212: C=====
213: C
214: if ( JBREM.ne.0 ) then
215: C
216: if ( NEIN.gt.NCOH ) then
217: JSTK = 1
218: ITYP = 4
219: NELC = NEIN - NCOH
220: C
221: call ELGEN2 ( IOW, ITYP, JSTK , IRAND,
222: S ISEL(NCOH+1), NELC , NBANK,
223: B A0 , B0 , C0 ,
224: B AAE , BBE , CCE ,
225: B EIN , EELE , WE ,
226: B COSL(NCOH+1), EEEJ(NCOH+1),
227: B AA(NCOH+1) , BB(NCOH+1) , CC(NCOH+1) ,
228: W R , WK1 , IWK2 , WK2 )
229: C
230: do 200 J = 1, NELC
231: WK1(J) = EELE( ISEL(NCOH+J) )
232: 200 continue
233: C
234: call BSVDEC( EEL, NEE, WK1, IWK1, NELC )
235: C
236: call RANU2( IRAND, R, NELC*4, ICON )
237: N2 = NELC*2
238: N3 = NELC*3
239: MTOP1 = MTOP - 1
240: C
241: *VOCL LOOP,NOVREC
242: do 210 JJ = 1, NELC
243: J = NCOH + JJ
244: I = ISEL(J)
245: R1 = (EEL(IWK1(JJ))-WK1(JJ)) /
246: & (EEL(IWK1(JJ))-EEL(IWK1(JJ)+1))
247: if ( R1.gt.1.0 ) R1 = 1.0
248: IMED = KZMAT(IZZ(IBP(I),1),1)
249: YIELD = (1.0-R1)*PBT(IWK1(JJ),IMED)
250: & + R1*PBT(IWK1(JJ)+1,IMED)
251: C
252: RR = R1 + R(JJ)
253: M = IWK1(JJ) + RR
254: LS = MTOP1*R(NELC+JJ) + 1
255: C/#IF ROUNDOFF(NEAREST)
256: M = MIN(M,IWK1(JJ)+1)
257: LS = MIN(LS,MTOP1)
258: C/#ENDIF
259: LS2 = LS*2
260: LS2 = min(LS2,int(LS2 + R(N2+JJ) - EBTP(LS,M,IMED)))

```

src/mvp/nxphr4.f

```

261:          LS3      = IEFTP(LS2,M,IMED)
262: C
263:          EBTX      : PROBABILITY FOR K/T RATIO SEARCH.
264: C          RKT1     : K/T RATIO.
265: C
266:          EBT1      = EBT(LS3,M,IMED)
267:          EBT2      = EBT(LS3+1,M,IMED)
268:          EBTX      = EBT2 + (EBT1-EBT2)*R(N3+JJ)
269:          EBTA      = (EBT1+EBT2) /2.0
270:          if ( LS3.eq.1 .or. EBTX.gt.EBTA ) then
271:              L      = LS3
272:              EBT3    = EBT(L+2,M,IMED)
273:          else
274:              L      = LS3 - 1
275:              EBT3    = EBT2
276:              EBT2    = EBT1
277:              EBT1    = EBT(L,M,IMED)
278:          end if
279:          RKT1      = (EBTX-EBT2)*(EBTX-EBT3)*RKT(L) /
280:          &          ((EBT1-EBT2)*(EBT1
281:          &          -EBT3)) + (EBTX-EBT1)*(EBTX-EBT3)*RKT(L+1) /
282:          &          ((EBT1-EBT2)*(EBT3
283:          &          -EBT2)) + (EBTX-EBT1)*(EBTX-EBT2)*RKT(L+2) /
284:          &          ((EBT3-EBT1)*(EBT3-EBT2))
285:          EBRM(I) = EELE(L)*RKT1
286: C
287: C.... energy cut off
288:          if ( EBRM(I).lt.EBOTP ) then
289:              WWE(J) = 0.0
290:          else
291:              WWE(J) = WWE(J) * YIELD
292: C ... change electron energy
293:              EELE(I) = SQRT(EELE(I)*(EELE(I)+2.*DMOC2))
294:          &          / (EELE(I)+DMOC2)
295:          end if
296: C
297: 210      continue
298: C
299:          end if
300: C
301:          NBRM = NPHE - NCOH
302:          call RANU2( IRAND, R(NCOH+1), NBRM, ICON )
303: C
304: *VOCL LOOP,NOVREC
305:          do 220 J = NCOH+1, NPHE
306:              I      = ISEL(J)
307: C
308:              CS     = AA(J)*AAE(I)+BB(J)*BBE(I)+CC(J)*CCE(I)
309:              WWEJ    = WWE(J) *
310:          &          (1.-EELE(I)**2)/(EELE(I)*CS-1.)**2
311: C
312:              WW(J)  = WW(J) + WWEJ
313:              if ( WW(J)*R(J).lt.WWEJ ) then
314:                  EEEJ(J) = EBRM(I)
315:              end if
316: 220      continue
317: C
318:          if ( JGAMM.eq.1 ) then
319:              LSIN = 1
320:              NIN  = NEIN
321:              LSPE = NEIN + 1
322:              NPE  = NEPE - NEIN
323:              LSPP = NEPE + 1
324:              NPP  = NPPR - NEPE
325:          else

```

```

326: C          LSIN = NCOH + 1
327: C          NIN  = NINC - NCOH
328: C          LSPP = NINC + 1
329: C          NPP  = NPPR - NINC
330: C          LSPE = NPPR + 1
331: C          NPE  = NPHE - NPPR
332: C          end if
333: C
334: C          ITYP = 1/2/3/4
335: C          = photo electron/auger electron/ pair creation/Compton recoil
336: C
337: C          WK1 : probability of electron in detector direction ( not Compton)
338: C          WK2 : scattered angle cosine of photon ( Compton )
339: C
340: C          if ( NPE.gt.0 ) then
341: C              ITYP = 1
342: C              call ELGEN3 ( IOW, ITYP, JSTK , IRAND,
343: C                  S      ISEL(LSPE) , NELC , NBANK,
344: C                  B      A0 , B0 , C0 ,
345: C                  B      AAE , BBE , CCE ,
346: C                  B      EIN , EELE , WE ,
347: C                  B      COSL(LSPE) , EOUT(LSPE) ,
348: C                  B      AA(LSPE) , BB(LSPE) , CC(LSPE) ,
349: C                  W      R , WK1 , IWK2 , WK2 )
350: C          end if
351: C
352: C          if ( NPP.gt.0 ) then
353: C              ITYP = 3
354: C          end if
355: C
356: C          if ( NIN.gt.0 ) then
357: C              ITYP = 4
358: C          end if
359: C
360: C          end if
361: C
362: C          return
363: C          end

```

src/mvp/nxphr.f

```

1:      subroutine NXPHR( IOW, A, H, LSCLP,
2:      B      XXX , YYY , ZZZ , AAA , BBB , CCC ,
3:      B      EEE , WWW , IZZ , IGG , TTT , ITT ,
4:      B      LEVL , LZZ , LPOS , LCRS ,
5:      B      DBNK , KDBNK , MDBNK , IBNK , KIBNK , MIBNK ,
6:      B      NDBNK , NIBNK , NCNTR , WCNTR ,
7:      B      LZZI , LPOSI , LCRSI ,
8:      B      OPTI , PATH , KDETP , KLSFI , SMACI , MMACI ,
9:      G      KZMAT , KZREG , TIMEB , XPDET , IPDET , IPDT2 ,
10:     S      ENGYB , JPUSD , IMSFL , IMNFL , IMZFL ,
11:     S      IMDED , IMSLT , IMZLT , IMNLT ,
12:     W      X , Y , Z , A0 , B0 , C0 ,
13:     W      AA , BB , CC , DI , TI , WW ,
14:     W      IBP , ICLA , ISTB , NTBL , NRTYP , EIN ,
15:     W      W , AAE , BBE , CCE , WE , EELE ,
16:     W      EBRM , EEE7 , WWE , IT0 , IGO , ISEL ,
17:     W      IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
18:     W      IWK7 , IWK8 , IWK9 , IWK10 , WK1 , WK2 ,
19:     W      WK3 , WK4 , WK5 , COSL , R )
20: C=====
21: C
22: C PURPOSE: Selection of photon collision point for next event
23: C estimator & call the estimation module
24: C CALLED IN: ACTION
25: C CALLS: VMNTR1,RANU2,LATUP2,LATDW2,NXPHR2,NXPHR3,NXPHR4,GMACNX
26: C+MVP_DETETE
27: C Updates:
28: C 14 Mar 2007: add the process of photonuclear. (K.Kosako)
29: C 9 Apr 2007: bug fix the time for periodic-time. (K.Kosako)
30: C-MVP_DETETE
31: C
32: C=====
33: C
34: C === INTER-STACK DATA FLOW ===
35: C
36: C COLLISION STACK -----> FREE-FLIGHT STACK FOR IMAGINARY PARICLE
37: C (LSCLP,NCOLP) (IMSFL,IMZFL,IMNFL) (IMSLT,IMZLT,IMNFL)
38: C (NOT REMOVED)
39: C
40: C=====
41: C implicit real*8 (A-H,O-Z)
42: C
43: C real A(*)
44: C real H(*)
45: C
46: C include '../shared/INC/_SIZES'
47: C include '../shared/INC/_PGEOM'
48: C include 'INC/_SIZES2'
49: C include 'INC/_FLAGS'
50: C include 'INC/_STACK'
51: C include 'INC/_KPIDS'
52: C include 'INC/_NGPS'
53: C include 'INC/_CXSEC'
54: C include 'INC/_PXSEC'
55: C include 'INC/_SBANK'
56: C include '../shared/INC/_PTALY0'
57: C include 'INC/_PTALY'
58: C include 'INC/_PTALY2'
59: C
60: C**** DETECTOR DATA FOR NEXT EVENT (POINT) ESTIMATOR
61: C
62: C real*8 XPDET(NPLEN,NPDET)
63: C integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
64: C integer JPUSD(NPDET)
65: C

```

```

66: C**** TALLY BIN DATA
67: C real TIMEB(*)
68: C real ENGYB(*)
69: C
70: C**** GEOMETRY ARRAY DATA
71: C
72: C integer KZMAT(NZONE,2), KZREG(NZONE)
73: C
74: C**** CROSS SECTION DATA
75: C
76: C**** SIGMA BANK
77: C
78: C real SMACI(IMPMAX,MB,NSMACI)
79: C integer MMACI(IMPMAX,2)
80: C
81: C
82: C**** STACK FOR IMAGINARY PARTICLE
83: C
84: C integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(NZONE+1),
85: C & IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
86: C
87: C**** STACK FOR REAL PARTICLE
88: C
89: C integer LSCLP(NBANK)
90: C
91: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
92: C
93: C real*8 XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2), AAA(NBANK,2),
94: C & BBB(NBANK,2), CCC(NBANK,2)
95: C
96: C real*8 TTT(NBANK,2)
97: C real EEE(NBANK,2), WWW(NBANK,2)
98: C
99: C integer IZZ(NBANK,2), IGG(NBANK,2), ITT(NBANK,2), LEVL(NBANK,2),
100: C & LZZ(NBANK,NEST), LPOS(NBANK,NEST), LCRS(NBANK,NEST),
101: C & LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
102: C
103: C real*8 DBNK(NBANK,NDBNK,2)
104: C integer KDBNK(0:MDBNK)
105: C integer IBNK(NBANK,NIBNK,2)
106: C integer KIBNK(0:MIBNK)
107: C****
108: C real*8 OPTI(IMPMAX), PATH(IMPMAX)
109: C integer KDETP(IMPMAX), KLSFI(IMPMAX)
110: C
111: C**** TALLY (MONITOR) ARRAY
112: C
113: C real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
114: C
115: C**** WORKING AREA (THESE AREA CAN BE DESTROYED IN OTHER ROUTINES
116: C (IWK1,IWK2),(IWK3,IWK4),(IWK5,IWK6),..(WK1,WK2) ..(WK5,COSL)
117: C (R ) are available as real*8 work area
118: C
119: C integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
120: C & IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
121: C & IWK9(NBANK), IWK10(NBANK)
122: C real WK1(NBANK), WK2(NBANK)
123: C real WK3(NBANK), WK4(NBANK), WK5(NBANK), COSL(NBANK), R(4*NBANK)
124: C real SMCW(NBANK,NSMICI)
125: C
126: C**** WORK AREA ( CANNOT BE DESTROYED IN SUB-PROCESS (NXTEE) )
127: C
128: C integer IBP(NBANK), ICLA(NBANK), ISTB(NBANK), NTBL(NBANK),
129: C & IT0(NBANK), IGO(NBANK), ISEL(NBANK)
130: C integer NRTYP(10)

```


src/mvp/nxphr.f

```

131:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
132:      real*8 A0(NBANK), B0(NBANK), C0(NBANK),
133:      & AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
134:      real*8 AAE(NBANK), BBE(NBANK), CCE(NBANK)
135:      real W(NBANK), EIN(NBANK), WW(NBANK), EEEJ(NBANK),
136:      & WE(NBANK), EELE(NBANK), EBRM(NBANK), WWE(NBANK)
137:      real*8 TI(NBANK)
138: C
139: C 0..NBK.2NBK.3NBK.4NBK.5NBK.6NBK.7NBK.8NBK.9NBK,10NBK,11NBK
140: C****
141: C
142: C
143: C parameter( PI = 3.1415926535897932D+00 )
144: C parameter( PI2 = 2.0D0*PI )
145: C parameter( PIH = 0.5D0*PI )
146: C parameter( PI4I = 1.0D0/(4.0D0*PI ) )
147: C
148: C .... constants ....
149: C
150: C --- light speed --- (cm/s)
151: C
152: C real*8 CLIGHT
153: C parameter( CLIGHT = 2.99792458D10 )
154: C
155: C --- neutron mass --- (gram)
156: C
157: C real*8 NMASS
158: C parameter( NMASS = 1.6749286D-24 )
159: C
160: C --- erg / eV --- electron charge
161: C
162: C real*8 EECHRG
163: C parameter( EECHRG = 1.60217733D-12 )
164: C
165: C --- neutron's mc**2 in eV
166: C
167: C real*8 MC2
168: C parameter( MC2 = NMASS*CLIGHT**2/EECHRG )
169: C
170: C
171: C=====
172: C if ( JVMNT.ne.0 ) call VMNTR1( 12, NCOLP )
173: C
174: C=====
175: C selection of reaction type generating imaginary particles.
176: C=====
177: C
178: C call NXPHR2( IOW, A, H, LSCLP, NIGEN,
179: C B EEE , WWW ,
180: C B DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
181: C B NDBNK, NIBNK,
182: C X A(LCXP) , A(LCXP) , A(LKLB1), A(LXLB1), A(LKLB2), A(LXLB2),
183: C X A(LXLB1),A(LEEL) , A(LPBT) , A(LRKT) , A(LEBT) , A(LEBTP) ,
184: C X A(LIEBTP),A(LWWAG), A(LEEDG) , A(LEEEK) , XK0 , XPIR2 ,
185: C X H(LKMAC), H(LMMAC), H(LSMIC) , A(LLMIC) , H(LKSPI) ,
186: C B IBP , ICLA , ISTB , NTBL , NRTYP,
187: C B EIN , W , WE ,
188: C W IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
189: C W WW , R )
190: C
191: C IBP(I) : bank pointer
192: C ICLA(I) : collision atom
193: C EIN(I) : incident energy
194: C W(I) : photon weights
195: C WE(I) : electron weights

```

```

196: C
197: C ISTB(I) : pointer to form factor table
198: C NTBL(I) : table length
199: C
200: C=====
201: C
202: C .... CUTOFF ENERGY OF ELECTRON ....
203: C set in INTRO2
204: C EBOTE = 1.0E+3
205: C .... EFFECTIVE ONLY IF ELECTRON TRANSPORT IS NOT CARRIED OUT.
206: C if ( EBOTE.lt.EBOTP ) EBOTE = EBOTP
207: C
208: C XK0 = XLBP1(1,4)
209: C XPIR2 = XLBP1(1,6)
210: C
211: C .... AUGER ELECTRONS ARE TAKEN INTO ACCOUNT WHEN JAUGE = 1
212: C
213: C JAUGE = 0
214: C
215: C=====
216: C gather spatial data of particles generating imaginary particles.
217: C=====
218: C
219: C if ( JLATT.eq.0 ) then
220: C ..... NO LATTICE
221: C do 100 I = 1, NIGEN
222: C X(I) = XXX(IBP(I),1)
223: C Y(I) = YYY(IBP(I),1)
224: C Z(I) = ZZZ(IBP(I),1)
225: C A0(I) = AAA(IBP(I),1)
226: C B0(I) = BBB(IBP(I),1)
227: C C0(I) = CCC(IBP(I),1)
228: C 100 continue
229: C
230: C else
231: C
232: C ..... LATTICE
233: C ***** TRANSFORMATION OF CORDINATES AND DIRECTION
234: C FROM CELL-ORDINATE SYSTEM TO LABORATORY SYSTEM
235: C (XXX,YYY,ZZZ)&(AAA,BBB,CCC) --> (X,Y,Z)&(A0,B0,C0)
236: C
237: C JDIR = 1
238: C JLS = 0
239: C call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
240: C & NZONE, NIGEN, NBANK, JDIR, JLS, IBP,
241: C & X, Y, Z, A0, B0, C0,
242: C & XXX, YYY, ZZZ,
243: C & AAA, BBB, CCC,
244: C & LEVL, LZZ, LPOS, LCRS,
245: C & DBNK, KDBNK, MDBNK,
246: C & A(LKZMAT), A(LKDALT), A(LDALT), A(LNVLAT), A(LSZLAT),
247: C & A(LCELSZ), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LLTYP),
248: C & A(LICTYP), A(LMLBZZ),
249: C & IWK2 )
250: C
251: C end if
252: C
253: C=====
254: C set parameters for generating imaginary particles.
255: C=====
256: C
257: C call NXPHR3( IOW, A, H, JAUGE,
258: C X A(LCXP) , A(LCXP) , A(LKLB1), A(LXLB1), A(LKLB2), A(LXLB2),
259: C X A(LXLB1),A(LEEL) , A(LPBT) , A(LRKT) , A(LEBT) , A(LEBTP) ,
260: C X A(LIEBTP),A(LWWAG), A(LEEDG) , A(LEEEK),

```

src/mvp/nxphr.f

```

261:      D  KZMAT, IZZ ,
262:      B  NIGEN, IBP , ICLA , ISTB , NTBL , NRTYP,
263:      B  EIN , W , WE , EELE , EBRM ,
264:      B  A0 , B0 , C0 , AAE , BBE , CCE ,
265:      W  IWK1 , IWK2 , WK1 , WK2 , R )
266: C
267: C      IBP(I) : bank pointer
268: C      ICLA(I) : collision atom
269: C      EIN(I) : incident photon energy (coherent, incoherent)
270: C      secondary photon energy (photo-e, pair creation)
271: C      EELE(I) : electron energy
272: C      EBRM(I) : bremsstrahlung photon energy
273: C      W(I) : imaginary photon weights including multiplicity
274: C      and 1/4pai
275: C      WE(I) : electron weights including multiplicity
276: C      and 1/4pai
277: C
278: C      ISTB(I) : pointer to form factor table
279: C      NTBL(I) : table length
280: C-----
281: C
282: C
283: C
284: C-----<< DO - LOOP INDEX DEFINITION HEREAFTER >> -----
285: C
286: C
287: C
288: C I : INDICATE PARTICLES IN COLLISION STACK( IBP ).
289: C
290: C << VARIABLES REFERENCED BY INDEX 'I' >>
291: C
292: C      IBP(I) : bank pointer
293: C      ICLA(I) : collision atom
294: C
295: C      NRTYP(10) : cummulative number of particles
296: C      1 : coherent scattering
297: C      2 : incoherent scattering
298: C      3 : pair creation
299: C      4 : photo electric
300: C      5 : electron by incoherent scattering in JGAMM=1
301: C      (NRTYP(2) >= NRTYP(5))
302: C      6 : electron by photo electric in JGAMM=1
303: C
304: C      X(I),Y(I),Z(I) : collision points
305: C      EIN(I) : incident photon energy
306: C      EELE(I) : electron energy
307: C      EBRM(I) : bremsstrahlung photon energy
308: C      A0(I),B0(I),C0(I) : flight direction of incident photon
309: C      AAE(I),BBE(I),CCE(I) : flight direction of incident photon
310: C      W(I) : photon weights including multiplicity + 1/4pai
311: C      WE(I) : electron weights including multiplicity + 1/4pai
312: C
313: C
314: C J : INDICATE PARTICLES TO EMIT AN IMAGINARY PARTICLE
315: C TOWARD ND'TH DETECTOR.
316: C (SELECTED BY RUSSIAN ROULETTE AT THE BEGINNIG OF
317: C DETECTOR LOOP)
318: C
319: C < VARIABLES REFERENCED BY INDEX 'J' >
320: C
321: C      ISEL(J) : indicator of selected particles in collision stack.
322: C      ( variables native to I-index are referred as
323: C      A0(ISEL(J)) etc. )
324: C      AA(J),BB(J),CC(J) : flight vectors of emitted imaginary
325: C      particles.

```

```

326: C      WW(J) : imaginary photon weight
327: C      WWE(J) : imaginary photon weight by electron
328: C      COSL(J)
329: C
330: C
331: C**** LOOP OVER DETECTOR (NPDET) *****
332: C
333: C
334: C
335: C
336: C      do 200 ND = 1, NPDET
337: C
338: C      if ( JPUSD(ND).eq.0 ) go to 200
339: C
340: C=====
341: C
342: C      CALCULATE DISTANCES TO ND'TH DETECTOR
343: C      & SELECT COLLISIONS COUNTED AT THE DETECTOR
344: C
345: C=====
346: C
347: C
348: C
349: C
350: C      XDA = XPDET(3,ND)
351: C      YDA = XPDET(4,ND)
352: C      ZDA = XPDET(5,ND)
353: C      RMAX = XPDET(2,ND)
354: C      RMA2 = RMAX*RMAX
355: C      RMIN = XPDET(1,ND)
356: C      RMIN2 = RMIN*RMIN
357: C
358: C      call RANU2( IRAND, R, NIGEN, ICON )
359: C
360: C      NN : NUMBER OF SELECTED COLLISION POINTS
361: C      NBS : NUMBER OF SELECTED COLLISION POINTS WHICH REQUIRE BOUNDED SPHERE
362: C
363: C      NN = 0
364: C      NBS = 0
365: C
366: C      if ( JBREM.eq.0 ) then
367: C      do 210 I = 1, NIGEN
368: C      AT = XDA - X(I)
369: C      BT = YDA - Y(I)
370: C      CT = ZDA - Z(I)
371: C      D = AT*AT + BT*BT + CT*CT
372: C
373: C      ..... CALCULATE CONTRIBUTION WITH PROBABILITY RMIN2/D
374: C      WHEN D > RMIN2.
375: C
376: C      if ( D.le.RMIN2 .or. D*R(I).le.RMIN2 ) then
377: C      NN = NN + 1
378: C      ISEL(NN) = I
379: C      DI(NN) = D
380: C      AA(NN) = AT
381: C      BB(NN) = BT
382: C      CC(NN) = CT
383: C      WW(NN) = W(I)
384: C      c##<2007/03/14:PN4:
385: C      IWK10(NN) = IBP(I)
386: C      c##>
387: C      end if
388: C      210 continue
389: C
390: C      do 220 J = 1, NN

```

src/mvp/nxphr.f

```

391: C
392: C      .... THE BOUNDED SPHERE APPROXIMATION IF DI < PMAX**2 ...
393: C
394: C      if ( DI(J).lt.RMA2 ) then
395: C          NBS      = NBS + 1
396: C
397: C
398: C      .... DEVIDE WEIGHT BY (DISTANCE)**2 HERE ....
399: C      .... WEIGHT CORRECTION NECESSARY IF DI > RMIN2 ...
400: C
401: C          MEANING : WW(J) = WW(J) / DI(J) *1/(RMIN2/DI(J))
402: C
403: C
404: C      else if ( DI(J).gt.RMIN2 ) then
405: C          WW(J)      = WW(J) /RMIN2
406: C      else
407: C          WW(J)      = WW(J) /DI(J)
408: C      end if
409: C
410: C      .... 'DI' MEANS DISTANCE ITSELF HEREAFTER. ....
411: C
412: C
413: C          DI(J)      = SQRT(DI(J))
414: C
415: C      .... (AA,BB,CC) IS NORMALIZED DIRECTION VECTOR HEREAFTER ...
416: C
417: C
418: C      if ( DI(J).gt.0. ) then
419: C          AA(J)      = AA(J) /DI(J)
420: C          BB(J)      = BB(J) /DI(J)
421: C          CC(J)      = CC(J) /DI(J)
422: C      else
423: C          AA(J)      = 0.
424: C          BB(J)      = 0.
425: C          CC(J)      = 0.
426: C      end if
427: C      220      continue
428: C
429: C      else
430: C          do 230 I = 1, NIGEN
431: C              AT      = XDA - X(I)
432: C              BT      = YDA - Y(I)
433: C              CT      = ZDA - Z(I)
434: C              D        = AT*AT + BT*BT + CT*CT
435: C
436: C          ..... CALCULATE CONTRIBUTION WITH PROBABILITY  RMIN2/D
437: C          WHEN D > RMIN2.
438: C
439: C              if ( D.le.RMIN2 .or. D*R(I).le.RMIN2 ) then
440: C                  NN      = NN + 1
441: C                  ISEL(NN) = I
442: C                  DI(NN)  = D
443: C                  AA(NN)  = AT
444: C                  BB(NN)  = BT
445: C                  CC(NN)  = CT
446: C                  WW(NN)  = W(I)
447: C                  WWE(NN) = WE(I)
448: C          c##<2007/03/14:PN4:
449: C              IWK10(NN) = IBP(I)
450: C          c##>
451: C              end if
452: C          230      continue
453: C
454: C          do 240 J = 1, NN
455: C

```

```

456: C      .... THE BOUNDED SPHERE APPROXIMATION IF DI < PMAX**2 ...
457: C
458: C          if ( DI(J).lt.RMA2 ) then
459: C              NBS      = NBS + 1
460: C
461: C
462: C          .... DEVIDE WEIGHT BY (DISTANCE)**2 HERE ....
463: C          .... WEIGHT CORRECTION NECESSARY IF DI > RMIN2 ...
464: C
465: C              MEANING : WW(J) = WW(J) / DI(J) *1/(RMIN2/DI(J))
466: C
467: C
468: C          else if ( DI(J).gt.RMIN2 ) then
469: C              WW(J)      = WW(J) /RMIN2
470: C              WWE(J)      = WWE(J) /RMIN2
471: C          else
472: C              WW(J)      = WW(J) /DI(J)
473: C              WWE(J)      = WWE(J) /DI(J)
474: C          end if
475: C
476: C      .... 'DI' MEANS DISTANCE ITSELF HEREAFTER. ....
477: C
478: C
479: C          DI(J)      = SQRT(DI(J))
480: C
481: C      .... (AA,BB,CC) IS NORMALIZED DIRECTION VECTOR HEREAFTER ...
482: C
483: C
484: C      if ( DI(J).gt.0. ) then
485: C          AA(J)      = AA(J) /DI(J)
486: C          BB(J)      = BB(J) /DI(J)
487: C          CC(J)      = CC(J) /DI(J)
488: C      else
489: C          AA(J)      = 0.
490: C          BB(J)      = 0.
491: C          CC(J)      = 0.
492: C      end if
493: C      240      continue
494: C      end if
495: C
496: C=====
497: C      .... calculate energy and scattering probability of imaginary particle
498: C=====
499: C
500: C      call NXPHR4(      IOW, A, H, XK0, XPIR2
501: C          X      A(LCXP) , A(LCXP) , A(LKLBPl), A(LXLBPl), A(LKLBPl2), A(LXLBPl2),
502: C          X      A(LXLBE1),A(LEEL) , A(LPBT) , A(LRKT) , A(LEBT) , A(LEBTP) ,
503: C          X      A(LIEBTP),A(LWWAG), A(LEEDG) , A(LEEEK),
504: C          D      KZMAT, IZZ , H(LSMIC),
505: C          B      IBP , ICLA , ISTB , NTBL , NRTYP,
506: C          B      EIN , W , WE , EELE , EBRM ,
507: C          B      A0 , B0 , C0 , AAE , BBE , CCE ,
508: C          B      ISEL , NN , AA , BB , CC , COSL ,
509: C          B      WW , EEEJ , WWE ,
510: C          W      IWK1 , IWK2 , IWK3 , IWK4 , WK1 , WK2 ,
511: C          W      R )
512: C
513: C      .... remove no-probability particle and energy cut-off .....
514: C
515: C          NNN      = 0
516: C
517: C      *VOCL LOOP,NOVREC
518: C          do 250 J = 1, NN
519: C              if ( WW(J).gt.0.0 .and.EEEJ(J).ge.EBOTP ) then
520: C                  NNN      = NNN + 1

```

src/mvp/nxphr.f

```

521:         ISEL(NNN) = ISEL(J)
522:         IWK1(NNN) = J
523:         EEEJ(NNN) = EEEJ(J)
524:         DI(NNN) = DI(J)
525:     end if
526: 250     continue
527: C
528:     NN = NNN
529: C
530:     do 260 J = 1, NN
531:         EEEJ(J) = MIN( EEEJ(J),ETOPP )
532: 260     continue
533: C
534: C .... TIME CUT OFF .....
535: C
536:     if ( NTIME.ne.0 ) then
537:         if ( IPTIM.eq.0 ) then
538:             NNN = 0
539: *VOCL LOOP,NOVREC
540:             do 270 J = 1, NN
541:                 TTTT = TTT(IBP(ISEL(J)),1) + DI(J) /CLIGHT
542:                 if ( TTTT.le.TCUT ) then
543:                     NNN = NNN + 1
544:                     ISEL(NNN) = ISEL(J)
545:                     IWK1(NNN) = IWK1(J)
546:                     TI(NNN) = TTTT
547:                     EEEJ(NNN) = EEEJ(J)
548:                     DI(NNN) = DI(J)
549:                 end if
550: 270             continue
551: C
552:             NN = NNN
553:             else
554:                 do 280 J = 1, NN
555:                     TTTT = TTT(IBP(ISEL(J)),1) + DI(J) /CLIGHT
556:                     if ( TTTT.ge.TCUT ) then
557:                         ITREP = TTTT / TCUT
558:                         TI(J) = TTTT - TCUT*ITREP
559: c##<2007/04/09: bug fix
560:                         else
561:                             TI(J) = TTTT
562: c##>
563:                             end if
564: 280                         continue
565:                     end if
566: C
567:                     call BS0ISD( TIMEB, NTIME+1, TI, IT0, NN )
568: C
569:                     do 290 J = 1, NN
570:                         IT0(J) = MAX(1,MIN(NTIME,IT0(J)))
571: 290                     continue
572: C
573:                     end if
574: C
575: C .... condense imaginary particle data ....
576: C
577: *VOCL LOOP,NOVREC
578:     do 300 J = 1, NN
579:         WW(J) = WW(IWK1(J))
580:         AA(J) = AA(IWK1(J))
581:         BB(J) = BB(IWK1(J))
582:         CC(J) = CC(IWK1(J))
583: c##<2007/03/14:PN4:
584:         IWK10(J) = IWK10(IWK1(J))
585: c##>

```

```

586: 300     continue
587: C
588: C .... energy group
589: C
590:     call BSVDEC( ENGYB(KENGP(KPPHOT)), NGP(KPPHOT)+1,
591:         &         EEEJ, IGO, NN )
592:     NGG = KNGP(KPPHOT) - 1
593:     do 310 J = 1, NN
594:         IGO(J) = IGO(J) + NGG
595: 310     continue
596: C
597: C
598: C=====
599: C
600: C     SEND PARTICLES TO STACK FOR IMAGINARY PARTICLES
601: C
602: C=====
603: C
604: C
605:         IZD = IPDET(ND,1)
606:         LVL = IPDET(ND,2)
607:         if ( JLATT.ne.0.and.LVL.gt.0 ) then
608:             if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
609:             LP = 2 + 3*LVL
610:             XDA = XPDET(LP+1,ND)
611:             YDA = XPDET(LP+2,ND)
612:             ZDA = XPDET(LP+3,ND)
613:         end if
614: C
615: 320     continue
616: C
617:         if ( NN.gt.0 ) then
618: C
619:             NNN = MIN(NN,NDIMPT)
620: C
621:             if ( NNN.gt.0 ) then
622:                 NDIMPT = NDIMPT - NNN
623:                 N = IMNFL(NZONE+1)
624:                 N1 = NN - NNN
625: *VOCL LOOP,NOVREC
626:                 do 330 J = 1, NNN
627:                     J1 = N1 + J
628:                     IMSFL(N+J) = IMDED(NDIMPT+J)
629:                     IMZFL(N+J) = IZD
630: C
631: C ---- BEWARE THAT IMAGINARY PARTICLES FLY FROM THE DETECTOR POINT
632: C     TO COLLISION POINTS. ----
633: C
634:                     XXX(IMSFL(N+J),2) = XDA
635:                     YYY(IMSFL(N+J),2) = YDA
636:                     ZZZ(IMSFL(N+J),2) = ZDA
637:                     AAA(IMSFL(N+J),2) = -AA(J1)
638:                     BBB(IMSFL(N+J),2) = -BB(J1)
639:                     CCC(IMSFL(N+J),2) = -CC(J1)
640:                     WWW(IMSFL(N+J),2) = WW(J1)
641:                     EEE(IMSFL(N+J),2) = EEEJ(J1)
642:                     IGG(IMSFL(N+J),2) = IGO(J1)
643:                     TTT(IMSFL(N+J),2) = TI(J1)
644:                     PATH(IMSFL(N+J)) = DI(J1)
645:                     OPTI(IMSFL(N+J)) = 0.0
646:                     KDETP(IMSFL(N+J)) = ND
647:                     KLSFI(IMSFL(N+J)) = 0
648:                     if ( JTIME.ne.0 ) then
649:                         ITT(IMSFL(N+J),2) = IT0(J1)
650:                     end if

```

src/mvp/nxphr.f

```

651:         if ( KIBNK(5).ne.0 ) then
652:             IBNK(IMSFL(N+J),KIBNK(5),2) = -1
653:         end if
654: c##<2007/03/14:PN4:
655:         if ( KIBNK(17).ne.0 ) IBNK(IMSFL(N+J),KIBNK(17),2) =
656:             1 IBNK(IWK10(J1),KIBNK(17),1)
657:         if ( KIBNK(18).ne.0 ) IBNK(IMSFL(N+J),KIBNK(18),2) =
658:             1 IBNK(IWK10(J1),KIBNK(18),1)
659:         if ( KIBNK(19).ne.0 ) IBNK(IMSFL(N+J),KIBNK(19),2) =
660:             1 IBNK(IWK10(J1),KIBNK(19),1)
661: c##>
662: 330 continue
663: C
664:     NN2 = NNN
665:     JSTG = 0
666: C
667: C=====
668: C
669: C In LATTICE geometry, the following parameters should be set.
670: C     LEVL, LZZI, LPOSI, LCRSI
671: C     DBNK, IBNK (JFISX.ne.0)
672: C     JSTG : flag to indicate the detector is located in STG region.
673: C     NN2 : number of remaining particles when JFISX.ne.0.
674: C
675: C=====
676: C
677:         if ( JLATT.ne.0 ) then
678:             call LATDW2(IOW, JDEBG, JHLAT, JFISX,
679:                 N IMPMAX,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
680:                 N DINF ,DEPS ,
681:                 S IMSFL(N+1),NNN ,JSTG ,NN2 ,
682:                 I IPDET(ND,2),IPDT2(1,1,ND),IPDT2(1,2,ND)
683:                 I IPDT2(1,3,ND),XPDET(3,ND),
684:                 B AAA(1,2),BBB(1,2),CCC(1,2) ,
685:                 B LEVL(1,2),LZZI ,LPOSI ,LCRSI ,
686:                 B DBNK(1,1,2),KDBNK ,MDBNK ,IBNK(1,1,2),KIBNK ,MIBNK,
687:                 S IMDED ,NDIMPT,
688:                 G A(LSDA) ,A(LKZDA) ,A(LKZAA) ,
689:                 G A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LKLBZZ),A(LCELSZ),
690:                 G A(LSZLAT),A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),
691:                 G A(LNVLAT),A(LIPCEL),A(LLTYP) ,A(LICTYP),A(LKDALT),
692:                 G A(LDALT) ,A(LKZREG),
693:                 W IWK1 ,IWK3 ,IWK5 ,IWK7 ,IWK9 ,WK1
694:                 W WK3 ,WK5 ,R ,WW(N1+1),IGO(N1+1) )
695:         end if
696: C
697:         if ( JSTG.eq.0 ) then
698:             IMNFL(IZD) = IMNFL(IZD) + NN2
699:             IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
700: C
701: C ... detector is located in STG region. move to LATTICE stack.
702:         else
703:             NL = IMNLT(NLBZ+1)
704:             do 340 J = 1, NN2
705:                 IMSLT(NL+J) = IMSFL(N+J)
706:                 IMZLT(NL+J) = IZD
707: 340 continue
708:                 IMNLT(IZD) = IMNLT(IZD) + NN2
709:                 IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
710:             end if
711: C
712:             NN = NN - NNN
713:             NIMPT = NIMPT + NN2
714:         end if
715: C

```

```

716: C *** prepare cross sections *****
717: C
718:         MAT = KZMAT(IZD,1)
719:         JAMXCT = 0
720:         JMIC = 1
721:         JNP = 2
722:         call GMACNX( IRAND, NN2, IMSFL(N+1), MAT,
723:             & JMIC, JAMXCT, JNP, JGAMM, JDEBG, IMPMAX,
724:             & EEE(1,2), MB, NUC, NSTAL,
725:             & NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
726:             & NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTLI),
727:             & A(LLPDNP), A(LIATMT), A(LDNSTP), NMAT, NPATOM, NMTP,
728:             & A(LXCP), MCXP, A(LKLBP1), A(LKLBP2), A(LXLBP1),
729:             & A(LCRES), A(LKCRSI), MCRES, NNCSCI, A(LINCSI),
730:             W IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
731:             W IWK7, IWK8, R, WK1 )
732: C
733: C
734: C .... BOUNDED SPHERE APPROXIMATION
735: C
736:         if ( NBS.gt.0 ) then
737:             if ( MAT.gt.0 ) then
738:                 *VOCL LOOP,NOVREC
739:                 do 350 J = 1, NN2
740:                     IP = IMSFL(N+J)
741:                     if ( PATH(IP).le.RMAX ) then
742:                         RMD = SMACI(IP,MMACI(IP,1),1) * RMAX
743:                         RMI = 1./RMD
744: C
745:                         if ( RMD.le.500. ) then
746:                             C2 = RMA2*
747:                             & (RMI*RMI*2.+(1.+2.0*RMI)/(1.-EXP(RMD)))
748:                         else
749:                             C2 = RMA2* RMI*RMI*2.
750:                         end if
751: C
752:                         WWW(IP,2) = WWW(IP,2) /C2
753:                     end if
754: 350 continue
755:                 else
756:                     *VOCL LOOP,NOVREC
757:                     do 360 J = 1, NN2
758:                         IP = IMSFL(N+J)
759:                         if ( PATH(IP).le.RMAX ) then
760:                             WWW(IP,2) = WWW(IP,2) /RMA2 *3.0
761:                         end if
762: 360 continue
763:                     end if
764:                 end if
765: C
766: C
767: C *** CALL TRACKING CONTROL ROUTINE FOR IMAGINARY PARTICLES
768: C
769:         if ( NDIMPT.eq.0 ) then
770:             call NXTEE( 2, A, H, NDIMPT, NIMPT )
771:         end if
772: C
773:         go to 320
774: C
775:         end if
776: C
777: C
778: 200 continue
779: C
780: C **** END OF DETECTOR LOOP *****

```

src/mvp/nxphr.f

```
781: C
782:   return
783: end
```



src/mvp/nxtee.f

```

1: *VOCL TOTAL,SCALAR
2:   SUBROUTINE NXTEE( MODE, A, H, NDIMP2, NIMPT2 )
3: C=====
4: C PURPOSE: TRACKING OF IMAGINARY PARTICLES FOR NEXT EVENT ESTIMATOR.
5: C CALLED IN: ACTION,SOURCE,NXTNR
6: C CALLS:
7: C=====
8: C OPERATION MODE :
9: C  MODE = 1 : PROCESS ALL IMAGINARY PARTICLES
10: C          UNTIL FINISHED.
11: C  MODE = 2 : PROCESS IMAGINARY PARTICLES UNTIL
12: C            ENTRY OF BANK IS LESS THAN 'NTHIM' PARTICLES.
13: C  MODE = N : PROCESS N EVENTS.
14: C=====
15: C
16:   real A(*)
17:   real H(*)
18:   include 'INC/_FLAGS'
19:   include '../shared/INC/_SIZES'
20:   include 'INC/_SIZES2'
21:   include 'INC/_XBANK'
22:   include 'INC/_STACK'
23:   include 'INC/_XWORK'
24:   include 'INC/_SBANK'
25:   include '../shared/INC/_PGEOM'
26:   include 'INC/_PKSEC'
27:   include 'INC/_KPIDS'
28:   include 'INC/_CXSEC'
29:   include '../shared/INC/_PVRED'
30:   include '../shared/INC/_PTALY0'
31:   include 'INC/_PTALY'
32:   include '../shared/INC/_IOUNIT'
33:   include 'INC/_IOUNIT2'
34:   include 'INC/_WKNXT'
35:   include '../shared/INC/_STALY'
36:   include '../shared/INC/_PTLSP'
37: c##<2007/03/14:PN4:
38:   include 'INC/_PTALY2'
39: c##>
40: C
41:   NIMPT  = NIMPT2
42:   NDIMP2 = NDIMP2
43: C
44: C .... CLEAR EVENT COUNTER ....
45: C
46:   IEVENT = 0
47: C
48: C .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
49: C
50:   2222 CALL NEESEL(MODE, IEVENT, MACT, MZONE, NIMPT,NDIMP2,
51:     & h(LIMNFL), h(LIMNNX), IMDEFR, h(LIMNLT) )
52: C
53: C.....
54: C
55: C  MACT
56: C  ** = 0 : GENERATE NEXT BATCH OF PARTICLES.
57: C  ** = 1 : FREE FLIGHT
58: C  ** = 2 : NEXT ZONE SEARCH
59: C  ** = 3 : COLLISION
60: C  ** = 5 : REFLECTION
61: C  ** = 6 : LATTICE-SEARCH
62: C  ** = 99 : END OF RUN
63: C
64: C MZONE : SELECTED ZONE NUMBER (IF ZONEWISE LOGIC)
65: C

```

```

66: C.....
67: C
68: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT
69: C    & TAKE TALLY IF NECESSARY .....
70: C
71: check
72: c   write(6,*) ' nxtee, mact=',mact,' nimpt,ndimpt=',nimpt,ndimpt
73: c   call chkwrh(h(liggi),h(lpath),h(limsfl),h(limsnx),h(limslt),
74: c   & h(limnfl),h(limnnx),h(limnlt),impmax,nzone,nlbz)
75:   IF(MACT.EQ.1) THEN
76:     call NXTFLI(IOW,A,H, MZONE, NIMPT, NDIMP2,h(lNLOST),DEPS,DINF,
77:     B h(LXXXI), h(LYYYI), h(LZZZI), h(LAAAI), h(LBBBI), h(LCCCI),
78:     B h(LWWWI), h(LLEEI), h(LIGGI), h(LTTTI), h(LITTI),
79:     B h(LLEVI),h(LZZZI), h(LLPOSI),h(LLCRSI),h(LKLSFI),
80:     B h(LPATH),h(LOPTI),h(LKDETP),h(LSMACI),h(LMMACI),
81:     B h(LDBNKI),KDBNK,MDBNK,h(LIBNKI),KIBNK,MIBNK,
82:     S h(LIMSFL),h(LIMNFL),h(LIMZFL),
83:     S h(LIMSNX),h(LIMNNX),h(LIMZNX),h(LIMDED) ,
84:     G A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LDALT),
85:     G A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),A(LIPCEL),A(LKSLAT),
86:     G A(LLTYP),A(LMLBZZ),A(LKZLBZ),A(LKCELL),A(LICTYP),A(LKLATT),
87:     G A(LCELSZ),A(LPNND),A(LKNND),A(LPPPF),A(LKPPF),
88:     T A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
89:     T h(LDTALY),A(LRESP),h(LFLPD),
90:     W h(PF(1)),h(PF(2)),h(PF(3)),h(PF(4)),h(PF(5)),h(PF(6)),
91:     W h(PF(7)),h(PF(8)),h(PF(9)),h(PF(10)),h(PF(11)),h(PF(12)),
92:     W h(PF(13)),h(PF(14)),h(PF(15)),h(PF(16)),h(PF(17)),h(PF(18)),
93:     W h(PF(19)),h(PF(20)),h(PF(21)),h(PF(22)),h(PF(23)),
94:   c##<2007/03/14:PN4:
95:   c## W h(PF(24)),h(PF(25)),h(PF(26)),h(PF(27)) )
96:   W h(PF(24)),h(PF(25)),h(PF(26)),h(PF(27)),
97:   & KPNPRD, A(LDNZON), H(LIZZI), NSTALY, A(LXPDET) )
98: c##>
99: C* W X ,Y ,Z ,AI ,BI ,CI ,
100: C* W W ,DI ,IGI ,IBP ,IFC ,R ,
101: C* W DLOC1 ,DLOC2 ,IKL ,IKL2 ,XUP ,YUP ,
102: C* W ZUP ,AUP ,BUP ,CUP ,
103: C* W ISTG ,ILUP ,ILST ,SMCW )
104: C
105: C .... FIND NEXT ZONE TO ENTER .....
106: C
107:   ELSE IF(MACT.EQ.2) THEN
108:     call NEESEA(IOW, JVMNT, JLATT, JSIMP, MZONE, NIMPT, NDIMP2,
109:     E h(lNLOST), DEPS, JMEM, h(LKMEMO), h(LMEMC), h(LMEMZ),
110:     B h(LXXXI),h(LYYYI),h(LZZZI),h(LAAAI),h(LBBBI),h(LCCCI),
111:     B h(LWWWI), h(LIZZI), h(LIGGI), h(LTTTI), h(LLEVI), h(LLZZI),
112:     B h(LLPOSI), h(LLCRSI), h(LKLSFI),
113:     S h(LIMSFL),h(LIMNFL),h(LIMZFL),h(LIMSNX), h(LIMNNX), h(LIMZNX),
114:     S h(LIMDED), IMDEFR ,
115:     * h(LIMSLT), h(LIMZLT), h(LIMNLT),
116:     G A(LSDA), A(LKZMAT), A(LKZDA), A(LKZAA),
117:     * A(LKCELL), A(LIPCEL), A(LMLBZZ),
118:     T h(LNCNTR),h(LWCNTR),
119:     W h(PS(1)),h(PS(2)),h(PS(3)),h(PS(4)),h(PS(5)),h(PS(6)),
120:     W h(PS(7)),h(PS(8)),h(PS(9)),h(PS(10)) )
121: C** W X ,Y ,Z ,W ,IBP ,IFI ,
122: C** W FXYZ ,ISRF ,IZI ,IFL )
123: C
124: C .... LATTICE SEARCH .....
125: C
126:   ELSE IF(MACT.EQ.6) THEN
127:     JTLLT1 = 0
128:     call LATICE(JDEBG,JVMNT,JTLLT1,JHLAT,JFISX,h(lNLOST),
129:     N NBANK,NZONE,NREG,NLATT,NCELL,NLBZ,NEST,NSUZON,NSPACE,NUNV,
130:     N DINF,DEPS,IRAND,NMKREG,

```

src/mvp/nxtee.f

```
131:      B H(LXXXI ), H(LYYYI ), H(LZZZI ), H(LAAAI ), H(LBBBI ), H(LCCCI ),
132:      B H(LIZZI ), H(LLEVLI), H(LLZZI ), H(LLPOSI), H(LIMDED), NDIMPT ,
133:      B H(LLCRSI), H(LKLSFI), H(LIBREG), H(LIBSPC),
134:      B H(LDBNKI),KDBNK,MDBNK,H(LIBNKI),KIBNK,MIBNK,
135:      S H(LIMSNX), H(LIMZNX), H(LIMNNX), H(LIMSLT), H(LIMZLT), H(LIMNLT),
136:      S H(LIMSFL), H(LIMNFL), H(LIMZFL),
137:      G A(LSDA), A(LKZDA), A(LKZAA),
138:      G A(LKZMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
139:      G A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
140:      G A(LLTYP), A(LICTYP), A(LKDALT), A(LDALT ), A(LKZREG),
141:      G A(LISUSP),A(LKSPSU), A(LKTCSP), NKTCSF, A(LKHLAT),A(LMKREG),
142:      G A(LPPFF), A(LKPPF), NPPFF,
143:      W H(PL( 1)), H(PL( 2)), H(PL( 3)), H(PL( 4)), H(PL( 5)), H(PL( 6)),
144:      W H(PL( 7)), H(PL( 8)), H(PL( 9)), H(PL(10)),
145:      W H(PL(11)), H(PL(12)), H(PL(13)), H(PL(14)),H(PL(15)), H(PL(16)))
146:      NIMPT = IMPMAX - NDIMPT
147:
148: C
149:      ELSE IF( MACT.EQ. 99.) THEN
150:          GOTO 9999
151:      ENDIF
152: C
153:      GOTO 2222
154: C
155: C ..... END OF TASK .....
156: C
157: 9999 NDIMP2 = NDIMPT
158:      NIMPT2 = NIMPT
159:      RETURN
160:      END
161: check
162:      subroutine chkwrtr(iggi,path,imsfl,imsnx,imslt,
163:      &      imnfl,imnnx,imnlt,impmax,nzone,nlbz)
164:      integer imsfl(impmax),imsnx(impmax),imslt(impmax)
165:      integer imnfl(nzone+1),imnnx(nzone+1),imnlt(nlbz+1)
166:      integer iggi(impmax)
167:      real*8 path(impmax)
168:      write(6,*) ' iggi:',(iggi(i),i=1,impmax)
169:      write(6,*) ' path:'
170:      write(6, '(1p5e12.5)') (path(i),i=1,impmax)
171:      write(6,*) ' imsfl:',(imsfl(i),i=1,imnfl(nzone+1))
172:      write(6,*) ' imsnx:',(imsnx(i),i=1,imnnx(nzone+1))
173:      write(6,*) ' imslt:',(imslt(i),i=1,imnlt(nlbz+1))
174:      return
175:      end
```


src/mvp/nxtfli.f

```

1:      subroutine NXTFLI( IOW, A,      H,
2:      &                MZONE, NIMPT, NDIMPT,NLOST, DEPS,  DINF,
3:      &                XXXI,  YYYYI, ZZZI,  AAAI,  BBBI,  CCCI,  WWWI,
4:      &                EEI,   IGGI,  TTTI,  ITTI,  LEVLI, LZZI,  LPOSI,
5:      &                LCRSI, KLSFI, PATH, OPTI, KDET, SMACI, MMACI,
6:      &                DBNK,  KDBNK, MDBNK, IBNK,  KIBNK, MIBNK,
7:      &                IMSFL, IMNFL, IMZFL, IMSNX, IMNNX, IMZNX, IMDED,
8:      &                SDA,   KZMAT, KZREG, KZDA,  KZAA,  DALT,  KDALT,
9:      &                NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP,  MLBZ,
10:     &                KZLBZ, KCELL, ICTYP, KLATT, CELSZ,
11:     &                PNND,  KNND,  PPPF,  KPPF,
12:     &                IDTAL, JTEVE, LTEVE, KTEVE, NTEVE,
13:     &                DTALY, RESP,  FLDP,
14:     &                X,     Y,     Z,     AI,   BI,   CI,   OPT,
15:     &                DI,   IGI,   IBP,  IFC,   R,   DLOC1,
16:     &                DLOC2, IKL,   IKL2, XUP,   YUP,  ZUP,   BUP,
17: c##<2007/03/14:PN4:
18: c##      &                BUP,   CUP,  SIGT,  ISTG,  ILUP,  ILST,  SMCW )
19: c##      &                BUP,   CUP,  SIGT,  ISTG,  ILUP,  ILST,  SMCW,
20: c##      &                KENPRD,DNZON, IZZI,  NSTALY,XPDET )
21: c##>
22: C      W      XYZ1, ABC1, XYZ2, ABC2,
23: C      W      DDI, IBP0, IBP1, IBP2, KPLT, JKSF )
24: C===== FLIGHT >=====
25: C PURPOSE:  FREE-FLIGHT OF ONE-ZONE LOGIC FOR NEXT EVENT ESTIMATOR.
26: C TAKE TALLY OF POINT DETECTOR.
27: C
28: C CALLED IN:  NXTEE
29: C CALLS:      NEEFLI,STALN9,GMACNX
30: C
31: C UPDATE:
32: C 15 Aug 2003: for LTYP=10 (more than 1 cells in STG-region)
33: C      add dummy arguments PPPF&KPPF and pass them to NEEFLI
34: C 14 Mar 2007: add the process of photonuclear. (K.Kosako)
35: C=====
36: C
37: C === INTER-STACK DATA FLOW ===
38: C
39: C FREE-FLIGHT STACK -----+-----> NEXT-ZONE-SEARCH STACK
40: C (IMSFL,IMZFL,IMNFL)      |      (IMSNX,IMZNX,IMNNX)
41: C (AFTER TALLY) +-----> DEAD PARTICLE STACK
42: C                      (IMDED,NDIMPT)
43: C
44: C === BANK DATA TO BE UPDATED ===
45: C
46: C (XXXI,YYYYI,ZZZI) : POSITION
47: C (AAAI,BBBI,CCCI) : DIRECTION (IF HEXAGONAL-LATTICE EXISTS)
48: C LEVLI             : LATTICE LEVEL (IF HEXAGONAL-LATTICE EXISTS)
49: C KLSFI             : SURFACE DATA POINTER (IF ON-SURFACE)
50: C=====
51: C      implicit real*8(D)
52: C      real A(*)
53: C      real H(*)
54: C      include 'INC/_FLAGS'
55: C      include '../shared/INC/_SIZES'
56: C      include 'INC/_SIZES2'
57: C      include 'INC/_KPIDS'
58: C      include 'INC/_NGPS'
59: C      include 'INC/_CXSEC'
60: C      include 'INC/_PXSEC'
61: C      include 'INC/_SBANK'
62: C      include '../shared/INC/_TASKDT'
63: C      include '../shared/INC/_PTALY0'
64: C
65: C .... PARTICLE BANK .....
```

```

66: C
67: C      real*8 XXXI(IMPMAX), YYYYI(IMPMAX), ZZZI(IMPMAX), AAAI(IMPMAX),
68: C      &      BBBI(IMPMAX), CCCI(IMPMAX)
69: C      real WWWI(IMPMAX), EEI(IMPMAX)
70: C      real*8 TTTI(IMPMAX)
71: C      integer ITTI(IMPMAX)
72: C      integer IGGI(IMPMAX), LEVLI(IMPMAX), LZZI(IMPMAX,NEST),
73: C      &      LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST), KLSFI(IMPMAX)
74: c##<2007/03/14:PN4:
75: C      integer IZZI(IMPMAX)
76: c##>
77: C
78: C      ===== IMAGINARY PARTICLES =====
79: C
80: C      integer KDET(IMPMAX)
81: C      real*8 OPTI(IMPMAX), PATH(IMPMAX)
82: C
83: C      ===== optional bank parameters =====
84: C
85: C      real*8 DBNK(IMPMAX,*)
86: C      integer KDBNK(0:MDBNK)
87: C      integer IBNK(IMPMAX,*)
88: C      integer KIBNK(0:MIBNK)
89: C
90: C
91: C .... STACKS .....
```

```

92: C
93: C      integer IMSFL(IMPMAX), IMNFL(NZONE+1), IMZFL(IMPMAX),
94: C      &      IMSNX(IMPMAX), IMNNX(NZONE+1), IMZNX(IMPMAX),
95: C      &      IMDED(IMPMAX)
96: C
97: C .... GEOMETRY .....
```

```

98: C
99: C      real*8 SDA(NSDA), DALT(*), SZLAT(8,NLATT), CELSZ(6,*)
100: C      integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
101: C      &      KDALT(NLBZ), MLBZ(NLBZ), LTYP(NLATT), IPLAT(*), KSLAT(*),
102: C      &      KCELL(NZONE), ICTYP(NCELL), KLATT(*), NVLAT(4,*)
103: C      integer IPCEL(NCELL)
104: C      integer KZLBZ(NLBZ)
105: C
106: C      ... Nearest Neighbour Distribution for STGM region.
107: C
108: C      real PNND(NPNND)
109: C      integer KNND(NPNND)
110: C
111: C      ... Partial Packing Fraction for STGM-region
112: C
113: C      real PPPF(NPPPF)
114: C      integer KPPF(NPPPF)
115: C
116: C .... CROSS SECTION .....
```

```

117: C
118: C      real SMACI(IMPMAX,MB,NSMACI)
119: C      integer MMACI(IMPMAX,2)
120: c##<2007/03/14:PN4:
121: C      real DNZON(NUC,NZONE,2)
122: c##>
123: C
124: C .... TALLY .....
```

```

125: C
126: C      real*8 FLDP(NGROUP,*)
127: C      integer JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
128: C      integer IDTAL(*)
129: C      real*8 DTALY(*)
130: C      real RESP(NGROUP,NRESP)
```

src/mvp/nxtfli.f

```

131: c##<2007/03/14:PN4:
132:   real*8 XPDET(NPLEN,NPDET)
133: c##>
134: C
135: C .... WORKING AREA (LENGTH = IMPMAX) .....
136: C
137:   real*8 X(IMPMAX), Y(IMPMAX), Z(IMPMAX), AI(IMPMAX), BI(IMPMAX),
138:   &      CI(IMPMAX), OPT(IMPMAX), DI(IMPMAX), DLOC1(IMPMAX),
139:   &      DLOC2(IMPMAX), XUP(IMPMAX), YUP(IMPMAX), ZUP(IMPMAX),
140:   &      AUP(IMPMAX), BUP(IMPMAX), CUP(IMPMAX)
141:   real R(3*IMPMAX)
142:   integer IGI(IMPMAX), IBP(IMPMAX), IFC(IMPMAX)
143:   integer ILST(IMPMAX), ILUP(IMPMAX), ISTG(IMPMAX)
144:   real SIGT(IMPMAX), SMCW(IMPMAX,NOMICI)
145: C
146: C .... FOR SURFACE POINTER ...
147: C
148:   integer IKL(IMPMAX), IKL2(IMPMAX)
149: C
150:   include '../shared/INC/_PMLATT'
151:   include '../shared/INC/_SFLATT'
152: C
153: c##<2007/03/14:PN4:
154:   integer MATZ(100)
155:   logical LFZ, LFRR, LFWK
156: C
157:   data IFMZPD / -1 /
158:   save MATZ
159: C
160: C-----
161: C .... set of material number placed point detector ...
162: C-----
163: C
164:   if ( IFMZPD.eq.-1 ) then
165:     IFMZPD = 1
166:     if ( NPDET.gt.100 ) then
167:       write(IOW,991) NPDET, 100
168:       stop 666
169:     end if
170:     NP = 1
171:     LFRR = .false.
172:     do I = 1, NPDET
173:       do J = 1, NZONE
174:         IZ = J
175:         call JUDGE( 'NXTFLI', IZ, NP, XPDET(3,I), XPDET(4,I),
176:         &          XPDET(5,I), LFZ, SDA, KZDA, KZAA, NSDA, NZDA,
177:         &          NZONE+1, JVMNT0, LFWK, JSIMP0, LFRR, DUM1,
178:         &          DUM2, DUM3 )
179:         if ( LFZ ) then
180:           MATZ(I) = KZMAT(IZ,1)
181:           go to 70
182:         end if
183:       end do
184:       write(IOW,992) I, (XPDET(J,I),J=3,5)
185:       stop 666
186:     70 continue
187:   end do
188: end if
189: 991 format(/' XXX(NXTFLI) Storage size was lack for material number',
190: & ' of point detector.')
191: &/' NPDET=',i6,' limit of MATZ=',i6)
192: 992 format(/' XXX(NXTFLI) Zone number placed point detector was',
193: & ' unknown.')
194: &/' I=',i3,' point detector position (' ,lp,3e13.4,' )')
195: c##>

```

```

196: C
197: C-----
198: C .... set of total cross sections ...
199: C-----
200: C
201:   MAT = KZMAT(MZONE,1)
202: C
203: C ... this is a base material (matrix) region of STG region.
204: C
205:   JJJNND = 0
206:   if ( MAT.lt.0 ) then
207:     if ( ISLATT(MAT).and.LTYP(LATTNM(MAT)).eq.10 ) then
208:       MAT = KZMAT(MZONE,2)
209:       JJJNND = 1
210:     else
211:       write(IOW,*) 'XXX(NXTFLI) Program error ? Invalid zone :',
212:       &      ' MZONE=', MZONE, ' KZMAT(MZONE,*) ',
213:       &      KZMAT(MZONE,1), KZMAT(MZONE,2), ' LTYP ',
214:       &      LTYP(LATTNM(MAT))
215:       stop 666
216:     end if
217:   end if
218: C
219:   II = 0
220:   do 100 I = 1, IMNFL(NZONE+1)
221:     if ( IMZFL(I).eq.MZONE ) then
222:       II = II + 1
223:       IBP(II) = IMSFL(I)
224:     end if
225:   100 continue
226: C
227: C GATHER DATA OF NEUTRONS & PHOTONS SEPARATELY IN WORKING ARRAYS
228: C AS FOLLOWS:
229: C
230: C :-----:
231: C :      : NEUTRONS :      PHOTONS      :
232: C :-----:
233: C : INDEX : 1 ...  INEUT : INEUT+1 ... NFFL(MZONE) :
234: C :-----:
235: C
236: C
237: C .... NEUTRON/PHOTON COUPLED PROBLEM ...
238: C
239: C
240:   if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
241:     INEUT = 0
242:     NGG = KNGP(KPPHOT) - 1
243:   *VOCL LOOP,NOVREC
244:   do 110 I = 1, II
245:     if ( IGGI(IBP(I)).le.NGG ) then
246:       INEUT = INEUT + 1
247:       IFC(INEUT) = IBP(I)
248:     end if
249:   110 continue
250: C
251:   if ( INEUT.lt.II ) then
252:     IPHOT = IPHOT + 1
253:   *VOCL LOOP,NOVREC
254:   do 120 I = 1, II
255:     if ( IGGI(IBP(I)).gt.NGG ) then
256:       IPHOT = IPHOT + 1
257:       IFC(IPHOT) = IBP(I)
258:     end if
259:   120 continue
260:   end if

```

src/mvp/nxtfli.f

```

261: C
262:       do 130 I = 1, II
263:         IBP(I) = IFC(I)
264:       130 continue
265: C     .... ( IFC IS FREE TO USE HEREFTER )
266: C
267: C     .... NEUTRON OR PHOTON ONLY PROBLEM ....
268: C
269:       else
270:         if ( JPHOT.eq.0 ) then
271:           INEUT = II
272:         else
273:           INEUT = 0
274:         end if
275:       end if
276:
277:       if ( MAT.gt.0 ) then
278: C
279: C     ..... NEUTRONS .....
280: C
281:         if ( INEUT.gt.0 ) then
282:           JAMXCT = 0
283:           JMIC = 0
284:           call GMACNX( IRAND, INEUT, IBP, MAT,
285: &           JMIC, JAMXCT, JNEUT, JGAMM, JDEBG, IMPMAX,
286: &           EEEI, MB, NUC, NSTAL,
287: &           NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI,
288: &           NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTII),
289: &           A(LLPDEN), A(LINUCT), A(LDENST), NMAT, NUC, NMT,
290: &           A(LCX), MCX, A(LKLIB1), A(LKLIB2), A(LXLIB1),
291: &           A(LCRES), A(LKCRSI), MCRES, NNCSI, INCSI,
292: W           X, Y, Z, AI, BI, CI,
293: W           DI, OPT, R, SMCW )
294:
295:         end if
296:         if ( JPHOT.ne.0 ) then
297:           NM = II - INEUT
298:           if ( NM.gt.0 ) then
299:             JAMXCT = 0
300:             JMIC = 0
301:             JNP = 2
302:             call GMACNX( IRAND, NM, IBP(INEUT+1), MAT,
303: &             JMIC, JAMXCT, JNP, JGAMM, JDEBG, IMPMAX,
304: &             EEEI, MB, NUC, NSTAL,
305: &             NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI,
306: &             NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTII),
307: &             A(LLPDNP), A(LIATMT), A(LDNSTP), NMAT, NPATOM, NMTP,
308: &             A(LCXP), MCXP, A(LKLB1), A(LKLB2), A(LXLB1),
309: W             X, Y, Z, AI, BI, CI,
310: W             DI, OPT, R, SMCW )
311:
312:           end if
313:         end if
314: C
315: C     *VOCL LOOP,NOVREC
316:       do 140 I = 1, II
317:         SIGT(IBP(I)) = SMACI(IBP(I),MMACI(IBP(I),1),1)
318:       140 continue
319:       else
320: C     *VOCL LOOP,NOVREC
321:       do 150 I = 1, II
322:         SIGT(IBP(I)) = 0.0
323:         MMACI(IBP(I),1) = 0
324:       150 continue
325:       end if

```

```

326: C
327:       call NEEFLI( IOW, JVMNT, JFISX, JHLAT, JDEBG, JJJNND,
328: &       MZONE, NIMPT, NDMPT, NLOST, DEPS, DINF,
329: &       XXXI, YYI, ZZI, AAAI, BBBI, CCCI, WWI,
330: &       IGGI, TTTI, ITTI, LEVLI, LZZI, LPOSI, LCRSI,
331: &       KLSFI, PATH, OPTI, KDETP, SIGT,
332: &       DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
333: &       IMSFL, IMNFI, IMZFL, IMSNX, IMNX, IMZNX, IMDED,
334: &       NDII, NTRM,
335: &       SDA, KZMAT, KZREG, KZDA, KZAA, DALT, KDALT,
336: &       NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP, MLBZZ,
337: &       KZLBZ, KCELL, ICTYP, KLATT, CELSZ,
338: &       PNND, KNND, PPPF, KPPF,
339: &       X, Y, Z, AI, BI, CI, OPT,
340: &       DI, IGI, IBP, IFC, R, DLOC1,
341: &       DLOC2, IKL, IKL2, XUP, YUP, ZUP, AUP,
342: &       BUP, CUP, ISTG, ILUP, ILST )
343: C
344: C     === data for tallies =====
345: C     bank pointer : IMDED(NDII+1) ---> IMDED(NDII+NTRM)
346: C     NDII : starting bank pointer of particles to be tallied
347: C           by next event estimator.
348: C     NTRM : number of particles to be tallied by next event estimator.
349: C     OPT(1:NTRM) : contribution of i'th particle
350: C=====
351: C
352: C-----
353: C     .... TAKE SPECIAL TALLYS .....
354: C-----
355: C
356:       if ( NTEVE.gt.0.and.JTEVE(1).gt.0 ) then
357: C
358:         NNEUT = 0
359:         NPHOT = 0
360:         if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
361:           do 210 I = 1, NTRM
362:             if ( IGGI(IMDED(NDII+I)).le.NGG ) then
363:               NNEUT = NNEUT + 1
364:             end if
365:           210 continue
366:           NPHOT = NTRM - NNEUT
367:           else if ( JNEUT.ne.0 ) then
368:             NNEUT = NTRM
369:           else if ( JPHOT.ne.0 ) then
370:             NPHOT = NTRM
371:           end if
372: C
373:           do 220 J = 0, JTEVE(1) - 1
374: C
375: C     ... pointer to direct tally (idt) & d-tally # (it) ...
376:           IDT = KTEVE(LTEVE(1)+J) - 1
377:           IT = IDTAL(IDT+9)
378: C
379: C     .. neutron or photon ?
380:           if ( IDTAL(IDT+5).eq.1.and.NNEUT.eq.0 ) go to 220
381:           if ( IDTAL(IDT+5).eq.2.and.NPHOT.eq.0 ) go to 220
382: C
383:           I1 = 1
384:           I2 = 0
385:           if ( IDTAL(IDT+5).eq.1 ) then
386:             I1 = 1
387:             I2 = NNEUT
388:           else if ( IDTAL(IDT+5).eq.2 ) then
389:             I1 = NNEUT + 1
390:             I2 = NPHOT

```

src/mvp/nxtfli.f

```
391:         end if
392: c
393:         if ( I2.gt.0 ) then
394:             call STALN9(IOW, IDTAL(IDT+1), OPT(I1), IMDED(NDI1+I1),
395: &                 I2, DTALY,
396: &                 IMPMAX, NGROUP, NREG, NRESP, NSTAL,
397: &                 NZONE, NTIME, NMKREG, A(LMKREG),
398: &                 H(LSGTLI), RESP,
399: &                 NUC, NSMICI,
400: c##<2007/03/14:PN4:
401: c## &                 H(LSMICI), A(LLMICI),
402: &                 H(LSMICI), A(LLMICI), H(LSMACI), A(LLMACI),
403: &                 NSMACI, MB, H(LKMACI),
404: c##>
405: B                 KDET, IGGI, ITTI,
406: B                 DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
407: c##<2007/03/14:PN4:
408: c## W                 AI, BI, R )
409: W                 AI, BI, R,
410: &                 NPMOM, NMT, NMTP, KPNPRD, DNZON, IZZI, EEI,
411: &                 CI, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, A(LCXPNI),
412: &                 MCXPN, A(LINCSTPN), A(LMTMPN), NMIMP,
413: &                 A(LKLBPN1), A(LKLBPN2), A(LXBBPN1), EBOTLPN,
414: &                 NUCPNA, A(LMNEUTPN), A(LDNSTPN), A(LMNUCPN),
415: &                 A(LLPIDPN), ISTG, ILUP, ILST, MATZ, IKL )
416: c##>
417: C
418:         end if
419: 220 continue
420: c
421:     end if
422: C
423:     return
424: end
```

src/mvp/nxtmvt.f

```

1:      subroutine NXTMVT( IOW, NF      , NBANK, IPSK ,
2:      &                  EIN  , A0    , B0    , C0    , EMIU  ,
3:      &                  ISF5  , ISF4  , ILF    , ICLN  , IMT    ,
4:      &                  CX    , ICX    , MCX    , XLIB1 , XLIB2 ,
5:      &                  NUC    , NMT    ,
6:      &                  IWK1  , WK1    , WK2    , WK3    , WK4    , WK5    ,
7:      &                  WK6    , WK7    , WK8    ,
8:      &                  R      , IRAND  )
9: C-----1-----5-----2-----5-----3-----5-----4-----5-----5-----6-----5-----7-----
10: C=====
11: C PURPOSE: calculate effective flight direction and target mass for
12: C the case of moving target. ( only elastic scattering )
13: C
14: C IPSK(I) : giving I
15: C EIN(I)  : incident energy ( changed to effective energy )
16: C EMIU(I) : effective mass of target <---- mass
17: C ISF5(I) : pointer to used data in CX array
18: C ISF4(I) : pointer to energy meshes in CX array
19: C ILF(I)  : incident energy mesh
20: C ICLN(I) : collision nuclide
21: C IMT(I)  : reaction MT
22: C A0(I),B0(I),C0(I) : flight direction of neutrons ( changed )
23: C
24: C
25: C CALLED IN: NXTNR
26: C CALL      : RANU2
27: C=====
28: C
29:      implicit real*8(A-H,O-Z)
30: C
31:      integer IPSK(NBANK)
32: C
33:      integer ISF5(NBANK), ISF4(NBANK), ILF(NBANK), ICLN(NBANK),
34:      &          IMT(NBANK)
35:      real      EIN(NBANK), EMIU(NBANK)
36:      real*8    A0(NBANK), B0(NBANK), C0(NBANK)
37: C
38: C**** CROSS SECTION DATA IN CX ARRAY
39:      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
40:      integer ICX(MCX)
41: c##<2007/03/14:PN3:
42: c## integer KLIB1(NUC,27)
43: integer KLIB1(NUC,31)
44: c##>
45: C
46: C
47: C**** WORK AREA
48: C
49:      integer IWK1(NBANK)
50:      real      WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
51:      &          WK5(NBANK), WK6(NBANK), WK7(NBANK), WK8(NBANK)
52:      real      R(NBANK*7)
53: C
54: C****
55:      parameter( PI = 3.141592653589793200D+00 )
56:      parameter( PIH = 0.5D0*PI )
57:      parameter( PI2 = 2.0D0*PI )
58:      parameter( PI2I = 1.0D0/(2.0D0*PI ) )
59:      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
60:      parameter( SMALL = 1.0E-35 )
61: C
62: C
63: C
64: C**** Free gas model ( moving target )
65: C

```

```

66:      call RANU2( IRAND, R, NF*7, ICON )
67: C
68:      NF2 = NF*2
69:      NF3 = NF*3
70:      NF4 = NF*4
71:      NF5 = NF*5
72:      NF6 = NF*6
73: C
74: C      WK2 : ATW
75: C      WK3 : ATW/KT*EEE
76: C      WK4 : ATW/KTeff*EEE WHEN EEE<=5eV
77: C..... WK5 : V(TARGET NUCLIDE) / V(NEUTRON) : XT
78: C      X2 : WK5**2
79: C      (WK6,WK7,WK8) : DIRECTION COSINE OF TARGET
80: C      WK1 : SQRT(1.+X2-2*XMU*XT)
81: C      IWK1(NN) : IWK1(k)---> JJ
82: C
83:      NN = 0
84: C
85: *VOCL LOOP,NOVREC
86:      do 180 J = 1, NF
87:          I = IPSK(J)
88:          WK2(J) = EMIU(I)
89:          WK3(J) = XLIB1(ICLN(I),9)*EIN(I)
90: C .... neglect short collision approximation
91: c      if ( EEEJ(I).le.5. ) then
92: c          WK4(J) = XLIB1(ICLN(I),11)*EIN(I)
93: c      else
94: c          WK4(J) = WK3(J)
95: c      end if
96: c      if ( (4.*R(J)*R(J)).gt.
97: c          & (PI*WK4(J)*(1.-R(J))*(1.-R(J))) ) then
98: c          X2 = -LOG(R(NF+J)*R(NF2+J)) /WK4(J)
99: c      else
100: c          COS2 = COS(PIH*R(NF+J))
101: c          COS2 = COS2*COS2
102: c          X2 = -(LOG(R(NF2+J))+LOG(R(NF3+J))*COS2) /WK4(J)
103: c      end if
104: C
105:      WK5(J) = SQRT(X2)
106: C
107: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
108: C
109:      WK6(J) = 2.*R(NF4+J) - 1.0
110:      SINT = SQRT(1.-WK6(J)*WK6(J))
111:      FAI = PI2*R(NF5+J)
112:      WK7(J) = SINT*COS(FAI)
113:      WK8(J) = SINT*SIN(FAI)
114:      XMU = WK6(J)*A0(I) + WK7(J)*B0(I) + WK8(J)*C0(I)
115: C
116:      WK1(J) = SQRT(1.+X2-2.*XMU*WK5(J))
117: C
118: c      if ( WK1(J).lt.R(NF6+J)*(1.+WK5(J)) ) then
119: c          NN = NN + 1
120: c          IWK1(NN) = J
121: c      end if
122: 180 continue
123: C
124: C**** RESAMPLING FOR REJECTED PARTICLES
125: C
126: 190 continue
127: C
128: c      if ( NN.gt.0 ) then
129: C
130:      NF1 = NN

```

src/mvp/nxtmvt.f

```

131:      NN      = 0
132:      call RANU2( IRAND, R, NF1*7, ICON )
133: C
134:      NF2      = NF1*2
135:      NF3      = NF1*3
136:      NF4      = NF1*4
137:      NF5      = NF1*5
138:      NF6      = NF1*6
139: C
140: *VOCL LOOP,NOVREC
141: do 200 K = 1, NF1
142:   J      = IWK1(K)
143:   I      = IPSK(J)
144:   AEKT   = WK4(J)
145:   if ( (4.*R(K)*R(K)).gt.(PI*AEKT*(1.-R(K))*(1.-R(K))) )
146: &      then
147:     X2    = -LOG(R(NF1+K)*R(NF2+K)) /AEKT
148:   else
149:     COS2  = COS(PIH*R(NF1+K))
150:     COS2  = COS2*COS2
151:     X2    = -(LOG(R(NF2+K))+LOG(R(NF3+K))*COS2) /AEKT
152:   end if
153:   WK5(J) = SQRT(X2)
154: C
155: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
156: C
157:   WK6(J) = 2.*R(NF4+K) - 1.0
158:   SINT   = SQRT(1.-WK6(J)*WK6(J))
159:   FAI    = PI2*R(NF5+K)
160:   WK7(J) = SINT*COS(FAI)
161:   WK8(J) = SINT*SIN(FAI)
162:   XMU    = WK6(J)*A0(I) + WK7(J)*B0(I) + WK8(J)*C0(I)
163: C
164:   WK1(J) = SQRT(1.+X2-2.*XMU*WK5(J))
165: C
166:   if ( WK1(J).lt.R(NF6+K)*(1.+WK5(J)) ) then
167:     NN      = NN + 1
168:     IWK1(NN) = J
169:   end if
170: 200 continue
171: C
172: go to 190
173: C
174: end if
175: C
176: *VOCL LOOP,NOVREC
177: do 210 J = 1, NF
178:   I      = IPSK(J)
179:   AX     = WK2(J) * WK5(J)
180: C ..... effective direction
181:   WK6(J) = A0(I) + AX*WK6(J)
182:   WK7(J) = B0(I) + AX*WK7(J)
183:   WK8(J) = C0(I) + AX*WK8(J)
184:   SS2    = WK6(J)**2 + WK7(J)**2 + WK8(J)**2
185:   SS     = SQRT(SS2)
186:   if ( SS.gt.0. ) then
187:     A0(I) = WK6(J) / SS
188:     B0(I) = WK7(J) / SS
189:     C0(I) = WK8(J) / SS
190: C .... effective mass
191:     EMIU(I) = WK2(J) * WK1(J) / SS
192:   else
193:     EMIU(I) = 0.0
194:   end if
195: C .... effective energy

```

```

196:      EIN(I) = EIN(I) * SS2 * ((EMIU(I)+1.)/(WK2(J)+1.))**2
197: 210 continue
198: C
199: return
200: end
201: C
202: C..... intital one
203: c subroutine NXTMVT( IOW, NF , NBANK,
204: c &      EIN , A0 , B0 , C0 , EMIU ,
205: c &      ISF5 , ISF4 , ILF , ICLN , IMT ,
206: c &      CX , ICX , MCX , KLIB1, XLIB1, XLIB2,
207: c &      NUC , NMT ,
208: c &      IWK1 , WK1 , WK2 , WK3 , WK4 , WK5 ,
209: c &      WK6 , WK7 , WK8 ,
210: c &      R , IRAND )
211: C-----5-----1-----5-----2-----5-----3-----5-----4-----5-----5-----6-----5-----7---
212: C=====
213: C PURPOSE: calculate effective flight direction and target mass for
214: C the case of moving target. ( only elastic scattering )
215: C
216: C EIN(I) : incident energy ( changed to effective energy )
217: C EMIU(I) : effective mass of target
218: C ISF5(I) : pointer to used data in CX array
219: C ISF4(I) : pointer to energy meshes in CX array
220: C ILF(I) : incident energy mesh
221: C ICLN(I) : collision nuclide
222: C IMT(I) : reaction MT
223: C A0(I),B0(I),C0(I) : flight direction of neutrons ( changed )
224: C
225: C
226: C CALLED IN: NXTNR
227: C CALL : RANU2
228: C UPDATE :
229: C=====
230: C
231: c implicit real*8(A-H,O-Z)
232: C
233: c integer ISF5(NBANK), ISF4(NBANK), ILF(NBANK), ICLN(NBANK),
234: c &      IMT(NBANK)
235: c real EIN(NBANK), EMIU(NBANK)
236: c real*8 A0(NBANK), B0(NBANK), C0(NBANK)
237: C
238: C**** CROSS SECTION DATA IN CX ARRAY
239: c real CX(MCX), KLIB1(NUC,11), XLIB2(NUC,NMT,4)
240: c integer ICX(MCX)
241: c integer KLIB1(NUC,27)
242: C
243: C
244: C**** WORK AREA
245: C
246: c integer IWK1(NBANK)
247: c real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
248: c &      WK5(NBANK), WK6(NBANK), WK7(NBANK), WK8(NBANK)
249: c real R(NBANK*7)
250: C
251: C****
252: c parameter( PI = 3.141592653589793200D+00 )
253: c parameter( PIH = 0.5D0*PI )
254: c parameter( PI2 = 2.0D0*PI )
255: c parameter( PI2I = 1.0D0/(2.0D0*PI ) )
256: c parameter( PI4I = 1.0D0/(4.0D0*PI ) )
257: c parameter( SMALL = 1.0E-35 )
258: C
259: C
260: C

```

src/mvp/nxtmvt.f

```

261: C**** Free gas model ( moving target )
262: C
263: C      call RANU2( IRAND, R, NF*7, ICON )
264: C
265: C      NF2      = NF*2
266: C      NF3      = NF*3
267: C      NF4      = NF*4
268: C      NF5      = NF*5
269: C      NF6      = NF*6
270: C
271: C      WK2 : ATW
272: C      WK3 : ATW/KT*EEE
273: C      WK4 : ATW/KTeff*EEE WHEN EEE<=5eV
274: C..... WK5      : V(TARGET NUCLIDE) / V(NEUTRON) : XT
275: C      X2      : WK5**2
276: C      (WK6,WK7,WK8) : DIRECTION COSINE OF TARGET
277: C      WK1      : SQRT(1.+X2-2*XMU*XT)
278: C      IWK1(NN) : IWK1(k)---> JJ
279: C
280: C      NN      = 0
281: C
282: C*VOCL LOOP,NOVREC
283: C      do 180 I = 1, NF
284: C      WK2(I) = XLIB1(ICLN(I),1)
285: C      WK3(I) = XLIB1(ICLN(I),9)*EIN(I)
286: C      .... neglect short collision approximation
287: C      if ( EEEJ(I).le.5e-5 ) then
288: C      WK4(I) = XLIB1(ICLN(I),11)*EIN(I)
289: C      else
290: C      WK4(I) = WK3(I)
291: C      end if
292: C      if ( (4.*R(I)*R(I)).gt.
293: C      &      (PI*WK4(I)*(1.-R(I))*(1.-R(I))) ) then
294: C      X2      = -LOG(R(NF+I)*R(NF2+I)) /WK4(I)
295: C      else
296: C      COS2      = COS(PIH*R(NF+I))
297: C      COS2      = COS2*COS2
298: C      X2      = -(LOG(R(NF2+I))+LOG(R(NF3+I))*COS2) /WK4(I)
299: C      end if
300: C
301: C      WK5(I) = SQRT(X2)
302: C
303: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
304: C
305: C      WK6(I) = 2.*R(NF4+I) - 1.0
306: C      SINT      = SQRT(1.-WK6(I)*WK6(I))
307: C      FAI      = PI2*R(NF5+I)
308: C      WK7(I) = SINT*COS(FAI)
309: C      WK8(I) = SINT*SIN(FAI)
310: C      XMU      = WK6(I)*A0(I) + WK7(I)*B0(I) + WK8(I)*C0(I)
311: C
312: C      WK1(I) = SQRT(1.+X2-2.*XMU*WK5(I))
313: C
314: C      if ( WK1(I).lt.R(NF6+I)*(1.+WK5(I)) ) then
315: C      NN      = NN + 1
316: C      IWK1(NN) = I
317: C      end if
318: C 180      continue
319: C
320: C**** RESAMPLING FOR REJECTED PARTICLES
321: C
322: C 190      continue
323: C
324: C      if ( NN.gt.0 ) then
325: C

```

```

326: C      NF1      = NN
327: C      NN      = 0
328: C      call RANU2( IRAND, R, NF1*7, ICON )
329: C
330: C      NF2      = NF1*2
331: C      NF3      = NF1*3
332: C      NF4      = NF1*4
333: C      NF5      = NF1*5
334: C      NF6      = NF1*6
335: C
336: C*VOCL LOOP,NOVREC
337: C      do 200 K = 1, NF1
338: C      I      = IWK1(K)
339: C      AEKT      = WK4(I)
340: C      if ( (4.*R(K)*R(K)).gt.(PI*AEKT*(1.-R(K))*(1.-R(K))) )
341: C      &      then
342: C      X2      = -LOG(R(NF1+K)*R(NF2+K)) /WK4(I)
343: C      else
344: C      COS2      = COS(PIH*R(NF1+K))
345: C      COS2      = COS2*COS2
346: C      X2      = -(LOG(R(NF2+K))+LOG(R(NF3+K))*COS2) /AEKT
347: C      end if
348: C      WK5(I) = SQRT(X2)
349: C
350: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
351: C
352: C      WK6(I) = 2.*R(NF4+K) - 1.0
353: C      SINT      = SQRT(1.-WK6(I)*WK6(I))
354: C      FAI      = PI2*R(NF5+K)
355: C      WK7(I) = SINT*COS(FAI)
356: C      WK8(I) = SINT*SIN(FAI)
357: C      XMU      = WK6(I)*A0(I) + WK7(I)*B0(I) + WK8(I)*C0(I)
358: C
359: C      WK1(I) = SQRT(1.+X2-2.*XMU*WK5(I))
360: C
361: C      if ( WK1(I).lt.R(NF6+K)*(1.+WK5(I)) ) then
362: C      NN      = NN + 1
363: C      IWK1(NN) = I
364: C      end if
365: C 200      continue
366: C
367: C      go to 190
368: C
369: C      end if
370: C
371: C      do 210 I = 1, NF
372: C      AX      = WK2(I) * WK5(I)
373: C      .... effective direction
374: C      WK6(I) = A0(I) + AX*WK6(I)
375: C      WK7(I) = B0(I) + AX*WK7(I)
376: C      WK8(I) = C0(I) + AX*WK8(I)
377: C      SS2      = WK6(I)**2 + WK7(I)**2 + WK8(I)**2
378: C      SS      = SQRT(SS2)
379: C      if ( SS.gt.0. ) then
380: C      A0(I) = WK6(I) / SS
381: C      B0(I) = WK7(I) / SS
382: C      C0(I) = WK8(I) / SS
383: C      .... effective mass
384: C      EMIU(I) = WK2(I) * WK1(I) / SS
385: C      else
386: C      EMIU(I) = 0.0
387: C      end if
388: C      .... effective energy
389: C      EIN(I) = EIN(I) * SS2 * ((EMIU(I)+1.)/(WK2(I)+1.))**2
390: C 210      continue

```

src/mvp/nxtmvt.f

```
391: C  
392: c   return  
393: c   end
```



src/mvp/nxtnr2.f

```

1:      subroutine NXTNR2(   IOW, A, H,   LSCOL, NIGEN,
2:      B      EEE   , WWW   ,
3:      B      DBNK  , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
4:      B      NDBNK, NIBNK,
5:      X      CX    , ICX   , KLIB1, XLIB1, KLIB2, XLIB2,
6:      X      KMAC  , MMAC  , SMIC , LMIC , KSPI ,
7:      B      IBP   , ICLN , IMT   , ISF4 , ISF5 , ILF   ,
8:      B      EIN   , W    , THEA , THEB , EMIU , NRTYP,
9:      W      IWK1  , IWK2  , IWK3 , IWK4 , IWK5 , IWK6 ,
10:     W      IWK7  , IWK8  , IWK9 , IWK10, IWK11,
11:     W      WK1   , WK2   , WK3   , WK4   , WW    ,
12:     W      R     )
13: C=====
14: C
15: C  PURPOSE: Selection of neutron reaction for next event estimator
16: C  CALLED IN: NXTNR
17: C  CALLS: RANU2,BSDEC3
18: C
19: C=====
20: C
21: C  === INTER-STACK DATA TREATED ===
22: C
23: C  COLLISION STACK ----->
24: C  (LSCOL,NCOLS)
25: C  (NOT REMOVED)
26: C  IBP(I)   : bank pointer
27: C  ICLN(I)  : collision nuclide
28: C  IMT(I)   : neutron emitting reaction
29: C  EIN(I)   : incident energy
30: C  W(I)     : particle weights including multiplicity
31: C  NRTYP(10) : cummulative number of particles
32: C      1 : elastic scattering
33: C      2 : reaction with 2-body kinematics
34: C      3 : reaction with files 4 & 5
35: C      4 : fission
36: C      5 : reaction with file 6
37: C      6 : thermal inelastic
38: C      7 : incoherent thermal elastic
39: C      8 : coherent thermal elastic
40: C      9 : free gas
41: C     10 : isotropic without energy change
42: C
43: C=====
44: C  implicit real*8 (A-H,O-Z)
45: C
46: C  real A(*)
47: C  real H(*)
48: C
49: C  include '../shared/INC/_SIZES'
50: C  include 'INC/_SIZES2'
51: C  include 'INC/_FLAGS'
52: C  include 'INC/_STACK'
53: C  include 'INC/_KPIDS'
54: C  include 'INC/_CXSEC'
55: C  include 'INC/_PXSEC'
56: C  include 'INC/_SBANK'
57: C  include '../shared/INC/_PTALY0'
58: C  include 'INC/_PTALY'
59: C  include 'INC/_PTALY2'
60: C
61: C
62: C**** CROSS SECTION DATA
63: C
64: C  real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
65: C  c##<2007/03/14:PN3:

```

```

66: C##  integer ICX(MCX), KLIB1(NUC,27), KLIB2(NUC,NMT,21)
67: C  integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
68: C##>
69: C
70: C**** SIGMA BANK
71: C
72: C  real SMIC(NBANK,NUC,NSMIC)
73: C  integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8)
74: C
75: C**** STACK FOR REAL PARTICLE
76: C
77: C  integer LSCOL(NBANK)
78: C
79: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
80: C
81: C  real EEE(NBANK,2), WWW(NBANK,2)
82: C
83: C  real*8 DBNK(NBANK,NDBNK,2)
84: C  integer KDBNK(0:MDBNK)
85: C  integer IBNK(NBANK,NIBNK,2)
86: C  integer KIBNK(0:MIBNK)
87: C
88: C**** TALLY (MONITOR) ARRAY
89: C
90: C  real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
91: C
92: C  integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
93: C  & IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
94: C  & IWK9(NBANK), IWK10(NBANK), IWK11(NBANK)
95: C  real R(3*NBANK)
96: C  real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK)
97: C
98: C**** Prepared stack and bank
99: C
100: C  integer IBP(NBANK), ICLN(NBANK), IMT(NBANK), ISF4(NBANK),
101: C  & ISF5(NBANK), ILF(NBANK)
102: C  integer NRTYP(10)
103: C  real W(NBANK), WW(NBANK), EIN(NBANK), THEA(NBANK), THEB(NBANK),
104: C  & EMIU(NBANK)
105: C
106: C****
107: C
108: C
109: C  parameter( PI = 3.1415926535897932D+00 )
110: C  parameter( PI2 = 2.0D0*PI )
111: C  parameter( PIH = 0.5D0*PI )
112: C  parameter( PI4I = 1.0D0/(4.0D0*PI) )
113: C
114: C
115: C=====
116: C  if ( JVMNT.ne.0 ) call VMNTR1( 5, NCOLS )
117: C
118: C=====
119: C  selection of reaction type generating imaginary particles.
120: C      Bank pointer, collision nuclide, weight
121: C      LSCOL ---> IBP , IWK1(LSCOL(I)) WW(LSCOL(I))
122: C      IWK2, IWK1(LSCOL(I)) WW(LSCOL(I))
123: C=====
124: C
125: C  .... NF: Number of fissions, NS: Number of scattering
126: C
127: C  NS = 0
128: C  NF = 0
129: C
130: C  .... No fission .....

```

src/mvp/nxtnr2.f

```

131: C
132:   if ( JEIGN.ne.0.or.JFISS.eq.0 ) then
133:     if ( JPHOT.eq.0 ) then
134: *VOCL LOOP,NOVREC
135:       do 110 I = 1, NCOLS
136:         MAT = MMAC(LSCOL(I),1)
137:         KN = KMAC(LSCOL(I),MAT,1)
138:         IWK1(LSCOL(I)) = KN
139:         IBP(I) = LSCOL(I)
140: C       .... PS : scattering probability ...
141:         PS = 1.0 -
142: &         ( SMIC(LSCOL(I),KN,LMIC(3)) +
143: &         SMIC(LSCOL(I),KN,LMIC(5)) )
144: &         / SMIC(LSCOL(I),KN,LMIC(1))
145: C       .... WW = WWW * (scattering probability)
146:         WW(LSCOL(I)) = WWW(LSCOL(I),1)*PS
147: 110       continue
148: C
149:       else
150:         call RANU2( IRAND, R, NCOLS, ICON )
151: *VOCL LOOP,NOVREC
152:       do 120 I = 1, NCOLS
153:         MAT = MMAC(LSCOL(I),1)
154:         KN = KMAC(LSCOL(I),MAT,1)
155:         IWK1(LSCOL(I)) = KN
156:         IBP(I) = LSCOL(I)
157: C       .... PS : scattering probability ...
158:         PS = 1.0 -
159: &         ( SMIC(LSCOL(I),KN,LMIC(3)) +
160: &         SMIC(LSCOL(I),KN,LMIC(5)) )
161: &         / SMIC(LSCOL(I),KN,LMIC(1))
162: C       .... PFS : (scattering+fission) probability ...
163:         PFS = 1.0 -
164: &         SMIC(LSCOL(I),KN,LMIC(5))
165: &         / SMIC(LSCOL(I),KN,LMIC(1))
166: C       .... WW = WWW * (scattering probability)
167:         WW(LSCOL(I)) = WWW(LSCOL(I),1)*PS
168: C       .... capture gamma ...
169:         if ( R(I).gt.PFS ) then
170:           IWK1(LSCOL(I)) = 0
171: C       .... fission gamma ...
172:         else if ( R(I).gt.PS ) then
173:           IWK1(LSCOL(I)) = -18
174: C       .... neutron scattering (inelastic gamma etc.) ...
175:         else
176:           IWK1(LSCOL(I)) = 1
177:         end if
178: 120       continue
179:       end if
180: C
181:       NS = NCOLS
182: C
183: C       .... Fission occurs .....
184: C
185:       else
186: C
187:         if ( JPHOT.eq.0 ) then
188:           call RANU2( IRAND, R, NCOLS, ICON )
189: C
190: *VOCL LOOP,NOVREC
191:       do 130 I = 1, NCOLS
192:         MAT = MMAC(LSCOL(I),1)
193:         KN = KMAC(LSCOL(I),MAT,1)
194:         IWK1(LSCOL(I)) = KN
195: C

```

```

196: C       .... SIG : (fission+scattering) cross section ( total - capture )
197: C       .... PF : fission probability when scattering+fission is adopted.
198:         SIG = SMIC(LSCOL(I),KN,LMIC(1))
199: &         - SMIC(LSCOL(I),KN,LMIC(5))
200:         PFS = SIG / SMIC(LSCOL(I),KN,LMIC(1))
201:         PF = SMIC(LSCOL(I),KN,LMIC(3)) / SIG
202: C
203:         if ( R(I).gt.PF ) then
204:           NS = NS + 1
205:           IBP(NS) = LSCOL(I)
206: C       .... WW = WWW * (scattering+fission probability)
207:           WW(LSCOL(I)) = WWW(LSCOL(I),1)*PFS
208:         else
209:           NF = NF + 1
210:           IWK2(NF) = LSCOL(I)
211: C       .... WW = WWW * (scattering+fission probability) * nu
212:           WW(LSCOL(I)) = WWW(LSCOL(I),1)*PFS*
213: &           SMIC(LSCOL(I),KN,LMIC(2)) /
214: &           SMIC(LSCOL(I),KN,LMIC(3))
215:         end if
216: 130       continue
217:       else
218:         call RANU2( IRAND, R, NCOLS*2, ICON )
219: C
220: *VOCL LOOP,NOVREC
221:       do 140 I = 1, NCOLS
222:         MAT = MMAC(LSCOL(I),1)
223:         KN = KMAC(LSCOL(I),MAT,1)
224:         IWK1(LSCOL(I)) = KN
225: C
226: C       .... SIG : (fission+scattering) cross section ( total - capture )
227: C       .... PF : fission probability when scattering+fission is adopted.
228:         SIG = SMIC(LSCOL(I),KN,LMIC(1))
229: &         - SMIC(LSCOL(I),KN,LMIC(5))
230:         PFS = SIG / SMIC(LSCOL(I),KN,LMIC(1))
231:         PF = SMIC(LSCOL(I),KN,LMIC(3)) / SIG
232: C
233:         if ( R(I).gt.PF ) then
234:           NS = NS + 1
235:           IBP(NS) = LSCOL(I)
236: C       .... WW = WWW * (scattering+fission probability)
237:           WW(LSCOL(I)) = WWW(LSCOL(I),1)*PFS
238:         else
239:           NF = NF + 1
240:           IWK2(NF) = LSCOL(I)
241: C       .... WW = WWW * (scattering+fission probability) * nu
242:           WW(LSCOL(I)) = WWW(LSCOL(I),1)*PFS*
243: &           SMIC(LSCOL(I),KN,LMIC(2)) /
244: &           SMIC(LSCOL(I),KN,LMIC(3))
245:         end if
246: C       .... capture gamma ...
247:         if ( R(I+NCOLS).gt.PFS ) then
248:           IWK1(LSCOL(I)) = 0
249: C       .... neutron scattering or fission gamma ...
250:         else
251:           IWK1(LSCOL(I)) = 1
252:         end if
253: 140       continue
254:       end if
255: C
256:       end if
257: C
258: C =====
259: C Neutron scattering reaction
260: C =====

```

src/mvp/nxtnr2.f

```

261: C
262: C-----
263: C.... Determination of reaction type
264: C   IBP(I),NS ----> IBP , NS,      IWK5(IBP(I)) (non-thermal)
265: C                        IWK3 , NCOLF      (free-gas)
266: C                        IWK4 , NCOLT,      (thermal)
267: C                        STACK NO.      MT
268: C   NELS: # of neutrons making elastic scattering,
269: C   NKIN: # of neutrons making kinematics available reaction
270: C   except elastic scattering
271: C   NF5: # of neutrons making reaction with file5
272: C   NF6: # of neutrons making reaction with file6
273: C-----
274: C
275: C
276: C   NN      = 0
277: C   NCOLF   = 0
278: C   NCOLT   = 0
279: C   NTHR    = 0
280: C   NELS    = 0
281: C   NKIN    = 0
282: C   NF5     = 0
283: C   NF6     = 0
284: C
285: C *VOCL LOOP,NOVREC
286: C   do 200 I = 1, NS
287: C     IP      = IBP(I)
288: C     KN      = IWK1(IP)
289: C
290: C   -----
291: C   Non-thermal reaction   ( E > ETHMAX )
292: C   -----
293: C
294: C   if ( EEE(IP,1).gt.ETHMAX ) then
295: C     NN      = NN + 1
296: C     IBP(NN) = IP
297: C
298: C   -----
299: C   Free gas               ( E > ESAB and E > ETHERM )
300: C   -----
301: C
302: C   else if ( KLIB1(KN,15).eq.0 .or.
303: C   &        ( EEE(IP,1).gt.XLIB1(KN,4) .and.
304: C   &        EEE(IP,1).gt.XLIB1(KN,3) ) ) then
305: C     NCOLF   = NCOLF + 1
306: C     IWK3(NCOLF) = IP
307: C
308: C   -----
309: C   Thermal reaction ..... S(A,B)   ( E <= ESAB or E<=ETHERM)
310: C   -----
311: C
312: C   else
313: C     NCOLT   = NCOLT + 1
314: C     IWK4(NCOLT) = IP
315: C   end if
316: C   200 continue
317: C   NS      = NN
318: C
319: C   if ( NS.gt.0 ) then
320: C
321: C-----
322: C.... Determination of threshold reaction (MT --> IWK5)
323: C-----
324: C
325: C   call RANU2( IRAND, R, NS, ICON )

```

```

326: C
327: C *VOCL LOOP,NOVREC
328: C   do 210 I = 1, NS
329: C     IP      = IBP(I)
330: C
331: C   .... X1 : elastic   X2 : (Total - Absorption)*(Random number)
332: C
333: C     X1      = SMIC(IP,IWK1(IP),LMIC(4))
334: C     X2      = (SMIC(IP,IWK1(IP),LMIC(1))
335: C   &          -SMIC(IP,IWK1(IP),LMIC(3))
336: C   &          -SMIC(IP,IWK1(IP),LMIC(5)))*R(I)
337: C
338: C   -----
339: C   X1 < X2 and E > ETH   ==> Threshold reaction
340: C   -----
341: C
342: C   MTTHR     = KLIB2(IWK1(IP),1,10)
343: C   if ( MTTHR.gt.0.and.X1.lt.X2
344: C   &       .and.XLIB2(IWK1(IP),MTTHR,2).lt.EEE(IP,1) ) then
345: C     NTHR     = NTHR + 1
346: C     IWK6(NTHR) = IP
347: C   ... MT = 2 : elastic scattering
348: C   else
349: C     NELS     = NELS + 1
350: C     IBP(NELS) = IP
351: C     IWK5(IP) = 2
352: C   end if
353: C   210 continue
354: C
355: C   if ( NTHR.gt.0 ) then
356: C     call RANU2( IRAND, R, NTHR, ICON )
357: C     MA      = 0
358: C     NN      = 0
359: C   *VOCL LOOP,NOVREC
360: C     do 220 I = 1, NTHR
361: C       IP      = IWK6(I)
362: C       KN      = IWK1(IP)
363: C
364: C   MT # of total threshold reaction
365: C   If only one threshold reaction is possible,
366: C   this is the reaction number selected.
367: C   -----
368: C     MT      = KLIB2(KN,1,10)
369: C     IWK5(IP) = MT
370: C
371: C   ..... Number of threshold reaction (NLEV) .....
372: C
373: C     NLEV     = KLIB1(KN,6)
374: C     if ( NLEV.ge.2 ) then
375: C       NN      = NN + 1
376: C       IWK8(NN) = IP
377: C       IWK7(IP) = NLEV
378: C
379: C   ..... Maximum number of threshold reaction .....
380: C
381: C     MA      = MAX(MA,NLEV)
382: C
383: C   .... Energy mesh interval (pointer)
384: C
385: C     L2      = KLIB1(KN,1) + KSPI(IP,KN)
386: C     WK1(IP) = (EEE(IP,1)-CX(L2)) / (CX(L2-1)-CX(L2))
387: C     WK2(IP) = 1. - WK1(IP)
388: C     L1      = KLIB2(KN,MT,11) - KLIB2(KN,MT,3)
389: C   &          + KSPI(IP,KN)
390: C

```

src/mvp/nxtnr2.f

```

391: C....      Threshold total cross section * random number
392: C
393:           WK3(IP) = (CX(L1)*WK1(IP)+CX(L1+1)*WK2(IP))*R(I)
394:           end if
395: 220         continue
396: C
397: C-----
398: C   Select a threshold reaction if more than two threshold reactions
399: C   are possible.  (MA > 1)
400: C-----
401: C
402:           do 230 N = 2, MA
403:             NTHR2 = NN
404:             NN     = 0
405: *VOCL LOOP,NOVREC
406:           do 240 I = 1, NTHR2
407:             IP = IWK8(I)
408:             KN = IWK1(IP)
409:             MT = KLIB2(KN,N,10)
410: C
411: C   Pick up cross section if energy point (KSPI) is within energy
412: C   point-range of MT=KLIB2(KN,N,10), else gives zero x-sec.
413: C-----
414:           X1 = 0.
415:           if ( KLIB2(KN,MT,3).le.KSPI(IP,KN)
416: &           .and.KLIB2(KN,MT,4).gt.KSPI(IP,KN) ) then
417:             L1 = KLIB2(KN,MT,11) - KLIB2(KN,MT,3)
418: &             + KSPI(IP,KN)
419:             X1 = CX(L1)*WK1(IP) + CX(L1+1)*WK2(IP)
420:           end if
421: C-----
422: C   X1 >= (total threshold cross section)*(random number)
423: C   ==> This reaction is selected
424: C-----
425:           if ( X1.ge.WK3(IP) ) then
426:             IWK5(IP) = MT
427:             if ( N.lt.IWK7(IP) ) then
428:               NN = NN + 1
429:               IWK8(NN) = IP
430:             end if
431:           end if
432: 240         continue
433:           if ( NN.le.0 ) go to 250
434: 230       continue
435: C
436: 250       continue
437: C
438: C   .... multiplicity of selected reaction is multiplied.
439: C   select MT=6-9 and MT for which file6 is used.
440: C
441:           NFH = 0
442:           N69 = 0
443: *VOCL LOOP,NOVREC
444:           do 260 I = 1, NTHR
445:             IP = IWK6(I)
446:             MULTI = KLIB2(IWK1(IP),IWK5(IP),5)
447: C   .... MULTI>4, multiplicity is given in file 6 ....
448:             if ( MULTI.gt.4 ) then
449:               NF6 = NF6 + 1
450:               IWK7(NF6) = IP
451:             else
452:               NFH = NFH + 1
453:               IWK6(NFH) = IP
454:               if ( MULTI.gt.1 ) WW(IP) = WW(IP) * MULTI
455:             end if

```

```

456:           if ( IWK5(IP).ge.6.and.IWK5(IP).le.9 ) then
457:             N69 = N69 + 1
458:             IWK8(N69) = IP
459:           end if
460:         continue
461: C   .... case of ENDF/B-IV (n,2n) : MT=6-9, 46-49 ....
462:           if ( N69.gt.0 ) then
463:             call RANU2( IRAND, R, N69, ICON )
464: *VOCL LOOP,NOVREC
465:           do 270 I = 1, N69
466:             if ( R(I).gt.0.5 ) then
467:               IWK5(IWK8(I)) = IWK5(IWK8(I)) + 40
468:             end if
469: 270         continue
470:           end if
471: C
472: C   .... case of file6 in ENDF/B-VI .....
473: C
474: C   IWK8 : pointer of incident energy for yield data
475: C   IWK9 : number of incident energy (NEF5)
476: C   WK3 : incident neutron energy
477: C   WK4 : work area in BSDEC3
478: C   IWK10 : energy bin selected.
479: C
480:           if ( NF6.gt.0 ) then
481:             do 280 I = 1, NF6
482:               IP = IWK7(I)
483:               NKF5 = KLIB2(IWK1(IP),IWK5(IP),8)
484:               LSTF5 = KLIB2(IWK1(IP),IWK5(IP),13)
485:               IWK8(I) = LSTF5 + ABS(NKF5)
486:               IWK9(I) = KLIB2(IWK1(IP),IWK5(IP),9)
487:               WK3(I) = EEE(IP,1)
488: 280             continue
489: C
490:             call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NF6 )
491: C
492: *VOCL LOOP,NOVREC
493:           do 290 I = 1, NF6
494:             L2 = IWK8(I) + IWK10(I)
495:             L1 = L2 - 1
496: C   .... LINEAR INTERPOLATION
497:             X1 = (CX(L1)-WK3(I)) / (CX(L1)-CX(L2))
498:             if ( X1.lt.0.0 ) X1 = 0.0
499:             if ( X1.gt.1.0 ) X1 = 1.0
500:             L2 = L2 + IWK9(I)*2
501:             L1 = L2 - 1
502:             YIELD = CX(L1)*(1.0-X1) + CX(L2)*X1
503:             WW(IWK7(I)) = WW(IWK7(I)) * YIELD
504: 290           continue
505:           end if
506: C
507: C-----
508: C   Reordering of IBP
509: C   type elastic,kinematics,file5,fission,file6,Thermal,FREGAS
510: C   number NELS NKIN NF5 NF NF6 NCOLT NCOLF
511: C-----
512: C
513: *VOCL LOOP,NOVREC
514:           do 300 I = 1, NFH
515:             IP = IWK6(I)
516:             NKF5 = KLIB2(IWK1(IP),IWK5(IP),8)
517: C
518: C   .... threshold reaction with kinematics ....
519:           if ( (IWK5(IP).ge.6.and.IWK5(IP).le.9) .or.
520: &           (IWK5(IP).ge.51.and.IWK5(IP).le.90 ) ) then

```

src/mvp/nxtnr2.f

```

521:          NKIN = NKIN + 1
522:          IBP(NELS+NKIN) = IP
523: C
524: C      .... file6 is used and multiplicity is taken from NEUMT ...
525:       else if ( NK5.lt.0 ) then
526:           NF6 = NF6 + 1
527:           IWK7(NF6) = IP
528: C
529: C      .... file5 is used ...
530:       else
531:           NF5 = NF5 + 1
532:           IWK6(NF5) = IP
533:       end if
534: 300 continue
535: C
536: end if
537: C
538: end if
539: C
540: NKINT = NELS + NKIN
541: NF5T = NKINT + NF5
542: NFT = NF5T + NF
543: NF6T = NFT + NF6
544: if ( NF5.ne.0 ) then
545:     do 310 I = 1, NF5
546:         IBP(NKINT+I) = IWK6(I)
547: 310 continue
548:     end if
549:     if ( NF.ne.0 ) then
550: *VOCL LOOP,NOVREC
551:         do 320 I = 1, NF
552:             IBP(NF5T+I) = IWK2(I)
553: C      .... MT for fission set ...
554:             IWK5(IWK2(I)) = 18
555: 320 continue
556:         end if
557:         if ( NF6.ne.0 ) then
558:             do 330 I = 1, NF6
559:                 IBP(NFT+I) = IWK7(I)
560: 330 continue
561:             end if
562: C
563: C-----
564: C.... Classification of thermal reaction and addition to IBP
565: C-----
566: C
567:     NSABT = NF6T
568:     NTEL = 0
569:     NTEL2 = 0
570:     if ( NCOLT.gt.0 ) then
571:         call RANU2( IRAND, R, NCOLT, ICON )
572: C
573: C..... THERMAL INELASTIC CROSS SECTION ARE NOT GIVEN IN SMIC ARRAY
574:       if ( LMIC(6).eq.0 ) then
575: *VOCL LOOP,NOVREC
576:         do 340 I = 1, NCOLT
577:             IP = IWK4(I)
578:             if ( KSPI(IP,IWK1(IP)).ge.KLIB2(IWK1(IP),4,3) ) then
579:                 L2 = KLIB1(IWK1(IP),1) + KSPI(IP,IWK1(IP))
580:                 R1 = (EEE(IP,1)-CX(L2)) / (CX(L2-1)-CX(L2))
581:                 L1 = KLIB2(IWK1(IP),4,11) - KLIB2(IWK1(IP),4,3)
582:                 &      + KSPI(IP,IWK1(IP))
583:                 XIN = CX(L1)*R1 + CX(L1+1)*(1.-R1)
584:             else
585:                 XIN = 0.0

```

```

586:         end if
587:         X2 = (XIN+SMIC(IP,IWK1(IP),LMIC(4)))*R(I)
588:         if ( X2.le.XIN ) then
589:             NSABT = NSABT + 1
590:             IBP(NSABT) = IP
591:             IWK5(IP) = 92
592:         else if ( KLIB2(IWK1(IP),93,7).eq.0 ) then
593:             NTEL2 = NTEL2 + 1
594:             IWK6(NTEL2) = IP
595:         else
596:             NTEL = NTEL + 1
597:             IWK4(NTEL) = IP
598:         end if
599: 340 continue
600: C..... THERMAL INELASTIC CROSS SECTION ARE GIVEN IN SMIC ARRAY
601:       else
602: *VOCL LOOP,NOVREC
603:         do 350 I = 1, NCOLT
604:             IP = IWK4(I)
605:             XIN = SMIC(IP,IWK1(IP),LMIC(6))
606:             X2 = (XIN+ SMIC(IP,IWK1(IP),LMIC(4)))*R(I)
607:             if ( X2.le.XIN ) then
608:                 NSABT = NSABT + 1
609:                 IBP(NSABT) = IP
610:                 IWK5(IP) = 92
611:             else if ( KLIB2(IWK1(IP),93,7).eq.0 ) then
612:                 NTEL2 = NTEL2 + 1
613:                 IWK6(NTEL2) = IP
614:             else
615:                 NTEL = NTEL + 1
616:                 IWK4(NTEL) = IP
617:             end if
618: 350 continue
619:         end if
620:         if ( NTEL.gt.0 ) then
621: *VOCL LOOP,NOVREC
622:             do 360 I = 1, NTEL
623:                 IBP(NSABT+I) = IWK4(I)
624:                 IWK5(IWK4(I)) = 93
625: 360 continue
626:             end if
627:             if ( NTEL2.gt.0 ) then
628: *VOCL LOOP,NOVREC
629:                 do 370 I = 1, NTEL2
630:                     IBP(NSABT+NTEL+I) = IWK6(I)
631:                     IWK5(IWK6(I)) = 93
632: 370 continue
633:                 end if
634:             end if
635: C
636:             NSAB = NSABT - NF6T
637:             NTELT = NSABT + NTEL
638:             NTEL2T= NTELT + NTEL2
639: C
640:             NFRGST = NTEL2T
641:             NWOEC = 0
642:             if ( NCOLF.gt.0 ) then
643: *VOCL LOOP,NOVREC
644:                 do 380 I = 1, NCOLF
645:                     IP = IWK3(I)
646:                     if ( XLIB1(IWK1(IP),1).le.AMLIM ) then
647:                         NFRGST = NFRGST + 1
648:                         IBP(NFRGST) = IP
649:                         IWK5(IP) = -1
650:                     else

```

src/mvp/nxtnr2.f

```

651:          NWOEC = NWOEC + 1
652:          IWK3(NWOEC) = IP
653:          end if
654: 380      continue
655:          if ( NWOEC.gt.0 ) then
656: *VOCL LOOP,NOVREC
657:          do 390 I = 1, NWOEC
658:              IBP(NFRGST+I) = IWK3(I)
659:              IWK5(IWK3(I)) = 0
660: 390      continue
661:          end if
662:          end if
663:          NFRGST = NFRGST - NTEL2T
664:          NWOECT = NFRGST + NWOEC
665: check
666:          if ( NWOECT.ne.NCOLS ) then
667:              write(10,*) ' NXTNR2, NWOECT, NCOLS=',NWOECT,NCOLS
668:              stop
669:          end if
670: C
671:          NRTYP(1) = NELS
672:          NRTYP(2) = NKINT
673:          NRTYP(3) = NF5T
674:          NRTYP(4) = NFT
675:          NRTYP(5) = NF6T
676:          NRTYP(6) = NSABT
677:          NRTYP(7) = NTELT
678:          NRTYP(8) = NTEL2T
679:          NRTYP(9) = NFRGST
680:          NRTYP(10) = NWOECT
681: C
682:          NIGEN = NCOLS
683: C
684: C=====
685: C      Selection of secondary photons
686: C=====
687: C
688:          if ( JPHOT.ne.0 ) then
689: C
690:              N1 = 0
691:              N2 = 0
692:              N3 = 0
693:              N4 = 0
694:              N5 = 0
695:              N6 = 0
696:              N7 = 0
697:              N8 = 0
698:              N9 = 0
699: C
700:              NN = 0
701:              NP = 0
702:              BANKNI = 1. / (1.-BANKP)
703: C
704:              call RANU2( IRAND, R, NCOLS, ICON )
705: C
706: *VOCL LOOP,NOVREC
707:          do 400 I = 1, NCOLS
708:              IP = IBP(I)
709: C          .... gamma ....
710:              if ( R(I).le.BANKP ) then
711:                  NP = NP + 1
712:                  IWK2(NP) = IP
713: C          .... neutron ....
714:              else
715:                  NN = NN + 1

```

```

716:          IBP(NN) = IP
717:          WW(IP) = WW(IP) * BANKNI
718:          if ( I.le.NELS ) then
719:              N1 = N1 + 1
720:          else if ( I.le.NKINT ) then
721:              N2 = N2 + 1
722:          else if ( I.le.NF5T ) then
723:              N3 = N3 + 1
724:          else if ( I.le.NFT ) then
725:              N4 = N4 + 1
726:          else if ( I.le.NF6T ) then
727:              N5 = N5 + 1
728:          else if ( I.le.NSABT ) then
729:              N6 = N6 + 1
730:          else if ( I.le.NTELT ) then
731:              N7 = N7 + 1
732:          else if ( I.le.NTEL2T ) then
733:              N8 = N8 + 1
734:          else if ( I.le.NFRGST ) then
735:              N9 = N9 + 1
736:          end if
737:          end if
738: 400      continue
739: C
740:          NRTYP(1) = N1
741:          NRTYP(2) = NRTYP(1) + N2
742:          NRTYP(3) = NRTYP(2) + N3
743:          NRTYP(4) = NRTYP(3) + N4
744:          NRTYP(5) = NRTYP(4) + N5
745:          NRTYP(6) = NRTYP(5) + N6
746:          NRTYP(7) = NRTYP(6) + N7
747:          NRTYP(8) = NRTYP(7) + N8
748:          NRTYP(9) = NRTYP(8) + N9
749:          NRTYP(10) = NN
750: C
751:          if ( NP.gt.0 ) then
752:              NPC = 0
753: *VOCL LOOP,NOVREC
754:          do 410 I = 1, NP
755:              IMTP = IWK1(IWK2(I))
756: C          .... capture gamma ....
757:              if( IMTP.eq.0 ) then
758:                  NPC = NPC + 1
759:                  IWK3(NPC) = IWK2(I)
760:                  IWK5(IWK2(I)) = 0
761:              end if
762: C          .... fission gamma ....
763:              if ( IMTP.lt.0 ) then
764:                  if ( KLIB2(IWK1(IWK2(I)),18,15).gt.0 ) IMTP = -IMTP
765:                  IWK5(IWK2(I)) = IMTP
766:              end if
767: 410      continue
768: C
769:          if ( NPC.gt.0 ) then
770:              IDET = 0
771: C
772: C          ..... IWK4 : Cross section pointer ....
773: C          ..... WK1(1:IMTTP) : Cross section summation ....
774: C          ..... WK2(IMTTP+1:2*IMTTP) : Total capture cross section ....
775: C
776: C
777:          call RANU2( IRAND, R, NPC, ICON )
778: C
779:          MXCAPG = 0
780:          do 420 I = 1, NPC

```

src/mvp/nxtnr2.f

```

781:      IP      = IWK3(I)
782: C
783: C      ..... KSPI: Energy point #
784: C
785:      IWK6(I) = KSPI(IP,IWK1(IP))
786:      IWK4(I) = KLIB1(IWK1(IP),1) + IWK6(I)
787: C
788: C      ..... NCAPG ....
789: C
790:      IWK7(I) = KLIB1(IWK1(IP),27)
791:      MXCAPG = MAX(IWK7(I),MXCAPG)
792:      WK1(I)  = 0.0
793:      WK2(I)  = SMIC(IP,IWK1(IP),LMIC(5)) * R(I)
794: 420      continue
795: C
796:      do 440 MMT = 1, MXCAPG
797:      if ( IDET.eq.NPC ) go to 450
798: C
799: *VOCL LOOP,NOVREC
800:      do 430 I = 1, NPC
801:      IP      = IWK3(I)
802:      if ( MMT.le.IWK7(I).and.IWK5(IP).eq.0 ) then
803: C
804: C      .... MTCAPG -> MT ....
805: C
806:      MT      = KLIB2(IWK1(IP),MMT,21)
807: C
808: C      .... Cross section for MT ....
809: C
810:      E1      = CX(IWK4(I))
811:      TI      = (EEE(IP,1)-E1) / (CX(IWK4(I))-E1)
812:      L1      = KLIB2(IWK1(IP),MT,11)
813:      &      - KLIB2(IWK1(IP),MT,3) + IWK6(I)
814: C
815: C      ..... SC: Capture cross section ...
816: C
817:      if ( IWK6(I).ge.KLIB2(IWK1(IP),MT,3)
818:      &      .and.IWK6(I).lt.KLIB2(IWK1(IP),MT,4) ) then
819:      SC      = CX(L1)*TI + CX(L1+1)*(1.0-TI)
820:      else
821:      SC      = 0.0
822:      end if
823: C
824: C      .... Check
825: C
826:      WK1(I)  = WK1(I) + SC
827: C
828: C      (Partial sum of cap. xsec)/(Total cap. x-sec) > Random.n
829: C      ----> MT determined
830: C
831:      if ( WK2(I).le.WK1(I) ) then
832:      IWK5(IP) = MT
833:      IDET     = IDET + 1
834:      end if
835:      end if
836: 430      continue
837: 440      continue
838: C
839: 450      if ( IDET.lt.NPC ) then
840: *VOCL LOOP,NOVREC
841:      do 460 I = 1, NPC
842:      IP      = IWK3(I)
843: C      .... MTCAPG(NCAPG) -> MT IF MTTOG(MTCAPG)=3 ...
844:      if ( IWK5(IP).eq.0.and.IWK7(I).ne.0 ) then
845:      MTCAPG  = KLIB2(IWK1(IP),IWK7(I),21)

```

```

846:      if ( KLIB2(IWK1(IP),MTCAPG,15).eq.3 ) then
847:      IWK5(IP) = MTCAPG
848:      end if
849:      end if
850: 460      continue
851:      end if
852:      end if
853: C
854:      NPP     = 0
855: *VOCL LOOP,NOVREC
856:      do 470 I = 1, NP
857:      IP      = IWK2(I)
858:      IMTP    = ABS( IWK5(IP) )
859:      MTTOG   = KLIB2(IWK1(IP),IMTP,15)
860:      if ( IWK5(IP).gt.0.and.MTTOG.gt.0 ) then
861:      NPP     = NPP + 1
862:      IWK2(NPP) = IP
863:      IWK3(NPP) = MTTOG
864:      end if
865: 470      continue
866: *VOCL LOOP,NOVREC
867:      do 480 I = 1, NPP
868:      IP      = IWK2(I)
869:      MTTOG   = IWK3(I)
870:      MT3     = MTTOG/1000000
871:      MTTOG   = MTTOG - 1000000*MT3
872: C
873:      MT1     = MTTOG/1000
874:      MT2     = MTTOG - 1000*MT1
875: C
876:      WK1(I)  = EEE(IP,1)
877: C
878: C      ..... for JENDL 3.2
879:      if ( MT3.ne.0 ) then
880:      if ( WK1(I).le.XLIB2(IWK1(IP),IWK5(IP),4) ) then
881:      IWK5(IP) = MT2
882:      else if ( WK1(I).ge.XLIB2(IWK1(IP),IWK5(IP),3) ) then
883:      IWK5(IP) = MT1
884:      else
885:      IWK5(IP) = MT3
886:      end if
887:      else if ( MT1.ne.0 ) then
888: C
889: C      .... E <= EG1L , WHICH IS CHANGED TO E <= EG2U
890: C      EG1L=XLIB2(...3), EG2U=XLIB2(...4)
891:      if ( WK1(I).le.XLIB2(IWK1(IP),IWK5(IP),4) ) then
892:      IWK5(IP) = MT2
893:      else
894:      IWK5(IP) = MT1
895:      end if
896:      else
897:      IWK5(IP) = MT2
898:      end if
899: C      .... ENERGY MESH POINTER & ENERGY TABLE LENGTH
900: C      IWK6(I) POINTS EMESHG(1) FOR MT=IWK5(IP)
901: C
902:      IWK6(I) = KLIB2(IWK1(IP),IWK5(IP),14) + 1
903:      IWK7(I) = ICX(IWK6(I)-1)
904: 480      continue
905: C-----
906: C      TABLE SEARCH: WK(I) -> ( CX(IWK6(I)+L-1),L=1,IWK7(I) ) -> IWK8(I)
907: C
908: C      IWK9 : WORKING AREA ONLY FOR TABLE SEARCH
909: C      IWK8 : TABLE POSITION AS RESULT ( 1 <= IWK8 <= IWK7-1 )
910: C-----

```

src/mvp/nxtnr2.f

```
911: C
912:      call BSDEC3( CX, IWK7, IWK9, IWK6, WK1, IWK8, NPP )
913: C
914:      NP      = 0
915:      BANKPI = 1. / BANKP
916: *VOCL LOOP,NOVREC
917:      do 490 I = 1, NPP
918:          IP      = IWK2(I)
919:          IPE      = IWK6(I) + IWK8(I)
920:          IPY      = IPE + IWK7(I)
921:          if ( CX(IPE-1).ge.WK1(I).and.CX(IPE).le.WK1(I) ) then
922:              WW(IP) =
923:              &      (CX(IPY)+(CX(IPE)-WK1(I))/(CX(IPE)-CX(IPE-1))*
924:              &      (CX(IPY-1)-CX(IPY))) * WWW(IP,1) * BANKPI
925:          else
926:              WW(IP) = 0.0
927:          end if
928: C
929:          if ( WW(IP).gt.0. ) then
930:              NP      = NP + 1
931:              IWK2(NP) = IP
932:              IWK9(IP) = IWK6(I) + IWK7(I)*2 + 1
933:          end if
934:      490      continue
935: C
936: *VOCL LOOP,NOVREC
937:      do 500 I = 1, NP
938:          IBP(NN+I) = IWK2(I)
939:      500      continue
940: C
941:      end if
942: C
943:      NIGEN = NN + NP
944: C
945:      end if
946: C
947: C=====
948: C
949: C**** gather particle data, and set pointer for cross sections
950: C
951: C=====
952: C
953:      do 600 I = 1, NIGEN
954:          ICLN(I) = IWK1(IBP(I))
955:          IMT(I)  = IWK5(IBP(I))
956:          W(I)    = WW(IBP(I))
957:          EIN(I)  = EEE(IBP(I),1)
958:      600      continue
959: C
960: C .... photon production ILF : position of LSTF5G-1
961:      if ( NIGEN.gt.NRTYP(10) ) then
962:          do 610 I = NRTYP(10)+1, NIGEN
963:              ILF(I) = IWK9(IBP(I))
964:          610      continue
965:      end if
966: C
967:      return
968:      end
```


src/mvp/nxtnr3.f

```

1:      subroutine NXTNR3(   IOW, A, H,
2:      X      CX      , ICX  , KLIB1, XLIB1, KLIB2, XLIB2,
3:      B      NIGEN, A0    , B0   , C0   ,
4:      B      IBP   , ICLN , IMT   , ISF4 , ISF5 , ILF   ,
5:      B      EIN   , W    , THEA  , THEB , EMIU , NRTYP,
6:      W      IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
7:      W      IWK7 , IWK8 , IWK9 , IWK10,
8:      W      WK1  , WK2  , WK3  , WK4  , WW   ,
9:      W      R     )
10: C=====
11: C
12: C  PURPOSE: Set parameters for next event estimator
13: C  CALLED IN: NXTNR
14: C  CALLS: RANU2,BSDEC3, BSINC3
15: C
16: C=====
17: C
18: C      IBP(I)   : bank pointer
19: C      ICLN(I)  : collision nuclide
20: C      IMT(I)   : neutron emitting reaction
21: C
22: C      EIN(I)   : incident energy
23: C      ISF4(I)  : pointer to equiprobable cosine bins for reaction
24: C                  given by files 4&5 and thermal elastic.
25: C                  for thermal inelastic, pointer to energy meshes
26: C                  or to another energy-angle distribution
27: C                  (LF=67,INT>10)
28: C      ISF5(I)  : pointer to energy(-angle) distribution
29: C      ILF(I)   : LF value for energy(-angle) distribution,
30: C                  for thermal inelastic, incident energy mesh
31: C      THEA(I)  : theta for LF=5,7,9,11
32: C                  maximum energy for LF=1,6,61 (INT>10)
33: C                  interpolation ratio for LF=67
34: C      THEB(I)  : theta2 for LF=11
35: C                  minimum energy for LF=1,6,61 (INT>10)
36: C      EMIU(I)  : target mass for reactions with use of kinematics
37: C                  (effective target mass for moving target )
38: C                  EIN-U for LF=5,7,9,11
39: C      A0(I),B0(I),C0(I) : flight direction
40: C                  (effective direction for moving target )
41: C      NRTYP(10) : cumulative number of particles
42: C          1 : elastic scattering
43: C          2 : reaction with 2-body kinematics
44: C          3 : reaction with files 4 & 5
45: C          4 : fission
46: C          5 : reaction with file 6
47: C          6 : thermal inelastic
48: C          7 : incoherent thermal elastic
49: C          8 : coherent thermal elastic
50: C          9 : free gas
51: C         10 : isotropic without energy change
52: C
53: C=====
54: C      implicit real*8 (A-H,O-Z)
55: C
56: C      real A(*)
57: C      real H(*)
58: C
59: C      include '../shared/INC/_SIZES'
60: C      include 'INC/_SIZES2'
61: C      include 'INC/_FLAGS'
62: C      include 'INC/_STACK'
63: C      include 'INC/_KPIDS'
64: C      include 'INC/_CXSEC'
65: C      include 'INC/_PXSEC'

```

```

66:      include 'INC/_SBANK'
67:      include '../shared/INC/_PTALY0'
68:      include 'INC/_PTALY'
69:      include 'INC/_PTALY2'
70: C
71: C
72: C**** CROSS SECTION DATA
73: C
74: C      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
75: C      c##<2007/03/14:PN3:
76: C      integer ICX(MCX), KLIB1(NUC,27), KLIB2(NUC,NMT,21)
77: C      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
78: C      c##>
79: C
80: C**** TALLY (MONITOR) ARRAY
81: C
82: C      real*8   WCNTR(NEVENT,2), NCNTR(NEVENT,2)
83: C
84: C      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
85: C      &      IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
86: C      &      IWK9(NBANK), IWK10(NBANK)
87: C      real    R(7*NBANK)
88: C      real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK)
89: C
90: C**** Prepared stack and bank
91: C
92: C      integer IBP(NBANK), ICLN(NBANK), IMT(NBANK), ISF4(NBANK),
93: C      &      ISF5(NBANK), ILF(NBANK)
94: C      integer NRTYP(10)
95: C      real    W(NBANK), WW(NBANK), EIN(NBANK), THEA(NBANK), THEB(NBANK),
96: C      &      EMIU(NBANK)
97: C      real*8   A0(NBANK), B0(NBANK), C0(NBANK)
98: C
99: C****
100: C
101: C      parameter( PI   = 3.1415926535897932D+00 )
102: C      parameter( PI2  = 2.0D0*PI )
103: C      parameter( PIH  = 0.5D0*PI )
104: C      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
105: C
106: C
107: C
108: C=====
109: C      if ( JVMNT.ne.0 ) call VMNTRI( 5, NIGEN )
110: C
111: C=====
112: C
113: C      NELS      = NRTYP(1)
114: C      NKINT     = NRTYP(2)
115: C      NF5T      = NRTYP(3)
116: C      NFT       = NRTYP(4)
117: C      NF6T      = NRTYP(5)
118: C      NSABT     = NRTYP(6)
119: C      NTELT     = NRTYP(7)
120: C      NTEL2T    = NRTYP(8)
121: C      NFRGST    = NRTYP(9)
122: C      NWOECT    = NRTYP(10)
123: C
124: C=====
125: C      for moving target (freegas), calculate effective values
126: C=====
127: C
128: C      NF = 0
129: C      if ( NKINT.gt.0 ) then
130: C          do 100 I = 1, NKINT

```

src/mvp/nxtnr3.f

```

131:      EMIU(I) = XLIB1(ICLN(I),1)
132:      if ( EMIU(I).lt.1.5 ) then
133:        NF = NF + 1
134:        IWK10(NF) = I
135:      end if
136: 100    continue
137:  end if
138: C
139:      if ( NFRGST.gt.NTEL2T ) then
140:        do 110 I = NTEL2T+1, NFRGST
141:          EMIU(I) = XLIB1(ICLN(I),1)
142:          NF = NF + 1
143:          IWK10(NF) = I
144: 110    continue
145:        end if
146: C
147:        if ( NF.gt.0 ) then
148:          call NXTMVT( IOW, NF, NBANK, IWK10,
149:            &      EIN, A0, B0, C0, EMIU,
150:            &      ISF5, ISF4, ILF, ICLN, IMT,
151:            &      CX, CX, MCX, KLIB1, XLIB1, XLIB2,
152:            &      NUC, NMT,
153:            &      IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
154:            &      IWK7, IWK8, IWK9,
155:            &      R, IRAND )
156: C *:: NXTMVT uses 7 R
157: C
158:        end if
159: C
160: C=====
161: C
162: C**** set pointer for cross sections
163: C
164: C=====
165: C
166: C .... used equi-probable cosine bins ...
167:      do 420 I = 1, NF5T
168:        IWK1(I) = KLIB2(ICLN(I),IMT(I),7)
169:        IWK2(I) = KLIB2(ICLN(I),IMT(I),12)
170: 420    continue
171:      call BSDEC3( CX, IWK1, WK3, IWK2, EIN, IWK3, NF5T )
172:      call RANU2( IRAND, R, NF5T, ICON )
173:      do 430 I = 1, NF5T
174:        if ( IWK3(I).eq.0 ) IWK3(I) = 1
175:        L2 = IWK2(I) + IWK3(I)
176:        L1 = L2 - 1
177:        INTE = ICX(L2+IWK1(I))
178:        if ( INTE.eq.5 .or. INTE.eq.3 ) then
179: C..... LOG INTERPOLATION
180:          X1 = LOG(CX(L1)/EIN(I)) /LOG(CX(L1)/CX(L2))
181:        else
182: C..... LINEAR INTERPOLATION
183:          X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
184:        end if
185:        if ( X1.lt.0.0 ) X1 = 0.0
186:        if ( X1.gt.1.0 ) X1 = 1.0
187:
188: C/#IF ROUNDOFF(NEAREST)
189:      IT = min(1,INT(R(I)+X1)) + IWK3(I)
190: C/#ELSE
191:      *      R1 = R(I) + X1
192:      *      IT = IWK3(I) + R1
193: C/#ENDIF
194:      ISF4(I) =
195:      &      IWK2(I) + IWK1(I)*2 + KLIB1(ICLN(I),20)*(IT-1)

```

```

196: 430 continue
197: C
198: C .... used energy or angle-energy distribution ...
199: C .... and equi-probable cosine bins for thermal elastic
200:      if ( NF6T.gt.NKINT ) then
201:        NN = 0
202:        N6 = 0
203:        do 440 I = NKINT+1,NF6T
204:          NKF5 = KLIB2(ICLN(I),IMT(I),8)
205:          IWK4(I) = ABS(NKF5)
206:          IWK5(I) = KLIB2(ICLN(I),IMT(I),13)
207:          if ( IWK4(I).ge.2 ) then
208:            NN = NN + 1
209:            IWK6(NN) = I
210:            IWK1(NN) = KLIB2(ICLN(I),IMT(I),9)
211:            IWK2(NN) = IWK5(I) + IWK4(I)
212:            WK2(NN) = EIN(I)
213:          end if
214:          if ( NKF5.le.-2 ) N6 = N6 + 1
215: 440    continue
216:      check
217:      if ( N6.gt.0 ) then
218:        write(IOW,*) ' I found some nuclides with multiple',
219:          &      ' subsections for file 6 data. stop !!!'
220:      stop
221:      end if
222: C
223:      call BSDEC3( CX, IWK1, WK3, IWK2, WK2, IWK3, NN )
224:      call RANU2( IRAND, R, NN*3, ICON )
225:      NN2 = NN*2
226: C ***** selection of subsection for file 5 data *****
227: C ***** assuming 1 subsection for file 6 data *****
228: *VOCL LOOP,NOVREC
229:      do 450 J = 1, NN
230:        I = IWK6(J)
231:        if ( IWK3(J).eq.0 ) IWK3(J) = 1
232:        L2 = IWK2(J) + IWK3(J)
233:        L1 = L2 - 1
234:        INTE = ICX(L2+IWK1(J))
235: C      if ( INTE.eq.5 .or. INTE.eq.3 ) then
236: C ..... LOG INTERPOLATION
237: C      X1 = LOG(CX(L1)/WK2(J)) /LOG(CX(L1)/CX(L2))
238: C      else
239: C ..... LINEAR INTERPOLATION
240: C      X1 = (CX(L1)-WK2(J)) / (CX(L1)-CX(L2))
241: C      end if
242: C      if ( X1.lt.0.0 ) X1 = 0.0
243: C      if ( X1.gt.1.0 ) X1 = 1.0
244:
245: C/#IF ROUNDOFF(NEAREST)
246:      IT = min(1,INT(R(J)+X1)) + IWK3(J)
247: C/#ELSE
248:      *      R1 = R(J) + X1
249:      *      IT = IWK3(J) + R1
250: C/#ENDIF
251:
252: C..... BMC SAMPLING
253:      NKF5 = IWK4(I)
254:      L1 = IWK2(J) + IWK1(J)*2 + 3*NKF5*(IT-1) - 1
255:
256: C/#IF ROUNDOFF(NEAREST)
257:      L2 = MIN( INT(NKF5*R(NN+J)) + 1, NKF5 )
258: C/#ELSE
259:      *      L2 = NKF5*R(NN+J) + 1
260: C/#ENDIF

```

src/mvp/nxtnr3.f

```

261:          L3      = L1 + L2
262:
263: C/#IF ROUNDOFF(NEAREST)
264:          L4      = 2*L2
265:          L4      = MIN(L4,INT(L4+R(NN2+J)-CX(L3))) + L1 + NK5
266: C/#ELSE
267:          *        R2      = R(NN2+J) - CX(L3)
268:          *        L4      = L1 + NK5 + 2*L2 + R2
269: C/#ENDIF
270:
271:          IWK4(I)   = ICX(L4)
272: 450 continue
273: C
274: C          IWK4 : subsection selected
275: C          IWK5 : LSTF5 ----> LSTF5S
276: C
277: do 460 I = NKINT+1,NF6T
278:   IWK5(I) = ICX(IWK5(I)+IWK4(I)-1)
279:   IWK2(I) = IWK5(I) + 2
280:   IWK1(I) = ICX(IWK5(I)+1)
281:   ILF(I)  = ICX(IWK5(I))
282: 460 continue
283:   NN = NF6T - NKINT
284:   call BSDEC3( CX, IWK1(NKINT+1), WK3, IWK2(NKINT+1),
285: &            EIN(NKINT+1), IWK3(NKINT+1), NN )
286:   call RANU2( IRAND, R(NKINT+1), NN, ICON )
287: C
288: do 470 I = NKINT+1,NF6T
289:   if ( IWK3(I).eq.0 ) IWK3(I) = 1
290:   L2 = IWK2(I) + IWK3(I)
291:   L1 = L2 - 1
292:   INTE0 = ICX(L2+IWK1(I))
293:   INTE1 = INTE0 / 10
294:   INTE = INTE0 - 10*INTE1
295:   if ( INTE.eq.5 .or. INTE.eq.3 ) then
296: C ..... LOG INTERPOLATION
297:          X1 = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)-CX(L2))
298:   else
299: C ..... LINEAR INTERPOLATION
300:          X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
301:   end if
302:   if ( X1.lt.0.0 ) X1 = 0.0
303:   if ( X1.gt.1.0 ) X1 = 1.0
304:
305:   L1 = IWK2(I) + IWK1(I)*2
306:   if ( ILF(I).eq.1.or.ILF(I).eq.61.or.ILF(I).eq.67 ) then
307: C/#IF ROUNDOFF(NEAREST)
308:          IWK6(I) = min(1,INT(R(I)+X1)) + IWK3(I)
309: C/#ELSE
310:          *        R1      = R(I) + X1
311:          *        IWK6(I) = IWK3(I) + R1
312: C/#ENDIF
313:          ISF5(I) = ICX(L1+IWK6(I)-1)
314:          THEA(I) = -1.0
315:   else if ( ILF(I).eq.5 ) then
316:          L5 = L1 + IWK3(I)
317:          THEA(I) = CX(L5)*X1 + CX(L5-1)*(1.-X1)
318:          ISF5(I) = L1 + IWK1(I)
319:          EMIU(I) = EIN(I) - CX(ISF5(I)+ICX(ISF5(I))*5+3)
320:   else if ( ILF(I).eq.7 ) then
321:          L7 = L1 + IWK3(I)
322:          THEA(I) = CX(L7)*X1 + CX(L7-1)*(1.-X1)
323:          EMIU(I) = EIN(I) - CX(L1+IWK1(I))
324:   else if ( ILF(I).eq.9 ) then
325:          L9 = L1 + IWK3(I)

```

```

326:          THEA(I) = CX(L9)*X1 + CX(L9-1)*(1.-X1)
327:          EMIU(I) = EIN(I) - CX(L1+IWK1(I))
328:   else if ( ILF(I).eq.11 ) then
329:          L10 = L1 + IWK3(I)
330:          L11 = L10 + IWK1(I)
331:          THEA(I) = CX(L10)*X1 + CX(L10-1)*(1.-X1)
332:          THEB(I) = CX(L11)*X1 + CX(L11-1)*(1.-X1)
333:          EMIU(I) = EIN(I) - CX(L1+IWK1(I)*2)
334:   else if ( ILF(I).eq.66 ) then
335:          ISF5(I) = L1
336:   end if
337: C
338: C
339: if ( INTE1.gt.0.and.(ILF(I).eq.1.or.ILF(I).eq.61) ) then
340:   L2 = L1 + IWK3(I)
341:   LSTF51 = ICX(L2-1)
342:   LSTF52 = ICX(L2)
343:   NEP1 = ICX(LSTF51)
344:   NEP2 = ICX(LSTF52)
345:   if ( ILF(I).eq.61 ) then
346:     LSTF51 = LSTF51 + 1
347:     LSTF52 = LSTF52 + 1
348:   end if
349:   LLE1 = LSTF51 + 3*NEP1 + 1
350:   LLE2 = LSTF52 + 3*NEP2 + 1
351:   EMAX1 = CX(LLE1)
352:   THEA(I) = EMAX1 + (CX(LLE2)-EMAX1)*X1
353:   THEB(I) = 0.0
354:   if ( INTE1.eq.1 ) then
355:     ND1 = ICX(LSTF51)
356:     ND2 = ICX(LSTF52)
357:     EMIN1 = CX(LLE1+ND1)
358:     THEB(I) = EMIN1 + (CX(LLE2+ND2)-EMIN1)*X1
359:   end if
360: end if
361: if ( INTE1.gt.0.and.ILF(I).eq.67 ) then
362:   THEB(I) = 0.0
363:   if ( INTE1.eq.1 ) THEB(I) = 1.0
364:   if ( IWK3(I).eq.IWK6(I) ) then
365:     ISF4(I) = ICX(L1+IWK6(I))
366:     THEA(I) = X1
367:   else
368:     ISF4(I) = ICX(L1+IWK6(I)-2)
369:     THEA(I) = 1. - X1
370:   end if
371: end if
372: 470 continue
373: C .... thermal scattering
374:   if ( NTEL2T.gt.NF6T ) then
375:     if ( NTELT.gt.NF6T ) then
376:       do 480 I = NF6T+1, NTELT
377:         if ( I.le.NSABT ) then
378:           IWK2(I) = KLIB2(ICLN(I),92,13) + 3
379:           IWK1(I) = ICX(IWK2(I)-1)
380:         else
381:           IWK2(I) = KLIB2(ICLN(I),93,12)
382:           IWK1(I) = KLIB2(ICLN(I),93,7)
383:         end if
384:       continue
385:     end if
386:   if ( NTEL2T.gt.NTELT ) then
387:     do 490 I = NTELT+1, NTEL2T
388:       IWK2(I) = ICX(KLIB2(ICLN(I),93,13)) + 2
389:       IWK1(I) = ICX(IWK2(I)-1)
390:     continue

```

src/mvp/nxtnr3.f

```

391:         end if
392: C
393:         NN = NTEL2T - NF6T
394: C
395:         call BSINC3( CX, IWK1(NF6T+1), WK3, IWK2(NF6T+1), EIN(NF6T+1),
396: &                 IWK3(NF6T+1), NN )
397: C
398:         if ( NTEL2T.gt.NF6T ) then
399:             NN = NTEL2T - NF6T
400:             call RANU2( IRAND, R(NF6T+1), NN, ICON )
401: C
402:             do 500 I = NF6T+1, NTEL2T
403:                 if ( IWK3(I).le.0 ) IWK3(I) = 1
404:                 L2 = IWK2(I) + IWK3(I)
405:                 L1 = L2 - 1
406:                 INTE = ICX(L2+IWK1(I))
407:                 if ( INTE.eq.5 .or. INTE.eq.3 ) then
408: C ..... LOG INTERPOLATION
409:                     X1 = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
410:                 else
411: C ..... LINEAR INTERPOLATION
412:                     X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
413:                 end if
414: C/#IF ROUND OFF(NEAREST)
415:                 IT = min(1,INT(R(I)+X1)) + IWK3(I)
416: C/#ELSE
417:                 R1 = R(I) + X1
418:                 IT = IWK3(I) + R1
419: C/#ENDIF
420:                 if ( I.le.NSABT ) then
421:                     ISF5(I) = IWK2(I) + 3*IWK1(I)*IT - IWK1(I)
422:                     ISF4(I) = IWK2(I)
423:                     ILF(I) = IT
424:                 else
425:                     ILF(I) = KLIB1(ICLN(I),20)
426:                     ISF4(I) =
427: &                 IWK2(I) + IWK1(I)*2 + ILF(I)*(IT-1)
428:                 end if
429:
430:             500 continue
431:         end if
432:         if ( NTEL2T.gt.NTEL2T ) then
433:             NN = NTEL2T - NTEL2T
434:             call RANU2( IRAND, R, NN*2, ICON )
435: C
436:             do 510 I = NTEL2T+1, NTEL2T
437:                 if ( EIN(I).gt.CX(IWK2(I)+IWK1(I)-1) ) then
438:                     IWK3(I) = IWK1(I)
439:                 end if
440:                 if ( IWK3(I).eq.0 ) IWK3(I) = 1
441:                 LSBRAG = ICX(IWK2(I)+2*IWK1(I)+IWK3(I)-1)
442:                 NBIN = ICX(LSBRAG)
443: C
444:                 J = I - NTEL2T
445: C/#IF ROUND OFF(NEAREST)
446:                 LL2 = MIN(INT(NBIN*R(J))+1,NBIN)
447: C/#ELSE
448:                 LL2 = NBIN*R(J) + 1
449: C/#ENDIF
450:
451:                 LL3 = LSBRAG + LL2
452:
453: C/#IF ROUND OFF(NEAREST)
454:                 LL4 = 2*LL2
455:                 LL4 = MIN(LL4,INT(LL4+R(NN+J)-CX(LL3))) + LSBRAG +

```

```

456: &                                     NBIN
457: C/#ELSE
458: *                                     R2 = R(NN+J) - CX(LL3)
459: *                                     LL4 = LSBRAG + NBIN + 2*LL2 + R2
460: C/#ENDIF
461:                                     L3 = ICX(LL4)
462: C
463:                                     THEA(I) = 1.0 - 2.0 * CX(IWK2(I)+L3-1) / EIN(I)
464:                                     ILF(I) = NBIN
465:                                     ISF4(I) = LSBRAG
466:                                     ISF5(I) = IWK2(I)
467: 510 continue
468:         end if
469:     end if
470: C
471: C .... photon production .....
472: C
473:         if ( NIGEN.gt.NWOECT ) then
474:             MNKG = 0
475:             N1 = NWOECT+1
476:             do 520 I = N1, NIGEN
477:                 IWK1(I) = KLIB2(ICLN(I),IMT(I),17)
478:                 IWK4(I) = KLIB2(ICLN(I),IMT(I),18)
479:                 IWK2(I) = ILF(I) + 2*IWK4(I) + 1
480:                 MNKG = MAX(MNKG,IWK4(I))
481:             520 continue
482: C
483:             N2 = NIGEN - NWOECT
484: C
485:             call BSDEC3( CX, IWK1(N1), WK3, IWK2(N1), EIN(N1), IWK3(N1),
486: &                     N2 )
487: C
488:             call RANU2( IRAND, R(N1), N2, ICON )
489: C
490:             IDET = 0
491: *VOCL LOOP,NOVREC
492:             do 530 I = N1, NIGEN
493:                 IPE = IWK2(I) + IWK3(I)
494:                 IWK6(I) = IPE + 2*IWK1(I)
495:                 IPY = IWK6(I)
496:                 INTF5G = ICX(IPE+IWK1(I))
497:                 if ( INTF5G.eq.2 ) then
498:                     WK1(I) = (CX(IPE)-EIN(I)) / (CX(IPE)-CX(IPE-1))
499:                     R(I) = R(I)*(CX(IPY)+WK1(I)*(CX(IPY-1)-CX(IPY)))
500:                 else
501:                     XC = LOG(CX(IPE))
502:                     WK1(I) = (XC-LOG(EIN(I))) / (XC-LOG(CX(IPE-1)))
503:                     R(I) = R(I)*(CX(IPY)+WK1(I)*(CX(IPY-1)-CX(IPY)))
504:                 end if
505:                 if ( IWK4(I).eq.1 ) then
506:                     IWK5(I) = 1
507:                     IDET = IDET + 1
508:                 else
509:                     IWK5(I) = 0
510:                 end if
511:             530 continue
512: C
513: C ..... determined subsection number --> IWK5(I)
514: C
515:             do 550 M = 2, MNKG
516:                 if ( IDET.eq.N2 ) go to 560
517: C
518:             do 540 I = N1, NIGEN
519:                 if ( IWK4(I).ge.M.and.IWK5(I).eq.0 ) then
520: C

```

src/mvp/nxtnr3.f

```

521:          IPY      = IWK6(I) + IWK1(I)*(M-1)
522: C
523:          CY      = CX(IPY) + WK1(I)*(CX(IPY-1)-CX(IPY))
524: C
525:          if ( R(I).gt.CY ) then
526:              IDET  = IDET + 1
527:              IWK5(I) = M - 1
528:          end if
529:      end if
530: 540      continue
531: 550      continue
532: C
533: 560      continue
534:      if ( IDET.lt.N2 ) then
535:          do 570 I = N1, NIGEN
536:              if ( IWK5(I).eq.0 ) IWK5(I) = IWK4(I)
537: 570          continue
538:      end if
539: C
540:      NLF6 = 0
541: *VOCL LOOP,NOVREC
542:      do 580 I = N1, NIGEN
543:          LST = ICX(ILF(I)+IWK5(I))
544:          ILF(I) = ICX(LST)
545:          if ( ILF(I).eq.2 ) then
546:              THEA(I) = CX(LST+2) + CX(LST+3)*EIN(I)
547:              LST2 = LST + 4
548:              IWK7(I) = ICX(LST2)
549:              IWK6(I) = ICX(LST2+1)
550: C      .... assuming LF = 6 or LF = 61 !
551:          else
552:              NLF6 = NLF6 + 1
553:              IWK4(NLF6) = I
554:              NEF5S = ICX(LST+1)
555:              IWK1(NLF6) = NEF5S
556:              IWK2(NLF6) = LST + 2
557:              WK1(NLF6) = EIN(I)
558: C      NBINE1 = KLIB1(ICLN(I),21)
559: C      if ( NBINE1.gt.1 ) then
560: C      .... Old MVP library, equiprobable energy bin
561: C      LST3 = LST + 2 + 2*(1+NBINE1)*NEF5S
562: C      IWK7(I) = ICX(LST3)
563: C      IWK6(I) = ICX(LST3+1)
564: C      else
565: C      LSTGL = ICX(LST+1+3*NEF5S)
566: C      NEP = ICX(LSTGL)
567: C      IWK3(I) = LSTGL + 3 + NEP*5
568: C      if ( ILF(I).eq.61 ) IWK3(I) = IWK3(I) + NEP*2 + 3
569: C      IWK7(I) = ICX(IWK3(I))
570: C      IWK6(I) = ICX(IWK3(I)+1)
571: C      end if
572:      end if
573: 580      continue
574: C
575:      call BSDEC3( CX, IWK1, WK3, IWK2, WK1, IWK3, NLF6 )
576:      call RANU2( IRAND, R, NLF6, ICON )
577: C
578: *VOCL LOOP,NOVREC
579:      do 590 J = 1, NLF6
580:          if ( IWK3(J).eq.0 ) IWK3(J) = 1
581:          L2 = IWK2(J) + IWK3(J)
582:          L1 = L2 - 1
583:          INTE0 = ICX(L2+IWK1(J))
584:          INTE1 = INTE0 / 10
585:          INTE = INTE0 - 10*INTE1

```

```

586:          if ( INTE.eq.5 .or. INTE.eq.3 ) then
587: C      .... LOG INTERPOLATION
588:          X1 = LOG(CX(L1)/WK1(J)) / LOG(CX(L1)/CX(L2))
589:          else
590: C      .... LINEAR INTERPOLATION
591:          X1 = (CX(L1)-WK1(J)) / (CX(L1)-CX(L2))
592:          end if
593:          if ( X1.lt.0.0 ) X1 = 0.0
594:          if ( X1.gt.1.0 ) X1 = 1.0
595:
596:          L1 = IWK2(J) + IWK1(J)*2
597: C/#IF ROUNDOFF(NEAREST)
598:          IT = min(1,INT(R(J)+X1)) + IWK3(J)
599: C/#ELSE
600: *          R1 = R(J) + X1
601: *          IT = IWK3(J) + R1
602: C/#ENDIF
603:          I = IWK4(J)
604:          ISF5(I) = ICX(L1+IT-1)
605:          THEA(I) = -1.0
606: C
607:          if ( INTE1.gt.0 ) then
608:              L2 = L1 + IWK3(J)
609:              LSTF51 = ICX(L2-1)
610:              LSTF52 = ICX(L2)
611:              NEP1 = ICX(LSTF51)
612:              NEP2 = ICX(LSTF52)
613:              if ( ILF(I).eq.61 ) then
614:                  LSTF51 = LSTF51 + 1
615:                  LSTF52 = LSTF52 + 1
616:              end if
617:              LLE1 = LSTF51 + 3*NEP1 + 1
618:              LLE2 = LSTF52 + 3*NEP2 + 1
619:              EMAX1 = CX(LLE1)
620:              THEA(I) = EMAX1 + (CX(LLE2)-EMAX1)*X1
621:              THEB(I) = 0.0
622:              if ( INTE1.eq.1 ) then
623:                  ND1 = ICX(LSTF51)
624:                  ND2 = ICX(LSTF52)
625:                  EMIN1 = CX(LLE1+ND1)
626:                  THEB(I) = EMIN1 + (CX(LLE2+ND2)-EMIN1)*X1
627:              end if
628:          end if
629: 590      continue
630: C
631:          NANG = 0
632: *VOCL LOOP,NOVREC
633:          do 600 I = N1, NIGEN
634:              if ( IWK6(I).le.0 ) then
635:                  ISF4(I) = 0
636:              else
637:                  NANG = NANG + 1
638:                  IWK4(NANG) = I
639:                  IWK1(NANG) = IWK6(I)
640:                  IWK2(NANG) = IWK7(I)
641:                  WK1(NANG) = EIN(I)
642:              end if
643: 600      continue
644: C
645:          call BSDEC3( CX, IWK1, WK3, IWK2, WK1, IWK3, NANG )
646:          call RANU2( IRAND, R, NANG, ICON )
647: C
648: *VOCL LOOP,NOVREC
649:          do 610 J = 1, NANG
650:              if ( IWK3(J).eq.0 ) IWK3(J) = 1

```

src/mvp/nxtnr3.f

```
651:          L2      = IWK2(J) + IWK3(J)
652:          L1      = L2 - 1
653:          INTE     = ICX(L2+IWK1(J))
654:          if ( INTE.eq.5 .or. INTE.eq.3 ) then
655: C ..... LOG      INTERPOLATION
656:          X1       = LOG(CX(L1)/WK1(J)) /LOG(CX(L1)/CX(L2))
657:          else
658: C ..... LINEAR INTERPOLATION
659:          X1       = (CX(L1)-WK1(J)) /(CX(L1)-CX(L2))
660:          end if
661:          if ( X1.lt.0.0 ) X1 = 0.0
662:          if ( X1.gt.1.0 ) X1 = 1.0
663:
664: C/#IF ROUNDOFF(NEAREST)
665:          IT       = min(1,INT(R(J)+X1)) + IWK3(J)
666: C/#ELSE
667: *          R1      = R(J) + X1
668: *          IT      = IWK3(J) + R1
669: C/#ENDIF
670:          I        = IWK4(J)
671:          ISF4(I) =
672:          &        IWK2(J) + IWK1(J)*2 + KL1B1(ICLN(I),20)*(IT-1)
673: 610      continue
674: C
675:      end if
676: C
677:      return
678:      end
```

src/mvp/nxtnr4.f

```

1:      subroutine NXTNR4(   IOW, A, H,
2:      X      CX      , ICX  , KLIB1, XLIB1, KLIB2, XLIB2,
3:      B      IBP      , ICLN , IMT   , ISF4 , ISF5 , ILF   ,
4:      B      EIN      , THEA , THEB  , EMIU , NRTYP,
5:      B      A0       , B0   , C0   ,
6:      B      ISEL     , NSEL  , NPHOT,
7:      B      AA       , BB    , CC    , COSL , WW    , EEEJ ,
8:      W      ICNJ     , IMTJ  , IWK3 , IWK4 , IWK5 , IWK6 ,
9:      W      IWK7     , IWK8 , IWK9 , IWK10,
10:     W      WK1      , WK2   , WK3   , WK4   , WK5   , WK6   ,
11:     W      R        )
12: C=====
13: C
14: C PURPOSE: calculation of secondary energy and scattering probability
15: C           for next event estimator
16: C CALLED IN: NXTNR
17: C CALLS: RANU2,BSDEC3, BSINC3, SEF5N3, SEF6N3
18: C
19: C=====
20: C
21: C      IBP(I)      : mother bank pointer
22: C      ICLN(I)     : collision nuclide
23: C      IMT(I)      : neutron emitting reaction
24: C      ISF4(I)     : pointer to equi-probable cosine bins for reaction
25: C                   given by files 4&5 and thermal elastic.
26: C                   for thermal inelastic, pointer to energy meshes
27: C                   or to another energy-angle distribution
28: C                   (LF=67,INT>10)
29: C      ISF5(I)     : pointer to energy(-angle) distribution
30: C      ILF(I)      : LF value for energy(-angle) distribution,
31: C                   for thermal inelastic, number of energy meshes
32: C      EIN(I)      : incident energy
33: C      THEA(I)     : theta for LF=5,7,9,11
34: C                   maximum energy for LF=1,6,61 (INT>10)
35: C                   interpolation ratio for LF=67
36: C      THEB(I)     : theta2 for LF=11
37: C                   minimum energy for LF=1,6,61 (INT>10)
38: C      EMIU(I)     : target mass for reactions with use of kinematics
39: C                   (effective target mass for moving target )
40: C                   EIN-U for LF=5,7,9,11
41: C      A0(I),B0(I),C0(I) : flight direction
42: C                   (effective direction for moving target )
43: C      NRTYP(10) : cumulative number of particles
44: C      1 : elastic scattering
45: C      2 : reaction with 2-body kinematics
46: C      3 : reaction with files 4 & 5
47: C      4 : fission
48: C      5 : reaction with file 6
49: C      6 : thermal inelastic
50: C      7 : incoherent thermal elastic
51: C      8 : coherent thermal elastic
52: C      9 : free gas
53: C      10 : isotropic without energy change
54: C      ISEL(J) : indicator of selected particles in collision stack.
55: C              ( variables native to I-index are referred as
56: C                EIN(ISEL(J)) etc. )
57: C      NSEL      : i: number of parent neutrons
58: C                  o: number of generated imaginary neutrons
59: C      NPHOT     : o: number of generated imaginary photons
60: C      NSEL(i) = NSEL(o)+NPHOT(o)
61: C      AA(J),BB(J),CC(J) : flight vectors of emitted imaginary
62: C                          particles
63: C      COSL(J)   :
64: C      WW(J)     : i: particle weights
65: C                  o: imaginary particle weight

```

```

66: C      EEEJ(J)   : o: energy of imaginary particles
67: C
68: C=====
69: C      implicit real*8 (A-H,O-Z)
70: C
71: C      real A(*)
72: C      real H(*)
73: C
74: C      include '../shared/INC/_SIZES'
75: C      include 'INC/_SIZES2'
76: C      include 'INC/_FLAGS'
77: C      include 'INC/_STACK'
78: C      include 'INC/_KPIDS'
79: C      include 'INC/_CXSEC'
80: C      include 'INC/_PXSEC'
81: C      include 'INC/_SBANK'
82: C      include '../shared/INC/_PTALY0'
83: C      include 'INC/_PTALY'
84: C      include 'INC/_PTALY2'
85: C
86: C
87: C**** CROSS SECTION DATA
88: C
89: C      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
90: C      integer ICX(MCX)
91: C      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
92: C
93: C**** TALLY (MONITOR) ARRAY
94: C
95: C      real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
96: C
97: C      integer ICNJ(NBANK), IMTJ(NBANK), IWK3(NBANK), IWK4(NBANK),
98: C      & IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
99: C      & IWK9(NBANK), IWK10(NBANK)
100: C      real R(3*NBANK)
101: C      real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
102: C      & WK5(NBANK), WK6(NBANK)
103: C
104: C**** Prepared stack and bank
105: C
106: C      integer IBP(NBANK), ICLN(NBANK), IMT(NBANK), ISF4(NBANK),
107: C      & ISF5(NBANK), ILF(NBANK), ISEL(NBANK)
108: C      integer NRTYP(10)
109: C      real EIN(NBANK), THEA(NBANK), THEB(NBANK),
110: C      & EMIU(NBANK)
111: C      real*8 A0(NBANK), B0(NBANK), C0(NBANK)
112: C
113: C      real COSL(NBANK), WW(NBANK), EEEJ(NBANK)
114: C
115: C****
116: C
117: C      parameter( PI = 3.1415926535897932D+00 )
118: C      parameter( PI2 = 2.0D0*PI )
119: C      parameter( PIH = 0.5D0*PI )
120: C      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
121: C      parameter( SMALL = 1.0D-35 )
122: C
123: C
124: C
125: C=====
126: C      if ( JVMNT.ne.0 ) call VMNTR1( 5, NSEL )
127: C
128: C=====
129: C
130: C      NELS      = 0

```

src/mvp/nxtnr4.f

```

131:      NKINT      = 0
132:      NF5T       = 0
133:      NFT        = 0
134:      NF6T       = 0
135:      NSABT      = 0
136:      NTELT      = 0
137:      NTEL2T     = 0
138:      NFRGST     = 0
139:      NWOECT     = 0
140:      do 100 J = 1, NSEL
141:      I = ISEL(J)
142: C .... collision nuclide
143:      ICNJ(J) = ICLN(I)
144: C .... reaction MT
145:      IMT(J) = IMT(I)
146: C .... incident energy
147:      EEEJ(J) = EIN(I)
148: C
149:      if ( I.le.NRTYP(1) ) then
150:      NELS = NELS + 1
151:      else if ( I.le.NRTYP(2) ) then
152:      NKINT = NKINT + 1
153:      else if ( I.le.NRTYP(3) ) then
154:      NF5T = NF5T + 1
155:      else if ( I.le.NRTYP(4) ) then
156:      NFT = NFT + 1
157:      else if ( I.le.NRTYP(5) ) then
158:      NF6T = NF6T + 1
159:      else if ( I.le.NRTYP(6) ) then
160:      NSABT = NSABT + 1
161:      else if ( I.le.NRTYP(7) ) then
162:      NTELT = NTELT + 1
163:      else if ( I.le.NRTYP(8) ) then
164:      NTEL2T = NTEL2T + 1
165:      else if ( I.le.NRTYP(9) ) then
166:      NFRGST = NFRGST + 1
167:      else if ( I.le.NRTYP(10) ) then
168:      NWOECT = NWOECT + 1
169:      end if
170: 100 continue
171:      NKINT = NELS + NKINT
172:      NF5T = NKINT + NF5T
173:      NFT = NF5T + NFT
174:      NF6T = NFT + NF6T
175: C      NSABT = NF6T + NSABT
176:      NSABT = NSABT
177:      NTELT = NSABT + NTELT
178:      NTEL2T = NTELT + NTEL2T
179:      NFRGST = NTEL2T + NFRGST
180:      NWOECT = NFRGST + NWOECT
181: C
182:      NNEUT = NF6T + NWOECT
183:      if ( JPHOT.eq.0 ) then
184:      if ( NSEL.ne.NNEUT ) then
185:      write(IOW,*) ' XXX at NXTNR3, NF6T+NWOECT, NSEL=',
186:      & NF6T+NWOECT,NSEL
187:      stop
188:      end if
189:      NPHOT = 0
190:      else
191:      NPHOT = NSEL - NNEUT
192:      end if
193: C
194: C**** reaction with 2-body kinematics
195: C
196:      if ( NKINT.gt.0 ) then
197: C
198: C .... N1 : number of particles which may have non-zero probability
199: C .... N2 : number of particles with gamma < 1.0 and 2 COS-CM
200:      N1 = 0
201:      N2 = 0
202: *VOCL LOOP,NOVREC
203:      do 110 J = 1, NKINT
204:      I = ISEL(J)
205: C      ATW = XLIB1(ICNJ(J),1)
206:      ATW = EMIU(I)
207:      GE = 1. - XLIB2(ICNJ(J),IMT(J),2)/EEEJ(J)
208:      if ( GE.lt.0.0 ) GE = 0.0
209:      GE = ATW*SQRT(GE)
210:      GE2 = GE*GE
211:      COSLJ = COSL(J)
212:      if ( GE.le.1.0 ) then
213:      COSMIN = SQRT( 1.0 - GE2 )
214:      else
215:      COSMIN = -2.0
216:      end if
217: C ... GE=0 : incident energy is just equal to threshold.
218: C      only when COSL=1, non-zero probability ( P=1 )
219: C      if ( GE.eq.0.0 ) then
220: C      EEEJ(J) = EEEJ(J) / (ATW+1.)/(ATW+1.)
221: C      if ( COSLJ.ne.1.0 ) WW(J) = 0.0
222: C      else if ( COSLJ.ge.COSMIN ) then
223: C .... neglect the tangential case
224: C      if ( COSLJ.gt.COSMIN ) then
225: C .... neglect GE < 1 except H-1 (GE=0.99917)
226: C      if ( GE.ge.1. .or. (ATW.lt.1.5.and.COSLJ.gt.COSMIN) ) then
227: C      COS2M = COSLJ * COSLJ - 1.0
228: C      XX = COS2M + GE2
229: C      CSQRD = COSLJ * SQRT(XX)
230: C      N1 = N1 + 1
231: C      IWK3(N1) = J
232: C      WK2(N1) = MIN(1.0D0, MAX(-1.0D0,
233: C      & (COS2M + CSQRD) / GE ) )
234: C      WK3(N1) = GE
235: C      EEEJ(J) = EEEJ(J) / (ATW+1.)/(ATW+1.)
236: C .... neglect the case of GE < 1
237: C      WK4(N1) = -10.0
238: C      if ( GE.lt.1.0 .and. CSQRD.ne.0.0 )
239: C      & WK4(N1) = MIN(1.0D0, MAX(-1.0D0,
240: C      & (COS2M - CSQRD) / GE ) )
241: C .... no rprobability to COSL(J) direction
242: C      else
243: C      WW(J) = 0.0
244: C      end if
245: C
246: 110 continue
247: C
248: C ---- find equi-probable cosine bin
249: C
250:      if ( N1.gt.0 ) then
251: *VOCL LOOP,NOVREC
252:      do 120 JJ = 1, N1
253:      J = IWK3(JJ)
254:      I = ISEL(J)
255:      IWK5(JJ) = ISF4(I)
256:      IWK6(JJ) = XLIB1( ICNJ(J),20 )
257: C .... this case does not occur, we hope
258: C      if ( WK2(JJ).lt.CX(IWK5(JJ)).or.
259: C      & WK2(JJ).gt.CX(IWK5(JJ)+IWK6(JJ)-1) ) then
260: C      WW(J) = -WW(J)

```


src/mvp/nxtnr4.f

```

261:      end if
262: C .... neglect the case of GE < 1
263: c      if ( WK4(JJ).ge.CX(IWK5(JJ)).or.
264: c      &      WK4(JJ).le.CX(IWK5(JJ)+IWK6(JJ)-1) ) then
265: c          N2 = N2 + 1
266: c          IWK4(N2) = JJ
267: c          IWK8(N2) = IWK5(JJ)
268: c          IWK9(N2) = IWK6(JJ)
269: c          WK4(NN) = WK4(JJ)
270: c      end if
271: 120      continue
272: C
273:      call BSINC3( CX, IWK6, WK6, IWK5, WK2, IWK7, N1 )
274: C
275: *VOCL LOOP,NOVREC
276:      do 130 JJ = 1, N1
277:      J = IWK3(JJ)
278:      if ( IWK7(JJ).le.0 ) IWK7(JJ) = 1
279: C .... this case does not occur, we hope
280:      if ( WW(J).le.0.0 ) then
281:      WW(J) = abs(WW(J))
282:      WK5(JJ) = 0.0
283:      else
284:      LL = IWK5(JJ)+IWK7(JJ)
285:      DMUPI2 = PI2 * (CX(LL) - CX(LL-1)) * (IWK6(JJ)-1)
286:      GE2 = WK3(JJ)*WK3(JJ)
287:      WK6(JJ) = 1. + 2.*WK3(JJ)*WK2(JJ) + GE2
288:      if ( WK3(JJ).gt.1. ) then
289:      WK5(JJ) = WK6(JJ)**(1.5) / GE2
290:      &      / ( WK3(JJ) + WK2(JJ) ) / DMUPI2
291: C .... we assume H-1 is only the case GE < 1, and GE=1 for H-1
292:      else
293:      WK5(JJ) = 4. * COSL(J) / DMUPI2
294:      end if
295:      end if
296: 130      continue
297: C
298: C .... from the assumption above, N2=0.
299:      if ( N2.gt.0 ) then
300: C
301:      call BSINC3( CX, IWK9, WK2, IWK8, WK4, IWK10, N2 )
302:      call RANU2( IRAND, R, N2, ICON )
303: C
304: *VOCL LOOP,NOVREC
305:      do 140 JJJ = 1, N2
306:      if ( IWK10(JJJ).le.0 ) IWK10(JJJ) = 1
307:      JJ = IWK4(JJJ)
308:      J = IWK3(JJ)
309:      GE2 = WK3(JJ) * WK3(JJ)
310:      X2 = 1. + 2.*WK3(JJ)*COSL(JJ) + GE2
311:      P2 = X2**(3./2.)
312: 140      continue
313:      end if
314: C
315: *VOCL LOOP,NOVREC
316:      do 150 JJ = 1, N1
317:      J = IWK3(JJ)
318:      if ( WK5(JJ).gt.0. ) then
319:      WW(J) = WW(J) * WK5(JJ)
320:      EEEJ(J) = EEEJ(J) * WK6(JJ)
321:      else
322:      WW(J) = 0.0
323:      end if
324: 150      continue
325:      end if

```

```

326: C
327:      end if
328: C
329: C**** reaction with files 4 & 5
330: C
331:      if ( NF5T.gt.NKINT ) then
332:      do 160 J = NKINT+1, NF5T
333:      I = ISEL(J)
334:      IWK5(J) = ISF4(I)
335:      IWK6(J) = KLIB1( ICNJ(J),20 )
336: 160      continue
337: C
338:      N = NF5T - NKINT
339:      call BSINC3( CX, IWK6(NKINT+1), WK6, IWK5(NKINT+1),
340:      &      COSL(NKINT+1), IWK7(NKINT+1), N )
341: C
342:      do 170 J = NKINT+1, NF5T
343:      if ( COSL(J).ge.CX(IWK5(J)).and.
344:      &      COSL(J).le.CX(IWK5(J)+IWK6(J)-1) ) then
345:      LL = IWK5(J)+IWK7(J)
346:      DMUPI2 = PI2 * (CX(LL) - CX(LL-1)) * (IWK6(J)-1)
347:      WW(J) = WW(J) / DMUPI2
348:      else
349:      WW(J) = 0.0
350:      end if
351: 170      continue
352:      end if
353: C
354: C ... for fission, P=1/4pi
355:      if ( NFT.gt.NF5T ) then
356:      do 180 J = NF5T+1, NFT
357:      WW(J) = WW(J)*PI4I
358: 180      continue
359:      end if
360: C
361: C**** selection of secondary energy from file 5 data
362: C
363:      if ( NFT.gt.NKINT ) then
364:      N = NFT - NKINT
365:      call SEF5N3( IOW, N ,
366:      &      ISEL(NKINT+1) , EEEJ(NKINT+1) ,
367:      &      ISF5 , ILF , THEA , THEB , EMIU ,
368:      &      CX , ICX , MCX , NBANK ,
369:      &      IWK3 , R , IRAND )
370: C .... SEF5N3 uses 4 random numbers
371:      end if
372: C
373: C**** selection of secondary energy from file 6 data
374: C
375:      if ( NF6T.gt.NFT ) then
376:      N = NF6T - NFT
377:      call SEF6N3( IOW, N ,
378:      &      ISEL(NFT+1) , EEEJ(NFT+1) , WW(NFT+1) , COSL(NFT+1),
379:      &      ISF5 , ILF , ICLN , IMT ,
380:      &      ISF4 , THEA , THEB ,
381:      &      CX , ICX , MCX , KLIB1, XLIB1, XLIB2,
382:      &      NUC , NMT , NBANK ,
383:      &      IWK3 , IWK4 , IWK5 , IWK6 , IWK7 , IWK8 ,
384:      &      WK2 , WK3 , WK4 , WK5 , WK6 , R ,
385:      &      IRAND )
386: C .... SEF6N3 uses 5 random numbers
387:      end if
388: C
389: C**** all thermal scattering is processed in NXTTHS
390: C

```

src/mvp/nxtnr4.f

```

391:      if ( NWOECT.gt.0 ) then
392:        call NXTTHS( IOW, NSABT, NTELT, NTEL2T, NFRGST, NWOECT, EBOT,
393:          &          ISEL(NF6T+1), EEEJ(NF6T+1), WW(NF6T+1), COSL(NF6T+1),
394:          &          ISF5, ISF4, ILF, ICLN, IMT,
395:          &          A0, B0, C0, EMIU, THEA,
396:          &          CX, ICX, MCX, KLIB1, XLIB1, KLIB2,
397:          &          NUC, NMT, NBANK,
398:          &          IWK3, IWK4, IWK5,
399:          &          IWK6,
400:          &          WK1, WK2, WK3, WK4, WK5, WK6,
401:          &          R, IRAND )
402: C*** NXTTHS uses 3 random numbers
403:      end if
404: C
405: C**** photon production
406: C
407:      if ( NPHOT.gt.0 ) then
408: C
409:        call RANU2( IRAND, R, NPHOT*3, ICON )
410:        N2 = NPHOT*2
411: C
412:        NANG = 0
413: *VOCL LOOP,NOVREC
414:        do 400 JJ = 1, NPHOT
415:          J = NNEUT + JJ
416:          I = ISEL(2)
417:          if ( ILF(I).eq.2 ) then
418:            EEEJ(J) = THEA(I)
419:          else if ( ILF(I).eq.6.or.ILF(I).eq.61 ) then
420:            LSTF5E = ISF5(I)
421:            NEP = ICX(LSTF5E)
422: C
423:            if ( ILF(I).eq.61 ) LSTF5E = LSTF5E + 1
424:            ND = ICX(LSTF5E)
425:
426: C/#IF ROUNDOFF(NEAREST)
427:            LL2 = MIN(INT(NEP*R(JJ))+1,NEP)
428: C/#ELSE
429:            *
430:            LL2 = NEP*R(JJ)+1
431: C/#ENDIF
432:            LL3 = LSTF5E + LL2
433:            LL4 = NEP + 2*LL2
434: C/#IF ROUNDOFF(NEAREST)
435:            LL4 = MIN(LL4,INT(LL4+R(N2+JJ)-CX(LL3))) + LSTF5E
436: C/#ELSE
437:            *
438:            R2 = R(N2+JJ) - CX(LL3)
439:            *
440:            LL4 = LSTF5E + LL4 + R2
441: C/#ENDIF
442:
443:            LL5 = ICX(LL4)
444:            LLL = LSTF5E + 3*NEP
445:            LE = LLL + LL5
446:            LP = LE + NEP + 1
447:            RR = R(NPHOT+JJ)
448: C
449:            if ( LL5.gt.ND ) then
450:              EEEJ(J) = CX(LE+1)
451: C
452:            else
453:              E0 = CX(LE)
454:              E1 = CX(LE+1)
455:              P0 = CX(LP)
456:              P1 = CX(LP+1)
457:              RRR = (P0+P1)*RR
458:              EEEJ(J) = E1 + (E0-E1)*RRR/

```

```

456:      &          (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
457: C
458:      if ( THEA(I).gt.0. .and. THEB(I).gt.0. ) then
459:        LLE = LLL + 1
460:        EEEJ(J) = THEB(I) + (THEA(I)-THEB(I))*
461:          &          (EEEJ(J)-CX(LLE+ND))/(CX(LLE)-CX(LLE+ND))
462:      else if ( THEA(I).gt.0. ) then
463:        LLE = LLL + 1
464:        EEEJ(J) = THEA(I)*EEEJ(J)/CX(LLE)
465:      end if
466: C
467:      end if
468:      end if
469: C
470:      NBINA = KLIB1( ICNJ(J),7 )
471:      IS = ISF4(I)
472: C
473:      if ( IS.eq.0 ) then
474:        WW(J) = WW(J) * PI4I
475:      else if
476:        &          ( COSL(J).lt.CX(IS).or.COSL(J).gt.CX(IS+NBINA) ) then
477:        WW(J) = 0.0
478:      else
479:        NANG = NANG + 1
480:        IWK5(NANG) = NBINA + 1
481:        IWK6(NANG) = IS
482:        IWK4(NANG) = J
483:        WK1(NANG) = COSL(J)
484:      end if
485: 400      continue
486: C
487:      if ( NANG.gt.0 ) then
488: C
489:        call BSINC3( CX, IWK5, WK3, IWK6, WK1, IWK3, NANG )
490: C
491: *VOCL LOOP,NOVREC
492:        do 410 JJ = 1, NANG
493:          LL = IWK6(JJ)+IWK3(JJ)
494:          DMUPI2 = PI2 * (CX(LL) - CX(LL-1)) * (IWK5(JJ)-1)
495:          J = IWK4(JJ)
496:          WW(J) = WW(J) / DMUPI2
497: 410      continue
498: C
499:      end if
500: C
501:      NSEL = NNEUT
502: C
503:      end if
504: C
505:      return
506:      end

```

src/mvp/nxtnr.f

```

1:      subroutine NXTNR( IOW, A, H, LSCOL,
2:      B      XXX , YYY , ZZZ , AAA , BBB , CCC ,
3:      B      EEE , WWW , IZZ , IGG , TTT , ITT ,
4:      B      LEVL , LZZ , LPOS , LCRS ,
5:      B      DBNK , KDBNK , MDBNK , IBNK , KIBNK , MIBNK ,
6:      B      NDBNK , NIBNK , NCNTR , WCNTR ,
7:      B      LZZI , LPOSI , LCRSI ,
8:      B      OPTI , PATH , KDETP , KLSFI , SMACI , MMACI ,
9:      G      KZMAT , KZREG , TIMEB , XPDET , IPDET , IPDT2 ,
10:     S      ENGYB , JPUSD , IMSFL , IMNFL , IMZFL ,
11:     S      IMDED , IMSLT , IMZLT , IMNLT ,
12:     W      X , Y , Z , A0 , B0 , C0 ,
13:     W      AA , BB , CC , DI , TI , WW ,
14:     W      IBP , ICLN , IMT , ISF4 , ISF5 , ILF ,
15:     W      EIN , W , THEA , THEB , EMIU , NRTYP ,
16:     W      EEEJ , IT0 , IGO , ISEL , IWK1 , IWK2 ,
17:     W      IWK3 , IWK4 , IWK5 , IWK6 , IWK7 , IWK8 ,
18:     W      IWK9 , IWK10 , WK1 , WK2 , WK3 , WK4 ,
19: c##<2007/03/14:PN4:
20: c## W      WK5 , WK6 , COSL , R , SMCW )
21:     W      WK5 , WK6 , COSL , R , SMCW ,
22:     &      IWK11 )
23: c##>
24: C=====
25: C
26: C PURPOSE: Selection of neutron collision point for next event
27: C estimator & call the estimation module
28: C CALLED IN: ACTION
29: C CALLS: VMNTR1,RANU2,LATUP2,LATDW2,NXTNR2,NXTNR3,NXTNR4,NXTNVT,GMACNX
30: C
31: C=====
32: C
33: C === INTER-STACK DATA FLOW ===
34: C
35: C COLLISION STACK -----> FREE-FLIGHT STACK FOR IMAGINARY PARTICLE
36: C (LSCOL,NCOLS) (IMSFL,IMZFL,IMNFL) (IMSLT,IMZLT,IMNLT)
37: C (NOT REMOVED)
38: C
39: C=====
40:     implicit real*8 (A-H,O-Z)
41: c
42:     real A(*)
43:     real H(*)
44: c
45:     include '../shared/INC/_SIZES'
46:     include '../shared/INC/_PGEOM'
47:     include 'INC/_SIZES2'
48:     include 'INC/_FLAGS'
49:     include 'INC/_STACK'
50:     include 'INC/_KPIDS'
51:     include 'INC/_NGPS'
52:     include 'INC/_CXSEC'
53:     include 'INC/_PXSEC'
54:     include 'INC/_SBANK'
55:     include '../shared/INC/_PTALY0'
56:     include 'INC/_PTALY'
57:     include 'INC/_PTALY2'
58: C
59: C**** DETECTOR DATA FOR NEXT EVENT (POINT) ESTIMATOR
60: C
61:     real*8 XPDET(NPLEN,NPDET)
62:     integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
63:     integer JPUSD(NPDET)
64: C
65: C**** TALLY BIN DATA

```

```

66:     real TIMEB(*)
67:     real ENGYB(*)
68: C
69: C**** GEOMETRY ARRAY DATA
70: C
71:     integer KZMAT(NZONE,2), KZREG(NZONE)
72: C
73: C**** CROSS SECTION DATA
74: C
75: C**** SIGMA BANK
76: C
77:     real SMACI(IMPMAX,MB,NSMACI)
78:     integer MMACI(IMPMAX,2)
79: C
80: C
81: C**** STACK FOR IMAGINARY PARTICLE
82: C
83:     integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(NZONE+1),
84:     & IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
85: C
86: C**** STACK FOR REAL PARTICLE
87: C
88:     integer LSCOL(NBANK)
89: C
90: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
91: C
92:     real*8 XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2), AAA(NBANK,2),
93:     & BBB(NBANK,2), CCC(NBANK,2)
94: C
95:     real*8 TTT(NBANK,2)
96:     real EEE(NBANK,2), WWW(NBANK,2)
97: C
98:     integer IZZ(NBANK,2), IGG(NBANK,2), ITT(NBANK,2), LEVL(NBANK,2),
99:     & LZZ(NBANK,NEST), LPOS(NBANK,NEST), LCRS(NBANK,NEST),
100:    & LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
101: C
102:     real*8 DBNK(NBANK,NDBNK,2)
103:     integer KDBNK(0:MDBNK)
104:     integer IBNK(NBANK,NIBNK,2)
105:     integer KIBNK(0:MIBNK)
106: C****
107:     real*8 OPTI(IMPMAX), PATH(IMPMAX)
108:     integer KDETP(IMPMAX), KLSFI(IMPMAX)
109: C
110: C**** TALLY (MONITOR) ARRAY
111: C
112:     real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
113: C
114: C**** WORKING AREA (THESE AREA CAN BE DESTROYED IN OTHER ROUTINES
115: C (IWK1,IWK2),(IWK3,IWK4),(IWK5,IWK6),...(WK1,WK2) ....
116: C (R ) are available as real*8 work area
117: C
118:     integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
119:     & IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
120:     & IWK9(NBANK), IWK10(NBANK)
121: c##<2007/03/14:PN4:
122:     integer IWK11(NBANK)
123: c##>
124:     real COSL(NBANK)
125:     real WK1(NBANK), WK2(NBANK)
126:     real WK3(NBANK), WK4(NBANK), WK5(NBANK), WK6(NBANK), R(8*NBANK)
127:     real SMCW(NBANK,NSMICI)
128: C
129: C**** WORK AREA ( CANNOT BE DESTROYED IN SUB-PROCESS (NXTEE) )
130: C

```

src/mvp/nxtnr.f

```

131:      integer IBP(NBANK), ICLN(NBANK), IMT(NBANK), ISF4(NBANK),
132:      &        ISF5(NBANK), ILF(NBANK), IT0(NBANK), IGO(NBANK),
133:      &        ISEL(NBANK)
134:      integer NRTYP(10)
135:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
136:      real*8 A0(NBANK), B0(NBANK), C0(NBANK),
137:      &        AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
138:      real W(NBANK), WW(NBANK), EIN(NBANK), THEA(NBANK), THEB(NBANK),
139:      &        EMIU(NBANK), EEEJ(NBANK)
140:      real*8 TI(NBANK)
141: C
142: C 0.1NBK.2NBK.3NBK.4NBK.5NBK.6NBK.7NBK.8NBK.9NBK,10NBK,11NBK
143: C****
144: C
145: C
146:      parameter( PI = 3.1415926535897932D+00 )
147:      parameter( PI2 = 2.0D0*PI )
148:      parameter( PIH = 0.5D0*PI )
149:      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
150: C
151: C .... constants ....
152: C
153: C --- light speed --- (cm/s)
154: C
155:      real*8 CLIGHT
156:      parameter( CLIGHT = 2.99792458D10 )
157: C
158: C --- neutron mass --- (gram)
159: C
160:      real*8 NMASS
161:      parameter( NMASS = 1.6749286D-24 )
162: C
163: C --- erg / eV --- electron charge
164: C
165:      real*8 EECHRG
166:      parameter( EECHRG = 1.60217733D-12 )
167: C
168: C --- neutron's mc**2 in eV
169: C
170:      real*8 MC2
171:      parameter( MC2 = NMASS*CLIGHT**2/EECHRG )
172: C
173: C
174: C=====
175: C      if ( JVMNT.ne.0 ) call VMNTR1( 5, NCOLS )
176: C
177: C=====
178: C      selection of reaction type generating imaginary particles.
179: C=====
180: C
181:      call NXTNR2( IOW, A, H, LSCOL, NIGEN,
182:      B   EEE , WWW ,
183:      B   DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
184:      B   NDBNK, NIBNK,
185:      X   A(LCX) , A(LCX) , A(LKLIB1), A(LXLIB1), A(LKLIB2), A(LXLIB2),
186:      X   H(LKMAC), H(LMMAC), H(LSMIC) , A(LLMIC) , H(LKSPI),
187:      B   IBP , ICLN , IMT , ISF4 , ISF5 , ILF ,
188:      B   EIN , W , THEA , THEB , EMIU , NRTYP,
189:      W   IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
190:      W   IWK7 , IWK8 , IWK9 , IWK10, WK1 ,
191:      W   AA , BB , CC , DI , WW ,
192:      W   R )
193: C
194: C      IBP(I) : bank pointer
195: C      ICLN(I) : collision nuclide

```

```

196: C      IMT(I) : neutron emitting reaction
197: C      EIN(I) : incident energy
198: C      W(I) : particle weights including multiplicity
199: C
200: C=====
201: C      gather spatial data of particles generating imaginary particles.
202: C=====
203: C
204: C
205:      if ( JLATT.eq.0 ) then
206: C ..... NO LATTICE
207:      do 100 I = 1, NIGEN
208:          X(I) = XXX(IBP(I),1)
209:          Y(I) = YYY(IBP(I),1)
210:          Z(I) = ZZZ(IBP(I),1)
211:          A0(I) = AAA(IBP(I),1)
212:          B0(I) = BBB(IBP(I),1)
213:          C0(I) = CCC(IBP(I),1)
214:      100 continue
215: C
216:      else
217: C
218: C ..... LATTICE
219: C ***** TRANSFORMATION OF CORDINATES AND DIRECTION
220: C FROM CELL-ORDINATE SYSTEM TO LABORATORY SYSTEM
221: C (XXX,YYY,ZZZ)&(AAA,BBB,CCC) --> (X,Y,Z)&(A0,B0,C0)
222: C
223:      JDIR = 1
224:      JLS = 0
225:      call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
226:      &           NZONE, NIGEN, NBANK, JDIR, JLS, IBP,
227:      &           X, Y, Z, A0, B0, C0,
228:      &           XXX, YYY, ZZZ,
229:      &           AAA, BBB, CCC,
230:      &           LEVL, LZZ, LPOS, LCRS,
231:      &           DBNK,KDBNK,MDBNK,
232:      &           A(LKZMAT),A(LKDALT),A(LDALT),A(LNVLAT),A(LSZLAT),
233:      &           A(LCELSZ),A(LIPLAT),A(LKLATT),A(LKSLAT),A(LLTYP),
234:      &           A(LICTYP),A(LMLBZZ),
235:      &           IWK1 )
236: C
237:      end if
238: C
239: C=====
240: C      Set parameters for generating imaginary particles.
241: C=====
242: C
243:      call NXTNR3( IOW, A, H,
244:      X   A(LCX) , A(LCX) , A(LKLIB1), A(LXLIB1), A(LKLIB2), A(LXLIB2),
245:      B   NIGEN, A0 , B0 , C0 ,
246:      B   IBP , ICLN , IMT , ISF4 , ISF5 , ILF ,
247:      B   EIN , W , THEA , THEB , EMIU , NRTYP,
248:      W   IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
249:      W   IWK7 , IWK8 , IWK9 , IWK10,
250:      W   AA , BB , CC , DI , WW ,
251:      W   R )
252: C
253: C      ISF4(I) : pointer to equi-probable cosine bins for reaction
254: C given by flies 4&5 and thermal elatic.
255: C for thermal inelastic, pointer to energy meshes
256: C or to another energy-angle distribution
257: C (LF=67,INT>10)
258: C      ILF(I) : LF value for energy(-angle) distribution
259: C      THEA(I) : theta for LF=5,7,9,11
260: C      maximum energy for LF=1,6,61 (INT>10)

```

src/mvp/nxtnr.f

```

261: C      interpolation ratio for LF=67
262: C      THEB(I) : theta2 for LF=11
263: C      minimum energy for LF=1,6,61 (INT>10)
264: C      EMIU(I) : target mass for reactions with use of kinematics
265: C      (effective target mass for moving target )
266: C      EIN-U for LF=5,7,9,11
267: C      A0(I),B0(I),C0(I) : flight direction
268: C      (effective direction for moving target )
269: C
270: C
271: C
272: C
273: C-----<< DO : LOOP INDEX DEFINITION HEREAFTER >> -----
274: C
275: C
276: C
277: C      I : INDICATE PARTICLES IN COLLISION STACK( IBP ).
278: C
279: C      << VARIABLES REFERENCED BY INDEX 'I' >>
280: C
281: C      IBP(I) : bank pointer
282: C      ICLN(I) : collision nuclide
283: C      IMT(I) : neutron emitting reaction
284: C      ISF4(I) : pointer to equi-probable cosine bins for reaction
285: C      given by files 4&5 and thermal elastic.
286: C      for thermal inelastic, pointer to energy meshes
287: C      or to another energy-angle distribution
288: C      (LF=67,INT>10)
289: C      ISF5(I) : pointer to energy(-angle) distribution
290: C      ILF(I) : LF value for energy(-angle) distribution,
291: C      for thermal inelastic, number of energy meshes
292: C      EIN(I) : incident energy
293: C      THEA(I) : theta for LF=5,7,9,11
294: C      maximum energy for LF=1,6,61 (INT>10)
295: C      interpolation ratio for LF=67
296: C      THEB(I) : theta2 for LF=11
297: C      minimum energy for LF=1,6,61 (INT>10)
298: C      EMIU(I) : target mass for reactions with use of kinematics
299: C      (effective target mass for moving target )
300: C      EIN-U for LF=5,7,9,11
301: C      NRTYP(10) : cumulative number of particles
302: C      1 : elastic scattering
303: C      2 : reaction with 2-body kinematics
304: C      3 : reaction with files 4 & 5
305: C      4 : fission
306: C      5 : reaction with file 6
307: C      6 : thermal inelastic
308: C      7 : incoherent thermal elastic
309: C      8 : coherent thermal elastic
310: C      9 : free gas
311: C      10 : isotropic without energy change
312: C
313: C      X(I),Y(I),Z(I) : collision points
314: C      A0(I),B0(I),C0(I) : flight direction
315: C      (effective direction for moving target )
316: C      W(I) : particle weights including multiplicity
317: C
318: C
319: C      J : INDICATE PARTICLES TO EMIT AN IMAGINARY PARTICLE
320: C      TOWARD ND'TH DETECTOR.
321: C      (SELECTED BY RUSSIAN ROULETTE AT THE BEGINNIG OF
322: C      DETECTOR LOOP)
323: C
324: C      < VARIABLES REFERENCED BY INDEX 'J' >
325: C

```

```

326: C      ISEL(J) : indicator of selected particles in collision stack.
327: C      ( variables native to I-index are referred as
328: C      A0(ISEL(J)) etc. )
329: C      AA(J),BB(J),CC(J) : flight vectors of emitted imaginary
330: C      particles.
331: C      WW(J) : imaginary particle weight
332: C      COSL(J)
333: C
334: C
335: C**** LOOP OVER DETECTOR (NPDET) *****
336: C
337: C
338: C
339: C
340: C      do 200 ND = 1, NPDET
341: C
342: C      if ( JPUSD(ND).eq.0 ) go to 200
343: C
344: C=====
345: C
346: C      CALCULATE DISTANCES TO ND'TH DETECTOR
347: C      & SELECT COLLISIONS COUNTED AT THE DETECTOR
348: C
349: C=====
350: C
351: C
352: C
353: C
354: C      XDA = XPDET(3,ND)
355: C      YDA = XPDET(4,ND)
356: C      ZDA = XPDET(5,ND)
357: C      RMAX = XPDET(2,ND)
358: C      RMA2 = RMAX*RMAX
359: C      RMIN = XPDET(1,ND)
360: C      RMIN2 = RMIN*RMIN
361: C
362: C      call RANU2( IRAND, R, NIGEN, ICON )
363: C
364: C      C NN : NUMBER OF SELECTED COLLISION POINTS
365: C      C NBS : NUMBER OF SELECTED COLLISION POINTS WHICH REQUIRE BOUNDED SPHERE
366: C
367: C      NN = 0
368: C      NBS = 0
369: C
370: C      do 210 I = 1, NIGEN
371: C      AT = XDA - X(I)
372: C      BT = YDA - Y(I)
373: C      CT = ZDA - Z(I)
374: C      D = AT*AT + BT*BT + CT*CT
375: C
376: C      ..... CALCULATE CONTRIBUTION WITH PROBABILITY RMIN2/D
377: C      WHEN D > RMIN2.
378: C
379: C      if ( D.le.RMIN2 .or. D*R(I).le.RMIN2 ) then
380: C      NN = NN + 1
381: C      ISEL(NN) = I
382: C      DI(NN) = D
383: C      AA(NN) = AT
384: C      BB(NN) = BT
385: C      CC(NN) = CT
386: C      WW(NN) = W(I)
387: C      c##<2007/03/14:PN4:
388: C      IWK11(NN) = IBP(I)
389: C      c##>
390: C      end if

```

src/mvp/nxtnr.f

```

391: 210 continue
392: C
393: do 220 J = 1, NN
394: C
395: C ..... THE BOUNDED SPHERE APPROXIMATION IF DI < PMAx**2 ...
396: C
397: if ( DI(J).lt.RMA2 ) then
398: NBS = NBS + 1
399: C
400: C
401: C ..... DEVIDE WEIGHT BY (DISTANCE)**2 HERE ....
402: C ..... WEIGHT CORRECTION NECESSARY IF DI > RMIN2 ...
403: C
404: C MEANING : WW(J) = WW(J) / DI(J) *1/(RMIN2/DI(J))
405: C
406: C
407: else if ( DI(J).gt.RMIN2 ) then
408: WW(J) = WW(J) /RMIN2
409: C
410: else
411: WW(J) = WW(J) /DI(J)
412: C
413: C
414: C ..... 'DI' MEANS DISTANCE ITSELF HEREAFTER .....
415: C
416: DI(J) = SQRT(DI(J))
417: C
418: C ..... (AA,BB,CC) IS NORMALIZED DIRECTION VECTOR HEREAFTER ...
419: C
420: C
421: if ( DI(J).gt.0. ) then
422: AA(J) = AA(J) /DI(J)
423: BB(J) = BB(J) /DI(J)
424: CC(J) = CC(J) /DI(J)
425: C
426: else
427: AA(J) = 0.
428: BB(J) = 0.
429: CC(J) = 0.
430: C
431: C
432: C ..... CALCULATION OF SCATTERING ANGLE
433: C ASSUME COSL=0.0 FOR PARTICLES WITH DI(J)=0.0
434: C
435: do 230 J = 1, NN
436: I = ISEL(J)
437: COSL(J) = MIN(1.0D0, MAX(-1.0D0,
438: & A0(I)*AA(J)+B0(I)*BB(J)+C0(I)*CC(J) ))
439: C
440: C
441: C .... calculate energy and scattering probability of imaginary particle
442: C ( neutron + photon )
443: C NN ---> NN ( neutron ) + NP ( photon )
444: C
445: call NXTNR4( IOW, A, H,
446: X A(LCX) , A(LCX) , A(LKLIB1), A(LXLIB1), A(LKLIB2), A(LXLIB2),
447: B IBP , ICLN , IMT , ISF4 , ISF5 , ILF ,
448: B EIN , THEA , THEB , EMIU , NRTYP,
449: B A0 , B0 , C0 ,
450: B ISEL , NN , NP ,
451: B AA , BB , CC , COSL , WW , EEEJ ,
452: W IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
453: W IWK7 , IWK8 , IWK9 , IWK10,
454: W WK1 , WK2 , WK3 , WK4 , WK5 , WK6 ,
455: W R )

```

```

456: C
457: C .... remove no-probability particle and energy cut-off .....
458: C
459: NNN = 0
460: N1 = NN
461: C
462: if ( ETHMAX.ge.EBOT ) then
463: *VOCL LOOP,NOVREC
464: do 240 J = 1, NN
465: if ( EEEJ(J).lt.EBOT ) EEEJ(J) = EBOT
466: if ( WW(J).gt.0.0 ) then
467: NNN = NNN + 1
468: ISEL(NNN) = ISEL(J)
469: IWK1(NNN) = J
470: EEEJ(NNN) = EEEJ(J)
471: DI(NNN) = DI(J)
472: end if
473: 240 continue
474: C
475: else
476: C
477: *VOCL LOOP,NOVREC
478: do 250 J = 1, NN
479: if ( WW(J).gt.0.0 .and.EEEJ(J).ge.EBOT ) then
480: NNN = NNN + 1
481: ISEL(NNN) = ISEL(J)
482: IWK1(NNN) = J
483: EEEJ(NNN) = EEEJ(J)
484: DI(NNN) = DI(J)
485: end if
486: 250 continue
487: end if
488: NN = NNN
489: C
490: do 260 J = 1, NN
491: EEEJ(J) = MIN( EEEJ(J),ETOP )
492: 260 continue
493: C
494: C .... photon .....
495: C
496: if ( NP.gt.0 ) then
497: *VOCL LOOP,NOVREC
498: do 270 J = N1+1, N1+NP
499: if ( WW(J).gt.0.0 .and.EEEJ(J).ge.EBOTP ) then
500: NNN = NNN + 1
501: ISEL(NNN) = ISEL(J)
502: IWK1(NNN) = J
503: EEEJ(NNN) = EEEJ(J)
504: DI(NNN) = DI(J)
505: end if
506: 270 continue
507: C
508: do 280 J = NN+1, NNN
509: EEEJ(J) = MIN( EEEJ(J),ETOPP )
510: 280 continue
511: NP = NNN - NN
512: C
513: end if
514: C
515: C .... TIME CUT OFF .....
516: C
517: if ( JTIME.ne.0 ) then
518: if ( JPTIM.eq.0 ) then
519: NNNN = NNN
520: NNN = 0

```

src/mvp/nxtnr.f

```

521:      N1      = NN + 1
522: *VOCL LOOP,NOVREC
523:      do 290 J = 1, NN
524:        DDE    = EEEJ(J)
525:        VEL    = CLIGHT*SQRT(DDE*(DDE+2*MC2)) / (DDE+MC2)
526:        TTTT   = TTT(IBP(ISEL(J)),1) + DI(J) /VEL
527:        if ( TTTT.le.TCUT ) then
528:          NNN   = NNN + 1
529:          ISEL(NNN) = ISEL(J)
530:          IWK1(NNN) = IWK1(J)
531:          TI(NNN)  = TTTT
532:          EEEJ(NNN) = EEEJ(J)
533:          DI(NNN)  = DI(J)
534:        end if
535:      290 continue
536:      NN      = NNN
537: C
538:      if ( NP.gt.0 ) then
539: *VOCL LOOP,NOVREC
540:      do 300 J = N1, NNNN
541:        DDE    = EEEJ(J)
542:        TTTT   = TTT(IBP(ISEL(J)),1) + DI(J) /CLIGHT
543:        if ( TTTT.le.TCUT ) then
544:          NNN   = NNN + 1
545:          ISEL(NNN) = ISEL(J)
546:          IWK1(NNN) = IWK1(J)
547:          TI(NNN)  = TTTT
548:          EEEJ(NNN) = EEEJ(J)
549:          DI(NNN)  = DI(J)
550:        end if
551:      300 continue
552:      NP      = NNN - NN
553:      end if
554: C
555:      else
556:      do 310 J = 1, NN
557:        DDE    = EEEJ(J)
558:        VEL    = CLIGHT*SQRT(DDE*(DDE+2*MC2)) / (DDE+MC2)
559:        TTTT   = TTT(IBP(ISEL(J)),1) + DI(J) /VEL
560:        if ( TTTT.ge.TCUT ) then
561:          ITREP = TTTT / TCUT
562:          TI(J) = TTTT - TCUT*ITREP
563: c##<2007/04/09: bug fix
564:          else
565:            TI(J) = TTTT
566: c##>
567:          end if
568:      310 continue
569: C
570:      if ( NP.gt.0 ) then
571:      do 320 J = NN+1, NNN
572:        TTTT   = TTT(IBP(ISEL(J)),1) + DI(J) /CLIGHT
573:        if ( TTTT.ge.TCUT ) then
574:          ITREP = TTTT / TCUT
575:          TI(J) = TTTT - TCUT*ITREP
576: c##<2007/04/09: bug fix
577:          else
578:            TI(J) = TTTT
579: c##>
580:          end if
581:      320 continue
582:      end if
583: C
584:      end if
585: C

```

```

586:      call BS0ISD( TIMEB, NTIME+1, TI, IT0, NNN )
587: C
588:      do 330 J = 1, NNN
589:        IT0(J) = MAX(1,MIN(NTIME,IT0(J)))
590:      330 continue
591: C
592:      end if
593: C
594: C .... condense imaginary particle data ....
595: C
596: *VOCL LOOP,NOVREC
597:      do 340 J = 1, NNN
598:        WW(J) = WW(IWK1(J))
599:        AA(J) = AA(IWK1(J))
600:        BB(J) = BB(IWK1(J))
601:        CC(J) = CC(IWK1(J))
602: c##<2007/03/14:PN4:
603:        IWK11(J) = IWK11(IWK1(J))
604: c##>
605:      340 continue
606: C
607: C .... energy group
608: C
609:      call BSVDEC( ENGYB(KENGP(KPNEUT))), NGP(KPNEUT)+1,
610:      &          EEEJ, IGO, NN )
611:      NGG      = KNGP(KPNEUT) - 1
612:      do 350 J = 1, NN
613:        IGO(J) = IGO(J) + NGG
614:      350 continue
615: C
616:      if ( NP.gt.0 ) then
617:      call BSVDEC( ENGYB(KENGP(KPPHOT))), NGP(KPPHOT)+1,
618:      &          EEEJ(NN+1), IGO(NN+1), NP )
619:      NGG      = KNGP(KPPHOT) - 1
620:      do 355 J = NN+1, NNN
621:        IGO(J) = IGO(J) + NGG
622:      355 continue
623:      end if
624: C
625: C
626: C=====
627: C
628: C      SEND PARTICLES TO STACK FOR IMAGINARY PARTICLES
629: C
630: C=====
631: C
632: C
633:      IZD      = IPDET(ND,1)
634:      LVL      = IPDET(ND,2)
635:      if ( JLATT.ne.0.and.LVL.gt.0 ) then
636:      if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
637:      LP       = 2 + 3*LVL
638:      XDA      = XPDET(LP+1,ND)
639:      YDA      = XPDET(LP+2,ND)
640:      ZDA      = XPDET(LP+3,ND)
641:      end if
642: C
643:      360 continue
644: C
645:      if ( NNN.gt.0 ) then
646: C
647:      NNNN     = MIN(NNN,NDIMPT)
648: C
649:      if ( NNNN.gt.0 ) then
650:      NDIMPT   = NDIMPT - NNNN

```

src/mvp/nxtnr.f

```

651:      N      = IMNFL(NZONE+1)
652:      N1     = NNN - NNNN
653:      *VOCL LOOP,NOVREC
654:      do 370 J = 1, NNNN
655:      J1      = N1 + J
656:      IMSFL(N+J) = IMDED(NDIMPT+J)
657:      IMZFL(N+J) = IZD
658: C
659: C ---- BEWARE THAT IMAGINARY PARTICLES FLY FROM THE DETECTOR POINT
660: C TO COLLISION POINTS. ----
661: C
662:      XXX(IMSFL(N+J),2) = XDA
663:      YYY(IMSFL(N+J),2) = YDA
664:      ZZZ(IMSFL(N+J),2) = ZDA
665:      AAA(IMSFL(N+J),2) = -AA(J1)
666:      BBB(IMSFL(N+J),2) = -BB(J1)
667:      CCC(IMSFL(N+J),2) = -CC(J1)
668:      WWW(IMSFL(N+J),2) = WW(J1)
669:      EEE(IMSFL(N+J),2) = EEEJ(J1)
670:      IGG(IMSFL(N+J),2) = IGO(J1)
671:      TTT(IMSFL(N+J),2) = TI(J1)
672:      PATH(IMSFL(N+J)) = DI(J1)
673:      OPTI(IMSFL(N+J)) = 0.0
674:      KDETP(IMSFL(N+J)) = ND
675:      KLSFI(IMSFL(N+J)) = 0
676:      if ( JTIME.ne.0 ) then
677:      ITT(IMSFL(N+J),2) = IT0(J1)
678:      end if
679:      if ( KIBNK(5).ne.0 ) then
680:      IBNK(IMSFL(N+J),KIBNK(5),2) = -1
681:      end if
682: c##<2007/03/14:PN4:
683:      if ( KIBNK(17).ne.0 ) IBNK(IMSFL(N+J),KIBNK(17),2) =
684:      1 IBNK(IWK11(J1),KIBNK(17),1)
685:      if ( KIBNK(18).ne.0 ) IBNK(IMSFL(N+J),KIBNK(18),2) =
686:      1 IBNK(IWK11(J1),KIBNK(18),1)
687:      if ( KIBNK(19).ne.0 ) IBNK(IMSFL(N+J),KIBNK(19),2) =
688:      1 IBNK(IWK11(J1),KIBNK(19),1)
689: c##>
690:      370 continue
691: C
692:      NN2 = NNNN
693:      JSTG = 0
694: C
695: C=====
696: C
697: C In LATTICE geometry, the following parameters should be set.
698: C LEVL, LZZI, LPOSI, LCRSI
699: C DBNK, IBNK (JFISX.ne.0)
700: C JSTG : flag to indicate the detector is located in STG region.
701: C NN2 : number of remaining particles when JFISX.ne.0.
702: C
703: C=====
704: C
705:      if ( JLATT.ne.0 ) then
706:      call LATDW2(IOW, JDEBG, JHLAT, JFISX,
707:      N IMPMAX,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
708:      N DINF ,DEPS ,
709:      S IMSFL(N+1),NNNN ,JSTG ,NN2 ,
710:      I IPDET(ND,2),IPDT2(1,1,ND),IPDT2(1,2,ND),
711:      I IPDT2(1,3,ND),XPDET(3,ND),
712:      B AAA(1,2),BBB(1,2),CCC(1,2) ,
713:      B LEVL(1,2),LZZI ,LPOSI ,LCRSI ,
714:      B DBNK(1,1,2),KDBNK ,MDBNK ,IBNK(1,1,2),KIBNK ,MIBNK,
715:      S IMDED ,NDIMPT,

```

```

716:      G      A(LSDA) ,A(LKZDA) ,A(LKZAA) ,
717:      G      A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LMLBZZ),A(LCELSZ),
718:      G      A(LSZLAT),A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),
719:      G      A(LNVLAT),A(LIPCEL),A(LLTYP) ,A(LICTYP),A(LKDALT),
720:      G      A(LDALT) ,A(LKZREG),
721:      W      IWK1 ,IWK3 ,IWK5 ,IWK7 ,IWK9 ,WK1 ,
722:      W      WK3 ,WK5 ,R ,WW(N1+1),IGO(N1+1) )
723:      end if
724: C
725:      if ( JSTG.eq.0 ) then
726:      IMNFL(IZD) = IMNFL(IZD) + NN2
727:      IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
728: C
729: C ... detector is located in STG region. move to LATTICE stack.
730:      else
731:      NL = IMNLT(NLBZ+1)
732:      do 380 J = 1, NN2
733:      IMSLT(NL+J) = IMSFL(N+J)
734:      IMZLT(NL+J) = IZD
735:      380 continue
736:      IMNLT(IZD) = IMNLT(IZD) + NN2
737:      IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
738:      end if
739: C
740:      NNN = NNN - NNNN
741:      NIMPT = NIMPT + NN2
742:      end if
743: C
744: C *** prepare cross sections *****
745: C
746:      MAT = KZMAT(IZD,1)
747:      JAMXCT = 0
748:      JMIC = 1
749:      if ( JPHOT.eq.0 ) then
750:      call GMACNX( IRAND, NN2, IMSFL(N+1), MAT,
751:      & JMIC, JAMXCT, JNEUT, JGAMM, JDEBG, IMPMAX,
752:      & EEE(1,2), MB, NUC, NSTAL,
753:      & NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
754:      & NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTLI),
755:      & A(LLPDEN), A(LINUCT), A(LDENST), NMAT, NUC, NMT,
756:      & A(LCX), MCX, A(LKLIB1), A(LKLIB2), A(LXLIB1),
757:      & A(LCRES), A(LKCRSI), MCRES, NNCSI, A(LINCSI),
758:      W IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
759:      W IWK7, IWK8, R, SMCW )
760:      else
761:      NGG = KNGP(KPPHOT) - 1
762:      N1 = 0
763:      N2 = 0
764:      do 390 J = 1, NN2
765:      IP = IMSFL(N+J)
766:      if ( IGG(IP,2).le.NGG ) then
767:      N1 = N1 + 1
768:      IWK9(N1) = IP
769:      else
770:      N2 = N2 + 1
771:      IWK10(N2) = IP
772:      end if
773:      390 continue
774:      if ( N1.gt.0 ) then
775:      call GMACNX( IRAND, N1, IWK9, MAT,
776:      & JMIC, JAMXCT, JNEUT, JGAMM, JDEBG, IMPMAX,
777:      & EEE(1,2), MB, NUC, NSTAL,
778:      & NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
779:      & NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTLI),
780:      & A(LLPDEN), A(LINUCT), A(LDENST), NMAT, NUC, NMT,

```


src/mvp/nxtnr.f

```

781:      &      A(LCX),      MCX,      A(LKLIB1), A(LKLIB2), A(LXLIB1),
782:      &      A(LCRES), A(LKCRSI), MCRES, NNCSI, A(LINC SI),
783:      W      IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
784:      W      IWK7, IWK8, R,      SMCW )
785:      end if
786:      if ( N2.gt.0 ) then
787:      JNP = 2
788:      call GMACNX( IRAND, N2, IWK10, MAT,
789:      &      JM IC, JAMXCT, JNP, JGAMM, JDEBG, IMPMAX,
790:      &      EEE(1,2), MB, NUC, NSTAL,
791:      &      NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
792:      &      NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGT LI),
793:      &      A(LLPDNP), A(LIATMT), A(LDNSTP), NMAT, NPATOM, NMTP,
794:      &      A(LCXP), MCKP, A(LKLB P1), A(LKLB P2), A(LXLB P1),
795:      &      A(LCRES), A(LKCRSI), MCRES, NNCSI, A(LINC SI),
796:      W      IWK1, IWK2, IWK3, IWK4, IWK5, IWK6,
797:      W      IWK7, IWK8, R,      SMCW )
798:      end if
799: C
800:      end if
801:
802: C
803: C .... BOUNDED SPHERE APPROXIMATION
804: C
805:      if ( NBS.gt.0 ) then
806:      if ( MAT.gt.0 ) then
807: *VOCL LOOP,NOVREC
808:      do 400 J = 1, NN2
809:      IP = IMSFL(N+J)
810:      if ( PATH(IP).le.RMAX ) then
811:      RMD = SMACI(IP,MMACI(IP,1),1) * RMAX
812:      RMI = 1./RMD
813:      C2 = RMA2*
814:      &      (RMI*RMI*2.+(1.+2.0*RMI)/(1.-EXP(RMD)))
815:      WWW(IP,2) = WWW(IP,2) /C2
816:      end if
817: 400      continue
818:      else
819: *VOCL LOOP,NOVREC
820:      do 410 J = 1, NN2
821:      IP = IMSFL(N+J)
822:      if ( PATH(IP).le.RMAX ) then
823:      WWW(IP,2) = WWW(IP,2) /RMA2 *3.0
824:      end if
825: 410      continue
826:      end if
827:      end if
828: C
829: C
830: C *** CALL TRACKING CONTROL ROUTINE FOR IMAGINARY PARTICLES
831: C
832:      if ( NDIMPT.eq.0 ) then
833:      call NXTEE( 2, A, H, NDIMPT, NIMPT )
834:      end if
835: C
836:      go to 360
837: C
838:      end if
839: C
840: C
841: 200 continue
842: C
843: C **** END OF DETECTOR LOOP *****
844: C
845:      return

```

846: end

src/mvp/nxtths.f

```

1:      subroutine NXTTHS( IOW, NSABT, NTELT, NTEL2T,NFRGST,NWOECT, EBOT,
2:      &                  ISEL, EEEJ, WW,   COSL,
3:      &                  ISF5, ISF4, ILF,   ICLN, IMT,
4:      &                  A0,   B0,   C0,   EMIU, THEA,
5:      &                  CX,   ICX,   MCX,   KLIB1, XLIB1, KLIB2,
6:      &                  NUC,   NMT,   NBANK,
7:      &                  IWK1, IWK2, IWK3,
8:      &                  WK1,  WK2,  WK3,  WK4,  WK5,  WK6,
9:      &                  WK7,
10:     &                  R,   IRAND )
11: C---5---1---5---2---5---3---5---4---5---5---5---6---5---7---
12: C=====
13: C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM THERMAL SCATTERING
14: C using the following data already determined.
15: C
16: C ISEL(J) : I = ISEL(J)
17: C NSABT ..: number of particles to be processed (J=1,...)
18: C EEEJ(J) : incident energy
19: C WW(J)   :
20: C COSL(J) : scattering angle cosine in Lab system
21: C AA(J),BB(J),CC(J) : direction to detector
22: C ISF5(I) : pointer to used data in CX array
23: C ISF4(I) : pointer to energy meshes in CX array
24: C ILF(I)  : incident energy mesh
25: C ICLN(I) : collision nuclide
26: C IMT(I)  : reaction MT
27: C A0(I),B0(I),C0(I) : flight direction of neutrons
28: C EMIU(I) : effective target mass in case of moving target
29: C THEA(I) : scattering angle cosine by thermal coherent
30: C           elastic scattering ( sampled )
31: C
32: C -----> EEEJ(J) : OUTGOING ENERGY
33: C           WW(J)
34: C
35: C CALLED IN: NXTNR4
36: C CALL      : RANU2, BSINC3
37: C=====
38: C
39: C implicit real*8(A-H,O-Z)
40: C
41: C integer ISF5(NBANK), ISF4(NBANK), ILF(NBANK), ICLN(NBANK),
42: C &      IMT(NBANK), ISEL(NBANK)
43: C real    EEEJ(NBANK), WW(NBANK), COSL(NBANK), EMIU(NBANK),
44: C &      THEA(NBANK)
45: C real*8  A0(NBANK), B0(NBANK), C0(NBANK)
46: C real    EBOT
47: C
48: C**** CROSS SECTION DATA IN CX ARRAY
49: C real CX(MCX), XLIB1(NUC,11)
50: C integer ICX(MCX)
51: C##<2007/03/14:PN3:
52: C## integer KLIB1(NUC,31), KLIB2(NUC,NMT,13)
53: C integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
54: C##>
55: C
56: C
57: C**** WORK AREA
58: C
59: C integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK)
60: C real    WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
61: C &      WK5(NBANK), WK6(NBANK), WK7(NBANK), R(NBANK*3)
62: C
63: C****
64: C parameter( PI = 3.141592653589793200D+00 )
65: C parameter( PIH = 0.5D0*PI )

```

```

66: C parameter( PI2 = 2.0D0*PI )
67: C parameter( PI2I = 1.0D0/(2.0D0*PI ) )
68: C parameter( PI4I = 1.0D0/(4.0D0*PI ) )
69: C parameter( PEPS = 1.0D-4 )
70: C
71: C
72: C if ( NSABT.gt.0 ) then
73: C
74: C**** THERMAL INELASTIC SCATTERING
75: C
76: C N1 = NSABT
77: C call RANU2( IRAND, R, N1*3, ICON )
78: C N12 = N1*2
79: C
80: C*VOCL LOOP,NOVREC
81: C do 150 J = 1, N1
82: C I = ISEL(J)
83: C..... DETERMINE THE OUTGOING ENERGY
84: C IIN = ILF(I)
85: C NELF = ICX(ISF4(I)-1)
86: C L1 = ISF5(I) - 1
87: C
88: C/#IF ROUNDOFF(NEAREST)
89: C L2 = MIN( INT(NELF*R(J))+1,NELF )
90: C/#ELSE
91: C * L2 = NELF*R(J) + 1
92: C/#ENDIF
93: C
94: C L3 = L1 + L2
95: C
96: C/#IF ROUNDOFF(NEAREST)
97: C L4 = 2*L2
98: C L4 = min( L4,INT(L4+R(N1+J)-CX(L3))) + L1 + NELF
99: C/#ELSE
100: C * R1 = R(N1+J) - CX(L3)
101: C * L4 = L1 + NELF + 2*L2 + R1
102: C/#ENDIF
103: C
104: C IOUT = ICX(L4)
105: C*VOCL STMT,IF(0)
106: C if ( IOUT.eq.1 ) then
107: C EEEJ(J) = CX(ISF4(I))
108: C else
109: C L6 = ISF4(I) + IOUT - 1
110: C L5 = L6 - 1
111: C EEEJ(J) = (CX(L5)-CX(L6))*R(N12+J) + CX(L6)
112: C IOUTT = R(N12+J) + 0.5
113: C IOUT = IOUT - IOUTT
114: C..... SET EEE(IP) =< ETHMAX
115: C end if
116: C
117: C..... DETERMINE SCATTERING PROBABILITY
118: C
119: C if ( KLIB1(ICLN(I),31).ne.1 ) then
120: C I1 = MIN(IIN,IOUT)
121: C I2 = MAX(IIN,IOUT)
122: C I22 = I2*(I2-1) /2
123: C L7 = KLIB2(ICLN(I),92,12) + NELF*2 + (I22+I1-1)*5
124: C else
125: C L7 = KLIB2(ICLN(I),92,12) + NELF*2 +
126: C & NELF*5*(IIN-1) + 5*(IOUT-1)
127: C end if
128: C P1 = CX(L7)
129: C P2 = CX(L7+1)
130: C P3 = CX(L7+2)

```

src/mvp/nxtths.f

```

131:      P4      = CX(L7+3)
132:      P5      = CX(L7+4)
133:      P3P2    = P3 - P2
134:      P4P3    = P4 - P3
135: C
136:      if ( COSL(J).ge.0.0 .and. P3P2.gt.0.0 ) then
137:      WK1(J)   = P3P2*(6.*COSL(J)-2.0) + P1
138:      end if
139:      if ( COSL(J).ge.0.0 .and. P3P2.le.0.0 ) then
140:      WK1(J)   = P5*(4.-6.*COSL(J)) + P1
141:      end if
142:      if ( COSL(J).lt.0.0 .and. P4P3.gt.0.0 ) then
143:      WK1(J)   = P4P3*(2.0-6.*COSL(J)) + P2 - P1
144:      end if
145:      if ( COSL(J).lt.0.0 .and. P4P3.le.0.0 ) then
146:      WK1(J)   = (1.0-P4-P5)*(6.*COSL(J)+4.0)
147:      &      + P2 - P1
148:      end if
149: C
150:      WW(J)    = WW(J) * WK1(J) * PI2I
151:      EEEJ(J)  = MAX(EEEJ(J),EBOT)
152: 150 continue
153: end if
154: C
155: C**** INCOHERENT THERMAL ELASTIC SCATTERING
156: C
157:      if ( NTELT.gt.NSABT ) then
158:      N = NTELT - NSABT
159:      do 160 J = NSABT+1, NTELT
160:      IWK2(J)  = ILF(ISEL(J))
161:      IWK3(J)  = ISF4(ISEL(J))
162: 160 continue
163:      call BSINC3( CX, IWK2(NSABT+1), WK2, IWK3(NSABT+1),
164:      &      COSL(NSABT+1), IWK1(NSABT+1), N )
165: C
166:      do 170 J = NSABT+1, NTELT
167:      if ( COSL(J).ge.CX(IWK3(J)).and.
168:      &      COSL(J).le.CX(IWK3(J)+IWK2(J)-1) ) then
169:      LL      = IWK3(J)+IWK1(J)
170:      DMUPI2  = PI2 * (CX(LL) - CX(LL-1)) * (IWK2(J)-1)
171:      WW(J)   = WW(J) / DMUPI2
172:      else
173:      WW(J)   = 0.0
174:      end if
175: 170 continue
176: end if
177: C
178: C**** Coherent thermal elastic scattering
179: C
180:      if ( NTELT2T.gt.NTELT ) then
181:      DELOM   = 1.D-1
182:      CONT    = DELOM / 2.0
183: C
184:      do 180 J = NTELT+1, NTELT2T
185:      DEL     = ABS(COSL(J)-THEA(ISEL(J))) * PI2
186:      if ( DEL.le.CONT ) then
187:      OMMAX   = PI2*COSL(J) + CONT
188:      OMAX    = MIN( OMMAX, PI2 )
189:      OMMIN   = PI2*COSL(J) - CONT
190:      OMIN    = MAX( OMMIN, -PI2 )
191:      WW(J)   = WW(J) / ( OMAX - OMIN )
192:      else
193:      WW(J)   = 0.0
194:      end if
195: 180 continue

```

```

196:      end if
197: C
198: C**** Free gas model
199: C
200:      if ( NFRGST.gt.NTELT2T ) then
201: C
202: C..... JFREE = 0 : FIRST TRIAL FOR SHORT COLLISION TIME APPROXIMATION
203: C      : BY USING THE SAME METHOD AS THAT FOR FREE GAS MODEL
204: C      JFREE = 0
205: C
206:      NF = NFRGST - NTELT2T
207: c      call RANU2( IRAND, R, NF*7, ICON )
208: C
209: c      NF2    = NF*2
210: c      NF3    = NF*3
211: c      NF4    = NF*4
212: c      NF5    = NF*5
213: c      NF6    = NF*6
214: C
215: C      WK2 : ATW
216: C      WK3 : ATW/KT*EEE
217: C      WK4 : ATW/KTeff*EEE WHEN EEE<=5eV
218: C..... WK5      : V(TARGET NUCLIDE) / V(NEUTRON) : XT
219: C      X2      : WK5**2
220: C      (WK6,WK7,WK8) : DIRECTION COSINE OF TARGET
221: C      WK9      : SQRT(1.+X2-2*XMU*XT)
222: C      IWK1(NN)  : IWK1(k)---> JJ
223: C
224: c      NN      = 0
225: C
226: c*VOCL LOOP,NOVREC
227: c      do 200 JJ = 1, NF
228: c      J       = JJ + NTELT2T
229: c      I       = ISEL(J)
230: c      WK2(JJ) = XLIB1(ICLN(I),1)
231: c      WK3(JJ) = XLIB1(ICLN(I),9)*EEEJ(J)
232: C      .... neglect short collision approximation
233: c      if ( EEEJ(J).le.5. ) then
234: c      WK4(JJ) = XLIB1(ICLN(I),11)*EEEJ(J)
235: c      else
236: c      WK4(JJ) = WK3(JJ)
237: c      end if
238: c      if ( (4.*R(JJ)*R(JJ)).gt.
239: c      &      (PI*WK4(JJ)*(1.-R(JJ))*(1.-R(JJ))) ) then
240: c      X2      = -LOG(R(NF+JJ)*R(NF2+JJ)) / WK4(JJ)
241: c      else
242: c      COS2    = COS(PIH*R(NF+JJ))
243: c      COS2    = COS2*COS2
244: c      X2      = -(LOG(R(NF2+JJ))+LOG(R(NF3+JJ))*COS2) / WK4(JJ)
245: c      end if
246: C
247: c      WK5(JJ) = SQRT(X2)
248: C
249: C..... TARGET DIRECTION SAMPLED ISOTROPICALLY
250: C
251: c      WK6(JJ) = 2.*R(NF4+JJ) - 1.0
252: c      SINT    = SQRT(1.-WK6(JJ)*WK6(JJ))
253: c      FAI     = PI2*R(NF5+JJ)
254: c      WK7(JJ) = SINT*COS(FAI)
255: c      WK8(JJ) = SINT*SIN(FAI)
256: c      XMU     = WK6(JJ)*A0(I) + WK7(JJ)*B0(I) + WK8(JJ)*C0(I)
257: C
258: c      WK9(JJ) = SQRT(1.+X2-2.*XMU*WK5(JJ))
259: C
260: c      if ( WK9(JJ).lt.R(NF6+JJ)*(1.+WK5(JJ)) ) then

```

src/mvp/nxtths.f

```

261: c          NN      = NN + 1
262: c          IWK1(NN) = JJ
263: c          end if
264: c 200      continue
265: c
266: c**** RESAMPLING FOR REJECTED PARTICLES
267: c
268: c 210      continue
269: c
270: c          if ( NN.gt.0 ) then
271: c
272: c          NFF      = NN
273: c          NN      = 0
274: c          call RANU2( IRAND, R, NFF*7, ICON )
275: c
276: c          NF2      = NFF*2
277: c          NF3      = NFF*3
278: c          NF4      = NFF*4
279: c          NF5      = NFF*5
280: c          NF6      = NFF*6
281: c
282: c*VOCL LOOP,NOVREC
283: c          do 220 K = 1, NFF
284: c              JJ      = IWK1(K)
285: c              J        = JJ + NTEL2T
286: c              AEKT      = WK4(JJ)
287: c              if ( ( 4.*R(K)*R(K)).gt.(PI*AEKT*(1.-R(K))*(1.-R(K)) ) )
288: c              &          then
289: c                  X2      = -LOG(R(NFF+K)*R(NF2+K)) /WK4(JJ)
290: c              else
291: c                  COS2      = COS(PIH*R(NFF+K))
292: c                  COS2      = COS2*COS2
293: c                  X2      = -(LOG(R(NF2+K))+LOG(R(NF3+K))*COS2) /AEKT
294: c              end if
295: c              WK5(JJ) = SQRT(X2)
296: c
297: c..... TARGET DIRECTION SAMPLED ISOTROPICALLYO
298: c
299: c              WK6(JJ) = 2.*R(NF4+K) - 1.0
300: c              SINT      = SQRT(1.-WK6(JJ)*WK6(JJ))
301: c              FAI        = PI2*R(NF5+K)
302: c              WK7(JJ) = SINT*COS(FAI)
303: c              WK8(JJ) = SINT*SIN(FAI)
304: c              I          = ISEL(J)
305: c              XMU        = WK6(JJ)*A0(I) + WK7(JJ)*B0(I) + WK8(JJ)*C0(I)
306: c
307: c              WK9(JJ) = SQRT(1.+X2-2.*XMU*WK5(JJ))
308: c
309: c              if ( WK9(JJ).lt.R(NF6+JJ)*(1.+WK5(JJ)) ) then
310: c                  NN      = NN + 1
311: c                  IWK1(NN) = JJ
312: c              end if
313: c 220      continue
314: c
315: c          go to 210
316: c
317: c      end if
318: c
319: c      do 230 JJ = 1, NF
320: c          J        = JJ + NTEL2T
321: c          I          = ISEL(J)
322: c          AX        = WK2(JJ) * WK5(JJ)
323: c          ..... effective direction
324: c          WK6(JJ) = A0(I) + AX*WK6(JJ)
325: c          WK7(JJ) = B0(I) + AX*WK7(JJ)

```

```

326: c          WK8(JJ) = C0(I) + AX*WK8(JJ)
327: c          WK5(JJ) = WK6(JJ)**2 + WK7(JJ)**2 + WK8(JJ)*2
328: c          SS      = SQRT(WK5(JJ))
329: c          if ( SS.gt.0. ) then
330: c              WK6(JJ) = WK6(JJ) / SS
331: c              WK7(JJ) = WK7(JJ) / SS
332: c              WK8(JJ) = WK8(JJ) / SS
333: c          .... effective mass
334: c          WK2(JJ) = WK2(JJ) * WK9(JJ) / SS
335: c          end if
336: c 230      continue
337: c
338: c          N1      = 0
339: c          *VOCL LOOP,NOVREC
340: c          do 240 J = NTEL2T+1, NFRGST
341: c              I          = ISEL(J)
342: c              .... effective target mass
343: c              ATW        = EMIU(I)
344: c              GE          = ATW
345: c              GE2        = GE*GE
346: c              if ( GE2.le.0.0 ) then
347: c                  COSMIN = 2.0
348: c              else if ( GE2.le.1.0 ) then
349: c                  COSMIN = SQRT( 1.0 - GE2 )
350: c              else
351: c                  COSMIN = -2.0
352: c              end if
353: c
354: c              COSLJ      = COSL(J)
355: c
356: c              if ( COSLJ.gt.COSMIN ) then
357: c                  N1      = N1 + 1
358: c                  IWK1(N1) = J
359: c                  COS2M    = COSLJ*COSLJ - 1.0
360: c                  XX        = COS2M + GE2
361: c                  WK1(N1) = SQRT(XX)
362: c                  WK2(N1) = GE
363: c                  WK3(N1) = COSMIN
364: c                  CSQRD    = WK1(N1)*COSLJ
365: c                  WK4(N1) = MIN(1.0D0, MAX(-1.0D0,
366: c                  &                      (COS2M + CSQRD) / GE ) )
367: c                  EEJ(J) = EEJ(J) / (ATW+1.)/(ATW+1.)
368: c
369: c                  WK5(N1) = -10.0
370: c                  if ( GE.lt.1.0 .and. CSQRD.ne.0.0 )
371: c                  &                      WK5(N1) = MIN(1.0D0, MAX(-1.0D0,
372: c                  &                      (COS2M - CSQRD) / GE ) )
373: c              else
374: c                  WW(J) = 0.0
375: c              end if
376: c          240      continue
377: c
378: c          N2      = 0
379: c          *VOCL LOOP,NOVREC
380: c          do 250 JJ = 1, N1
381: c              J        = IWK1(JJ)
382: c              WK6(JJ) = (COSL(J)+WK1(JJ))**2
383: c              if ( WK5(JJ).ne.-10. ) then
384: c                  N2      = N2 + 1
385: c                  IWK2(N2) = JJ
386: c                  WK5(N2) = WK5(JJ)
387: c                  WK7(N2) = (COSL(J)-WK1(JJ))**2
388: c              end if
389: c          250      continue
390: c

```

src/mvp/nxtths.f

```
391:          call RANU2( IRAND, R, N2, ICON )
392: C
393: *VOCL LOOP,NOVREC
394:       do 260 K = 1, N2
395:         JJ      = IWK2(K)
396:         J        = IWK1(JJ)
397: C
398:         if ( COSL(J).lt.WK3(JJ)+PEPS ) then
399:           WK6(JJ) = 2.*(WK3(JJ)+PEPS)*SQRT(1.+2.*WK3(JJ)/PEPS)
400:           WK1(JJ) = 1.0
401:           WK7(K)  = WK6(JJ) / 2.0
402:         else
403:           WK6(JJ) = WK6(JJ) + WK7(K)
404:         end if
405:         if ( WK6(JJ)*R(K).lt.WK7(K) ) then
406:           WK4(JJ) = WK5(K)
407:         end if
408:       260 continue
409: C
410: *VOCL LOOP,NOVREC
411:       do 270 JJ = 1, N1
412:         J      = IWK1(JJ)
413:         EEEJ(J) = EEEJ(J)*(1.+WK2(JJ)*(WK2(JJ)+2.*WK4(JJ)))
414:         WW(J)   = WW(J)*WK6(JJ)*PI4I/WK1(JJ)/WK2(JJ)
415:       270 continue
416: C
417:       end if
418: C
419: C**** scattering with heavy nuclide
420: C
421:       if ( NWOECT.gt.NFRGST ) then
422:         do 280 J = NFRGST+1, NWOECT
423:           I      = ISEL(J)
424:           ATW    = XLIB1(ICLN(I),1)
425:           P1     = COSL(J)*COSL(J) - 1.0
426:           P2     = SQRT( P1 + ATW*ATW )
427:           WW(J)  = WW(J) * PI4I *
428:             &      (COSL(J)+P2)**2 / (ATW*P2)
429:           if ( EEEJ(J).gt.0.625 ) then
430:             XCM    = (P1+COSL(J)*P2) / ATW
431:             P3     = 1.0 / (ATW+1.0)**2
432:             EEEJ(J) = EEEJ(J) * (1.0+ATW*(ATW+2.*XCM)) * P3
433:           end if
434:         280 continue
435:       end if
436: C
437:       return
438: end
```

src/mvp/option.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine OPTION( MLIMIT )
3: C=====
4: C PURPOSE: READ OPTION PARAMETERS
5: C CALLED IN:  INTRO
6: C CALLS      :
7: C=====
8:   include '../shared/INC/_ARRAY'
9:   include '../shared/INC/_IUNIT'
10:  include '../shared/INC/_TASKDT'
11:  include 'INC/_FLAGS'
12: C
13:  include 'INC/_KPIDS'
14: C
15: C ---- PRINT OUT OF COMMON /FLAGS/ IS SUPPRESSED. -----
16: C
17:  character LINE*72, CH*1, PFLAG*72
18:  parameter( MAXPRT = 20 )
19:  integer IT(MAXPRT)
20: C/#IF INTEGER8
21: *   integer*8 IT8(MAXPRT)
22: C/#ELSE
23:   integer IT8(MAXPRT)
24: C/#ENDIF
25:   character CPARAM*72 ! pert
26: C
27:   include 'INC/_REACC'
28: C
29: C-----
30: C
31: C/#IF DYNAMIC
32: C
33: C ... reset limit size of array A in common /ARRAY/ here,
34: C   when dynamic allocation of memory is available.
35: C
36:   LIMIT = MLIMIT
37: C/# IF PARA( CRAY SX* )
38: *   MLMTL = LIMITL
39: C/# ENDIF
40: C/#ENDIF
41: C
42:   NTASK0 = NTASK
43: C
44: C-----
45: C ..... READ NEW RECORD .....
46: C-----
47: C
48:   100 call FREADB( LINE, NLEN, IEND )
49:   if ( IEND.ne.0 ) go to 190
50:   JJ = 1
51: C-----
52: C ..... A 'NO-' ON HEAD MEANS NOT TO USE THIS OPTION .....
53: C-----
54:   IS = 1
55:   IE = NLEN
56:   if ( NLEN.gt.3.and.LINE(IS:IS+2).eq.'NO-' ) then
57:     JJ = 0
58:     IS = IS + 3
59:   end if
60: C   IEI = IE + 1
61: C
62:   NHYP = 0
63:   do 110 I = IS, IE
64:     if ( LINE(I:I).eq.'-' ) NHYP = NHYP + 1
65:   110 continue

```

```

66: C
67: C-----
68: C ..... CHECK OPTIONS WITH NUMERICAL DATA ...   XXXX-YYY(N)
69: C-----
70: C****IBR = INDEX(LINE(IS:IE),'(')
71: C ... GET NEXT NON-BLANK CHARACTER IN THE INPUT FILE ....
72: C   .... ( DETECT CHARACTER OTHER THAN ' ' )
73: C
74:   IBR = 0
75:   call FPROBE( IBR, ' ', CH )
76:   if ( IBR.ne.0.and.CH.ne.'(' ) IBR = 0
77: C
78: C   .... IBRU IS SET TO 0 IF NUMERICAL DATA ARE USED ....
79: C
80:   IBRU = 1
81: C
82: C**** IF(IBR.NE.0) IEI = IBR
83: C
84: C-----
85: C   SET FLAGS
86: C-----
87: C
88: C ===== RESTART
89: C
90:   if ( NHYP.eq.0.and.IMATCH('REST*',LINE(IS:IE)).eq.1 ) then
91:     JREST = JJ
92: C
93: C ===== AUTO-RESTART
94: C
95:   else if ( NHYP.eq.1.and.IMATCH('AUTO-REST*',LINE(IS:IE)).eq.1 )
96: &   then
97:     JARST = JJ
98: C
99: C ===== NEUTRON
100: C
101:   else if ( NHYP.eq.0.and.IMATCH('NEUT*',LINE(IS:IE)).eq.1 ) then
102:     JNEUT = JJ
103:     JKPARI(KPNEUT) = JNEUT
104: C
105: C ===== PHOTON
106: C
107:   else if ( NHYP.eq.0.and.IMATCH('PHOT*',LINE(IS:IE)).eq.1 ) then
108:     JPHOT = JJ
109:     JKPARI(KPPHOT) = JPHOT
110: C
111:   if ( IBR.ne.0 ) then
112:     PFLAG = ' '
113:   120 call CHREAD( ' ', PFLAG, ' ', IJ, ITERM, IEND )
114:   if ( IEND.ne.0 ) then
115:     write(IMG,*)
116: &   'XXX UNEXPECTED END OF DATA IN OPTION BLOCK'
117:     call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
118:     stop 888
119:   end if
120: C
121:   if ( IJ.ne.0 ) then
122: C
123: C   ..... "GAMMA RAY" MODEL FOR PHOTON .....
124: C
125:     if ( PFLAG(1:1).eq.'G' ) then
126:       JGAMM = 1
127:     else if ( PFLAG(1:1).eq.'P' ) then
128:       JGAMM = 0
129:     else
130:       write(IMG,*)

```

src/mvp/option.f

```

131:      &                'XXX unsupported sub-option in ',
132:      &                'PHOTON (', PFLAG(:IJ), ' )'
133:      call CNTERR( 'FATAL' )
134:      end if
135:      end if
136:      if ( ITERM.eq.0 ) go to 120
137:      IBRU      = 0
138:      end if
139: C
140: C ===== PHOTO-NUCLEAR
141: C
142:      else if ( NHYP.eq.1.and.IMATCH('PHOT*-NUCL*',LINE(IS:IE)).eq.1 )
143:      &      then
144:      JPHNU      = JJ
145: C
146: C ===== PARTICLE
147: C
148:      else if ( NHYP.eq.0.and.IMATCH('PART*',LINE(IS:IE)).eq.1 ) then
149: C
150:      if ( IBR.ne.0 ) then
151:      PFLAG      = ' '
152: 130      call CHREAD( ' ', PFLAG, ' )', IJ, ITERM, IEND )
153:      if ( IEND.ne.0 ) then
154:      write(IMG,*)
155:      &      'XXX Unexpected end of data in option block'
156:      call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
157:      stop 888
158:      end if
159:
160:      if ( IJ.ne.0 ) then
161: C
162: C ..... check supported particle or not .....
163: C
164:      call KPSYMB( PFLAG(:IJ), '>', IKPID, 0 )
165: C
166:      if ( IKPID.eq.0 ) then
167:      write(IMG,*)
168:      &      'XXX unsupported particle in PARTICLE()',
169:      &      ' <', PFLAG(:IJ), '>'
170:      call CNTERR( 'FATAL' )
171:      else
172:      JKPARIKPID)      = JJ
173:      if ( IKPID.eq.KPNEUT ) then
174:      JNEUT      = JJ
175:      end if
176:      if ( IKPID.eq.KPPHOT ) then
177:      JPHOT      = JJ
178:      end if
179:      end if
180:      end if
181:      if ( ITERM.eq.0 ) go to 130
182:      IBRU      = 0
183:      end if
184: C
185: C ===== BREMSSTRAHLUNG
186: C
187:      else if ( NHYP.eq.0.and.IMATCH('BREM*',LINE(IS:IE)).eq.1 ) then
188:      JBREM      = JJ
189: C
190: C ===== IMAGINARY-PARTICLE
191: C
192:      else if ( NHYP.eq.1.and.IMATCH('IMAG*-PART*',LINE(IS:IE)).eq.1 )
193:      &      then
194:      JIMAG      = JJ
195: C

```

```

196: C ===== TIME-DEPENDENT
197: C
198:      else if ( NHYP.eq.1.and.IMATCH('TIME*-DEPE*',LINE(IS:IE)).eq.1 )
199:      &      then
200:      JTIME      = JJ
201: C
202: C ===== PERIODIC-TIME
203: C
204:      else if ( NHYP.eq.1.and.IMATCH('PERI*-TIME*',LINE(IS:IE)).eq.1 )
205:      &      then
206:      JPTIM      = JJ
207: C
208: C ===== DELTA-TRACKING
209: C
210:      else if ( NHYP.eq.1.and.IMATCH('DELT*-TRAC*',LINE(IS:IE)).eq.1 )
211:      &      then
212:      JDELT      = JJ
213: C
214: C ===== RELATIVE-WEIGHT
215: C
216:      else if ( NHYP.eq.1.and.IMATCH('RELA*-WEIG*',LINE(IS:IE)).eq.1 )
217:      &      then
218:      JRWVR      = JJ
219: C
220: C ===== FISSION
221: C
222:      else if ( NHYP.eq.0.and.IMATCH('FISS*',LINE(IS:IE)).eq.1 ) then
223:      JFISS      = JJ
224: C
225: C ===== ANALOG-FISSION
226: C
227:      else if ( NHYP.eq.1.and.IMATCH('ANAL*-FISS*',LINE(IS:IE)).eq.1 )
228:      &      then
229:      JANLF      = JJ
230: C
231: C ===== FISSION-MULTIPLICITY
232: C
233:      else if ( NHYP.eq.1.and.IMATCH('FISS*-MULT*',LINE(IS:IE)).eq.1 )
234:      &      then
235: C
236:      if ( JJ.eq.0 ) then
237:      JFMUL      = 0
238:      if ( IBR.ne.0 ) then
239:      call DMREAD( ' ', NA, NB, IRET )
240:      IBRU      = 0
241:      end if
242:      else if ( IBR.ne.0 ) then
243:      PFLAG      = ' '
244:      call CHREAD( ' ', PFLAG, ' )', IJ, ITERM, IEND )
245:      if ( IEND.ne.0 ) then
246:      write(IMG,*)
247:      &      'XXX Unexpected end of data in option block'
248:      call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
249:      stop 888
250:      end if
251:
252:      if ( IJ.ne.0 ) then
253: C
254: C ..... Select fission-nu value .....
255: C
256:      if ( PFLAG(:IJ).eq.'TOTAL' ) then
257:      JFMUL      = 0
258:      else if ( PFLAG(:IJ).eq.'PROMPT' ) then
259:      JFMUL      = 1
260:      else

```

src/mvp/option.f

```

261: CM          write(IMG, '(lx,a,a/)' )
262: CM &        '!!! Cannot identify sub-option for FISS-MULT.',
263: CM &        ' "TOTAL" is used.'
264: CM          JFMUL = 0
265: CM          call CNTERR( 'WARNING' )
266:          write(IMG,*)
267:          &        'XXX unsupported sub-option in ',
268:          &        'FISS-MULT (', PFLAG(:IJ), ', )'
269:          call CNTERR( 'FATAL' )
270:          end if
271:          else
272:          write(IMG,*)
273:          &        'XXX no sub-option for FISS-MULT'
274:          call CNTERR( 'FATAL' )
275:          end if
276:          if ( ITERM.ne.0 ) IBRU = 0
277:          else
278:          write(IMG,*)
279:          &        'XXX no sub-option for FISS-MULT'
280:          call CNTERR( 'FATAL' )
281:          end if
282: C
283: C ===== DELAYED-NEUTRON
284: C
285:          else if ( NHYP.eq.1.and.IMATCH('DELA*NEUT*',LINE(IS:IE)).eq.1 )
286:          &        then
287:          JDLYN = JJ
288: C
289: C ===== DOPPLER-SCATTERING( )
290: C
291:          else if ( NHYP.eq.1.and.IMATCH('DOPP*-SCAT*',LINE(IS:IE)).eq.1 )
292:          &        then
293: C
294:          if ( JJ.eq.0 ) then
295:          JEXDP = 0
296:          if ( IBR.ne.0 ) then
297:          call DMREAD( ' ', NA, NB, IRET )
298:          IBRU = 0
299:          end if
300:          else if ( IBR.ne.0 ) then
301:          PFLAG = ' '
302:          call CHREAD( ' ', PFLAG, ' ', IJ, ITERM, IEND )
303:          if ( IEND.ne.0 ) then
304:          write(IMG,*)
305:          &        'XXX Unexpected end of data in option block'
306:          call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
307:          stop 888
308:          end if
309:
310:          if ( IJ.ne.0 ) then
311: C
312: C ..... Select method to treat elastic scattering in resonance region
313: C
314:          if ( PFLAG(:IJ).eq.'EXACT+' ) then
315:          JEXDP = 3
316:          else if ( PFLAG(:IJ).eq.'EXACT' ) then
317:          JEXDP = 2
318:          else if ( PFLAG(:IJ).eq.'NORES' ) then
319:          JEXDP = 1
320:          else if ( PFLAG(:IJ).eq.'NONE' ) then
321:          JEXDP = 0
322:          else
323: CM          write(IMG, '(lx,a,a/)' )
324: CM &        '!!! Cannot identify sub-option for DOPP-SCAT',
325: CM &        ' "NONE" is used.'

```

```

326: CM          JEXDP = 0
327: CM          call CNTERR( 'WARNING' )
328:          write(IMG,*)
329:          &        'XXX unsupported sub-option in ',
330:          &        'DOPP-SCAT (', PFLAG(:IJ), ', )'
331:          call CNTERR( 'FATAL' )
332:          end if
333:          else
334:          write(IMG,*)
335:          &        'XXX no sub-option for DOPP-SCAT'
336:          call CNTERR( 'FATAL' )
337:          end if
338:          if ( ITERM.ne.0 ) IBRU = 0
339:          else
340:          write(IMG,*)
341:          &        'XXX no sub-option for DOPP-SCAT'
342:          call CNTERR( 'FATAL' )
343:          end if
344: C
345: C ===== EIGEN-VALUE
346: C
347:          else if ( NHYP.eq.1.and.IMATCH('EIGE*-VALU*',LINE(IS:IE)).eq.1 )
348:          &        then
349:          JEIGN = JJ
350:          JFIXD = 1 - JJ
351: C
352: C ===== FIXED-SOURCE
353: C
354:          else if ( NHYP.eq.1.and.IMATCH('FIXE*-SOUR*',LINE(IS:IE)).eq.1 )
355:          &        then
356:          JFIXD = JJ
357:          JEIGN = 1 - JJ
358: C
359: C ===== RUSSIAN-ROULETTE
360: C
361:          else if ( NHYP.eq.1.and.IMATCH('RUSS*-ROUL*',LINE(IS:IE)).eq.1 )
362:          &        then
363:          JRRLT = JJ
364:          if ( JRRLT.eq.1 ) JWWND = 0
365: C
366: C ===== IMPORTANCE
367: C
368:          else if ( NHYP.eq.0.and.IMATCH('IMPO*',LINE(IS:IE)).eq.1 ) then
369:          JIMPT = JJ
370:          if ( JIMPT.eq.1 ) JWWND = 0
371: C
372: C ===== WEIGHT-WINDOW
373: C
374:          else if ( NHYP.eq.1.and.IMATCH('WEIG*-WIND*',LINE(IS:IE)).eq.1 )
375:          &        then
376:          JWWND = JJ
377:          if ( JWWND.eq.1 ) JIMPT = 0
378:          if ( JWWND.eq.1 ) JRRLT = 0
379: C
380: C ===== PATH-STRETCHING
381: C
382:          else if ( NHYP.eq.1.and.IMATCH('PATH*-STRE*',LINE(IS:IE)).eq.1 )
383:          &        then
384:          JPSTR = JJ
385: C
386: C ===== FORCED-COLLISION
387: C
388:          else if ( NHYP.eq.1.and.IMATCH('FORC*-COLL*',LINE(IS:IE)).eq.1 )
389:          &        then
390:          JFCOL = JJ

```


src/mvp/option.f

```

391: C
392: C ===== COLLISION-LESS
393: C
394:   else if ( NHYP.eq.1.and.IMATCH('COLL*-LESS',LINE(IS:IE)).eq.1 )
395:   &       then
396:       JNCOL = JJ
397: C
398: C ===== SOURCE-BIASING
399: C
400:   else if ( NHYP.eq.1.and.IMATCH('SOUR*-BIAS*',LINE(IS:IE)).eq.1 )
401:   &       then
402:       JBIAS = JJ
403: C
404: C ===== RESPONSE
405: C
406:   else if ( NHYP.eq.0.and.IMATCH('RESP*',LINE(IS:IE)).eq.1 ) then
407:       JRESP = JJ
408: C
409: C ===== MONITOR
410: C
411:   else if ( NHYP.eq.0.and.IMATCH('MONI*',LINE(IS:IE)).eq.1 ) then
412:       JMNTR = JJ
413: C
414: C ===== VP-MONITOR
415: C
416:   else if ( NHYP.eq.1.and.IMATCH('VP-MONI*',LINE(IS:IE)).eq.1 ) then
417:       JVMNT = JJ
418: C
419: C ===== DEBUG-PRINT
420: C
421:   else if ( NHYP.eq.1.and.IMATCH('DEBUG-PRIN*',LINE(IS:IE)).eq.1 )
422:   &       then
423:       if ( JJ.eq.0 ) then
424:           do 140 I = 1, MAXJDB
425:               JDEBG(I) = 0
426:           continue
427:       if ( IBR.ne.0 ) then
428:           call DMREAD( ' ', NA, NB, IRET )
429:           IBRU = 0
430:       end if
431:   else if ( IBR.ne.0 ) then
432:       call I4READ( 'DEBUG-PRINT', JDEBG, NA, -MAXJDB, IERR )
433:       if ( IERR.ne.0 ) then
434:           write(IMG,*)
435:           &       'XXX Invalid parameter for option "DEBUG-PRINT".'
436:           call CNTERR( 'FATAL' )
437:       end if
438:       IBRU = 0
439:   else
440:       JDEBG(1) = JJ
441:   end if
442: C
443: C ===== VPP-SPECIFIC
444: C
445:   else if ( NHYP.eq.1.and.IMATCH('VPP-SPEC*',LINE(IS:IE)).eq.1 )
446:   &       then
447:       if ( JJ.eq.0 ) then
448:           do 150 I = 1, MXJVP
449:               JVPPS(I) = 0
450:           continue
451:       if ( IBR.ne.0 ) then
452:           call DMREAD( ' ', NA, NB, IRET )
453:           IBRU = 0
454:       end if
455:   else if ( IBR.ne.0 ) then

```

```

456:       call I4READ( 'VPP-SPECIFIC', JVPPS, NA, -MXJVP, IERR )
457:       if ( IERR.ne.0 ) then
458:           write(IMG,*)
459:           &       'XXX Invalid parameter for option "VPP-SPECIFIC".'
460:           call CNTERR( 'FATAL' )
461:       end if
462:       IBRU = 0
463:   else
464:       JVPPS(1) = JJ
465:   end if
466: C
467: C ===== PICTURE
468: C
469:   else if ( NHYP.eq.0.and.IMATCH('PICT*',LINE(IS:IE)).eq.1 ) then
470:       JPIC = JJ
471: C
472: C ===== ALL-ZONE (1988/2/25)
473: C OR EVENT-SELECTION (1989/2/14)
474: C
475:   else if ( NHYP.eq.1.and.IMATCH('ALL-ZONE',LINE(IS:IE)).eq.1
476:   &       .or. IMATCH('EVEN*-SELE*',LINE(IS:IE)).eq.1 ) then
477:       JALLZ = JJ
478:       JONEZ = 1 - JJ
479: C
480: C ===== ONE-ZONE (1988/2/25)
481: C OR ZONE-SELECTION (1989/2/14)
482: C
483:   else if ( NHYP.eq.1.and.IMATCH('ONE-ZONE',LINE(IS:IE)).eq.1
484:   &       .or. IMATCH('ZONE-SELE*',LINE(IS:IE)).eq.1 ) then
485:       JONEZ = JJ
486:       JALLZ = 1 - JJ
487: C
488: C ===== LATTICE (1988/10/19)
489: C
490:   else if ( NHYP.eq.0.and.IMATCH('LATT*',LINE(IS:IE)).eq.1 ) then
491:       JLATT = JJ
492: C
493: C ===== REPEATED-GEOMETRY
494: C ( ALIAS OF LATTICE )
495: C
496:   else if ( NHYP.eq.1.and.IMATCH('REPE*-GEOM*',LINE(IS:IE)).eq.1 )
497:   &       then
498:       JLATT = JJ
499: C
500: C ===== FREE-LATTICE-FRAME (1988/12/28)
501: C
502:   else if ( NHYP.eq.2
503:   &       .and. IMATCH('FREE*-LATT*-FRAM*',LINE(IS:IE)).eq.1 ) then
504:       JFISX = JJ
505: C
506: C ===== TALLY-LATTICE (1992/3/18)
507: C
508:   else if ( NHYP.eq.1.and.IMATCH('TALL*-LATT*',LINE(IS:IE)).eq.1 )
509:   &       then
510:       JTLLT = JJ
511: C
512: C ===== UNIVERSE-DEPENDENT-TALLY
513: C FRAME-DEPENDENT-TALLY
514: C ( ALIAS OF TALLY-LATTICE )
515: C
516:   else if ( NHYP.eq.2.and.IMATCH('UNIV*-DEPE*-TALL*',LINE(IS:IE))
517:   &       .eq.1 .or. IMATCH('FRAM*-DEPE*-TALL*',LINE(IS:IE)).eq.1 )
518:   &       then
519:       JTLLT = JJ
520: C
521: C ===== FLUX-PRINT (1989/08/22)

```

src/mvp/option.f

```

521: C
522:   else if ( NHYP.eq.1.and.IMATCH('FLUX-PRIN*',LINE(IS:IE)).eq.1 )
523:     &       then
524:       JFPRT = JJ
525: C
526: C ===== EDIT-BY-TRACK-LENGTH
527: C
528:   else if ( NHYP.eq.3
529:     &       .and.IMATCH('EDIT-BY-TRAC*-LENG*',LINE(IS:IE)).eq.1 ) then
530:     JRTTR = JJ
531:     JRTCL = 1 - JJ
532: C
533: C ===== EDIT-BY-COLLISION
534: C
535:   else if ( NHYP.eq.2.and.IMATCH('EDIT-BY-COLL*',LINE(IS:IE)).eq.1 )
536:     &       then
537:     JRTCL = JJ
538:     JRTTR = 1 - JJ
539: C
540: C ===== EDIT-MICROSCOPIC-DATA
541: C
542:   else if ( NHYP.eq.2.and.IMATCH('EDIT-MICR*-DATA',LINE(IS:IE)).eq.1
543:     &       ) then
544:
545:     if ( JJ.eq.0 ) then
546:       do I = 1, 8
547:         JMICE(I) = 0
548:       end do
549:       if ( IBR.ne.0 ) then
550:         call DMREAD( ' ', NA, NB, IRET )
551:         IBRU = 0
552:       end if
553:       else if ( IBR.ne.0 ) then
554:
555:         call I4READ( 'EDIT-MICROSCOPIC-DATA', IT, NA, -1, IERR )
556:         if ( IERR.ne.0 ) then
557:           write(IMG,*)
558:           &       'XXX Invalid parameter for option "EDIT-MICRO-DATA".'
559:           call CNTERR( 'FATAL' )
560:         end if
561:         IBRU = 0
562: C
563:         if ( NA.gt.0 ) then
564:           IJJ = IT(1)
565:           do 160 I = 1, 8
566:             JMICE(I) = IJJ/(10**(8-I))
567:             IJJ = IJJ - 10**(8-I)*JMICE(I)
568:             JMICE(I) = JMICE(I)*JJ
569:             CM if ( JMICE(I).gt.4 ) JMICE(I) = 0
570:             CM if ( JMICE(I).gt.4 ) JMICE(I) = 4
571:           160 continue
572:           else
573:             write(IMG,*)
574:             &       'XXX no data for EDIT-MICRO-DATA'
575:             call CNTERR( 'FATAL' )
576:           end if
577:           else
578:             write(IMG,*)
579:             &       'XXX no data for EDIT-MICRO-DATA'
580:             call CNTERR( 'FATAL' )
581:           end if
582: C
583: C ===== EDIT-MACROSCOPIC-DATA
584: C
585:   else if ( NHYP.eq.2.and.IMATCH('EDIT-MACR*-DATA',LINE(IS:IE)).eq.1

```

```

586:   &       ) then
587:
588:     if ( JJ.eq.0 ) then
589:       do I = 1, 8
590:         JMACE(I) = 0
591:       end do
592:       if ( IBR.ne.0 ) then
593:         call DMREAD( ' ', NA, NB, IRET )
594:         IBRU = 0
595:       end if
596:       else if ( IBR.ne.0 ) then
597:
598:         call I4READ( 'EDIT-MACROSCOPIC-DATA', IT, NA, -1, IERR )
599:
600:         if ( IERR.ne.0 ) then
601:           write(IMG,*)
602:           &       'XXX Invalid parameter for option "EDIT-MACROSCOPIC-DATA".'
603:           call CNTERR( 'FATAL' )
604:         end if
605:         IBRU = 0
606: C
607:         if ( NA.gt.0 ) then
608:           IJJ = IT(1)
609:           do 170 I = 1, 8
610:             JMACE(I) = IJJ/(10**(8-I))
611:             IJJ = IJJ - 10**(8-I)*JMACE(I)
612:             CM JMACE(I) = JMACE(I)*JJ
613:             CM if ( JMACE(I).gt.4 ) JMACE(I) = 0
614:             if ( JMACE(I).gt.4 ) JMACE(I) = 4
615:           170 continue
616:           else
617:             write(IMG,*)
618:             &       'XXX no data for EDIT-MACROSCOPIC-DATA'
619:             call CNTERR( 'FATAL' )
620:           end if
621:           else
622:             write(IMG,*)
623:             &       'XXX no data for EDIT-MACROSCOPIC-DATA'
624:             call CNTERR( 'FATAL' )
625:           end if
626: C
627: C ===== SCATTERING-MATRIX
628: C
629:   else if ( NHYP.eq.1
630:     &       .and.IMATCH('SCAT*-MATR*',LINE(IS:IE)).eq.1 ) then
631:     JSCTM = JJ
632:     if ( JJ.eq.0 ) then
633:       if ( IBR.ne.0 ) then
634:         call DMREAD( ' ', NA, NB, IRET )
635:         IBRU = 0
636:       end if
637:       else if ( IBR.ne.0 ) then
638:         call I4READ( 'SCATTERING-MATRIX', IT, NA, -1, IERR )
639:         if ( IERR.ne.0 ) then
640:           write(IMG,*)
641:           &       'XXX Invalid parameter for option "SCATTERING-MATRIX".'
642:           call CNTERR( 'FATAL' )
643:         end if
644:         IBRU = 0
645:         if ( NA.gt.0 ) JSCTM = IT(1) + 1
646:       end if
647: C
648: C ===== SCATTERING-MUBAR
649: C
650:   else if ( NHYP.eq.1

```

src/mvp/option.f

```

651:      & .and.IMATCH('SCAT*-MUBA*',LINE(IS:IE)).eq.1 ) then
652:      JSCMU = JJ
653: C
654: C ===== ADAPTIVE-MICRO-CALCULATION
655: C
656:      else if ( NHYP.eq.2
657:      & .and.IMATCH('ADAP*-MICR*-CALC*',LINE(IS:IE)).eq.1 ) then
658:      JAMXC = JJ
659: C
660: C ===== PERTURBATION
661: C
662:      else if ( NHYP.eq.0.and.IMATCH('PERT*',LINE(IS:IE)).eq.1 ) then
663:      JPERT = JJ
664: C
665: C JPDEN : Flag of density or number-density perturbation
666: C JPTMP : Flag of temperature perturbation (currently not used)
667: C ... These flags must be defined after reading the PERT block
668: C To be modified in the future.
669: C
670:      JPDEN = 1
671:      JPTMP = 0
672: C
673: C ===== BETA-EFFECTIVE
674: C
675:      else if ( NHYP.eq.1.and.IMATCH('BETA-EFFE*',LINE(IS:IE)).eq.1 ) then
676:      & then
677:      JBEFF = JJ
678: C
679: C ===== PRINT-SUPPRESS
680: C
681:      else if ( NHYP.eq.1
682:      & .and.IMATCH('PRIN*-SUPP*',LINE(IS:IE)).eq.1 ) then
683:      if ( JJ.eq.0 ) then
684:      do I = 1, MAXPRT
685:      JPRTS(I) = 0
686:      end do
687:      if ( IBR.ne.0 ) then
688:      call DMREAD( ' ', NA, NB, IRET )
689:      IBRU = 0
690:      end if
691:      else if ( IBR.ne.0 ) then
692: C
693:      call I4READ( 'PRINT-SUPPRESS', IT, NA, -MAXPRT, IERR )
694:      if ( IERR.ne.0 ) then
695:      write(IMG, '(1x,a)')
696:      & 'XXX Invalid parameter for option "PRINT-SUPPRESS".'
697:      call CNTERR( 'FATAL' )
698:      end if
699:      IBRU = 0
700: C
701:      if ( NA.gt.0 ) then
702:      do 180 I = 1, NA
703:      if ( IT(I).ge.1.and.IT(I).le.MAXPRT ) JPRTS(IT(I)) = 1
704: 180 continue
705:      else
706:      write(IMG,*)
707:      & '!!! no data for PRINT-SUPPRESS'
708:      call CNTERR( 'WARNING' )
709:      end if
710:      else
711:      write(IMG,*)
712:      & '!!! no data for PRINT-SUPPRESS'
713:      call CNTERR( 'WARNING' )
714:      end if
715: C

```

```

716: C ===== RUN-MODE[( mode )]
717: C
718:      else if ( NHYP.eq.1.and.IMATCH('RUN-MODE*',LINE(IS:IE)).eq.1 )
719:      & then
720:      if ( JJ.eq.0 ) then
721:      JRUNM = 0
722:      if ( IBR.ne.0 ) then
723:      call DMREAD( ' ', NA, NB, IRET )
724:      IBRU = 0
725:      end if
726:      else if ( IBR.ne.0 ) then
727: C
728:      call I4READ( 'RUN-MODE', IT, NA, -1, IERR )
729:      if ( IERR.ne.0 ) then
730:      write(IMG,*)
731:      & 'XXX Invalid parameter for option "RUN-MODE".'
732:      call CNTERR( 'FATAL' )
733:      end if
734:      IBRU = 0
735:      if ( NA.gt.0 ) then
736:      JRUNM = IT(1)
737:      else
738:      write(IMG,*)
739:      & '!!! No data for RUN-MODE option.'
740:      call CNTERR( 'WARNING' )
741:      end if
742:      else
743:      JRUNM = 0
744:      write(IMG,*)
745:      & '!!! No data for RUN-MODE option.'
746:      call CNTERR( 'WARNING' )
747:      end if
748: C
749: C << OPTION FOR DEBUGGING >> CHECK FOR /ARRAY/
750: C
751: C ===== MEMORY-CHECK
752: C
753:      else if ( NHYP.eq.1
754:      & .and.IMATCH('MEMO*-CHEC*',LINE(IS:IE)).eq.1 ) then
755:      JMCHK = JJ
756: C
757: C .... SIZE OF DYNAMICALLY ALLOCATED MEMORY ...
758: C
759: C ===== DYNAMIC-MEMORY
760: C
761:      else if ( NHYP.eq.1
762:      & .and.IMATCH('DYNA*-MEMO*',LINE(IS:IE)).eq.1 ) then
763:      if ( JJ.eq.0 ) then
764: C/#IF DYNAMIC
765:      LIMIT = MLIMIT
766: C/# IF PARA( CRAY SX* )
767:      * LIMITL = MLMTL
768: C/# ENDIF
769: C/#ENDIF
770:      if ( IBR.ne.0 ) then
771:      call DMREAD( ' ', NA, NB, IRET )
772:      IBRU = 0
773:      end if
774:      else if ( IBR.ne.0 ) then
775:      IT8(1) = 0
776:      IT8(2) = 0
777:      call I8READ( 'DYNAMIC-MEMORY', IT8, NA, -MAXPRT, IERR )
778:      if ( IERR.ne.0 ) then
779:      write(IMG,*)
780:      & 'XXX Invalid parameter for option "DYNAMIC-MEMORY".'

```

src/mvp/option.f

```

781:          call CNTERR( 'FATAL' )
782:        end if
783:        IBRU   = 0
784: C/#IF DYNAMIC
785: C/# IF PARA( CRAY SX* )
786: *          if ( NA.gt.0 ) LIMIT   = IT8(1)
787: *          if ( NA.gt.1 ) LIMITL  = IT8(2)
788: C/# ELSE
789:          if ( NA.gt.0 ) LIMIT   = IT8(1) + IT8(2)
790: C/# ENDIF
791:          if ( NA.eq.0 ) then
792:            write(IMG,*)
793:            &      'XXX No size-parameter given for DYNAMIC-MEMORY option.'
794:            call CNTERR( 'FATAL' )
795:          end if
796: C/#ELSE
797: *          write(IMG,7000) LIMIT
798: *          format(/IX,'!!! DYNAMIC-MEMORY option is not '
799: *          &      'effective for this installation of the code.'
800: *          &      ' When you want to change memory size, please',
801: *          &      ' change "MAXMEM" parameter in the AAALOG routine and',
802: *          &      ' remake load-module.'/IX,' (current limit = ',I10,
803: *          &      ' words)')
804: *          call CNTERR( 'WARNING' )
805: C
806: C/# IF .NOT.PARA
807: *          if ( NA.gt.0.and.IT(1)+IT(2).gt.LIMIT ) then
808: *            write(IMG,*) '!!! Your demand for dynamic memory',
809: *            &      ' exceeds fixed limit of ', LIMIT, ' words of',
810: *            &      ' current load-module.'
811: *          end if
812: C/# ELSE
813: *          if ( NA.eq.1 ) then
814: *            write(IMG,*) '!!! It may lead to error stop before ',
815: *            &      'starting histories.'
816: *            call CNTERR( 'WARNING' )
817: *          end if
818: C/# ENDIF
819: C/#ENDIF
820:          else
821:            write(IMG,*)
822:            &      'XXX No size-parameter given for DYNAMIC-MEMORY option.'
823:            call CNTERR( 'FATAL' )
824:          end if
825: C
826: C      .... NUMBER OF TASKS FOR MULTI-TASKING MODE ...
827: C
828: C      ===== MULTI-TASK
829: C
830:          else if ( NHYP.eq.1
831:          &      .and.IMATCH('MULT*-TASK*',LINE(IS:IE)).eq.1 ) then
832:            if ( JJ.eq.0 ) then
833:              NTASK   = NTASK0
834:              if ( IBR.ne.0 ) then
835:                call DMREAD( ' ', NA, NB, IRET )
836:                IBRU   = 0
837:              end if
838:            else if ( IBR.ne.0 ) then
839:              IT(1)   = 0
840:              call I4READ( 'MULTI-TASK', IT, NA, -MAXPRT, IERR )
841:              if ( IERR.ne.0 ) then
842:                write(IMG,*)
843:                &      'XXX Invalid parameter for option "MULTI-TASKING".'
844:                call CNTERR( 'FATAL' )
845:              end if

```

```

846:          if ( NA.gt.0 ) then
847:            NTASK   = IT(1)
848:          else
849:            write(IMG,*)
850:            &      'XXX No task-number given for MULTI-TASK option.'
851:            call CNTERR( 'FATAL' )
852:          end if
853:          IBRU   = 0
854:        else
855:          write(IMG,*)
856:          &      'XXX No task-number given for MULTI-TASK option.'
857:          call CNTERR( 'FATAL' )
858:        end if
859: C
860: C      .... spawn "tasker" in multitask mode if necessary (only in PVM)
861: C
862: C      ===== SPAWN-TASKER
863: C
864:          else if ( NHYP.eq.1.and.IMATCH('SPAW*-TASK*',LINE(IS:IE)).eq.1 )
865:          &      then
866:            JTSKR   = JJ
867: C
868: C      ... run multitasking mode in a way to obtain reproducible result.
869: C
870: C      ===== REPRODUCIBLE-RUN
871: C
872:          else if ( NHYP.eq.1.and.IMATCH('REPR*-RUN',LINE(IS:IE)).eq.1 )
873:          &      then
874:            JREPR   = JJ
875: C
876: C      ... save source data on file
877: C      (currently data of last batch on eigenvalue problem)
878: C
879: C      ===== SOURCE-OUTPUT
880: C
881:          else if ( NHYP.eq.1.and.IMATCH('SOUR*-OUT*',LINE(IS:IE)).eq.1 )
882:          &      then
883:            JOSRC   = JJ
884: C
885: C      ... Open scratch I/O units if they are not opened ...
886: C
887: C      ===== OPEN-SCRATCH
888: C
889:          else if ( NHYP.eq.1.and.IMATCH('OPEN-SCRA*',LINE(IS:IE)).eq.1 )
890:          &      then
891:            JSCRT   = JJ
892: C
893: C      ... output restart file. A restart file is output by default,
894: C      so this option is used to suppress restart file output.
895: C
896: C      ===== RESTART-FILE
897: C
898:          else if ( NHYP.eq.1.and.IMATCH('REST*-FILE',LINE(IS:IE)).eq.1 )
899:          &      then
900:            JRSTF   = JJ
901: C
902: C      ===== SUBTASK-PRINT
903: C      (but this option should be controlled by environment variable
904: C      or command line!!)
905: C
906:          else if ( NHYP.eq.1.and.IMATCH('SUBT*-PRIN*',LINE(IS:IE)).eq.1 )
907:          &      then
908:            if ( JJ.eq.0 ) then
909:              JSTPRN(1) = 1
910: C/#IF PARA(SX* CRAY*)

```

src/mvp/option.f

```

911: *          JSTPRN(1) = 2
912: C/#ENDIF
913:          JSTPRN(2) = 2
914: C
915:          if ( IBR.ne.0 ) then
916:              call DMREAD( ' ', NA, NB, IRET )
917:              IBRU = 0
918:          end if
919:          else if ( IBR.ne.0 ) then
920:              IT(1) = 0
921:              call I4READ( 'SUBTASK-PRINT', IT, NA, -MAXPRT, IERR )
922:              if ( IERR.ne.0 ) then
923:                  write(IMG,*)
924:                  & 'XXX Invalid parameter for option "SUBTASK-PRINT".'
925:                  call CNTERR( 'FATAL' )
926:              end if
927:              if ( NA.gt.0 ) JSTPRN(1) = IT(1)
928:              if ( NA.gt.1 ) JSTPRN(2) = IT(2)
929:              if ( NA.eq.0 ) then
930:                  write(IMG,*)
931:                  & '!!! No data given to SUBTASK-PRINT() option.'
932:                  call CNTERR( 'WARNING' )
933:              end if
934:              IBRU = 0
935:          else
936:              write(IMG,*) '!!! No data given to SUBTASK-PRINT() option.'
937:              call CNTERR( 'WARNING' )
938:          end if
939: C
940: C ===== NUCLIDE-WISE-MACRO-XSEC
941: C
942:          else if ( NHYP.eq.3
943:              & .and.IMATCH('NUCL*-WISE-MACRO*-XSEC',LINE(IS:IE)).eq.1 ) then
944:              JNWMX = JJ
945: C
946: C ===== TIME-LIST
947: C
948:          else if ( NHYP.eq.1.and.IMATCH('TIME-LIST',LINE(IS:IE)).eq.1 )
949:              & then
950:              JTLST = JJ
951:          else
952: C
953: C ..... ERROR .....
954: C
955:              write(IMG,('(1X,'XXX Invalid option: ',A)') LINE(IS:IE)
956:              call CNTERR( 'FATAL' )
957:          end if
958: C
959: C ..... SKIP UNUSED NUMERIC PARAMETERS ....
960: C
961:          if ( IBR.ne.0.and.IBRU.ne.0 ) then
962:              call DMREAD( ' ', NA, NB, IRET )
963:              write(IMG,7020) LINE(IS:IE)
964: 7020 format(1X,'!!! Numeric parameters are given but not',
965:              & ' effective for option <',A,'>')
966:              call CNTERR( 'WARNING' )
967:          end if
968: C
969:          go to 100
970: C
971: 190 continue
972: C
973: C ..... ERROR OR WARNING .....
974: C          AND ADJUSTMENT OF FLAGS
975: C

```

```

976: C
977:          if ( JEIGN.ne.0.and.JFISS.eq.0 ) then
978:              write(IPR,*)
979:              & '<<MESSAGE>> "FISSION" option is set active because ',
980:              & 'you are going to solve an eigenvalue problem.'
981:              JFISS = 1
982:              call CNTERR( 'MESSAGE' )
983:          end if
984: C
985:          if ( JTLT.ne.0.and.JLATT.eq.0 ) then
986:              write(IMG,('(1X,a,a)')
987:              & '!!! "TALLY-LATTICE" or "FRAME-DEPENDENT-TALLY" ',
988:              & 'option needs "LATTICE" option.'
989:              JTLT = 0
990:              call CNTERR( 'WARNING' )
991:          end if
992: C
993: C/#IF PARA
994: C
995:          if ( JREPR.eq.0.and.JEIGN.ne.0 ) then
996:              write(IMG,('(1X,a,a)')
997:              & '!!! Eigenvalue problems with the current multitasking ',
998:              & 'version are always REPRODUCIBLE.'
999:              JREPR = 1
1000:              call CNTERR( 'WARNING' )
1001:          end if
1002: C
1003: C/#ELSE
1004: C
1005:          if ( JREPR.eq.0 ) then
1006:              write(IMG,('(1X,a,a,1X,a)')
1007:              & '!!! In single task version, the result is always',
1008:              & 'reproducible.',
1009:              & 'so "NO-REPRODUCIBLE-RUN" has no meaning.'
1010:              JREPR = 1
1011:              call CNTERR( 'WARNING' )
1012:          end if
1013: C
1014: C/#ENDIF
1015: C
1016:          if ( JPHNU.ne.0 ) then
1017:              if ( JNEUT.eq.0.or.JPHOT.eq.0 ) then
1018:                  write(IMG,('(1X,A)')
1019:                  & 'XXX(option) No neutron or photon option was used.'
1020:                  call CNTERR( 'FATAL' )
1021:              endif
1022:          end if
1023: C
1024:          if ( JPHOT.eq.0.and.JBREM.eq.1 ) JBREM = 0
1025: C
1026:          if ( JFPRT.eq.1 ) JPRTS(1) = 0
1027:          if ( JPRTS(17).eq.1 ) then
1028:              do 200 I = 1, 8
1029:                  if ( JMICE(I).eq.4 ) then
1030:                      JMICE(I) = 2
1031:                  else if ( JMICE(I).eq.3 ) then
1032:                      JMICE(I) = 1
1033:                  end if
1034: 200 continue
1035:          end if
1036:          if ( JPRTS(18).eq.1 ) then
1037:              do 210 I = 1, 8
1038:                  if ( JMICE(I).eq.4 ) then
1039:                      JMICE(I) = 3
1040:                  else if ( JMICE(I).eq.2 ) then

```

src/mvp/option.f

```

1041:          JMICE(I)      = 1
1042:        end if
1043: 210    continue
1044:    end if
1045:    if ( JPRTS(19).eq.1 ) then
1046:      do 220 I = 1, 8
1047:        if ( JMACE(I).eq.4 ) then
1048:          JMACE(I)      = 2
1049:        else if ( JMACE(I).eq.3 ) then
1050:          JMACE(I)      = 1
1051:        end if
1052: 220    continue
1053:    end if
1054:    if ( JPRTS(20).eq.1 ) then
1055:      do 230 I = 1, 8
1056:        if ( JMACE(I).eq.4 ) then
1057:          JMACE(I)      = 3
1058:        else if ( JMACE(I).eq.2 ) then
1059:          JMACE(I)      = 1
1060:        end if
1061: 230    continue
1062:    end if
1063: C
1064:    if ( JANLF.eq.1 ) then
1065:      JFISS = 1
1066:      if ( JFIXD.eq.0 ) then
1067:        write(IMG,'(1x,a,a)') '!!! "ANALOG-FISSION" option needs ',
1068:      &      ' "FIXED-SOURCE" option.'
1069:      JFIXD = 1
1070:      JEIGN = 0
1071:      call CNTERR( 'WARNING' )
1072:    end if
1073:  end if
1074:
1075:    if ( JSCTM.ge.1 ) then
1076:      if ( JMICE(4).eq.0.and.JMACE(4).eq.0.and.
1077:      &      JMICE(6).eq.0.and.JMACE(6).eq.0.and.
1078:      &      JMICE(7).eq.0.and.JMACE(7).eq.0 ) then
1079:        write(IMG,'(1x,a,a,a/1x,a)') '!!! "SCATTERING-MATRIX" ',
1080:      &      'option needs the request of elastic, inelastic ',
1081:      &      'scattering, or (n,2n) in "EDIT-MICROSCOPIC-DATA"',
1082:      &      ' or "EDIT-MACROSCOPIC-DATA" options.'
1083:      JSCTM = 0
1084:      call CNTERR( 'WARNING' )
1085:    end if
1086:  end if
1087:
1088:    if ( JSCMU.eq.1 ) then
1089:      if ( JMICE(4).eq.0.and.JMACE(4).eq.0.and.
1090:      &      JMICE(6).eq.0.and.JMACE(6).eq.0.and.
1091:      &      JMICE(7).eq.0.and.JMACE(7).eq.0 ) then
1092:        write(IMG,'(1x,a,a,a/1x,a)') '!!! "SCATTERING-MUBAR" ',
1093:      &      'option needs the request of elastic, inelastic ',
1094:      &      'scattering, or (n,2n) in "EDIT-MICROSCOPIC-DATA"',
1095:      &      ' or "EDIT-MACROSCOPIC-DATA" options.'
1096:      JSCMU = 0
1097:      call CNTERR( 'WARNING' )
1098:    end if
1099:  end if
1100:
1101:    if(JSCTM.ge.2) then
1102:      if(JSCMU.eq.0) then
1103:        write(IMG,'(1x,a,a)')
1104:      &      '<<MESSAGE>> "SCATTERING-MUBAR" option is set active ',
1105:      &      'to calculate higher-order scattering moments.'

```

```

1106:          JSCMU = 1
1107:          call CNTERR('MESSAGE')
1108:        end if
1109:      end if
1110: C
1111: C ..... PRINT OUT OPTION PARAMETERS
1112: C
1113:       write(IPR,7040)
1114: 7040 format(/1X,'      OPTION PARAMETERS      (1/0 = ON/OFF)')//
1115:      & '      FLAG      KEY-WORD      ON/OFF      INCOMPATIBLE OPTIONS'//
1116:      & '      -----'
1117:      & '      '-----')
1118: 7060 format(1X,2X,A,T10,2X,A20,2X,I2,:4X,5(:' (' ,A,')'))
1119: 7080 format(1X,2X,A,T10,2X,A20,2X,I2,:4X,5(:' (' ,A,')'))
1120: C
1121:       write(IPR,7060) 'JREST', 'RESTART', ' ', JREST
1122:       write(IPR,7060) 'JARST', 'AUTO-RESTART', ' ', JARST
1123:       write(IPR,7060) 'JRSTF', 'RESTART-FILE', ' ', JRSTF
1124:       write(IPR,7060) 'JRUNM', 'RUN-MODE', ' ', JRUNM
1125:       write(IPR,7060) 'JNEUT', 'NEUTRON', ' ', JNEUT, 'JIMAG'
1126:       write(IPR,7060) 'JPHOT', 'PHOTON', ' ', JPHOT, 'JIMAG'
1127:       if ( JPHOT.ne.0 ) write(IPR,7060) 'JGAMM', 'GAMMA-RAY MODEL', ' ',
1128:      &      JGAMM
1129:       write(IPR,7060) 'JBREM', 'BREMSSTRAHLUNG', ' ', JBREM
1130:       write(IPR,7060) 'JPHNU', 'PHOTO-NUCLEAR', ' ', JPHNU ! photonuc
1131:       write(IPR,7060) 'JIMAG', 'IMAGINARY-PARTICLE', ' ', JIMAG, 'JNEUT',
1132:      &      'JPHOT'
1133:       write(IPR,7060) 'JNCOL', 'COLLISION-LESS', ' ', JNCOL
1134:       write(IPR,7060) 'JTIME', 'TIME-DEPENDENT', ' ', JTIME
1135:       write(IPR,7060) 'JPTIM', 'PERIODIC-TIME', ' ', JPTIM
1136: CC  WRITE(IPR,7200) 'JDELT', 'DELTA-TRACKING', ' ', JDELT
1137:       write(IPR,7060) 'JEIGN', 'EIGEN-VALUE', ' ', JEIGN, 'JFIXD'
1138:       write(IPR,7060) 'JFIXD', 'FIXED-SOURCE', ' ', JFIXD, 'JEIGN'
1139:       write(IPR,7060) 'JFISS', 'FISSION PROBLEM', ' ', JFISS
1140:       write(IPR,7060) 'JANLF', 'ANALOG-FISSION', ' ', JANLF
1141:       write(IPR,7060) 'JDLYN', 'DELAYED-NEUTRON', ' ', JDLYN
1142:       if(JFMUL.eq.0) then
1143:         write(IPR,'(1X,2X,A,T10,2X,A20,2X,a)')
1144:         &      'JFMUL', 'FISSION-MULTIPLICITY', 'TOTAL'
1145:       else if(JFMUL.eq.1) then
1146:         write(IPR,'(1X,2X,A,T10,2X,A20,2X,a)')
1147:         &      'JFMUL', 'FISSION-MULTIPLICITY', 'PROMPT'
1148:       end if
1149:       write(IPR,7060) 'JEXDP', 'DOPPLER-SCATTERING', ' ', JEXDP
1150:       write(IPR,7060) 'JOSRC', 'SOURCE-OUTPUT', ' ', JOSRC
1151:       write(IPR,7060) 'JRRLT', 'RUSSIAN-ROULETTE', ' ', JRRLT, 'JWWND'
1152:       write(IPR,7060) 'JIMPT', 'IMPORTANCE', ' ', JIMPT, 'JWWND'
1153:       write(IPR,7060) 'JWWND', 'WEIGHT-WINDOW', ' ', JWWND, 'JIMPT',
1154:      &      'JRRLT'
1155:       write(IPR,7060) 'JPSTR', 'PATH-STRETCHING', ' ', JPSTR
1156:       write(IPR,7060) 'JRWVR', 'RELATIVE-WEIGHT', ' ', JRWVR
1157: CCC  WRITE(IPR,7200) 'JFCOL', 'FORCED-COLLISION', ' ', JFCOL
1158: CCC  WRITE(IPR,7200) 'JBIAS', 'SOURCE-BIAS', ' ', JBIAS
1159:       write(IPR,7060) 'JRESP', 'RESPONSE', ' ', JRESP
1160:       write(IPR,7060) 'JMNTR', 'MONITOR', ' ', JMNTR
1161:       write(IPR,7060) 'JVMNT', 'VP-MONITOR', ' ', JVMNT
1162:       do 240 I = 1, MAXJDB
1163:         if ( I.eq.1 .or. JDEBG(I).ne.0 ) then
1164:           LINE = ' '
1165:           write(LINE,(' ' 'DEBUG-PRINT (' ,i2,')')') I
1166:           call CCOMP( LINE, LEN(LINE), LINE, IERR )
1167:           write(IPR,7060) 'JDEBG', LINE(:20), JDEBG(I)
1168:         end if
1169: 240 continue
1170:       write(IPR,7060) 'JSCRT', 'OPEN-SCRATCH', ' ', JSCRT

```

src/mvp/option.f

```
1171:      write(IPR,7060) 'JALLZ', 'EVENT-SELECTION      ', JALLZ, 'JONEZ'
1172:      write(IPR,7060) 'JONEZ', 'ZONE-SELECTION      ', JONEZ, 'JALLZ'
1173:      write(IPR,7060) 'JREPR', 'REPRODUCIBLE-RUN    ', JREPR
1174:      write(IPR,7060) 'JTSKR', 'SPAWN-TASKER      ', JTSKR
1175:      write(IPR,7060) 'JLATT', 'LATTICE            ', JLATT
1176:      write(IPR,7060) 'JFPRT', 'FLUX-PRINT      ', JFPRT
1177:      write(IPR,7060) 'JTLLT', 'TALLY-LATTICE   ', JTLLT
1178:      write(IPR,7060) 'JRTTR', 'EDIT-BY-TRACK-LENGTH', JRTTR, 'JRTCL'
1179:      write(IPR,7060) 'JRTCL', 'EDIT-BY-COLLISION   ', JRTCL, 'JRTTR'
1180:      write(IPR,7100) 'JMICE', 'EDIT-MICROSCOPIC-DATA      '
1181:      write(IPR,7120) (REAC(L),JMICE(L),L=1,8)
1182:      write(IPR,7100) 'JMACE', 'EDIT-MACROSCOPIC-DATA      '
1183:      write(IPR,7120) (REAC(L),JMACE(L),L=1,8)
1184:      write(IPR,7060) 'JSCTM', 'SCATTERING-MATRIX    ', JSCTM ! scat-mtx
1185:      write(IPR,7060) 'JSCMU', 'SCATTERING-MUBAR     ', JSCMU ! scat-mtx
1186:      write(IPR,7060) 'JAMXC', 'ADAPTIVE-MICRO-CALCULATION    ', JAMXC
1187:      write(IPR,7060) 'JPERT', 'PERTURBATION        ', JPERT ! pert
1188:      write(IPR,7060) 'JTLST', 'TIME-LIST             ', JTLST ! time-list
1189:      do 250 I = 1, 2
1190:         LINE = ' '
1191:         write(LINE, ' (' SUBTASK-PRINT ('',i2,'')') ) I
1192:         call CCOMP( LINE, LEN(LINE), LINE, IERR )
1193:         write(IPR,7060) 'JSTPRN', LINE(:20), JSTPRN(I)
1194:      250 continue
1195:      7100 format(/1X,2X,A,T10,2X,A32)
1196:      7120 format(1X,'<',A12,'>',I2)
1197:      write(IPR,*) ' '
1198:      if ( JPRTS(1).eq.1 ) write(IPR,*)
1199:      & ' * PRINT OF GROUP-WISE FLUX IS SUPPRESSED.'
1200:      if ( JPRTS(2).eq.1 ) write(IPR,*)
1201:      & ' * PRINT OF ANGULAR FLUX IS SUPPRESSED.'
1202:      if ( JPRTS(3).eq.1 ) write(IPR,*)
1203:      & ' * PRINT OF TIME-DEPENDENT FLUX IS SUPPRESSED.'
1204:      if ( JPRTS(4).eq.1 ) write(IPR,*)
1205:      & ' * PRINT OF RESPONSE IS SUPPRESSED.'
1206:      if ( JPRTS(5).eq.1 ) write(IPR,*)
1207:      & ' * PRINT OF TIME-DEPENDENT RESPONSE ' , ' IS SUPPRESSED.'
1208:      if ( JPRTS(6).eq.1 ) write(IPR,*)
1209:      & ' * PRINT OF SOURCE INFORMATION #1 ' , ' IS SUPPRESSED.'
1210:      if ( JPRTS(7).eq.1 ) write(IPR,*)
1211:      & ' * PRINT OF SOURCE INFORMATION #2 ' , ' IS SUPPRESSED.'
1212:      if ( JPRTS(17).eq.1 ) write(IPR,*)
1213:      & ' * PRINT OF MICROSCOPIC REACTION RATE', ' IS SUPPRESSED.'
1214:      if ( JPRTS(18).eq.1 ) write(IPR,*)
1215:      & ' * PRINT OF MICROSCOPIC CROSS SECTION', ' IS SUPPRESSED.'
1216:      if ( JPRTS(19).eq.1 ) write(IPR,*)
1217:      & ' * PRINT OF MACROSCOPIC REACTION RATE', ' IS SUPPRESSED.'
1218:      if ( JPRTS(20).eq.1 ) write(IPR,*)
1219:      & ' * PRINT OF MACROSCOPIC CROSS SECTION', ' IS SUPPRESSED.'
1220: C
1221:      return
1222:      end
```

src/mvp/panlgf.f

```

1:      subroutine PANLGF( IOW,
2:      &                  IWK5,IMTTP,IWK2,IWK3,IWK11,IWK12,IWK13,WK4,WK5,
3:      &                  NGROUP,NZONE, NMAT, NREG, NTIME,
4:      &                  NBANK, NFBANK,NFBNK0,NEST, NUC, NMT,
5:      &                  MB, NSMIC, MMAC, MXPGEN,
6:      &                  ETOP, EBOT, ETHMAX, CX, MCX,
7:      &                  KLIB1, KLIB1, KLIB2, XLIB2, SMIC, LMIC, KSPI,
8:      &                  LSFFL, NFFL, IZFFL, LSCOL, NCOLS, LSDED,
9:      &                  NDEAD, NFISB,
10:     &                  XXX, YYY, ZZZ, AAA, BBB, CCC,
11:     &                  WWW, EEE, TTT, ITT, XIM,
12:     &                  KKP,IZZ,IGG, LEVE, LZZ, LPOS, LCRS, KLSF,
13:     &                  IBREG, IBSPC, XXXF, YYYYF, ZZZF, EEEF, INUF,
14:     &                  IZZF, LEVLF, LZZF, LPOSF, LCRSF, IBRGF,
15:     &                  IBSPF, DBNK, KDBNK, MDBNK, IBNK, KIBNK,
16:     &                  MBNK, NECUT, WECUT, NCNTR, WCNTR, IRAND,
17:     T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY,
18:     &                  RNU,
19:     &                  IWK1, IWK4, IWK6, IWK7, IWK8, IWK9, IWK10,
20:     &                  IWK14, IWK15, DWK1, DWK2, WK3, WK6, WK7, R,
21:     &                  NUC_MAX, ! photonuc
22:     &                  DWK3, IOTL) ! time-list
23: C=====
24: C PURPOSE: Fission neutron generation for noise analysis
25: C CALLED IN: NEUTR
26: C CALLS: RANU2, SEF5N2, STAN7
27: C-----
28: C Argument: (i=input, o=output, w=work)
29: C
30: C wo IWK5(*) : can be used as work area, but finally, bank pointer of
31: C generated fission neutrons must be stored if JEIGN=0.
32: C i IWK2(*) : region # set in caller routine (NEUTR)
33: C i IWK3(*) : colliding nuc # set in caller routine (NEUTR) and modified
34: C in this routine if contents of collision neutron stack
35: C are changed.
36: C io LSCOL(NCOLS) : collision neutron stack. Contents of this stack
37: C may be changed if parent neutron is replaced by fission
38: C neutron (by lack of particle bank space).
39: C io NCOLS : may be changed by the same reason above.
40: C
41: C---- Data for gamma-generation neutrons (effective if JPHOT=1)
42: C o IMTTP : Number of assumed-gamma-generation-neutrons
43: C o IWK11(*) : Reaction MT number ( not MTTP !! )
44: C o IWK12(*) : Bank pointer for assumed-gamma-generation neutrons
45: C o IWK13(*) : Colliding nuclide ( <- IWK3 )
46: C o WK4(*) : Weight of absorbed parent neutron
47: C o WK5(*) : Weight of scattered parent neutron
48: C
49: C=====
50:
51: C*****IMPLICIT REAL*8 (A-H,O-Z)
52:      implicit real*8(A-H,O-Q,S-Z)
53: C
54:      include 'INC/_KPIDS'
55:      include 'INC/_NGPS'
56: C
57:      include 'INC/_FLAGS'
58: C
59:      integer IWK2(NBANK), IWK3(NBANK), IWK5(NBANK)
60:      integer IWK11(NBANK), IWK12(NBANK), IWK13(NBANK)
61:      real WK4(NBANK), WK5(NBANK)
62: C
63: C**** CROSS SECTION DATA IN CX ARRAY
64: C
65:      real EBOT, ETOP, ETHMAX
66: c##<2007/03/14:PN3:
67: c## real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,2)
68: c## integer KLIB1(NUC,30), KLIB2(NUC,NMT,21)
69: c## real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
70: c## integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
71: c##>
72: C
73: C
74: C**** SIGMA BANK
75: C
76:      real SMIC(NBANK,NUC,NSMIC)
77: C ... RNU : nu-bar
78: c##<2007/03/14:PN3:
79: c##C RNU(NBANK,NUC,1) : nu-total
80: c##C RNU(NBANK,NUC,2) : nu-delayed
81: c##C RNU(NBANK,NUC,3) : nu-prompt
82: c## real RNU(NBANK,NUC,3)
83: C RNU(NBANK,NUC_MAX,1) : nu-total
84: C RNU(NBANK,NUC_MAX,2) : nu-delayed
85: C RNU(NBANK,NUC_MAX,3) : nu-prompt
86:      real RNU(NBANK,NUC_MAX,3)
87: c##>
88: C
89:      integer MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8)
90: C
91: C**** STACK
92: C
93:      integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE), LSCOL(NBANK),
94:      &          LSDED(NBANK)
95: C
96: C**** PARTICLE BANK
97: C
98:      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
99:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
100:      real*8 TTT(NBANK)
101:      real EEE(NBANK), WWW(NBANK), XIM(NBANK)
102:      integer KKP(NBANK)
103:      integer IZZ(NBANK), IGG(NBANK)
104:      integer ITT(NBANK)
105:      integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST),
106:      &          LCRS(NBANK,NEST)
107:      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
108:      integer KLSF(NBANK)
109: C
110:      real*8 DBNK(NBANK,*)
111:      integer KDBNK(0:MDBNK)
112:      integer IBNK(NBANK,*)
113:      integer KIBNK(0:MIBNK)
114: C
115: C**** fission neutron bank
116: C
117:      real*8 XXXF(NFBNK0), YYYYF(NFBNK0), ZZZF(NFBNK0)
118:      real EEEF(NFBNK0)
119:      integer IZZF(NFBNK0), INUF(NFBNK0)
120:      integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST)
121:      integer LCRSF(NFBNK0,NEST), LPOSF(NFBNK0,NEST)
122:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
123: C
124: C**** TALLY ARRAY
125: C
126:      integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
127:      integer IDTAL(*)
128:      real*8 DTALY(*)
129: C
130: C ... assuming KPNEUT=1 for WCNTR/NCNTR :-<

```


src/mvp/panlgf.f

```

131:      real*8 WCNTR(*), NCNTR(*)
132:      real*8 WECUT(NREG)
133:      real*8 NECUT(NREG)
134: C
135: C**** WORKING ARRAYS
136: C
137:      integer IWK1(NBANK), IWK4(NBANK), IWK6(NBANK), IWK7(NBANK),
138:      &      IWK8(NBANK), IWK9(NBANK), IWK10(NBANK),
139:      &      IWK14(NBANK), IWK15(NBANK)
140:      real*8 DWK1(NBANK), DWK2(NBANK)
141:      real*8 DWK3(NBANK) ! time-list
142:      real WK3(NBANK), WK6(NBANK), WK7(NBANK)
143:      real I(8*NBANK)
144: C
145: C .... local data ...
146: C
147:      real BX(50)
148: C
149: C***** INPUT
150: C IWK2 : IREG
151: C IWK3 : COLLIDING NUCLIDE
152: C
153:      integer MNAL
154:      parameter( MWARN = 200 )
155:      parameter( PI = 3.1415926535897932D0 )
156:      parameter( PI2 = 2.0D0*PI )
157: C
158:      data BX /
159:      & 5.00000E-01,4.60170E-01,4.20740E-01,3.82090E-01,
160:      & 3.44580E-01,3.08540E-01,2.74250E-01,2.41960E-01,
161:      & 2.11860E-01,1.84060E-01,1.58660E-01,1.35670E-01,
162:      & 1.15070E-01,9.68000E-02,8.07570E-02,6.68070E-02,
163:      & 5.47990E-02,4.45650E-02,3.59300E-02,2.87170E-02,
164:      & 2.27500E-02,1.78640E-02,1.39030E-02,1.07240E-02,
165:      & 8.19750E-03,6.20970E-03,4.66120E-03,3.46700E-03,
166:      & 2.55510E-03,1.86580E-03,1.34990E-03,9.67600E-04,
167:      & 6.87140E-04,4.83420E-04,3.36930E-04,2.32630E-04,
168:      & 1.59110E-04,1.07800E-04,7.23480E-05,4.80960E-05,
169:      & 3.16170E-05,2.06580E-05,1.33460E-05,8.53990E-06,
170:      & 5.41250E-06,3.39770E-06,2.11250E-06,1.30080E-06,
171:      & 7.93330E-07,4.79180E-07 /
172: C
173:      data MNAL /0/
174: C
175: C=====
176: C
177: C**** Analog treatment of absorption
178: C-----
179: C      LSCOL ---> LSCOL, IWK10, IWK12
180: C-----
181: C
182: C      .... NAF: Number of fissions, NAC: Number of captures
183: C
184:      NN = 0
185:      NAF = 0
186:      NAC = 0
187:      call RANU2( IRAND, R, NCOLS, ICON )
188: C
189: *VOCL LOOP,NOVREC
190:      do 100 I = 1, NCOLS
191:      IP = LSCOL(I)
192:      IWK2I = IWK2(I)
193: C
194: C      .... WF : fission probability
195: C      .... WA : absorption probability ( capture + fission )

```

```

196:      WF = SMIC(IP,IWK3(I),LMIC(3)) /SMIC(IP,IWK3(I),LMIC(1))
197:      WA = WF + SMIC(IP,IWK3(I),LMIC(5)) /
198:      &      SMIC(IP,IWK3(I),LMIC(1))
199: C
200:      if ( R(I).gt.WA ) then
201:      NN = NN + 1
202:      LSCOL(NN) = IP
203:      IWK7(NN) = IWK3(I)
204:      IWK6(NN) = IWK2I
205:      else if ( R(I).gt.WF ) then
206:      NAC = NAC + 1
207:      IWK10(NAC) = IP
208:      IWK9(NAC) = IWK3(I)
209:      IWK8(NAC) = IWK2I
210:      else
211:      NAF = NAF + 1
212:      IWK12(NAF) = IP
213:      IWK13(NAF) = IWK3(I)
214:      IWK2(NAF) = IWK2I
215:      end if
216:      100 continue
217:      NCOLS = NN
218: C
219:      do 110 I = 1, NAC
220:      WCNTR(23) = WCNTR(23) + WWW(IWK10(I))
221:      110 continue
222:      do 120 I = 1, NAF
223:      WCNTR(23) = WCNTR(23) + WWW(IWK12(I))
224:      120 continue
225:      NCNTR(23) = NCNTR(23) + NAC + NAF
226: C
227: C
228: C-----
229: C      .... TAKE special tallies .....
230: C-----
231: C
232:      if ( NTEVE.gt.0.and.JTEVE(8).gt.0 ) then
233: C
234:      do 190 J = 0, JTEVE(8) - 1
235: C
236: C      ... pointer to direct tally (idt) & d-tally # (it) ...
237:      IDT = KTEVE(LTEVE(8)+J) - 1
238:      IMULT = IDTAL(IDT+7)
239:      IT = IDTAL(IDT+9)
240: C
241:      if ( IMULT.eq.9001.and.NCOLS.gt.0 ) then
242: C
243: C      ... skip if current region does not need this tally ...
244: C
245:      do 130 I = 1, NCOLS
246:      if ( JDTRG(IWK6(I),IT).ne.0 ) go to 140
247:      130 continue
248:      go to 190
249: C
250:      140 continue
251:      do 150 I = 1, NCOLS
252:      IWK14(I) = IGG(LSCOL(I))
253:      DWK1(I) = WWW(LSCOL(I))
254:      if ( JTIME.ne.0 ) IWK15(I) = ITT(LSCOL(I))
255:      if ( JTIME.ne.0 ) DWK3(I) = TTT(LSCOL(I)) ! time-list
256:      150 continue
257: C
258:      call STALN7( IOW, IDTAL(IDT+1),DWK1, LSCOL, NCOLS, IWK14,
259:      &      IWK6, IWK15, NGROUP, NREG, NTIME, NBANK,
260:      &      DTALY, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,

```

src/mvp/panlgf.f

```

261:      &          DWK2, IWK4,
262:      &          DWK3, JTLST, IOTL) ! time-list
263:      else if ( IMULT.eq.9002.and.NAC.gt.0 ) then
264: C
265: C      ... skip if current region does not need this tally ...
266: C
267:      do 160 I = 1, NAC
268:      if ( JDTRG(IWK8(I),IT).ne.0 ) go to 170
269: 160      continue
270:      go to 190
271: C
272: 170      continue
273:      do 180 I = 1, NAC
274:      IWK14(I) = IGG(IWK10(I))
275:      DWK1(I) = WWW(IWK10(I))
276:      if ( JTIME.ne.0 ) IWK15(I) = ITT(IWK10(I))
277:      if ( JTIME.ne.0 ) DWK3(I) = TTT(IWK10(I)) ! time-list
278: 180      continue
279: C
280:      call STALN7( IOW, IDTAL(IDT+1), DWK1, IWK10, NAC, IWK14,
281:      &          IWK8, IWK15, NGROUP, NREG, NTIME, NBANK,
282:      &          DTALY, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
283:      &          DWK2, IWK4,
284:      &          DWK3, JTLST, IOTL) ! time-list
285: C
286:      else if ( IMULT.eq.9003.and.NAF.gt.0 ) then
287: C
288: C      ... skip if current region does not need this tally ...
289: C
290:      do 181 I = 1, NAF
291:      if ( JDTRG(IWK2(I),IT).ne.0 ) go to 182
292: 181      continue
293:      go to 190
294: C
295: 182      continue
296:      do 183 I = 1, NAF
297:      IWK14(I) = IGG(IWK12(I))
298:      DWK1(I) = WWW(IWK12(I))
299:      if ( JTIME.ne.0 ) IWK15(I) = ITT(IWK12(I))
300:      if ( JTIME.ne.0 ) DWK3(I) = TTT(IWK12(I)) ! time-list
301: 183      continue
302: C
303:      call STALN7( IOW, IDTAL(IDT+1), DWK1, IWK12, NAF, IWK14,
304:      &          IWK2, IWK15, NGROUP, NREG, NTIME, NBANK,
305:      &          DTALY, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
306:      &          DWK2, IWK4,
307:      &          DWK3, JTLST, IOTL) ! time-list
308:      else if ( IMULT.eq.9004.and.NAC+NAF.gt.0 ) then
309:      if ( NAF.gt.0 ) then
310:      do 184 I = 1, NAF
311:      JJ = NAC + I
312:      IWK8(JJ) = IWK2(I)
313:      IWK10(JJ) = IWK12(I)
314: 184      continue
315:      end if
316:      NAA = NAC + NAF
317: C
318: C      ... skip if current region does not need this tally ...
319: C
320:      do 185 I = 1, NAA
321:      if ( JDTRG(IWK8(I),IT).ne.0 ) go to 186
322: 185      continue
323:      go to 190
324: C
325: 186      continue

```

```

326:      do 187 I = 1, NAA
327:      IWK14(I) = IGG(IWK10(I))
328:      DWK1(I) = WWW(IWK10(I))
329:      if ( JTIME.ne.0 ) IWK15(I) = ITT(IWK10(I))
330:      if ( JTIME.ne.0 ) DWK3(I) = TTT(IWK10(I)) ! time-list
331: 187      continue
332: C
333:      call STALN7( IOW, IDTAL(IDT+1), DWK1, IWK10, NAA, IWK14,
334:      &          IWK8, IWK15, NGROUP, NREG, NTIME, NBANK,
335:      &          DTALY, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
336:      &          DWK2, IWK4,
337:      &          DWK3, JTLST, IOTL) ! time-list
338:      end if
339: 190      continue
340:      end if
341: C
342: C
343:      if ( JPHOT.eq.0 ) then
344: C
345: C-----
346: C      neutorn only problem
347: C-----
348:      do 200 I = 1, NAC
349: C      ..... absorption without fission .....
350:      NDEAD = NDEAD + 1
351:      LSDED(NDEAD) = IWK10(I)
352: 200      continue
353: C
354:      else if ( JPHOT.ne.0 ) then
355: C
356: C-----
357: C      neutorn/photon calculation
358: C-----
359: C      << STORED DATA FOR PHOTON GENERATION >>
360: C
361: C      Data for assumed-gamma-generation neutrons
362: C
363: C      IMTTP : Number of assumed-gamma-generation neutrons
364: C      IWK11(I) : Reaction MT number ( not MTTP !! )
365: C      IWK12(I) : Bank pointer for assumed-gamma-generation neutrons
366: C      IWK13(I) : Colliding nuclide ( <- IWK3 )
367: C      WK4(I) : Weight of absorbed parent neutron
368: C      WK5(IP) : Weight of scattered parent neutron
369: C      .....
370: C
371:      do 210 I = 1, NCOLS
372:      WK5(LSCOL(I)) = WWW(LSCOL(I))
373: 210      continue
374: C
375: C      .... Add gamma-production neutrons by capure to IWK12 ....
376: C
377:      IMTTP = NAF + NAC
378: C
379:      do 220 I = 1, NAF
380:      if ( KLIB2(IWK13(I),18,15).gt.0 ) then
381: C      ..... select particles with positive mtog .....
382:      IWK11(I) = 18
383:      else
384: C      ..... negative value means no photon generation ..
385:      IWK11(I) = -18
386:      end if
387: 220      continue
388:      do 230 I = 1, NAC
389:      IWK11(I+NAF) = 0
390:      IWK12(I+NAF) = IWK10(I)

```

src/mvp/panlgf.f

```

391:          IWK13(I+NAF) = IWK9(I)
392: 230      continue
393:          do 240 I = 1, IMTTP
394:              WK4(I) = WWW(IWK12(I))
395: 240      continue
396: C
397: C-----
398: C      Determine MT-number for assumed photon-generation from capture
399: C      if necessary.
400: C      IWK11 : MT-number to be determined (initially= 0)
401: C      IWK12 : Bank pointer
402: C      IWK13 : Nuclide #
403: C-----
404: C
405: C      if ( NAF.lt.IMTTP ) then
406: C          IDET = 0
407: C
408: C      ..... IWK4 : Cross section pointer ....
409: C      ..... R(1:IMTTP) : Cross section summation ....
410: C      ..... R(IMTTP+1:2*IMTTP) : Total capture cross section ....
411: C
412: C
413: C      MXCAPG = 0
414: C      do 250 I = NAF + 1, IMTTP
415: C          IP = IWK12(I)
416: C
417: C          ..... KSPI: Energy point #
418: C
419: C          IWK6(I) = KSPI(IP,IWK13(I))
420: C          IWK4(I) = KLIB1(IWK13(I),1) + IWK6(I)
421: C
422: C          ..... NCAPG ....
423: C
424: C          IWK7(I) = KLIB1(IWK13(I),27)
425: C          MXCAPG = MAX(IWK7(I),MXCAPG)
426: C          R(I) = 0.0
427: C          R(I+IMTTP) = SMIC(IP,IWK13(I),LMIC(5))
428: 250      continue
429: C
430: C      call RANU2( IRAND, R(2*IMTTP+NAF+1), IMTTP, ICON )
431: C
432: C      do 270 MMT = 1, MXCAPG
433: C          if ( IDET.eq.NAC ) go to 280
434: C
435: C      *VOCL LOOP,NOVREC
436: C          do 260 I = NAF + 1, IMTTP
437: C              if ( MMT.le.IWK7(I).and.IWK11(I).eq.0 ) then
438: C
439: C          .... MTCAPG -> MT ....
440: C
441: C              MT = KLIB2(IWK13(I),MMT,21)
442: C              IP = IWK12(I)
443: C
444: C          .... Cross section for MT ....
445: C
446: C              E1 = CX(IWK4(I))
447: C              TI = (EEE(IP)-E1)/(CX(IWK4(I))-1)-E1
448: C              L1 = KLIB2(IWK13(I),MT,11)
449: C              & - KLIB2(IWK13(I),MT,3) + IWK6(I)
450: C
451: C          ..... SC: Capture cross section ...
452: C
453: C              if ( IWK6(I).ge.KLIB2(IWK13(I),MT,3)
454: C              & .and.IWK6(I).lt.KLIB2(IWK13(I),MT,4) ) then
455: C                  SC = CX(L1)*TI + CX(L1+1)*(1.0-TI)

```

```

456:          else
457:              SC = 0.0
458:          end if
459: C
460: C      .... Check
461: C
462: C          R(I) = R(I) + SC
463: C
464: C      (Partial sum of cap. xsec)/(Total cap. x-sec) > Random.n
465: C      ----> MT determined
466: C
467: C          if ( R(IMTTP+I)*R(I+2*IMTTP).le.R(I) ) then
468: C              IWK11(I) = MT
469: C              IDET = IDET + 1
470: C          end if
471: C      end if
472: 260      continue
473: 270      continue
474: C
475: 280      if ( IDET.lt.NAC ) then
476: C          do 290 I = NAF + 1, IMTTP
477: C          .... MTCAPG(NCAPG) -> MT IF MTOG(MTCAPG)=3 ...
478: C              if ( IWK11(I).eq.0.and.IWK7(I).ne.0 ) then
479: C                  MTCAPG = KLIB2(IWK13(I),IWK7(I),21)
480: C                  if ( KLIB2(IWK13(I),MTCAPG,15).eq.3 ) then
481: C                      IWK11(I) = MTCAPG
482: C                  end if
483: C              end if
484: 290      continue
485: C          end if
486: C
487: C      end if
488: C
489: C      NN = NAF
490: C
491: C      *VOCL LOOP,NOVREC
492: C          do 300 I = NAF + 1, IMTTP
493: C              IWK12I = IWK12(I)
494: C              if ( IWK11(I).eq.0 ) then
495: C
496: C          ..... no gamma production by capture reactions .....
497: C              NDEAD = NDEAD + 1
498: C              LSDED(NDEAD) = IWK12I
499: C          else
500: C              NN = NN + 1
501: C              IWK11(NN) = IWK11(I)
502: C              IWK12(NN) = IWK12I
503: C              IWK13(NN) = IWK13(I)
504: C              WK4(NN) = WK4(I)
505: C          end if
506: 300      continue
507: C          IMTTP = NN
508: C
509: C      end if
510: C
511: C      ( can be used freely hereafter )
512: C
513: C      if ( NAF.eq.0 ) then
514: C          NFISB = 0
515: C          return
516: C      end if
517: C
518: C=====
519: C***** Fission neutron generation
520: C=====

```

src/mvp/panlgf.f

```

521: C
522: C   NAF      : Number of neutrons inducing fission
523: C   IWK12(I) : Bank pointer for parent neutrons
524: C   IWK13(I) : Colliding nuclide
525: C   WK4(I)   : Weight of absorbed parent neutron
526: C   WK5(IP)  : Weight of scattered parent neutron
527: C
528: C ... IWK4(I) is number of fission neutrons to be generated,
529: C all of them are launched to random walk (fixed source) if possible
530: C
531: C -----
532: C
533: C call RANU2( IRAND, R, NAF*2, ICON )
534: C
535: C   NFISS = 0
536: C   SIGMA = 1.08D0
537: C
538: C do 310 I = 1, NAF
539: C   IP = IWK12(I)
540: C
541: C ... Nu-bar value
542: C
543: C   if ( SMIC(IP,IWK13(I),LMIC(3)).gt.0. ) then
544: C     DWK1(I) = SMIC(IP,IWK13(I),LMIC(2)) /
545: C &         SMIC(IP,IWK13(I),LMIC(3))
546: C   else
547: C     DWK1(I) = 0.0
548: C   end if
549: C
550: C   if ( JFMUL.eq.1 ) then
551: C ... Nu-prompt
552: C     DWK1(I) = RNU(IP,IWK13(I),3)
553: C   else
554: C ... Nu-total
555: C     DWK1(I) = RNU(IP,IWK13(I),1)
556: C   end if
557: C
558: C ... correction factor B for nu-bar
559: C
560: C   X = (DWK1(I)+0.5D0) /SIGMA
561: C   IX = X/0.1D0
562: C   if ( IX.lt.49 ) then
563: C     XR = (X-0.1D0*IX) /0.1D0
564: C     B = BX(IX+1) + XR*(BX(IX+2)-BX(IX+1))
565: C   else
566: C     B = 0.0
567: C   end if
568: C Check
569: C   B = 0.0
570: C
571: C ... sampling from normal distribution ( sigma = SIGMA(1.08) )
572: C
573: C   DWK2(I) = SQRT(-2.D0*LOG(R(I)))*COS(PI2*(NAF+I))
574: C   if ( DWK2(I).gt.8.D0 ) DWK2(I) = 8.D0
575: C   DWK2(I) = DWK2(I)*SIGMA
576: C
577: C   DWK2(I) = DWK2(I) + DWK1(I) - B
578: C
579: C   if ( DWK1(I).ne.0.0.and.DWK2(I).ge.0.5D0 ) then
580: C     IWK4(I) = DWK2(I) + 0.5D0
581: C   else
582: C     IWK4(I) = 0
583: C   end if
584: C
585: C ... check generation cutoff

```

```

586: C
587: C   if ( MXPGEN.gt.0 .and. IBNK(IP,KIBNK(4)).ge.MXPGEN ) then
588: C     IWK4(I) = 0
589: C   end if
590: C
591: C   NFISS = NFISS + IWK4(I)
592: C 310 continue
593: C
594: C
595: C =====
596: C   Fixed source problem
597: C =====
598: C
599: C   if ( JEIGN.eq.0 ) then
600: C
601: C     Fission neutrons are stored in particle bank(XXX,YYY,.....)
602: C
603: C     IWK5 : fission neutron stack
604: C
605: C     if ( NFISS.le.0 ) then
606: C       if ( JPHOT.eq.0 ) then
607: C         do 320 I = 1, NAF
608: C           NDEAD = NDEAD + 1
609: C           LSDED(NDEAD) = IWK12(I)
610: C 320       continue
611: C         else
612: C           NN = 0
613: C *VOCL LOOP,NOVREC
614: C           do 330 I = 1, IMTTP
615: C             IWK12I = IWK12(I)
616: C             if ( IWK11(I).gt.0 ) then
617: C               NN = NN + 1
618: C               IWK11(NN) = IWK11(I)
619: C               IWK12(NN) = IWK12I
620: C               IWK13(NN) = IWK13(I)
621: C               WK4(NN) = WK4(I)
622: C             else
623: C               NDEAD = NDEAD + 1
624: C               LSDED(NDEAD) = IWK12I
625: C             end if
626: C 330       continue
627: C           IMTTP = NN
628: C         end if
629: C         NFISB = 0
630: C         return
631: C       end if
632: C
633: C ***** All fission neutron can be stored in bank
634: C
635: C     if ( NFISS.le.NDEAD ) then
636: C       MA = 0
637: C       do 340 I = 1, NAF
638: C         MA = MAX(MA,IWK4(I))
639: C 340       continue
640: C
641: C     ..... IWK6 : STACK POINTER
642: C
643: C     N = 0
644: C     do 360 K = 1, MA
645: C       do 350 I = 1, NAF
646: C         if ( IWK4(I).ge.K ) then
647: C           N = N + 1
648: C           IWK6(N) = I
649: C           WK3(N) = WWW(IWK12(I))
650: C         end if

```

src/mvp/panlgf.f

```

651: 350      continue
652: 360      continue
653: C
654: C ..... I1 : number of parent neutrons generating fission neutron
655: C           in analog manner.
656: C
657: C       I1      = NAF
658: C
659: C***** not all fission neutrons can be stored in bank
660: C       Caution: In this case, random walk is not in analog manner.
661: C
662: C       else
663: C
664: C       NFISS = 0
665: C       N     = 0
666: C       I1    = 0
667: C       do 380 I = 1, NAF
668: C         K = IWK4(I)
669: C         NFISS = NFISS + K
670: C         if ( NFISS.gt.NDEAD ) go to 390
671: C         I1 = I
672: C         do 370 II = 1, K
673: C           N = N + 1
674: C           IWK6(N) = I
675: C           WK3(N) = WWW(IWK12(I))
676: C       370      continue
677: C       380      continue
678: C
679: C ..... I1 : number of parent neutrons generating fission neutron
680: C           in analog manner.
681: C
682: C
683: C .... number of fission neutron is modified here ....
684: C
685: C 390      NFISS = N
686: C
687: C .... count fission neutron not launched to random walk ....
688: C
689: C       do 400 I = I1 + 1, NAF
690: C         NCNTR(13) = NCNTR(13) + IWK4(I)
691: C         WCNTR(13) = WCNTR(13) + IWK4(I)*WWW(IWK12(I))
692: C       400      continue
693: C
694: C .... Fission neutrons not banked above are stored in the bank
695: C       for parent neutrons if JPHOT=0.
696: C       Select neutron or photon with equal probability if JPHOT=1.
697: C
698: C       if ( JPHOT.eq.0 ) then
699: C *VOCL LOOP,NOVREC
700: C       do 410 I = I1 + 1, NAF
701: C         IP = IWK12(I)
702: C         if ( IWK4(I).gt.0 ) then
703: C           N = N + 1
704: C           IWK6(N) = I
705: C           WK3(N) = WWW(IP)*IWK4(I)
706: C           WWW(IP) = WK3(N)
707: C         else
708: C           NDEAD = NDEAD + 1
709: C           LSDED(NDEAD) = IP
710: C         end if
711: C       410      continue
712: C       else
713: C
714: C       call RANU2( IRAND, R, NAF-I1, ICON )
715: C

```

```

716: PNFIS = 0.5
717: C
718: C *VOCL LOOP,NOVREC
719: C       do 420 I = I1 + 1, NAF
720: C         if ( IWK4(I).gt.0 ) then
721: C           IP = IWK12(I)
722: C           if ( R(I-I1).gt.PNFIS ) then
723: C ..... parent neutron produces photons without n ...
724: C           WWW(IP) = WWW(IP) /(1.0-PNFIS)
725: C         else
726: C ..... parent neutron does not produce photons ...
727: C           IWK11 : negative
728: C           IWK11(I) = -1
729: C           N = N + 1
730: C           IWK6(N) = I
731: C           WK3(N) = WWW(IP)*IWK4(I) /PNFIS
732: C           WWW(IP) = WK3(N)
733: C         end if
734: C       end if
735: C 420      continue
736: C       end if
737: C
738: C .... warning message for non-positive cross section fixup...
739: C
740: C       if ( MNAL.lt.MWARN ) then
741: C         NNAL = N - NFISS
742: C         write(IOW,7000) NNAL
743: C 7000      format(// '!!! ',I5,' FISSION NEUTRONS ARE GENERATED ',
744: C       &          'IN NON-ANALOG MANNER')
745: C         MNAL = MNAL + NNAL
746: C
747: C       if ( MNAL.ge.MWARN ) then
748: C         write(IOW,*) ' !!! MESSAGES FOR NON-ANALOG FISSION ',
749: C       &          'WILL BE SUPPRESSED HEREAFTER.'
750: C       end if
751: C       end if
752: C
753: C       end if
754: C
755: C ..... N = NFISS when all fission neutrons are stored in the bank
756: C       N = NFISS (fission n with weight WWW) +
757: C       prevented fission with expected weight WWW*Number
758: C
759: C
760: C       if ( N.le.0 ) then
761: C         if ( JPHOT.eq.0 ) then
762: C           do 430 I = 1, I1
763: C             NDEAD = NDEAD + 1
764: C             LSDED(NDEAD) = IWK12(I)
765: C 430          continue
766: C         else
767: C           NN = 0
768: C *VOCL LOOP,NOVREC
769: C           do 440 I = 1, IMTTP
770: C             IWK12I = IWK12(I)
771: C             if ( IWK11(I).gt.0 ) then
772: C               NN = NN + 1
773: C               IWK11(NN) = IWK11(I)
774: C               IWK12(NN) = IWK12I
775: C               IWK13(NN) = IWK13(I)
776: C               WK4(NN) = WK4(I)
777: C             else
778: C               NDEAD = NDEAD + 1
779: C               LSDED(NDEAD) = IWK12I
780: C             end if

```

src/mvp/panlgf.f

```

781: 440      continue
782:      IMTTP = NN
783:      end if
784:      NFISB = 0
785:      return
786:      end if
787: C
788: C ***** count fission neutrons *****
789: C
790: *VOCL LOOP,NOVREC
791: do 450 I = 1, N
792:   WCNTR(2) = WCNTR(2) + WK3(I)
793: 450   continue
794:   NCNTR(2) = NCNTR(2) + N
795: C
796: C
797: C***** determine energy of fission neutrons *****
798: C
799: C
800: C Incident neutron energy : WK6
801: C Colliding nuclide       : IWK1
802: C Fission neutron energy ---> WK7
803: C
804:   do 460 I = 1, N
805:     II = IWK6(I)
806:     IWK1(I) = IWK13(II)
807:     WK6(I) = EEE(IWK12(II))
808: 460   continue
809: C
810:   call SEF5N2( IOW, IWK1, N, WK6, WK7, CX, CX, MCX, KLIB1, XLIB1,
811: &             KLIB2, XLIB2, NBANK, NUC, NMT, IRAND, JDEB, IWK3,
812: &             IWK4, IWK5, IWK7, IWK8, IWK9, IWK10, IWK14, IWK15, R )
813: C
814:   do 470 I = 1, N
815:     WK7(I) = MIN(WK7(I),ETOP)
816: 470   continue
817: C
818:   if ( ETHMAX.ge.EBOT ) then
819:     do 480 I = 1, N
820:       WK7(I) = MAX(WK7(I),EBOT)
821: 480   continue
822:   NFISB = NFISS
823:   NN = N
824: C
825: C ... ENERGY CUT OFF .....
826: C
827:   else
828: C
829:   NN = 0
830: C
831:   if ( JMNTR.ne.0 ) then
832:     do 490 I = 1, N
833:       if ( WK7(I).lt.EBOT ) then
834:         IREG = IWK2(IWK6(I))
835:         NECUT(IREG) = NECUT(IREG) + 1
836:         WECUT(IREG) = WECUT(IREG) + WK3(I)
837:       end if
838: 490     continue
839:   end if
840: *VOCL LOOP,NOVREC
841:   do 500 I = 1, NFISS
842:     WK3I = WK3(I)
843:     if ( WK7(I).lt.EBOT ) then
844:       NCNTR(8) = NCNTR(8) + 1
845:       WCNTR(8) = WCNTR(8) + WK3I

```

```

846:   else
847:     NN = NN + 1
848:     IWK6(NN) = IWK6(I)
849:     WK7(NN) = WK7(I)
850:     WK3(NN) = WK3I
851:   end if
852: 500   continue
853: C
854:   NFISB = NN
855: C
856: *VOCL LOOP,NOVREC
857:   do 510 I = NFISS + 1, N
858:     IP = IWK12(IWK6(I))
859:     if ( WK7(I).lt.EBOT ) then
860:       NCNTR(8) = NCNTR(8) + 1
861:       WCNTR(8) = WCNTR(8) + WWW(IP)
862:       NDEAD = NDEAD + 1
863:       LSDED(NDEAD) = IP
864:     else
865:       NN = NN + 1
866:       IWK6(NN) = IWK6(I)
867:       WK7(NN) = WK7(I)
868:     end if
869: 510   continue
870:   end if
871: C
872:   if ( NN.eq.0 ) then
873:     if ( JPHOT.eq.0 ) then
874:       do 520 I = 1, II
875:         NDEAD = NDEAD + 1
876:         LSDED(NDEAD) = IWK12(I)
877: 520       continue
878:     else
879:       NN = 0
880: *VOCL LOOP,NOVREC
881:       do 530 I = 1, IMTTP
882:         IWK12I = IWK12(I)
883:         IWK11I = IWK11(I)
884:         if ( IWK11I.gt.0 ) then
885:           NN = NN + 1
886:           IWK11(NN) = IWK11I
887:           IWK12(NN) = IWK12I
888:           IWK13(NN) = IWK13(I)
889:           WK4(NN) = WK4(I)
890:         else if ( IWK11I.eq.-18 ) then
891:           NDEAD = NDEAD + 1
892:           LSDED(NDEAD) = IWK12I
893:         end if
894: 530       continue
895:       IMTTP = NN
896:     end if
897:     return
898:   end if
899: C
900:   NDEAD = NDEAD - NFISB
901:   call RANU2( IRAND, R, NN*2, ICON )
902: C
903: C
904: C .... for newly created fission neutron
905: C   IWK7(I) : bank pointer of parent neutron.
906: C   IWK5(I) : bank pointer of generated neutron.
907: C
908: *VOCL LOOP,NOVREC
909:   do 540 I = 1, NFISB
910:     IO = IWK6(I)

```

src/mvp/panlgf.f

```

911:      IWK7(I) = IWK12(I0)
912:      L1      = LSDED(NDEAD+I)
913:      IWK5(I) = L1
914: C
915:      W      = 1. - 2.*R(I)
916:      A      = SQRT(1.-W*W)
917:      PHI     = PI2*R(NN+I)
918:      U      = A*COS(PHI)
919:      V      = A*SIN(PHI)
920:      S      = 1./SQRT(U*U+V*V+W*W)
921:      AAA(IWK5(I)) = U*S
922:      BBB(IWK5(I)) = V*S
923:      CCC(IWK5(I)) = W*S
924:      XXX(IWK5(I)) = XXX(IWK7(I))
925:      YYY(IWK5(I)) = YYY(IWK7(I))
926:      ZZZ(IWK5(I)) = ZZZ(IWK7(I))
927:      WWW(IWK5(I)) = WK7(I)
928:      KKP(IWK5(I)) = KKP(IWK7(I))
929:      IZZ(IWK5(I)) = IZZ(IWK7(I))
930:      TTT(IWK5(I)) = TTT(IWK7(I))
931:      EEE(IWK5(I)) = WK7(I)
932:      KLSF(IWK5(I)) = 0
933: C
934: C---- SET MMAC(L1,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
935: C      FOR THIS PARTICLE
936: C
937:      MMAC(IWK5(I),2) = 0
938:      if ( JLATT.ne.0 ) LEVL(IWK5(I)) = LEVL(IWK7(I))
939:      if ( JIMPT.ne.0 ) XIM(IWK5(I)) = XIM(IWK7(I))
940: CCCCC if ( JTLT.ne.0 ) IBREG(IWK5(I)) = IBREG(IWK7(I))
941:      IBREG(IWK5(I)) = IBREG(IWK7(I))
942:      if ( JTIME.ne.0 ) ITT(IWK5(I)) = ITT(IWK7(I))
943: C
944: 540 continue
945: C
946:      if ( JLATT.ne.0 ) then
947:
948:          do 560 K = 1, NEST
949: *VOCL LOOP,NOVREC
950:          do 550 I = 1, NFISB
951:              LZZ(IWK5(I),K) = LZZ(IWK7(I),K)
952:              LPOS(IWK5(I),K) = LPOS(IWK7(I),K)
953:              if ( JHLAT.ne.0 ) LCRS(IWK5(I),K) = LCRS(IWK7(I),K)
954:              if ( JTLT.ne.0 ) IBSPC(IWK5(I),K) = IBSPC(IWK7(I),K)
955:          550 continue
956:          560 continue
957:          end if
958: C
959: C ==== resample angle only for fission neutrons generated by killing
960: C      parent neutrons
961: C
962: *VOCL LOOP,NOVREC
963:      do 570 I = NFISB + 1, NN
964:          IWK5(I) = IWK12(IWK6(I))
965: C
966:      W      = 1. - 2.*R(I)
967:      A      = SQRT(1.-W*W)
968:      PHI     = PI2*R(NN+I)
969:      U      = A*COS(PHI)
970:      V      = A*SIN(PHI)
971:      S      = 1./SQRT(U*U+V*V+W*W)
972:      AAA(IWK5(I)) = U*S
973:      BBB(IWK5(I)) = V*S
974:      CCC(IWK5(I)) = W*S
975:      EEE(IWK5(I)) = WK7(I)
976: C
977: C---- SET MMAC(L1,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
978: C      FOR THIS PARTICLE
979: C
980:      MMAC(IWK5(I),2) = 0
981: 570 continue
982: C
983: C ... copy or change optional bank parameters
984: C
985: C      IWK7(1:NFISB) : bank pointer of parent neutron.
986: C      IWK5(1:NN) : bank pointer of generated neutron.
987: C
988:      do 620 K = 1, KDBNK(0)
989:          if ( K.eq.KDBNK(1) ) then
990: C
991: C
992: C ... birth weight is current weight !!!
993: C
994: *VOCL LOOP,NOVREC
995:          do 580 I = 1, NN
996:              DBNK(IWK5(I),KDBNK(1)) = WWW(IWK5(I))
997:          580 continue
998: C
999: C ... time of birth is just now !!!
1000: C
1001:      else if ( K.eq.KDBNK(2) ) then
1002: *VOCL LOOP,NOVREC
1003:          do 590 I = 1, NN
1004:              DBNK(IWK5(I),KDBNK(2)) = TTT(IWK5(I))
1005:          590 continue
1006: C
1007: C ... energy on birth !!!
1008: C
1009:      else if ( K.eq.KDBNK(3) ) then
1010: *VOCL LOOP,NOVREC
1011:          do 600 I = 1, NN
1012:              DBNK(IWK5(I),KDBNK(3)) = EEE(IWK5(I))
1013:          600 continue
1014: C
1015: C ... other optional bank parameters are only copy of those
1016: C      of parents
1017: C      or gadget for I=NFISB+1,NN ....(don't know how to set)
1018: C
1019:      else
1020: *VOCL LOOP,NOVREC
1021:          do 610 I = 1, NFISB
1022:              DBNK(IWK5(I),K) = DBNK(IWK7(I),K)
1023:          610 continue
1024:          end if
1025:          620 continue
1026: C
1027:          do 670 K = 1, KIBNK(0)
1028:              if ( K.eq.KIBNK(4) ) then
1029: C
1030: C ... increment "particle generation" if necessary
1031: C
1032: *VOCL LOOP,NOVREC
1033:              do 630 I = 1, NFISB
1034:                  IBNK(IWK5(I),KIBNK(4)) = IBNK(IWK7(I),KIBNK(4)) + 1
1035:              630 continue
1036: *VOCL LOOP,NOVREC
1037:              do 640 I = NFISB + 1, NN
1038:                  IBNK(IWK5(I),KIBNK(4)) = IBNK(IWK5(I),KIBNK(4)) + 1
1039:              640 continue
1040: C

```

src/mvp/panlgf.f

```

1041: C    ... reset flight path counter
1042: C
1043:         else if ( K.eq.KIBNK(5) ) then
1044:             do 650 I = 1, NFISB
1045:                 IBNK(IWK5(I),K) = 0
1046:             650         continue
1047:         else
1048: *VOCL LOOP,NOVREC
1049:             do 660 I = 1, NFISB
1050:                 IBNK(IWK5(I),K) = IBNK(IWK7(I),K)
1051:             660         continue
1052:             end if
1053:         670         continue
1054:
1055: C
1056:         NFISB = NN
1057: C
1058:         if ( JPHOT.eq.0 ) then
1059: C ..... put into dead particle stack parent neutrons
1060: C ..... inducing analog fission
1061: C
1062: *VOCL LOOP,NOVREC
1063:         do 680 I = 1, I1
1064:             NDEAD = NDEAD + 1
1065:             LSDED(NDEAD) = IWK12(I)
1066:         680         continue
1067:         else if ( JPHOT.eq.1 ) then
1068:             N = 0
1069: C
1070: C ..... remove parent neutrons for which fission neutron are
1071: C ..... selected between fission neutrons and photons
1072: C ..... from photon generating stack
1073: *VOCL LOOP,NOVREC
1074:         do 690 I = 1, IMTTP
1075:             IWK12I = IWK12(I)
1076:             IWK11I = IWK11(I)
1077:             if ( IWK11I.gt.0 ) then
1078:                 N = N + 1
1079:                 IWK11(N) = IWK11I
1080:                 IWK12(N) = IWK12I
1081:                 IWK13(N) = IWK13(I)
1082:                 WK4(N) = WK4(I)
1083:             else if ( IWK11I.eq.-18 ) then
1084:                 NDEAD = NDEAD + 1
1085:                 LSDED(NDEAD) = IWK12I
1086:             end if
1087:         690         continue
1088:             IMTTP = N
1089:         end if
1090: C
1091:         else
1092: C
1093: C=====
1094: C      Eigenvalue problem
1095: C=====
1096: C      Fission neutrons are banked in fission bank
1097: C      (XXXF,YYYF,ZZZF,IZZF.....)
1098: C
1099:         if ( NFISS.le.0 ) return
1100: C
1101: C***** Bank fission neutrons *****
1102: C
1103:         NCS = 1
1104:         700         continue
1105: C

```

```

1106: C .....
1107: C
1108: C      IWK5(I) : position in fission bank
1109: C      IWK1(I) : colliding nuclide
1110: C      IWK6(I) : mother neutron
1111: C      NN      : number of fission neutrons to be banked
1112: C .....
1113: C
1114:         if ( NFISS.le.NBANK ) then
1115:             NFISR = 0
1116: C
1117:             MA = 0
1118:             do 710 I = NCS, NAF
1119:                 MA = MAX(MA,IWK4(I))
1120:             710         continue
1121:             NN = 0
1122:             do 730 K = 1, MA
1123:                 do 720 I = NCS, NAF
1124:                     if ( IWK4(I).ge.K ) then
1125:                         NN = NN + 1
1126:                         IWK6(NN) = IWK12(I)
1127:                         IWK1(NN) = IWK13(I)
1128:                     end if
1129:                 720         continue
1130:             730         continue
1131: C
1132:         else
1133:             NNN = 0
1134:             NN = 0
1135:             do 750 I = NCS, NAF
1136:                 NNN = NNN + IWK4(I)
1137:                 ICS = I
1138:                 if ( NNN.gt.NBANK ) go to 760
1139:                 do 740 K = 1, IWK4(I)
1140:                     NN = NN + 1
1141:                     IWK6(NN) = IWK12(I)
1142:                     IWK1(NN) = IWK13(I)
1143:                 740         continue
1144:             750         continue
1145:
1146:             NFISR = 0
1147:             go to 770
1148:
1149:         760         continue
1150:
1151:             NFISR = NFISS - NN
1152:             NFISS = NN
1153:             NCS = ICS
1154:
1155:         770         continue
1156:         end if
1157: C
1158:         if ( NFISB.ge.NFBANK ) then
1159:             call RANU2( IRAND, R, NFISS, ICON )
1160: C
1161:             NN = 0
1162:
1163: *VOCL LOOP,NOVREC
1164:             do 780 I = 1, NFISS
1165: C/#IF ROUNDOFF(NEAREST)
1166:                 NCN2 = NCNTR(2) + I
1167:                 IP = MIN( INT(NCN2*R(I)) + 1, NCN2 )
1168: C/#ELSE
1169:                 IP = (NCNTR(2)+I)*R(I) + 1
1170: C/#ENDIF

```


src/mvp/panlgf.f

```

1171:         if ( IP.le.NFBANK ) then
1172:             XXXF(IP) = XXX(IWK6(I))
1173:             YYYYF(IP) = YYYY(IWK6(I))
1174:             ZZZF(IP) = ZZZ(IWK6(I))
1175:             EEEF(IP) = EEE(IWK6(I))
1176:             IZZF(IP) = IZZ(IWK6(I))
1177:             INUF(IP) = IWK1(I)
1178:             if ( JLATT.ne.0 ) LEVLF(IP) = LEVL(IWK6(I))
1179:             if ( JTLLT.ne.0 ) IBRGF(IP) = IBREG(IWK6(I))
1180:             IBRGF(IP) = IBREG(IWK6(I))
1181: C
1182:             NN = NN + 1
1183:             IWK5(NN) = IP
1184:             IWK6(NN) = IWK6(I)
1185:         end if
1186: 780 continue
1187:
1188:         if ( JLATT.ne.0.and.NN.gt.0 ) then
1189:             do 800 K = 1, NEST
1190: *VOCL LOOP,NOVREC
1191:                 do 790 I = 1, NN
1192:                     LZZF(IWK5(I),K) = LZZ(IWK6(I),K)
1193:                     LPOSF(IWK5(I),K) = LPOS(IWK6(I),K)
1194:                     if ( JHLAT.ne.0 ) then
1195:                         LCRSF(IWK5(I),K) = LCRS(IWK6(I),K)
1196:                     end if
1197:                     if ( JTLLT.ne.0 ) then
1198:                         IBSPF(IWK5(I),K) = IBSPC(IWK6(I),K)
1199:                     end if
1200:                 continue
1201:             800 continue
1202:         end if
1203: C
1204:     else
1205:         NTEMP = MIN(NFISS,NFBANK-NFISB)
1206: C
1207:         do 810 I = 1, NTEMP
1208:             IWK5(I) = NFISB + I
1209:             XXXF(IWK5(I)) = XXX(IWK6(I))
1210:             YYYYF(IWK5(I)) = YYYY(IWK6(I))
1211:             ZZZF(IWK5(I)) = ZZZ(IWK6(I))
1212:             EEEF(IWK5(I)) = EEE(IWK6(I))
1213:             IZZF(IWK5(I)) = IZZ(IWK6(I))
1214:             INUF(IWK5(I)) = IWK1(I)
1215: C
1216:             if ( JLATT.ne.0 ) then
1217:                 LEVLF(IWK5(I)) = LEVL(IWK6(I))
1218:             end if
1219:             if ( JTLLT.ne.0 ) then
1220:                 IBRGF(IWK5(I)) = IBREG(IWK6(I))
1221:             end if
1222:             IBRGF(IWK5(I)) = IBREG(IWK6(I))
1223: 810 continue
1224:
1225:         if ( JLATT.ne.0 ) then
1226:             do 830 K = 1, NEST
1227:                 do 820 I = 1, NTEMP
1228:                     LZZF(IWK5(I),K) = LZZ(IWK6(I),K)
1229:                     LPOSF(IWK5(I),K) = LPOS(IWK6(I),K)
1230:                     if ( JHLAT.ne.0 ) then
1231:                         LCRSF(IWK5(I),K) = LCRS(IWK6(I),K)
1232:                     end if
1233:                     if ( JTLLT.ne.0 ) then
1234:                         IBSPF(IWK5(I),K) = IBSPC(IWK6(I),K)
1235:                     end if

```

```

1236: 820 continue
1237: 830 continue
1238:         end if
1239: C
1240:         NFISB = NFISB + NTEMP
1241:         NN = NTEMP
1242: C
1243:         if ( NTEMP.lt.NFISS ) then
1244:             call RANU2( IRAND, R, NFISS-NTEMP, ICON )
1245:             NTEMP1 = NTEMP + 1
1246: *VOCL LOOP,NOVREC
1247:             do 840 I = NTEMP1, NFISS
1248: C/#IF ROUNDOFF(NEAREST)
1249:                 NFNI = NFBANK - NTEMP + I
1250:                 IP = MIN(INT(NFNI*R(I-NTEMP))+1,NFNI)
1251: C/#ELSE
1252:                 IP = (NFBANK-NTEMP+I)*R(I-NTEMP) + 1
1253: C/#ENDIF
1254:             if ( IP.le.NFBANK ) then
1255:                 XXXF(IP) = XXX(IWK6(I))
1256:                 YYYYF(IP) = YYYY(IWK6(I))
1257:                 ZZZF(IP) = ZZZ(IWK6(I))
1258:                 EEEF(IP) = EEE(IWK6(I))
1259:                 IZZF(IP) = IZZ(IWK6(I))
1260:                 INUF(IP) = IWK1(I)
1261: C
1262:                 if ( JLATT.ne.0 ) LEVLF(IP) = LEVL(IWK6(I))
1263:                 if ( JTLLT.ne.0 ) IBRGF(IP) = IBREG(IWK6(I))
1264:                 IBRGF(IP) = IBREG(IWK6(I))
1265: C
1266:                 NN = NN + 1
1267:                 IWK5(NN) = IP
1268:                 IWK6(NN) = IWK6(I)
1269:             end if
1270: 840 continue
1271:             if ( JLATT.ne.0 ) then
1272:                 do 860 K = 1, NEST
1273: *VOCL LOOP,NOVREC
1274:                     do 850 I = NTEMP1, NN
1275:                         LZZF(IWK5(I),K) = LZZ(IWK6(I),K)
1276:                         LPOSF(IWK5(I),K) = LPOS(IWK6(I),K)
1277:                         if ( JHLAT.ne.0 ) then
1278:                             LCRSF(IWK5(I),K) = LCRS(IWK6(I),K)
1279:                         end if
1280:                         if ( JTLLT.ne.0 ) then
1281:                             IBSPF(IWK5(I),K) = IBSPC(IWK6(I),K)
1282:                         end if
1283:                     continue
1284:                 860 continue
1285:             end if
1286:         end if
1287: C
1288: C**** No. of total fission neutron generated in this batch
1289: C
1290:         NCNTR(2) = NCNTR(2) + NFISS
1291: C
1292:         WCNTR( 2) = WCNTR( 2) + W
1293:         if ( NFISR.gt.0 ) then
1294:             NFISS = NFISR
1295:             go to 700
1296:         end if
1297: C
1298:     end if
1299: C
1300:     return

```

src/mvp/panlgf.f

1301: end



src/mvp/pfcninp.f

```

1:      subroutine PFCNINP( IMMSG, A, IA, LIMIT, LAST, JDEBG, NMAT, NREG,
2:      &                    NUCPN, NPFCN, LPFCNR, NCIDPN, MNUCPN, LPIDPN,
3:      &                    JTLLT, IDMAT, KMAT, KREG, KREGI, KCELI, NINPZ,
4:      &                    ISPNM, ISUSP, KSUZ, IRGSP, NEST, NSPACE,
5:      &                    NSUZON, NZONE, TNAMS, NNAMES, ITRNM, NTREG,
6:      &                    IPTRG, LPTRG, KR, NRRRO )
7: C=<MVP>=====
8: C  purpose:  input array data for probability of forced collision
9: C            nuclides for photon incident.
10: C  called in:  INTRO
11: C  calls:
12: C=====
13:      include '../shared/INC/_LNAM'
14: C
15:      real A(LIMIT)
16:      integer IA(LIMIT), JDEBG(*)
17: C
18: C      .... data for photonuclear ....
19:      integer MNUCPN(*), LPIDPN(*)
20:      character NCIDPN(*)*16
21: C
22: C      .... data for material ....
23:      integer IDMAT(NMAT)
24: C
25: C      .... data for region name matching ....
26:      integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ),
27:      &      ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
28:      &      KSUZ(NINPZ,2), IRGSP(2,NREG)
29:      integer ITRNM(NTREG), IPTRG(NTREG+1), LPTRG(*), KR(NRRRO)
30:      character TNAMS(NNAMES)*(LNAM)
31: C
32: C      .... local variable ....
33:      character LINE*72, CHAR*4, NM*5
34: C
35: C-----
36: C
37: C      ..... check and initialize .....
38:      if ( NUCPN.le.0 ) go to 901
39:      if ( NREG .le.0 ) go to 903
40:      NPFCN = 0
41:      JSKIP = 0
42:      NUCPN1 = NUCPN + 1
43:      NN = NUCPN1 * NREG
44:      if ( LPFCNR.eq.0 )
45:      &      call KEPV( A(1), 'PFCNR', LPFCNR, NN, 'R4', LAST, 0.0, JDEBG )
46: C
47: C=====
48: C  region-wise form:  PFCN( !region-name( data ... ) ... )
49: C  all-region form:  PFCN( data ... )
50: C=====
51: C
52:      100 call FPROBE( IPP, ' (', LINE )
53:      if ( IPP.eq.0 ) then
54:      call GTLINE( IEND )
55:      if ( IEND.ne.0 ) go to 905
56:      go to 100
57:      end if
58:      if ( LINE(1:1).ne.'!' ) go to 200
59:      CHAR(1:2) = '()'
60: C
61: C      >>> !region-name( ... )
62: C
63:      110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
64: C
65:      if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.'') go to 300      ! end

```

```

of input
66:      if ( NLEN.eq.0 ) then
67:      if ( IEND.eq.0 ) go to 110
68:      go to 907
69:      end if
70:      NR = 0
71:      IK = INDEX( LINE(:NLEN), '@' )
72:      if ( IK.gt.0 ) then
73: C
74: C      ..... @NAME without wildcard character .....
75:      if ( INDEX( LINE(:NLEN), '?' ).eq.0 .and.
76:      &      INDEX( LINE(:NLEN), '*' ).eq.0 ) then
77:      call CBSINC( TNAMS, NNAMES, LINE(IK:NLEN), IPOS )
78:      if ( IPOS.eq.0 ) then
79:      write( IMMSG, 910 ) LINE(IK:NLEN)
80:      call CNTERR( 'FATAL' )
81:      call DMREAD( ' ', ID1, N1, IIER )
82:      if ( IIER.ne.0 ) go to 911
83:      go to 110
84:      end if
85:      do I = 1, NTREG
86:      if ( ITRNM(I).lt.0.and.IPOS.eq.abs(ITRNM(I)) ) go to 120
87:      end do
88:      go to 913
89:      120      IRD = I
90:      if ( JSKIP.ne.0 ) then
91:      call DMREAD( ' ', ID1, N1, IIER )
92:      if ( IIER.ne.0 ) go to 911
93:      go to 110
94:      end if
95:      do K = IPTRG(IRD), IPTRG(IRD+1)-1
96:      NR = NR + 1
97:      KR(NR) = LPTRG(K)
98:      end do
99: C
100: C      ..... @NAME with wildcard character .....
101:      else
102:      do J = 1, NTREG
103:      if ( ITRNM(J).lt.0 ) then
104:      if ( IMATCH( LINE(IK:NLEN), TNAMS(abs(ITRNM(J))) ) ) .ne.0 )
105:      &      then
106:      do 130 I = IPTRG(J), IPTRG(J+1)-1
107:      do K = 1, NR
108:      if ( KR(K).eq.LPTRG(I) ) go to 130
109:      end do
110:      NR = NR + 1
111:      KR(NR) = LPTRG(I)
112:      130      continue
113:      end if
114:      end if
115:      end do
116:      end if
117: C
118: C      ..... regular region name .....
119:      else if ( IK.eq.0 ) then
120:      IBEGIN = 0
121:      JEND = 0
122:      150      continue
123:      call RGFIND( IMMSG, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
124:      &      JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
125:      &      KSUZ, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
126:      &      NNAMES )
127:      if ( IERR.ne.0 ) then
128:      write( IMMSG, 916 ) LINE(:NLEN)
129:      call CNTERR( 'FATAL' )

```

src/mvp/pfcninp.f

```

130:      call DMREAD( ' ', IDM1, N1, IIER )
131:      if ( IIER.ne.0 ) go to 911
132:      go to 110
133:    end if
134:    if ( IREG.ne.0 ) then
135:      do K = 1, NR
136:        if ( KR(K).eq.IREG ) go to 150
137:      end do
138:      NR = NR + 1
139:      KR(NR) = IREG
140:    end if
141:    if ( JEND.eq.0 ) go to 180
142:  end if
143: C
144: C ..... read the data (probability and nuclide name) .....
145: ND = -1
146: NDN = 0
147: NM = ' '
148: 170 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
149: if ( IEND.ne.0 ) go to 917
150: ND = ND + 1
151: if ( ND.eq.0 ) then
152: C ..... probability .....
153: read(LINE(:NLEN),*) PFCN
154: CM2016 if ( PFCN.le.0.0 ) go to 110
155: if ( PFCN.le.0.0 ) then
156:   if ( ITERM.ne.0 ) go to 110
157:   171 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
158:   if ( ITERM.ne.0 ) go to 110
159:   if ( IEND.ne.0 ) go to 917
160:   go to 171
161: end if
162: if ( PFCN.ge.1.0 ) go to 921
163: do K = 1, NR
164:   I1 = NUCPN1 * (KR(K)-1)
165:   A(LPFCNR+I1) = PFCN
166: end do
167: else
168: C ..... nuclide names .....
169: if ( NLEN.eq.5 ) then
170:   NM = LINE(:NLEN)
171: else if ( NLEN.eq.3 ) then
172:   NM = LINE(:NLEN) // ' '
173: else if ( NLEN.eq.2 ) then
174:   NM = LINE(:NLEN) // ' '
175: else if ( NLEN.eq.1 ) then
176:   NM = LINE(:NLEN) // '0 '
177: else
178:   go to 929
179: end if
180: IKW = 0
181: if ( NM(3:5).eq.'000' ) then
182:   IKW = 1
183: else if ( NM(3:3).eq.'*' ) then
184:   IKW = 1
185: else if ( NM(2:2).eq.'*' ) then
186:   IKW = 1
187:   NM(2:2) = '0'
188: else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
189:   IKW = 1
190: end if
191: if ( IKW.eq.0 ) then
192:   do I = 1, NUCPN
193:     if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 180
194:   end do

```

```

195:      write(IMG,891) NM, (KR(K),K=1,NR)
196:      call CNTERR( 'WARNING' )
197:      ND = ND - 1
198:      go to 190
199: 180 do K = 1, NR
200:    I1 = NUCPN1 * (KR(K)-1) + I
201:    A(LPFCNR+I1) = 1
202:  end do
203:  else if ( IKW.eq.1 ) then
204:    II = 0
205: 184 do I = II+1, NUCPN
206:    if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 185
207:  end do
208:  if ( II.le.0 ) then
209:    write(IMG,891) NM(1:2), (KR(K),K=1,NR)
210:    call CNTERR( 'WARNING' )
211:    ND = ND - 1
212:  end if
213:  go to 190
214: 185 do K = 1, NR
215:    I1 = NUCPN1 * (KR(K)-1) + I
216:    A(LPFCNR+I1) = 1
217:  end do
218:  if ( I.lt.NUCPN ) then
219:    II = I
220:    go to 184
221:  end if
222: end if
223: 190 continue
224: end if
225: if ( ITERM.eq.0 ) go to 170
226: go to 110
227: C
228: C >>> all region
229: C
230: 200 continue
231: ND = -1
232: NDN = 0
233: 210 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
234: if ( IEND.ne.0 ) go to 919
235: ND = ND + 1
236: if ( ND.eq.0 ) then
237:   read(LINE(:NLEN),*) PFCN
238: CM2016 if ( PFCN.le.0.0 ) go to 300
239:   if ( PFCN.le.0.0 ) then
240:     if ( ITERM.ne.0 ) go to 300
241:     211 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
242:     if ( ITERM.ne.0 ) go to 300
243:     if ( IEND.ne.0 ) go to 919
244:     go to 211
245:   end if
246:   if ( PFCN.ge.1.0 ) go to 921
247:   do K = 1, NREG
248:     I1 = NUCPN1 * (K-1)
249:     A(LPFCNR+I1) = PFCN
250:   end do
251: else
252:   if ( NLEN.eq.5 ) then
253:     NM = LINE(:NLEN)
254:   else if ( NLEN.eq.3 ) then
255:     NM = LINE(:NLEN) // ' '
256:   else if ( NLEN.eq.2 ) then
257:     NM = LINE(:NLEN) // ' '
258:   else if ( NLEN.eq.1 ) then
259:     NM = LINE(:NLEN) // '0 '

```

src/mvp/pfcninp.f

```

260:         else
261:             go to 929
262:         end if
263:         IKW      = 0
264:         if ( NM(3:5).eq.'000' ) then
265:             IKW = 1
266:         else if ( NM(3:3).eq.'*' ) then
267:             IKW = 1
268:         else if ( NM(2:2).eq.'*' ) then
269:             IKW = 1
270:             NM(2:2) = '0'
271:         else if ( NM(3:3).eq.'N'.or. NM(3:3).eq.' ' ) then
272:             IKW = 1
273:         end if
274:         if ( IKW.eq.0 ) then
275:             do I = 1, NUCPN
276:                 if ( NM(I:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 220
277:             end do
278:             write(IMG,991) NM
279:             call CNTERR( 'WARNING' )
280:             ND      = ND - 1
281:             go to 230
282:         220 do K = 1, NREG
283:             I1      = NUCPN1 * (K-1) + I
284:             A(LPFCNR+I1) = 1
285:         end do
286:         else if ( IKW.eq.1 ) then
287:             II      = 0
288:         224 do I = II+1, NUCPN
289:             if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 225
290:         end do
291:         if ( II.le.0 ) then
292:             write(IMG,991) NM(1:2)
293:             call CNTERR( 'WARNING' )
294:             ND      = ND - 1
295:         end if
296:         go to 230
297:         225 do K = 1, NREG
298:             I1      = NUCPN1 * (K-1) + I
299:             A(LPFCNR+I1) = 1
300:         end do
301:         if ( I.lt.NUCPN ) then
302:             II      = I
303:             go to 224
304:         end if
305:         end if
306:         230 continue
307:         end if
308:         if ( ITERM.eq.0 ) go to 210
309: C
310: C      >>> check nuclides in the region
311: C
312: 300 continue
313:         do K = 1, NREG
314:             I1      = NUCPN1 * (K-1) + LPFCNR
315:             if ( A(I1).gt.0.0 ) then
316:                 do I = 1, NUCPN
317:                     A(I1+I) = -A(I1+I)
318:                 end do
319:                 do I = 1, NINPZ
320:                     if ( K.eq.KREG(I).and.KMAT(I).gt.0 ) then
321:                         do MM = 1, NMAT
322:                             if ( KMAT(I).eq.IDMAT(MM) ) go to 310
323:                         end do
324:                     go to 320

```

```

325: 310         continue
326:             do J = MNUCPN(MM), MNUCPN(MM+1)-1
327:                 A(I1+LPIDPN(J)) = abs(A(I1+LPIDPN(J)))
328:             end do
329:         320 continue
330:         end if
331:         end do
332:         do I = 1, NUCPN
333:             if ( A(I1+I).gt.0.0 ) then
334:                 NPFCN      = NPFCN + 1
335:                 go to 330
336:             end if
337:         end do
338:         A(I1) = 0 ! this region hasn't PFCN.
339:         330 continue
340:             do I = 1, NUCPN
341:                 if ( A(I1+I).lt.0.0 ) A(I1+I) = 0
342:             end do
343:         end if
344:         end do
345:         return
346: C
347: C      .... warning message ....
348: 891 format(/' !!!(pfcninp) Nuclide name <','a,> is undefined as',
349:           & ' photonuclear.':'
350:           &/' these region number:','15i6
351:           &/(21x,15i6))
352: C
353: C      ..... error process .....
354: 901 write(IMG,902)
355:         call PRSTOP( 1, 'Problem in order of input data.' )
356:         go to 999
357: 903 write(IMG,904)
358:         call PRSTOP( 1, 'Problem in order of input data.' )
359:         go to 999
360: 905 write(IMG,906)
361:         call PRSTOP( 1, 'Unexpected end of input file.' )
362:         go to 999
363: 907 write(IMG,908)
364:         call PRSTOP( 1, 'Unexpected end of input file.' )
365:         go to 999
366: 911 write(IMG,912)
367:         call PRSTOP( 1, 'Error in region dependent data input.' )
368:         go to 999
369: 913 write(IMG,914) (ITRNM(I),I=1,NTREG)
370:         call PRSTOP( 1, 'Problem in region dependent data input.' )
371:         go to 999
372: 917 write(IMG,918)
373:         call PRSTOP( 1, 'Unexpected end of input file.' )
374:         go to 999
375: 919 write(IMG,920)
376:         call PRSTOP( 1, 'Unexpected end of input file.' )
377:         go to 999
378: 921 write(IMG,922) PFCN
379:         call PRSTOP( 1, 'Illegal probability data in input file.' )
380:         go to 999
381: 929 write(IMG,930) NLEN
382:         call PRSTOP( 1, 'Incomplete data structure of input file.' )
383:         go to 999
384: 902 format(/' XXX(pfcninp) Number of "photonuclear nuclide" (NUCPN)',
385:           & ' is not determined before photonuclear dependent data input.'
386:           &/' Your input for <PFCN> must be placed after',
387:           & ' $XSEC block.':'
388: 904 format(/' XXX(pfcninp) Number of "region" (NREG) is not determin',
389:           & 'ed before REGION dependent data input.'

```

src/mvp/pfcninp.f

```
390:      &'      Your input for <PFCN> must be placed after',
391:      &' $GEOMETRY block.')
```

906 format('// XXX(pfcninp) End of file is encountered for <PFCN>.')
908 format('// XXX(pfcninp) End of file is encountered during',
&' '!region-name(...)" search for <PFCN>.')
910 format('// XXX(pfcninp) Tally region <',a,'> does not exist.'
&' Data of <PFCN> for this region are meaningless.')

912 format('// XXX(pfcninp) Unexpected end of data or input error',
&' during skipping unnecessary or invalid data.'
&' (REGION dependent data input)')

914 format('// XXX(pfcninp) Fatal error for tally-region name (ITRNM).'
&' (i5,i10))
916 format('// XXX(pfcninp) Region <',a,'> is not defined or invalid',
&' name.'

&' Data of <PFCN> for this region are meaningless.')

918 format('// XXX(pfcninp) End of file is encountered during data in',
&' '!region-name(...)" for <PFCN>.')
920 format('// XXX(pfcninp) End of file is encountered during data in',
&' "(...)" of all-region for <PFCN>.')
922 format('// XXX(pfcninp) Probability of forced collision nuclide',
&' is greater than 1 for <PFCN>.'
&' probability=',lpe12.5)

930 format('// XXX(pfcninp) Length of data in nuclide name is',
&' illegal for <PFCN>.'
&' NLEN=',i5)

end

src/mvp/pfcnnuc.f

```

1:      subroutine PFCNNUC( IOW, IRAND,
2:      N  NMAT, NREG, NBANK, NUC,  MB,  NSMAC, NSMIC, NSMICPN,
3:      N  NUCPN, NUCPNI, DNSTP, LPDNP, IATMT, NATMT, DNSTPN, MNUCPN, LPIDPN,
4:      N  IZTBNP,
5:      V  PFCNR,
6:      X  XLBPN1, SMAC, KMAC, MMAC, SMIC, SMICPN, ETOPLPN, EBOTLPN,
7:      %  LSCLP, NCOLP, NCOLPP, NCOLPG,
8:      B  WWW, EEE, IBREG,
9:      W  R,
10:     W  IWK1, IWK2, IWK3, IWK4, IWK5 )
11: C=<MVP>=====
12: C purpose: select the collided nuclides with photon by the probability
13: C         of forced collision nuclide.
14: C called in: PHOTNUC
15: C call: RANU2
16: C=====
17:      implicit real*8 (A-H,O-Z)
18: C
19: C ..... variance reduction data
20:      real PFCNR(NUCPN+1,NREG)
21: C
22: C ..... cross section data
23:      real ETOPLPN, EBOTLPN
24:      real XLBPN1(NUCPNI,6)
25: C
26: C ..... material data
27:      real DNSTP(*), DNSTPN(*)
28:      integer LPDNP(NMAT+1), IATMT(*), NATMT(*), MNUCPN(NMAT+1)
29:      &      LPIDPN(*), IZTBNP(*)
30: C
31: C ..... sigma bank
32:      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
33:      &      SMICPN(NBANK,NUCPN,NSMICPN)
34:      integer KMAC(NBANK,MB,2), MMAC(NBANK,2)
35: C
36: C ..... particle bank
37:      real WWW(NBANK), EEE(NBANK)
38:      integer IBREG(NBANK), LSCLP(NBANK)
39: C
40: C ..... working area
41:      real R(6*NBANK)
42:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
43:      &      IWK5(NBANK)
44: C
45: C ..... local data
46:      parameter (MAXNPN=200)
47:      real PRBNPN(MAXNPN), CUMNPN(MAXNPN)
48:      integer MRKNPN(MAXNPN)
49: C
50: C <note> The atom selected in GTMACPN is ignored.
51: C
52: C-----
53: C**** sort by collided material and region
54: C-----
55: C
56:     N      = 0
57:     K      = 0
58:     do I = 1, NCOLP
59:         IP      = LSCLP(I)
60:         if ( EEE(IP).ge.EBOTLPN.and.EEE(IP).le.ETOTLPN ) then
61:             MI      = MMAC(IP,1)
62:             MAT      = KMAC(IP,MI,2)
63:             IR      = IBREG(IP)
64:             N      = N + 1
65:             IWK1(N) = MAT * 10000 + IR

```

```

66:             IWK2(N) = IP
67:         else
68:             K      = K + 1
69:             IWK3(K) = IP          ! IWK3 stores the bank pointer of phot
on interaction.
70:         end if
71:     end do
72:     if ( N.le.0 ) go to 900      ! energies of all photons are less tha
n the threshold energy.
73:     NCOLPG = N
74:     NCOLPP = K
75:     call MSORTI( NCOLPG, IWK1, IWK2, IWK4, IWK5 )
76: C
77: C-----
78: C**** select the nuclides colliding with photon
79: C-----
80: C
81:     call RANU2( IRAND, R, NCOLPG, ICON )
82:     K      = 0
83:     do I = 1, NCOLPG
84:         IP      = IWK2(I)
85:         MAT      = IWK1(I) / 10000
86:         IR      = IWK1(I) - MAT * 10000
87:         N1      = 0
88:         N3      = 0
89:         do J = LPDNP(MAT), LPDNP(MAT+1)-1
90:             NI      = IATMT(J)
91:             IZ      = NATMT(NI)
92:             ST      = SMIC(IP,NI,1)
93:             N2      = N1
94:             do M = MNUCPN(MAT), MNUCPN(MAT+1)-1
95:                 if ( IZTBNP(M).eq.IZ ) then
96:                     N1      = N1 + 1
97:                     if ( N1.gt.MAXNPN ) go to 911
98:                     PRBNPN(N1) = (ST + SMICPN(IP,LPIDPN(M),1)) * DNSTPN(M)
99:                     if ( PFCNR(1,IR).le.0.0 ) then
100:                        MRKNPN(N1) = - M
101:                     else if ( PFCNR(LPIDPN(M)+1,IR).gt.0.0 ) then
102:                        MRKNPN(N1) = M
103:                     else
104:                        MRKNPN(N1) = - M
105:                     end if
106:                 end if
107:             end do
108:             if ( N1.eq.N2 ) then          ! for atom without photonuclear
109:                 N1      = N1 + 1
110:                 N3      = N3 + 1
111:                 if ( N1.gt.MAXNPN ) go to 911
112:                 PRBNPN(N1) = ST * DNSTP(J)
113:                 MRKNPN(N1) = - NI * 1000
114:             end if
115:         end do
116:         if ( N1.eq.N3 ) then          ! all target atom are no photonuclear.
117:             NCOLPP = NCOLPP + 1
118:             IWK3(NCOLPP) = IP
119:             go to 190
120:         end if
121:         if ( N1.eq.1 ) then
122:             J      = 1
123:             S1      = 1
124:             S2      = 1
125:             S0      = 1
126:             MRKNPN(N1) = - abs(MRKNPN(N1))
127:             go to 110
128:         end if

```

src/mvp/pfcnnuc.f

```

129: C
130: C ..... make the probability table for sampling
131:   S1      = 0
132:   S2      = 0
133:   do J = 1, N1
134:     if ( MRKNPN(J).gt.0 ) then
135:       S1      = S1 + PRBNPN(J)
136:     else
137:       S2      = S2 + PRBNPN(J)
138:     end if
139:   end do
140:   S0      = S1 + S2
141:   do J = 1, N1
142:     if ( MRKNPN(J).gt.0 ) then
143:       PRBNPN(J) = PFCNR(1,IR) * PRBNPN(J) / S1
144:     else
145:       PRBNPN(J) = (1 - PFCNR(1,IR)) * PRBNPN(J) / S2
146:     end if
147:   end do
148:   CUMNPN(1) = PRBNPN(1)
149:   do J = 2, N1
150:     CUMNPN(J) = CUMNPN(J-1) + PRBNPN(J)
151:   end do
152:   do J = 1, N1
153:     CUMNPN(J) = CUMNPN(J) / CUMNPN(N1)
154:   end do
155:   do J = 1, N1-1
156:     if ( CUMNPN(J).gt.R(I) ) go to 110
157:   end do
158:   J      = N1
159:   110 MN    = MRKNPN(J)
160: C
161: C ..... select the nuclide, adjust the weight, and check the energy
162: C range of photonuclear
163: C if ( MN.gt.0 ) then
164:   IS      = LPIDPN(MN)
165:   WWW(IP) = WWW(IP) * S1 / S0 / PFCNR(1,IR)      ! forced collision nuclide
166: else
167:   MN      = abs(MN)
168:   if ( MN.ge.1000 ) then
169:     IS      = 0
170:   else
171:     IS      = LPIDPN(MN)
172:   end if
173:   if ( PFCNR(1,IR).gt.0.0.and.N1.gt.1 )
174:     &      WWW(IP) = WWW(IP) * S2 / S0 / (1 - PFCNR(1,IR)) ! no forced collision nuclide
175:   end if
176:   if ( IS.le.0 ) then      ! select nuclide without photo-nuclear data, in natural element
177:     NCOLPP = NCOLPP + 1
178:     IWK3(NCOLPP) = IP
179:     if ( MN.ge.1000 ) KMAC(IP,MMAC(IP,1),1) = MN / 1000
180:     go to 190
181:   else if ( EEE(IP).lt.XLBPN1(IS,3) ) then      ! incident energy is less than lower energy.
182:     NCOLPP = NCOLPP + 1
183:     IWK3(NCOLPP) = IP
184:   else if ( EEE(IP).gt.XLBPN1(IS,4) ) then      ! incident energy is larger than higher energy.
185:     NCOLPP = NCOLPP + 1
186:     IWK3(NCOLPP) = IP
187:   else
188:     K      = K + 1
189:     IWK4(K) = IP
190:     IWK5(K) = IS
191:   end if
192:   IZ      = IZTBN(MN)
193:   do IK = LPDNP(MAT), LPDNP(MAT+1)-1
194:     if ( NATMT(IATMT(IK)).eq.IZ ) go to 120
195:   end do
196:   write(IOW,*) 'XXX(pfcnnuc) Atom collided by photon inter',
197:   &      'action is not found. IZ=',IZ,MAT
198:   stop 666
199:   120 KMAC(IP,MMAC(IP,1),1) = IATMT(IK)
200:   190 continue
201: end do
202: if ( K.le.0 ) go to 900      ! energies of all photons are less than lower energy.
203: NCOLPG = K
204: C
205: C-----
206: C*** process of photo-atomic reaction (photon interaction)
207: C-----
208: C
209:   if ( NCOLPP.gt.0 ) then
210:     do I = 1, NCOLPP
211:       LSCLP(I) = IWK3(I)
212:     end do
213:   end if
214: C
215:   return
216: C
217: C ..... process of no photo-nuclear reaction
218:   900 continue
219:   NCOLPP = NCOLP
220:   NCOLPG = 0
221:   return
222: C
223: C ..... error message
224:   911 write(IOW,912) MAXNPN
225:   912 format(/' XXX(pfcnnuc) Local array size was lacked. MAXNPN=',i8)
226:   stop 667
227: end

```


src/mvp/photnuc.f

```

1:      subroutine PHOTNUC( IOW, IRAND,
2:      N NPKIND,NGROUP,NZONE, NMAT, NREG, NTIME, NBANK, NEVENT,NEST,
3:      N NUC, MB, NSMAC, NSMIC, NSMICPN, NUCPN, NUCPNI,MCXPN,
4:      N NMTPN, NSTAL, NUC_MAX,
5:      M DNSTP, LPDNP, IATMT, NATMT, DNSTPN,MNUCPN,LPIDPN,IZTBPN,NCIDPN,
6:      G KZREG,
7:      V XIMP, WKIL, WSRV, WGTPN, WTIME, WLLIM, MXPGEN,EBOTX, PPNBR,
8:      V NPPNBR,PFCNR, NPFEN, PMT, NPMT, TCUT, TIMEB,
9:      X CX, MCX, KLIB1, KLIB2, XLIB1, NMT,
10:     X CXP, MCXP, KLBPN1, KLBPN2, XLBPN2, NMTP, NPATOM,
11:     X SMAC, LMAC, KMAC, MMAC, SMIC, LMIC, KSPI, SMICPN,
12:     X CXPN, ICXPN, KLBPN1,KLBPN2,XLBPN1,XLBPN2,ETOPLPN,EBOTLPN,
13:     X CRES, KCRES, MCRES, SGTAL, NNCST, INCST,
14:     % LSFFL, NFFL, IZFFL, LSCLP, NCOLP, LSDED, NDEAD, NCOLPP,
15:     B XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE, TTT,
16:     B ITT, XIM, KLSF, KKP, IZZ, IGG, LEVL, LZZ, LPOS,
17:     B LCRS, IBREG, IBSPC, DBNK, IBNK, KDBNK, MDBNK, KIBNK, MIBNK,
18:     T KENPRD,ENGYB, NKILD, WKILD, NSURV, WSRV, NSPLT, WSPLT, NCNTR,
19:     T WCNTR, NMONPNW, MONPNWGT, NMPNWGT, WMPNWGT,
20:     W R, DWK1, DWK2, WK1, WK2, WK3, WK4, WK5,
21:     W IWK1, IWK2, IWK3, IWK4, IWK5, IWK6, IWK7, IWK8, IWK9,
22:     W IWK10, IWK11, IWK12, IWK13, IWK14, IWK15, SMCW, RNU, KPNFG,
23:     & A, H,
24:     & XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZFL, IMDED, IMSLT, IMNLT,
25:     & IMZLT, LZZI, LPOSI, LCRS1, OPTI, PATH, KDET, KLSFI, SMACI,
26:     & MMACI, NPDET, NPLEN, IMPMAX, KZMAT,
27:     & SMICP, SMACP, NPTDS, NGSP, WWD, WWD2, WWR, WSD, WSC,
28:     & NORDDS, NOPSDS,
29:     & WWB, WSB, NPTBE, WWBD, WSBD, WWLD, WSLD)
30: C=<MVP>=====
31: C purpose: generate particles from photonuclear reactions
32: C called in: PHOTR
33: C calls: RANU2, BSVDEC, BSDEC3, BSINC3, GETMIC, GTMICP, GTMICPN,
34: C GMICN1, GMICP1, GMICPN1, GTSCTL, SPLITB, PN4LCT2, MSORTI,
35: C PNKINM, PFCNNUC
36: C=====
37:     implicit real*8 (A-H,O-Z)
38: C
39:     include 'INC/_KPIDS'
40:     include 'INC/_KPMAS'
41:     include 'INC/_NGPS'
42:     include 'INC/_FLAGS'
43:     include 'INC/_WKPHT'
44: C
45:     real A(*)
46:     real H(*)
47: C
48: C ..... variance reduction data
49:     real*8 EBOTX(KPLIM)
50:     real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG),
51:     & WGTPN(NMTPN,NUCPN,NREG), WTIME(NTIME), WLLIM, PPNBR(NUCPN),
52:     % TIMEB(NTIME+1), TCUT,
53:     & PFCNR(NUCPN+1,NREG)
54:     real PMT(2,NUCPN,NREG)
55: C
56: C ..... cross section data
57:     real ETOPLPN, EBOTLPN
58:     real CXPN(MCXPN), XLBPN1(NUCPNI,6), XLBPN2(NUCPNI,NMTPN,2)
59:     integer ICXPN(MCXPN), KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
60:     real CX(MCX), XLIB1(NUC,11), CXP(MCXP), XLBPN2(NPATOM,NMTP,4)
61:     integer KLIB1(NUC,31), KLIB2(NUC,NMT,21), KLBPN1(NPATOM,20),
62:     & KLBPN2(NPATOM,NMTP,4)
63:     real CRES(MCRES)
64:     integer KCRES(NSTAL,4), INCST(NUC,*)
65: C

```

```

66: C ..... material data
67:     real DNSTP(*), DNSTPN(*)
68:     integer LPDNP(NMAT+1), IATMT(*), NATMT(*), MNUCPN(NMAT+1),
69:     & LPIDPN(*), IZTBPN(*)
70:     character NCIDPN(NUCPNI)*16
71: C
72: C ..... sigma bank
73:     real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
74:     & SMICPN(NBANK,NUCPN,NSMICPN), SGTAL(NBANK,NSTAL),
75:     & RNU(NBANK,NUC_MAX,3)
76:     integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8),
77:     & LMAC(8)
78: C
79: C ..... variables for perturbation calculation ...
80:     real*8 WWD(NBANK,NPTDS,NORDDS), WWD2(NBANK,NPTDS)
81:     real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
82:     real*8 WWR(NBANK,NPTCS)
83:     real*8 WSC(NBANK,0:NGSP,NPTCS)
84:     real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
85: c+beff2
86:     real*8 WWB(NBANK)
87:     real*8 WSB(NBANK,0:NGSP)
88:     real*8 WWBD(NBANK)
89:     real*8 WSBD(NBANK,NGSP)
90:     real*8 WWLD(NBANK)
91:     real*8 WSLD(NBANK,NGSP)
92: c-beff2
93: C
94: C ..... stack
95:     integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSCLP(NBANK),
96:     & LSDED(NBANK), NCOLP, NDEAD, NCOLPP, NCOLPN
97: C
98: C ..... particle bank
99:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
100:     & CCC(NBANK), TTT(NBANK), DBNK(NBANK,*)
101:     real WWW(NBANK), EEE(NBANK), XIM(NBANK)
102:     integer ITT(NBANK), KLSF(NBANK), KKP(NBANK), IZZ(NBANK),
103:     & IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST),
104:     & LCRS(NBANK,NEST), IBREG(NBANK), IBSPC(NBANK,0:NEST),
105:     & IBNK(NBANK,*), KDBNK(0:MDBNK), KIBNK(0:MIBNK)
106: c##<2007/03/14:PN4:
107:     integer KPNFG(NBANK)
108: c##>
109: C
110: C ..... tally bank
111:     real*8 WKILD(NGROUP,NREG), NKILD(NGROUP,NREG),
112:     & WSRV(NGROUP,NREG), NSURV(NGROUP,NREG),
113:     & WSPLT(NGROUP,NREG), NSPLT(NGROUP,NREG),
114:     & WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
115:     real ENGYB(*)
116:     real*8 NMPNWGT(*), WMPNWGT(2,*)
117:     integer MONPNWGT(4,*)
118: C
119: C ..... geometry data
120:     integer KZREG(NZONE)
121: c##<2007/03/14:PN4:
122:     integer KZMAT(NZONE)
123: C
124: C ..... data for next event (point) estimator
125:     real*8 XPDET(NPLEN,NPDET)
126:     integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
127: C
128: C ..... stack for imaginary particle
129:     integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(*), IMDED(IMPMAX),
130:     & IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)

```

src/mvp/photnuc.f

```

131: C
132: C ..... particle bank for imaginary particle
133:   real*8 OPTI(IMPMAX), PATH(IMPMAX)
134:   integer LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST),
135:   &      KDTEP(IMPMAX), KLSFI(IMPMAX)
136: C
137: C ..... cross section and geometry data for imaginary particle
138:   real SMACI(*)
139:   integer MMACI(*)
140: c##>
141: C
142: C ..... working area
143:   real*8 DWK1(NBANK), DWK2(NBANK)
144:   real R(6*NBANK), WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
145:   &      WK5(NBANK), SMCW(NBANK,NSMIC)
146:   integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
147:   &      IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
148:   &      IWK9(NBANK), IWK10(NBANK), IWK11(NBANK), IWK12(NBANK),
149:   &      IWK13(NBANK), IWK14(NBANK), IWK15(NBANK)
150: C
151: C ..... local data
152:   parameter (MKWNUC=2000, N30=30)
153:   integer KWNUC(MKWNUC,3)
154:   real*8 MNC2
155:   parameter (PI = 3.1415926535897932D+0)
156:   parameter (PT2 = 2.0D+0*PI)
157:   parameter (PIH = 0.5D+0*PI)
158:   parameter (MNC2 = 9.234905D+8) ! neutron rest energy [eV]
159:   parameter (SMALL = 1.0D-35)
160:   parameter (EBRELA = 2.0D+7) ! energy boundary between non- and relativistic [eV]
161: C
162:   ZERO = 0.0D0
163: C
164: C-----
165: C**** process of forced collided nuclide with photon
166: C-----
167: C
168:   if ( NPFNC.gt.0 ) then
169:     call PFCNNUC( IOW, IRAND, NMAT, NREG, NBANK, NUC, MB, NSMAC,
170:     N      NSMIC, NSMICPN, NUCPN, NUCPNI, DNSTP, LPDNP, IATMT, NATMT,
171:     N      DNSTPN, MNUCPN, LPIDPN, IZTBP,
172:     V      PFCNR,
173:     X      XLBPN1, SMAC, KMAC, MMAC, SMIC, SMICPN, ETOPLPN, EBOTLPN,
174:     %      LSCLP, NCOLP, NCOLPP, NCOLPG,
175:     B      WWW, EEE, IBREG,
176:     W      R, IWK1, IWK2, IWK3, IWK4, IWK5 )
177:     if ( NCOLP.eq.NCOLPP ) go to 900
178:     if ( NCOLPG.eq.0 ) go to 900
179:     go to 1000
180:   end if
181: C
182: C-----
183: C**** sort by collided nuclide and material
184: C-----
185: C
186:   N = 0
187:   K = 0
188:   do I = 1, NCOLP
189:     IP = LSCLP(I)
190:     if ( EEE(IP).ge.EBOTLPN.and.EEE(IP).le.ETOPLPN ) then
191:       MI = MMAC(IP,1)
192:       NI = KMAC(IP,MI,1)
193:       MAT = KMAC(IP,MI,2)
194:       N = N + 1
195:       IWK1(N) = MAT * 1000 + NI
196:       IWK2(N) = IP
197:     else
198:       K = K + 1
199:       IWK3(K) = IP ! IWK3 stores the bank pointer of photon interaction.
200:     end if
201:   end do
202:   if ( N.le.0 ) go to 900 ! energies of all photons are less than the threshold energy.
203:   NCOLPG = N
204:   NCOLPP = K
205:   call MSORTI( NCOLPG, IWK1, IWK2, IWK4, IWK5 )
206: C
207: C-----
208: C**** select the nuclides colliding with photon
209: C-----
210: C
211:   call RANU2( IRAND, R, NCOLPG, ICON )
212:   MN0 = 0
213:   K = 0
214:   RHO = 0
215:   IZ = 0
216: *VOCL LOOP,NOVREC
217:   do N = 1, NCOLPG
218:     IP = IWK2(N)
219:     MN = IWK1(N)
220:     if ( MN.eq.MN0 ) go to 120
221:     MAT = MN / 1000
222:     NI = MN - MAT * 1000
223:     do IK = LPDNP(MAT), LPDNP(MAT+1)-1
224:       if ( IATMT(IK).eq.NI ) go to 110
225:     end do
226:     write(IOW,*) 'XXX(PHOTNUC) Requested atom is not found in',
227:     &      ' photon atom table. NI=',NI,MAT
228:   stop 666
229: 110   RHO = DNSTP(IK)
230:   IZ = NATMT(NI)
231: 120   RHOI = 0
232:   IS = 0
233:   RHOR = RHO * R(N)
234:   do J = MNUCPN(MAT), MNUCPN(MAT+1)-1
235:     if ( IZ.eq.IZTBP(J) ) then
236:       RHOI = RHOI + DNSTPN(J) ! summation of number density by isotopes
237:     if ( IS.eq.0.and.RHOI.ge.RHOR ) IS = LPIDPN(J) ! pre-select collided nuclide
238:   end if
239:   end do
240:   if ( IS.le.0 ) then ! select nuclide without photo-nuclear data, in natural element
241:     NCOLPP = NCOLPP + 1
242:     IWK3(NCOLPP) = IP
243:   else if ( EEE(IP).lt.XLBPN1(IS,3) ) then ! incident energy is less than lower energy.
244:     NCOLPP = NCOLPP + 1
245:     IWK3(NCOLPP) = IP
246:   else if ( EEE(IP).gt.XLBPN1(IS,4) ) then ! incident energy is larger than higher energy.
247:     NCOLPP = NCOLPP + 1
248:     IWK3(NCOLPP) = IP
249:   else ! select collided nuclide
250:     K = K + 1
251:     IWK4(K) = IP
252:     IWK5(K) = IS

```

src/mvp/photnuc.f

```

253:         end if
254:         MN0      = MN
255:       end do
256:       if ( K.le.0 ) go to 900          ! energies of all photons are less tha
n lower energy.
257:       NCOLPG = K
258:       1000 continue
259: C
260: C-----
261: C**** select the nuclides with photo-nuclear reaction
262: C-----
263: C
264:       if ( NPPNBR.gt.0 ) then          ! select with probability PPNBR
265:         call RANU2( IRAND, R, NCOLPG, ICON )
266:         K      = 0
267:         do I = 1, NCOLPG
268:           IP    = IWK4(I)
269:           NI    = KMAC(IP,MMAC(IP,1),1)
270:           S1    = SMIC(IP,NI,1)
271:           S2    = SMICPN(IP,IWK5(I),1)
272:           S3    = S1 + S2
273:           C3    = PPNBR(IWK5(I))
274:           if ( C3.lt.SMALL ) then          ! select with truth reaction p
robability
275:             if ( R(I).le.S2/S3 ) then          ! selected
276:               K      = K + 1
277:               IWK1(K) = IWK4(I)
278:               IWK2(K) = IWK5(I)
279:               DWK1(IWK4(I)) = 1
280:             else                              ! no selected
281:               NCOLPP = NCOLPP + 1
282:               IWK3(NCOLPP) = IWK4(I)
283:             end if
284:           else if ( R(I).le.C3 ) then          ! selected with PPNBR
285:             K      = K + 1
286:             IWK1(K) = IWK4(I)
287:             IWK2(K) = IWK5(I)
288:             DWK1(IP) = (S2 / S3) / C3          ! adjustment weight wi
th PPNBR
289:           else                              ! no selected with PPNBR
290:             NCOLPP = NCOLPP + 1
291:             IWK3(NCOLPP) = IWK4(I)
292:             WWW(IP) = WWW(IP) * (S1 / S3) / (1 - C3) ! weight adjusted with
PPNBR for photon interaction
293:           end if
294:         end do
295:       if ( K.le.0 ) then          ! photo-nuclear nuclide isn't selected
296:         if ( NPFCN.gt.0 ) then
297:           do I = 1, NCOLP
298:             LSCLP(I) = IWK3(I)
299:           end do
300:         end if
301:         go to 900
302:       end if
303:       NCOLPN = K
304:       else          ! select with truth reaction probabili
ty
305:         call RANU2( IRAND, R, NCOLPG, ICON )
306:         K      = 0
307:         do I = 1, NCOLPG
308:           IP    = IWK4(I)
309:           NI    = KMAC(IP,MMAC(IP,1),1)
310:           S1    = SMIC(IP,NI,1)
311:           S2    = SMICPN(IP,IWK5(I),1)
312:           WK4(I) = S2 / (S1 + S2)
313:         end do
314:         do I = 1, NCOLPG
315:           if ( R(I).le.WK4(I) ) then          ! selected
316:             K      = K + 1
317:             IWK1(K) = IWK4(I)
318:             IWK2(K) = IWK5(I)
319:             DWK1(IWK4(I)) = 1
320:           else                              ! no selected
321:             NCOLPP = NCOLPP + 1
322:             IWK3(NCOLPP) = IWK4(I)
323:           end if
324:         end do
325:         if ( K.le.0 ) then          ! photo-nuclear nuclide isn't selected
326:           if ( NPFCN.gt.0 ) then
327:             do I = 1, NCOLP
328:               LSCLP(I) = IWK3(I)
329:             end do
330:           end if
331:           go to 900
332:         end if
333:         NCOLPN = K
334:       endif
335: C
336: C ..... sort bank pointer (IWK1) in the order of collided nuclide
337: C ..... number (IWK2), and gather pointers of collided nuclides
338: call MSORTI( NCOLPN, IWK2, IWK1, IWK4, IWK5 )
339: NNNUC      = 1          ! number of nuclide species collided with phot
on
340: KWNUC(1,1) = IWK2(1)          ! collided nuclide number
341: KWNUC(1,2) = 1          ! start position of the collided nuclide
342: KWNUC(1,3) = NCOLPN          ! number of photons collided with the nuclide
343: if ( NCOLPN.gt.1 ) then
344:   *VOCL LOOP, SCALAR
345:   do I = 2, NCOLPN
346:     if ( IWK2(I).ne.IWK2(I-1) ) then
347:       NNNUC      = NNNUC + 1
348:       KWNUC(NNNUC,1) = IWK2(I)
349:       KWNUC(NNNUC,2) = I
350:       KWNUC(NNNUC-1,3) = I - KWNUC(NNNUC-1,2)
351:     end if
352:   end do
353:   if ( NNNUC.gt.1 ) KWNUC(NNNUC,3) = NCOLPN + 1 - KWNUC(NNNUC,2)
354: end if
355: C ..... initialize
356: NBANK4 = 4 * NBANK
357: NBANK5 = 5 * NBANK
358: IFFLII = NFFL(NZONE+1)          ! initial total number of particles stacked in
bank
359: AMP      = 0          ! mass of incident photon
360: C
361: C-----
362: C**** loop over collided nuclides
363: C-----
364: C
365: do 5000 NC = 1, NNNUC
366:   KN      = KWNUC(NC,1)
367:   LEMESH  = KLBPNI(KN,1)
368:   NPTS    = KLBPNI(KN,2)
369:   NBINA   = KLBPNI(KN,5)
370:   LNUD    = KLBPNI(KN,12)
371:   LSTPEA  = KLBPNI(KN,15)
372:   if ( LSTPEA.le.0 ) then
373:     write(IOW,911) KN, NCIDPN(KN), XLBPNI(KN,5), LSTPEA

```

src/mvp/photnuc.f

```

374:      stop 666
375:      end if
376:      ATW      = XLBPN1(KN,1)
377:      NK2G98   = KLBPN2(KN,98,7)      ! number of su
bsections for delayed fission neutron
378:      NBINA1   = NBINA + 1
379:      IN       = KWNNUC(NC,3)
380:      I1       = KWNNUC(NC,2)
381:      I2       = I1 + IN - 1
382:      I3       = 0
383:      do I = I1, I2
384:          I3   = I3 + 1
385:          IWK1(I3) = EEE(IWK1(I))
386:      end do
387:      call BSVDEC( CXP(N,LEMESH), NPTS, WK1, IWK4, IN )      ! energy mesh
point number, IWK4
388:      C
389:      if ( KLBPN1(KN,4).eq.1 ) then      ! single production reaction,
MT --> IWK5
390:          do I = I1, I2
391:              IWK5(I) = KLBPN2(KN,1,9)
392:          end do
393:          go to 145
394:      end if
395:      C
396:      C ..... sampling of the reaction type (MT)
397:      call RANU2( IRAND, R, IN, ICON )
398:      I3 = 0
399:      do I = I1, I2
400:          I3 = I3 + 1
401:          IP = IWK1(I)
402:          X2 = SMICPN(IP,KN,1) * R(I3)      ! nonelastic * random
number
403:          X1 = 0
404:          L2 = LEMESH + IWK4(I3)
405:          DE = (WK1(I3)-CXP(N,L2)) / (CXP(N,L2-1)-CXP(N,L2))
406:          DE1 = 1 - DE
407:      C ..... pre-process for sampling of forced reaction type by PMT
408:      NPMT0 = 0
409:      PMT1 = 0
410:      PMT2 = 1
411:      if ( NPMT.gt.0 ) then
412:          if ( JTLLT.eq.0 ) then
413:              IRG0 = KZREG(IZZ(IP))
414:          else
415:              IRG0 = IBREG(IP)
416:          end if
417:          MT0 = nint(PMT(1,KN,IRG0))
418:          if ( MT0.ne.0 ) then
419:              if ( KLBPN2(KN,MT0,1).gt.0 ) then
420:                  if ( KLBPN2(KN,MT0,3).le.IWK4(I3).and.
421:                      KLBPN2(KN,MT0,4).gt.IWK4(I3) ) then
422:                      &
423:                      PMT0 = PMT(2,KN,IRG0)
424:                      if ( PMT0.le.ZERO.or.PMT0.ge.1.0D+0 ) go to 125
425:                      L1 = KLBPN2(KN,MT0,10)-KLBPN2(KN,MT0,3)+IWK4(I3)
426:                      X3 = CXP(N,L1)*DE + CXP(N,L1+1)*DE1
427:                      if ( X3.le.ZERO ) go to 125
428:                      R0 = abs(X3/SMICPN(IP,KN,1)-1)
429:                      if ( R0.le.1.0D-8 ) then      ! case of single r
eaction
430:                          MT = MT0
431:                          go to 140
432:                      else
433:                          R0 = X3 / SMICPN(IP,KN,1)
434:                          if ( R0.ge.PMT0 ) go to 125
435:
436:                          PMT1 = PMT0 / R0
437:                          PMT2 = (1-PMT0) / (1-R0)
438:                          NPMT0 = MT0
439:                      end if
440:                  end if
441:              end if
442:          continue
443:          MT = 4
444:          130 MT = MT + 1
445:          if ( MT.gt.111 ) then
446:              R0 = abs(X1/X2-1)
447:              if ( R0.le.1.0D-6 ) then
448:                  X2 = X1 * 0.9999999D+0
449:                  X1 = 0
450:                  MT = 4
451:                  go to 130
452:              end if
453:              write(IOW,912) KN, NCIDPN(KN), XLBPN1(KN,5), WK1(I3),
454:                  & 111, I, IP, X2, X1, R(I3), SMICPN(IP,KN,1)
455:              stop 666
456:          end if
457:          if ( MT.ge.98 .and. MT.le.100 ) go to 130
458:          if ( KLBPN2(KN,MT,1).le.0 ) go to 130
459:          if ( KLBPN2(KN,MT,3).le.IWK4(I3) .and.
460:              & KLBPN2(KN,MT,4).gt.IWK4(I3) ) then
461:              L1 = KLBPN2(KN,MT,10)-KLBPN2(KN,MT,3)+IWK4(I3)
462:              if ( NPMT0.eq.0 ) then      ! normal samp
ling
463:                  X1 = X1 + CXP(N,L1)*DE + CXP(N,L1+1)*(1-DE)
464:                  if ( X1.ge.X2 ) go to 140
465:                  else      ! forced samp
ling by PMT
466:                  if ( MT.eq.NPMT0 ) then      ! for
ced MT is sampled.
467:                      X1 = X1 + (CXP(N,L1)*DE+CXP(N,L1+1)*(1-DE))*PMT1
468:                      if ( X1.ge.X2 ) then
469:                          DWK1(IP) = DWK1(IP)/PMT1
470:                          go to 140
471:                      end if
472:                      else      ! for
ced MT isn't sampled.
473:                          X1 = X1 + (CXP(N,L1)*DE+CXP(N,L1+1)*(1-DE))*PMT2
474:                          if ( X1.ge.X2 ) then
475:                              DWK1(IP) = DWK1(IP)/PMT2
476:                              go to 140
477:                          end if
478:                      end if
479:                  end if
480:                  end if
481:                  go to 130
482:                  140 IWK5(I) = MT
483:              end do
484:              145 continue
485:      C
486:      C ..... sort bank pointer in the order of reaction types
487:      call MSORTI( IN, IWK5(I1), IWK1(I1), IWK4, IWK6 )
488:      N2NUC = 1
489:      KWNNUC(NNNNUC+N2NUC,1) = IWK5(I1)
490:      KWNNUC(NNNNUC+N2NUC,2) = I1
491:      KWNNUC(NNNNUC+N2NUC,3) = IN
492:      if ( IN.gt.1 ) then
493:          *VOCL LOOP,SCALAR
494:          do I = I1+1, I2

```

src/mvp/photnuc.f

```

495:         if ( IWK5(I).ne.IWK5(I-1) ) then
496:             N2NUC          = N2NUC + 1
497:             KWNNUC(NNNNUC+N2NUC,1) = IWK5(I)
498:             KWNNUC(NNNNUC+N2NUC,2) = I
499:             KWNNUC(NNNNUC+N2NUC-1,3) = I - KWNNUC(NNNNUC+N2NUC-1,2)
500:         end if
501:     end do
502:     if ( N2NUC.gt.1 )
503: &         KWNNUC(NNNNUC+N2NUC,3) = I2 + 1 - KWNNUC(NNNNUC+N2NUC,2)
504:     end if
505: C
506:     do I = I1, I2
507:         WK1(I) = EEE(IWK1(I))          ! energy
508:     end do
509:     if ( JTLLT.eq.0 ) then
510:         do I = I1, I2
511:             IWK7(I) = KZREG(I2,IWK1(I)) ! region number
512:         end do
513:     else
514:         do I = I1, I2
515:             IWK7(I) = IBREG(IWK1(I))    ! region number
516:         end do
517:     end if
518: C
519: C-----
520: C**** loop over reaction types
521: C-----
522: C
523:     do 4000 MM = 1, N2NUC
524:         MT = KWNNUC(NNNNUC+MM,1)
525:         IM = KWNNUC(NNNNUC+MM,3)
526:         I3 = KWNNUC(NNNNUC+MM,2)
527:         I4 = I3 + IM - 1
528:         LCTMT = KLBPN2(KN,MT,5)
529:         if ( LCTMT.eq.2 .and. KLBPN2(KN,MT,7).eq.0 ) then
530: C ..... angular distribution is CM system (MF=5 and LCT=2)
531:             call PN4LCT2( IOW, IRAND, NZONE, NREG, NBANK, NEVENT,
532: &             NEST, NUCPN, NUCPNI, MCXPN, NMTPN, KN, MT, IM,
533: &             I3, I4, NBINA, ATW, MNC2, EBRELA, PI2, NUC,
534: &             NPATOM, NMT, NMTP, IWK7, WGTPTN, MXPGEN, EBOTX,
535: &             MMAC, CXPN, ICXPN, KLBPN2, XLBPN2, LSFFL, NFFL,
536: &             IZFFL, LSDED, NDEAD, XXX, YYY, ZZZ, AAA, BBB,
537: &             CCC, WWW, EEE, TTT, ITT, XIM, KLSF, KKP, LZZ,
538: &             IGG, LEVL, LZZ, LPOS, LCRS, IBREG, IBSPQ, DBNK,
539: &             IBNK, KDBNK, MDBNK, KIBNK, MIBNK, KPNPRD, ENGYB,
540: &             NCNTR, WCNTR, R, DWK1, WK1, WK2, WK3, WK4, IWK11,
541: c##<2007/03/14:PN4:
542: c## &             IWK12, IWK1, IWK4, IWK5, IWK8, IWK9, IWK10 )
543: &             IWK12, IWK1, IWK4, IWK5, IWK8, IWK9, IWK10,
544: &             KPNFG )
545: c##>
546:             go to 4000
547:         else if ( MT.ge.98 ) then          ! secondary particle data isn't
548:             go to 4000
549:         end if
550: C
551:         LST2G = ICXPN(LSTPEA+MT-1)
552:         NE2G = ICXPN(LST2G)
553:         NK2GMT = ICXPN(LST2G+1)
554:         IEPRO2G = LST2G + 2 + 2*N30 + 5*NK2GMT
555:         Y5 = 1
556:         if ( KLBPN2(KN,MT,7).gt.0 ) then    ! preprocess for MF=5
557:             KPLIM0 = 1
558:             if ( NK2GMT.eq.1 ) then
559:                 do I = I3, I4
560:                     IWK6(I) = 1              ! single subsection
561:                 end do
562:             else
563:                 call BSVDEC( CXPN(IEPRO2G), NE2G, WK1(I3), IWK4(I3),
564: &                     IM )
565:                 call RANU2( IRAND, R, IM*3, ICON )
566:                 IM2 = 2 * IM
567:                 I5 = 0
568:                 do I = I3, I4
569:                     if ( IWK4(I).le.0 ) IWK4(I) = 1
570:                 end do
571:                 *VOCL LOOP,NOVREC
572:                 do I = I3, I4
573:                     I5 = I5 + 1
574:                     L2 = IEPRO2G + IWK4(I)
575:                     L1 = L2 - 1
576:                     X1 = (CXPN(L1)-WK1(I)) / (CXPN(L1)-CXPN(L2))
577:                     X1 = min(1.0D0, max(0.0D0, X1))
578:                     C/#IF ROUNDOFF(NEAREST)
579:                     IT = min(1, int(R(I5)+X1)) + IWK4(I)
580:                     C/#ELSE
581:                     R1 = R(I5) + X1
582:                     IT = IWK4(I) + R1
583:                     C/#ENDIF
584:                     L1 = IEPRO2G + 2*NE2G + 3*(IT-1)*NK2GMT - 1
585:                     C/#IF ROUNDOFF(NEAREST)
586:                     L2 = min(int(NK2GMT*R(IM+I5))+1, NK2GMT)
587:                     C/#ELSE
588:                     L2 = NK2GMT * R(IM+I5) + 1
589:                     C/#ENDIF
590:                     L3 = L1 + L2
591:                     C/#IF ROUNDOFF(NEAREST)
592:                     L4 = 2 * L2
593:                     L4 = min(L4, int(L4+R(IM2+I5)-CXPN(L3))) +
594:                         L1 + NK2GMT
595:                     C/#ELSE
596:                     R2 = R(IM2+I5) - CXPN(L3)
597:                     L4 = L1 + NK2GMT + 2*L2 + R2
598:                     C/#ENDIF
599:                     IWK6(I) = ICXPN(L4)          ! selected subsection number
600:                 end do
601:             end if
602:             L1 = IEPRO2G + (2+3*NK2GMT)*NE2G
603:             Y5 = CXPN(L1)          ! yield
604:         else
605:             KPLIM0 = N30
606:         end if
607: C
608: C-----
609: C**** loop over possible production particles
610: C-----
611: C
612:         do 3000 IV = 1, KPLIM0
613:             if ( IV.ge.20 ) go to 3010      ! escape for residual and unde
614:             NK2GG = ICXPN(LST2G+IV+1)
615:             KSUBS = ICXPN(LST2G+IV+1+N30)
616:             if ( NK2GG.le.0 ) go to 3000      ! don't given
617:             if ( IV.eq.1 ) then              ! neutron
618:                 KP = KPNEUT
619:             else if ( IV.eq.7 ) then          ! photon
620:                 KP = KPPHOT
621:             else if ( IV.eq.8 ) then          ! electron
622:                 KP = KPELEC

```

src/mvp/photnuc.f

```

623:         else
624:             KP      = 2                ! other particles are
dummy
625:         end if
626:         AWP      = PMASS(KP)
627:         if ( KLBPN2(KN,MT,7).gt.0 ) then      ! 2nd preprocess for M
F=5
628:             NK2GG  = 1
629:             KSUBS  = 1
630:         end if
631: C
632: C-----
633: C**** loop over subsections (NK)
634: C-----
635: C
636:         do 2000 IK = KSUBS, KSUBS+NK2GG-1
637:             call RANU2( IRAND, R(I3), IM, ICON )
638:             JJDLYN = JJDLYN
639:             if ( JJDLYN.ne.0 ) then
640:                 if ( LNUD.eq.0.or.NK2GG98.le.0 ) JJDLYN = 0
641:                 do I = I3, I4
642:                     WK5(I) = 0
643:                     IWK15(I) = 0
644:                 end do
645:             end if
646: C ..... MF=5 form ... (NK2G(MT)>0) : energy distributions
647:             if ( KLBPN2(KN,MT,7).gt.0 ) then
648: C ..... photo-fission (MT=18) ..... neutron generation
649:                 if ( MT.eq.18 ) then
650:                     if ( JJDLYN.ne.0 )
651:                         & call RANU2( IRAND, R(I3+IM), IM, ICON )
652:                 *VOCL LOOP,NOVREC
653:                 do I = I3, I4
654:                     IP      = IWK1(I)
655:                     NCNTR(13,2) = NCNTR(13,2) + 1      ! event mo
nitor: photo-fission
656:                     WCNTR(13,2) = WCNTR(13,2) + WWW(IP)*DWK1(IP)
657: C ..... total or prompt neutron yield from fission (MT=18)
658:                     if ( WGTPN(MT,KN,IWK7(I)).le.0.0 ) then ! no contr
ol over generation weight
659:                         WK2(I) = WWW(IP) * DWK1(IP)      ! ad
justed weight
660:                     if ( RNU(IP,KN,2)+RNU(IP,KN,3).le.0.0.or.
661:                         & JJDLYN.eq.0 ) then
662:                         IWK4(I) = RNU(IP,KN,1) + R(I)      ! ne
t integer yield of nu_total
663:                     else
664:                         IWK4(I) = RNU(IP,KN,3) + R(I)      ! ne
t integer yield of nu_prompt
665:                     end if
666:                     else ! control
over generation weight
667:                         if ( RNU(IP,KN,2)+RNU(IP,KN,3).le.0.0.or.
668:                             & JJDLYN.eq.0 ) then
669:                             WP      = RNU(IP,KN,1)*WWW(IP)*DWK1(IP) ! nu
_total * adjusted weight
670:                         else
671:                             WP      = RNU(IP,KN,3)*WWW(IP)*DWK1(IP) ! nu
_prompt * adjusted weight
672:                         end if
673:                         NIY      = WP/WGTPN(MT,KN,IWK7(I))+R(I)      ! ne
t integer yield
674:                         if ( NIY.gt.100 ) then      ! to escap
e bank-overflow
675:                             R81      = 100.d0 / NIY
676:                             WK2(I) = WGTPN(MT,KN,IWK7(I)) / R81      ! ph
oto-nuclear generation weight
677:                             IWK4(I) = WP / WK2(I) + R(I)      ! in
teger yield is normalized to 100
678:                         else
679:                             IWK4(I) = NIY      ! ne
t integer yield
680:                             WK2(I) = WGTPN(MT,KN,IWK7(I))      ! ph
oto-nuclear generation weight
681:                         end if
682:                         end if
683:                         NCNTR(14,2) = NCNTR(14,2) + IWK4(I)      ! ev
ent monitor: total or prompt neutron generation
684:                         WCNTR(14,2) = WCNTR(14,2) + IWK4(I)*WK2(I)
685: C ..... delayed neutron yield from fission (MT=18,98(455))
686:                         if ( JJDLYN.ne.0 ) then
687:                             if ( RNU(IP,KN,2).le.0.0 ) then      ! no nu_de
layed
688:                                 IWK15(I)= 0
689:                             else if ( WGTPN(98,KN,IWK7(I)).le.0.0 ! no contr
ol over generation weight
690:                                 & ) then
691:                                     IWK15(I)= RNU(IP,KN,2) + R(I+IM)      ! ne
t integer yield of nu_delayed
692:                                     WK5(I) = WWW(IP) * DWK1(IP)      ! ge
neration weight of delayed
693:                                 else ! control
over generation weight
694:                                     WP2      = RNU(IP,KN,2)*WWW(IP)*DWK1(IP) ! nu
_delayed * adjusted weight
695:                                     NIY2      = WP2 / WGTPN(98,KN,IWK7(I)) +      ! ne
t integer yield
696:                                     & R(I+IM)
697:                                     if ( NIY2.gt.100 ) then
698:                                         R81      = 100.d0 / NIY2
699:                                         WK5(I) = WGTPN(98,KN,IWK7(I))/R81      ! ge
neration weight of delayed
700:                                         IWK15(I) = WP2 / WK5(I) + R(I+IM)      ! de
layed yield is normlized to 100
701:                                     else
702:                                         IWK15(I) = NIY2      ! de
layed yield
703:                                         WK5(I) = WGTPN(98,KN,IWK7(I))      ! ge
neration weight of delayed
704:                                     end if
705:                                     end if
706:                                     NCNTR(30,2) = NCNTR(30,2)+IWK15(I)      ! ev
ent monitor: delayed neutron generation
707:                                     WCNTR(30,2) = WCNTR(30,2)+IWK15(I)*WK5(I)
708:                                     end if
709:                                     end do
710: C ..... reaction except photo-fission
711:                         else
712:                             *VOCL LOOP,NOVREC
713:                             do I = I3, I4
714:                                 IP      = IWK1(I)
715:                                 if ( WGTPN(MT,KN,IWK7(I)).le.0.0 ) then ! no contr
ol over generation weight
716:                                     WK2(I) = WWW(IP) * DWK1(IP)      ! ad
justed weight
717:                                     IWK4(I) = Y5 + R(I)      ! ne
t integer yield
718:                                     else ! control
over generation weight
719:                                         WP      = Y5 * WWW(IP) * DWK1(IP)      ! yi

```

src/mvp/photnuc.f

```

eld * adjusted weight
720:      NIY      = WP / WGTPN(MT,KN,IWK7(I)) + R(I)  ! ne
t integer yield
721:      if ( NIY.gt.100 ) then                        ! to escap
e bank-overflow
722:      R81      = 100.d0 / NIY
723:      WK2(I)   = WGTPN(MT,KN,IWK7(I)) / R81      ! ph on
oto-nuclear generation weight
724:      IWK4(I) = WP / WK2(I) + R(I)                ! in inate
teger yield is normalized to 100
725:      else
726:      IWK4(I) = NIY                                ! ne
t integer yield
727:      WK2(I)   = WGTPN(MT,KN,IWK7(I))            ! ph
oto-nuclear generation weight
728:      end if
729:      end if
730:      end do
731:      end if
732: C ..... MF=6 form ... (NK2G(MT)<0) : energy-angle distributions
733:      else
734:      IYP2NK = IEPRO2G + 3*NE2G + (IK-1)*NE2G
735:      call BSVDEC( CXPNI(IEPRO2G), NE2G, WK1(I3),
736:      &          IWK5(I3), IM )
737:      do I = I3, I4
738:      II      = IWK5(I)
739:      E1      = CXPNI(IEPRO2G+II-1)
740:      E2      = CXPNI(IEPRO2G+II)
741:      Y1      = CXPNI(IYP2NK+II-1)
742:      Y2      = CXPNI(IYP2NK+II)
743:      WK3(I)  = Y2 + (WK1(I)-E2)/(E1-E2) * (Y1-Y2)  ! yiel
d
744:      end do
745:      do I = I3, I4
746:      IP      = IWK1(I)
747:      if ( WGTPN(MT,KN,IWK7(I)).le.0.0 ) then      ! no contr
ol over generation weight
748:      WK2(I)  = WWW(IP) * DWK1(IP)                ! ad
justed weight
749:      IWK4(I) = WK3(I) + R(I)                      ! ne
t integer yield
750:      else
over generation weight
751:      WP      = WK3(I) * WWW(IP) * DWK1(IP)        ! yi
eld * adjusted weight
752:      NIY      = WP / WGTPN(MT,KN,IWK7(I)) + R(I)  ! ne
t integer yield
753:      if ( NIY.gt.100 ) then                        ! to escap
e bank-overflow
754:      R81      = 100.d0 / NIY
755:      WK2(I)   = WGTPN(MT,KN,IWK7(I)) / R81      ! ph
oto-nuclear generation weight
756:      IWK4(I) = WP / WK2(I) + R(I)                ! in
teger yield is normalized to 100
757:      else
758:      IWK4(I) = NIY                                ! ne
t integer yield
759:      WK2(I)   = WGTPN(MT,KN,IWK7(I))            ! ph
oto-nuclear generation weight
760:      end if
761:      end if
762:      if ( MT.eq.18.and.IV.eq.1 ) then
763:      NCNTR(14,2) = NCNTR(14,2) + IWK4(I)          ! ev
ent monitor: prompt fission neutron generation
764:      WCNTR(14,2) = WCNTR(14,2) + IWK4(I)*WK2(I)

765:      end if
766:      end do
767:      end if
768: C ..... treatment of products by reaction
769:      if ( IV.eq.1.or.IV.eq.7 ) go to 150          ! 1:neutron and 7:phot
770:      go to 2000                                    ! other particles term
771: C ..... particle generation cutoff
772:      150
773:      continue
774:      if ( MXPGEN.gt.0 ) then
775:      III      = 0
776:      WII      = 0
777:      do I = I3, I4
778:      IP      = IWK1(I)
779:      I5      = IWK4(I)
780:      if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.I5.gt.0 )
781:      &          then
782:      III      = III + I5
783:      WII      = WII + I5 * WK2(I)
784:      IWK4(I) = 0
785:      end if
786:      if ( JJDLYN.ne.0 ) then
787:      &          if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.
788:      &              IWK15(I).gt.0 ) then
789:      III      = III + IWK15(I)
790:      WII      = WII + IWK15(I) * WK5(I)
791:      IWK15(I) = 0
792:      end if
793:      end if
794:      end do
795:      if ( III.gt.0 ) then
796:      NCNTR(27,KP) = NCNTR(27,KP) + III
797:      WCNTR(27,KP) = WCNTR(27,KP) + WII
798:      end if
799:      end if
800: C ..... check the total number of babies
801:      III      = 0
802:      WII      = 0
803:      MA      = 0
804:      if ( JJDLYN.eq.0 ) then
805:      do I = I3, I4
806:      III      = III + IWK4(I)                      ! total number
807:      WII      = WII + IWK4(I) * WK2(I)              ! total weight
808:      MA      = max(MA, IWK4(I))                    ! maximum number
809:      end do
810:      end if
811:      else
812:      do I = I3, I4
813:      III      = III + IWK4(I) + IWK15(I)            ! total number
814:      WII      = WII+IWK4(I)*WK2(I)+IWK15(I)*WK5(I) ! total
815:      MA      = max(MA, IWK4(I), IWK15(I))          ! maximum number
816:      end do
817:      end if
818: C ..... too many babies. sorry, no possible method of birth control
819: C ..... currently. you are forced to play a sudden death game.
820:      if ( III.gt.NDEAD ) then
821:      write(IOW,913) KP, MT, IK, III, NDEAD, IM, KN, MA,
822:      &          (WWW(IWK1(I)),I=I3,I4)
823: C /#IF UNIX
824:      call FLUSHSTD()

```

src/mvp/photnuc.f

```

825: C/#ENDIF
826:           stop 666
827:       end if
828:       if ( III.le.0 ) go to 2000           ! no baby
829: C
830: C ..... monitoring process of generation particle weight
831:       if ( NMOPNW.gt.0 ) then
832:           do I = I3, I4
833:               if ( JJDLYN.eq.0 ) then
834:                   if ( IWK4(I).le.0 ) go to 160
835:               else
836:                   if ( IWK4(I)+IWK15(I).le.0 ) go to 160
837:               end if
838:               do J = 1, NMOPNW
839:                   if ( MONPNWGT(1,J).eq.IWK7(I).and.           ! region
840:                       MONPNWGT(2,J).eq.KN.and.                 ! nuclide
841:                       MONPNWGT(3,J).eq.MT ) then                ! reaction
842:                       M = MONPNWGT(4,J)
843:                       if ( JJDLYN.eq.0 ) then
844:                           WT = IWK4(I)*WK2(I)
845:                           WT2 = IWK4(I)*(WK2(I)**2)
846:                           NMPNWT(M)=NMPNWT(M)+IWK4(I)
847:                       else
848:                           WT = IWK4(I)*WK2(I)+IWK15(I)*WK5(I)
849:                           WT2 = IWK4(I)*(WK2(I)**2)+
850:                               IWK15(I)*(WK5(I)**2)
851:                           NMPNWT(M)=NMPNWT(M)+IWK4(I)+IWK15(I)
852:                       end if
853:                       WMPNWT(1,M)=WMPNWT(1,M)+WT
854:                       WMPNWT(2,M)=WMPNWT(2,M)+WT2
855:                   end if
856:               end do
857:           continue
858:       end do
859:       end if
860: C
861: C ..... keep space in bank for babies
862:       IFFLI = NFFL(NZONE+1)
863:       KIII = 0
864:       N3 = 3 * NBANK
865:       do I = I3, I4
866:           R(I+N3) = WK1(I)           ! tentative st
867: orage of incident energy
868:       end do
869:       do M = 1, MA
870:           do I = I3, I4
871:               if ( IWK4(I).ge.M ) then
872:                   KIII = KIII + 1
873:                   LSFFL( IFFLI+KIII ) = LSDED(NDEAD)
874:                   NDEAD = NDEAD - 1
875:                   WK1(NCOLPN+KIII) = R(I+N3)           ! copy inciden
876: t energy
877:                   IWK5(KIII) = IWK1(I)                 ! copy bank po
878: inter
879:                   IWK11(KIII) = IWK6(I)                 ! copy selecte
880: d subsection for MF=5
881:                   WK3(KIII) = WK2(I)                   ! copy weight
882:                   WK2(KIII) = IWK7(I)                   ! temporary
883:                   NCNTR(29,KP)= NCNTR(29,KP) + 1
884:                   WCNTR(29,KP)= WCNTR(29,KP) + WK2(I)
885:               end if
886:           end do
887:       end do
888:       if ( (KPNPRD.eq.0.or.KPNPRD.eq.1).and.KIBNK(17).ne.0 )
889:           &
890:           do I = 1, KIII
891:               IBNK( LSFFL( IFFLI+I ), KIBNK(17) ) = IWK2(I)           ! p
892:           end do
893:       end if
894:       C ..... process of delayed babies
895:       C ..... keep space in bank for them
896:       if ( JJDLYN.ne.0.and.KIII.lt.III ) then
897:           KDDD = KIII
898:           do M = 1, MA
899:               do I = I3, I4
900:                   if ( IWK15(I).ge.M ) then
901:                       KDDD = KDDD + 1
902:                       LSFFL( IFFLI+KDDD ) = LSDED(NDEAD)
903:                       NDEAD = NDEAD - 1
904:                       WK1(NCOLPN+KDDD) = R(I+N3)           ! copy inciden
905: t energy
906:                       IWK5(KDDD) = IWK1(I)                 ! copy bank po
907: inter
908:                       WK3(KDDD) = WK5(I)                   ! copy weight
909:                       WK2(KDDD) = IWK7(I)                   ! temporary
910:                       NCNTR(29,KP)= NCNTR(29,KP) + 1
911:                       WCNTR(29,KP)= WCNTR(29,KP) + WK5(I)
912:                   end if
913:               end do
914:           end do
915:       end if
916:       if ( (KPNPRD.eq.0.or.KPNPRD.eq.1).and.
917:           KIBNK(17).ne.0 ) then
918:           do I = KIII+1, KDDD
919:               IBNK( LSFFL( IFFLI+I ), KIBNK(17) ) = IWK2(I)           ! p
920:           end do
921:       end if
922:       C ..... sampling of energy and direction for them
923:       KDDD = KDDD - KIII
924:       if ( KLBPN2(KN,98,7).le.0 ) then
925:           write( IOW,918 ) KN,KLBPN2(KN,98,7),KDDD
926:           stop 666
927:       end if
928:       L1 = NCOLPN + KIII
929:       call DELAYPN( IOW, JTIME, NBANK, IRAND, KDDD, KN,
930:                   WK1(L1+1), WK2(L1+1), WK5(L1+1),
931:                   MCXPN, CXPN, ICXPN, KLBPN1, KLBPN2, NUCPNI, NMTPN,
932:                   R, IWK2, IWK8, IWK9, IWK10 )
933:       do I = 1, KDDD
934:           IP = LSFFL( IFFLI+KIII+I )
935:           EEE(IP) = WK1(L1+I)
936:           DWK2(KIII+I) = WK2(L1+I)
937:       end do
938:       if ( JTIME.ne.0 ) then
939:           do I = 1, KDDD
940:               IP = LSFFL( IFFLI+KIII+I )
941:               TT0 = TTT(IP) + WK5(L1+I)           ! ad
942:           end do
943:           if ( JPTIM.ne.0.and.TT0.gt.TCUT ) then           ! tr
944:               ITREP = TT0 / TCUT
945:               TT0 = TT0 - ITREP * TCUT
946:           end if
947:           TTT(IP) = TT0
948:           R(I) = TT0
949:       end do
950:       call BSVINC( TIMEB, NTIME+1, R(1), IWK8, KDDD )
951:       do I = 1, KDDD

```


src/mvp/photnuc.f

```

944:          IP      = LSFFL(IFFLI+KIII+I)
945:          ITT(IP) = max(1, min(NTIME, IWK8(I)))
946:          end do
947:        end if
948:      else
949:        KDDD      = 0          ! no delayed baby
950:      end if
951:      if ( JJDLYN.ne.0.and.KIII.le.0 ) go to 250    ! no prompt
baby
952: C
953:       N2 = KIII * 2
954:       N3 = KIII * 3
955:       N4 = KIII * 4
956:       N5 = KIII * 5
957:       N6 = KIII * 6
958:       N7 = KIII * 7
959: C
960: C-----
961: C**** sampling of energy and direction (cosine) in CM system
962: C-----
963: C
964:       LST2GS = ICXPN(LST2G+1+2*N30+IK)
965:       LF      = ICXPN(LST2GS)
966:       KCM2LAB = 0
967: C
968: C ..... MF=6 ... energy-angle distribution
969: C ..... lf=61 : continuum energy-angle distributions (law1)
970:       if ( LF.eq.61 ) then
971:         NEF5S = ICXPN(LST2GS+1)
972:         LEENG = LST2GS + 2
973:         call BSVDEC( CXPN(LEENG), NEF5S, WK1(NCOLPN+1),
974:           &          IWK2, KIII )
975:         call RANU2( IRAND, R, KIII*6, ICON )
976:         LLSTF5E = LEENG + 2 * NEF5S
977:         do J = 1, KIII
978:           if ( IWK2(J).le.0 ) IWK2(J) = 1
979:         end do
980:         do J = 1, KIII
981:           IP      = LSFFL(IFFLI+J)
982:           L2      = LEENG + IWK2(J)
983:           L1      = L2 - 1
984:           INTE0   = ICXPN(L2+NEF5S)
985:           INTE1   = INTE0 / 10
986:           INTE    = INTE0 - INTE1 * 10
987:           if ( INTE.eq.5.or.INTE.eq.3 ) then ! log interpol
988:             X1    = log(CXPN(L1)/WK1(NCOLPN+J)) /
989:               &    log(CXPN(L1)/CXPN(L2))
990:           else ! linear interpol
991:             X1    = (CXPN(L1)-WK1(NCOLPN+J)) /
992:               &    (CXPN(L1)-CXPN(L2))
993:           end if
994:           X1      = min(1.0D0, max(0.0D0, X1))
995: C ..... select distribution table after nep
996: C/#IF ROUNDOFF(NEAREST)
997:       IT      = min(1, INT(R(J)+X1)) + IWK2(J)
998: C/#ELSE
999:       R1      = R(J) + X1
1000:       IT      = IWK2(J) + R1
1001: C/#ENDIF
1002:       LSTF5E = ICXPN(LLSTF5E+IT-1)
1003:       NEP    = ICXPN(LSTF5E)
1004:       LQ1    = LSTF5E + 1          ! pre-position
of Q_value
1005:       ND      = ICXPN(LSTF5E+1)
1006: C ..... B.M.C. sampling of emitted energy bin
1007: C/#IF ROUNDOFF(NEAREST)
1008:       LL2     = min(int(NEP*R(KIII+J))+1, NEP)
1009: C/#ELSE
1010:       LL2     = NEP * R(KIII+J) + 1
1011: C/#ENDIF
1012:       LL3     = LQ1 + LL2
1013: C/#IF ROUNDOFF(NEAREST)
1014:       LL4     = LL2 * 2
1015:       LL4     = min(LL4, int(LL4+R(N3+J)-CXPN(LL3))) +
1016:         &      LQ1 + NEP
1017: C/#ELSE
1018:       R2      = R(N3+J) - CXPN(LL3)
1019:       LL4     = LQ1 + NEP + 2 * LL2 + R2
1020: C/#ENDIF
1021:       LL5     = ICXPN(LL4)
1022:       LLL     = LQ1 + 3 * NEP
1023:       L3      = LLL + LL5
1024:       L4      = L3 + NEP + 1
1025:       L5      = L4 + NEP + 1
1026:       L6      = L5 + NEP + 1
1027:       E0      = CXPN(L3)
1028:       E1      = CXPN(L3+1)
1029:       P0      = CXPN(L4)
1030:       P1      = CXPN(L4+1)
1031:       R0      = CXPN(L5)
1032:       R1      = CXPN(L5+1)
1033:       A0      = CXPN(L6)
1034:       A1      = CXPN(L6+1)
1035:       if ( LL5.le.ND ) then ! continuous spectra
1036:         RRR    = (P0 + P1) * R(N2+J)
1037:         RRR    = RRR / (sqrt(P1*P1+(P0-P1)*RRR) +
1038:           &      P1 + SMALL)
1039:         PRECF  = R1 + (R0 - R1) * RRR
1040:         ANGDS  = A1 + (A0 - A1) * RRR
1041:         EEE(IP) = E1 + (E0 - E1) * RRR ! secondary en
1042:       else ! discrete spectra
1043:         RRR    = 0
1044:         PRECF  = R1
1045:         ANGDS  = A1
1046:         EEE(IP) = abs(E1) ! secondary en
1047:       end if
1048:       if ( R(N4+J).gt.PRECF ) then
1049:         T      = (2 * R(N5+J) - 1) * sinh(ANGDS)
1050:         WK2(NCOLPN+J) = log(T+sqrt(T*T+1)) / ANGDS ! scat
1051:       else
1052:         WK2(NCOLPN+J) = log(R(N5+J)*exp(ANGDS)+
1053:           &      (1-R(N5+J))*exp(-ANGDS)) / ANGDS ! scat
1054:       end if
1055: C ..... INT > 10
1056:       if ( INTE1.gt.0.and.LL5.le.ND ) then
1057:         LLE    = LLL + 1
1058:         if ( IT.eq.IWK2(J) ) then
1059:           NEP1  = NEP
1060:           ND1   = ND
1061:           LLE1  = LLE
1062:           LSTF5T = ICXPN(LLSTF5E+IT)
1063:           NEP2  = ICXPN(LSTF5T)
1064:           ND2   = ICXPN(LSTF5T+1)
1065:           LLE2  = LSTF5T + 3 * NEP2 + 2

```

src/mvp/photnuc.f

```

1066:      else
1067:         NEP2 = NEP
1068:         ND2 = ND
1069:         LLE2 = LLE
1070:         LSTF5T = ICXPN(LLSTF5E+IT-2)
1071:         NEP1 = ICXPN(LSTF5T)
1072:         ND1 = ICXPN(LSTF5T+1)
1073:         LLE1 = LSTF5T + 3 * NEP1 + 2
1074:      end if
1075:      C ..... linear interpolation is assumed.
1076:      if ( INTE1.eq.2 ) then ! unit base in
      terpolation
1077:         EMAX = CXPN(LLE1) + X1 *
1078:         &      (CXPN(LLE2)-CXPN(LLE1))
1079:         EEE(IP) = EEE(IP) * EMAX / CXPN(LLE)
1080:      else if ( INTE1.eq.1 ) then ! method of co
      rresponding point
1081:         EMAX = CXPN(LLE1) + X1 *
1082:         &      (CXPN(LLE2)-CXPN(LLE1))
1083:         EMIN = CXPN(LLE1+ND1) + X1 *
1084:         &      (CXPN(LLE2+ND2)-CXPN(LLE1+ND1))
1085:         EEE(IP) = EMIN + (EMAX-EMIN) *
1086:         &      (EEE(IP)-CXPN(LLE+ND)) /
1087:         &      (CXPN(LLE)-CXPN(LLE+ND))
1088:      end if
1089:      end if
1090:      end do
1091:      C ..... lf=62 : discrete two-body scattering (law=2)
1092:      C ..... lf=62 : discrete two-body scattering (law=2)
1093:      else if ( LF.eq.62 ) then
1094:         NEF5S = ICXPN(LST2GS+1)
1095:         LEENG = LST2GS + 2
1096:         call BSVDEC( CXPN(LEENG), NEF5S, WK1(NCOLPN+1),
1097:         &      IWK2, KIII )
1098:         call RANU2( IRAND, R, KIII*3, ICON )
1099:         E2 = ATW * MNC2 ! energy of ta
      rget nuclide
1100:         LLSTF5E = LEENG + 2 * NEF5S
1101:         do J = 1, KIII
1102:            IP = LSFFL(IFFLI+J)
1103:            if ( IWK2(J).le.0 ) IWK2(J) = 1
1104:            L2 = LEENG + IWK2(J)
1105:            L1 = L2 - 1
1106:            INTE0 = ICXPN(L2+NEF5S)
1107:            INTE1 = INTE0 / 10
1108:            INTE = INTE0 - INTE1 * 10
1109:            if ( INTE.eq.5.or.INTE.eq.3 ) then ! log interpol
      ation
1110:               X1 = log(CXPN(L1)/WK1(NCOLPN+J)) /
1111:               &      log(CXPN(L1)/CXPN(L2))
1112:            else ! linear inter
      polation
1113:               X1 = (CXPN(L1)-WK1(NCOLPN+J)) /
1114:               &      (CXPN(L1)-CXPN(L2))
1115:            end if
1116:            X1 = min(1.0D0, max(0.0D0, X1))
1117:            C ..... select distribution table after nep
1118:            C/#IF ROUNDOff(NEAREST)
1119:            IT = min(1, int(R(J)+X1)) + IWK2(J)
1120:            C/#ELSE
1121:            R1 = R(J) + X1
1122:            IT = IWK2(J) + R1
1123:            C/#ENDIF
1124:            LSTF5E = ICXPN(LLSTF5E+IT-1)
1125:            C ..... sample scattering angle from equi-probable bin utab
1126:            C/#IF ROUNDOff(NEAREST)
1127:            IANG = min(NBINA, int(R(KIII+J)*NBINA+1))
1128:            C/#ELSE
1129:            IANG = int(R(KIII+J)*NBINA+1)
1130:            C/#ENDIF
1131:            LU1 = LSTF5E + IANG
1132:            XMU = CXPN(LU1-1) + R(N2+J) *
1133:            &      (CXPN(LU1)-CXPN(LU1-1)) ! scattering a
      ngle cosine
1134:            WK2(NCOLPN+J) = XMU
1135:            C ..... calculate outgoing energy in CM by kinematics
1136:            EIN = WK1(NCOLPN+J)
1137:            Q = - XLBPN2(KN,MT,1)
1138:            call PNKIN( 2, ATW, XMU, MNC2, EIN, Q, AMP,
1139:            &      AWP, EEE0, IOW )
1140:            EEE(IP) = EEE0
1141:            end do
1142:            C ..... lf=66 : n-body phase-space distributions (law=6)
1143:            C ..... lf=66 : n-body phase-space distributions (law=6)
1144:            else if ( LF.eq.66 ) then
1145:               call RANU2( IRAND, R, KIII*5, ICON )
1146:               APSX = CXPN(LST2GS+6) ! total mass in n.m.u
1147:               NPSX = ICXPN(LST2GS+8) ! number of particles
1148:               do J = 1, KIII
1149:                  EIMAX = ATW * WK1(NCOLPN+J) / (ATW + AMP) -
1150:                  &      XLBPN2(KN,MT,1) ! ener
1151:                  WK1(NCOLPN+J) = EIMAX * (APSX - AWP) / APSX ! maxi
1152:                  T1 = PIH * R(KIII+J)
1153:                  X1 = cos(T1) ** 2
1154:                  DWK1(J) = - X1 * log(R(J)) - log(R(N2+J))
1155:                  T2 = PIH * R(N4+J)
1156:                  X2 = cos(T2) ** 2
1157:                  DWK2(J) = - X2 * log(R(N3+J))
1158:               end do
1159:               if ( NPSX.eq.3 ) then
1160:                  call RANU2( IRAND, R, KIII*2, ICON )
1161:                  do J = 1, KIII
1162:                     IP = LSFFL(IFFLI+J)
1163:                     P = R(KIII+J)
1164:                     EEE(IP) = WK1(NCOLPN+J) * DWK1(J) /
1165:                     &      (DWK1(J) + DWK2(J) - log(P))
1166:                     WK2(NCOLPN+J) = 2 * R(J) - 1
1167:                  end do
1168:                  else if ( NPSX.eq.4 ) then
1169:                     call RANU2( IRAND, R, KIII*3, ICON )
1170:                     do J = 1, KIII
1171:                        IP = LSFFL(IFFLI+J)
1172:                        P = R(KIII+J) * R(N2+J)
1173:                        EEE(IP) = WK1(NCOLPN+J) * DWK1(J) /
1174:                        &      (DWK1(J) + DWK2(J) - log(P))
1175:                        WK2(NCOLPN+J) = 2 * R(J) - 1
1176:                     end do
1177:                     else
1178:                        call RANU2( IRAND, R, KIII*5, ICON )
1179:                        do J = 1, KIII
1180:                           IP = LSFFL(IFFLI+J)
1181:                           P = R(KIII+J) * R(N2+J) * R(N3+J) * R(N4+J)
1182:                           EEE(IP) = WK1(NCOLPN+J) * DWK1(J) /
1183:                           &      (DWK1(J) + DWK2(J) - log(P))
1184:                           WK2(NCOLPN+J) = 2 * R(J) - 1
1185:                        end do

```

src/mvp/photnuc.f

```

1186:                                end if
1187: C
1188: C ..... lf=67 : laboratory angle-energy law (law=7)
1189:                                else if ( LF.eq.67 ) then
1190:                                N67I = 0
1191:                                NEF5S = ICXPN(LST2GS+1)
1192:                                LEENG = LST2GS + 2
1193:                                LLSTF5E = LEENG + 2 * NEF5S
1194:                                call BSVDEC( CXPN(LEENG), NEF5S, WK1(NCOLPN+1),
1195:                                & IWK2, KIII)
1196:                                call RANU2( IRAND, R, KIII*3, ICON )
1197:                                do J = 1, KIII
1198:                                if ( IWK2(J).le.0 ) IWK2(J) = 1
1199:                                L2 = LEENG + IWK2(J)
1200:                                L1 = L2 - 1
1201:                                INTE0 = ICXPN(L2+NEF5S)
1202:                                INTE1 = INTE0 / 10
1203:                                INTE = INTE0 - INTE1 * 10
1204:                                if ( INTE.eq.5.or.INTE.eq.3 ) then ! log interpol
1205:                                X1 = log(CXPN(L1) / WK1(NCOLPN+J)) /
1206:                                & log(CXPN(L1) / CXPN(L2))
1207:                                else ! linear inter
1208:                                X1 = (CXPN(L1) - WK1(NCOLPN+J)) /
1209:                                & (CXPN(L1) - CXPN(L2))
1210:                                end if
1211:                                X1 = min(1.0D0, max(0.0D0, X1))
1212: C ..... select distribution table after utab
1213: C/#IF ROUNDOFF(NEAREST)
1214:                                IT = min(1, int(R(J)+X1)) + IWK2(J)
1215: C/#ELSE
1216:                                *
1217:                                *
1218: C/#ENDIF
1219:                                LSTF5E = ICXPN(LLSTF5E+IT-1)
1220:                                LL = LSTF5E + NBINA1
1221:                                IWK8(J) = LL + 2 ! position of xmu
1222:                                IWK9(J) = ICXPN(LL+1) ! nmu
1223: C ..... sample scattering angle from equi-probable bin utab
1224: C/#IF ROUNDOFF(NEAREST)
1225:                                IANG = min(NBINA, int(R(KIII+J)*NBINA+1))
1226: C/#ELSE
1227:                                *
1228: C/#ENDIF
1229:                                IANG = int(R(KIII+J)*NBINA+1)
1230:                                LU1 = LSTF5E + IANG
1231:                                WK2(NCOLPN+J) = CXPN(LU1-1) + R(N2+J) *
1232:                                & (CXPN(LU1) - CXPN(LU1-1)) ! scat
1233: C ..... store working data for INT>10
1234:                                IWK10(J) = INTE1
1235:                                if ( INTE1.gt.0 ) then
1236:                                N67I = N67I + 1
1237:                                IWK11(N67I) = J
1238:                                WK4(N67I) = WK2(NCOLPN+J)
1239:                                if ( IT.eq.IWK2(J) ) then
1240:                                LSTF5T = ICXPN(LLSTF5E+IT)
1241:                                IWK12(N67I) = LSTF5T + NBINA1 + 2
1242:                                IWK13(N67I) = ICXPN(IWK12(N67I)-1)
1243:                                DWK2(N67I) = X1
1244:                                else
1245:                                LSTF5T = ICXPN(LLSTF5E+IT-2)
1246:                                IWK12(N67I) = LSTF5T + NBINA1 + 2
1247:                                IWK13(N67I) = ICXPN(IWK12(N67I)-1)
1248:                                DWK2(N67I) = 1 - X1
1249:                                end if
1250:                                end do
1251: C ..... search xmu and save position in iwk14
1252:                                call BSINC3( CXPN, IWK9, R, IWK8, WK2(NCOLPN+1),
1253:                                & IWK14, KIII)
1254:                                call RANU2( IRAND, R, KIII*2, ICON )
1255:                                do J = 1, KIII
1256:                                IP = LSFFL(IFFLI+J)
1257:                                LL = IWK8(J)
1258:                                K1 = max(1, IWK14(J)) - 1
1259:                                INTMU = ICXPN(LL-2)
1260:                                NMU = ICXPN(LL-1)
1261:                                LST2A = ICXPN(LL+NMU+K1)
1262:                                NEP = ICXPN(LST2A)
1263:                                LQ1 = LST2A ! pre-position of Q_va
1264: C ..... B.M.C. sampling of emitted energy bin
1265: C/#IF ROUNDOFF(NEAREST)
1266:                                LL2 = min(int(NEP*R(J))+1, NEP)
1267: C/#ELSE
1268:                                *
1269: C/#ENDIF
1270:                                LL2 = NEP * R(J) + 1
1271:                                LL3 = LST2A + LL2
1272:                                LL4 = LL2 * 2
1273:                                LL4 = min(LL4, int(LL4+R(KIII+J)-CXPN(LL3)))
1274:                                & + LST2A + NEP
1275: C/#ELSE
1276:                                *
1277:                                *
1278: C/#ENDIF
1279:                                R2 = R(KIII+J) - CXPN(LL3)
1280:                                LL4 = LST2A + NEP + 2 * LL2 + R2
1281:                                LLL = LST2A + 3 * NEP
1282:                                L3 = LLL + ICXPN(LL4)
1283:                                L4 = L3 + NEP + 1
1284:                                E0 = CXPN(L3)
1285:                                E1 = CXPN(L3+1)
1286:                                P0 = CXPN(L4)
1287:                                P1 = CXPN(L4+1)
1288:                                RRR = (P0 + P1) * R(N2+J)
1289:                                RRR = RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1290:                                EEE(IP) = E1 + (E0 - E1) * RRR
1291:                                ! continuous spectra
1292:                                ! secondary energy in LAB
1293:                                C
1294:                                C
1295:                                C
1296:                                C
1297:                                C
1298:                                C
1299:                                C
1300:                                *VOCL LOOP,NOVREC
1301:                                do J = 1, N67I
1302:                                L2 = IWK12(J) + IWK2(J)
1303:                                L1 = L2 - 1
1304:                                L3 = IWK12(J) + IWK13(J) - 1
1305:                                INTA = ICXPN(IWK12(J)-2)
1306:                                if ( INTA.eq.1 ) then ! step interpo
1307:                                LST2A = ICXPN(IWK2(J)+L3)
1308:                                else ! linear inter

```

src/mvp/photnuc.f

```

polation
1309:      X1      = (CXPN(L1)-WK4(J)) /
1310:      &      (CXPN(L1)-CXPN(L2))
1311: C/#IF ROUND OFF(NEAREST)
1312:      IT      = min(1, int(R(J)+X1)) + IWK2(J)
1313: C/#ELSE
1314:      *      R1      = R(J) + X1
1315:      *      IT      = IWK2(J) + R1
1316: C/#ENDIF
1317:      LST2A    = ICXPN(IT+L3)
1318: end if
1319:      NEP      = ICXPN(LST2A)
1320:      LLE      = LST2A + 3 * NEP + 1
1321:      I        = IWK11(J)
1322:      IP       = LSFFL(IFFLI+I)
1323: C ..... linear interpolation is assumed.
1324:      if ( IWK10(I).eq.2 ) then ! unit base in
terpolation
1325:      EMAX     = R(NBANK4+I) + DWK2(J) *
1326:      &      (CXPN(LLE)-R(NBANK4+I))
1327:      EEE(IP) = EEE(IP) * EMAX / R(NBANK4+I)
1328: else ! method of co
responding point
1329:      EMAX     = R(NBANK4+I) + DWK2(J) *
1330:      &      (CXPN(LLE)-R(NBANK4+I))
1331:      EMIN     = R(NBANK5+I) + DWK2(J) *
1332:      &      (CXPN(LLE+NEP)-R(NBANK5+I))
1333:      EEE(IP) = EMIN + (EMAX-EMIN) *
1334:      &      (EEE(IP)-R(NBANK5+I)) /
1335:      &      (R(NBANK4+I)-R(NBANK5+I))
1336: end if
1337: end do
1338: end if
1339: C
1340: C ..... lf=60 : unknown distribution (law=7)
1341:      else if ( LF.eq.60 ) then
1342:      write(IOW,917) LF, KN, MT, IK, NK2GMT
1343:      stop 999
1344: C
1345: C ..... MF=5 ... energy distribution
1346:      else if ( LF.eq.1 .or. LF.eq.5 .or. LF.eq.7 .or.
1347:      &      LF.eq.9 .or. LF.eq.11 ) then
1348:      if ( LCTMT.ne.1 ) then
1349:      write(IOW,916) LCTMT, KN, MT, NK2GMT
1350:      stop 666
1351: end if
1352:      NEANG    = KLBNP2(KN,MT,6)
1353:      LSTF4    = KLBNP2(KN,MT,11)
1354:      LLST2GS  = LST2G + 1 + 2 * N30
1355:      do J = 1, KIII
1356:      IWK8(J)  = LSTF4
1357:      IWK9(J)  = NEANG
1358:      IWK6(J)  = ICXPN(LLST2GS+IWK11(J)) ! LST2GS(IK)
1359:      IWK12(J) = IWK6(J) + 2 ! LEENG
1360:      IWK13(J) = ICXPN(IWK6(J)+1) ! NEF5S
1361: end do
1362:      call BSDEC3( CXPN, IWK9, R, IWK8, WK1(NCOLPN+1),
1363:      &      IWK10, KIII ) ! for angular
distribution
1364:      call BSDEC3( CXPN, IWK13, R, IWK12, WK1(NCOLPN+1),
1365:      &      IWK14, KIII ) ! for energy d
istribution
1366:      N7RR    = KIII * 16 + 4
1367:      if ( N7RR.gt.6*NBANK ) then
1368:      N7RR    = 6 * NBANK
1369:      else if ( N7RR.lt.28 ) then
1370:      N7RR    = 28
1371: end if
1372:      call RANU2( IRAND, R, N7RR, ICON )
1373:      N7RR    = N7RR - 4
1374:      N7R     = N7
1375:      N7D     = N7RR - N7 ! number of ra
ndom numbers for resampling
1376:      do J = 1, KIII
1377:      IP      = LSFFL(IFFLI+J)
1378: C ..... calculate the scattering angle cosine from equal-probability
1379: C      bins of angular distribution
1380:      if ( IWK10(J).eq.0 ) IWK10(J) = 1
1381:      L2      = IWK8(J) + IWK10(J)
1382:      L1      = L2 - 1
1383:      INTE    = mod(ICXPN(L2+IWK9(J)), 10)
1384:      if ( INTE.eq.5 .or. INTE.eq.3 ) then ! log
interpolation
1385:      X1      = log(CXPN(L1)/WK1(NCOLPN+J)) /
1386:      &      log(CXPN(L1)/CXPN(L2))
1387: else ! line
ar interpolation
1388:      X1      = (CXPN(L1)-WK1(NCOLPN+J)) /
1389:      &      (CXPN(L1)-CXPN(L2))
1390: end if
1391:      X1      = min(1.0D0, max(0.0D0, X1))
1392: C/#IF ROUND OFF(NEAREST)
1393:      IT      = min(1, int(R(J)+X1)) + IWK10(J)
1394:      L3      = min(int(NBINA*R(KIII+J))+1, NBINA) +
1395:      &      IWK8(J) + 2*IWK9(J) + NBINA1*(IT-1)
1396: C/#ELSE
1397:      *      R1      = R(J) + X1
1398:      *      IT      = IWK10(J) + R1
1399:      *      R2      = R(KIII+J) * NBINA + 1
1400:      *      L3      = IWK8(J)+2*IWK9(J)+NBINA1*(IT-1)+R2
1401: C/#ENDIF
1402:      XMU     = (CXPN(L3)-CXPN(L3-1))*R(N2+J) +
1403:      &      CXPN(L3-1)
1404:      WK2(NCOLPN+J) = XMU ! scattering a
ngle cosine
1405: C ..... calculate the fraction of incident energy bin
1406:      if ( IWK14(J).eq.0 ) IWK14(J) = 1
1407:      L2      = IWK12(J) + IWK14(J)
1408:      L1      = L2 - 1
1409:      INTE0   = ICXPN(L2+IWK13(J))
1410:      INTE1   = INTE0 / 10
1411:      INTE    = INTE0 - 10 * INTE1
1412:      if ( INTE.eq.5 .or. INTE.eq.3 ) then ! log
interpolation
1413:      X1      = log(CXPN(L1)/WK1(NCOLPN+J)) /
1414:      &      log(CXPN(L1)/CXPN(L2))
1415: else ! line
ar interpolation
1416:      X1      = (CXPN(L1)-WK1(NCOLPN+J)) /
1417:      &      (CXPN(L1)-CXPN(L2))
1418: end if
1419:      X1      = min(1.0D0, max(0.0D0, X1))
1420:      L1      = IWK12(J) + 2 * IWK13(J)
1421: C
1422:      LF      = ICXPN(IWK6(J))
1423: C
1424: C ..... lf=1 : arbitrary tabulated function
1425:      if ( LF.eq.1 ) then
1426: C/#IF ROUND OFF(NEAREST)
1427:      IT      = min(1, int(R(N3+J)+X1)) + IWK14(J)

```

src/mvp/photnuc.f

```

1428: C/#ELSE
1429: *
1430: *
1431: C/#ENDIF
1432:
1433:
1434: C/#IF ROUNDOFF(NEAREST)
1435:
1436: C/#ELSE
1437: *
1438: C/#ENDIF
1439:
1440: C/#IF ROUNDOFF(NEAREST)
1441:
1442:
1443: &
1444: C/#ELSE
1445: *
1446: *
1447: C/#ENDIF
1448:
1449:
1450:
1451:
1452: if ( IT.eq.IWK14(J) ) then
1453:   NEP1 = NEP
1454:   LLE1 = LLE
1455:   LSTF5T = ICXPN(L1+IT)
1456:   NEP2 = ICXPN(LSTF5T)
1457:   LLE2 = LSTF5T + 3*NEP2 + 1
1458: else
1459:   NEP2 = NEP
1460:   LLE2 = LLE
1461:   LSTF5T = ICXPN(L1+IT-2)
1462:   NEP1 = ICXPN(LSTF5T)
1463:   LLE1 = LSTF5T + 3*NEP1 + 1
1464: end if
1465: E0 = CXPN(L3)
1466: E1 = CXPN(L3+1)
1467: P0 = CXPN(L4)
1468: P1 = CXPN(L4+1)
1469: RRR = (P0 + P1) * R(N6+J)
1470: EEE(IP) = E1 + (E0-E1)*RRR /
1471:   (sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1472: &
1473:
1474:
1475:
1476:
1477:
1478:
1479:
1480:
1481:
1482:
1483:
1484:
1485: C
1486: C ..... lf=5 : general evaporation spectrum
1487:
1488:
1489:
1490:
1491:
1492:
1493:
1494:
1495:
1496:
1497:
1498:
1499:
1500:
1501:
1502:
1503:
1504:
1505:
1506:
1507:
1508:
1509:
1510:
1511:
1512:
1513:
1514:
1515:
1516:
1517:
1518:
1519:
1520:
1521:
1522:
1523:
1524:
1525:
1526:
1527:
1528:
1529:
1530: C
1531: C ..... lf=7 : simple fission spectrum (Maxwellian)
1532:
1533:
1534:
1535:
1536:
1537:
1538:
1539:
1540:
1541:
1542:
1543:
1544:
1545:
1546:
1547:
1548:
1549:
1550:
1551:
1552:
1553: C
1554: C ..... lf=9 : evaporation spectrum
1555:

```

src/mvp/photnuc.f

```

1556:      L9      = L1 + IWK14(J)
1557:      THETA9  = CXPN(L9)*X1 + CXPN(L9-1)*(1-X1)
1558:      U9      = CXPN(L1+IWK13(J))
1559:      EHI9    = WK1(NCOLPN+J) - U9
1560: 230      continue
1561:      EEE(IP) = - THETA9 * log(R(N4+J)*R(N3+J))
1562:      if ( EEE(IP).gt.EHI9 ) then
1563:          R(N3+J) = R(N7R+1)
1564:          R(N4+J) = R(N7R+2)
1565:          N7R     = N7R + 2
1566:          if ( N7R.gt.N7RR ) then
1567:              call RANU2(IRAND,R(N7+1),N7D+4,ICON)
1568:              N7R     = N7
1569:          end if
1570:          go to 230
1571:      end if
1572: C
1573: C ..... If=11 : energy dependent Watt fission spectrum
1574:      else if ( LF.eq.11 ) then
1575:          L10    = L1 + IWK14(J)
1576:          L11    = L10 + IWK13(J)
1577:          A1     = CXPN(L10)*X1 + CXPN(L10-1)*(1-X1)
1578:          B      = CXPN(L11)*X1 + CXPN(L11-1)*(1-X1)
1579:          U11    = CXPN(L1+IWK13(J)*2)
1580:          EHI11  = WK1(NCOLPN+J) - U11
1581: 240      continue
1582:          COSI  = cos(PIH*R(N3+J))
1583:          COSI2 = COSI * COSI
1584:          V2    = - A1 *
1585:              (log(R(N4+J))+log(R(N5+J))*COSI2)
1586:          &      XMU    = 2 * R(N6+J) - 1
1587:          EEE(IP) = V2 + A1*XMU*sqrt(V2*B) + A1*A1*B/4
1588:          if ( EEE(IP).gt.EHI11 ) then
1589:              R(N3+J) = R(N7R+1)
1590:              R(N4+J) = R(N7R+2)
1591:              R(N5+J) = R(N7R+3)
1592:              R(N6+J) = R(N7R+4)
1593:              N7R     = N7R + 4
1594:              if ( N7R.gt.N7RR ) then
1595:                  call RANU2(IRAND,R(N7+1),N7D+4,ICON)
1596:                  N7R     = N7
1597:              end if
1598:              go to 240
1599:          end if
1600: C
1601: C ..... error process of unknown lf
1602:      else
1603:          write(IOW,914) LF, KN, MT
1604:          stop 999
1605:      end if
1606:      end do
1607:      else
1608:          write(IOW,915) LF, KN, MT, LST2G, LST2GS, IK
1609:          stop 999
1610:      end if
1611: C
1612: C ..... get energy and angle in LAB system using relativistic formula
1613:      if ( KCM2LAB.eq.1 ) then
1614: C ..... convert from CM to LAB system
1615:          do J = 1, KIII
1616:              IP    = LSFFL(IFFLI+J)
1617:              EE    = EEE(IP)                ! outgoing energy in C
M
1618:              EIN   = WK1(NCOLPN+J)          ! incident energy in L
AB
1619:              XMU   = WK2(NCOLPN+J)
1620:              WK1(NCOLPN+J) = EE              ! outgoing energy to d
etermine igg
1621: C ..... non relativistic
1622:          if ( EIN.le.EBRELA ) then
1623:              A1    = ATW + AMP
1624:              EEE(IP) = EE + (EIN+2*A1*XMU*sqrt(EIN*EE))
1625:              &      / A1 / A1
1626:              CTH   = XMU * sqrt(EE/EEE(IP)) +
1627:              &      sqrt(EIN/EEE(IP)) / A1
1628: C ..... relativistic
1629:          else
1630:              E12   = EIN + MNC2 * (AMP + ATW) ! energy of in
cident particle and target nuclide
1631:              B     = sqrt(EIN*(EIN+2*AMP*MNC2)) / E12
1632:              G     = 1 / sqrt(1-B*B)
1633:              ECM   = E12 / G
1634:              PMC2  = PMASS(KP) * MNC2
1635:              PMC4  = PMC2 ** 2
1636:              E4CM  = EE + PMC2
1637:              P4CMCX = sqrt(E4CM*E4CM-PMC4) * XMU
1638:              E4     = G * (E4CM + B * P4CMCX) ! Lorentz tran
sformation (energy)
1639:              P4CX   = G * (P4CMCX + B * E4CM) ! Lorentz tran
sformation
1640:              CTH    = P4CX / sqrt(E4*E4-PMC4)
1641:              EEE(IP) = E4 - PMC2            ! relativistic
outgoing energy in Lab
1642:          end if
1643:          DWK2(J) = min(1.0D0, max(-1.0D0, CTH))
1644:          end do
1645:      else if ( KCM2LAB.eq.0 ) then
1646:          do J = 1, KIII
1647:              IP    = LSFFL(IFFLI+J)
1648:              DWK2(J) = WK2(NCOLPN+J)
1649:              WK1(NCOLPN+J) = EEE(IP)
1650:          end do
1651:      end if
1652: 250      continue
1653:      do J = 1, KIII
1654:          IP    = LSFFL(IFFLI+J)
1655:          IP0   = IWK5(J)
1656:          TTT(IP) = TTT(IP0)                ! uncorrected
time
1657:          if ( JTIME.ne.0 ) ITT(IP) = ITT(IP0)
1658:      end do
1659:      if ( JJDLYN.ne.0.and.KDDD.gt.0 ) KIII = KIII + KDDD
1660: C ..... make direction vector
1661:      call RANU2( IRAND, R, KIII, ICON )
1662:      do J = 1, KIII
1663:          IP    = LSFFL(IFFLI+J)
1664:          IP0   = IWK5(J)
1665:          CTH   = DWK2(J)
1666:          AAAAA0 = AAA(IP0)
1667:          BBBB0  = BBB(IP0)
1668:          CCCCC0 = CCC(IP0)
1669:          CTH2   = 1 - CTH * CTH
1670:          if ( CTH2.le.1.0d-15 ) then
1671:              SINPSI = 0
1672:          else
1673:              SINPSI = sqrt(CTH2)
1674:          end if
1675:          ETA     = PI2 * R(J)
1676:          SINETA  = sin(ETA)
1677:          COSETA  = cos(ETA)

```

src/mvp/photnuc.f

```

1678:      STHETA = 1 - AAAAA0 * AAAAA0
1679:      *VOCL STMT,IF(100)
1680:      if ( STHETA.gt.0.0 ) then
1681:        STHETA = sqrt(STHETA)
1682:        COSPHI = BBBB0 / STHETA
1683:        SINPHI = CCCCC0 / STHETA
1684:      else
1685:        STHETA = 0
1686:        COSPHI = 1
1687:        SINPHI = 0
1688:      end if
1689:      BBBB0 = BBBB0 * CTH +
1690:      & AAAAA0 * COSPHI * COSETA * SINPSI -
1691:      & SINPHI * SINPSI * SINETA
1692:      CCCCC0 = CCCCC0 * CTH +
1693:      & AAAAA0 * SINPHI * COSETA * SINPSI +
1694:      & COSPHI * SINPSI * SINETA
1695:      AAAAA0 = AAAAA0 * CTH -
1696:      & COSETA * SINPSI * STHETA
1697:      S = sqrt(AAAAA0**2+BBBB0**2+CCCC0**2)
1698:      AAA(IP) = AAAAA0 / S
1699:      BBB(IP) = BBBB0 / S
1700:      CCC(IP) = CCCCC0 / S
1701:    end do
1702:  C
1703:  C ..... check the energy of produced particle with bottom energy
1704:  KJJJ = 0
1705:  KKDEAD = NDEAD + 1
1706:  do J = 1, KIII
1707:    IP = LSFFL(IFFLI+J)
1708:    if ( EEE(IP).gt.EBOTX(KP) ) then
1709:      KJJJ = KJJJ + 1
1710:      IWK8(KJJJ) = IP
1711:      IWK2(KJJJ) = J
1712:    else
1713:      NDEAD = NDEAD + 1
1714:      LSDED(NDEAD) = IP
1715:      NCNTR(8,KP) = NCNTR(8,KP) + 1
1716:      WCNTR(8,KP) = WCNTR(8,KP) + WK3(J)
1717:    end if
1718:  end do
1719:  if ( NDEAD.ge.KKDEAD ) then
1720:    if ( KIBNK(17).ne.0 ) then
1721:      do K = KKDEAD, NDEAD
1722:        IBNK(LSDED(K),KIBNK(17)) = 0
1723:      end do
1724:    end if
1725:    if ( KIBNK(18).ne.0 ) then
1726:      do K = KKDEAD, NDEAD
1727:        IBNK(LSDED(K),KIBNK(18)) = 0
1728:      end do
1729:    end if
1730:    if ( KIBNK(19).ne.0 ) then
1731:      do K = KKDEAD, NDEAD
1732:        IBNK(LSDED(K),KIBNK(19)) = 0
1733:      end do
1734:    end if
1735:  c##<2007/03/14:PN4:
1736:  do K = KKDEAD, NDEAD
1737:    KPNFG(LSDED(K)) = 0
1738:  end do
1739:  c##>
1740:  end if
1741:  if ( KJJJ.le.0 ) then
1742:    go to 2000      ! no particle production
1743:
1744:  else if ( KJJJ.lt.KIII ) then
1745:    do J = 1, KJJJ
1746:      LSFFL(IFFLI+J) = IWK8(J)
1747:      IWK5(J) = IWK5(IWK2(J))
1748:      WK3(J) = WK3(IWK2(J))
1749:      WK1(NCOLPN+J) = WK1(NCOLPN+IWK2(J))
1750:    end do
1751:    KIII = KJJJ
1752:  end if
1753:  C ..... set energy group number for outgoing energy
1754:  call BSVDEC( ENGYB(KENG(KP)), NGP(KP)+1,
1755:  & WK1(NCOLPN+1), IWK2, KIII)
1756:  NNG = KNGP(KP) - 1
1757:  do J = 1, KIII
1758:    IP = LSFFL(IFFLI+J)
1759:    IGG(IP) = NNG + IWK2(J)
1760:  end do
1761:  C ..... copy or set particle attributies except of energy and directions
1762:  do J = 1, KIII
1763:    IP = LSFFL(IFFLI+J)
1764:    IP0 = IWK5(J)
1765:    IBREG(IP) = IBREG(IP0)
1766:    IZFFL(IFFLI+J) = IZZ(IP0)
1767:    IZZ(IP) = IZZ(IP0)
1768:    KKP(IP) = KP
1769:    WWW(IP) = WK3(J)
1770:    XXX(IP) = XXX(IP0)
1771:    YYY(IP) = YYY(IP0)
1772:    ZZZ(IP) = ZZZ(IP0)
1773:    KLSF(IP) = 0
1774:    MMAC(IP,2) = 0
1775:  end do
1776:  do J = 1, KIII
1777:    IP = LSFFL(IFFLI+J)
1778:    IP0 = IWK5(J)
1779:    if ( JLATT.ne.0 ) LEVL(IP) = LEVL(IP0)
1780:    if ( JIMPT.ne.0 ) XIM(IP) = XIM(IP0)
1781:  end do
1782:  if ( JLATT.ne.0 ) then
1783:    do K = 1, NEST
1784:      do J = 1, KIII
1785:        IP = LSFFL(IFFLI+J)
1786:        IP0 = IWK5(J)
1787:        LZZ(IP,K) = LZZ(IP0,K)
1788:        LPOS(IP,K) = LPOS(IP0,K)
1789:        if ( JHLAT.ne.0 ) LCRS(IP,K) = LCRS(IP0,K)
1790:        if ( JTLT.ne.0 ) IBSPC(IP,K)=IBSPC(IP0,K)
1791:      end do
1792:    end do
1793:  end if
1794:  do K = 1, KDBNK(0)
1795:    if ( K.eq.KDBNK(1) ) then
1796:      do J = 1, KIII
1797:        IP = LSFFL(IFFLI+J)
1798:        DBNK(IP,KDBNK(1)) = WWW(IP)      ! birth weight
1799:      end do
1800:    else if ( K.eq.KDBNK(2) ) then
1801:      do J = 1, KIII
1802:        IP = LSFFL(IFFLI+J)
1803:        DBNK(IP,KDBNK(2)) = TTT(IP)      ! time of birt
1804:      end do
1805:    end if
1806:  end do

```

src/mvp/photnuc.f

```

1806:      else if ( K.eq.KDBNK(3) ) then
1807:        do J = 1, KIII
1808:          IP = LSFFL(IFFLI+J)
1809:          DBNK(IP,KDBNK(3)) = EEE(IP)      ! energy on bi
rth
1810:        end do
1811:      else
1812:        do J = 1, KIII
1813:          IP = LSFFL(IFFLI+J)
1814:          IP0 = IWK5(J)
1815:          DBNK(IP,KDBNK(K)) = DBNK(IP0,KDBNK(K))
1816:        end do
1817:      end if
1818:    end do
1819:    do K = 1, KIBNK(0)
1820:      if ( K.eq.KIBNK(4) ) then
1821:        do J = 1, KIII
1822:          IP = LSFFL(IFFLI+J)
1823:          IP0 = IWK5(J)
1824:          IBNK(IP,KIBNK(4)) = IBNK(IP0,KIBNK(4)) + 1 ! part
icle generation counter
1825:        end do
1826:      else if ( K.eq.KIBNK(5) ) then
1827:        do J = 1, KIII
1828:          IP = LSFFL(IFFLI+J)
1829:          IBNK(IP,KIBNK(5)) = 0 ! part
icle flight counter
1830:        end do
1831:      else if ( K.eq.KIBNK(17) ) then
1832:        --- stored in above ---
1833:      else if ( K.eq.KIBNK(18) ) then
1834:        if ( KPNPRD.eq.0 .or. KPNPRD.eq.1 ) then
1835:          do J = 1, KIII
1836:            IP = LSFFL(IFFLI+J)
1837:            IBNK(IP,KIBNK(18)) = NUC + NPATOM + KN ! prod
uce nuclide
1838:          end do
1839:        end if
1840:      else if ( K.eq.KIBNK(19) ) then
1841:        if ( KPNPRD.eq.0 .or. KPNPRD.eq.1 ) then
1842:          if ( MT.eq.18.and.JJDLYN.ne.0.and.KDDD.gt.0 )
1843:            & then
1844:            do J = 1, KIII-KDDD
1845:              IP = LSFFL(IFFLI+J)
1846:              IBNK(IP,KIBNK(19)) = NMT + NMTP + MT ! prod
uce reaction (prompt neutron)
1847:            end do
1848:          do J = KIII-KDDD+1, KIII
1849:            IP = LSFFL(IFFLI+J)
1850:            IBNK(IP,KIBNK(19)) = NMT + NMTP + 98 ! prod
uce reaction (delayed neutron)
1851:          end do
1852:        else
1853:          do J = 1, KIII
1854:            IP = LSFFL(IFFLI+J)
1855:            IBNK(IP,KIBNK(19)) = NMT + NMTP + MT ! prod
uce reaction
1856:          end do
1857:        end if
1858:      end if
1859:    else
1860:      do J = 1, KIII
1861:        IP = LSFFL(IFFLI+J)
1862:        IP0 = IWK5(J)
1863:        IBNK(IP,KIBNK(K)) = IBNK(IP0,KIBNK(K))
1864:      end do
1865:    end if
1866:  end do
1867:  c##<2007/03/14:PN4:
1868:    do J = 1, KIII
1869:      IP = LSFFL(IFFLI+J)
1870:      KPNFG(IP) = NUC + NPATOM + KN
1871:    end do
1872:  c##>
1873:  C
1874:  C ..... update IZFFL and NFFL
1875:    do J = 1, KIII
1876:      IZ = IZFFL(IFFLI+J)
1877:      NFFL(IZ) = NFFL(IZ) + 1
1878:    end do
1879:    NFFL(NZONE+1) = NFFL(NZONE+1) + KIII
1880:  C
1881:  2000      continue
1882:  3000      continue
1883:  3010      continue
1884:  4000      continue
1885:  5000      continue
1886:  C
1887:  C ..... all parent photons are transformed to dead stack.
1888:    do I = 1, NCOLPN
1889:      NDEAD = NDEAD + 1
1890:      LSDED(NDEAD) = IWK1(I)
1891:    end do
1892:    if ( KIBNK(17).ne.0 ) then
1893:      do I = 1, NCOLPN
1894:        IBNK(IWK1(I),KIBNK(17)) = 0
1895:      end do
1896:    end if
1897:    if ( KIBNK(18).ne.0 ) then
1898:      do I = 1, NCOLPN
1899:        IBNK(IWK1(I),KIBNK(18)) = 0
1900:      end do
1901:    end if
1902:    if ( KIBNK(19).ne.0 ) then
1903:      do I = 1, NCOLPN
1904:        IBNK(IWK1(I),KIBNK(19)) = 0
1905:      end do
1906:    end if
1907:  c##<2007/03/14:PN4:
1908:    do I = 1, NCOLPN
1909:      KPNFG(IWK1(I)) = 0
1910:    end do
1911:  c##>
1912:  c##<2007/03/14:PN4:
1913:  C
1914:  C-----
1915:  C**** uncollided contribution to next event estimators
1916:  C-----
1917:  C
1918:    if ( JP TDT.ne.0 ) then
1919:      JN = 0
1920:      JP = 0
1921:      do I = IFFLII+1, NFFL(NZONE+1)
1922:        IP = LSFFL(I)
1923:        if ( KKP(IP).eq.KPNEUT ) then
1924:          JN = JN + 1
1925:          IWK8(JN) = IP
1926:        else if ( KKP(IP).eq.KPPHOT ) then
1927:          JP = JP + 1
1928:          IWK9(JP) = IP

```


src/mvp/photnuc.f

```

1929:         end if
1930:     end do
1931: C
1932:     if ( JN.gt.0 ) then
1933:         JNP = 1
1934:         call PNUNCL( IOW, A, H, JNP, JN, IWK8,
1935: &                 XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZFL,
1936: &                 IMDED, IMSLT, IMNLT, IMZLT, TIMEB, KZMAT,
1937: &                 XXX, YYY, ZZZ, AAA, BBB, CCC, EEE, WWW, IGG,
1938: &                 TTT, ITT, LEVL, LZZ, LPOS, LCRS, LZZI, LPOSI,
1939: &                 LCRSI, KDBNK(0), KIBNK(0), DBNK, KDBNK, MDBNK,
1940: &                 IBNK, KIBNK, MIBNK, OPTI, PATH, KDET, KLSFI,
1941: &                 SMACI, MMACI,
1942: &                 H(PP(32)), H(PP(33)), H(PP(34)), H(PP(35)),
1943: &                 H(PP(36)), H(PP(37)), H(PP(38)), H(PP(39)),
1944: &                 H(PP(40)), H(PP(41)), H(PP(42)), H(PP(43)),
1945: &                 H(PP(44)), H(PP(45)), DWK1, DWK2, IWK10, IWK11,
1946: &                 IWK12, IWK13, R, SMCW )
1947:     end if
1948:     if ( JP.gt.0 ) then
1949:         JNP = 2
1950:         call PNUNCL( IOW, A, H, JNP, JP, IWK9,
1951: &                 XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZFL,
1952: &                 IMDED, IMSLT, IMNLT, IMZLT, TIMEB, KZMAT,
1953: &                 XXX, YYY, ZZZ, AAA, BBB, CCC, EEE, WWW, IGG,
1954: &                 TTT, ITT, LEVL, LZZ, LPOS, LCRS, LZZI, LPOSI,
1955: &                 LCRSI, KDBNK(0), KIBNK(0), DBNK, KDBNK, MDBNK,
1956: &                 IBNK, KIBNK, MIBNK, OPTI, PATH, KDET, KLSFI,
1957: &                 SMACI, MMACI,
1958: &                 H(PP(32)), H(PP(33)), H(PP(34)), H(PP(35)),
1959: &                 H(PP(36)), H(PP(37)), H(PP(38)), H(PP(39)),
1960: &                 H(PP(40)), H(PP(41)), H(PP(42)), H(PP(43)),
1961: &                 H(PP(44)), H(PP(45)), DWK1, DWK2, IWK10, IWK11,
1962: &                 IWK12, IWK13, R, SMCW )
1963:     end if
1964: end if
1965: c##>
1966: C
1967: C-----
1968: C**** calculation of microscopic cross section (smic array)
1969: C-----
1970: C
1971:     JN      = 0
1972:     JP      = 0
1973:     do I = IFLLI+1, NFFL(NZONE+1)
1974:         IP   = LSFFL(I)
1975:         if ( KKP(IP).eq.KPNEUT ) then
1976:             JN      = JN + 1
1977:             IWK8(JN) = IP
1978:         else if ( KKP(IP).eq.KPPHOT ) then
1979:             JP      = JP + 1
1980:             IWK9(JP) = IP
1981:         end if
1982:     end do
1983:     do N = 1, NUC
1984:         do I = 1, JN+JP
1985:             KSPI(LSFFL(I),N) = 0
1986:         end do
1987:     end do
1988: C
1989:     if ( JAMXC.eq.0 ) then
1990:         if ( JNEUT.ne.0.and.JN.gt.0 ) then
1991:             if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
1992:             if ( JVMNT.ne.0 ) call VMNTR1( 8, JN )
1993:             call GETMIC( JN, IWK8, IRAND, NUC, NMT, NBANK, NSMIC,

```

```

1994: &                 EEE, SMIC, LMIC, KSPI, CX, CX, MCX, KLBI1, KLBI2,
1995: &                 XLBI1, DWK1, DWK2, IWK11, IWK12, IWK13, IWK14,
1996: &                 IWK10, R, SMCW, RNU, NUC_MAX )
1997:         if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
1998:     end if
1999:     if ( JPHOT.ne.0.and.JP.gt.0 ) then
2000:         if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
2001:         if ( JVMNT.ne.0 ) call VMNTR1( 8, JP )
2002:         call GTMICP( JGAMM, JP, IWK9, NUC, NMTP, NPATOM, NBANK,
2003: &                 NSMIC, EEE, SMIC, KSPI, CXP, CXP, MCXP, KLBP1,
2004: &                 KLBP2, XLBP2, DWK1, IWK11, DWK2, IWK12 )
2005:         if ( JPHNU.ne.0 ) then
2006:             call GTMICPN( JP, IWK9, NUC, NUCPN, NUCPNI, NMTPN, NBANK,
2007: &                 NSMICPN, NUC_MAX, EEE, SMICPN, CXP, CXP, MCXP,
2008: &                 KLBP1, KLBP2, XLBP1, EBOTLPN, DWK1, DWK2,
2009: &                 IWK11, IWK12, IWK13, SMCW, RNU )
2010:         end if
2011:         if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
2012:     end if
2013: C
2014: C ..... ADAPTIVE-MICRO-CALCULATION mode (JAMXC > 0)
2015:     else
2016:         if ( JNEUT.ne.0.and.JN.gt.0 ) then
2017:             if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
2018:             if ( JVMNT.ne.0 ) call VMNTR1( 8, JN )
2019:             do I = 1, JN
2020:                 WK4(I) = EEE(IWK8(I))
2021:             end do
2022:             do NC = 1, NNCST
2023:                 IN      = INCST(NC,1)
2024:                 if ( IN.ne.0 ) then
2025:                     call GMICN1( IN, JN, IWK8, WK4, IRAND, NUC,
2026: &                     NMT, NBANK, NSMIC, SMIC, LMIC, KSPI, CX, CX,
2027: &                     MCX, KLBI1, KLBI2, XLBI1, DWK2, IWK11, IWK12,
2028: &                     IWK13, IWK14, R, SMCW, RNU, NUC_MAX)
2029:                 end if
2030:             end do
2031:             if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
2032:         end if
2033:         if ( JPHOT.ne.0.and.JP.gt.0 ) then
2034:             if ( JVMNT.ne.0 ) call VMNT00( 8, 5 )
2035:             if ( JVMNT.ne.0 ) call VMNTR1( 8, JP )
2036:             do I = 1, JP
2037:                 WK3(I) = log(EEE(IWK9(I)))
2038:                 WK4(I) = EEE(IWK9(I))
2039:             end do
2040:             do NC = 1, NNCST
2041:                 IN      = INCST(NC,2)
2042:                 if ( IN.ne.0 ) then
2043:                     call GMICP1( IN, JP, IWK9, WK3, JGAMM, NUC, NMTP,
2044: &                     NPATOM, NBANK, NSMIC, SMIC, KSPI, CXP, CXP,
2045: &                     MCXP, KLBP1, KLBP2, XLBP2, IWK11, DWK2, IWK12)
2046:                 end if
2047:             end do
2048:             if ( JPHNU.ne.0 ) then
2049:                 do NC = 1, NUCPN
2050:                     call GMICPN1( NC, JP, IWK9, WK4, NUC, NUCPN, NUCPNI,
2051: &                     NMTPN, NBANK, NUC_MAX, NSMICPN, SMICPN, CXP,
2052: &                     CXP, MCXP, KLBP1, KLBP2, XLBP1, EBOTLPN,
2053: &                     DWK2, IWK11, IWK12, IWK13, SMCW, RNU )
2054:                 end do
2055:             end if
2056:             if ( JVMNT.ne.0 ) call VMNT22( 8, 5 )
2057:         end if
2058:     end if

```

src/mvp/photnuc.f

```
2059: C
2060: C ..... pointwise response function
2061:   if ( NSTAL.gt.0 ) then
2062:     if ( JN.gt.0 ) then
2063:       JNP = 1
2064:       call GTSGL( JNP, JN, IWK8, NBANK, EEE, SGTAL, CRES, KCRES,
2065:         & NSTAL, MCRES, DWK1, IWK11 )
2066:     end if
2067:     if ( JP.gt.0 ) then
2068:       JNP = 2
2069:       call GTSGL( JNP, JP, IWK9, NBANK, EEE, SGTAL, CRES, KCRES,
2070:         & NSTAL, MCRES, DWK1, IWK11 )
2071:     end if
2072:   end if
2073: C
2074: C-----
2075: C**** russian roulette kill and splitting based on importance, weight
2076: C-----
2077: C
2078:   if ( JRRBT+JIMPT+JWIND.ne.0 ) then
2079:     ISFO = IFFLII + 1
2080:     NVARI = JN + JP
2081:     JJCOL = 1
2082:     IPAR0 = 0
2083:     call SPLITB( JJCOL, IPAR0, ISFO, NVARI, IRAND, NPKIND, NBANK,
2084:       & NZONE, NGRGUP, NREG, NTIME, NEST, LSFFL, IZFFL, MFFL,
2085:       & LSGED, NDEAD, NSMAC, NSMIC, XIMP, WKIL, WSRV, WTIME,
2086:       & WLLIM, KZREG, NCNTR, WCNTR, NEVENT, XXX, YYY, IZZ, AAA,
2087:       & BBB, CCC, WWW, KKP, IZZ, IGG, TTT, ITT, LEVL, LZZ,
2088:       & LPOS, LCRS, XIM, KLSF, EEE, IBREG, IBSPC, MB, NUC,
2089:       & NSTAL, SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL, DBNK,
2090:       & KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NKILD, WKILD, NSURV,
2091:       & WSURV, NSPLT, WSPLT, R, DWK1, DWK2, WK3, IWK11, IWK12,
2092:       & IWK13,
2093:       & SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD,
2094:       & WSC, NORDDS, NOPSDS,
2095:       & NUCPN, NSMICPN, SMICPN,
2096:       & WWB, WSB, NPTBE, WWBD, WSBD, WWLD, WSLD )
2097:   end if
2098: C
2099: C-----
2100: C**** process of photo-atomic reaction (photon interaction)
2101: C-----
2102: C
2103:   if ( NCOLPP.gt.0 ) then
2104:     do I = 1, NCOLPP
2105:       LSCLP(I) = IWK3(I)
2106:     end do
2107:   end if
2108: C
2109:   return
2110: C
2111: C ..... process of no photo-nuclear reaction
2112:   900 continue
2113:   NCOLPP = NCOLP
2114:   return
2115: C
2116: C ..... error messages
2117:   911 format(/' XXX(PHOTNUC) Starting position of product energy-angle',
2118:     & ' distribution data is illegal (LSTPEA<1). XXX'
2119:     & ' KN=',I5,' MATT=',A,' ZA=',1PE12.5,' LSTPEA=',I9)
2120:   912 format(/' XXX(PHOTNUC) Sampling of the reaction type (MT)',
2121:     & ' failed. XXX'
2122:     & ' KN=',I5,' MATT=',A,' ZA=',1PE12.5,' EEE=',E12.5,
2123:     & ' allowed maximum MT=',I4,' I=',I5,' IP=',I8
```

```
2124:     & ' sampling sigma=',E15.8,' cumulative sigma=',E15.8,
2125:     & ' random=',E15.8,' reaction sigma=',E15.8)
2126:   913 format(/' XXX(PHOTNUC) Too many particles are generated from',
2127:     & ' photo-nuclear reaction. XXX'
2128:     & ' Currently, the only remedy for this situation is to adjust',
2129:     & ' NBANK() and NHIST() (or NHSUB()) in your input data.'
2130:     & ' generated particle=',I3,' reaction MT=',I3,' subsection=',I4,
2131:     & ' generated number=',I8,' free space in bank=',I8
2132:     & ' number of mother particle=',I8,' KN=',I5,' MA=',I8
2133:     & ' weight of mother =',1p10e11.4))
2134:   914 format(/' XXX(PHOTNUC) Unimplemented lf is found. XXX'
2135:     & ' LF=',I4,' KN=',I5,' MT=',I3)
2136:   915 format(/' XXX(PHOTNUC) Unimplemented lf is found. XXX'
2137:     & ' LF=',I4,' KN=',I5,' MT=',I3,' LST2G=',I8,' LST2GS=',I8,
2138:     & ' IK=',i3)
2139:   916 format(/' XXX(PHOTNUC) In MF=5, the reference system of angular',
2140:     & ' distribution must be LAB. XXX'
2141:     & ' LCTMT=',I3,' KN=',I5,' MT=',I3,' NK2GMT=',I4)
2142:   917 format(/' XXX(PHOTNUC) Law=0 (unknown distribution) is not',
2143:     & ' allowed for particle production. XXX'
2144:     & ' LF=',I4,' KN=',I5,' MT=',I3,' IK=',I4,' NK2GMT=',I4)
2145:   918 format(/' XXX(photnuc) Energy distribution data of delayed',
2146:     & ' neutron is not MF=5 form.'
2147:     & ' KN=',i6,' NK2G=',i6,' KDDD=',i5)
2148:   end
```

src/mvp/photr.f

```

1: C/#IF ARGSAVE
2: *      subroutine PHOTR( IOW,IRAND, NPKIND,
3: *      N      NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,NBANK, BANKP, NEVENT,
4: *      N      NEST,NUC,NMTP,NPATOM,MB,NSMAC,NSMIC, NSTAL,
5: *      N      NEE, MTOP,
6: *      T      KZMAT, KZREG, XIMP, WKIL, WSRV, WTIME, WLLIM, RESP,
7: *      X      CXP, ICXP, MCXP, KLBPN1, XLBP1, KLBPN2, XLBP2,
8: *      X      SMAC, LMAC, KMAC, MMAC, SMIC, LMIC,
9: *      X      KSPI, SGTAL, ETOPP, EBOTP, ENGYB, NNCST, INCST, DNZON,
10: *      X      XLBE1, EEL, PBT, RKT, EBT, EBTP, IEFTP, WWAG, EEDG, EEEK,
11: *      X      LEMIC, NEMIC, CRES, KCRES, MCRES, NMKREG, MKREG,
12: c##<2007/03/14:PN3:PN4:
13: c##*      T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY )
14: *      T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY,
15: *      &      NUCNA, NUC_MAX, DNSTP, LPDNP, IATMT, NATMT, DNSTPN, MNUCPN,
16: *      &      LPIDPN, IZTBNP, NCIDPN, MNEUTPN, WGTNP, PPNBR, PFCNR, NPPNBR,
17: *      &      NPFCN, PMT, NPMT, MXPGEN, EBOTX, NSTALY, TCUT, TIMEB, CX, MCX,
18: *      &      KLBI1, KLBI2, XLBI1, NMT, NUCPN, NUCPNI, NMTPN, NSMICPN, MCXPN,
19: *      &      CXPN, ICXPN, KLBPN1, KLBPN2, XLBP1, XLBP2, SMICPN, ETOPLPN,
20: *      &      EBOTLPN, NNCSTPN, INCSTPN, NMTMPN, MTMPN, KPNPRD, NMONPNW,
21: *      &      MONPNWGT, NMPNWT, WMPNWT, KPNFG )
22: c##>
23: C/#ELSE
24:      subroutine PHOTR( IOW,IRAND, NPKIND,
25:      N      NGROUP,NZONE,NMAT,NREG,NTIME,NRESP,NBANK, BANKP, NEVENT,
26:      N      NEST,NUC,NMTP,NPATOM,MB,NSMAC,NSMIC, NSTAL,
27:      N      NEE, MTOP,
28:      T      KZMAT, KZREG, XIMP, WKIL, WSRV, WTIME, WLLIM, RESP,
29:      X      CXP, ICXP, MCXP, KLBPN1, XLBP1, KLBPN2, XLBP2,
30:      X      SMAC, LMAC, KMAC, MMAC, SMIC, LMIC,
31:      X      KSPI, SGTAL, ETOPP, EBOTP, ENGYB, NNCST, INCST, DNZON,
32:      X      XLBE1, EEL, PBT, RKT, EBT, EBTP, IEFTP, WWAG, EEDG, EEEK,
33:      X      LEMIC, NEMIC, CRES, KCRES, MCRES, NMKREG, MKREG,
34:      %      LSFFL, NFFL, IZFFL, LSCLP, NCOLP, LSDSD, NDEAD,
35:      B      XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE,
36:      B      TTT,ITT,XIM,KLSF,KKP,IZZ,IGG,LEVL,LZZ,LPOS,LCSR,IBREG,IBSPC,
37:      B      DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
38:      T      FLCL, RECL, NCLSN, WCLSN, NABSB, WABSB,
39:      T      NECUT, WECUT, NKILD, WKILD, NSURV, WSURV,
40:      T      NSPLT, WSPLT, NCNTR, WCNTR, RMIC, DNFLX,
41:      T      RSCL, JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY,
42:      W      R, DWK1, DWK2, WK3, WK4,
43:      W      ICLA, EIN, UOUT,
44:      W      IP2, IP3, IP4, IP5,
45:      W      IWK1, IWK2, IWK3, IWK4, IWK5, IWK6, IWK7,
46:      W      IWK10, IWK11,
47:      W      IWK20, WK20, WK21, WK22, WK23,
48:      W      WK41, WK42, WK43,
49:      &      SMICP, SMACP, NPTDS, NGSP, WWD, WWD2, WWR, WSD, WSC,
50:      &      NORDDS, NOPSDS,
51: c##<2007/03/14:PN3:PN4:
52:      &      NUCPNA, NUC_MAX, DNSTP, LPDNP, IATMT, NATMT, DNSTPN, MNUCPN,
53:      &      LPIDPN, IZTBNP, NCIDPN, MNEUTPN, WGTNP, PPNBR, PFCNR, NPPNBR,
54:      &      NPFCN, PMT, NPMT, MXPGEN, EBOTX, NSTALY, TCUT, TIMEB, CX, MCX,
55:      &      KLBI1, KLBI2, XLBI1, NMT, NUCPN, NUCPNI, NMTPN, NSMICPN, MCXPN,
56:      &      CXPN, ICXPN, KLBPN1, KLBPN2, XLBP1, XLBP2, SMICPN, ETOPLPN,
57:      &      EBOTLPN, NNCSTPN, INCSTPN, NMTMPN, MTMPN, KPNPRD, NMONPNW,
58:      &      MONPNWGT, NMPNWT, WMPNWT, WK5, SMCW, RNU, KPNFG, A, H,
59:      &      XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZFL, IMDED, IMSLT, IMNLT,
60:      &      IMZLT, LZZI, LPOS, LCRSI, OPTI, PATH, KDET, KLSFI, SMACI,
61:      &      MMACI, NPDET, NPLEN, IMPMAX, DWK17,
62: c##>
63: c+beff2
64:      &      WWB, WSB, NPTBE, WWBD, WSBD, WWLD, WSLD)
65: c-beff2

```

```

66: C/#ENDIF
67: C=====
68: C PURPOSE: COLLISION ANALYSIS FOR PHOTONS
69: C      (ENTRY PHOTR)
70: C CALLED IN: ACTION
71: C CALLS: TALLY,RANU2,GTMICP,ELGEN,TTBR
72: C=====
73:      implicit real*8(A-H,O-Z)
74: C
75:      include 'INC/_KPIDS'
76:      include 'INC/_NGPS'
77: C
78:      include 'INC/_FLAGS'
79: c##<2007/03/14:PN4:
80: C
81:      real A(*)
82:      real H(*)
83: c##>
84: C
85: C**** GEOMETRY ARRAY DATA
86: C
87:      integer KZMAT(NZONE), KZREG(NZONE)
88: C
89: C**** VARIANCE REDUCTION DATA
90: C
91: c##<2007/03/14:PN3:
92:      real*8 EBOTX(KPLIM)
93: c##>
94:      real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
95:      real WTIME(NTIME)
96:      real WLLIM
97: c##<2007/03/14:PN3:
98:      real WGTNP(NMTPN,NUCPN,NREG), PPNBR(NUCPN), TIMEB(NTIME+1), TCUT,
99:      &      PFCNR(NUCPN+1,NREG), PMT(2,NUCPN,NREG)
100: c##>
101: C
102: c##<2007/03/14:PN3:
103: c##C**** CROSS SECTION DATA IN CXP ARRAY
104: C**** CROSS SECTION DATA
105: C
106:      real CX(MCX), XLBI1(NUC,11)
107:      integer KLBI1(NUC,31), KLBI2(NUC,NMT,21)
108: c##>
109: C
110:      real ETOPP, EBOTP
111:      real CXP(MCXPN), XLBP1(NPATOM,10), XLBP2(NPATOM,NMTP,4)
112:      integer ICXP(MCXPN)
113:      integer KLBPN1(NPATOM,20), KLBPN2(NPATOM,NMTP,4)
114: c##<2007/03/14:PN3:
115: C
116:      real ETOPLPN, EBOTLPN
117:      real CXPN(MCXPN), XLBP1(NUCPNI,6), XLBP2(NUCPNI,NMTPN,2)
118:      integer ICXPN(MCXPN), KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
119: c##>
120: C
121: C**** CROSS SECTION DATA FOR BREMSSTRAHLUNG OF ELECTRON
122: C
123:      real EBOTE
124:      real XLBE1(NPATOM,6)
125:      real EEL(NEE), RKT(MTOP), PBT(NEE,NMAT), EBT(MTOP,NEE,NMAT),
126:      &      EBTP(MTOP-1,NEE,NMAT), WWAG(NMAT), EEDG(NMAT), EEEK(NMAT)
127:      integer IEFTP(2*(MTOP-1),NEE,NMAT)
128: C
129:      real CRES(MCRES)
130:      integer KCRES(NSTAL,4)

```

src/mvp/photr.f

```
131: c##<2007/03/14:PN3:
132: C
133: C**** Material data
134: C
135:     real DNSTP(*), DNSTPN(*)
136:     integer LPDNP(NMAT+1), IATMT(*), NATMT(*), MNUCPN(NMAT+1),
137: &         LPIDPN(*), IZTBPN(*), MNEUTPN(*)
138:     character NCIDPN(NUCPNI)*16
139: c##>
140: C
141: C**** SIGMA BANK
142: C
143:     real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
144: &         SGTAL(NBANK,NSTAL)
145: c##<2007/03/14:PN3:
146:     real SMICPN(NBANK,NUCPN,NSMICPN), RNU(NBANK,NUC_MAX,3)
147: c##>
148:     integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC), LMIC(8),
149: &         LMAC(8), LEMIC(8,2)
150: C
151: C ... variables for perturbation calculation ...
152: C
153:     real*8 WWD(NBANK,NPTDS,NORDDSD), WWD2(NBANK,NPTDS)
154:     real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
155:     real*8 WWR(NBANK,NPTCS)
156:     real*8 WSC(NBANK,0:NGSP,NPTCS)
157:     real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
158: c+beff2
159:     real*8 WWB(NBANK)
160:     real*8 WSB(NBANK,0:NGSP)
161:     real*8 WWBD(NBANK)
162:     real*8 WSBD(NBANK,NGSP)
163:     real*8 WWLD(NBANK)
164:     real*8 WSLD(NBANK,NGSP)
165: c-beff2
166: C
167: C**** STACK
168: C
169:     integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE+1), LSCLP(NBANK),
170: &         NCOLP, LSDED(NBANK), NDEAD
171: C
172: C**** PARTICLE BANK
173: C
174:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
175:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
176:     integer KKP(NBANK)
177:     integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
178: &         LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
179:     real EEE(NBANK), WWW(NBANK), XIM(NBANK)
180:     real*8 TTT(NBANK)
181:     integer ITT(NBANK)
182:     integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
183: C
184:     real*8 DBNK(NBANK,*)
185:     integer KDBNK(0:MDBNK)
186:     integer IBNK(NBANK,*)
187:     integer KIBNK(0:MIBNK)
188: c##<2007/03/14:PN4:
189:     integer KPNFG(NBANK)
190: c##>
191: C
192: C**** TALLY ARRAY
193: C
194: CCCC real ENGPB(NGP2+1)
195:     real ENGYB(*)

196:     integer INCST(NUC,*)
197: c##<2007/03/14:PN3:
198:     integer INCSTPN(NSTALY,2), MTMPN(NMTMPN)
199: c##>
200: C
201:     integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
202:     integer IDTAL(*)
203:     real*8 DTALY(*)
204: C
205: c##<2007/03/14:PN3:
206: c##     real RESP(NGROUP,NRESP), DNZON(NUC,NZONE)
207:     real RESP(NGROUP,NRESP), DNZON(NUC,NZONE,2)
208: c##>
209: CCCC real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
210:     real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
211:     real*8 FLCL(NGROUP,NREG), RECL(NREG,NRESP),
212: &         RMIC(NGROUP,NREG,NUC,NEMIC), RSCL(NREG,NSTAL),
213: &         DNFLX(NGROUP,NREG,NUC)
214:     real*8 WCLSN(NGROUP,NREG), WABSB(NGROUP,NREG), WECUT(NREG),
215: &         WKILD(NGROUP,NREG), WSURV(NGROUP,NREG),
216: &         WSPLT(NGROUP,NREG), NCLSN(NGROUP,NREG),
217: &         NABSB(NGROUP,NREG), NECUT(NREG), NKILD(NGROUP,NREG),
218: &         NSURV(NGROUP,NREG), NSPLT(NGROUP,NREG)
219:     integer MKREG(NREG,2)
220: c##<2007/03/14:PN3:
221:     real*8 NMPNWGT(*), WMPNWGT(2,*)
222:     integer MONPNWGT(4,NMONPNW)
223: c##>
224: c##<2007/03/14:PN4:
225: C
226: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR
227: C
228:     real*8 XPDET(NPLEN,NPDET)
229:     integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
230: C
231: C**** STACK FOR IMAGINARY PARTICLE
232: C
233:     integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(*), IMDED(IMPMAX),
234: &         IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
235: C
236: C**** PARTICLE BANK FOR IMAGINARY PARTICLE
237: C
238:     real*8 OPTI(IMPMAX), PATH(IMPMAX)
239:     integer LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST),
240: &         KDETP(IMPMAX), KLSFI(IMPMAX)
241: C
242: C**** CROSS SECTION AND GEOMETRY DATA FOR IMAGINARY PARTICLE
243: C
244:     real SMACI(*)
245:     integer MMACI(*)
246: c##>
247: C
248: C**** WORKING AREA
249: C
250:     integer ICLA(NBANK), IP2(NBANK), IP3(NBANK), IP4(NBANK),
251: &         IP5(NBANK), IWK1(NBANK), IWK2(NBANK), IWK3(NBANK),
252: &         IWK4(NBANK), IWK5(NBANK), IWK6(2*NBANK), IWK7(2*NBANK),
253: &         IWK10(2*NBANK), IWK11(NBANK)
254: c##<2007/03/14:PN3:
255: c##     real*8 DWK1(NBANK), DWK2(NBANK)
256:     real*8 DWK1(NBANK), DWK2(NBANK), DWK17(NBANK)
257: c##>
258:     real WK3(2*NBANK), WK4(2*NBANK), R(4*NBANK), EIN(NBANK),
259: &         UOUT(NBANK)
260:     integer IWK20(NBANK)
```

src/mvp/photr.f

```

261:      real WK20(NBANK), WK21(NBANK), WK22(NBANK), WK23(NBANK)
262:      real*8 WK41(2*NBANK), WK42(2*NBANK), WK43(2*NBANK)
263: c##<2007/03/14:PN3:
264:      real WK5(NBANK), SMCW(NBANK,NSMIC)
265: c##>
266: C
267: C
268: C      (TEMPORARY)
269:      integer INTTE0(4)
270: C
271:      parameter( PI  = 3.141592653589793238D0 )
272:      parameter( PI2 = 2*PI )
273: c##<2007/03/14:PN3:
274: c##      parameter( PIH = 0.5*PI )
275:      parameter( PIH = 0.5D0*PI )
276: c##>
277: C
278: C-----
279: C
280: C      .... CUTOFF ENERGY OF ELECTRON ....
281: C
282:      EBOTE  = 1.0E+3
283: C      .... EFFECTIVE ONLY IF ELECTRON TRANSPORT IS NOT CARRIED OUT.
284:      if ( EBOTE.lt.EBOTP ) EBOTE = EBOTP
285: c##<2007/03/14:PN3:
286: c##      ONE  = 0.999999
287:      ONE  = 0.999999D0
288: c##>
289: C
290: C      PI    = ACOS(-1.0D0)
291: C      PI2   = 2*PI
292: C      PIH   = 0.5*PI
293: C
294: C      .... REST ENERGY OF ELECTRON .....
295: C
296:      DMOC2  = 0.511D6
297: C
298: C      .... PHOTOELECTRIC THRESHOULD ENERGY ....
299: C
300:      EPRTH  = 1.022D6
301: C
302:      XK0    = XLBP1(1,4)
303: C
304: C      .... AUGER ELECTRONS ARE TAKEN INTO ACCOUNT WHEN JAUGE = 1
305: C
306:      JAUGE  = 0
307: C
308: C
309: C/#IF ARGSAVE
310: *      RETURN
311: C
312: C-----
313: C
314: *      entry PHOTR
315: *      S ( LSFFL, NFFL , IZFFL, LSCLP, NCOLP, LSDDED, NDEAD,
316: *      B   XXX , YY , ZZZ , AAA , BBB , CCC , WWW , EEE ,
317: *      B   TTT,ITT,XIM,KLSF,KKP,IZZ,IGG,LEVL,LZZ,LPOS,LCRS,IBREG,IBSPC,
318: *      B   DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
319: *      T   FLCL , RECL , NCLSN, WCLSN, NABSB, WABSB,
320: *      T   NECUT, WECUT, NKILD, WKILD, NSURV, WSURV,
321: *      T   NSPLT, WSPLT, NCNTR, WCNTR, RMIC , DNFLX,
322: *      T   RSCL,
323: *      W   R , DWK1 , DWK2 , WK3 , WK4 ,
324: *      W   ICLA , EIN ,UOUT,
325: *      W   IP2 , IP3 , IP4 , IP5 ,

```

```

326: *      W   IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 , IWK7 ,
327: *      W   IWK10, IWK11,
328: *      W   IWK20, WK20, WK21, WK22, WK23,
329: c##<2007/03/14:PN3:PN4:
330: c##*      W   WK41, WK42, WK43 )
331: *      W   WK41, WK42, WK43,
332: *      &   WK5, SMCW, RNU, A, H )
333: *      &   XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZFL, IMDED, IMSLT, IMNLT,
334: *      &   IMZLT, LZZI, LPOSI, LCRSI, OPTI, PATH, KDETP, KLSFI, SMACI,
335: *      &   MMACI, NPDET, NPLEN, IMPMAX, DWK17 )
336: c##>
337: C/#ENDIF
338: C
339: C
340: C=====
341: C
342: C      === BANK DATA TO BE UPDATED ===
343: C
344: C      (XXX,YYY,ZZZ),(AAA,BBB,CCC)      : POSITION,DIRECTION
345: C      EEE,IGG : ENERGY AND ITS GROUP.      WWW : WEIGHT
346: C
347: C      MMAC(,2) : NO. OF CALCULATED MACRO CROSS SECTIONS ----> 0
348: C      SMIC, KSPI : MICRO CROSS SECTIONS
349: C
350: C
351: C=====
352: C      if ( JVMNT.ne.0 ) call VMNTRI( 12, NCOLP )
353: C
354: C      ND0    = NDEAD
355: C
356: C-----
357: C**** TAKE MONITOR
358: C-----
359: C
360: C      if ( JMNTR.ne.0 ) then
361: *VOCL LOOP,SCALAR
362:      do 100 I = 1, NCOLP
363:          IGT  = IGG(LSCLP(I))
364:          if ( JTLT.eq.0 ) then
365:              IRT  = KZREG(IZZ(LSCLP(I)))
366:          else
367:              IRT  = IBREG(LSCLP(I))
368:          end if
369:          NCLSN(IGT,IRT) = NCLSN(IGT,IRT) + 1
370:          WCLSN(IGT,IRT) = WCLSN(IGT,IRT) + WWW(LSCLP(I))
371:      100 continue
372:      end if
373: C
374: C-----
375: C**** GATHER MB-VALUE, IREG, KNCOL, IGG, WEIGHT/SGTOT
376: C      ----> IWK1, IWK2, ICLA, IWK4, DWK1
377: C-----
378: C
379: *VOCL LOOP,NOVREC
380:      do 110 I = 1, NCOLP
381:          WCNTR(4,KPPHOT) = WCNTR(4,KPPHOT) + WWW(LSCLP(I))
382:          IWK1(I) = MMAC(LSCLP(I),1)
383:          DWK1(I) = WWW(LSCLP(I)) /SMAC(LSCLP(I),IWK1(I),LMAC(1))
384:          if ( JTLT.eq.0 ) then
385:              IWK2(I) = KZREG(IZZ(LSCLP(I)))
386:          else
387:              IWK2(I) = IBREG(LSCLP(I))
388:          end if
389:          ICLA(I) = KMAC(LSCLP(I),IWK1(I),1)
390:          IWK4(I) = IGG(LSCLP(I))

```

src/mvp/photr.f

```

391:      EIN(I) = EEE(LSCLP(I))
392:      110 continue
393: C
394:      NCNTR(4,KPPHOT) = NCNTR(4,KPPHOT) + NCOLP
395: C
396: C-----
397: c##<2007/03/14:PN3:
398: c##C**** Take "spectial" tally by photon collision estimator
399: C**** Take "special" tally by photon collision estimator
400: c##>
401: C-----
402: C
403:      if ( JTEVE.gt.0.and.JTEVE(11).gt.0 ) then
404:      do 120 I = 1, NCOLP
405:          IWK5(I) = IZZ(LSCLP(I))
406:          if ( JTIME.ne.0 ) then
407:              IWK7(I) = ITT(LSCLP(I))
408:              DWK2(I) = TTT(LSCLP(I))
409: c##<2007/03/14:PN3:
410:          DWK17(I) = 0
411: c##>
412:          end if
413:      120 continue
414:      do 150 J = 0, JTEVE(11) - 1
415: C
416: C ... pointer to direct tally (idt) & d-tally # (it) ...
417:          IDT = KTEVE(JTEVE(11)+J) - 1
418:          IT = IDTAL(IDT+9)
419: C
420: C ... skip if current region does not need this tally ...
421: C
422:          do 130 I = 1, NCOLP
423:              if ( JDTRG(IWK2(I),IT).ne.0 ) go to 140
424:      130 continue
425:          go to 150
426: C
427:      140 continue
428: C
429:          JPTIM2 = 0
430:          MZONE0 = 0
431: C
432: C << comment on Feb 23 2000 >>
433: c##<2007/03/14:PN3:
434: c##C ... TCUT00 is used for TCUT not passed to this routine,
435: c##C and it should not be used in this call.
436: c##C ... TIMEB0 is also a spacer for real TIMEB(*)
437: c##>
438: C ... Currently treatment for JPTIM > 0 in STALN1 may be
439: C incorrect for collision estimator
440: C
441:      call STALN1( IOW, JTIME, JPTIM, MZONE0, IDTAL(IDT+1), DWK1,
442:      & NCOLP, IWK5, IWK4, IWK2, LSCLP, IWK1, IWK7, DWK2,
443: c##<2007/03/14:PN3:
444: c## & TDUMY0, NGROUP, NREG, NRESP, NUC, NBANK, NSMIC,
445: & DWK17, NGROUP, NREG, NRESP, NUC, NBANK, NSMIC,
446: c##>
447: & NSMAC, NSTAL, MB, DNZON, NZONE, NTIME, NEMIC, RESP,
448: c##<2007/03/14:PN3:
449: c## & TIMEB0, NMKREG, MKREG, DTALY, SMIC, LMIC, SMAC,
450: c## & LMAC, SGTAL, TCUT00, DBNK, KDBNK, MDBNK, IBNK,
451: c## & KIBNK, MIBNK, R, IWK3, IWK11, JPTIM2 )
452: & TIMEB, NMKREG, MKREG, DTALY, SMIC, LMIC, SMAC,
453: & LMAC, SGTAL, TCUT, DBNK, KDBNK, MDBNK, IBNK,
454: & KIBNK, MIBNK, R, IWK3, IWK11, JPTIM2,
455: & EEE, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, CXPN,

```

```

456: & MCXPN, INCSTPN, MTMPN, NMTMPN, KLBPN1, KLBPN2,
457: & XLBPN1, EBOTLPN, NUCPNA, MNEUTPN, DNSTPN, MNUCPN,
458: & LPIDPN, SMCW(1,1), SMCW(1,2), SMCW(1,3), SMCW(1,4) )
459: c##>
460: C
461:      150 continue
462:      end if
463: C-----
464: C**** TAKE TALLY BY COLLISION ESTIMATOR
465: C-----
466: C
467: CTEMP---
468:      NEMICP = 0
469: C
470:      if ( JVMNT.ne.0 ) call VMNT00( 10, 12 )
471:      if ( JVMNT.ne.0 ) call VMNTR1( 10, NCOLP )
472: C
473: C .... CALL TALLY WITH JEIGN = 0 FOR PHOTONS ....
474: C
475: C
476:      JREGN = -1
477:      JZONE = 1
478:      call TALLYP( IOW, JRTCL, JREGN, JZONE, JRESP, DWK1, NCOLP, IWK4,
479:      & IWK2, LSCLP, DWK2, NGROUP, NGP(KPPHOT), NREG, NRESP, NUC,
480:      & NPATOM, NBANK, NSMIC, NSTAL, IZN, DNZON, NZONE, LEMIC,
481:      & NEMICP, RESP, SMIC, SGTAL, FLCL, RECL, RMIC, DNFLX, RSCL )
482: C
483: C .... DWK1, IWK4, IWK2 & IWK1 ARE FREE TO USE HEREFTER ....
484: C
485: C
486:      if ( JVMNT.ne.0 ) call VMNT22( 10, 12 )
487: c##<2007/03/14:PN3:PN4:
488: C
489: C-----
490: C**** separate photo-nuclear ractions
491: C-----
492: C
493:      if ( JPHNU.ne.0 ) then
494:          call PHOTNUC( IOW, IRAND,
495:      N NPKIND, NGROUP, NZONE, NMAT, NREG, NTIME, NBANK, NEVENT,
496:      N NEST, NUC, MB, NSMAC, NSMIC, NSMICPN, NUCPN, NUCPNI, MCXPN,
497:      N NMTPN, NSTAL, NUC_MAX,
498:      M DNSTP, LPDNP, IATMT, NATMT, DNSTPN, MNUCPN, LPIDPN, IZTBNP,
499:      M NCIDPN, KZREG,
500:      V XIMP, WKIL, WSRV, WGTN, WTIME, WLLIM, MXPGEN, EBOTX, PPNBR,
501:      V NPNBR, PFCNR, NPFCN, PMT, NPMT, TCUT, TIMEB,
502:      X CX, MCX, KL1B1, KL1B2, XL1B1, NMT, CXP, MCXP, KLBPN1, KLBPN2,
503:      X XLBP2, NMTP, NPATOM, SMAC, LMAC, KMAC, MMAC, SMIC, LMIC,
504:      X KSPI, SMICPN, CXPN, ICXPN, KLBPN1, KLBPN2, XLBPN1, XLBPN2,
505:      X ETOTLPN, EBOTLPN, CRES, KCRES, MCRES, SGTAL, NNCST, INCST,
506:      % LSFFL, NFFL, IZFFL, LSCLP, NCOLP, LSDED, NDEAD, NCOLPP,
507:      B XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE, TTT, ITT, XIM, KLSF,
508:      B KKP, IZZ, IGG, LEVL, LZZ, LPOS, LCRS, IBREG, IBSPC, DBNK,
509:      B IBNK, KDBNK, MDBNK, KIBNK, MIBNK,
510:      T KPNPRD, ENGB, NKILD, WKILD, NSURV, WSRV, NSPLT, WSPLT,
511:      T NCNTR, WCNTR, NMOPNPW, MONPNWGT, NMPNWT, WMPNWT,
512:      W R, DWK2, WK3, WK4, WK4(NBANK+1), UOUT, IP2, WK5, IWK1, IWK2,
513:      W IWK3, IWK4, IWK5, IWK6, IWK6(NBANK+1), IWK7, IWK7(NBANK+1),
514:      W IWK10, IWK10(NBANK+1), IWK11, IP3, IP4, IP5, SMCW, RNU,
515:      & KPNFG, A, H,
516:      & XPDET, IPDET, IPDT2, IMSFL, IMNFL, IMZF, IMDED, IMSLT,
517:      & IMNLT, IMZLT, LZZI, LPOSI, LCRSI, OPTI, PATH, KDET, KLSFI,
518:      & SMACI, MMACI, NPDET, NPLEN, IMPMAX, KZMAT,
519:      & SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD, WSC,
520:      & NORDDS, NOPS,

```

src/mvp/photr.f

```

521: c+beff2
522:   &      WWB, WSB, NPTBE, WWBD, WSBD, WWLD, WSLD)
523: c-beff2
524:   if ( NCOLPP.le.0 ) go to 1000      ! no photon interaction
525:   if ( NCOLPP.lt.NCOLP ) NCOLP = NCOLPP
526:   end if
527: C
528:   if ( KPNPRD.eq.0 ) then
529:     if ( KIBNK(17).ne.0 ) then
530:       if ( JTLTLT.eq.0 ) then
531: *VOCL LOOP,NOVREC
532:       do I = 1, NCOLP
533:         IBNK(LSCLP(I),KIBNK(17)) = - KZREG(IZZ(LSCLP(I)))      ! prod
534:       end do
535:     else
536: *VOCL LOOP,NOVREC
537:       do I = 1, NCOLP
538:         IBNK(LSCLP(I),KIBNK(17)) = - IBREG(LSCLP(I))      ! prod
539:       end do
540:     end if
541:   end if
542:   if ( KIBNK(18).ne.0 ) then
543: *VOCL LOOP,NOVREC
544:   do I = 1, NCOLP
545:     IBNK(LSCLP(I),KIBNK(18)) = NUC +
546:   &      KMAC(LSCLP(I),MMAC(LSCLP(I),1),1)      ! prod
547:   end do
548:   end if
549: c##<2007/03/14:PN4:
550:   if ( JPHNU.ne.0 ) then
551:     do I = 1, NCOLP
552:       KPNFG(LSCLP(I)) = 0
553:     end do
554:   end if
555: c##>
556:   end if
557: c##>
558: C
559: C=====
560: C GAMMA-RAY REACTION MODEL
561: C=====
562: C=====
563: C=====
564: C
565: C
566:   if ( JGAMM.ne.0 ) then
567: C
568: C-----
569: C
570: C.... WEIGHT REDUCTION FOR PHOTO ELECTRIC REACTION ....
571: C   AND ENRGY CUT.
572: C-----
573: C
574:   if ( JBREM.eq.0 ) then
575: *VOCL LOOP,NOVREC
576:   do 160 I = 1, NCOLP
577:     IP = LSCLP(I)
578:     TOTS = SMIC(IP,ICLA(I),1)
579: C
580:     WWW(IP) = WWW(IP)*(1.0-SMIC(IP,ICLA(I),5)/TOTS)
581: C
582:   160   continue

```

```

583: C
584:   else
585: C
586:     call RANU2( IRAND, R, NCOLP, ICON )
587: C
588:     NPHE = 0
589:     NPPE = 0
590:     NINE = 0
591: *VOCL LOOP,NOVREC
592:   do 170 I = 1, NCOLP
593:     IP = LSCLP(I)
594:     TOTS = SMIC(IP,ICLA(I),1)
595:     SPHE = SMIC(IP,ICLA(I),5)
596:     WW = WWW(IP)
597: C
598:     if ( TOTS*R(I).lt.SPHE ) then
599:       NPHE = NPHE + 1
600:       IWK4(NPHE) = IP
601:       WK3(NPHE) = EEE(IP)
602:       IP2(IP) = 0
603:     else
604:       IP2(IP) = 1
605:     end if
606:     DWK1(IP) = WW
607: C
608:     WWW(IP) = WW*(1.0-SPHE/TOTS)
609:   170   continue
610:   end if
611: C
612: C
613: C-----
614: C.... INCOHERENT SCATTERING / PAIR PRODUCTION
615: C-----
616: C
617: C
618:   call RANU2( IRAND, R, NCOLP, ICON )
619: C
620: C.... INCOHERENT SCATTERING ( E > 1.5 MEV )
621: C
622:   NINC = 0
623: C
624: C.... INCOHERENT SCATTERING ( E < 1.5 MEV )
625: C
626:   NINC2 = 0
627: C
628: C.... PAIR PRODUCTION .....
629: C
630:   NPPR = 0
631: *VOCL LOOP,NOVREC
632:   do 180 I = 1, NCOLP
633:     IP = LSCLP(I)
634:     WW = SMIC(IP,ICLA(I),3) /
635:   &      (SMIC(IP,ICLA(I),3)+SMIC(IP,ICLA(I),4))
636: C
637:     if ( R(I).lt.WW ) then
638: CM1993-10-19
639:       ICLAI = ICLA(I)
640:       EINI = EIN(I)
641: CM
642:       IF( EIN(I).GE. 1.5D6 ) THEN
643:         NINC = NINC + 1
644:         IP3(NINC) = IP
645: CM
646:         ICLA(NINC) = ICLA(I)
647: CM
648:         EIN(NINC) = EIN(I)

```

src/mvp/photr.f

```

648:      EIN(NINC) = EINI
649:      else
650:      NINC2 = NINC2 + 1
651:      IP5(NINC2) = IP
652: CM      IWK5(NINC2) = ICLA(I)
653:      IWK5(NINC2) = ICLAI
654: CM      DWK2(NINC2) = EIN(I)
655:      DWK2(NINC2) = EINI
656:      end if
657: C
658: CCCCCCCCCC ELSE IF( DWK2(I) .GE. EPRTH ) THEN
659:      else
660:      NPPR = NPPR + 1
661:      IP4(NPPR) = IP
662:      end if
663: C
664: 180 continue
665: C
666: C-----
667: C TREATMENT OF PAIR PRODUCTION
668: C
669: C UOUT(1:NPPR) : COSINES OF OUTGOING DIRECTIONS
670: C WOUT(1:NPPR) : NEW WEIGHT
671: C-----
672: C
673:      if ( NPPR.gt.0 ) then
674: C
675:      if ( JBREM.eq.1 ) then
676: C
677:      if ( DMOC2.gt.EBOTP ) then
678: *VOCL LOOP,NOVREC
679:      do 190 I = 1, NPPR
680:      if ( IP2(IP4(I)).gt.0 ) then
681:      NPPE = NPPE + 1
682:      WK20(NPPE) = DWK1(IP4(I))
683:      WK21(NPPE) = EEE(IP4(I))
684:      IWK20(NPPE) = IP4(I)
685:      end if
686: 190 continue
687:      else
688: *VOCL LOOP,NOVREC
689:      do 200 I = 1, NPPR
690:      if ( IP2(IP4(I)).gt.0 ) then
691:      NPPE = NPPE + 1
692:      WK20(NPPE) = DWK1(IP4(I))
693:      WK21(NPPE) = EEE(IP4(I))
694:      IWK20(NPPE) = -IP4(I)
695:      end if
696: 200 continue
697:      end if
698: C
699:      NINE = NPPE
700: C
701:      end if
702: C
703:      call RANU2( IRAND, R, NPPR, ICON )
704:      W = 0.0
705: *VOCL LOOP,NOVREC
706:      do 210 I = 1, NPPR
707:      W = W + WWW(IP4(I))
708:      UOUT(I) = 2.0*R(I) - 1.0
709:      WWW(IP4(I)) = WWW(IP4(I))*2.0
710:      EEE(IP4(I)) = DMOC2
711:      LSCLP(I) = IP4(I)
712: 210 continue

```

```

713: c##<2007/03/14:PN3:
714:      if ( KPNPRD.eq.0 ) then
715:      if ( KIBNK(19).ne.0 ) then
716: *VOCL LOOP,NOVREC
717:      do I = 1, NPPR
718:      IBNK(IP4(I),KIBNK(19)) = NMT + 6 ! pair production (MT=
516)
719:      end do
720:      end if
721:      end if
722: c##>
723: C
724:      NCNTR(23,KPPHOT) = NCNTR(23,KPPHOT) + NPPR
725:      NCNTR(20,KPPHOT) = NCNTR(20,KPPHOT) + NPPR
726:      WCNTR(23,KPPHOT) = WCNTR(23,KPPHOT) + W
727:      WCNTR(20,KPPHOT) = WCNTR(20,KPPHOT) + W*2.0
728:      end if
729: C
730: C
731: C
732: C-----
733: C TREATMENT OF INCOHERENT SCATTERING ( E > 1.5 MEV )
734: C
735: C SAMPLING IN ONE PATH.
736: C
737: C UOUT(NPPR+1:NPPR+NINC-1) : COSINES OF OUTGOING DIRECTIONS
738: C IP3(1:NINC) : BANK POINTER
739: C EIN(1:NINC) : BANK POINTER
740: C-----
741: C
742: C
743:      if ( NINC.gt.0 ) then
744:      call RANU2( IRAND, R, 2*NINC, ICON )
745:      do 220 I = 1, NINC
746:      AL = EIN(I) /DMOC2
747: C
748: C .... CALCULATE DIVISION RATIO FOR 4 TERMS IN KLINE-NISHINA FORMULA.
749: C
750: C
751:      A2 = 1.0/(AL*AL)
752:      A12 = 1.0 + 2.0*AL
753: C
754:      P1 = 2*A2
755:      P2 = (AL*(AL-2)-2)*A2/AL*LOG(A12) + P1
756:      P3 = P1 + P2
757:      P4 = (0.5-0.5/(A12*A12)) /AL + P3
758: C
759:      RR = R(NINC+I)*P4
760: C
761:      A12R = 1.0 + 2.0*R(I)*AL
762: C .....
763:      if ( RR.lt.P1 ) then
764:      EOUT = A12R
765: C
766:      else if ( RR.lt.P2 ) then
767:      EOUT = A12*R(I)
768: C
769:      else if ( RR.lt.P3 ) then
770:      EOUT = A12/A12R
771: C
772:      else
773:      EOUT = 1.0/SQRT(1+(1/(A12*A12)-1)*R(I))
774: C
775:      end if
776: C .....

```


src/mvp/photr.f

```

777: C
778:      UOUT(NPPR+I)      = 1.0 + (1.0-EOUT) /AL
779:      EEE(IP3(I)) = AL/EOUT*DMOC2
780: C
781:      LSCLP(NPPR+I)      = IP3(I)
782: C
783:      220      continue
784: c##<2007/03/14:PN3:
785:      if ( KPNPRD.eq.0 ) then
786:      if ( KIBNK(19).ne.0 ) then
787: *VOCL LOOP,NOVREC
788:      do I = 1, NINC
789:      IBNK(IP3(I),KIBNK(19)) = NMT + 4      ! incoherent scatterin
g (MT=504)
790:      end do
791:      end if
792:      end if
793: c##>
794: C
795: C
796:      if ( JBREM.eq.1 ) then
797: *VOCL LOOP,NOVREC
798:      do 230 I = 1, NINC
799:      if ( IP2(IP3(I)).gt.0 ) then
800:      NINE      = NINE + 1
801:      WK20(NINE) = DWK1(IP3(I))
802:      WK21(NINE) = BIN(I)
803:      WK22(NINE) = EEE(IP3(I))
804:      WK4(NINE)  = UOUT(I+NPPR)
805:      if ( WK22(NINE).gt.EBOTP ) then
806:      IWK20(NINE) = IP3(I)
807:      else
808:      IWK20(NINE) = -IP3(I)
809:      end if
810:      end if
811:      230      continue
812:      end if
813:      end if
814: C
815: C      ( IP3 IS FREE TO USE HEREAFTER )
816: C
817: C-----
818: C      TREATMENT OF INCOHERENT SCATTERING ( E < 1.5 MEV )
819: C
820: C
821: C      SAMPLING WITH REJECTION LOOP.
822: C
823: C      UOUT(NPPR+NINC+1:NCOLP) : COSINES OF OUTGOING DIRECTIONS
824: C      IWK5(1:NINC2)          : COLLIDING ATOM
825: C      IP5(1:NINC2)           : BANK POINTER
826: C      DWK2(1:NINC2)          : ENERGY
827: C      IWK6(1:NINC2)          : MEMORY OF INITIAL POSITION IN WORKING ARRAY
828: C      TO STORE SAMPLED OUTGOING ANGLES IN UOUT.
829: C-----
830: C
831: C
832:      if ( NINC2.gt.0 ) then
833:      NDET      = 0
834: C
835: C      .... DWK2 --> ALPHA
836: C
837:      if ( JBREM.eq.0 ) then
838:      do 240 I = 1, NINC2
839:      IWK6(I) = NPPR + NINC + I
840:      DWK2(I) = DWK2(I) /DMOC2

```

```

841:      240      continue
842: C
843:      else
844:      NN      = NINE
845:      do 250 I = 1, NINC2
846:      IWK6(I) = NPPR + NINC + I
847:      DWK2(I) = DWK2(I) /DMOC2
848:      IP4(I) = IP5(I)
849:      if ( IP2(IP5(I)).gt.0 ) then
850:      NN      = NN + 1
851:      WK20(NN) = DWK1(IP5(I))
852:      WK21(NN) = EEE(IP5(I))
853:      IWK20(NN) = IP5(I)
854:      end if
855:      250      continue
856:      end if
857: C
858:      260      NN      = NINC2 - NDET
859: C
860:      if ( NN.gt.0 ) then
861:      call RANU2( IRAND, R, 3*NN, ICON )
862:      NN2      = 0
863: C
864: C      .... SAMPLE:      X = OUTGOING-ENERGY/INCIDENT-ENERGY
865: C
866: *VOCL LOOP,NOVREC
867:      do 270 I = 1, NN
868:      IWK6O      = IWK6(I)
869:      IP5O      = IP5(I)
870: c##<2007/03/14:PN4:
871: c##      A      = DWK2(I)
872: c##      A12     = 1.0 + 2.0*A
873: c##      ASQ     = A*A
874: c##      A1      = DWK2(I)
875: c##      A12     = 1.0 + 2.0*A1
876: c##      ASQ     = A1*A1
877: c##>
878: C
879: C      .... SAMPLING      G(X) = GMIN * ( -A*X + 1.0 + A ) ...
880: C      (      1/(1+2*A) < X < 1 )
881: C
882: C      G(1/(1+2*A)) = GMIN * (2*A**2 + 2*A + 1) / (1+2*A)
883: C      G(      1 ) = GMIN * 1
884: C
885: C      X1 = 1 - ( 1 - 1/(1+2*A) ) * R1
886: C      X2 = 1/(1+2*A) + ( 1 - 1/(1+2*A) ) * R1
887: C
888: C      R1*G(1/(1+2*A)) + (1-R1)*G(1)
889: C      R2 < ----- ==> ADOPT X1
890: C      G(1/(1+2*A)) + G(1)      ELSE X2
891: C
892: C
893: c##<2007/03/14:PN4:
894: c##      X      = 2.0*A*R(I) /A12
895: c##      X      = 2.0*A1*R(I) /A12
896: c##>
897:      if ( 2.0*R(I+NN)*(ASQ+A12).lt.A12+2.0*ASQ*R(I) ) then
898:      X      = 1.0 - X
899:      else
900:      X      = 1.0/A12 + X
901:      end if
902: C
903: C      .... IF      H(X)/( 2/GMIN ) > R(I+2*NN) THEN ACCEPT
904: C
905: c##<2007/03/14:PN4:

```

src/mvp/photr.f

```

906: c##          if ( 2.0*ASQ*(1.0+A*(1.0
907:              if ( 2.0*ASQ*(1.0+A1*(1.0
908: c##>
909: &              -X))*R(I+2*NN)*X*X.lt.1.0
910: &              +X*(ASQ-A12-1.0+X*(A12+X*ASQ)) ) then
911: CM              UOUT(IWK6(I)) = 1.0 + (1.0 -1.0/X)/A
912: CM              LSCLP(IWK6(I)) = IP5(I)
913: CM              EEE(IP5(I)) = X * A * DM0C2
914: c##<2007/03/14:PN4:
915: c##              UOUT(IWK6O) = 1.0 + (1.0-1.0/X) /A
916:              UOUT(IWK6O) = 1.0 + (1.0-1.0/X) /A1
917: c##>
918:              LSCLP(IWK6O) = IP5O
919: c##<2007/03/14:PN4:
920: c##              EEE(IP5O) = X*A*DM0C2
921:              EEE(IP5O) = X*A1*DM0C2
922: c##>
923: c##<2007/03/14:PN3:
924:              if ( KPMPRD.eq.0 ) then
925:                  if ( KIBNK(19).ne.0 )
926: &                  IBNK(IP5O,KIBNK(19)) = NMT+4 ! incoherent scatterin
g (MT=504)
927:                  end if
928: c##>
929:                  else
930:                      NN2 = NN2 + 1
931: CM                      IP5(NN2) = IP5(I)
932: CM                      DWK2(NN2) = DWK2(I)
933: CM                      IWK6(NN2) = IWK6(I)
934:                      IP5(NN2) = IP5O
935: c##<2007/03/14:PN4:
936: c##                      DWK2(NN2) = A
937:                      DWK2(NN2) = A1
938: c##>
939:                      IWK6(NN2) = IWK6O
940:                  end if
941: 270                  continue
942:                  NDET = NDET + NN - NN2
943:                  go to 260
944:                  end if
945: C
946:                  if ( JBREM.eq.1 ) then
947: C
948: *VOCL LOOP,NOVREC
949:                  do 280 I = 1, NINC2
950:                      if ( IP2(IP4(I)).gt.0 ) then
951:                          NINE = NINE + 1
952:                          WK22(NINE) = EEE(IP4(I))
953:                          WK4(NINE) = UOUT(I+NPPR+NINC)
954:                          if ( WK22(NINE).le.EBOTP ) IWK20(NINE) = -IP4(I)
955:                      end if
956: 280                  continue
957:                  end if
958: C
959:                  end if
960: C
961: C ..... PHOTO-ELECTRON
962: C
963:                  if ( JBREM.eq.1 ) then
964: C
965:                      NN = NINE
966: C
967: C ..... ADD PHOTO-ELECTRON
968: C
969:                  if ( NPHE.gt.0 ) then

```

```

970: *VOCL LOOP,NOVREC
971:                  do 290 I = 1, NPHE
972:                      EP = WK3(I)
973:                      EPEEL = EP
974: C
975: C                      EP >> EEDG
976: C                      EPEEL = EP - EEDG( KZMAT(IZZ(IWK4(I))) )
977: C
978:                      if ( EPEEL.gt.EBOTE ) then
979:                          NN = NN + 1
980:                          WK20(NN) = DWK1(IWK4(I))
981:                          WK21(NN) = EP
982:                          WK22(NN) = EPEEL
983:                          if ( EEE(IWK4(I)).gt.EBOTP ) then
984:                              IWK20(NN) = IWK4(I)
985:                          else
986:                              IWK20(NN) = -IWK4(I)
987:                          end if
988:                      end if
989: 290                  continue
990:                  end if
991: C
992:                      IMTTE0(1) = NPPE
993:                      IMTTE0(2) = NINE - NPPE
994:                      IMTTE0(3) = NN - NINE
995:                      IMTTE0(4) = 0
996:                      IMTTE = NN
997: C
998:                  end if
999: C
1000: C
1001: C ( IP5, DWK2, IWK6 IS FREE TO USE HEREAFATER )
1002: C
1003: C
1004: C=====
1005: C=====
1006: C GENERALIZED PHOTON REACTION MODEL
1007: C=====
1008: C=====
1009: C
1010: C
1011:                  else if ( JGAMM.eq.0 ) then
1012: C
1013: C
1014: C-----
1015: C.... PAIR PRODUCTION .....
1016: C-----
1017: C
1018: C
1019: C
1020:                      NPPR = 0
1021:                      NN = 0
1022:                      call RANU2( IRAND, R, NCOLP, ICON )
1023: *VOCL LOOP,NOVREC
1024:                      do 300 I = 1, NCOLP
1025:                          IP = LSCLP(I)
1026:                          TOTS = SMIC(IP,ICLA(I),1)
1027: C
1028: CM1993-10-19
1029:                      IC LAI = IC LA(I)
1030:                      EINI = EIN(I)
1031: CM                      IF( R(I)*TOTS .LT. SMIC(IP,ICLA(I),4)) THEN
1032:                      if ( R(I)*TOTS.lt.SMIC(IP,ICLA(I),4) ) then
1033:                          NPPR = NPPR + 1
1034:                          LSCLP(NPPR) = IP

```

src/mvp/photr.f

```

1035: CM          IWK3(NPPR) = ICLA(I)
1036:          IWK3(NPPR) = ICLAI
1037: CM          DWK2(NPPR) = EIN(I)
1038:          DWK2(NPPR) = EINI
1039:          else
1040:            NN = NN + 1
1041:            IP4(NN) = IP
1042: CM          ICLA(NN) = ICLA(I)
1043:          ICLA(NN) = ICLAI
1044: CM          EIN(NN) = EIN(I)
1045:          EIN(NN) = EINI
1046:          end if
1047: 300 continue
1048:          NCOLP = NN
1049: C
1050: C-----
1051: C          UOUT(1:NPPR) : COSINES OF OUTGOING DIRECTIONS
1052: C-----
1053: C
1054:          if ( NPPR.gt.0 ) then
1055:            call RANU2( IRAND, R, NPPR, ICON )
1056: C
1057:            W = 0.0
1058:            if ( JBREM.eq.1 ) then
1059:              if ( DMOC2.gt.EBOTP ) then
1060:                *VOCL LOOP,NOVREC
1061:                do 310 I = 1, NPPR
1062:                  WK20(I) = WWW(LSCLP(I))
1063:                  W = W + WK20(I)
1064:                  WK21(I) = DWK2(I)
1065:                  IWK20(I) = LSCLP(I)
1066: C
1067:                  UOUT(I) = 2.0*R(I) - 1.0
1068:                  WWW(LSCLP(I)) = WWW(LSCLP(I))*2
1069:                  EEE(LSCLP(I)) = DMOC2
1070: 310 continue
1071:                else
1072:                *VOCL LOOP,NOVREC
1073:                do 320 I = 1, NPPR
1074:                  WK20(I) = WWW(LSCLP(I))
1075:                  W = W + WK20(I)
1076:                  WK21(I) = DWK2(I)
1077:                  IWK20(I) = -LSCLP(I)
1078: C
1079:                  UOUT(I) = 2.0*R(I) - 1.0
1080:                  WWW(LSCLP(I)) = WWW(LSCLP(I))*2
1081:                  EEE(LSCLP(I)) = DMOC2
1082: 320 continue
1083:                end if
1084: C
1085:                else
1086:                *VOCL LOOP,NOVREC
1087:                do 330 I = 1, NPPR
1088:                  W = W + WWW(LSCLP(I))
1089:                  UOUT(I) = 2.0*R(I) - 1.0
1090:                  WWW(LSCLP(I)) = WWW(LSCLP(I))*2
1091:                  EEE(LSCLP(I)) = DMOC2
1092: 330 continue
1093:                end if
1094: c##<2007/03/14:PN3:
1095:                if ( KPNPRD.eq.0 ) then
1096:                  if ( KIBNK(19).ne.0 ) then
1097:                    *VOCL LOOP,NOVREC
1098:                    do I = 1, NPPR
1099:                      IBNK(LSCLP(I),KIBNK(19)) = NMT + 6

```

! pair product

```

ion (MT=516)
1100:          end do
1101:          end if
1102:          end if
1103: c##>
1104: C
1105:          NCNTR(23,KPPHOT) = NCNTR(23,KPPHOT) + NPPR
1106:          NCNTR(20,KPPHOT) = NCNTR(20,KPPHOT) + NPPR
1107:          WCNTR(23,KPPHOT) = WCNTR(23,KPPHOT) + W
1108:          WCNTR(20,KPPHOT) = WCNTR(20,KPPHOT) + W*2.0
1109:          end if
1110: C
1111:          N1 = NPPR
1112: C
1113: C
1114: C-----
1115: C.... INCOHERENT SCATTERING
1116: C-----
1117: C
1118: C
1119:          NINC = 0
1120:          NINC2 = 0
1121:          NN = 0
1122:          call RANU2( IRAND, R, NCOLP, ICON )
1123:          *VOCL LOOP,NOVREC
1124:          do 340 I = 1, NCOLP
1125:            IP = IP4(I)
1126:            TOTS = SMIC(IP,ICLA(I),2) + SMIC(IP,ICLA(I),3)
1127:            & + SMIC(IP,ICLA(I),5)
1128: C
1129: CM1993-10-19
1130:          ICLAI = ICLA(I)
1131:          EINI = EIN(I)
1132: CM          IF( R(I)*TOTS .LT. SMIC(IP,ICLA(I),3)) THEN
1133:          if ( R(I)*TOTS.lt.SMIC(IP,ICLAI,3) ) then
1134: CM          IF( EIN(I).GT. 1.5D6 ) THEN
1135:          if ( EINI.gt.1.5D6 ) then
1136:            NINC = NINC + 1
1137:            IP3(NINC) = IP
1138: CM          IWK3(NINC) = ICLA(I)
1139:            IWK3(NINC) = ICLAI
1140: CM          DWK2(NINC) = EIN(I)
1141:            DWK2(NINC) = EINI
1142:          else
1143:            NINC2 = NINC2 + 1
1144:            IP5(NINC2) = IP
1145: CM          IWK5(NINC2) = ICLA(I)
1146:            IWK5(NINC2) = ICLAI
1147: CM          DWK1(NINC2) = EIN(I)
1148:            DWK1(NINC2) = EINI
1149:          end if
1150:          else
1151:            NN = NN + 1
1152:            IP4(NN) = IP
1153: CM          ICLA(NN) = ICLA(I)
1154:            ICLA(NN) = ICLAI
1155: CM          EIN(NN) = EIN(I)
1156:            EIN(NN) = EINI
1157:          end if
1158: 340 continue
1159:          NCOLP = NN
1160: C
1161: C-----
1162: C          TREATMENT OF INCOHERENT SCATTERING ( E > 1.5 MEV )
1163: C

```

src/mvp/photr.f

```

1164: C      UOUT(NPPR+1:NPPR+NINC-1) : COSINES OF OUTGOING DIRECTIONS
1165: C      IP3(1:NINC)                : BANK POINTER
1166: C      IWK3(1:NINC)              : COLLIDING ATOM
1167: C      DWK2(1:NINC)              : ENERGY
1168: C
1169: C      ( IP5, IWK5, DWK1 : RESERVED FOR NEXT STEP ( E < 1.5MEV )
1170: C
1171: C-----
1172: C
1173: C
1174: C      if ( NINC.gt.0 ) then
1175: C          IDET = 0
1176: C          NINC0 = NINC
1177: C          NREJ = 0
1178: C      c##<2017/03/14:PN3:
1179: C      if ( KPNPRD.eq.0 ) then
1180: C          if ( KIBNK(19).ne.0 ) then
1181: C              *VOCL LOOP,NOVREC
1182: C              do I = 1, NINC
1183: C                  IBNK(IP3(I),KIBNK(19)) = NMT + 4 ! incoherent scattering
1184: C              end do
1185: C          end if
1186: C      end if
1187: C      c##>
1188: C
1189: C      350      continue
1190: C      if ( IDET.lt.NINC0 ) then
1191: C          NREJ = NREJ + 1
1192: C
1193: C
1194: C      .... CALCULATE OUTGOING ENERGY FROM KLINE-NISHINA FORMULA.
1195: C
1196: C
1197: C      call RANU2( IRAND, R, 2*NINC, ICON )
1198: C      N2 = N1 + IDET
1199: C      *VOCL LOOP,NOVREC
1200: C      do 360 I = 1, NINC
1201: C          AL = DWK2(I) /DMOC2
1202: C          A2 = 1/(AL*AL)
1203: C          A12 = 1 + 2*AL
1204: C
1205: C          P1 = 2*A2
1206: C          P2 = (AL*(AL-2)-2)*A2/AL*LOG(A12) + P1
1207: C          P3 = P1 + P2
1208: C          P4 = (1-1/(A12*A12)) / (A12-1) + P3
1209: C
1210: C          RR = R(NINC+I)*P4
1211: C
1212: C          A12R = 1.0 + 2*R(I)*AL
1213: C          if ( RR.lt.P1 ) then
1214: C              EOUT = A12R
1215: C          else if ( RR.lt.P2 ) then
1216: C              EOUT = A12**R(I)
1217: C          else if ( RR.lt.P3 ) then
1218: C              EOUT = A12/A12R
1219: C          else
1220: C              EOUT = 1/SQRT(1+(1/(A12*A12)-1)*R(I))
1221: C          end if
1222: C
1223: C          XMU = 1 + (1-EOUT) /AL
1224: C          if ( ABS(XMU).eq.1.0 ) XMU = XMU*ONE
1225: C          UOUT(N2+I) = XMU
1226: C
1227: C          EOUT = 1/EOUT

```

```

1228: C          WK4(I) = AL*EOUT*DMOC2
1229: C
1230: C      .... V(Q,Z) & VMAX(Q,Z) (-> WK3 ) ....
1231: C
1232: C          WK3(I) = XK0*(AL*SQRT(1+EOUT*(EOUT-2*XMU)))
1233: C          WK3(I+NINC) = XK0*AL*(1.+1./A12)
1234: C
1235: C      .... LENGTH & START POSITION OF V-TABLE ....
1236: C
1237: C          IWK6(I) = KLBP1(IWK3(I),6)
1238: C          IWK6(NINC+I) = IWK6(I)
1239: C          IWK7(I) = KLBP1(IWK3(I),9)
1240: C          IWK7(NINC+I) = IWK7(I)
1241: C      360      continue
1242: C
1243: C      .... TABLE SEARCH FOR V & VMAX ....
1244: C
1245: C          T      NT      MW      LS      X      IT      N1
1246: C          call BSINC3( CXP, IWK6, R, IWK7, WK3, IWK10, NINC*2 )
1247: C
1248: C      .... INTERPOLATE S(V,Z) TABLE ....
1249: C
1250: C
1251: C      do 370 II = 1, 2*NINC
1252: C          LS = IWK7(II) + IWK10(II)
1253: C          TI = (CXP(LS)-WK3(II)) / (CXP(LS)-CXP(LS-1))
1254: C
1255: C          if ( TI.ge.0. ) then
1256: C              LS1 = LS + IWK6(II)
1257: C              WK3(II) = CXP(LS1) + TI*(CXP(LS1-1)-CXP(LS1))
1258: C          else
1259: C              WK3(II) = 0.0
1260: C          end if
1261: C      370      continue
1262: C
1263: C      .... CHECK OF REJECTION / ACCEPTION ....
1264: C
1265: C      call RANU2( IRAND, R, NINC, ICON )
1266: C      NN = 0
1267: C
1268: C      if ( JBREM.eq.1 ) then
1269: C          *VOCL LOOP,NOVREC
1270: C          do 380 I = 1, NINC
1271: C              IP30 = IP3(I)
1272: C              EPIN = DWK2(I)
1273: C              if ( R(I)*WK3(I+NINC).le.WK3(I) ) then
1274: C                  IDET = IDET + 1
1275: C                  LSCLP(N1+IDET) = IP30
1276: C                  EEE(IP30) = WK4(I)
1277: C                  UOUT(N1+IDET) = UOUT(N2+I)
1278: C
1279: C                  WK21(N1+IDET) = EPIN
1280: C              else
1281: C                  NN = NN + 1
1282: C                  IP3(NN) = IP30
1283: C                  DWK2(NN) = EPIN
1284: C                  IWK3(NN) = IWK3(I)
1285: C              end if
1286: C          380      continue
1287: C      else
1288: C          *VOCL LOOP,NOVREC
1289: C          do 390 I = 1, NINC
1290: C              CM1992-7-7      IF(R(I)*WK3(I+NINC).LT.WK3(I)) THEN
1291: C                  IP30 = IP3(I)
1292: C                  if ( R(I)*WK3(I+NINC).le.WK3(I) ) then

```

src/mvp/photr.f

```

1293:      IDET      = IDET + 1
1294:      LSCLP(N1+IDET) = IP3O
1295:      EEE(IP3O)      = WK4(I)
1296:      UOUT(N1+IDET)   = UOUT(N2+I)
1297:      else
1298:      NN          = NN + 1
1299:      IP3(NN)     = IP3O
1300:      DWK2(NN)    = DWK2(I)
1301:      IWK3(NN)    = IWK3(I)
1302:      end if
1303: 390      continue
1304:      end if
1305:      NINC      = NN
1306: C
1307:      go to 350
1308:      end if
1309:      NINC      = IDET
1310:      N1        = N1 + NINC
1311:      end if
1312: C
1313: C ( IP3,DWK2,WK3,WK4,IWK6,IWK7,IWK10,IWK11 ARE FREE TO USE HEREAFTER )
1314: C
1315: C -----
1316: C TREATMENT OF INCOHERENT SCATTERING ( E < 1.5 MEV )
1317: C
1318: C SAMPLING WITH REJECTION LOOP.
1319: C
1320: C UOUT(NPPR+NINC+1:NCOLP) : COSINES OF OUTGOING DIRECTIONS
1321: C IWK5(1:NINC2)           : COLLIDING ATOM
1322: C IP5(1:NINC2)           : BANK POINTER
1323: C DWK1(1:NINC2)          : ENERGY
1324: C IWK6(1:NINC2)          : MEMORY OF INITIAL POSITION IN WORKING ARRAY
1325: C IWK5, IP5, DWK1
1326: C -----
1327: C
1328: C
1329: C if ( NINC2.gt.0 ) then
1330: C
1331: C
1332: C ..... DWK1 --> ALPHA .....
1333: C
1334: C do 400 I = 1, NINC2
1335: C   DWK1(I) = DWK1(I) /DMOC2
1336: C 400   continue
1337: C c##<2007/03/14:PN3:
1338: C   if ( KPNPRD.eq.0 ) then
1339: C     if ( KIBNK(19).ne.0 ) then
1340: C *VOCL LOOP,NOVREC
1341: C   do I = 1, NINC2
1342: C     IBNK(IP5(I),KIBNK(19)) = NMT + 4 ! incoherent scatterin
g (MT=504)
1343: C   end do
1344: C   end if
1345: C   end if
1346: C c##>
1347: C
1348: C ..... RETURNING POINT OF SCATTERING FUNCTION REJECTION LOOP ...
1349: C
1350: C 410   continue
1351: C
1352: C   NDET      = 0
1353: C   do 420 I = 1, NINC2
1354: C     IWK6(I) = I
1355: C   420   continue
1356: C

```

```

1357: C ..... RETURNING POINT OF KLEIN-NISHINA SAMPLING LOOP .....
1358: C
1359: C 430   NN      = NINC2 - NDET
1360: C
1361: C   if ( NN.gt.0 ) then
1362: C     call RANU2( IRAND, R, 3*NN, ICON )
1363: C     NN2      = 0
1364: C
1365: C .... SAMPLE: X = OUTGOING-ENERGY/INCIDENT-ENERGY .....
1366: C
1367: C *VOCL LOOP,NOVREC
1368: C   do 440 I = 1, NN
1369: C     CM      A      = DWK1(IWK6(I))
1370: C     IWK6O    = IWK6(I)
1371: C c##<2007/03/14:PN4:
1372: C c##      A      = DWK1(IWK6O)
1373: C c##      A12     = 1.0 + 2.0*A
1374: C c##      ASQ     = A*A
1375: C c##      A1      = DWK1(IWK6O)
1376: C c##      A12     = 1.0 + 2.0*A1
1377: C c##      ASQ     = A1*A1
1378: C c##>
1379: C
1380: C
1381: C   if ( 2.0*R(I+NN)*(ASQ+A12).lt.A12+2.0*ASQ*R(I) ) then
1382: C c##<2007/03/14:PN4:
1383: C c##      X      = 1.0 - 2.0*A*R(I) /A12
1384: C c##      else
1385: C c##      X      = (1.0+2.0*A*R(I)) /A12
1386: C c##      X      = 1.0 - 2.0*A1*R(I) /A12
1387: C c##      else
1388: C c##      X      = (1.0+2.0*A1*R(I)) /A12
1389: C c##>
1390: C   end if
1391: C
1392: C .... IF H(X)/( 2/GMIN ) > R(I+2*NN) THEN ACCEPT
1393: C
1394: C c##<2007/03/14:PN4:
1395: C c##   if ( 2.0*ASQ*(1.0+A*(1.0
1396: C   if ( 2.0*ASQ*(1.0+A1*(1.0
1397: C c##>
1398: C   &      -X))*R(I+2*NN)*X.X.lt.1.0
1399: C   &      +X*(ASQ-A12-1.0+X*(A12+X*ASQ)) ) then
1400: C
1401: C .... ACCEPTED: SAVE A,1/X,MU TO CALCULATE V(Q,Z) & VMAX(Q,Z)
1402: C
1403: C   NDET      = NDET + 1
1404: C C950717 ----- WK3(NDET)      = X
1405: C   DWK2(NDET) = X
1406: C CM      IWK11(NDET) = IWK6(I)
1407: C   IWK11(NDET) = IWK6O
1408: C   else
1409: C
1410: C .... REJECTED: COMPRESS PARAMETERS FOR THE NEXT TRIAL OF SAMPLING.
1411: C
1412: C   NN2      = NN2 + 1
1413: C CM      IWK6(NN2)      = IWK6(I)
1414: C   IWK6(NN2) = IWK6O
1415: C   end if
1416: C 440   continue
1417: C
1418: C   go to 430
1419: C   end if
1420: C
1421: C ..... SAMPLED QUANTITIES IN THE ABOVE LOOP (KLEIN - NISHINA)

```

src/mvp/photr.f

```

1422: C
1423: C      WK3(1:NDET) :   X      = A'/A
1424: C      IWK11(I) :   IWK6(I) IN ORDER OF ACCEPTION
1425: C
1426: C
1427: C      ..... V(Q,Z) ETC. ....
1428: C
1429: C      do 450 I = 1, NINC2
1430: C##<2007/03/14:PN4:
1431: C##      A      = DWK1(IWK11(I))
1432: C##      A1     = DWK1(IWK11(I))
1433: C##>
1434: C950717 ----- X      = WK3(I)
1435: C      X      = DWK2(I)
1436: C      XI     = 1.0/X
1437: C##<2007/03/14:PN4:
1438: C##      XMU   = 1.0 + (1.0-XI) /A
1439: C##      XMU   = 1.0 + (1.0-XI) /A1
1440: C##>
1441: C      if ( ABS(XMU).eq.1.0 ) XMU = XMU*ONE
1442: C
1443: C      ..... V(Q,Z)
1444: C##<2007/03/14:PN4:
1445: C##      WK3(I) = XK0*(A*SQR(1.+X*(X-2.*XMU)))
1446: C##      WK3(I) = XK0*(A1*SQR(1.+X*(X-2.*XMU)))
1447: C##>
1448: C
1449: C      ..... VMAX(Q,Z)
1450: C##<2007/03/14:PN4:
1451: C##      WK3(I+NINC2) = XK0*A*(1.+1./(1.+2.*A))
1452: C##      WK3(I+NINC2) = XK0*A1*(1.+1./(1.+2.*A1))
1453: C##>
1454: C
1455: C      ..... ENERGY & COSINE ...
1456: C
1457: C##<2007/03/14:PN4:
1458: C##      WK4(I) = A*X*DMOC2
1459: C##      WK4(I) = A1*X*DMOC2
1460: C##>
1461: C      WK4(I+NINC2) = XMU
1462: C
1463: C      IWK6(I) = KLBP1(IWK5(IWK11(I)),6)
1464: C      IWK6(NINC2+I) = IWK6(I)
1465: C      IWK7(I) = KLBP1(IWK5(IWK11(I)),9)
1466: C      IWK7(NINC2+I) = IWK7(I)
1467: C      450 continue
1468: C
1469: C      call BSINC3( CXP, IWK6, R, IWK7, WK3, IWK10, NINC2*2 )
1470: C
1471: C      do 460 II = 1, 2*NINC2
1472: C      LS      = IWK7(II) + IWK10(II)
1473: C      TI      = (CXP(LS)-WK3(II)) /(CXP(LS)-CXP(LS-1))
1474: C      if ( TI.ge.0.0 ) then
1475: C      LS1     = LS + IWK6(II)
1476: C      WK3(II) = CXP(LS1) + TI*(CXP(LS1-1)-CXP(LS1))
1477: C      else
1478: C      WK3(II) = 0.0
1479: C      end if
1480: C      460 continue
1481: C
1482: C      call RANU2( IRAND, R, NINC2, ICON )
1483: C      NNN      = 0
1484: C      NOK      = 0
1485: C
1486: C      if ( JBREM.eq.1 ) then

```

```

1487: *VOCL LOOP,NOVREC
1488: do 470 I = 1, NINC2
1489: IP      = IP5(IWK11(I))
1490: if ( R(I)*WK3(I+NINC2).le.WK3(I) ) then
1491: NOK     = NOK + 1
1492: C
1493: WK21(N1+NOK) = DWK1(IWK11(I))*DMOC2
1494: C
1495: IWK6(IWK11(I)) = 0
1496: EEE(IP) = WK4(I)
1497: UOUT(N1+NOK) = WK4(I+NINC2)
1498: LSCLP(N1+NOK) = IP
1499: else
1500: IWK6(IWK11(I)) = 1
1501: end if
1502: 470 continue
1503: else
1504: *VOCL LOOP,NOVREC
1505: do 480 I = 1, NINC2
1506: IP      = IP5(IWK11(I))
1507: if ( R(I)*WK3(I+NINC2).le.WK3(I) ) then
1508: IWK6(IWK11(I)) = 0
1509: EEE(IP) = WK4(I)
1510: NOK     = NOK + 1
1511: UOUT(N1+NOK) = WK4(I+NINC2)
1512: C
1513: C      ..... MERGE BANK INDEXES TO ARRAY LSCLP OF E > 1.5MEV CASE
1514: C
1515: LSCLP(N1+NOK) = IP
1516: else
1517: IWK6(IWK11(I)) = 1
1518: end if
1519: 480 continue
1520: end if
1521: C
1522: if ( NOK.lt.NINC2 ) then
1523: *VOCL LOOP,NOVREC
1524: do 490 I = 1, NINC2
1525: if ( IWK6(I).eq.1 ) then
1526: NNN      = NNN + 1
1527: DWK1(NNN) = DWK1(I)
1528: IWK5(NNN) = IWK5(I)
1529: IP5(NNN) = IP5(I)
1530: end if
1531: 490 continue
1532: end if
1533: C
1534: NINC2     = NNN
1535: NINC      = NINC + NOK
1536: N1        = N1 + NOK
1537: C
1538: if ( NINC2.gt.0 ) go to 410
1539: C
1540: end if
1541: C
1542: if ( JBREM.eq.1 ) N3      = N1
1543: C
1544: C-----
1545: C.... COHERENT SCATTERING
1546: C-----
1547: C
1548: C
1549: C
1550: NCOH      = 0
1551: NN        = 0

```

src/mvp/photr.f

```

1552:      call RANU2( IRAND, R, NCOLP, ICON )
1553: *VOCL LOOP,NOVREC
1554:      do 500 I = 1, NCOLP
1555:          IP      = IP4(I)
1556:          TOTS     = SMIC(IP,ICLA(I),2) + SMIC(IP,ICLA(I),5)
1557: C
1558: CM1993-10-12
1559:          ICLAI    = ICLA(I)
1560:          EINI     = EIN(I)
1561: CM          IF( R(I)*TOTS.LT. SMIC(IP,ICLA(I),2)) THEN
1562:          if ( R(I)*TOTS.lt.SMIC(IP,ICLAI,2) ) then
1563:              NCOH  = NCOH + 1
1564:              IP2(NCOH) = IP
1565: CM              IWK3(NCOH) = ICLAI(I)
1566:              IWK3(NCOH) = ICLAI
1567: CM              DWK2(NCOH) = EIN(I)
1568:              DWK2(NCOH) = EINI
1569:          else
1570:              NN     = NN + 1
1571:              IP5(NN) = IP
1572: CM              ICLA(NN) = ICLA(I)
1573:              ICLA(NN) = ICLAI
1574: CM              EIN(NN) = EIN(I)
1575:              EIN(NN) = EINI
1576:          end if
1577:      500      continue
1578:          NCOLP = NN
1579: C
1580: C-----
1581: C      TREATMENT OF COHERENT SCATTERING
1582: C
1583: C      UOUT(N1+1:N1+NCOH) : COSINES OF OUTGOING DIRECTIONS
1584: C      IWK3(1:NCOH)       : COLLIDING ATOM
1585: C      IP2(1:NCOH)        : BANK POINTER
1586: C      DWK2(1:NINC2)      : INCIDENT ENERGY
1587: C-----
1588: C
1589: C
1590: C      if ( NCOH.gt.0 ) then
1591: C
1592: C
1593: C      ..... SELECT FORM FACTOR TABLE
1594: C
1595: C
1596: C      ..... NFFE ..... NFFE IS ALWAYS 1 .....
1597: C
1598: C      NFFE = 1
1599: *VOCL LOOP,NOVREC
1600:      do 510 I = 1, NCOH
1601:          LS      = KLBP1(IWK3(I),10)
1602: C
1603: C      ..... NFF .....
1604: C      IWK6(I) = KLBP1(IWK3(I),7)
1605: C
1606: C      ..... Y-TABLE ADDRESS ....
1607: C      IWK7(I) = LS + NFFE + IWK6(I)
1608: C
1609: C      ..... A-TABLE ADDRESS ....
1610: C      IWK7(I+NBANK) = IWK7(I) + IWK6(I)
1611: C
1612: C      ..... YMAX .....
1613: C      WK3(I) = (XK0*DWK2(I)/DMOC2)**2*4.
1614:      510      continue
1615: C
1616: C      ..... Y-TABLE POSITION OF YMAX .... : IWK10(1:NCOH)

```

```

1617: C
1618: C      T      NT      MW      LS      X      IT      N1
1619:      call BSINC3( CXP, IWK6, IWK5, IWK7(1), WK3(1), IWK10(1),
1620:          &          NCOH )
1621: C
1622: C      ..... A-TABLE VALUE FOR YMAX
1623: C
1624: *VOCL LOOP,NOVREC
1625:      do 520 I = 1, NCOH
1626:          LY      = IWK7(I) + IWK10(I)
1627:          L1      = IWK7(I+NBANK) + IWK10(I)
1628:          TI      = (CXP(LY)-WK3(I)) / (CXP(LY)-CXP(LY-1))
1629:          if ( TI.lt.0. ) TI = 0.0
1630:          WK3(I+NBANK) = CXP(L1) + (CXP(L1-1)-CXP(L1))*TI
1631:          IWK10(I) = IWK10(I) + 1
1632:      520      continue
1633: C
1634: C##<2007/03/14:PN3:
1635:      if ( KPNPRD.eq.0 ) then
1636:          if ( KIBNK(19).ne.0 ) then
1637: *VOCL LOOP,NOVREC
1638:          do I = 1, NCOH
1639:              IBNK(IP2(I),KIBNK(19)) = NMT + 2 ! coherent scattering
1640:          end do
1641:          end if
1642:          end if
1643: C##>
1644: C
1645: C      ..... Y-SAMPLING & REJECTION .....
1646: C
1647: C
1648: C      IDET      = 0
1649: C      NCOH0     = NCOH
1650: C      NREJ      = 0
1651: C
1652:      530      continue
1653:      if ( IDET.lt.NCOH0 ) then
1654:          NREJ = NREJ + 1
1655:          NN   = 0
1656: C
1657: C      ..... SAMPLE Y FROM A-TABLE (LINEAR)
1658: C
1659: C      call RANU2( IRAND, R, 2*NCOH, ICON )
1660: C      do 540 I = 1, NCOH
1661: C          WK4(I) = WK3(I+NBANK)*R(I)
1662:      540      continue
1663: C
1664: C      T      NT      MW      LS
1665: C      X      IT      N1
1666:      call BSINC3( CXP, IWK10(1), IWK5, IWK7(NBANK+1), WK4(1),
1667:          &          IWK10(NBANK+1), NCOH )
1668: C
1669: C      ..... DETERMINE Y -> MU -> WK4(I) .....
1670: C
1671: *VOCL LOOP,NOVREC
1672:      do 550 I = 1, NCOH
1673:          LY      = IWK7(I) + IWK10(I+NBANK)
1674:          L1      = IWK7(I+NBANK) + IWK10(I+NBANK)
1675:          YY      = CXP(LY) + (CXP(LY-1)-CXP(LY))*
1676:          &          (CXP(L1)-WK4(I)) / (CXP(L1)-CXP(L1-1))
1677: C
1678: C      DMU      = 1.0 - YY/WK3(I)*2.0
1679: CM1992-7-31 NEXT STATEMENT IS NECESSARY BECAUSE OF THE FOLLOWING IF
1680: C      IP2O     = IP2(I)

```

src/mvp/photr.f

```

1681: C
1682:       if ( R(I+NCOH)*2.0.lt.1.0+DMU*DMU ) then
1683:         IDET = IDET + 1
1684:         LSCLP(N1+IDET) = IP20
1685: C     ****NO ENERGY CHANG EEE(IP20) =
1686:       UOUT(N1+IDET) = DMU
1687:       else
1688:         NN = NN + 1
1689: C     IP2(NN) = IP2(I)
1690:       IP2(NN) = IP20
1691:       WK3(NN) = WK3(I)
1692:       WK3(NBANK+NN) = WK3(NBANK+I)
1693:       IWK7(NN) = IWK7(I)
1694:       IWK7(NBANK+NN) = IWK7(NBANK+I)
1695:       IWK10(NN) = IWK10(I)
1696:       end if
1697: 550     continue
1698:       NCOH = NN
1699: C
1700:       go to 530
1701:     end if
1702:     NCOH = IDET
1703:     N1 = N1 + NCOH
1704:   end if
1705: C
1706: C ( WK4 IS FREE TO USE )
1707: C
1708: C-----
1709: C.... PHOTOELECTRIC REACTION .....
1710: C-----
1711: C
1712: C
1713:       NPHE = NCOLP
1714: C
1715: C-----
1716: C   UOUT(N1+1:N1+NPHE) : COSINES OF OUTGOING DIRECTIONS
1717: C   ICLA(1:NPHE)       : COLLIDING ATOM
1718: C   IP5(1:NPHE)        : BANK POINTER
1719: C   EIN(1:NPHE)        : INCIDENT ENERGY
1720: C-----
1721: C
1722: C
1723:       NQ = 0
1724: C
1725:       if ( NPHE.gt.0 ) then
1726: C
1727: C
1728:         NEDGE = 2
1729: C
1730:         if ( JBREM.eq.1 ) then
1731: C
1732:           call RANU2( IRAND, R, NPHE*4, ICON )
1733: C
1734: C*VOCL LOOP,NOVREC
1735:       do 560 I = 1, NPHE
1736:         LSP1 = KLBPl(ICLA(I),12) + 1
1737:         EDGE1 = CXP(LSP1)
1738:         EDGE2 = CXP(LSP1+1)
1739:         WK3(I) = 0.0
1740: C
1741:         MK = KZMAT(IZZ(IP5(I)))
1742:         EG = EEDG(MK)
1743:         ED = 0.0
1744:         EAUGER = 0.0
1745: C

```

```

1746: C     .... WITH A FLUORESCENCE OR AUGER ELECTRON ....
1747: C
1748:       if ( EIN(I).gt.EDGE2 ) then
1749: C     .... AUGER ELECTRON ....
1750:       if ( R(NPHE*3+I).gt.WWAG(MK).and.JAUGE.eq.1 ) then
1751:         EAUGER = EEEK(MK)
1752: C     .... FLUORESCENCE ....
1753:       else
1754:         ND1 = ICXP(LSP1+NEDGE)
1755:         ND2 = ICXP(LSP1+NEDGE+1)
1756:         if ( EIN(I).ge.EDGE1 ) then
1757:           ND = ND1
1758:           LS = LSP1 + 3*NEDGE + 1
1759:           LF = LS + 3*(ND1+ND2)
1760:           EG = EDGE1
1761:         else
1762:           ND = ND2
1763:           LS = LSP1 + 3*NEDGE + 1 + 3*ND1
1764:           LF = LS + 3*ND2
1765:           EG = EDGE2
1766:         end if
1767: C/#IF ROUNDOFF(NEAREST)
1768: Ccc           L = INT(ND*R(I))
1769:           L = MIN(INT(ND*R(I)),ND-1)
1770: C97/11/07     K = INT(1+R(I+NPHE)-CXP(LS+L))+ LS+ND+2*L
1771:           K = MIN(1,INT(1+R(I+NPHE)-CXP(LS
1772:             +L)))+ LS + ND + 2*L
1773: C/#ELSE
1774: *           L = ND*R(I)
1775: *           R1 = R(I+NPHE) - CXP(LS+L)
1776: *           K = LS + ND + 2*L + 1 + R1
1777: C/#ENDIF
1778:           WK3(I) = CXP(ICXP(K)+LF-1)
1779:         end if
1780:       end if
1781: C     .... PHOTOELECTRON ....
1782: C
1783: C     .... IF ENERGY > CUTOFF ENERGY, PHOTOELECTRON ....
1784: C
1785:       EPEEL = EIN(I) - EG - ED
1786: C
1787:       if ( EPEEL.gt.EBOTE .or. EAUGER.gt.EBOTE ) then
1788:         NQ = NQ + 1
1789:         WK20(N3+NQ) = WWW(IP5(I))
1790:         WK21(N3+NQ) = EIN(I)
1791:         WK22(N3+NQ) = EPEEL
1792:         WK23(N3+NQ) = EAUGER
1793:         IP50 = IP5(I)
1794:         if ( WK3(I).le.EBOTP ) IP50 = -IP50
1795:         IWK20(N3+NQ) = IP50
1796:       end if
1797: 560     continue
1798: C
1799: C     .... ORIGINAL MODE : NO BREMSSTRAHLUNG ....
1800:       else
1801: C
1802:       call RANU2( IRAND, R, 3*NPHE, ICON )
1803: C
1804:       do 570 I = 1, NPHE
1805:         LSP1 = KLBPl(ICLA(I),12) + 1
1806:         EDGE1 = CXP(LSP1)
1807:         EDGE2 = CXP(LSP1+1)
1808: C     .... OUTGOING ENERGY STORAGE ....
1809:         WK3(I) = 0.0
1810: C

```


src/mvp/photr.f

```

1811:      if ( EIN(I).gt.EDGE2 ) then
1812:        ND1 = ICXP(LSP1+NEDGE)
1813:        ND2 = ICXP(LSP1+NEDGE+1)
1814:        if ( EIN(I).ge.EDGE1 ) then
1815:          ND = ND1
1816:          LS = LSP1 + 3*NEDGE + 1
1817:          LF = LS + 3*(ND1+ND2)
1818:        else
1819:          ND = ND2
1820:          LS = LSP1 + 3*NEDGE + 1 + 3*ND1
1821:          LF = LS + 3*ND2
1822:        end if
1823: C
1824: C/#IF ROUND OFF(NEAREST)
1825: Cccc      L = INT(ND*R(I))
1826:      L = MIN(INT(ND*R(I)),ND-1)
1827: C97/11/07      K = INT((1+R(I+NPHE)-CXP(LS+L))+ LS+ND+2*L)
1828:      K = MIN(1,INT((1+R(I+NPHE)-CXP(LS+L)
1829:      &      +L))) + LS + ND + 2*L
1830: C/#ELSE
1831:      *      L = ND*R(I)
1832:      *      R1 = R(I+NPHE) - CXP(LS+L)
1833:      *      K = LS + ND + 2*L + 1 + R1
1834: C/#ENDIF
1835:
1836:      WK3(I) = CXP(ICXP(K)+LF-1)
1837:    end if
1838: 570      continue
1839:    end if
1840: C
1841:      NN = 0
1842:      *VOCL LOOP,NOVREC
1843:      do 580 I = 1, NPHE
1844:        IP = IP5(I)
1845:        WCNTR(23,KPPHOT) = WCNTR(23,KPPHOT) + WWW(IP)
1846:        if ( WK3(I).gt.0.0 ) then
1847:          NN = NN + 1
1848:          CM      IP5(NN) = IP5(I)
1849:          CM      EEE(IP5(I)) = WK3(I)
1850:          LSCLP(N1+NN) = IP
1851:          EEE(IP) = WK3(I)
1852:          UOUT(N1+NN) = 2.0*R(I+2*NPHE) - 1.0
1853:          WCNTR(19,KPPHOT) = WCNTR(19,KPPHOT) + WWW(IP)
1854: c##<2007/03/14:PN3:
1855:          if ( KPNPRD.eq.0 ) then
1856:            if ( KIBNK(19).ne.0 ) IBNK(IP,KIBNK(19)) = NMT + 8 ! phot
oelectric (MT=522)
1857:          end if
1858: c##>
1859:          else
1860:            NDEAD = NDEAD + 1
1861:            LSDDED(NDEAD) = IP
1862: c##<2007/03/14:PN3:
1863:            if ( KIBNK(17).ne.0 ) IBNK(IP,KIBNK(17)) = 0
1864:            if ( KIBNK(18).ne.0 ) IBNK(IP,KIBNK(18)) = 0
1865:            if ( KIBNK(19).ne.0 ) IBNK(IP,KIBNK(19)) = 0
1866: c##>
1867: c##<2007/03/14:PN4:
1868:            if ( JPHNU.ne.0 ) KPNFG(IP) = 0
1869: c##>
1870:          end if
1871: 580      continue
1872:      NCNTR(23,KPPHOT) = NCNTR(23,KPPHOT) + NPHE
1873:      NCNTR(19,KPPHOT) = NCNTR(19,KPPHOT) + NN
1874: CM

```

```

1875:      NPHE = NN
1876:    end if
1877: C
1878:      if ( JBREM.eq.1 ) then
1879:        IMTTE0(1) = NPPR
1880:        IMTTE0(2) = NINC
1881:        IMTTE0(3) = NQ
1882:        if ( JAUGE.eq.0 ) then
1883:          IMTTE0(4) = 0
1884:        else
1885:          IMTTE0(4) = NQ
1886:        end if
1887:        IMTTE = NPPR + NINC + NQ
1888: C      .... ADD ELECTRON FROM INCOHERENT SCATTERING
1889:      do 590 I = NPPR + 1, NPPR + NINC
1890:        IP = LSCLP(I)
1891:        WK20(I) = WWW(IP)
1892:        WK22(I) = EEE(IP)
1893:        WK4(I) = UOUT(I)
1894:        if ( WK22(I).gt.EBOTP ) then
1895:          IWK20(I) = IP
1896:        else
1897:          IWK20(I) = -IP
1898:        end if
1899: 590      continue
1900:    end if
1901: C
1902:      NCOLP = NPPR + NINC + NCOH + NPHE
1903:    end if
1904: C
1905: C=====
1906: C
1907: C ---- OUTGOING ENERGY & ANGLE (COSINE) DETERMINED ]]] ----
1908: C
1909: C-----
1910: C-----
1911: C      CHECK ENERGY CUTOFF
1912: C-----
1913: C
1914:      NN = 0
1915:      *VOCL LOOP,NOVREC
1916:      do 600 I = 1, NCOLP
1917:        CM1993-10-19
1918:        LSCLPI = LSCLP(I)
1919:        if ( EEE(LSCLPI).gt.EBOTP ) then
1920:          NN = NN + 1
1921:          UOUT(NN) = UOUT(I)
1922:          CM      LSCLP(NN) = LSCLP(I)
1923:          LSCLP(NN) = LSCLPI
1924:        else
1925:          NDEAD = NDEAD + 1
1926:          CM      LSDDED(NDEAD) = LSCLP(I)
1927:          LSDDED(NDEAD) = LSCLPI
1928:          CM      WCNTR(8,2) = WCNTR(8,2) + WWW(LSCLP(I))
1929:          WCNTR(8,KPPHOT) = WCNTR(8,KPPHOT) + WWW(LSCLPI)
1930: c##<2007/03/14:PN3:
1931:          if ( KIBNK(17).ne.0 ) IBNK(LSCLPI,KIBNK(17)) = 0
1932:          if ( KIBNK(18).ne.0 ) IBNK(LSCLPI,KIBNK(18)) = 0
1933:          if ( KIBNK(19).ne.0 ) IBNK(LSCLPI,KIBNK(19)) = 0
1934: c##>
1935: c##<2007/03/14:PN4:
1936:          if ( JPHNU.ne.0 ) KPNFG(LSCLPI) = 0
1937: c##>
1938:        end if
1939: 600      continue

```

src/mvp/photr.f

```

1940:      NCNTR(8,KPPHOT) = NCNTR(8,KPPHOT) + NCOLP - NN
1941:      NCOLP = NN
1942: C
1943: C
1944: C-----
1945: C      GENERATION OF ELECTRONS BY PHOTON COLLISION.
1946: C      GENERATION OF BREMSSTRAHLUNG PHOTONS FROM ELECTRONS.
1947: C-----
1948: C
1949: C      if ( JBREM.eq.1.and.IMTTE.gt.0 ) then
1950: C
1951: C
1952: C      call ELGEN( IOW, JTIME, JIMPT, JLAT, JHLAT, JTLT, IRAND,
1953: C      &          NBANK, NEST, EBOTE, NELC, XXX, YYY, ZZZ, AAA, BBB, CCC,
1954: C      &          WWW, EEE, TTT, XIM, KLSF, IZZ, IGG, LEVL, LZZ, LPOS,
1955: C      &          LCRS, IBREG, IBSPC, R, IMTTE, IMTTE0, IWK20, WK20,
1956: C      &          WK21, WK22, WK23, WK4, DWK1, DWK2, WK3, IWK1, IWK2,
1957: C      &          IWK7, IWK6, IWK3, WK41, WK42, WK43 )
1958: C
1959: C      if ( NELC.gt.0 ) then
1960: C      call TTR( IOW, JTIME, JIMPT, JLAT, JHLAT, JTLT, IRAND,
1961: C      &          NZONE, NEST, NMAT, NBANK, NEVENT, NCNTR, WCNTR,
1962: C      &          KZMAT, EBOTP, NEE, MTOP, EEL, RRT, PBT, EBT, EBTP,
1963: C      &          IEETP, LSCLP, NCOLP, NELC, LSDED, NDEAD, ND0, XXX,
1964: C      &          YYY, ZZZ, AAA, BBB, CCC, WWW, EEE, TTT, ITT, XIM,
1965: C      &          KLSF, KRP, IZZ, IGG, LEVL, LZZ, LPOS, LCRS, IBREG,
1966: C      &          IBSPC, NKLP, R, IMTTE0, UOUT, DWK1, DWK2, WK3,
1967: C      &          IWK6, WK4, IWK7, IWK1, IWK2, IWK10, IP2, IP4, IWK3,
1968: C      &          IWK4, IWK5, IWK11, WK41, WK42, WK43 )
1969: C
1970: C      (NOTE: IP2 MEANS IP2+IP3, IP4 MEANS IP4+IP5.)
1971: C      (      IWK11 STORES PARENT PHOTON INDEX FOR )
1972: C      (      DIRECTION OF COMPTON SCATTERING. )
1973: C
1974: C
1975: C      else
1976: C      do 610 I = 1, 4
1977: C      IMTTE0(I) = 0
1978: C      continue
1979: C      end if
1980: C
1981: C-----
1982: C      else
1983: C      do 620 I = 1, 4
1984: C      IMTTE0(I) = 0
1985: C      continue
1986: C      end if
1987: C-----
1988: C
1989: C      if ( NCOLP.gt.0 ) then
1990: C
1991: C      if ( IMTTE0(1).gt.0 ) then
1992: C      NCOLPP = IMTTE0(1) - 1
1993: C      else if ( IMTTE0(3).gt.0 ) then
1994: C      NCOLPP = IMTTE0(3) - 1
1995: C      else
1996: C      NCOLPP = NCOLP
1997: C      end if
1998: C
1999: C-----
2000: C      CALCULATION OF SCATTERING ANGLE
2001: C-----
2002: C
2003: C      call RANU2( IRAND, R, NCOLPP, ICON )
2004: C

```

```

2005: *VOCL LOOP,NOVREC
2006:      do 630 I = 1, NCOLPP
2007:      IP = LSCLP(I)
2008:      DA0 = AAA(IP)
2009:      DB0 = BBB(IP)
2010:      DC0 = CCC(IP)
2011: C
2012: C      ETA = PI2*R(I)
2013: C      SINETA = SIN(ETA)
2014: C      COSETA = COS(ETA)
2015: C      STHETA = 1. - DA0*DA0
2016: C      *VOCL STMT,IF(100)
2017: C      if ( STHETA.gt.0.0 ) then
2018: C      STHETA = SQRT(STHETA)
2019: C      COSPHI = DB0/STHETA
2020: C      SINPHI = DC0/STHETA
2021: C      else
2022: C      STHETA = 0.0
2023: C      COSPHI = 1.0
2024: C      SINPHI = 0.0
2025: C      end if
2026: C      Csasa Jun 1995
2027: C      SINPSI = SQRT( abs(1. - UOUT(I)*UOUT(I)) )
2028: C      SINPSI = SQRT(1.-UOUT(I)*UOUT(I))
2029: C
2030: C      DB0 = DB0*UOUT(I) + (DA0*COSPHI*COSETA
2031: C      &      -SINPHI*SINETA)*SINPSI
2032: C
2033: C      DC0 = DC0*UOUT(I) + (DA0*SINPHI*COSETA
2034: C      &      +COSPHI*SINETA)*SINPSI
2035: C
2036: C      DA0 = DA0*UOUT(I) - COSETA*SINPSI*STHETA
2037: C
2038: C      S = 1./SQRT(DA0*DA0+DB0*DB0+DC0*DC0)
2039: C      AAA(IP) = DA0*S
2040: C      BBB(IP) = DB0*S
2041: C      CCC(IP) = DC0*S
2042: C      630 continue
2043: C
2044: C      .... CALCULATION OF DIRECTION OF BREMSSTRAHLUNG PHOTONS
2045: C      FROM COMPTON SCATTERING ....
2046: C
2047: C      if ( IMTTE0(3).gt.0 ) then
2048: C      J6 = 0
2049: C      NQ = IMTTE0(4) - IMTTE0(3) + 1
2050: C      call RANU2( IRAND, R, NQ, ICON )
2051: C
2052: *VOCL LOOP,NOVREC
2053:      do 640 I = IMTTE0(3), IMTTE0(4)
2054:      J6 = J6 + 1
2055:      IP = LSCLP(I)
2056:      DA0 = AAA(IP)
2057:      DB0 = BBB(IP)
2058:      DC0 = CCC(IP)
2059:      IB = IWK11(J6)
2060: C
2061: C      PARENT PHOTON ALIVES
2062: C      if ( IB.gt.0 ) then
2063: C      DA0 = WK41(J6)*DA0 + WK42(J6)*AAA(IB)
2064: C      DB0 = WK41(J6)*DB0 + WK42(J6)*BBB(IB)
2065: C      DC0 = WK41(J6)*DC0 + WK42(J6)*CCC(IB)
2066: C      S = 1.0/SQRT(DA0*DA0+DB0*DB0+DC0*DC0)
2067: C      AAA(IP) = DA0*S
2068: C      BBB(IP) = DB0*S
2069: C      CCC(IP) = DC0*S
2070: C
2071: C      PARENT PHOTON DIES

```

src/mvp/photr.f

```

2070:      else
2071:        ETA = PI2*R(J6)
2072:        SINETA = SIN(ETA)
2073:        COSETA = COS(ETA)
2074:        STHETA = 1.0 - DA0*DA0
2075:        if ( STHETA.gt.0.0 ) then
2076:          STHETA = SQRT(STHETA)
2077:          COSPHI = DB0/STHETA
2078:          SINPHI = DC0/STHETA
2079:        else
2080:          STHETA = 0.0
2081:          COSPHI = 1.0
2082:          SINPHI = 0.0
2083:        end if
2084:      Csasa SINPSI = SQRT( 1.0 - WK43(J6) * WK43(J6) )
2085:      SINPSI = SQRT(1.0-WK43(J6)*WK43(J6))
2086:      C SINPSI = SQRT( abs(1.0 - WK43(J6) * WK43(J6)) )
2087:      DB1 = DB0*WK43(J6)
2088:      & + (DA0*COSPHI*COSETA-SINPHI*SINETA)*SINPSI
2089:      DC1 = DC0*WK43(J6)
2090:      & + (DA0*SINPHI*COSETA+COSPHI*SINETA)*SINPSI
2091:      DA1 = DA0*WK43(J6) - COSETA*SINPSI*STHETA
2092:      S = 1.0/SQRT(DA1*DA1+DB1*DB1+DC1*DC1)
2093:      DA1 = DA1*S
2094:      DB1 = DB1*S
2095:      DC1 = DC1*S
2096:      DA0 = WK41(J6)*DA0 + WK42(J6)*DA1
2097:      DB0 = WK41(J6)*DB0 + WK42(J6)*DB1
2098:      DC0 = WK41(J6)*DC0 + WK42(J6)*DC1
2099:      S = 1.0/SQRT(DA0*DA0+DB0*DB0+DC0*DC0)
2100:      AAA(IP) = DA0*S
2101:      BBB(IP) = DB0*S
2102:      CCC(IP) = DC0*S
2103:    end if
2104:    640 continue
2105:  end if
2106: C
2107: C
2108: C
2109: C-----
2110: C DETERMINATION OF PHOTON ENERGY GROUP
2111: C-----
2112: C
2113: C.... WK3 : OUTGOING ENERGY VALUES --> IWK1 : ENERGY TABLE POSITION
2114: C
2115:      do 650 I = 1, NCOLP
2116:        WK3(I) = EEE(LSCLP(I))
2117:      650 continue
2118: C
2119:      call BSVDEC( ENGYB(KENGP(KPPHOT)), NGP(KPPHOT)+1, WK3, IWK1,
2120:        & NCOLP )
2121: C
2122: C.... MODIFY IGG ARRAY
2123: C
2124:      NGG = KNGP(KPPHOT) - 1
2125:      do 660 I = 1, NCOLP
2126:      check %%%%%%%%%%
2127:      c      if ( IWK1(I).le.0.or.IWK1(I).gt.NGP(KPPHOT) ) then
2128:      c        write(*,*) '%% photr iwk1 ',iwk1(i),' wk3 ',wk3(i)
2129:      c        write(*,*) '%% etopp ebtop ',etopp, ebtop
2130:      c        write(*,*) '%% ENGYBa ',(ENGYB(k),k=1,KENGP(KPLIM+1)-1)
2131:      c        write(*,*) '%% ENGYB ',
2132:      c      & (ENGYB(KENGP(KPPHOT)+j),j=0,NGP(KPPHOT))
2133:      c      end if
2134:      c %%%%%%%%%%

```

```

2135:      IGG(LSCLP(I)) = IWK1(I) + NGG
2136:      660 continue
2137: C
2138: C
2139: C-----
2140: C GET NEW MICROSCOPIC CROSS SECTION
2141: C-----
2142: C
2143: C
2144:      if ( JAMXC.eq.0 ) then
2145: C
2146:        if ( JVMNT.ne.0 ) call VMNT00( 8, 12 )
2147:        if ( JVMNT.ne.0 ) call VMNTR1( 8, NCOLP )
2148: C
2149:        call GTMICP( JGAMM, NCOLP, LSCLP, NUC, NMTP, NPATOM, NBANK,
2150:          & NSMIC, EEE, SMIC, KSPI, CXP, ICXP, MCXP, KLBPl,
2151:          & KLBp2, XLBP2, DWK1, IWK1, DWK2, IWK2 )
2152: C
2153: C      & SGTAL, CRES, KCRES, NSTAL, MCRES,
2154: C      c##<2007/03/14:PN3:
2155:        if ( JPHNU.ne.0 ) then
2156:          call GTMICPN( NCOLP, LSCLP, NUC, NUCPN, NUCPNI, NMTPN,
2157:            & NBANK, NSMICPN, NUC_MAX, EEE, SMICPN, CXP,
2158:            & ICXPN, MCXPN, KLBPN1, KLBPN2, XLBP1, EBOTLPN,
2159:            & DWK1, DWK2, IWK1, IWK2, WK3, R, RNU )
2160:        end if
2161: C      c##>
2162:        if ( JVMNT.ne.0 ) call VMNT22( 8, 12 )
2163: C
2164:      else
2165: C
2166: C      ... KSPI is used to judge that micro for the particle has
2167: C      been calculated or not
2168: C
2169:      do 680 N = 1, NUC
2170:        do 670 I = 1, NCOLP
2171:          KSPI(LSCLP(I),N) = 0
2172:        670 continue
2173:      680 continue
2174:      if ( NNCST.gt.0 ) then
2175:        if ( JVMNT.ne.0 ) call VMNT00( 8, 12 )
2176:        if ( JVMNT.ne.0 ) call VMNTR1( 8, NCOLP )
2177: C
2178:        do 690 I = 1, NCOLP
2179:          WK3(I) = LOG(EEE(LSCLP(I)))
2180: C      c##<2007/03/14:PN3:
2181:          WK4(I) = EEE(LSCLP(I))
2182: C      c##>
2183:        690 continue
2184: C
2185:        do 700 NC = 1, NNCST
2186:          IN = INCST(NC,2)
2187:          if ( IN.ne.0 ) then
2188:            call GMICP1( IN, NCOLP, LSCLP, WK3, JGAMM, NUC,
2189:              & NMTP, NPATOM, NBANK, NSMIC, SMIC, KSPI,
2190:              & CXP, ICXP, MCXP, KLBPl, KLBp2, XLBP2, IWK1,
2191:              & DWK2, IWK2 )
2192:          end if
2193:        700 continue
2194: C      c##<2007/03/14:PN3:
2195:        if ( JPHNU.ne.0 ) then
2196:          do NC = 1, NUCPN
2197:            call GMICPN1( NC, NCOLP, LSCLP, WK4, NUC, NUCPN,
2198:              & NUCPNI, NMTPN, NBANK, NUC_MAX, NSMICPN,
2199:              & SMICPN, CXP, ICXPN, MCXPN, KLBPN1, KLBPN2,

```

src/mvp/photr.f

```

2200:      &                XLBPN1, EBOTLPN, DWK1, IWK1, IWK2, DWK2, R,
2201:      &                RNU )
2202:      end do
2203:      end if
2204: c##>
2205:
2206:      if ( JVMNT.ne.0 ) call VMNT22( 8, 12 )
2207:      end if
2208:      end if
2209:
2210:      if ( NSTAL.gt.0 ) then
2211:      JNP      = 2
2212:      call GTSGTL( JNP, NCOLP, LSCLP, NBANK, EEE,
2213:      &          SGTAL, CRES, KCRES, NSTAL, MCRES, DWK1, IWK1 )
2214:      end if
2215: C
2216: C
2217: C-----
2218: C  TRANSFER PARTICLES TO FREE-FLIGHT STACK
2219: C-----
2220: C
2221: C
2222: C**** SET MMAC( ,2)=0 : NUMBER OF MACROSCOPIC CROSS SECTIONS PREPARED
2223: C          FOR THIS PARTICLE
2224: C
2225:      do 710 I = 1, NCOLP
2226:      MMAC(LSCLP(I),2) = 0
2227: 710      continue
2228: C
2229: C .... SEND PARTICLES TO FREE-FLIGHT STACK ....
2230: C
2231:      NN      = NFFL(NZONE+1)
2232:      do 720 I = 1, NCOLP
2233:      IP      = LSCLP(I)
2234:      LSFFL(NN+I) = IP
2235:      IZFFL(NN+I) = IZZ(IP)
2236: 720      continue
2237:      if ( KIBNK(5).ne.0 ) then
2238:      do 730 I = 1, NCOLP
2239:      IP      = LSCLP(I)
2240:      IBNK(IP,KIBNK(5)) = 0
2241: 730      continue
2242:      end if
2243: *VOCL LOOP, SCALAR
2244:      do 740 I = 1, NCOLP
2245:      NFFL(IZFFL(NN+I)) = NFFL(IZFFL(NN+I)) + 1
2246: 740      continue
2247:      NFFL(NZONE+1) = NN + NCOLP
2248: C
2249: C-----
2250: C  RUSSIAN ROULETTE KILL & SPLITTING BASED ON IMPORTANCE, WEIGHT
2251: C-----
2252: C
2253:      if ( JRRLT+JIMPT+JWWND.ne.0 ) then
2254:      ISF0      = NN + 1
2255: c##<2007/03/14:PN3:
2256: c##          JPHT      = 1
2257:          JJCOL      = 1
2258: c##>
2259: C
2260: c##<2007/03/14:PN3:
2261: c##          call SPLITB( 1, KPPHOT, ISF0, NCOLP, IRAND, NPKIND, NBANK,
2262:          call SPLITB( JJCOL, KPPHOT, ISF0, NCOLP, IRAND, NPKIND,
2263:          &          NBANK,
2264: c##>

```

```

2265:      &          NZONE, NGROUP, NREG, NTIME, NEST, LSFFL, IZFFL,
2266:      &          NFFL, LSDED, NDEAD, NSMAC, NSMIC, XIMP, WKIL, WSRV,
2267:      &          WTIME, WLLIM, KZREG, NCNTR, WCNTR, NEVENT, XXX, YYY,
2268:      &          ZZZ, AAA, BBB, CCC, WWW, KKP, IZZ, IGG, TTT, ITT,
2269:      &          LEVL, LZZ, LPOS, LCRS, XIM, KLSF, EEE, IBREG, IBSPC,
2270:      &          MB, NUC, NSTAL, SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL,
2271:      &          DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NKILD,
2272:      &          WKILD, NSURV, WSURV, NSPLT, WSPLT, R, DWK1, DWK2,
2273:      &          WK3, IWK1, IWK2, IWK3,
2274:      &          SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR,
2275:      &          WSD, WSC, NORDDS, NOPSDS,
2276: c##<2007/03/14:PN3:
2277:      &          NUCPN, NSMICPN, SMICPN,
2278: c##>
2279: c+beff2
2280:      &          WWB, WSB, NPTBE, WWBD, WSBD, WWLD, WSLD)
2281: c-beff2
2282:      end if
2283: C
2284:      end if
2285: C
2286: c##<2007/03/14:PN3:
2287: C ..... terminate all process
2288: 1000 continue
2289: c##>
2290:      NCOLP      = 0
2291:      return
2292:      end

```

src/mvp/phtgen.f

```

1:      subroutine PHTGEN(
2:      I      IMTTP, MTT0, IBP, INUCC, WPAR, IPGEN, NPGEN, ILSCP, NKILN,
3:      N      IOW, IRAND,
4:      N      JTIME, JIMPT, JWTIM, JRWVR, JMNTN, JDEBG, JLATT, JHLAT, JTLTT,
5:      N      NZONE, NREG, NTIME, NBANK, BANKP, NEST,
6:      N      NUC, NMT,
7:      6      ETOPP, EBOTP, ENGYB, CX, ICX, MCX, KLIB1,
8:      X      KLIB2, XLIB2, WGTP, WTIME, WLLIM, MXPGEN, KZREG,
9:      S      LSDED, NDEAD, LSCOL, IMTTP0,
10:     B      XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, EEE,
11:     B      TTT, ITT, XIM, KKP, IZZ, IGG, LEVL, LZZ, LPOS, LCRS,
12:     F      KLSF, IBREG, IBSPC,
13:     B      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
14:     T      NCNTR, WCNTR, NEVENT,
15:     R      IR1, IR2, IR3, IR4, IR5, IR6, IR7, IR8,
16:     W      NEG, LSTG, EWK, NKG, IWK, IPL,
17:     W      WK1, EWK1, EOUT, UOUT,
18:     &      KPNPRD, KPNFG, JPHNU) ! ph-nuc-3,4
19: C=<MVP>=====
20: C PURPOSE: PHOTON GENERATION FROM NEUTRON REACTION
21: C
22: C      DATA ABOUT ASSUMED-PHOTON-GENERATION-NEUTRON MUST BE PASSED
23: C      FROM 'NEUTR' ROUTINE.
24: C
25: C CALLED IN: NEUTR
26: C CALLS: RANU2
27: C=====
28:      implicit real*8(A-H,O-Z)
29: C
30:      include 'INC/_KPIDS'
31:      include 'INC/_NGPS'
32: C
33:      integer JDEBG(*)
34: C
35: C***** INPUT
36: C
37:      integer IMTTP
38:      integer MTT0(NBANK), IBP(NBANK), INUCC(NBANK), ILSCP(NBANK)
39:      real WPAR(NBANK)
40: C
41: C      (..... DATA ARE STORED AS INITIALLY )
42: C      INTEGER      MTT0(IMTTP), IBP(IMTTP), INUCC(IMTTP)
43: C      REAL          WPAR(IMTTP)
44: C
45: C***** OUTPUT
46: C
47:      integer IPGEN(NBANK), NPGEN, NKILN
48: C
49: C**** CROSS SECTION DATA IN CX ARRAY
50: C
51:      real EBOTP, ETOPP
52:      real CX(MCX), XLIB2(NUC,NMT,4)
53:      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
54: C
55: C**** PHOTON WEIGHT ****
56: C
57:      real WGTP(NREG)
58:      real WTIME(NTIME)
59:      real WLLIM
60:      real BANKP
61:      integer KZREG(NZONE)
62: C
63: C**** STACK
64: C
65:      integer LSDED(NBANK), NDEAD, LSCOL(NBANK)

```

```

66: C
67: C**** PARTICLE BANK
68: C
69:      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
70:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
71:      real EEE(NBANK), WWW(NBANK), XIM(NBANK)
72:      real*8 TTT(NBANK)
73:      integer ITT(NBANK)
74:      integer KKP(NBANK)
75:      integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
76:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
77:      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
78: C
79:      real*8 DBNK(NBANK,*)
80:      integer KDBNK(0:MDBNK)
81:      integer IBNK(NBANK,*)
82:      integer KIBNK(0:MIBNK)
83:      integer KPNFG(NBANK) ! ph-nuc-4
84: C
85: C**** TALLY ARRAY
86: C
87:      real ENGYB(*)
88:      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
89: C
90: C**** WORKING AREA
91: C
92: C      CAUTION: IR1 - IR8 SHARING MEMORY WITH R.
93: C
94:      real R(8*NBANK), EWK(NBANK), WK1(NBANK), EWK1(NBANK)
95:      real EOUT(NBANK), UOUT(NBANK)
96:      integer IR1(NBANK), IR2(NBANK), IR3(NBANK), IR4(NBANK),
97:      &      IR5(NBANK), IR6(NBANK), IR7(NBANK), IR8(NBANK),
98:      &      NEG(NBANK), LSTG(NBANK), NKG(NBANK), IWK(NBANK),
99:      &      IPL(NBANK)
100: C
101:      parameter( SMALL = 1.0E-35 )
102: C
103: C**** DATA STORED IN 'NEUTR' FOR PHOTON GENERATION *****
104: C
105: C -----
106: C      DATA FOR ASSUMED-GAMMA-GENERATION NEUTRONS
107: C      ( NO MORE EFFECTIVE AFTER 'GAMGEN' )
108: C
109: C
110: C      IMTTP      : NUMBER OF ASSUMED-GAMMA-GENERATION-NEUTRONS
111: C      MTT0(I)    : REACTION MT NUMBER ( NOT MTT0 !! )
112: C      IBP(I)     : BANK POINTER FOR ASSUMED-GAMMA-GENERATION NEUTRONS
113: C
114: C      INUCC(I)   : COLLIDING NUCLIDE ( <- IWK3 )
115: C      WPAR(I)    : WEIGHT OF PARENT NEUTRON
116: C
117: C      ILSCP(1:NBANK) : POSITION IN LSCOL(I) ARRAY OF CALLING ROUTINE.
118: C                      IF PARTICLES ARE IN NEUTRON-COLLISION BANK 'LSCOL'
119: C -----
120: C
121: C      << CAUTION !! >>
122: C
123: C      C** PARTICLE WEIGHTS FOR NEUTRON IN BANK MAY BE CHANGED IF "ANALOG
124: C      C**SELECTION" BETWEEN NEUTRON AND PHOTON IS TAKEN AS THE WAY TO DEAL
125: C      C*:WITH BANK-OVERFLOW.
126: C -----
127: C
128: C
129: C -----
130: C      DETERMINE 'MTTP' PARAMETER (MT # FOR WHICH PHOTON GENERATION DATA

```

src/mvp/phtgen.f

```

131: C                                ARE GIVEN )
132: C-----
133: C
134:      NKILN      = 0
135: C
136:      NCNTR(3,KPNEUT) = NCNTR(3,KPNEUT) + IMTTP
137: C
138: *VOCL LOOP,NOVREC
139:   do 100 I = 1, IMTTP
140:     WCNTR(3,KPNEUT) = WCNTR(3,KPNEUT) + WPAR(I)
141: C
142:     MTT0G      = KLIB2(INUCC(I),MTT0(I),15)
143: C
144: C     .... Jul 6 1994 , added for JENDL 3.2
145:     MT3        = MTT0G/1000000
146:     MTT0G      = MTT0G - 1000000*MT3
147: C
148:     MT1        = MTT0G/1000
149:     MT2        = MTT0G - 1000*MT1
150: C
151:     EWK(I)     = EEE(IBP(I))
152: C
153: C     .... for JENDL 3.2
154:   if ( MT3.ne.0 ) then
155:     if ( EWK(I).le.XLIB2(INUCC(I),MTT0(I),4) ) then
156:       MTT0(I) = MT3
157:     else if ( EWK(I).ge.XLIB2(INUCC(I),MTT0(I),3) ) then
158:       MTT0(I) = MT1
159:     else
160:       MTT0(I) = MT3
161:     end if
162:   else if ( MT1.ne.0 ) then
163: C
164: C     .... E <= EG1L , WHICH IS CHANGED TO E <= EG2U
165: C     EG1L=XLIB2(...3), EG2U=XLIB2(...4)
166:   if ( EWK(I).le.XLIB2(INUCC(I),MTT0(I),4) ) then
167:     MTT0(I) = MT2
168:   else
169:     MTT0(I) = MT1
170:   end if
171: else
172:   MTT0(I) = MT2
173: end if
174: C
175: C-----
176: C ENERGY MESH POINTER & ENERGY TABLE LENGTH
177: C-----
178: C      LSTG(I) POINTS EMESHG(1) FOR MT=MTT0(I)
179: C
180: C
181:   LSTG(I) = KLIB2(INUCC(I),MTT0(I),14) + 1
182: C*****NEG(I)      = KLIB2( INUCC(I), MTT0(I), 16 ) *****
183: C *** ALTERNATIVE ***
184:   NEG(I) = ICX(LSTG(I)-1)
185:   100 continue
186: C-----
187: C TABLE SEARCH: EWK(I) -> ( CX(LSTG(I)+L-1),L=1,NEG(I) ) -> IR3(I)
188: C
189: C   IWK : WORKING AREA ONLY FOR TABLE SEARCH
190: C   IR3 : TABLE POSITION AS RESULT ( 1 <= IR3 <= NEG-1 )
191: C-----
192: C
193:   call BSDEC3( CX, NEG, IWK, LSTG, EWK, IR3, IMTTP )
194: C
195: C   IWK IS FREE TO USE HEREFTER.

```

```

196: C
197: C-----
198: C
199: C CALCULATE THE NUMBER OF PHOTONS GENEATED
200: C-----
201: C
202:   call RANU2( IRAND, R, IMTTP, ICON )
203: C
204:   NPHOT      = 0
205:   do 110 I = 1, IMTTP
206:     IP       = IBP(I)
207: C
208: C     .... TOTAL PHOTON YIELD * WEIGHT ....
209: C
210:     IPE      = LSTG(I) + IR3(I)
211:     IPY      = IPE + NEG(I)
212: C
213:     if ( CX(IPE-1).ge.EWK(I).and.CX(IPE).le.EWK(I) ) then
214:       WP      =
215: &         (CX(IPY)+(CX(IPE)-EWK(I))/(CX(IPE)-CX(IPE-1)))*
216: &         (CX(IPY-1)-CX(IPY)))*WPAR(I)
217:     else
218:       WP      = 0.0
219:     end if
220: C
221: C     .... NUMBER OF PHOTON TO BE GENERATED --> IR3(I)
222: C     .... WEIGHT GIVEN TO EACH OF GENERATED PHOTON --> WPAR(I)
223: C
224:     if ( JTLLT.eq.0 ) then
225:       WGP      = WGTP(KZREG(IZZ(IP)))
226:     else
227:       WGP      = WGTP(IBREG(IP))
228:     end if
229: C
230:     if ( JWTIM.ne.0 ) WGP = WGP*WTIME(ITT(IP))
231:     if ( JRWVR.ne.0 ) WGP = WGP*DBNK(IP,KDBNK(1))
232: C
233:     IR3(I)     = WP/WGP + R(I)
234:     WPAR(I)    = WGP
235:     NPHOT      = NPHOT + IR3(I)
236:   110 continue
237: C
238: C     ... check generation cutoff
239: C
240:   if ( NPHOT.gt.0.and.MXPGEN.gt.0 ) then
241:     NGCUT      = 0
242:     do 120 I = 1, IMTTP
243:       IP       = IBP(I)
244:       if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IR3(I).gt.0 ) NGCUT =
245: &         NGCUT + 1
246:   120 continue
247: C
248:   if ( NGCUT.gt.0 ) then
249:     NCUT       = 0
250:     WWC        = 0.0D0
251:     do 130 I = 1, IMTTP
252:       IP       = IBP(I)
253:       if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IR3(I).gt.0 ) then
254:         NCUT    = NCUT + IR3(I)
255:         WWC     = WWC + WPAR(I)*IR3(I)
256:         IR3(I)  = 0
257:       end if
258:   130 continue
259:   NCNTR(27,KPPHOT) = NCNTR(27,KPPHOT) + NCUT
260:   WCNTR(27,KPPHOT) = WCNTR(27,KPPHOT) + WWC

```

src/mvp/phtgen.f

```

261:      NPHOT = NPHOT - NCUT
262:      end if
263:      end if
264: C
265: C
266: C----- NO PHOTON IS GENERATED ----> RETURN
267: C
268: C
269:      if ( NPHOT.le.0 ) then
270: C
271: C---- DEAD PARTICLES ARE ADDED TO TERMINATION STACK
272:      do 140 I = 1, IMTTP0
273:      NDEAD = NDEAD + 1
274:      LSDED(NDEAD) = IBP(I)
275:      if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0 ! ph-nuc-3
276:      if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0 ! ph-nuc-3
277:      if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0 ! ph-nuc-3
278:      if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0 ! ph-nuc-4
279: 140 continue
280: C
281:      return
282: C
283:      end if
284: C
285: C
286: C-----
287: C BANK HAS INSUFFICIENT SPACE TO ACCEPT ALL GENERATED PHOTON
288: C-----
289: C
290:      if ( NPHOT.gt.NDEAD ) then
291: C
292: C ... CHECK NUMBER OF PHOTON GENERATING CASE. ....
293: C
294:      INP = 0
295:      ND = NDEAD
296: *VOCL LOOP,NOVREC
297:      do 150 I = 1, IMTTP
298:      IBPI = IBP(I)
299:      if ( IR3(I).gt.0 ) then
300:      INP = INP + 1
301:      IBP(INP) = IBPI
302:      MTT0(INP) = MTT0(I)
303:      INUCC(INP) = INUCC(I)
304:      WPAR(INP) = WPAR(I)
305:      EWK(INP) = EWK(I)
306:      LSTG(INP) = LSTG(I)
307:      NEG(INP) = NEG(I)
308:      IR3(INP) = IR3(I)
309:      if ( I.le.IMTTP0 ) IPGEN(INP) = IBPI
310:      else if ( I.le.IMTTP0 ) then
311:      NDEAD = NDEAD + 1
312:      LSDED(NDEAD) = IBPI
313:      if ( KIBNK(17).ne.0 ) IBNK(IBPI,KIBNK(17)) = 0 ! ph-nuc-3
314:      if ( KIBNK(18).ne.0 ) IBNK(IBPI,KIBNK(18)) = 0 ! ph-nuc-3
315:      if ( KIBNK(19).ne.0 ) IBNK(IBPI,KIBNK(19)) = 0 ! ph-nuc-3
316:      if ( JPHNU.ne.0 ) KPNFG(IBPI) = 0 ! ph-nuc-4
317:      end if
318: 150 continue
319: C
320:      IMTTP0 = IMTTP0 + ND - NDEAD
321: C
322: C ..... RELATIVELY LUCKY CASE !! ... NDEAD >= INP ...
323: C
324: C
325:      if ( NDEAD.ge.INP-IMTTP0 ) then

```

```

326:      NPH = 0
327: C
328: *VOCL LOOP,SCALAR
329:      do 160 I = 1, INP
330:      NPH = NPH + IR3(I)
331:      if ( I.le.IMTTP0 ) NPH = NPH - 1
332:      if ( NPH+INP-I.gt.NDEAD ) go to 170
333: 160 continue
334:      I = INP + 1
335: 170 III = I
336: C
337:      do 180 I = III, INP
338:      WPAR(I) = WPAR(I)*IR3(I)
339:      IR3(I) = 1
340: 180 continue
341: C
342: C
343:      MA = 0
344:      ND = NDEAD + IMTTP0 + 1
345: C
346:      do 190 I = 1, INP
347:      IR3(I) = IR3(I) - 1
348:      MA = MAX(MA,IR3(I))
349:      if ( I.gt.IMTTP0 ) then
350:      IPGEN(I) = LSDED(ND-I)
351:      end if
352: 190 continue
353: C
354:      N = INP
355: C
356:      do 210 K = 1, MA
357: *VOCL LOOP,NOVREC
358:      do 200 I = 1, INP
359:      if ( IR3(I).ge.K ) then
360:      N = N + 1
361:      IPGEN(N) = LSDED(ND-N)
362:      IBP(N) = IBP(I)
363:      MTT0(N) = MTT0(I)
364:      INUCC(N) = INUCC(I)
365:      WPAR(N) = WPAR(I)
366:      EWK(N) = EWK(I)
367:      LSTG(N) = LSTG(I)
368:      NEG(N) = NEG(I)
369:      end if
370:      continue
371: 210 continue
372: C
373:      IMTTP = N
374:      NDEAD = NDEAD - IMTTP + IMTTP0
375: C
376: C
377: C
378: C ..... UNLUCKY CASE : SOME PHOTONS ARE PREVENTED TO BE BORN
379: C
380: C
381:      else
382:      N0 = IMTTP0 + NDEAD
383:      NT = INP - N0
384:      call RANU2( IRAND, R(N0+1), NT, ICON )
385: C
386:      do 220 I = 1, NDEAD
387:      IPGEN(IMTTP0+I) = LSDED(I)
388: 220 continue
389: C
390:      do 230 I = 1, INP

```

src/mvp/phtgen.f

```

391:          WPAR(I) = WPAR(I)*IR3(I)
392: 230      continue
393: C
394:          N      = N0
395: *VOCL LOOP,NOVREC
396:          do 240 I = N0 + 1, INP
397: C
398: C      .... PARTICLES ARE ALIVE AS PHOTON !! ....
399: C
400: C
401: C -----
402: C      NECESSARY TO CHECK NEUTRON COLLISION BANK INDEX ILSACP
403: C      AFTER THIS ROUTINE !!!
404: C -----
405: C
406: C
407: C
408:          IBPI    = IBP(I)
409:          if ( R(I).lt.BANKP ) then
410:              N      = N + 1
411:              IPGEN(N) = IBPI
412:              WPAR(N) = WPAR(I) /BANKP
413:              LSCOL(ILSCP(IBPI)) = -1
414: C              NKILN    = NKILN + 1
415:              IBP(N)    = IBPI
416:              MTT0(N)   = MTT0(I)
417:              INUCC(N)  = INUCC(I)
418:              EWK(N)    = EWK(I)
419:              LSTG(N)   = LSTG(I)
420:              NEG(N)    = NEG(I)
421:              WCNTR(24,KPNEUT) = WCNTR(24,KPNEUT) + WWW(IBPI)
422:          else
423: C
424: C      .... PARTICLES ARE ALIVE AS NEUTRON !! ....
425: C
426:              WWW(IBPI) = WWW(IBPI) /((1.0-BANKP)
427:          end if
428: C
429: 240      continue
430:          NKILN = N - N0
431:          IMTTP = N
432:          NDEAD = 0
433:          NCNTR(24,KPNEUT) = NCNTR(24,KPNEUT) + NKILN
434:          end if
435: C
436:          IMTTP0 = 0
437: C
438: C      < IR3 IS FREE TO USE HEREAFTER. >
439: C
440: C
441: C -----
442: C      ALL PHOTONS CAN BE GENERATED
443: C -----
444: C
445:          else
446: C
447: C----- IPL(IMTTP0) : BANK POINTER FOR MOTHER NEUTRONS ANALOG ABSORBED,
448: C      BUT NOT YET COUNTED INTO LSDED AND NDEAD
449: C      THESE NEUTRONS MUST BE ADDED TO LSDED AT THE END OF THIS TASK.
450: C
451:          do 250 I = 1, IMTTP0
452:              IPL(I) = IBP(I)
453: 250      continue
454: C
455: C

```

```

456: C-----
457: C      .... COMPRESS SURVIVED PHOTON PARAMETERS
458: C-----
459: C
460:          IMT      = 0
461: *VOCL LOOP,NOVREC
462:          do 260 I = 1, IMTTP
463:              if ( IR3(I).ne.0 ) then
464:                  IMT      = IMT + 1
465:                  IPGEN(IMT) = LSDED(NDEAD+1-IMT)
466:                  IBP(IMT)  = IBP(I)
467:                  MTT0(IMT) = MTT0(I)
468:                  INUCC(IMT) = INUCC(I)
469:                  WPAR(IMT) = WPAR(I)
470:                  EWK(IMT)  = EWK(I)
471:                  LSTG(IMT) = LSTG(I)
472:                  NEG(IMT)  = NEG(I)
473:                  IR3(IMT)  = IR3(I)
474:              end if
475: 260      continue
476:          IMTTP = IMT
477: C
478: C-----
479: C      INFLATE PHTON DATA
480: C-----
481: C
482:          MA      = 0
483:          do 270 I = 1, IMTTP
484:              MA      = MAX(MA,IR3(I))
485: 270      continue
486: C
487:          N      = IMTTP
488:          do 290 K = 2, MA
489: *VOCL LOOP,NOVREC
490:              do 280 I = 1, IMTTP
491:                  if ( IR3(I).ge.K ) then
492:                      N      = N + 1
493:                      IPGEN(N) = LSDED(NDEAD+1-N)
494:                      IBP(N)  = IBP(I)
495:                      MTT0(N) = MTT0(I)
496:                      INUCC(N) = INUCC(I)
497:                      WPAR(N) = WPAR(I)
498:                      EWK(N)  = EWK(I)
499:                      LSTG(N) = LSTG(I)
500:                      NEG(N)  = NEG(I)
501:                  end if
502: 280      continue
503: 290      continue
504:          IMTTP = N
505:          NDEAD = NDEAD - IMTTP
506: C
507: C---- DEAD PARTICLES ARE ADDED TO TERMINATION STACK
508:          do 300 I = 1, IMTTP0
509:              NDEAD = NDEAD + 1
510:              LSDED(NDEAD) = IPL(I)
511:              if ( KIBNK(17).ne.0 ) IBNK(IPL(I),KIBNK(17)) = 0 ! ph-nuc-3
512:              if ( KIBNK(18).ne.0 ) IBNK(IPL(I),KIBNK(18)) = 0 ! ph-nuc-3
513:              if ( KIBNK(19).ne.0 ) IBNK(IPL(I),KIBNK(19)) = 0 ! ph-nuc-3
514:              if ( JPHNU.ne.0 ) KPNUF(IPL(I)) = 0 ! ph-nuc-4
515: 300      continue
516: C
517:          end if
518: C
519: C      < IR3 & IPL IS FREE TO USE HEREAFTER. >
520: C

```


src/mvp/phtgen.f

```

521:      if ( IMTTP.gt.0 .and. KPNPRD.eq.0 ) then ! ph-nuc-3 >>
522:        if ( KIBNK(17).ne.0 ) then
523:          if ( JTLLT.eq.0 ) then
524:            do I = 1, IMTTP
525:              IBNK(IPGEN(I),KIBNK(17)) = - KZREG(IZZ(IBP(I)))
526:            end do
527:          else
528:            do I = 1, IMTTP
529:              IBNK(IPGEN(I),KIBNK(17)) = - IBREG(IBP(I))
530:            end do
531:          end if
532:        end if
533:        if ( KIBNK(18).ne.0 ) then
534:          do I = 1, IMTTP
535:            IBNK(IPGEN(I),KIBNK(18)) = INUCC(I)
536:          end do
537:        end if
538:        if ( KIBNK(19).ne.0 ) then
539:          do I = 1, IMTTP
540:            IBNK(IPGEN(I),KIBNK(19)) = abs(MTT0(I))
541:          end do
542:        end if
543:        if ( JPHNU.ne.0 ) then ! ph-nuc-4 >>
544:          do I = 1, IMTTP
545:            KPNFG(IPGEN(I)) = 0
546:          end do
547:        end if ! ph-nuc-4 <<
548:      end if ! ph-nuc-3 <<
549: C
550: C-----
551: C      .... PHOTONS GENERATED .....
552: C-----
553: C
554: C
555: C
556: C      if ( IMTTP.gt.0 ) then
557: C
558: C-----
559: C      SELECT PHOTON GENERATION SUBSECTION
560: C-----
561: C
562: C      MNKG      = 0
563: C      do 310 I = 1, IMTTP
564: C        IP      = IBP(I)
565: C        NEGP     = KLIB2(INUCC(I),MTT0(I),17)
566: C        NKG(I)  = KLIB2(INUCC(I),MTT0(I),18)
567: C
568: C        LSTG(I) POINTS EMESHG HERE.
569: C
570: C        ----> LSTG(I) POINTS EPROBG HEREAFTER .....
571: C
572: C      V----->V----->V -->V-->V----->V----->V
573: C      EMESHG(NEG),TYIELD(NEG),NEGP,NKG,LSTF5G(NKG),ESK(NKG),ERPOBG(NEGP)
574: C
575: C        LSTG(I) = LSTG(I) + 2*NEGP(I) + 2 + 2*NKG(I)
576: C
577: C        .... NEGP(I) = NEGP HEREAFTER .....
578: C
579: C        NEGP(I) = NEGP
580: C        MNKG     = MAX(MNKG,NKG(I))
581: C      310 continue
582: C
583: C      .... IR3(I) IS POINTER TO EPROBG TABLE ....
584: C
585: C      call BSDEC3( CX, NEG, IWK, LSTG, EWK, IR3, IMTTP )

```

```

586: C
587: C      IDET      = 0
588: C
589: C      IWK(I)    : DETERMINED SUBSECTION NUMBER
590: C
591: C      call RANU2( IRAND, R, IMTTP, ICON )
592: C
593: C      *VOCL LOOP,NOVREC
594: C      do 320 I = 1, IMTTP
595: C
596: C
597: C      .... CALCULATE ACCUMULATED YIELD FOR 1'ST SUBSECTION -> R
598: C      (MULTIPLIED BY RANDOM NUMBER)
599: C      ENERGY POINT          -> IR5
600: C      ENERGY INTERPOLATION FACTOR -> WK1
601: C      SUBSECTION YIELD POINTER -> IR6 + NEG*(NK-1)
602: C
603: C
604: C      LSTG  IPE,IR5(I)
605: C      V....V
606: C      EPROBG(NEGP)
607: C
608: C      IPE      = LSTG(I) + IR3(I)
609: C
610: C
611: C      LSTG      IR6(I),IPY
612: C      V----->V----->V....V
613: C      EPROBG(NEGP),INTF5G(NEGP),CY(NEGP,NKG)
614: C
615: C      IR6(I)    = LSTG(I) + 2*NEGP(I) + IR3(I)
616: C
617: C
618: C      IPY       = IR6(I)
619: C      INTF5G    = ICX(LSTG(I)+NEGP(I)+IR3(I))
620: C
621: C      if ( INTF5G.eq.2 ) then
622: C        WK1(I)  = (CX(IPE)-EWK(I)) / (CX(IPE)-CX(IPE-1))
623: C        R(I)    = R(I)*(CX(IPY)+WK1(I)*(CX(IPY-1)-CX(IPY)))
624: C      else
625: C        XC      = LOG(CX(IPE))
626: C        WK1(I)  = (XC-LOG(EWK(I))) / (XC-LOG(CX(IPE-1)))
627: C        R(I)    = R(I)*(CX(IPY)+WK1(I)*(CX(IPY-1)-CX(IPY)))
628: C      end if
629: C
630: C
631: C      .... IWK : DETERMINATION FLAG FOR THE NEXT LOOP. ....
632: C
633: C      if ( NKG(I).eq.1 ) then
634: C        IWK(I)  = 1
635: C        IDET    = IDET + 1
636: C      else
637: C        IWK(I)  = 0
638: C      end if
639: C
640: C      320 continue
641: C
642: C
643: C      .... DETERMINED SUBSECTION NUMBER --> IWK(I)
644: C
645: C
646: C      do 340 M = 2, MNKG
647: C        if ( IDET.eq.IMTTP ) go to 350
648: C
649: C      do 330 I = 1, IMTTP
650: C        if ( NKG(I).ge.M.and.IWK(I).eq.0 ) then

```

src/mvp/phtgen.f

```

651: C
652:          IPY      = IR6(I) + NEG(I)*(M-1)
653: C
654:          CY       = CX(IPY) + WK1(I)*(CX(IPY-1)-CX(IPY))
655: C
656:          if ( R(I).gt.CY ) then
657:              IDET   = IDET + 1
658:              IWK(I) = M - 1
659:          end if
660:          end if
661: 330      continue
662: 340      continue
663: C
664: C < IR5, IR6 & WK1 ARE FREE TO USE HEREFTER >
665: C
666: 350      continue
667:          if ( IDET.lt.IMTTP ) then
668:              do 360 I = 1, IMTTP
669:                  if ( IWK(I).eq.0 ) IWK(I) = NKG(I)
670: 360          continue
671:          end if
672: C
673: C
674: C -----
675: C DETERMINATION OF ENERGY ( -> EOUT(I) )
676: C -----
677: C
678: C
679:          NLF6 = 0
680: C
681: C COLLECT LF=6 OR LF=61 PARTICLES.
682: C
683: C ( STARTING ADDRESS OF ANGULAR DISTRIBUTION DATA -> LSTG(I) )
684: C
685: C *VOCL LOOP,NOVREC
686:       do 370 I = 1, IMTTP
687: C
688: C
689: C POSITION OF ENERGY DISTRIBUTION DATA OF SUBSECTION INK(I)
690: C
691: C ..... LST5FG(IWK(I)-1) ---> LST
692: C
693: C          LST          LSTG
694: C V<-----V<-----V
695: C LST5FG(NKG),ESK(NKG),ERPOBG(NEGP)
696: C
697: C
698: C          LST = ICX(LSTG(I)-2*NKG(I)+IWK(I)-1)
699: C          LF  = ICX(LST)
700: C
701: C ..... LF = 2 : LINEAR FUNCTION OF NEUTRON ENERGY
702: C
703: C DATA:          LF,1,GCOEF1,GCOEF2
704: C
705: C
706: C          if ( LF.eq.2 ) then
707: C              EOUT(I) = CX(LST+2) + CX(LST+3)*EWK(I)
708: C              LSTG(I) = LST + 4
709: C
710: C ..... ASSUMING LF = 6 OR LF = 61!
711: C
712: C          else
713: C              NLF6 = NLF6 + 1
714: C
715: C >>>>>> POSITION MEMORY FOR I=1,IMTTP >>>>>

```

```

716: C
717:          IR8(NLF6) = I
718: C
719: C >>>>>> NEF5S
720: C
721:          NEF5S = ICX(LST+1)
722:          IR5(NLF6) = NEF5S
723: C
724: C >>>>>> POSITION OF ENERGY TABLE >>>>>>
725: C
726:          IR6(NLF6) = LST + 2
727:          EWK1(NLF6) = EWK(I)
728: C
729: C >>>>>> LF VALUE >>>>>>
730: C
731:          IWK(NLF6) = LF
732: C
733: C IN CASE OF NBINE>1
734: C          LST
735: C          V--->V--->V----->V----->
736: C          LF, NEF5S,EENG(NEF5S),INTENG(NEF5S),
737: C                                          LSTG(I)
738: C          --->V----->V----->V
739: C          (EBINF1(1:NBINE1,J),EPRBF1(1:NBINE1,J=1,NEF5S),LSTGAN
740: C
741: C IN CASE OF NBINE=0, NBINE1=1
742: C          LST
743: C          V--->V--->V----->V----->V----->
744: C          LF, NEF5S,EENG(NEF5S),INTENG(NEF5S),LSTF5E(NEF5S),
745: C          --->V----->V----->V
746: C          (NEP(I),(Q(J,I),J=1,NEP(I)),
747: C          ----->
748: C          (IEPOS1(J,I),IEPOS2(J,I),J=1,NEP(I))
749: C          ----->
750: C          (EBINF1(J,I),J=1,NEP(I)+1),(EPRBF1(J,I),J=1,NEP(I)+1),
751: C          LSTG(I)
752: C          ----->V
753: C          I=1,NEF5S),LSTGAN
754: C          NBINE1 = KL1B1(INUCC(I),21)
755: C          if ( NBINE1.gt.1 ) then
756: C              LSTG(I) = LST + 2 + 2*(1+NBINE1)*NEF5S
757: C          else
758: C              LSTGL = ICX(LST+1+3*NEF5S)
759: C              NEP = ICX(LSTGL)
760: C              LSTG(I) = LSTGL + 3 + NEP*5
761: C              if ( LF.eq.61 ) LSTG(I) = LSTG(I) + NEP*2 + 3
762: C          end if
763: C          end if
764: 370      continue
765: C
766: C
767: C ..... LF = 6 OR 61 FOR SOME CASES .....
768: C
769: C
770: C          if ( NLF6.gt.0 ) then
771: C              call BSDEC3( CX, IR5, IR1, IR6, EWK1, IR7, NLF6 )
772: C
773: C              call RANU2( IRAND, R, 4*NLF6, ICON )
774: C              N2 = 2*NLF6
775: C              N3 = 3*NLF6
776: C
777: C < DON'T USE IR1 - IR4 BECAUSE THEY MAY BE SHARING MEMORY WITH
778: C R(1: 4*IMTTP) >
779: C
780: C *VOCL LOOP,NOVREC

```

src/mvp/phtgen.f

```

781:      do 380 J = 1, NLF6
782: C
783:         NEF5S = IR5(J)
784:         NBINE1 = KLIB1(INUCC(IR8(J)),21)
785: C
786: C         ( ..... IR6(J) POINTS EENG HERE )
787: C
788:         if ( IR7(J).eq.0 ) IR7(J) = 1
789:         L2 = IR6(J) + IR7(J)
790:         L1 = L2 -1
791: C
792: C ..... INTE = ICX(L2+NEF5S)
793: C
794:         INTE0 = ICX(L2+NEF5S)
795:         INTE1 = INTE0 / 10
796:         INTE = INTE0 - 10*INTE1
797:         if ( INTE.ne.3.and.INTE.ne.5 ) then
798:             X1 = (CX(L1)-EWK1(J)) / (CX(L1)-CX(L2))
799:         else
800:             X1 = LOG(CX(L1)/EWK1(J)) / LOG(CX(L1)/CX(L2))
801:         end if
802:         if ( X1.lt.0.0 ) X1 = 0.0
803:         if ( X1.gt.1.0 ) X1 = 1.0
804: C
805:         L1 = IR6(J) + NEF5S*2
806: C
807: C .... SELECTED ENERGY POINT NUMBER ....
808: C
809: C
810: C/#IF ROUNDOFF(NEAREST)
811:         IT = MIN(1,INT(R(J)+X1)) + IR7(J)
812: C/#ELSE
813:         R1 = R(J) + X1
814:         IT = IR7(J) + R1
815: C/#ENDIF
816: C
817: C
818: C*VOCL STMT,IF(0)
819: C         if ( NBINE1.gt.1 ) then
820: C             .... SELECTION OF OUTGOING ENERGY FROM EQUIPROBABLE BIN ....
821: C
822: C
823: C/#IF ROUNDOFF(NEAREST)
824:         LL5 = MIN(INT(R(NLF6+J)*(NBINE1-1)),NBINE1-2)
825: C/#ELSE
826:         LL5 = R(NLF6+J)*(NBINE1-1)
827: C/#ENDIF
828:         LLE = L1 + 2*NBINE1*(IT-1)
829:         LE = LLE + LL5
830: C
831:         LP = LE + NBINE1
832:         RR = R(N2+J)
833: C
834:         NEP = NBINE1 - 1
835:         ND = NEP
836:         else
837: C             .... SELECTION OF OUTGOING ENERGY BY BMC SAMPLING ....
838: C
839:         LSTF5E = ICX((L1+IT-1))
840:         NEP = ICX(LSTF5E)
841: C
842:         if ( IWK(J).eq.61 ) LSTF5E = LSTF5E + 1
843:         ND = ICX(LSTF5E)
844: C
845: C/#IF ROUNDOFF(NEAREST)

```

```

846:         LL2 = MIN(INT(NEP*R(NLF6+J))+1,NEP)
847: C/#ELSE
848:         LL2 = NEP*R(NLF6+J)+1
849: C/#ENDIF
850:         LL3 = LSTF5E + LL2
851: C
852:         LL4 = NEP + 2*LL2
853: C/#IF ROUNDOFF(NEAREST)
854:         LL4 = MIN(LL4,INT(LL4+R(N2+J)-CX(LL3))) + LSTF5E
855: C/#ELSE
856:         R2 = R(N2+J) - CX(LL3)
857:         LL4 = LSTF5E + LL4 + R2
858: C         LL4 = LSTF5E + NEP + 2*LL2 + R2
859: C/#ENDIF
860: C
861:         LL5 = ICX(LL4)
862:         LLL = LSTF5E + 3*NEP
863:         LE = LLL + LL5
864:         LP = LE + NEP + 1
865:         RR = R(N3+J)
866:         if ( LL5.gt.ND ) RR = 0.0
867: C
868:         LLE = LLL + 1
869:         end if
870: C
871:         E0 = CX(LE)
872:         E1 = CX(LE+1)
873:         P0 = CX(LP)
874:         P1 = CX(LP+1)
875:         RRR = (P0+P1)*RR
876:         EOUT(IR8(J)) = E1 + (E0-E1)*RRR/
877:         & (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
878: C
879:         if ( INTE1.gt.0 .and. LL5.le.ND ) then
880:             if ( NBINE1.gt.1 ) then
881:                 LLE2 = L1 + 2*NBINE1*IR7(J)
882:                 LLE1 = LLE2 - 2*NBINE1
883:                 NEP1 = NEP
884:                 NEP2 = NEP
885:                 ND1 = NEP
886:                 ND2 = NEP
887:             else if ( IT.eq.IR7(J) ) then
888:                 NEP1 = NEP
889:                 ND1 = ND
890:                 LLE1 = LLE
891:                 LSTF5T = ICX(L1+IT)
892:                 NEP2 = ICX(LSTF5T)
893:                 if ( IWK(J).eq.61 ) LSTF5T = LSTF5T + 1
894:                 ND2 = ICX(LSTF5T)
895:                 LLE2 = LSTF5T + 3*NEP2 + 1
896:             else
897:                 NEP2 = NEP
898:                 ND2 = ND
899:                 LLE2 = LLE
900:                 LSTF5T = ICX(L1+IT-2)
901:                 NEP1 = ICX(LSTF5T)
902:                 if ( IWK(J).eq.61 ) LSTF5T = LSTF5T + 1
903:                 ND1 = ICX(LSTF5T)
904:                 LLE1 = LSTF5T + 3*NEP1 + 1
905:             end if
906: C
907:         if ( INTE1.eq.2 ) then
908: C             ... linear interpolation is assumed
909:                 EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
910:                 EOUT(IR8(J)) = EOUT(IR8(J)) * EMAX / CX(LLE)

```

src/mvp/phtgen.f

```

911:      else
912: C    ... linear interpolation is assumed
913:      EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
914:      EMIN = CX(LLE1+ND1)
915:      &      + (CX(LLE2+ND2)-CX(LLE1+ND1))*X1
916:      EOUT(IR8(J)) = EMIN + (EMAX-EMIN)*
917:      &      (EOUT(IR8(J))-CX(LLE+ND))/(CX(LLE)-CX(LLE+ND))
918:      end if
919:      end if
920: 380      continue
921:  end if
922: C
923: C
924: C .... CHECK OUTGOING ENERGY .....
925: C
926: C
927:      NN = 0
928: *VOCL LOOP,NOVREC
929:      do 390 I = 1, IMTTP
930:      EOUT(I) = MIN(EOUT(I),ETOPP)
931:      IPGENI = IPGEN(I)
932:      if ( EOUT(I).ge.EBOTP ) then
933:      NN = NN + 1
934:      EOUT(NN) = EOUT(I)
935:      LSTG(NN) = LSTG(I)
936:      EWK(NN) = EWK(I)
937:      IBP(NN) = IBP(I)
938:      WPAR(NN) = WPAR(I)
939:      IPGEN(NN) = IPGENI
940:      else
941:      NDEAD = NDEAD + 1
942:      LSDED(NDEAD) = IPGENI
943:      if ( KIBNK(17).ne.0 ) IBNK(IPGENI,KIBNK(17)) = 0 ! ph-nuc-3
944:      if ( KIBNK(18).ne.0 ) IBNK(IPGENI,KIBNK(18)) = 0 ! ph-nuc-3
945:      if ( KIBNK(19).ne.0 ) IBNK(IPGENI,KIBNK(19)) = 0 ! ph-nuc-3
946:      if ( JPHNU.ne.0 ) KPNFG(IPGENI) = 0 ! ph-nuc-4
947:      end if
948: 390      continue
949:      IMTTP = NN
950: C
951: C
952: C < IR1,IR2,IR5,IR6,IR7,IR8 & IWK ARE FREE TO USE HEREAFER. >
953: C
954: C
955: C-----
956: C DETERMINATION OF ANGLE
957: C-----
958: C
959: C STARTING ADDRSS OF ANGULAR DISTRIBTION DATA IS NOW IN LSTG(I).
960: C
961: C .... GET ENERGY POINT NUMBER -> NEG(I)
962: C
963:      NA = 0
964:      call RANU2( IRAND, R, IMTTP, ICON )
965: *VOCL LOOP,NOVREC
966:      do 400 I = 1, IMTTP
967: C
968: C ..... LSTG(I) POINTS LSTGAN HERE ....
969: C
970: C      NEGANG --> NEG(I)
971: C
972: C      NEG(I) = ICX(LSTG(I)+1)
973: C
974: C      if ( NEG(I).gt.0 ) then
975: C      NA = NA + 1

```

```

976:      IR5(NA) = NEG(I)
977: C
978: C ..... IR6(I) POINTS EGANG ...
979: C
980: C      IR6(NA) = ICX(LSTG(I))
981: C      IR8(NA) = I
982: C      EWK1(NA) = EWK(I)
983: C      else
984: C
985: C ..... NEGANG = 0 --> ISOTROPIC
986: C
987: C      UOUT(I) = 2.0*R(I) - 1
988: C      end if
989: 400      continue
990: C
991: C      if ( NA.gt.0 ) then
992: C      call BSDEC3( CX, IR5, IR7, IR6, EWK1, IWK, NA )
993: C      call RANU2( IRAND, R, 2*NA, ICON )
994: C
995: C < DON'T USE IR1 & IR2 BECAUSE THEY MAY BE SHARING MEMORY WITH
996: C R(1: 2*IMTTP) >
997: C
998: *VOCL LOOP,NOVREC
999:      do 410 J = 1, NA
1000: C
1001: C      NEGANG = IR5(J)
1002: C      NBINA1 = KLIB1(INUCC(IR8(J)),20)
1003: C
1004: C ..... IR6(I) POINTS EGANG HERE ....
1005: C      L1 = IR6(J) + IWK(J)
1006: C
1007: C ..... INTGAN = ICX(L1+NEGANG)
1008: C
1009: C      if ( ICX(L1+NEGANG).ne.3.and.ICX(L1+NEGANG).ne.5 ) then
1010: C      TI = (CX(L1)-EWK1(J)) / (CX(L1)-CX(L1-1))
1011: C      else
1012: C      XC = LOG(CX(L1))
1013: C      TI = (XC-LOG(EWK1(J))) / (XC-LOG(CX(L1-1)))
1014: C      end if
1015: C
1016: C ..... SELECTED ENERGY POINT NUMBER ....
1017: C
1018: C
1019: C /#IF ROUNDOFF(NEAREST)
1020: C      L1 = MIN(1,INT(1+R(J)-TI)) + IWK(J)
1021: C /#ELSE
1022: C      R1 = R(J)-TI
1023: C      L1 = IWK(J) + 1 + R1
1024: C /#ENDIF
1025: C
1026: C
1027: C ..... COSINE OF OUTGOING ANGLE
1028: C
1029: C      RR = R(NA+J)*(NBINA1-1)
1030: C
1031: C /#IF ROUNDOFF(NEAREST)
1032: C      LE = IR6(J) + 2*NEGANG + NBINA1*(L1-1)
1033: C      &      + MIN(INT(RR),NBINA1-1-1)
1034: C /#ELSE
1035: C      LE = IR6(J) + 2*NEGANG + NBINA1*(L1-1)
1036: C      *      &      + RR
1037: C /#ENDIF
1038: C      RR = RR - INT(RR)
1039: C      UOUT(IR8(J)) = CX(LE) + RR*(CX(LE+1)-CX(LE))
1040: 410      continue

```

src/mvp/phtgen.f

```

1041:         end if
1042: C
1043: C < IWK, IR1-IR8, EWK, EWK1, LSTG ARE FREE TO USE HEREFTER >
1044: C
1045: C-----
1046: C      DETERMINATION OF ENERGY GROUP -> IWK
1047: C-----
1048: C
1049: C      call BSVDEC(ENGYB(KENGP(KPPHOT)),NGP(KPPHOT)+1,EOUT,IWK,IMTTP)
1050: C
1051: C      .... PHOTON GROUP IS NGP1+1 TO NGP1+NGP2 .....
1052: C
1053: C      NNG = KNGP(KPPHOT) - 1
1054: C      do 420 I = 1, IMTTP
1055: C          IWK(I) = NNG + IWK(I)
1056: C      420 continue
1057: C
1058: C
1059: C-----
1060: C      MONITOR TALLY & STORE IN PARTICLE BANK
1061: C-----
1062: C
1063: C
1064: C      *VOCL LOOP,NOVREC
1065: C      do 430 I = 1, IMTTP
1066: C
1067: C          IP = LSDEB(NDEAD-I+1)
1068: C          IPGEN(I) = IP
1069: C          IP = IPGEN(I)
1070: C
1071: C      .... TALLY WEIGHTS OF GENERATED PHOTON ....
1072: C
1073: C          WCNTR(3,KPPHOT) = WCNTR(3,KPPHOT) + WPAR(I)
1074: C
1075: C          KKP(IP) = KPPHOT
1076: C
1077: C      .... COPY POSITION, ZONE & TIME OF PARENT PARTICLE ....
1078: C
1079: C          XXX(IP) = XXX(IBP(I))
1080: C          YYY(IP) = YYY(IBP(I))
1081: C          ZZZ(IP) = ZZZ(IBP(I))
1082: C          IZZ(IP) = IZZ(IBP(I))
1083: C          TTT(IP) = TTT(IBP(I))
1084: C
1085: C          KLSF(IP) = 0
1086: C
1087: C          if ( JLATT.ne.0 ) LEVL(IP) = LEVL(IBP(I))
1088: C          if ( JIMPT.ne.0 ) XIM(IP) = XIM(IBP(I))
1089: C
1090: C      CCCC      if ( JTLLT.ne.0 ) IBREG(IP) = IBREG(IBP(I))
1091: C      IBREG(IP) = IBREG(IBP(I))
1092: C          if ( JTIME.ne.0 ) ITT(IP) = ITT(IBP(I))
1093: C
1094: C      .... NEW WEIGHT, ENERGY & GROUP .....
1095: C
1096: C          WWW(IP) = WPAR(I)
1097: C          EEE(IP) = EOUT(I)
1098: C          IGG(IP) = IWK(I)
1099: C
1100: C      .... CALCULATE NEW DIRECTION ....
1101: C
1102: C          DA0 = AAA(IBP(I))
1103: C          DB0 = BBB(IBP(I))
1104: C          DC0 = CCC(IBP(I))
1105: C          ETA = 6.28318530717958668D0*R(I)

```

```

1106: C
1107: C      ACOS(-1)*2
1108: C      ANSWER = 6.28319D+00 6.28318530717958668 D+00
1109: C
1110: C          SINETA = SIN(ETA)
1111: C          COSETA = COS(ETA)
1112: C          STHETA = 1.0 - DA0*DA0
1113: C      *VOCL STMT,IF(100)
1114: C          if ( STHETA.gt.0.0 ) then
1115: C              STHETA = SQRT(STHETA)
1116: C              COSPHI = DB0/STHETA
1117: C              SINPHI = DC0/STHETA
1118: C          else
1119: C              STHETA = 0.0
1120: C              COSPHI = 1.0
1121: C              SINPHI = 0.0
1122: C          end if
1123: C
1124: C          SINPSI = SQRT(1.0-UOUT(I)*UOUT(I))
1125: C          DB0 = DB0*UOUT(I) + (DA0*COSPHI*COSETA
1126: C      &          -SINPHI*SINETA)*SINPSI
1127: C          DC0 = DC0*UOUT(I) + (DA0*SINPHI*COSETA
1128: C      &          +COSPHI*SINETA)*SINPSI
1129: C          DA0 = DA0*UOUT(I) - COSETA*SINPSI*STHETA
1130: C
1131: C          S = 1./SQRT(DA0*DA0+DB0*DB0+DC0*DC0)
1132: C          AAA(IP) = DA0*S
1133: C          BBB(IP) = DB0*S
1134: C          CCC(IP) = DC0*S
1135: C      430 continue
1136: C
1137: C      .... TALLY NUMBER OF PHOTON GENERATION ....
1138: C
1139: C          NCNTR(3,KPPHOT) = NCNTR(3,KPPHOT) + IMTTP
1140: C
1141: C          if ( JLATT.ne.0 ) then
1142: C              do 450 K = 1, NEST
1143: C      *VOCL LOOP,NOVREC
1144: C              do 440 I = 1, IMTTP
1145: C                  IP = IPGEN(I)
1146: C                  LZZ(IP,K) = LZZ(IBP(I),K)
1147: C                  LPOS(IP,K) = LPOS(IBP(I),K)
1148: C                  if ( JHLAT.ne.0 ) LCRS(IP,K) = LCRS(IBP(I),K)
1149: C                  if ( JTLLT.ne.0 ) IBSPC(IP,K) = IBSPC(IBP(I),K)
1150: C      440 continue
1151: C      450 continue
1152: C          end if
1153: C
1154: C      ... copy or change optional bank parameters
1155: C
1156: C          do 500 K = 1, KDBNK(0)
1157: C
1158: C              if ( K.eq.KDBNK(1) ) then
1159: C
1160: C                  ... birth weight is current weight !!!
1161: C
1162: C      *VOCL LOOP,NOVREC
1163: C              do 460 I = 1, IMTTP
1164: C                  DBNK(IPGEN(I),KDBNK(1)) = WWW(IPGEN(I))
1165: C      460 continue
1166: C
1167: C              ... time of birth is just now !!!
1168: C
1169: C          else if ( K.eq.KDBNK(2) ) then
1170: C      *VOCL LOOP,NOVREC

```

src/mvp/phtgen.f

```
1171:          do 470 I = 1, IMTTP
1172:             DBNK(IPGEN(I),KDBNK(2)) = TTT(IPGEN(I))
1173: 470          continue
1174:
1175:          else if ( K.eq.KDBNK(3) ) then
1176: *VOCL LOOP,NOVREC
1177:             do 480 I = 1, IMTTP
1178:                DBNK(IPGEN(I),KDBNK(3)) = EEE(IPGEN(I))
1179: 480          continue
1180:
1181: C          ... other optional bank parameters are only copy of those
1182: C          of parents.
1183:          else
1184: *VOCL LOOP,NOVREC
1185:             do 490 I = 1, IMTTP
1186:                DBNK(IPGEN(I),K) = DBNK(IBP(I),K)
1187: 490          continue
1188:             end if
1189: 500          continue
1190: C
1191:          do 530 K = 1, KIBNK(0)
1192: C
1193: C          ... increment "particle generation" if necessary
1194: C
1195:             if ( K.eq.KIBNK(4) ) then
1196: *VOCL LOOP,NOVREC
1197:                do 510 I = 1, IMTTP
1198:                   IBNK(IPGEN(I),KIBNK(4)) = IBNK(IBP(I),KIBNK(4)) + 1
1199: 510                continue
1200:             else if ( K.eq.KIBNK(17) .or. K.eq.KIBNK(18) .or.
1201: &                  K.eq.KIBNK(19) ) then ! ph-nuc-3
1202: C             --- stored in above ---
1203:             else
1204: *VOCL LOOP,NOVREC
1205:                do 520 I = 1, IMTTP
1206:                   IBNK(IPGEN(I),K) = IBNK(IBP(I),K)
1207: 520                continue
1208:             end if
1209: 530          continue
1210: C
1211:          NPGEN = IMTTP
1212: C
1213:          end if
1214: C
1215:          return
1216:          end
```

src/mvp/pmtinp.f

```

1:      subroutine PMTINP(IMG, A, IA, LIMIT, LAST, JDEBG, NREG, NUCPN,
2:      &
3:      &
4:      &
5:      &
6:      &
7:      C=<MVP>=====
8:      C purpose: input array data for probability for a specified reaction
9:      C type (mt) by region and nuclide in photonuclear reaction.
10:     C called in: INTRO
11:     C calls:
12:     C=====
13:     include '../shared/INC/_LNAM'
14:     C
15:     real A(LIMIT)
16:     integer IA(LIMIT), JDEBG(*)
17:     C
18:     ..... data for photonuclear .....
19:     integer MNUCPN(*), LPIDPN(*)
20:     character NCIDPN(*)*16
21:     C
22:     ..... data for region name matching .....
23:     integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ),
24:     & ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
25:     & KSUZN(NINPZ,2), IRGSP(2,NREG)
26:     integer ITRNM(NTREG), IPTRG(NTREG+1), LPTRG(*), KR(NRRRO)
27:     character TNAMS(NNAMES)*LNAM
28:     C
29:     ..... local variable .....
30:     integer KN(11)
31:     character LINE*72, CHAR*4, NM*5
32:     C
33:     C-----
34:     C
35:     C ..... check and initialize .....
36:     if ( NUCPN.le.0 ) go to 901
37:     if ( NREG.le.0 ) go to 903
38:     if ( NMTPN.le.0.or.NMTPN.gt.135 ) go to 925
39:     NPMT = 0
40:     JSKIP = 0
41:     NN = 2 * NUCPN * NREG
42:     if ( LPMT.eq.0 )
43:     & call KEPV( A(1), 'PMT', LPMT, NN, 'R4', LAST, 0.0, JDEBG )
44:     C
45:     C =====
46:     C region-wise form: PMT( !region-name( ( nuclide, reaction, proba. ) ... )
47:     C
48:     C PMT( !region-name( nuclide, reaction, proba. ) ... )
49:     C PMT( !region-name( reaction, proba. ) ... )
50:     C nuclide-wise form: PMT( ( nuclide, reaction, proba. ) ... )
51:     C all-nuclide form: PMT( reaction, probability )
52:     C =====
53:     100 call FPROBE( IPP, '(', LINE )
54:     if ( IPP.eq.0 ) then
55:     call GTLINE( IEND )
56:     if ( IEND.ne.0 ) go to 905
57:     go to 100
58:     end if
59:     CHAR(1:2) = '()'
60:     if ( LINE(1:1).ne.'!' ) go to 300
61:     C
62:     >>> !region-name( ... )
63:     C
64:     110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
65:     C
66:     if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.' ' ) go to 500 ! end
67:     of input
68:     if ( NLEN.eq.0 ) then
69:     if ( IEND.eq.0 ) go to 110
70:     go to 907
71:     end if
72:     NR = 0
73:     IK = INDEX(LINE(:NLEN),'@')
74:     if ( IK.gt.0 ) then
75:     C
76:     ..... @NAME without wildcard character .....
77:     if ( INDEX(LINE(:NLEN),'?').eq.0 .and.
78:     & INDEX(LINE(:NLEN),'*').eq.0 ) then
79:     call CBSINC( TNAMS, NNAMES, LINE(IK:NLEN), IPOS )
80:     if ( IPOS.eq.0 ) then
81:     write(IMG,910) LINE(IK:NLEN)
82:     call CNTERR( 'FATAL' )
83:     call DMREAD( ' ', ID1, N1, IIER )
84:     if ( IIER.ne.0 ) go to 911
85:     go to 110
86:     end if
87:     do I = 1, NTREG
88:     if ( ITRNM(I).lt.0.and.IPOS.eq.abs(ITRNM(I)) ) go to 120
89:     end do
90:     go to 913
91:     120
92:     IRD = I
93:     if ( JSKIP.ne.0 ) then
94:     call DMREAD( ' ', ID1, N1, IIER )
95:     if ( IIER.ne.0 ) go to 911
96:     go to 110
97:     end if
98:     do K = IPTRG(IRD), IPTRG(IRD+1)-1
99:     NR = NR + 1
100:    KR(NR) = LPTRG(K)
101:    end do
102:    C
103:    ..... @NAME with wildcard character .....
104:    else
105:    do J = 1, NTREG
106:    if ( ITRNM(J).lt.0 ) then
107:    if ( IMATCH(LINE(IK:NLEN),TNAMS(abs(ITRNM(J)))) .ne.0 )
108:    & then
109:    do 130 I = IPTRG(J), IPTRG(J+1)-1
110:    do K = 1, NR
111:    if ( KR(K).eq.LPTRG(I) ) go to 130
112:    end do
113:    NR = NR + 1
114:    KR(NR) = LPTRG(I)
115:    130
116:    continue
117:    end if
118:    end if
119:    end do
120:    end if
121:    C
122:    ..... regular region name .....
123:    else if ( IK.eq.0 ) then
124:    IBEGIN = 0
125:    JEND = 0
126:    150
127:    continue
128:    call RGFIND( IMG, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
129:    &
130:    & JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
131:    & KSUZN, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
132:    & NNAMES )

```

src/mvp/pmtinp.f

```

128:         if ( IERR.ne.0 ) then
129:             write(IMG,916) LINE(:NLEN)
130:             call CNTERR( 'FATAL' )
131:             call DMREAD( ' ', IDM1, N1, IIER )
132:             if ( IIER.ne.0 ) go to 911
133:             go to 110
134:         end if
135:         if ( IREG.ne.0 ) then
136:             do K = 1, NR
137:                 if ( KR(K).eq.IREG ) go to 150
138:             end do
139:             NR = NR + 1
140:             KR(NR) = IREG
141:         end if
142:         if ( IEND.eq.0 ) go to 150
143:     end if
144: C
145:     if ( ITERM.eq.0 ) then
146: 155     call CHREAD( ' ', LINE, '(', NLEN, ITERM, IEND )
147:         if ( IEND.ne.0 ) go to 917
148:         if ( ITERM.eq.0 ) go to 155
149:     end if
150: C
151: C     ..... read the top of data block .....
152:     ND = 0
153: 160 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
154:     if ( IEND.ne.0 ) go to 917
155:     if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.' ' ) then
156:         if ( ND.eq.0 ) go to 921
157:         go to 110
158:     end if
159: C
160: C     ..... many data blocks : ((nuclide,reaction,proba.)...) .....
161:     if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.' ' ) then
162: C     ..... read the nuclide name .....
163:         call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
164:         if ( IEND.ne.0 ) go to 919
165:         if ( ITERM.ne.0 ) go to 921
166:         if ( NLEN.eq.5 ) then
167:             NM = LINE(:NLEN)
168:         else if ( NLEN.eq.3 ) then
169:             NM = LINE(:NLEN) // ' '
170:         else if ( NLEN.eq.2 ) then
171:             NM = LINE(:NLEN) // ' '
172:         else if ( NLEN.eq.1 ) then
173:             NM = LINE(:NLEN) // '0 '
174:         else
175:             go to 929
176:         end if
177:         IKW = 0
178:         if ( NM(3:5).eq.'000' ) then
179:             IKW = 1
180:         else if ( NM(3:3).eq.'*' ) then
181:             IKW = 1
182:         else if ( NM(2:2).eq.'*' ) then
183:             IKW = 1
184:             NM(2:2) = '0'
185:         else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
186:             IKW = 1
187:         end if
188:         IINUCPN = 0
189:         if ( IKW.eq.0 ) then                ! single isotope
190:             do I = 1, NUCPN
191:                 if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 170
192:             end do

```

```

193:             go to 923
194: 170     KN(1) = I
195:             IINUCPN = 1
196:         else if ( IKW.eq.1 ) then          ! element
197:             II = 0
198: 180     do I = II+1, NUCPN
199:                 if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 185
200:             end do
201:             if ( IINUCPN.le.0 ) go to 923
202:             go to 190
203: 185     IINUCPN = IINUCPN + 1
204:             KN(IINUCPN) = I
205:             if ( I.lt.NUCPN ) then
206:                 II = I
207:                 go to 180
208:             end if
209: 190     continue
210:         end if
211: C     ..... read the reaction type (mt) .....
212:         call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
213:         if ( IEND.ne.0 ) go to 919
214:         if ( ITERM.ne.0 ) go to 921
215:         if ( NLEN.le.0 ) go to 929
216:         read(LINE(:NLEN),*) MTPN0
217:         if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
218: C     ..... read the probability .....
219:         call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
220:         if ( IEND.ne.0 ) go to 919
221:         if ( NLEN.le.0 ) go to 929
222:         read(LINE(:NLEN),*) PRB0
223:         if ( PRB0.lt.0.0.or.PRB0.gt.1.0 ) PRB0 = 0
224:         ND = ND + 1
225: C     ..... store the probability .....
226:         if ( PRB0.gt.0.0 ) then
227:             do K = 1, NR
228:                 I1 = NUCPN * (KR(K)-1)
229:                 do I = 1, IINUCPN
230:                     I2 = 2 * (I1 + KN(I) - 1) - 1
231:                     A(LPMT+I2+1) = MTPN0
232:                     A(LPMT+I2+2) = PRB0
233:                 end do
234:             end do
235:             NPMT = 1
236:         end if
237: C
238: C     ..... single data block : (nuclide,reaction,proba.) .....
239:     else
240: C     ..... check and read the nuclide name .....
241:         if ( NLEN.le.0 ) go to 929
242:         K = INDEX('ABCDEFGHIJKLMNQRSTUUVWXYZ',LINE(1:1))
243:         if ( K.gt.0 ) then
244:             if ( NLEN.eq.5 ) then
245:                 NM = LINE(:NLEN)
246:             else if ( NLEN.eq.3 ) then
247:                 NM = LINE(:NLEN) // ' '
248:             else if ( NLEN.eq.2 ) then
249:                 NM = LINE(:NLEN) // ' '
250:             else if ( NLEN.eq.1 ) then
251:                 NM = LINE(:NLEN) // '0 '
252:             else
253:                 go to 929
254:             end if
255:             IKW = 0
256:             if ( NM(3:5).eq.'000' ) then
257:                 IKW = 1

```


src/mvp/pmtinp.f

```

258:         else if ( NM(3:3).eq.'*' ) then
259:             IKW = 1
260:         else if ( NM(2:2).eq.'*' ) then
261:             IKW = 1
262:             NM(2:2) = '0'
263:         else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
264:             IKW = 1
265:         end if
266:         IINUCPN = 0
267:         if ( IKW.eq.0 ) then                ! single isotope
268:             do I = 1, NUCPN
269:                 if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 200
270:             end do
271:             go to 923
272:             KN(1) = I
273:             IINUCPN = 1
274:         else if ( IKW.eq.1 ) then            ! element
275:             II = 0
276:             do I = II+1, NUCPN
277:                 if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 215
278:             end do
279:             if ( IINUCPN.le.0 ) go to 923
280:             go to 220
281:             IINUCPN = IINUCPN + 1
282:             KN(IINUCPN) = I
283:             if ( I.lt.NUCPN ) then
284:                 II = I
285:                 go to 210
286:             end if
287:             continue
288:         end if
289:         else                                  ! nuclide name is undefined --> all nuclides
in the region
290:             IINUCPN = NUCPN
291:             do I = 1, NUCPN
292:                 KN(I) = I
293:             end do
294:             go to 230
295:         end if
296: C      ..... read the reaction type (mt) .....
297:         call CHREAD( ' ', LINE, ' '), NLEN, ITERM, IEND )
298:         if ( IEND.ne.0 ) go to 919
299:         if ( ITERM.ne.0 ) go to 921
300:         if ( NLEN.le.0 ) go to 929
301:         230 continue
302:         read(LINE(:NLEN),*) MTPNO
303:         if ( MTPNO.le.0.or.MTPNO.gt.NMTPN ) go to 927
304: C      ..... read the probability .....
305:         call CHREAD( ' ', LINE, ' '), NLEN, ITERM, IEND )
306:         if ( IEND.ne.0 ) go to 919
307:         if ( NLEN.le.0 ) go to 929
308:         read(LINE(:NLEN),*) PRB0
309:         if ( PRB0.lt.0.0.or.PRB0.gt.1.0 ) PRB0 = 0
310:         ND = ND + 1
311: C      ..... store the probability .....
312:         if ( PRB0.gt.0.0 ) then
313:             do K = 1, NR
314:                 I1 = NUCPN * (KR(K)-1)
315:                 do I = 1, IINUCPN
316:                     I2 = 2 * (I1 + KN(I) - 1) - 1
317:                     A(LPMT+I2+1) = MTPNO
318:                     A(LPMT+I2+2) = PRB0
319:                 end do
320:             end do
321:             NPMT = 1
322:         end if
323:         if ( ITERM.gt.0 ) go to 110
324:         240 call CHREAD( ' ', LINE, ' '), NLEN, ITERM, IEND )
325:         if ( IEND.ne.0 ) go to 919
326:         if ( ITERM.eq.0 ) go to 240
327:         go to 110
328:     end if
329:     go to 160
330: C
331: C >>> nuclide-wise for all regions; (nuclide,reaction,probability)
332: C
333: 300 continue
334:     K = INDEX('ABCDEFGHIJKLMNOPQRSTUVWXYZ',LINE(1:1))
335:     if ( K.eq.0 ) go to 400
336: 310 continue
337:     call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
338:     if ( IEND.ne.0 ) go to 931
339:     if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.' ' ) go to 500
340: C      ..... read the nuclide name .....
341:     call CHREAD( ' ', LINE, ' '), NLEN, ITERM, IEND )
342:     if ( IEND.ne.0 ) go to 919
343:     if ( ITERM.ne.0 ) go to 921
344:     if ( NLEN.eq.5 ) then
345:         NM = LINE(:NLEN)
346:     else if ( NLEN.eq.3 ) then
347:         NM = LINE(:NLEN) // ' '
348:     else if ( NLEN.eq.2 ) then
349:         NM = LINE(:NLEN) // ' '
350:     else if ( NLEN.eq.1 ) then
351:         NM = LINE(:NLEN) // '0 '
352:     else
353:         go to 929
354:     end if
355:     IKW = 0
356:     if ( NM(3:5).eq.'000' ) then
357:         IKW = 1
358:     else if ( NM(3:3).eq.'*' ) then
359:         IKW = 1
360:     else if ( NM(2:2).eq.'*' ) then
361:         IKW = 1
362:         NM(2:2) = '0'
363:     else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
364:         IKW = 1
365:     end if
366:     IINUCPN = 0
367:     if ( IKW.eq.0 ) then                ! single isotope
368:         do I = 1, NUCPN
369:             if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 320
370:         end do
371:         go to 923
372:         320 KN(1) = I
373:         IINUCPN = 1
374:     else if ( IKW.eq.1 ) then            ! element
375:         II = 0
376:         do I = II+1, NUCPN
377:             if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 335
378:         end do
379:         if ( IINUCPN.le.0 ) go to 923
380:         go to 340
381:         335 IINUCPN = IINUCPN + 1
382:         KN(IINUCPN) = I
383:         if ( I.lt.NUCPN ) then
384:             II = I
385:             go to 330
386:         end if

```

src/mvp/pmtinp.f

```

387: 340 continue
388: end if
389: C ..... read the reaction type (mt) .....
390: call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
391: if ( IEND.ne.0 ) go to 919
392: if ( ITERM.ne.0 ) go to 921
393: if ( NLEN.le.0 ) go to 929
394: read(LINE(:NLEN),*) MTPN0
395: if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
396: C ..... read the probability .....
397: call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
398: if ( IEND.ne.0 ) go to 919
399: if ( NLEN.le.0 ) go to 929
400: read(LINE(:NLEN),*) PRB0
401: if ( PRB0.lt.0.0.or.PRB0.gt.1.0 ) PRB0 = 0
402: C ..... store the probability .....
403: if ( PRB0.gt.0.0 ) then
404:   do K = 1, NREG
405:     I1 = NUCPN * (K - 1)
406:     do I = 1, I1NUCPN
407:       I2 = 2 * (I1 + KN(I) - 1) - 1
408:       A(LPMT+I2+1) = MTPN0
409:       A(LPMT+I2+2) = PRB0
410:     end do
411:   end do
412:   NPMT = 1
413: end if
414: if ( ITERM.gt.0 ) go to 310
415: 350 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
416: if ( IEND.ne.0 ) go to 919
417: if ( ITERM.eq.0 ) go to 350
418: go to 310
419: C
420: C >>> common data for all nuclides and regions; (reaction,probability)
421: C
422: 400 continue
423: C ..... read the reaction type (mt) .....
424: call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
425: if ( IEND.ne.0 ) go to 933
426: if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.' ' ) go to 935
427: if ( NLEN.le.0 ) go to 929
428: read(LINE(:NLEN),*) MTPN0
429: if ( MTPN0.le.0.or.MTPN0.gt.NMTPN ) go to 927
430: C ..... read the probability .....
431: call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
432: if ( IEND.ne.0 ) go to 933
433: if ( NLEN.le.0 ) go to 929
434: read(LINE(:NLEN),*) PRB0
435: if ( PRB0.lt.0.0.or.PRB0.gt.1.0 ) PRB0 = 0
436: C ..... store the probability .....
437: if ( PRB0.gt.0.0 ) then
438:   do K = 1, NREG
439:     I1 = NUCPN * (K - 1)
440:     do I = 1, NUCPN
441:       I2 = 2 * (I1 + I - 1) - 1
442:       A(LPMT+I2+1) = MTPN0
443:       A(LPMT+I2+2) = PRB0
444:     end do
445:   end do
446:   NPMT = 1
447: end if
448: if ( ITERM.gt.0 ) go to 500
449: 410 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
450: if ( IEND.ne.0 ) go to 933
451: if ( ITERM.eq.0 ) go to 410

```

```

452: C
453: C >>> end of input
454: C
455: 500 continue
456: return
457: C
458: C ..... error process .....
459: 901 write(IMSG,902)
460: call PRSTOP( 1, 'Problem in order of input data.' )
461: go to 999
462: 903 write(IMSG,904)
463: call PRSTOP( 1, 'Problem in order of input data.' )
464: go to 999
465: 905 write(IMSG,906)
466: call PRSTOP( 1, 'Unexpected end of input file.' )
467: go to 999
468: 907 write(IMSG,908)
469: call PRSTOP( 1, 'Unexpected end of input file.' )
470: go to 999
471: 911 write(IMSG,912)
472: call PRSTOP( 1, 'Error in region dependent data input.' )
473: go to 999
474: 913 write(IMSG,914) (ITRNM(I),I=1,NTREG)
475: call PRSTOP( 1, 'Problem in region dependent data input.' )
476: go to 999
477: 917 write(IMSG,918)
478: call PRSTOP( 1, 'Unexpected end of input file.' )
479: go to 999
480: 919 write(IMSG,920)
481: call PRSTOP( 1, 'Unexpected end of input file.' )
482: go to 999
483: 921 write(IMSG,922)
484: call PRSTOP( 1, 'Incomplete data structure of input file.' )
485: go to 999
486: 923 write(IMSG,924) NM, NLEN
487: call PRSTOP( 1, 'Unknown uclide name in input file.' )
488: go to 999
489: 925 write(IMSG,926)
490: call PRSTOP( 1, 'Problem in order of input data.' )
491: go to 999
492: 927 write(IMSG,928) MTPN0
493: call PRSTOP( 1, 'Incomplete data structure of input file.' )
494: go to 999
495: 929 write(IMSG,930) NLEN
496: call PRSTOP( 1, 'Incomplete data structure of input file.' )
497: go to 999
498: 931 write(IMSG,932)
499: call PRSTOP( 1, 'Unexpected end of input file.' )
500: go to 999
501: 933 write(IMSG,934)
502: call PRSTOP( 1, 'Unexpected end of input file.' )
503: go to 999
504: 935 write(IMSG,936)
505: call PRSTOP( 1, 'Incomplete data structure of input file.' )
506: 999 stop 888
507: 902 format(/' XXX(pmtinp) Number of "photonuclear nuclide" (NUCPN)',
508:   & ' is not determined before photonuclear dependent data input.'
509:   & /' Your input for <PMT> must be placed after',
510:   & ' $XSEC block.')
511: 904 format(/' XXX(pmtinp) Number of "region" (NREG) is not determi',
512:   & 'ned before REGION dependent data input.'
513:   & /' Your input for <PMT> must be placed after',
514:   & ' $GEOMETRY block.')
515: 906 format(/' XXX(pmtinp) End of file is encountered for <PMT>.')
516: 908 format(/' XXX(pmtinp) End of file is encountered during',

```

src/mvp/pmtinp.f

```

517:      & ' "!region-name(...)" search for <PMT>.' )
518: 910 format(// 'XXX(pmtinp) Tally region <','a,>' does not exist.')
519:      &/'      Data of <PMT> for this region are meaningless.')
520: 912 format(// 'XXX(pmtinp) Unexpected end of data or input error',
521:      & ' during skipping unnecessary or invalid data.'
522:      &/'      (REGION dependent data input)')
523: 914 format(// 'XXX(pmtinp) Fatal error for tally-region name',
524:      & ' (ITRNM).'
525:      &/'(i6,i10))
526: 916 format(// 'XXX(pmtinp) Region <','a,>' is not defined or invalid',
527:      & ' name.'
528:      &/'      Data of <PMT> for this region are meaningless.')
529: 918 format(// 'XXX(pmtinp) End of file is encountered during data',
530:      & ' in "!region-name(nuclide, reaction, proba.)..." or '
531:      &/'      "!region-name(nuclide, reaction, proba.)" for <PMT>.' )
532: 920 format(// 'XXX(pmtinp) End of file is encountered during data',
533:      & ' in "(nuclide, reaction, proba.)" for <PMT>.' )
534: 922 format(// 'XXX(pmtinp) Structure of data block is incomplete in',
535:      & ' "(nuclide, reaction, proba.)" for <PMT>.' )
536: 924 format(// 'XXX(pmtinp) Nuclide name is not existed in nuclide',
537:      & ' name list for <PMT>.'
538:      &/'      name=' ,a, ' length=' ,i3)
539: 926 format(// 'XXX(pmtinp) Number of reactions on "photonuclear',
540:      & ' nuclide" (NMTFN) is not determined before photonuclear',
541:      & ' dependent data input.'
542:      &/'      Your input for <PMT> must be placed after',
543:      & ' $XSEC block.')
544: 928 format(// 'XXX(pmtinp) Reaction type for photonuclear is out',
545:      & ' of range (1 to 135) for <PMT>.'
546:      &/'      MTPN0=' ,i5)
547: 930 format(// 'XXX(pmtinp) Length of data in "(nuclide, reaction, ',
548:      & ' proba.)" is illegal for <PMT>.'
549:      &/'      NLEN=' ,i5)
550: 932 format(// 'XXX(pmtinp) End of file is encountered during data',
551:      & ' in "(nuclide, reaction, proba.)..." for <PMT>.' )
552: 934 format(// 'XXX(pmtinp) End of file is encountered during data',
553:      & ' in "(reaction, proba.)" for <PMT>.' )
554: 936 format(// 'XXX(pmtinp) Structure of data block is incomplete in',
555:      & ' "(reaction, proba.)" for <PMT>.' )
556:      end
557: c##
558:      subroutine PMTWRT( PMT, NUCPN, NREG, IPR )
559: C=<MVP>=====
560: C purpose: print array data for probability for a specified reaction
561: C          type (mt) by region and nuclide in photonuclear reaction.
562: C called in: INTR02
563: C calls:
564: C=====
565: C
566: C      .... data for photonuclear ....
567: C      real PMT(2,NUCPN,NREG)
568: C
569: C      .... local variable ....
570: C      character HL1*120,HL2*120,HL0(8)*15,HL*15
571: C      equivalence (HL0,HL2)
572: C
573: C-----
574: C
575: C      ..... initialize .....
576: C      INEG      = 0
577: C      IEMT      = 0
578: C
579: C      ..... print the reading data and check .....
580: C      write(IPR,600) NUCPN,NREG
581: C      IS        = 8

```

```

582:      do I = 1, NUCPN, IS
583:      I1      = min(I+IS-1, NUCPN)
584:      write(IPR,601) ('-----',I2=I,I1)
585:      write(IPR,602) (I2,I2=I,I1)
586:      write(IPR,601) ('-----',I2=I,I1)
587:      HL1      = ' '
588:      ISAME1    = 0
589:      do J = 1, NREG
590:      HL2      = ' '
591:      I3        = 0
592:      do I2 = I, I1
593:      I3        = I3 + 1
594:      MT3        = nint(PMT(1,I2,J))
595:      write(HL,'(i4,lp,e10.3)') MT3,PMT(2,I2,J)
596:      HL(5:5)    = '('
597:      HL(15:15)  = ')'
598:      HL0(I3)    = HL
599:      if ( MT3.lt.0.or.MT3.gt.135 ) IEMT = IEMT + 1
600:      if ( PMT(2,I2,J).lt.0.0.or.PMT(2,I2,J).gt.1.0 )
601: 1          INEG = INEG + 1
602:      end do
603:      I3        = I3 * 15
604:      if ( HL1(1:I3).eq.HL2(1:I3) ) then
605:      if ( ISAME1.eq.0 ) ISAME1 = J
606:      ISAME2    = J
607:      else
608:      if ( ISAME1.gt.0 ) then
609:      write(IPR,604) ISAME1,ISAME2
610:      ISAME1    = 0
611:      end if
612:      write(IPR,603) J,HL2(1:I3)
613:      HL1      = HL2
614:      end if
615:      end do
616:      if ( ISAME1.gt.0 ) write(IPR,604) ISAME1,ISAME2
617:      write(IPR,601) ('-----',I2=I,I1)
618:      end do
619: C
620:      if ( IEMT.gt.0.or.INEG.gt.0 ) then
621:      write(IPR,901) IEMT,INEG
622:      call CNTERR( 'FATAL' )
623:      end if
624:      return
625: C
626: 600 format(// '<<< ARRAY PMT(1:2,I,J) (I=1,' ,i4, ' , J=1,' ,i4, ' ) >>>',
627:      & ' ..... form: MT(probability)')
628: 601 format(// '-----',8a15)
629: 602 format('      | NUCLIDE'
630:      &/'      /' REGION |',8(i10:5x))
631: 603 format(i8, ' |',a)
632: 604 format(5x,'... | ( region ',i5, ' to ',i5,
633:      & ' are same as above region )')
634: 901 format(// 'XXX(pmtwrt) Probabilities of forced photo-nuclear',
635:      & ' reaction type PMT includes ',i5, ' illegal MT and ',i5,
636:      & ' negative probabilities.')
637:      end

```

src/mvp/pn4lct2.f

```

1:      subroutine PN4LCT2( IOW, IRAND,
2:      N  NZONE, NREG, NBANK, NEVENT,NEST,  NUCPN, NUCPNI,MCXPN, NMTPN,
3:      A  KN,   MT,   IM,   I3,   I4,   NBINA, ATW,   MNC2,  EBRELA,
4:      A  PI2,   NUC,   NPATOM,NMT,   NMTP,  IREG,
5:      V  WGTPN, MXPGEN,EBOTX,
6:      X  MMAC,  CXPN,  ICXPN,  KLBPN2,XLBPN2,
7:      %  LSFFL, NFFL,  IZFFL,  LSDED, NDEAD,
8:      B  XXX,   YYY,   ZZZ,   AAA,   BBB,   CCC,   WWW,   EEE,   TTT,
9:      B  ITT,   XIM,   KLSF,  KKP,   IZZ,   IGG,   LEVL,  LZZ,   LPOS,
10:     B  LCRS,  IBREG,  IBSPC,  DBNK,  IBNK,  KDBNK,  MDBNK,  KIBNK,  MIBNK,
11:     T  KPNPRD,ENGYB,  NCNTR,  WCNTR,
12:     W  R,     DWK1,  WK1,   WK2,   WK3,   WK4,   WK5,   WK6,
13: c##<2007/03/14:PN4:
14: c## W  IWK1,  IWK4,  IWK5,  IWK8,  IWK9,  IWK10 )
15:     W  IWK1,  IWK4,  IWK5,  IWK8,  IWK9,  IWK10, KPNFG )
16: c##>
17: C=<MVP>=====
18: C purpose: generate particles from kinematics based on angular
19: C           distribution in photonuclear reactions.
20: C called in: PHOTNUC
21: C calls: RANU2, BSVDEC, BSDEC3, PNKINM
22: C=====
23: C           given data: WK1 ..... incident energy
24: C                       IWK1 ..... bank pointer
25: C                       IREG ..... region number
26: C                       DWK1 ..... adjustment weight
27: C
28: C           implicit real*8 (A-H,O-Z)
29: C           real*8 MNC2
30: C
31: C           include 'INC/_KPIDS'
32: C           include 'INC/_NGPS'
33: C           include 'INC/_FLAGS'
34: C           include 'INC/_KPMASS'
35: C
36: C ..... variance reduction data
37: C           real*8 EBOTX(KPLIM)
38: C           real WGTPN(NMTPN,NUCPN,NREG)
39: C
40: C ..... cross section data
41: C           real CXPN(MCXPN), XLBPN2(NUCPNI,NMTPN,2)
42: C           integer ICXPN(MCXPN), KLBPN2(NUCPNI,NMTPN,13)
43: C
44: C ..... sigma bank
45: C           integer MMAC(NBANK,2)
46: C
47: C ..... stack
48: C           integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK),
49: C           & LSDED(NBANK), NDEAD
50: C
51: C ..... particle bank
52: C           real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
53: C           & CCC(NBANK), TTT(NBANK), DBNK(NBANK,*)
54: C           real WWW(NBANK), EEE(NBANK), XIM(NBANK)
55: C           integer ITT(NBANK), KLSF(NBANK), KKP(NBANK), IZZ(NBANK),
56: C           & IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),LPOS(NBANK,NEST),
57: C           & LCRS(NBANK,NEST), IBREG(NBANK), IBSPC(NBANK,0:NEST),
58: C           & IBNK(NBANK,*), KDBNK(0:MDBNK), KIBNK(0:MIBNK),
59: C           & IREG(NBANK)
60: c##<2007/03/14:PN4:
61:     integer KPNFG(NBANK)
62: c##>
63: C
64: C ..... tally bank
65:     real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)

```

```

66:     real ENGYB(*)
67: C
68: C ..... working area
69:     real*8 DWK1(NBANK)
70:     real R(6*NBANK), WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
71:     & WK5(NBANK), WK6(NBANK)
72:     integer IWK1(NBANK), IWK4(NBANK), IWK5(NBANK), IWK8(NBANK),
73:     & IWK9(NBANK), IWK10(NBANK)
74: C
75: C ..... local data
76:     data Y5 / 1.0D0 /           ! neutron yield
77:     data KMMODE / 1 /           ! calculation mode of kinematics (1 or 2)
78:     data AMP / 0.0D0 /           ! mass of incident photon
79:     data AWP / 1.0D0 /           ! mass of producing neutron
80: C
81: C-----
82: C**** initialize
83: C-----
84: C
85:     KP      = KPNEUT
86:     NBINA1  = NBINA + 1
87: C
88: C-----
89: C**** particle generation process
90: C-----
91: C
92:     call RANU2( IRAND, R(I3), IM, ICON )
93:     do I = I3, I4
94:         IP      = IWK1(I)
95:         if ( WGTPN(MT,KN,IREG(I)).le.0.0 ) then ! no control over generation
weight
96:             WK2(I) = WWW(IP) * DWK1(IP)           ! adjusted weight
97:             IWK4(I) = Y5 + R(I)                   ! net integer yield
98:         else                                         ! control over generation we
weight
99:             WP      = Y5 * WWW(IP) * DWK1(IP)           ! yield * adjusted wei
weight
100:             IWK4(I) = WP / WGTPN(MT,KN,IREG(I)) + R(I) ! net integer yield
101:             WK2(I)  = WGTPN(MT,KN,IREG(I))             ! photonuclear generat
ion weight
102:         end if
103:     end do
104: C
105: C ..... particle generation cutoff
106:     if ( MXPGEN.gt.0 ) then
107:         III      = 0
108:         WII      = 0
109:         do I = I3, I4
110:             IP      = IWK1(I)
111:             I5      = IWK4(I)
112:             if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.I5.gt.0 ) then
113:                 III  = III + I5
114:                 WII  = WII + I5 * WK2(I)
115:                 IWK4(I) = 0
116:             end if
117:         end do
118:     if ( III.gt.0 ) then
119:         NCNTR(27,KP) = NCNTR(27,KP) + III
120:         WCNTR(27,KP) = WCNTR(27,KP) + WII
121:     end if
122: end if
123: C
124: C ..... check the total number of babies
125:     III      = 0
126:     MA       = 0

```

src/mvp/pn4lct2.f

```

127:      do I = I3, I4
128:      III = III + IWK4(I)      ! total number
129:      MA = max(MA, IWK4(I))    ! maximum number per a subject
ion
130:      end do
131: C ..... too many babies. sorry, no possible method of birth control
132: C ..... currently. you are forced to play a sudden death game.
133:      if ( III.gt.NDEAD ) then
134:      write(IOW,911) KP, MT, III, NDEAD, IM, KN
135: C/#IF UNIX
136:      call FLUSHSTD()
137: C/#ENDIF
138:      stop 666
139:      end if
140:      if ( III.le.0 ) go to 2000      ! no baby
141: C ..... keep space in bank for babies
142:      IFFLI = NFFL(NZONE+1)
143:      KIII = 0
144:      do M = 1, MA
145:      *VOCL LOOP,NOVREC
146:      do I = I3, I4
147:      if ( IWK4(I).ge.M ) then
148:      KIII = KIII + 1
149:      LSFFL(IFFLI+KIII) = LSDED(NDEAD)
150:      NDEAD = NDEAD + 1
151:      IWK5(KIII) = IWK1(I)      ! copy bank pointer
152:      WK3(KIII) = WK2(I)      ! copy weight
153:      WK5(KIII) = WK1(I)      ! copy incident energy
154:      NCNTR(29,KP) = NCNTR(29,KP) + 1
155:      WCNTR(29,KP) = WCNTR(29,KP) + WK2(I)
156:      if ( KPNPRD.eq.0 .or. KPNPRD.eq.1 ) then
157:      if ( KIBNK(17).ne.0 )
158:      &      IBNK(LSFFL(IFFLI+KIII),KIBNK(17)) = IREG(I)      ! prod
159:      &
uce-region
160:      end if
161:      end if
162:      end do
163:      end do
164:      N2 = KIII * 2
165: C -----
166: C-----
167: C**** sampling of direction (cosine) from angular distribution in CM
168: C ..... and calculating of kinematics energy
169: C-----
170: C .....
171:      KCM2LAB = 1
172:      NEANG = KLBPN2(KN,MT,6)
173:      LSTF4 = KLBPN2(KN,MT,11)
174:      Q = XLBPN2(KN,MT,1)
175:      QTH = XLBPN2(KN,MT,2)
176:      Q = - Q      ! Q_value: opposite sign: same as ENDF
177:      QTH = abs(QTH)      ! threshold energy
178:      if ( NEANG.le.0 ) then
179:      write(IOW,912) KN, MT, NEANG, KIII
180:      stop 666
181:      end if
182:      X1 = 0
183:      CTH = 0
184:      do J = 1, KIII
185:      IWK8(J) = LSTF4
186:      IWK9(J) = NEANG
187:      end do
188:      call BSDEC3( CXP, IWK9, R, IWK8, WK5, IWK10, KIII )      ! for
angular distribution
189:      call RANU2( IRAND, R, KIII*3, ICON )
190: C .....
191: C ..... calculate the scattering angle cosine from equal-probability
192: C ..... bins of angular distribution
193:      do J = 1, KIII
194:      IP = LSFFL(IFFLI+J)
195:      if ( IWK10(J).eq.0 ) IWK10(J) = 1
196:      L2 = IWK8(J) + IWK10(J)
197:      L1 = L2 - 1
198:      INTE = mod(ICXPN(L2+IWK9(J)), 10)
199:      if ( INTE.eq.5 .or. INTE.eq.3 ) then      ! log interpolation
200:      X1 = log(CXPN(L1)/WK5(J)) / log(CXPN(L1)/CXPN(L2))
201:      else      ! linear interpolation
202:      X1 = (CXPN(L1)-WK5(J)) / (CXPN(L1)-CXPN(L2))
203:      end if
204:      X1 = min(1.0D0, max(0.0D0, X1))
205: C/#IF ROUNDOFF(NEAREST)
206:      IT = min(1, int(R(J)+X1)) + IWK10(J)
207:      L3 = min(int(NBINA*R(KIII+J))+1, NBINA) + IWK8(J) +
208:      &      2*IWK9(J) + NBINA1*(IT-1)
209: C/#ELSE
210:      R1 = R(J) + X1
211:      IT = IWK10(J) + R1
212:      R2 = R(KIII+J) * NBINA + 1
213:      L3 = IWK8(J) + 2*IWK9(J) + NBINA1*(IT-1) + R2
214: C/#ENDIF
215:      XMU = (CXPN(L3)-CXPN(L3-1))*R(N2+J)+CXPN(L3-1)
216:      WK6(J) = XMU      ! scattering angle cos
217: C .....
218: C ..... calculate outgoing energy in CM by kinematics
219:      EIN = WK5(J)
220:      call PNKINM( KMODE, ATW, XMU, MNC2, EIN, QTH, AMP, AWP, EEE0,
221:      &      IOW )
222:      &      EEE(IP) = EEE0
223:      end do
224: C -----
225: C-----
226: C**** get energy and angle in LAB system using relativistic formula
227: C-----
228: C .....
229:      call RANU2( IRAND, R, KIII, ICON )
230: C .....
231: C ..... convert from CM to LAB
232:      if ( KCM2LAB.eq.1 ) then
233:      do J = 1, KIII
234:      IP = LSFFL(IFFLI+J)
235:      IP0 = IWK5(J)
236:      EE = EEE(IP)      ! outgoing energy in CM
237:      EIN = WK5(J)      ! incident energy in LAB
238:      XMU = WK6(J)      ! scattering angle cosine
239:      WK5(J) = EE      ! outgoing energy to determine igg
240: C ..... non relativistic
241:      if ( EIN.le.EBRELA ) then
242:      A1 = ATW + AMP
243:      EEE(IP) = EE + (EIN+2*A1*XMU*sqrt(EIN*EE)) / A1 / A1
244:      CTH = XMU * sqrt(EE/EEE(IP)) + sqrt(EIN/EEE(IP)) / A1
245: C ..... relativistic
246:      else
247:      E12 = EIN + MNC2 * (AMP + ATW)      ! energy of incident p
article and target nuclide
248:      B = sqrt(EIN*(EIN+2*AMP*MNC2)) / E12
249:      G = 1 / sqrt(1-B*B)
250:      ECM = E12 / G
251:      PMC2 = PMASS(KP) * MNC2

```

src/mvp/pn4lct2.f

```

252:          PMC4      = PMC2 ** 2
253:          E4CM      = EE + PMC2
254:          P4CMCX    = sqrt(E4CM*E4CM-PMC4) * XMU
255:          E4        = G * (E4CM + B * P4CMCX)      ! Lorentz transformati
on (energy)
256:          P4CX      = G * (P4CMCX + B * E4CM)      ! Lorentz transformati
on
257:          CTH       = P4CX / sqrt(E4*E4-PMC4)
258:          EEE(IP)   = E4 - PMC2                  ! relativistic outgoin
g energy in Lab
259:          end if
260: C ..... make direction vector
261:          AAAAA0    = AAA(IP0)
262:          BBBBB0    = BBB(IP0)
263:          CCCCC0    = CCC(IP0)
264:          SINPSI    = sqrt(1-CTH*CTH)
265:          ETA       = PI2 * R(J)
266:          SINETA    = sin(ETA)
267:          COSETA    = cos(ETA)
268:          STHETA    = 1 - AAAAA0 * AAAAA0
269: *VOCL STMT,IF(100)
270:          if ( STHETA.gt.0.0 ) then
271:              STHETA = sqrt(STHETA)
272:              COSPHI = BBBBB0 / STHETA
273:              SINPHI = CCCCC0 / STHETA
274:          else
275:              STHETA = 0
276:              COSPHI = 1
277:              SINPHI = 0
278:          end if
279:          BBBBB0 = BBBBB0 * CTH +
280:          &          AAAAA0 * COSPHI * COSETA * SINPSI -
281:          &          SINPHI * SINPSI * SINETA
282:          CCCCC0 = CCCCC0 * CTH +
283:          &          AAAAA0 * SINPHI * COSETA * SINPSI +
284:          &          COSPHI * SINPSI * SINETA
285:          AAAAA0 = AAAAA0 * CTH - COSETA * SINPSI * STHETA
286:          S      = sqrt(AAAAA0**2+BBBBB0**2+CCCCC0**2)
287:          AAA(IP) = AAAAA0 / S
288:          BBB(IP) = BBBBB0 / S
289:          CCC(IP) = CCCCC0 / S
290:          end do
291: C
292: C ..... make direction vector only, because energy and cosine was
293: C ..... already in LAB
294:          else if ( KCM2LAB.eq.0 ) then
295:              do J = 1, KIII
296:                  IP      = LSFFL(IFFLI+J)
297:                  IP0     = IWK5(J)
298:                  CTH     = WK6(J)
299:                  WK5(J)  = EEE(IP)
300:                  AAAAA0  = AAA(IP0)
301:                  BBBBB0  = BBB(IP0)
302:                  CCCCC0  = CCC(IP0)
303:                  SINPSI  = sqrt(1-CTH*CTH)
304:                  ETA     = PI2 * R(J)
305:                  SINETA  = sin(ETA)
306:                  COSETA  = cos(ETA)
307:                  STHETA  = 1 - AAAAA0 * AAAAA0
308: *VOCL STMT,IF(100)
309:                  if ( STHETA.gt.0.0 ) then
310:                      STHETA = sqrt(STHETA)
311:                      COSPHI = BBBBB0 / STHETA
312:                      SINPHI = CCCCC0 / STHETA
313:                  else

```

```

314:                      STHETA = 0
315:                      COSPHI = 1
316:                      SINPHI = 0
317:                  end if
318:                  BBBBB0 = BBBBB0 * CTH +
319:                  &          AAAAA0 * COSPHI * COSETA * SINPSI -
320:                  &          SINPHI * SINPSI * SINETA
321:                  CCCCC0 = CCCCC0 * CTH +
322:                  &          AAAAA0 * SINPHI * COSETA * SINPSI +
323:                  &          COSPHI * SINPSI * SINETA
324:                  AAAAA0 = AAAAA0 * CTH - COSETA * SINPSI * STHETA
325:                  S      = sqrt(AAAAA0**2+BBBBB0**2+CCCCC0**2)
326:                  AAA(IP) = AAAAA0 / S
327:                  BBB(IP) = BBBBB0 / S
328:                  CCC(IP) = CCCCC0 / S
329:              end do
330:          end if
331: C
332: C ..... check the energy of produced particle with bottom energy
333:          KJJJ = 0
334:          do J = 1, KIII
335:              IP      = LSFFL(IFFLI+J)
336:              if ( EEE(IP).gt.EBOTX(KP) ) then
337:                  KJJJ = KJJJ + 1
338:                  IWK8(KJJJ) = IP
339:                  IWK9(KJJJ) = J
340:              else
341:                  NDEAD = NDEAD + 1
342:                  LSDED(NDEAD) = IP
343:                  NCNTR(8,KP) = NCNTR(8,KP) + 1
344:                  WCNTR(8,KP) = WCNTR(8,KP) + WK3(J)
345:                  if ( KIBNK(17).ne.0 ) IBNK(IP,KIBNK(17)) = 0
346:                  if ( KIBNK(18).ne.0 ) IBNK(IP,KIBNK(18)) = 0
347:                  if ( KIBNK(19).ne.0 ) IBNK(IP,KIBNK(19)) = 0
348: c##<2007/03/14:PN4:
349:                  KPNFG(IP) = 0
350: c##>
351:              end if
352:          end do
353:          if ( KJJJ.le.0 ) then
354:              go to 2000                      ! no particle production
355:          else if ( KJJJ.lt.KIII ) then
356:              do J = 1, KJJJ
357:                  LSFFL(IFFLI+J) = IWK8(J)
358:                  IWK5(J)       = IWK5(IWK9(J))
359:                  WK3(J)        = WK3(IWK9(J))
360:                  WK5(J)        = WK5(IWK9(J))
361:              end do
362:              KIII = KJJJ
363:          end if
364: C
365: C ..... set energy group number for outgoing energy
366:          call BSVDEC( ENGYB(KENGP(KP)), NGP(KP)+1, WK5, IWK9, KIII)
367:          NNG      = KNGP(KP) - 1
368:          do J = 1, KIII
369:              IP      = LSFFL(IFFLI+J)
370:              IGG(IP) = NNG + IWK9(J)
371:          end do
372: C
373: C ..... copy or set particle attributies except of energy and directions
374:          do J = 1, KIII
375:              IP      = LSFFL(IFFLI+J)
376:              IP0     = IWK5(J)
377:              IBREG(IP) = IBREG(IP0)
378:              IZFFL(IFFLI+J) = IZZ(IP0)

```

src/mvp/pn4lct2.f

```

379:      IZZ(IP) = IZZ(IP0)
380:      KKP(IP) = KP
381:      WWW(IP) = WK3(J)
382:      XXX(IP) = XXX(IP0)
383:      YYY(IP) = YYY(IP0)
384:      ZZZ(IP) = ZZZ(IP0)
385:      TTT(IP) = TTT(IP0)
386:      KLSF(IP) = 0
387:      MMAC(IP,2) = 0
388:      if ( JLAT.ne.0 ) LEVL(IP) = LEVL(IP0)
389:      if ( JIMPT.ne.0 ) XIM(IP) = XIM(IP0)
390:      if ( JTIME.ne.0 ) ITT(IP) = ITT(IP0)
391:    end do
392:    if ( JLAT.ne.0 ) then
393:      do K = 1, NEST
394:        *VOCL LOOP,NOVREC
395:        do J = 1, KIII
396:          IP = LSFFL(IFFLI+J)
397:          IP0 = IWK5(J)
398:          IZZ(IP,K) = IZZ(IP0,K)
399:          LPOS(IP,K) = LPOS(IP0,K)
400:          if ( JHLAT.ne.0 ) LCRS(IP,K) = LCRS(IP0,K)
401:          if ( JTLT.ne.0 ) IBSPC(IP,K) = IBSPC(IP0,K)
402:        end do
403:      end do
404:    end if
405:    do K = 1, KDBNK(0)
406:      if ( K.eq.KDBNK(1) ) then
407:        *VOCL LOOP,NOVREC
408:        do J = 1, KIII
409:          IP = LSFFL(IFFLI+J)
410:          DBNK(IP,KDBNK(1)) = WWW(IP)      ! birth weight is curr
411:        end do
412:      else if ( K.eq.KDBNK(2) ) then
413:        *VOCL LOOP,NOVREC
414:        do J = 1, KIII
415:          IP = LSFFL(IFFLI+J)
416:          DBNK(IP,KDBNK(2)) = TTT(IP)      ! time of birth is jus
417:        end do
418:      else if ( K.eq.KDBNK(3) ) then
419:        *VOCL LOOP,NOVREC
420:        do J = 1, KIII
421:          IP = LSFFL(IFFLI+J)
422:          DBNK(IP,KDBNK(3)) = EEE(IP)      ! energy on birth
423:        end do
424:      else
425:        *VOCL LOOP,NOVREC
426:        do J = 1, KIII
427:          IP = LSFFL(IFFLI+J)
428:          IP0 = IWK5(J)
429:          DBNK(IP,KDBNK(K)) = DBNK(IP0,KDBNK(K))
430:        end do
431:      end if
432:    end do
433:    do K = 1, KIBNK(0)
434:      if ( K.eq.KIBNK(4) ) then
435:        *VOCL LOOP,NOVREC
436:        do J = 1, KIII
437:          IP = LSFFL(IFFLI+J)
438:          IP0 = IWK5(J)
439:          IBNK(IP,KIBNK(4)) = IBNK(IP0,KIBNK(4)) + 1      ! particle gen
440:        end do
441:      else if ( K.eq.KIBNK(5) ) then
442:        *VOCL LOOP,NOVREC
443:        do J = 1, KIII
444:          IP = LSFFL(IFFLI+J)
445:          IBNK(IP,KIBNK(5)) = 0      ! particle fli
446:        end do
447:      else if ( K.eq.KIBNK(17) ) then
448:        C --- stored in above ---
449:      else if ( K.eq.KIBNK(18) ) then
450:        if ( KPNPRD.eq.0 .or. KPNPRD.eq.1 ) then
451:          *VOCL LOOP,NOVREC
452:          do J = 1, KIII
453:            IP = LSFFL(IFFLI+J)
454:            IBNK(IP,KIBNK(18)) = NUC + NPATOM + KN      ! produce-nucl
455:          end do
456:        end if
457:      else if ( K.eq.KIBNK(19) ) then
458:        if ( KPNPRD.eq.0 .or. KPNPRD.eq.1 ) then
459:          *VOCL LOOP,NOVREC
460:          do J = 1, KIII
461:            IP = LSFFL(IFFLI+J)
462:            IBNK(IP,KIBNK(19)) = NMT + NMTP + MT      ! produce-reac
463:          end do
464:        end if
465:      else
466:        *VOCL LOOP,NOVREC
467:        do J = 1, KIII
468:          IP = LSFFL(IFFLI+J)
469:          IP0 = IWK5(J)
470:          IBNK(IP,KIBNK(K)) = IBNK(IP0,KIBNK(K))
471:        end do
472:      end if
473:    end do
474:    c##<2007/03/14:PN4:
475:    do J = 1, KIII
476:      IP = LSFFL(IFFLI+J)
477:      KPNFG(IP) = NUC + NPATOM + KN
478:    end do
479:    c##>
480:    C
481:    C ..... update IZFFL and NFFL
482:    do J = 1, KIII
483:      IZ = IZFFL(IFFLI+J)
484:      NFFL(IZ) = NFFL(IZ) + 1
485:    end do
486:    NFFL(NZONE+1) = NFFL(NZONE+1) + KIII
487:    C
488:    C ..... terminate of all process
489:    2000 continue
490:    C
491:    return
492:    C
493:    C ..... error messages
494:    911 format(' XXX(PN4LCT2) Too many particles are generated from',
495:      & ' photonuclear reaction. XXX'
496:      & ' Currently, the only remedy for this situation is to adjust',
497:      & ' NBANK() and NHIST() (or NHSUB()) in your input data.'
498:      & ' generated particle=',I3,' reaction MT=',I3,
499:      & ' generated number=',I8,' free space in bank=',I8
500:      & ' number of mother particle=',I8,' KN=',I5)
501:    912 format(' XXX(PN4LCT2) Number of incident energy points for',
502:      & ' angular distribution is less than zero. XXX'

```

src/mvp/pn4lct2.f

```
503:      &' generated particle=',I3,' reaction MT=',I3,  
504:      &' NEANG (=KLBN2(KN,MT,6))=',I4,' generated number=',I8)  
505:      end
```

SAE

src/mvp/pnbrinp.f

```

1:      subroutine PNBRINP( IMMSG, A, IA, LIMIT, LAST, JDEBG, NUCPN,
2:      & NPPNBR, LPPNBR, NCIDPN, MNUCPN, LPIDPN )
3: C=<MVP>=====
4: C purpose:  input array data for probability of forced sampling of
5: C             photonuclear reaction (PPNBR) for photon incident.
6: C called in:  INTRO
7: C calls:
8: C=====
9:      real A(LIMIT)
10:     integer IA(LIMIT), JDEBG(*)
11: C
12: C ..... Data for photonuclear .....
13: C integer MNUCPN(*), LPIDPN(*)
14: C character NCIDPN(*)*16
15: C
16: C ..... local variable .....
17: C character LINE*72, CHAR*4, NM*5
18: C
19: C-----
20: C
21: C ..... check and initialize .....
22: C if ( NUCPN.le.0 ) go to 901
23: C NPPNBR = 0
24: C NN = NUCPN
25: C if ( LPPNBR.eq.0 )
26: C & call KEVP( A(1), 'PPNBR', LPPNBR, NN, 'R1', LAST, -1.0, JDEBG
27: C
28: C=====
29: C general form:  PPNBR( prob( nuclide ... ) ... )
30: C single form:  PPNBR( prob )
31: C=====
32: C
33: C 100 call FPROBE( IPP, '(', LINE )
34: C if ( IPP.eq.0 ) then
35: C   call GTLINE( IEND )
36: C   if ( IEND.ne.0 ) go to 903
37: C   go to 100
38: C end if
39: C CHAR(1:2) = '()'
40: C
41: C >>> probability( nuclide ... )
42: C
43: C 110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
44: C
45: C PPNBR0 = 0
46: C if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.'') then ! end of input
47: C   if ( NLEN.gt.0 ) then
48: C     read(LINE(:NLEN),*) PPNBR0
49: C     if ( PPNBR0.ge.1.0 ) go to 921
50: C     if ( PPNBR0.le.0.0 ) go to 300
51: C     do I = 1, NUCPN
52: C       if ( A(LPPNBR+I-1).eq.-1.0 ) A(LPPNBR+I-1) = PPNBR0
53: C     end do
54: C   end if
55: C   go to 300
56: C end if
57: C if ( NLEN.eq.0 ) then
58: C   if ( IEND.eq.0 ) go to 110
59: C   go to 907
60: C end if
61: C read(LINE(:NLEN),*) PPNBR0
62: C if ( PPNBR0.ge.1.0 ) go to 921
63: C CM2016 if ( PPNBR0.le.0.0 ) go to 110
64: C if ( PPNBR0.le.0.0 ) PPNBR0 = -1.0
65: C
66: C ..... read the nuclide names .....
67: C ND = 0
68: C 170 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
69: C if ( IEND.ne.0 ) go to 909
70: C ND = ND + 1
71: C if ( NLEN.eq.5 ) then
72: C   NM = LINE(:NLEN)
73: C else if ( NLEN.eq.3 ) then
74: C   NM = LINE(:NLEN) // ' '
75: C else if ( NLEN.eq.2 ) then
76: C   NM = LINE(:NLEN) // ' '
77: C else if ( NLEN.eq.1 ) then
78: C   NM = LINE(:NLEN) // '0 '
79: C else
80: C   go to 929
81: C end if
82: C IKW = 0
83: C if ( NM(3:5).eq.'000' ) then
84: C   IKW = 1
85: C else if ( NM(3:3).eq.'*' ) then
86: C   IKW = 1
87: C else if ( NM(2:2).eq.'*' ) then
88: C   IKW = 1
89: C   NM(2:2) = '0'
90: C else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
91: C   IKW = 1
92: C end if
93: C if ( IKW.eq.0 ) then
94: C   do I = 1, NUCPN
95: C     if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 180
96: C   end do
97: C   write(IMSG,891) NM
98: C   call CNTERR( 'WARNING' )
99: C   ND = ND - 1
100: C   go to 190
101: C 180 A(LPPNBR+I-1) = PPNBR0
102: C else if ( IKW.eq.1 ) then
103: C   II = 0
104: C 184 do I = II+1, NUCPN
105: C   if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 185
106: C   end do
107: C   if ( II.le.0 ) then
108: C     write(IMSG,891) NM(1:2)
109: C     call CNTERR( 'WARNING' )
110: C     ND = ND - 1
111: C   end if
112: C   go to 190
113: C 185 A(LPPNBR+I-1) = PPNBR0
114: C   if ( I.lt.NUCPN ) then
115: C     II = I
116: C     go to 184
117: C   end if
118: C end if
119: C 190 continue
120: C if ( ITERM.eq.0 ) go to 170
121: C go to 110
122: C
123: C 300 continue
124: C do I = 1, NUCPN
125: C   if ( A(LPPNBR+I-1).le.0.0 ) then ! set to default(=0)
126: C     A(LPPNBR+I-1) = 0
127: C   else
128: C     NPPNBR = NPPNBR + 1
129: C   end if
130: C end do

```

src/mvp/pnbrinp.f

```
131:      return
132: C
133: C ..... warning message .....
134: 891 format(/' !!!(pnbrinp) Nuclide name <'a,> is undefined as',
135:      & ' photonuclear.')
136: C
137: C ..... error process .....
138: 901 write(IMG,902) NUCPN
139:      call PRSTOP( 1, 'Problem in order of input data.' )
140:      go to 999
141: 903 write(IMG,904)
142:      call PRSTOP( 1, 'Unexpected end of input file.' )
143:      go to 999
144: 907 write(IMG,908)
145:      call PRSTOP( 1, 'Unexpected end of input file.' )
146:      go to 999
147: 909 write(IMG,910)
148:      call PRSTOP( 1, 'Unexpected end of input file.' )
149:      go to 999
150: 921 write(IMG,922) PPNBR0
151:      call PRSTOP( 1, 'Illegal probability data in input file.' )
152:      go to 999
153: 929 write(IMG,930) NLEN
154:      call PRSTOP( 1, 'Incomplete data structure of input file.' )
155: 999 stop 888
156: 902 format(/' XXX(pnbrinp) Number of "photonuclear nuclide" (NUCPN)',
157:      & ' is not determined before photonuclear dependent data input.'
158:      & /'          Your input for <PPNBR> must be placed after',
159:      & ' $XSEC block.'
160:      & /'          NUCPN=',i8)
161: 904 format(/' XXX(pnbrinp) End of file is encountered for <PPNBR>.')
162: 908 format(/' XXX(pnbrinp) End of file is encountered during',
163:      & ' "probability(...)" search for <PPNBR>.')
164: 910 format(/' XXX(pnbrinp) End of file is encountered during data in',
165:      & ' "probability(...)" for <PPNBR>.')
166: 922 format(/' XXX(pnbrinp) Probability of forced photonuclear',
167:      & 'reaction is greater than 1 for <PPNBR>.'
168:      & /'          probability=',lpe12.5)
169: 930 format(/' XXX(pnbrinp) Length of data in nuclide name is',
170:      & ' illegal for <PPNBR>.'
171:      & /'          NLEN=',i5)
172:      end
```

src/mvp/pnkinm.f

```

1:      subroutine PNKINM( KMMODE, ATW, XMU, MNC2, EIN, Q, AMP, AWP, EEE,
2:      & IOW )
3: C=<MVP>=====
4: C purpose: calculate the outgoing particle energy by kinematics.
5: C called in: PHOTNUC, PN4LCT2
6: C calls:
7: C=====
8:      implicit real*8 (A-H,O-Z)
9: C
10:     real*8 MNC2
11: C
12: C-----
13: C
14: C [note] The mass is in neutron unit.
15: C
16: C ATW : mass of the target nuclide
17: C XMU : cosine of angle between incident and outgoing particle
18: C       flight direction
19: C MNC2 : neutron rest energy [eV]
20: C EIN : energy of incident particle [eV]
21: C Q : threshold energy [eV] when KMMODE=1
22: C     Q_value of different mass [eV] when KMMODE=2
23: C AMP : mass of the incident particle
24: C AWP : mass of the outgoing particle
25: C EEE : calculating energy of outgoing particle [eV]
26: C
27: C ..... photonutron formula by P. Vertes [IAEA INDC(HUN) 035, 2001]
28: C <Maybe reference system is the center-of-mass system.>
29: C if ( KMMODE.eq.1 ) then
30: C     ATW1 = ATW - 1
31: C     EY0 = 2 * ATW1 / (MNC2 * ATW * (EIN - Q))
32: C     EY1 = EIN ** 2 / (2 * MNC2 * ATW1)
33: C     EY2 = EIN * sqrt(EY0) * XMU / ATW
34: C     EEE = ATW1 * (EIN - Q - EY1 + EY2) / ATW
35: C
36: C ..... standard kinematics formula
37: C <Reference system is the center-of-mass system>
38: C else if ( KMMODE.eq.2 ) then
39: C     E3 = AMP * MNC2 ! mass energy of incident part
icle
40:     E1 = EIN + E3 ! full energy of incident part
icle
41:     E2 = ATW * MNC2 ! mass energy of target nuclid
e
42:     E12 = E1 + E2 ! total energy of this collisi
on in LAB
43:     B = sqrt(EIN * (EIN + 2*E3)) / E12
44:     G = 1 / sqrt(1 - B**2)
45:     ECM = E12 / G ! total energy in CM
46:     E4 = AWP * MNC2 / G ! mass energy of outgoing part
icle
47:     E2 = E2 / G
48:     E3T = E2 - E4 - Q
49:     EC1 = ECM**2 - E3T**2 + E4**2
50:     EEE = EC1 / (2 * ECM) - E4 ! energy of outgoing particle
in CM
51: C
52: C else
53: C     write(IOW,901) KMMODE
54: C     stop 666
55: C end if
56: 901 format(/' XXX(PNKINM) Undefined kinematics mode is given. XXX'
57: &/' KMMODE=',I5)
58: C
59:     return
60: end

```

src/mvp/pnunch.f

```

1:      subroutine PNUNCH( IOW, A, H, JNP, NUNCL, LUNCL,
2:      &
3:      &
4:      &
5:      &
6:      &
7:      &
8:      &
9:      &
10:     &
11:     &
12: C=<MVP=====
13: C purpose: calculation of uncollided contribution to next event
14: C detectors, from production particles by photonuclear
15: C reaction.
16: C called in: PHOTNUC
17: C calls:
18: C=====
19:      implicit real*8 (A-H,O-Z)
20: C
21:      include '../shared/INC/_SIZES'
22:      include '../shared/INC/_PGEOM'
23:      include '../shared/INC/_PTALY0'
24:      include 'INC/_FLAGS'
25:      include 'INC/_STACK'
26:      include 'INC/_KPIDIS'
27:      include 'INC/_CXSEC'
28:      include 'INC/_SBANK'
29: C
30:      real A(*)
31:      real H(*)
32: C
33: C ..... bank index for photonuclear particles
34:      integer LUNCL(NUNCL)
35: C
36: C ..... data for next event (point) estimator
37:      real*8 XPDET(NPLEN,NPDET)
38:      integer IPDET(NPDET,2), IPDT2(NEST,3,NPDET)
39: C
40: C ..... tally bin data
41:      real TIMEB(NTIME+1)
42: C
43: C ..... stack for imaginary particle
44:      integer IMSFL(IMPMAX), IMNFL(NZONE+1), IMZFL(IMPMAX),
45:      &
46:      &
47: C ..... particle bank (1:real particle, 2:imaginary particle)
48:      real*8 XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2),
49:      &
50:      &
51:      real WWW(NBANK,2), EEE(NBANK,2)
52:      integer IGG(NBANK,2), ITT(NBANK,2), LEVL(NBANK,2),
53:      &
54:      &
55:      real*8 DBNK(NBANK,NDBNK,2), KIBNK(0:MIBNK), KDBNK(0:MDBNK)
56:      integer IBNK(NBANK,NIBNK,2), KIBNK(0:MIBNK), KDBNK(0:MDBNK)
57:      real*8 OPTI(IMPMAX), PATH(IMPMAX)
58:      integer KDETP(IMPMAX), KLSFI(IMPMAX)
59: C
60: C ..... cross section and geometry data
61:      real SMACI(IMPMAX,MB,NSMACI)
62:      integer MMACI(IMPMAX,2), KZMAT(NZONE)
63: C
64: C ..... working area
65:      real*8 X(NBANK), Y(NBANK), Z(NBANK), A0(NBANK), B0(NBANK),
66:      &
67:      &
68:      &
69:      real W(NBANK), R(2*NBANK), SMCW(NBANK,NSMICI)
70:      integer IBP(NBANK), IT0(NBANK), IGI(NBANK)
71: C
72: C ..... constants
73:      real*8 CLIGHT
74:      parameter ( CLIGHT = 2.99792458D+10 )      ! light speed [cm/s]
75:      real*8 NMASS
76:      parameter ( NMASS = 1.6749286D-24 )      ! neutron mass [g]
77:      real*8 EECHRG
78:      parameter ( EECHRG = 1.60217733D-12 )      ! electron charge [erg/
79:      real*8 MC2
80:      parameter ( MC2 = NMASS*CLIGHT**2/EECHRG )      ! neutron's mc**2 in eV
81: C
82:      external RANU2, BS0ISD, LATUP2, LATDW2
83: C
84: C=====
85: C
86: C ..... initialize
87:      DPI2 = 2*ACOS(-1.0D0)
88:      DPI4I = 1/(4*ACOS(-1.0D0))
89:      DZERO = 0
90: C
91:      if ( JLATT.eq.0 ) then
92:         do I = 1, NUNCL
93:            X(I) = XXX(LUNCL(I),1)
94:            Y(I) = YYY(LUNCL(I),1)
95:            Z(I) = ZZZ(LUNCL(I),1)
96:            A0(I) = AAA(LUNCL(I),1)
97:            B0(I) = BBB(LUNCL(I),1)
98:            C0(I) = CCC(LUNCL(I),1)
99:         end do
100:      else
101:         JDIR = 1
102:         JLS = 0
103:         call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ, NZONE,
104:         &
105:         &
106:         &
107:         &
108:         &
109:         &
110:         end if
111: C
112: C-----
113: C**** loop over detectors
114: C-----
115: C
116:      do ND = 1, NPDET
117: C
118: C ..... obtain the distance to the detector
119:         DXDA = XPDET(3,ND)
120:         DYDA = XPDET(4,ND)
121:         DZDA = XPDET(5,ND)
122:         RMIN = XPDET(1,ND)
123:         RMI2 = RMIN**2
124:         RMAX = XPDET(2,ND)
125:         RMA2 = RMAX**2
126:         IZD = IPDET(ND,1)
127:         MAT = KZMAT(IZD)
128:         NBS = 0
129: C

```

src/mvp/pnunch1.f

```

130:      call RANU2( IRAND, R, NUNCL, ICON )
131: C
132:      do I = 1, NUNCL
133:          DAT = DXDA - X(I)
134:          DBT = DYDA - Y(I)
135:          DCT = DZDA - Z(I)
136:          DDI = DAT**2 + DBT**2 + DCT**2
137: C ..... calculate contribution with probability RMI2/DDI
138: C      ... when DDI > RMIN**2
139:          if ( DDI*R(I).gt.RMI2 ) then
140:              W(I) = 0
141:          elseif ( DDI.gt.RMI2 ) then
142:              W(I) = WWW(LUNCL(I),1) / RMI2 * DPI4I
143: C ..... bounded sphere approximation when DDI < RMAX**2
144:          elseif ( DDI.lt.RMA2.and.MAT.gt.0 ) then
145:              NBS = NBS + 1
146:              W(I) = WWW(LUNCL(I),1) * DPI4I
147: C ..... weight / (distance**2 / (4*pi)
148:          else
149:              W(I) = WWW(LUNCL(I),1) / DDI * DPI4I
150:          end if
151:          DI(I) = sqrt(DDI)
152: C
153:          if ( DI(I).gt.DZERO ) then
154:              AA(I) = - DAT / DI(I)
155:              BB(I) = - DBT / DI(I)
156:              CC(I) = - DCT / DI(I)
157:          else
158:              AA(I) = - A0(I)
159:              BB(I) = - B0(I)
160:              CC(I) = - C0(I)
161:          end if
162:      end do
163: C
164: C ..... apply angular distribution function to weight W
165:      KUNCL = 1
166: C ..... isotropic distribution
167:      if ( KUNCL.eq.0 ) then
168:          DUMAX = 1
169:          DUMIN = -1
170:          DFACT = 2 / (DUMAX - DUMIN)
171:          do I = 1, NUNCL
172:              if ( -CC(I).le.DUMAX.and.-CC(I).ge.DUMIN ) then
173:                  W(I) = W(I) * DFACT
174:              else
175:                  W(I) = 0
176:              end if
177:          end do
178:      else if ( KUNCL.eq.1 ) then
179: C ..... calculate factor by using the direction vector
180:          DELMU = 1.0D-1      ! assumption of contribution mu range
181:          CONT = 1 - DELMU / DPI2
182:          DFACT = 2 * DPI2 / DELMU
183:          do I = 1, NUNCL
184:              XMU = - (A0(I)*AA(I) + B0(I)*BB(I) + C0(I)*CC(I))
185:              if ( XMU.ge.CONT ) then
186:                  W(I) = W(I) * DFACT
187:              else
188:                  W(I) = 0
189:              end if
190:          end do
191:      end if
192: C
193: C ..... time cut off

194:      if ( JTIME.ne.0 ) then
195:          do I = 1, NUNCL
196:              DDE = EEE(LUNCL(I),1)
197:              VEL = CLIGHT * sqrt(DDE*(DDE+2*MC2)) / (DDE+MC2)
198:              TI(I) = TTT(LUNCL(I),1) * DI(I) / VEL
199:          end do
200:          if ( JPTIM.eq.0 ) then
201:              do I = 1, NUNCL
202:                  if ( TI(I).gt.TCUT ) then
203:                      W(I) = 0
204:                      TI(I) = TCUT
205:                  end if
206:              end do
207:          else
208:              do I = 1, NUNCL
209:                  if ( TI(I).ge.TCUT ) then
210:                      ITREP = TI(I) / TCUT
211:                      TI(I) = TI(I) - TCUT * ITREP
212:                  end if
213:              end do
214:          end if
215:          call BS0ISD( TIMEB, NTIME+1, TI, IT0, NUNCL )
216:          do I = 1, NUNCL
217:              IT0(I) = max(1,min(NTIME,IT0(I)))
218:          end do
219:      end if
220: C
221: C-----
222: C**** send particles to free-flight or lattice stack for imaginary
223: C particles --- check number of particles having non-zero angular
224: C factor ---
225: C-----
226: C
227:      NN = 0
228:      do I = 1, NUNCL
229:          if ( W(I).ne.0.0 ) then
230:              NN = NN + 1
231:              IBP(NN) = LUNCL(I)
232:              DI(NN) = DI(I)
233:              AA(NN) = AA(I)
234:              BB(NN) = BB(I)
235:              CC(NN) = CC(I)
236:              W(NN) = W(I)
237:              if ( JTIME.ne.0 ) IT0(NN) = IT0(I)
238:          end if
239:      end do
240: C
241:      LVL = IPDET(ND,2)
242:      if ( JLATT.ne.0.and.LVL.gt.0 ) then
243:          if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
244:          LP = 2 + 3*LVL
245:          DXDA = XPDET(LP+1,ND)
246:          DYDA = XPDET(LP+2,ND)
247:          DZDA = XPDET(LP+3,ND)
248:      end if
249: C
250:      200 continue
251:      if ( NN.gt.0 ) then
252:          NNN = min(NN,NDIMPT)
253:          if ( NNN.gt.0 ) then
254:              NDIMPT = NDIMPT - NNN
255:          C
256:          C ..... send imaginary particles to free-flight stack
257:          N = IMNFI(NZONE+1)
258:          N1 = NN - NNN

```

src/mvp/pnunc1.f

```

259:      do J = 1, NNN
260:        J1 = N1 + J
261:        IP = IMDED(NDIMPT+J)
262:        IMSFL(N+J) = IP
263:        IMZFL(N+J) = IZD
264:        XXX(IP,2) = DXDA
265:        YYY(IP,2) = DYDA
266:        ZZZ(IP,2) = DZDA
267:        AAA(IP,2) = AA(J1)
268:        BBB(IP,2) = BB(J1)
269:        CCC(IP,2) = CC(J1)
270:        WWW(IP,2) = W(J1)
271:        EEE(IP,2) = EEE(IBM(J1),1)
272:        IGG(IP,2) = IGG(IBM(J1),1)
273:        TTT(IP,2) = TTT(IBM(J1),1)
274:        PATH(IP) = DI(J1)
275:        OPTI(IP) = 0
276:        KDETP(IP) = ND
277:        KLSFI(IP) = 0
278:        if ( JTIME.ne.0 ) ITT(IP,2) = IT0(J1)
279:        if ( KIBNK(5).ne.0 ) IBNK(IP,KIBNK(5),2) = -1
280:        if ( KIBNK(17).ne.0 )
281:          &      IBNK(IP,KIBNK(17),2) = IBNK(IBM(J1),KIBNK(17),1)
282:        if ( KIBNK(18).ne.0 )
283:          &      IBNK(IP,KIBNK(18),2) = IBNK(IBM(J1),KIBNK(18),1)
284:        if ( KIBNK(19).ne.0 )
285:          &      IBNK(IP,KIBNK(19),2) = IBNK(IBM(J1),KIBNK(19),1)
286:        end do
287: C
288: C ..... in lattice geometry, the following parameter should be set.
289: C ... -----> LEVL, LZZI, LPOSI, LCRSI, (DBNK, IBNK when JFISX.ne.0)
290:      NN2 = NNN      ! number of remaining particles when JFISX.ne
.0.
291:      JSTG = 0      ! flag to indicate the detector is located in
STG region.
292:      if ( JLATT.ne.0 ) then
293:        call LATDW2( IOW, JDEBG, JHLAT, JFISX, IMPMAX, NZONE,
294:          &      NLATT, NCELL, NLBZ, NEST, DINF, DEPS,
295:          &      IMSFL(N+1), NNN, JSTG, NN2, IPDET(ND,2),
296:          &      IPDT2(1,1,ND), IPDT2(1,2,ND),
297:          &      IPDT2(1,3,ND), XPDET(3,ND), AAA(1,2),
298:          &      BBB(1,2), CCC(1,2), LEVL(1,2), LZZI,
299:          &      LPOSI, LCRSI, DBNK(1,1,2), KDBNK, MDBNK,
300:          &      IBNK(1,1,2), KIBNK, MIBNK, IMDED, NDIMPT,
301:          &      A(LSDA), A(LKZDA), A(LKZAA), A(LKZMAT),
302:          &      A(LKCELL), A(LKZLBZ), A(LMLBZZ),
303:          &      A(LCELSZ), A(LSZLAT), A(LIDLAT),
304:          &      A(LIPLAT), A(LKLATT), A(LKSLAT),
305:          &      A(LNVLAT), A(LIPCEL), A(LLTYP),
306:          &      A(LICTYP), A(LKDALT), A(LDALT),
307:          &      A(LKZREG), DW1, DW2, DW3, AA(N1+1),
308:          &      BB(N1+1), CC(N1+1), DI(N1+1), DW4, DW5,
309:          &      W(N1+1), IGI(N1+1) )
310:      end if
311: C ..... adjust particle numbers in stack and number of imaginary particle
312:      if ( JSTG.eq.0 ) then
313:        IMNFL(IZD) = IMNFL(IZD) + NN2
314:        IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
315:      else
316:        NL = IMNLT(NLBZ+1)
317:        do J = 1, NN2
318:          IMSLT(NL+J) = IMSFL(N+J)
319:          IMZLT(NL+J) = IZD
320:        end do
321:        IMNLT(IZD) = IMNLT(IZD) + NN2

```

```

322:        IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
323:      end if
324:      NN = NN - NNN
325:      NIMPT = NIMPT + NN2
326:    end if
327: C
328: C ..... prepare cross sections
329:      JAMXCT = 0
330:      JMIC = 1
331:      if ( JNP.eq.1 ) then
332:        call GMACNX( IRAND, NN2, IMSFL(N+1), MAT, JMIC, JAMXCT,
333:          &      JNP, JGAMM, JDEBG, IMPMAX, EEE(1,2), MB,
334:          &      NUC, NSTAL, NSMACI, SMACI, A(LLMACI),
335:          &      H(LKMICI), MMACI, NSMICI, H(LSMICI),
336:          &      A(LLMICI), H(LKSPII), H(LSGTLI), A(LLPDEN),
337:          &      A(LINUCT), A(LDENST), NMAT, NUC, NMT,
338:          &      A(LCX), MCX, A(LKLIB1), A(LKLIB2),
339:          &      A(LXLIB1), A(LCRES), A(LKCRSI), MCRES,
340:          &      NNCSI, A(LINCSI), DW1, DW2, DW3, DW4, DW5,
341:          &      AA(N1+1), BB(N1+1), CC(N1+1), R, SMCW )
342:      end if
343:      if ( JNP.eq.2 ) then
344:        call GMACNX( IRAND, NN2, IMSFL(N+1), MAT, JMIC, JAMXCT,
345:          &      JNP, JGAMM, JDEBG, IMPMAX, EEE(1,2), MB,
346:          &      NUC, NSTAL, NSMACI, SMACI, A(LLMACI),
347:          &      H(LKMICI), MMACI, NSMICI, H(LSMICI),
348:          &      A(LLMICI), H(LKSPII), H(LSGTLI), A(LLPDNP),
349:          &      A(LIATMT), A(LDNSTP), NMAT, NPATOM, NMTP,
350:          &      A(LCXP), MCXP, A(LKLB1), A(LKLB2),
351:          &      A(LXLB1), A(LCRES), A(LKCRSI), MCRES,
352:          &      NNCSI, A(LINCSI), DW1, DW2, DW3, DW4, DW5,
353:          &      AA(N1+1), BB(N1+1), CC(N1+1), R, SMCW )
354:      end if
355: C
356: C ..... bounded sphere approximation
357:      if ( MAT.gt.0.and.NBS.gt.0 ) then
358:        do J = 1, NN2
359:          IP = IMSFL(N+J)
360:          if ( PATH(IP).le.RMAX ) then
361:            RMD = SMACI(IP,MMACI(IP,1),1) * RMAX
362:            RMI = 1 / RMD
363:            C2 = RMA2 * (RMI*RMI*2 + (1+2*RMI)/(1-exp(RMD)))
364:            WWW(IP,2) = WWW(IP,2) / C2
365:          end if
366:        end do
367:      end if
368: C
369: C-----
370: C**** call tracking control routine for imaginary particles
371: C      (values of NIMPT, NDIMPT, IMNFL may be changed.)
372: C-----
373: C
374:      if ( NDIMPT.eq.0 ) then
375:        MODEL = 2
376:        call NXTEE( MODEL, A, H, NDIMPT, NIMPT )
377:      end if
378:      go to 200
379:    end if
380: C
381: end do
382: C
383: return
384: end

```

src/mvp/popara.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine POPARA
3: C=====
4: C PURPOSE: This routine is obsolete ( June 1997 )
5: C           Function is moved to MTSUMS.
6: C CALLED IN: CENTER
7: C CALLS PESUM8
8: C
9: C UPDATE:
10: C=====
11: C/IF PARA(VPP)
12: c   include '../shared/INC/_ARRAY'
13: c   include 'INC/_FLAGS'
14: c   include '../shared/INC/_IOUNIT'
15: c   include '../shared/INC/_TASKDT'
16: c   include '../shared/INC/_VPPPARA'
17: c   include '../shared/INC/_PTALY0'
18: c   include 'INC/_PTALY'
19: c   include 'INC/_PTALY2'
20: c   include 'INC/_SBANK'
21: c   include 'INC/_SIZES2'
22: c   include '../shared/INC/_SIZES'
23: c   include '../shared/INC/_STALY'
24: c
25: c!xocl processor p( mpe )
26: c   call PESUM8( TMACT(1), NTASK )
27: c
28: c   call PESUMI( NTHIST, 1 )
29: c   if ( JEIGN.eq.0 ) call PESUMI( NBATCH, 1 )
30: c
31: c   call PESUM8( A(LWSUM), 1 )
32: c   call PESUM8( A(LWLEK), 1 )
33: c   call PESUM8( A(LNCNTR), NEVENT*2 )
34: c   call PESUM8( A(LWCNTR), NEVENT*2 )
35: c
36: c   call PESUM8( A(LSFLTR), NGROUP*NTREG*2 )
37: c   call PESUM8( A(LSFLCL), NGROUP*NTREG*2 )
38: c
39: c   if ( JRESP.ne.0.and.NRESP.gt.0 ) then
40: c     call PESUM8( A(LSRETR), NTREG*NRESP*2 )
41: c     call PESUM8( A(LSRECL), NTREG*NRESP*2 )
42: c   end if
43: c
44: c   if ( NEMIC.gt.0 ) then
45: c     call PESUM8( A(LWCXTY), (NGROUP+NPKIND)*NTREG )
46: c     call PESUM8( A(LSRMIC), NGROUP*NTREG*NUC*NEMIC*2 )
47: c     call PESUM8( A(LRMICR), NPKIND*NTREG*NUC*NEMIC*2 )
48: c     call PESUM8( A(LXMIC), (NGROUP+NPKIND)*NTREG*NUC*NEMIC*2 )
49: c   end if
50: c
51: c   if ( NEMAC.gt.0 ) then
52: c     call PESUM8( A(LRMAC), NGROUP*NTREG*NEMAC*2 )
53: c     call PESUM8( A(LRMACR), NPKIND*NTREG*NEMAC*2 )
54: c     call PESUM8( A(LXMAC), (NGROUP+NPKIND)*NTREG*NEMAC*2 )
55: c   end if
56: c
57: c   if ( JRESP.ne.0.and.NSTAL.gt.0 ) then
58: c     call PESUM8( A(LSRSTR), NTREG*NSTAL*2 )
59: c     call PESUM8( A(LSRSCCL), NTREG*NSTAL*2 )
60: c   end if
61: c
62: c   if ( JEIGN.eq.1 ) then
63: c     call PESUM8( A(LXSOC), NBATCH )
64: c     call PESUM8( A(LXKEF), NBATCH*7 )
65: c   end if
66: c
67: c   if ( NSTALY.gt.0 ) then
68: c     call PESUM8( A(LETALY), NLETAL*2 )
69: c   end if
70: c
71: c   if ( JTIME.gt.0 ) then
72: c     call PESUM8( A(LTFLHS), NPKIND*2 )
73: c   end if
74: c
75: c   if ( JMNTR.ne.0 ) then
76: c     call PESUM8( A(LNCLSN), NGROUP*NREG )
77: c     call PESUM8( A(LWCLSN), NGROUP*NREG )
78: c     call PESUM8( A(LNLEAK), NGROUP*NREG )
79: c     call PESUM8( A(LWLEAK), NGROUP*NREG )
80: c     call PESUM8( A(LNABSB), NGROUP*NREG )
81: c     call PESUM8( A(LWABSB), NGROUP*NREG )
82: c     call PESUM8( A(LNECUT), NREG )
83: c     call PESUM8( A(LWECUT), NREG )
84: c     call PESUM8( A(LNTCUT), NGROUP*NREG )
85: c     call PESUM8( A(LWTCUT), NGROUP*NREG )
86: c     call PESUM8( A(LNKILD), NGROUP*NREG )
87: c     call PESUM8( A(LWKILD), NGROUP*NREG )
88: c     call PESUM8( A(LNSURV), NGROUP*NREG )
89: c     call PESUM8( A(LWSURV), NGROUP*NREG )
90: c     call PESUM8( A(LNSPLT), NGROUP*NREG )
91: c     call PESUM8( A(LWSPLT), NGROUP*NREG )
92: c   end if
93: c
94: c/#ENDIF
95: c
96: c   return
97: c   end

```

src/mvp/prdate.c

```
1: /*****
2:  * Output date and time of compilation on standard output
3:  * using ANSI C predefined macros if possible.
4:  *
5:  * 12 Aug 1999: add fflush(stdout).
6:  *****/
7:
8: #include <stdio.h>
9:
10: void prdate()
11: {
12:     print_date_time() ;
13: }
14: void prdate_()
15: {
16:     print_date_time() ;
17: }
18: void PRDATE()
19: {
20:     print_date_time() ;
21: }
22: void _PRDATE()
23: {
24:     print_date_time() ;
25: }
26:
27: int print_date_time()
28: {
29:     int j ;
30:     j = 0 ;
31: #ifdef __DATE__
32: #ifdef __TIME__
33:     j = 1 ;
34:     printf("      COMPILATION DATE: %s  TIME: %s\n", __DATE__, __TIME__);
35: #endif
36: #endif
37:     if ( j == 0 ) {
38:         printf("      COMPILATION DATE: UNKNOWN\n");
39:     }
40:     fflush(stdout) ;
41: }
```


src/mvp/preinp.f

```
1:      subroutine PREINP( IPREIN,CSTRNG,NTASK0,TSKINF,MLIMIT )
2: C=<MVP>=====
3: C Purpose: Interpret command string for file-name/I/O-unit coupling
4: C or default-option overriding etc.
5: C Probably called from main routine before calling of 'CENTER'
6: C routine, or called before reading input to MVP/GMVP.
7: C
8: C=====
9: C argument :
10: C
11: C i iprein : counter of calling for this routine given externally.
12: C (when iprein = 1 , initialize data )
13: C i cstr : command string
14: C
15: C /nn[rfl][:file-name]
16: C
17: C Assign I/O unit nn to a file.
18: C
19: C xxx=yyy
20: C
21: C Execution parameter setting.
22: C
23: C o ntask0 : number of task effective in multiprocessing mode
24: C
25: C NTASK=... or ntask=...
26: C
27: C o tskinf : name of task control information file in multiprocessing
28: C mode.
29: C
30: C TASKINFO=... or taskinfo=...
31: C
32: C o mlimit : memory size limit (effective only in dynamic allocation
33: C mode )
34: C
35: C=====
36: C
37: C include '../shared/INC/_IOUNIT'
38: C
39: C ... external data ...
40: C
41: C character*(*) CSTRNG
42: C character*(*) TSKINF
43: C include 'INC/_FLAGS'
44: C/#IF PARA(PVM)
45: C include '../shared/INC/_PVMPARA'
46: C/#ELSEIF PARA(MPI)
47: C * include 'mpif.h'
48: C/#ENDIF
49: C include '../shared/INC/_TASKDT'
50: C
51: C ... local data ...
52: C
53: C character*20 CTEMP,CTEMP2
54: C
55: C-----
56: C
57: C ... initialization on first call ...
58: C
59: C if ( IPREIN.eq.1 ) then
60: C JRPCPU = 0
61: C/#IF PARA(PVM MPI)
62: C * HOST0 = ' '
63: C/#ENDIF
64: C end if
65: C
```

```
66: C
67: C
68: C write(IPR,'(lx,a,a,a)') ' PARAMETER : <', CSTRNG, '>'
69: C
70: C IE = 0
71: C
72: C .... 'CSTRNG' may be broken into some commands separated by blank.
73: C
74: C
75: C 100 IS = IE + 1
76: C call NOBLNK( CSTRNG, IS, IE, LEN(CSTRNG) )
77: C
78: C
79: C ... cstrng(is:ie) : Command ....
80: C
81: C if ( IE.le.LEN(CSTRNG) ) then
82: C
83: C-----
84: C File allocation
85: C-----
86: C if ( CSTRNG(IS:IS).eq.'/' ) then
87: C
88: C
89: C JJJJJ = JFOPEN(CSTRNG(IS:IE))
90: C
91: C-----
92: C Number of task in multitasking mode
93: C-----
94: C
95: C else if ( CSTRNG(IS:IS+5).eq.'ntask='
96: C & .or. CSTRNG(IS:IS+5).eq.'NTASK=' ) then
97: C
98: C CTEMP = CSTRNG(IS+6:IE)
99: C read(CTEMP(:20),'(bn,i20)') NTASK0
100: C
101: C-----
102: C Hosts name of the current task in multitasking mode
103: C-----
104: C
105: C else if ( CSTRNG(IS:IS+10).eq.'MYHOSTNAME='
106: C & .or. CSTRNG(IS:IS+10).eq.'myhostname=' ) then
107: C
108: C/#IF PARA(PVM MPI)
109: C * host0 = cstrng(is+11:ie)
110: C/#ELSE
111: C write(IPR,*) 'This parameter is not effective in this mode.'
112: C/#ENDIF
113: C
114: C-----
115: C taskinfo=<the name of file which contains MVP/GMVP specific task
116: C control parameters>
117: C ( effective in PVM , NCUBE etc. )
118: C-----
119: C
120: C else if ( CSTRNG(IS:IS+8).eq.'taskinfo='
121: C & .or. CSTRNG(IS:IS+8).eq.'TASKINFO=' ) then
122: C TSKINF = CSTRNG(IS+9:IE)
123: C
124: C-----
125: C Sub-task file name in multitasking mode
126: C-----
127: C
128: C else if(CSTRNG(IS:IS+10).eq.'SUBTASKOUT=' .or.
129: C & CSTRNG(IS:IS+10).eq.'subtaskout=') then
130: C/#IF PARA(PVM MPI)
```

src/mvp/preinp.f

```

131: *          STFN = cstrng(is+11:ie)
132: C/#ELSE
133:          write(IPR,*) '!!!(PREINP) This parameter is not effective ',
134:          &          'in this mode.'
135: C/#ENDIF
136:          else if (CSTRNG(IS:IS+11).eq.'TIMELISTOUT=' .or.
137:          &          CSTRNG(IS:IS+11).eq.'timelistout=') then
138: C/#IF PARA(PVM MPI)
139: *          TLFN = cstrng(is+12:ie)
140: C/#ELSE
141:          write(IPR,*) '!!!(PREINP) This parameter is not effective ',
142:          &          'in this mode.'
143: C/#ENDIF
144:
145: C-----
146: C      Override default options or sizes
147: C-----
148: C      DEBUG-PRINT ( OFF -> ON ) : [NO-]DEBUG-PRINT
149: C      -----
150: C      VP-MONITOR ( OFF -> ON ) : [NO-]VP-MONITOR
151: C      -----
152: C      MONITOR ( OFF -> ON ) : [NO-]MONITOR
153: C      -----
154: C      IMAGINARY-PARTICLE( OFF->ON ) : [NO-]IMAGINARY-PARTICLE
155: C      -----
156: C      DYNAMIC-MEMORY ( NEW-SIZE IN WORD ) : MEMORY=size
157: C-----
158: C
159:          else if ( IMATCH('*DEBU*-PRIN*',CSTRNG(IS:IE)).eq.1 ) then
160:              JDEBG(1) = 1
161:              if ( CSTRNG(IS:IS+2).eq.'NO-' ) JDEBG(1) = 0
162:              go to 110
163:          else if ( IMATCH('*VP-MONI*',CSTRNG(IS:IE)).eq.1 ) then
164:              JVMNT = 1
165:              if ( CSTRNG(IS:IS+2).eq.'NO-' ) JVMNT = 0
166:              go to 110
167:          else if ( IMATCH('*MONI*',CSTRNG(IS:IE)).eq.1 ) then
168:              JMNTR = 1
169:              if ( CSTRNG(IS:IS+2).eq.'NO-' ) JMNTR = 0
170:              go to 110
171:          else if ( IMATCH('*IMAG*-PART*',CSTRNG(IS:IE)).eq.1 ) then
172:              JIMAG = 1
173:              if ( CSTRNG(IS:IS+2).eq.'NO-' ) JIMAG = 0
174:              go to 110
175:          else if ( IMATCH('RUN-MODE=*',CSTRNG(IS:IE)).eq.1 ) then
176:              CTEMP = CSTRNG(IS+9:IE)
177:              read(CTEMP(:20),'(bn,i20)') JRUNM
178:              go to 110
179: C/#IF DYNAMIC
180:          else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
181:              CTEMP = CSTRNG(IS+7:IE)
182:              read(CTEMP(:20),'(bn,i20)') MLIMIT
183:              go to 110
184: C/#ELSE
185:          else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
186:              write(IPR,'(1x,a)')
187:              &          '<MEMORY=...> has meanings in dynamic memory mode.'
188:              go to 110
189: C/#ENDIF
190: C
191: C      SUBTASK-PRINT=i1[,i2]
192: C
193:          else if ( IMATCH('SUBT*-PRIN*',CSTRNG(IS:IE)).eq.1 ) then
194:              KK = index(CSTRNG(IS:IE),'=')
195:              if ( KK.ne.0 ) then

```

```

196:              CTEMP = CSTRNG(IS+KK:IE)
197:              K2 = index(CTEMP,',')
198:              if ( K2.eq.0 ) then
199:                  read(CTEMP(:20),'(BN,I20)') JSTPRN(1)
200:              else
201:                  CTEMP2 = CTEMP(K2+1:)
202:                  CTEMP(K2:) = ' '
203:                  read(CTEMP(:20),'(BN,I20)') JSTPRN(1)
204:                  read(CTEMP2(:20),'(BN,I20)') JSTPRN(2)
205:              end if
206:          end if
207:          go to 110
208: C-----
209: C      unsupported command
210: C-----
211: C
212:          else
213:              write(IMGF,'(1x,a,a)')
214:              &          '!!! Unsupported command line parameter: ',CSTRNG(IS:IE)
215:          end if
216: C
217: 110      continue
218:          go to 100
219:          end if
220: C
221:          return
222:          end

```

src/mvp/prmnttr.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine PRMNTR( IOT, A, CHA, JMNTR, NGROUP, NREG,
3:      &                  JTIME, JNEUT, JPHOT, JDLYN, NEVENT,NCLSN,
4:      &                  WCLSN,
5:      &                  NLEAK, WLEAK, NABSB, WABSB, NECUT, WECUT,
6:      &                  NTCUT, WTCUT, NKILD, WKILD, NSURV, WSURV,
7: c##<2007/03/14:PN3:
8: c## &                  NCNTR, WCNTR )
9:      &                  NCNTR, WCNTR,
10:     &                  JPHNU, NMONPNW, MONPNWGT, NMPNWGT, WMPNWGT )
11: c##>
12: C=<MOVE=====
13: C PURPOSE:  PRINT OUT MONITORING DATAS.
14: C CALLED IN: CADENZ
15: C=====
16:      implicit real*8(D,W)
17: C
18:      real A(*)
19:      character*4 CHA(*)
20: C
21:      include 'INC/_KPID.S'
22:      include 'INC/_KPSYMS'
23: C
24:      real*8 NCLSN(NGROUP,NREG), NLEAK(NGROUP,NREG)
25:      real*8 NABSB(NGROUP,NREG), NECUT(NREG), NTCUT(NGROUP,NREG)
26:      real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG)
27:      real*8 WCLSN(NGROUP,NREG), WLEAK(NGROUP,NREG)
28:      real*8 WABSB(NGROUP,NREG), WECUT(NREG), WTCUT(NGROUP,NREG)
29:      real*8 WKILD(NGROUP,NREG), WSURV(NGROUP,NREG)
30: CCC      real*8 NCNTR(NEVENT,2), WCNTR(NEVENT,2)
31:      real*8 NCNTR(NEVENT,KPLIM), WCNTR(NEVENT,KPLIM)
32: c##<2007/03/14:PN3:
33:      real*8 NMPNWGT(*), WMPNWGT(2,*)
34:      integer MONPNWGT(4,*)
35: c##>
36: C
37: C
38: C =====
39: C call HEADER( 6, 'EVENT MONITOR' )
40: C =====
41: C
42: C
43: C .... NEUTRON EVENT .....
44: C
45: C      if ( JNEUT.ne.0 ) then
46: C
47: C          *****
48: C          call LABEL( IOT, 'EVENTS OF NEUTRONS' )
49: C          *****
50: C
51: C          DW = WCNTR(1,1)
52: c##<2007/03/14:PN3:
53: c##      if (DW.eq.0.0d0 ) DW = 1.0d0
54:      if ( DW.eq.0.0d0 ) then
55:          if ( JPHNU.ne.0.and.WCNTR(1,2).gt.0.0d0 ) then
56:              DW = WCNTR(1,2)
57:          else
58:              DW = 1.0d0
59:          end if
60:      end if
61: c##>
62:      write(IOT,7000)
63: c##<2007/03/14:PN3:
64: c## &      NCNTR(1,1),WCNTR(1,1),
65: c## &      NCNTR(2,1),WCNTR(2,1), NCNTR(2,1)/DW,WCNTR(2,1)/DW,

```

```

66: c## &      NCNTR(25,1),WCNTR(25,1),NCNTR(25,1)/DW,WCNTR(25,1)/DW,
67: c## &      NCNTR(13,1),WCNTR(13,1),NCNTR(13,1)/DW,WCNTR(13,1)/DW,
68: c## &      (NCNTR(N,1),WCNTR(N,1), NCNTR(N,1)/DW,WCNTR(N,1)/DW,N=3,12),
69: c## &      NCNTR(24,1),WCNTR(24,1), NCNTR(24,1)/DW,WCNTR(24,1)/DW,
70: c## &      NCNTR(23,1), WCNTR(23,1), NCNTR(23,1)/DW, WCNTR(23,1)/DW
71: c##
72: c##      if ( JTIME.ne.0 ) then
73: c##          write(IOT,7020) NCNTR(26,1), WCNTR(26,1),
74: c## &                  NCNTR(26,1)/DW, WCNTR(26,1)/DW
75: c##      end if
76: c##      write(IOT,7030) NCNTR(27,1), WCNTR(27,1),
77: c## &                  NCNTR(27,1)/DW, WCNTR(27,1)/DW
78: c##      write(IOT,7032)
79: c## &                  NCNTR(28,1), WCNTR(28,1),
80: c## &                  NCNTR(28,1)/DW, WCNTR(28,1)/DW
81: c##c##<2003/08/15:
82: c##      if ( JDLYN.ne.0 ) write(IOT,7033)
83: c## &                  NCNTR(30,1), WCNTR(30,1), NCNTR(30,1)/DW, WCNTR(30,1)/DW
84: c##c##>
85: c## &      NCNTR( 1,1), WCNTR( 1,1),
86: c## &      NCNTR( 2,1), WCNTR( 2,1), NCNTR( 2,1)/DW, WCNTR( 2,1)/DW,
87: c## &      NCNTR(29,1), WCNTR(29,1), NCNTR(29,1)/DW, WCNTR(29,1)/DW,
88: c## &      NCNTR(25,1), WCNTR(25,1), NCNTR(25,1)/DW, WCNTR(25,1)/DW,
89: c## &      NCNTR(13,1), WCNTR(13,1), NCNTR(13,1)/DW, WCNTR(13,1)/DW,
90: c## &      (NCNTR( N,1), WCNTR( N,1), NCNTR( N,1)/DW, WCNTR( N,1)/DW,
91: c## &                  N=3,12),
92: c## &      NCNTR(24,1), WCNTR(24,1), NCNTR(24,1)/DW, WCNTR(24,1)/DW,
93: c## &      NCNTR(23,1), WCNTR(23,1), NCNTR(23,1)/DW, WCNTR(23,1)/DW
94: c##      if ( JTIME.ne.0 ) write(IOT,7020)
95: c## &      NCNTR(26,1), WCNTR(26,1), NCNTR(26,1)/DW, WCNTR(26,1)/DW
96: c##      write(IOT,7030)
97: c## &      NCNTR(27,1), WCNTR(27,1), NCNTR(27,1)/DW, WCNTR(27,1)/DW
98: c##      write(IOT,7032)
99: c## &      NCNTR(28,1), WCNTR(28,1), NCNTR(28,1)/DW, WCNTR(28,1)/DW
100: c## &      if ( JDLYN.ne.0 ) write(IOT,7033)
101: c## &      NCNTR(30,1), WCNTR(30,1), NCNTR(30,1)/DW, WCNTR(30,1)/DW
102: c##>
103: c##      write(IOT,7040) (NCNTR(N,1),NCNTR(N,1)/DW,N=16,18)
104: C
105: c##      end if
106: C
107: C
108: 7000 format(/1X,58X,'          TOTAL          PER SOURCE WEIGHT'
109: &          /1X,58X,'COUNT    WEIGHT SUM    COUNT    WEIGHT'
110: &          //10X,
111: c##<2007/03/14:PN3:
112: c## & 'SOURCE PARTICLES',F16.0,2X,E13.6/10X,
113: c## & 'FISSION NEUTRONS (WHEN JEIGN=0)',F16.0,3(2X,E13.6)/10X,
114: c## & 'NEUTRONS INCREASED BY (N,MN) REACTION',F16.0,3(2X,E13.6)/10X,
115: c## & 'FISSION REACTION PREVENTED (JEIGN=0)',F16.0,3(2X,E13.6)/10X,
116: c## & '(N,GAMMA+X) REACTION',F16.0,3(2X,E13.6)/10X,
117: c## & 'COLLISION',F16.0,3(2X,E13.6)/10X,
118: c## & 'SPLITTING (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
119: c## & 'SPLITTING PREVENTED',F16.0,3(2X,E13.6)/10X,
120: c## & 'LEAKAGE',F16.0,3(2X,E13.6)/10X,
121: c## & 'ENERGY CUTOFF',F16.0,3(2X,E13.6)/10X,
122: c## & 'KILLED (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
123: c## & 'SURVIVED (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
124: c## & 'KILLED (WEIGHT CUTOFF)',F16.0,3(2X,E13.6)/10X,
125: c## & 'SURVIVED (WEIGHT CUTOFF)',F16.0,3(2X,E13.6)/10X,
126: c## & 'KILLED AT FISGEN AND PHTGEN',F16.0,3(2X,E13.6)/10X,
127: c## & 'ANALOG ABSORPTION',F16.0,3(2X,E13.6))
128: c## & 'SOURCE PARTICLES',F16.0,1P, E15.6/10X,
129: c## & 'FISSION NEUTRONS (when JEIGN=0)',F16.0,1P,3E15.6/10X,
130: c## & 'GENERATED BY PHOTONUCLEAR',F16.0,1P,3E15.6/10X,

```

src/mvp/prmnttr.f

```

131:      & 'NEUTRONS INCREASED BY (N,MN) REACTION',0P,F16.0,1P,3E15.6/10X,
132:      & 'FISSION REACTION PREVENTED (JEIGN=0)',0P,F16.0,1P,3E15.6/10X,
133:      & '(N,GAMMA+X) REACTION',0P,F16.0,1P,3E15.6/10X,
134:      & 'COLLISION',0P,F16.0,1P,3E15.6/10X,
135:      & 'SPLITTING (IMPORTANCE OR WEIGHT WINDOW)',0P,F16.0,1P,3E15.6/10X,
136:      & 'SPLITTING PREVENTED',0P,F16.0,1P,3E15.6/10X,
137:      & 'LEAKAGE',0P,F16.0,1P,3E15.6/10X,
138:      & 'ENERGY CUTOFF',0P,F16.0,1P,3E15.6/10X,
139:      & 'KILLED (IMPORTANCE OR WEIGHT WINDOW)',0P,F16.0,1P,3E15.6/10X,
140:      & 'SURVIVED (IMPORTANCE OR WEIGHT WINDOW)',0P,F16.0,1P,3E15.6/10X,
141:      & 'KILLED (WEIGHT CUTOFF)',0P,F16.0,1P,3E15.6/10X,
142:      & 'SURVIVED (WEIGHT CUTOFF)',0P,F16.0,1P,3E15.6/10X,
143:      & 'KILLED AT FISGEN AND PHTGEN',0P,F16.0,1P,3E15.6/10X,
144:      & 'ANALOG ABSORPTION',0P,F16.0,1P,3E15.6)
145: c##>
146: 7020 format(10X,
147: c##<2007/03/14:PN3:
148: c## & 'TIME CUTOFF',F16.0,3(2X,E13.6))
149: & 'TIME CUTOFF',0P,F16.0,1P,3E15.6)
150: c##>
151: 7030 format(10X,
152: c##<2007/03/14:PN3:
153: c## & 'PARTICLE GENERATION CUTOFF',F16.0,3(2X,E13.6))
154: & 'PARTICLE GENERATION CUTOFF',0P,F16.0,1P,3E15.6)
155: c##>
156: 7032 format(10X,
157: c##<2007/03/14:PN3:
158: c## & 'ENDLESS LOOP CUTOFF',F16.0,3(2X,E13.6))
159: & 'ENDLESS LOOP CUTOFF',0P,F16.0,1P,3E15.6)
160: c##>
161: 7033 format(10X,
162: c##<2007/03/14:PN3:
163: c## & 'DELAYED FISSION NEUTRONS (JEIGN=0)',F16.0,3(2X,E13.6))
164: & 'DELAYED FISSION NEUTRONS (JEIGN=0)',0P,F16.0,1P,3E15.6)
165: 7034 format(10X,
166: & 'DELAYED NEUTRON FROM PHOTO-FISSION',0P,F16.0,1P,3E15.6)
167: 7035 format(10X,
168: & 'PROMPT NEUTRON FROM PHOTO-FISSION',0P,F16.0,1P,3E15.6)
169: c##>
170: 7040 format(/10X,
171: c##<2007/03/14:PN3:
172: c## & 'NUMBER OF FREE FLIGHT',F16.0,15X,2X,E13.6/10X,
173: c## & 'NUMBER OF BOUNDARY CROSSING',F16.0,15X,2X,E13.6/10X,
174: c## & 'NUMBER OF REFLECTION',F16.0,15X,2X,E13.6)
175: & 'NUMBER OF FREE FLIGHT',0P,F16.0,1P, E30.6/10X,
176: & 'NUMBER OF BOUNDARY CROSSING',0P,F16.0,1P, E30.6/10X,
177: & 'NUMBER OF REFLECTION',0P,F16.0,1P, E30.6)
178: c##>
179: C
180: C
181: C .... PHOTON EVENT .....
182: C
183: C if ( JPHOT.ne.0 ) then
184: C
185: C call LABEL( IOT, 'EVENTS OF PHOTONS' )
186: C
187: C
188: C DW = WCNTR(1,2)
189: C if (DW.eq.0.0d0) then
190: C DW = NCNTR(1,1)
191: C write(IOT,7001)
192: 7001 format(15X,'( "per source weight" is for neutron source.)')
193: C end if
194: C
195: c##<2007/03/14:PN3:

```

```

196: c## write(IOT,7060) NCNTR(1,2), WCNTR(1,2),
197: c## & NCNTR(3,2), WCNTR(3,2), NCNTR(3,2)/DW, WCNTR(3,2)/DW,
198: c## & NCNTR(19,2), WCNTR(19,2), NCNTR(19,2)/DW, WCNTR(19,2)/DW,
199: c## & NCNTR(20,2), WCNTR(20,2), NCNTR(20,2)/DW, WCNTR(20,2)/DW,
200: c## & NCNTR(2,2), WCNTR(2,2), NCNTR(2,2)/DW, WCNTR(2,2)/DW,
201: c## & (NCNTR(N,2),WCNTR(N,2), NCNTR(N,2)/DW,WCNTR(N,2)/DW, N=4,12),
202: c## & NCNTR(24,2), WCNTR(24,2), NCNTR(24,2)/DW, WCNTR(24,2)/DW,
203: c## & NCNTR(23,2), WCNTR(23,2), NCNTR(23,2)/DW, WCNTR(23,2)/DW
204: C
205: c## if ( JTIME.ne.0 ) then
206: c## write(IOT,7020) NCNTR(26,2),WCNTR(26,2),
207: c## & NCNTR(26,2)/DW,WCNTR(26,2)/DW
208: c## end if
209: C
210: c## write(IOT,7030) NCNTR(27,2), WCNTR(27,2),
211: c## & NCNTR(27,2)/DW, WCNTR(27,2)/DW
212: c## write(IOT,7060)
213: & NCNTR( 1,2), WCNTR( 1,2),
214: & NCNTR( 3,2), WCNTR( 3,2), NCNTR( 3,2)/DW, WCNTR( 3,2)/DW,
215: & NCNTR(29,2), WCNTR(29,2), NCNTR(29,2)/DW, WCNTR(29,2)/DW,
216: & NCNTR(13,2), WCNTR(13,2), NCNTR(13,2)/DW, WCNTR(13,2)/DW,
217: & NCNTR(19,2), WCNTR(19,2), NCNTR(19,2)/DW, WCNTR(19,2)/DW,
218: & NCNTR(20,2), WCNTR(20,2), NCNTR(20,2)/DW, WCNTR(20,2)/DW,
219: & NCNTR( 2,2), WCNTR( 2,2), NCNTR( 2,2)/DW, WCNTR( 2,2)/DW,
220: & (NCNTR( N,2), WCNTR( N,2), NCNTR( N,2)/DW, WCNTR( N,2)/DW,
221: & N=4,12),
222: & NCNTR(24,2), WCNTR(24,2), NCNTR(24,2)/DW, WCNTR(24,2)/DW,
223: & NCNTR(23,2), WCNTR(23,2), NCNTR(23,2)/DW, WCNTR(23,2)/DW
224: c## if ( JTIME.ne.0 ) write(IOT,7020)
225: & NCNTR(26,2), WCNTR(26,2), NCNTR(26,2)/DW, WCNTR(26,2)/DW
226: c## write(IOT,7030)
227: & NCNTR(27,2), WCNTR(27,2), NCNTR(27,2)/DW, WCNTR(27,2)/DW
228: c## write(IOT,7035)
229: & NCNTR(14,2), WCNTR(14,2), NCNTR(14,2)/DW, WCNTR(14,2)/DW
230: c## if ( JDLYN.ne.0 ) write(IOT,7034)
231: & NCNTR(30,2), WCNTR(30,2), NCNTR(30,2)/DW, WCNTR(30,2)/DW
232: c##>
233: C
234: C write(IOT,7040) (NCNTR(N,2),NCNTR(N,2)/DW,N=16,18)
235: C
236: C end if
237: C
238: C
239: C7060 format(/1X,58X,'COUNT WEIGHT SUM'/10X,
240: 7060 format(/1X,58X,' TOTAL PER SOURCE WEIGHT'
241: & /1X,58X,'COUNT WEIGHT SUM COUNT WEIGHT'
242: & //10X,
243: c##<2007/03/14:PN3:
244: c## & 'SOURCE PARTICLES',F16.0,2X,E13.6/10X,
245: c## & 'GENERATED BY (N,GAMMA)',F16.0,3(2X,E13.6)/10X,
246: c## & 'FLUORESCENCE',F16.0,3(2X,E13.6)/10X,
247: c## & 'ANNIHILATION',F16.0,3(2X,E13.6)/10X,
248: c## & 'BREMSSTRAHLUNG',F16.0,3(2X,E13.6)/10X,
249: c## & 'COLLISION',F16.0,3(2X,E13.6)/10X,
250: c## & 'SPLITTING (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
251: c## & 'SPLITTING PREVENTED',F16.0,3(2X,E13.6)/10X,
252: c## & 'LEAKAGE',F16.0,3(2X,E13.6)/10X,
253: c## & 'ENERGY CUTOFF',F16.0,3(2X,E13.6)/10X,
254: c## & 'KILLED (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
255: c## & 'SURVIVED (IMPORTANCE OR WEIGHT WINDOW)',F16.0,3(2X,E13.6)/10X,
256: c## & 'KILLED (WEIGHT CUTOFF)',F16.0,3(2X,E13.6)/10X,
257: c## & 'SURVIVED (WEIGHT CUTOFF)',F16.0,3(2X,E13.6)/10X,
258: c## & 'KILLED IN THICK-TARGET BREMSSTRAHLUNG',F16.0,3(2X,E13.6)/10X,
259: c## & 'ANALOG ABSORPTION',F16.0,3(2X,E13.6))
260: c## & 'SOURCE PARTICLES',0P,F16.0,1P, E15.6/10X,

```

src/mvp/prmnttr.f

```

261:      & 'GENERATED BY (N,GAMMA)                ',0P,F16.0,1P,3E15.6/10X,
262:      & 'GENERATED BY PHOTONUCLEAR              ',0P,F16.0,1P,3E15.6/10X,
263:      & 'PHOTO-FISSION                          ',0P,F16.0,1P,3E15.6/10X,
264:      & 'FLUORESCENCE                          ',0P,F16.0,1P,3E15.6/10X,
265:      & 'ANNIHILATION                          ',0P,F16.0,1P,3E15.6/10X,
266:      & 'BREMSSTRAHLUNG                       ',0P,F16.0,1P,3E15.6/10X,
267:      & 'COLLISION                            ',0P,F16.0,1P,3E15.6/10X,
268:      & 'SPLITTING (IMPORTANCE OR WEIGHT WINDOW)',0P,F16.0,1P,3E15.6/10X,
269:      & 'SPLITTING PREVENTED                  ',0P,F16.0,1P,3E15.6/10X,
270:      & 'LEAKAGE                              ',0P,F16.0,1P,3E15.6/10X,
271:      & 'ENERGY CUTOFF                        ',0P,F16.0,1P,3E15.6/10X,
272:      & 'KILLED (IMPORTANCE OR WEIGHT WINDOW) ',0P,F16.0,1P,3E15.6/10X,
273:      & 'SURVIVED (IMPORTANCE OR WEIGHT WINDOW)',0P,F16.0,1P,3E15.6/10X,
274:      & 'KILLED (WEIGHT CUTOFF)               ',0P,F16.0,1P,3E15.6/10X,
275:      & 'SURVIVED (WEIGHT CUTOFF)             ',0P,F16.0,1P,3E15.6/10X,
276:      & 'KILLED IN THICK-TARGET BREMSSTRAHLUNG',0P,F16.0,1P,3E15.6/10X,
277:      & 'ANALOG ABSORPTION                   ',0P,F16.0,1P,3E15.6/10X,
278: c##>
279: C7080 format(/10X,
280: C      & 'NUMBER OF FREE FLIGHT              ',F16.0/10X,
281: C      & 'NUMBER OF BOUNDARY CROSSING         ',F16.0/10X,
282: C      & 'NUMBER OF REFLECTION                ',F16.0)
283: C
284: C
285: c##<2007/03/14:PN3:
286:      if ( JPHNU.ne.0.and.NMONPNW.gt.0 ) then
287:          DW = WCNTR(1,2)
288:          if ( DW.eq.0.0d0 ) DW = NCNTR(1,1)
289:          M = MONPNWGT(4,NMONPNW)
290:          write(IOT,7130)
291:          do I = 1, M
292:              RT = 0
293:              STD = 0
294:              if ( NMPNWGT(I).gt.0.D0 ) then
295:                  RT = WMPNWGT(1,I) / NMPNWGT(I)
296:                  STD = WMPNWGT(2,I)/NMPNWGT(I) -
297:                      & (WMPNWGT(1,I)/NMPNWGT(I))**2
298:                  if ( STD.le.0.0d0 ) then
299:                      STD = 0
300:                  else
301:                      STD = sqrt( STD )
302:                  end if
303:              end if
304:              write(IOT,7131) I, NMPNWGT(I), WMPNWGT(1,I), NMPNWGT(I)/DW,
305:                  & WMPNWGT(2,I)/DW, RT, STD
306:          end do
307:          write(IOT,'(1h)')
308:      end if
309: 7130 format(/
310:      &' -----'
311:      &' -- average weight generated by photonuclear reaction --'
312:      &' -----'
313:      &' '
314:      &' monitor                (total)                (per',
315:      &' source)'
316:      &' set no.                count    weight sum    count',
317:      &' weight weight/count std.deviation')
318: 7131 format(i8,1x,f15.0,1p,3e15.6,2e15.5)
319: c##>
320: C
321:      if ( JMNTR.eq.0 ) return
322: C
323:      write(IOT,'(/1X,'DETAILED MONITORING DATA FROM NEXT PAGE ')')
324: c##<2007/03/14:PN3:
325: c##      write(IOT,7100) '          <COLLISION>          '

```

```

326: c7100 format('1',A40)
327:      write(IOT,7100) '          <<COLLISION>>          '
328: 7100 format('1',A41)
329: c##>
330:      call PRNUWE( IOT, NCLSN, WCLSN, NGROUP, NREG, 'GROUP ', 'REGION',
331:          & A, CHA )
332: CCCC WRITE(IOT,7100) '          <LEAKAGE>          '
333: CCCC CALL PRNUWE(IOT,NLEAK,WLEAK,NGROUP,NREG,'GROUP ', 'REGION',
334: CCCC & A, CHA )
335: CCCC WRITE(IOT,7100) '          <ABSORPTION>          '
336: CCCC CALL PRNUWE(IOT,NABSB,WABSB,NGROUP,NREG,'GROUP ', 'REGION',
337: CCCC & A, CHA )
338:      if ( JTIME.ne.0 ) then
339: c##<2007/03/14:PN3:
340: c##      write(IOT,7100) '          <KILLED PARTICLE BY TIME CUTOFF> '
341:      write(IOT,7100) '          <<KILLED PARTICLE BY TIME CUTOFF>> '
342: c##>
343:      call PRNUWE( IOT, NTCUT, WTCUT, NGROUP, NREG, 'GROUP ',
344:          & 'REGION', A, CHA )
345:      end if
346: CCCC WRITE(IOT,7100) '          <KILLED PARTICLE>          '
347: CCCC CALL PRNUWE(IOT,NKILD,WKILD,NGROUP,NREG,'GROUP ', 'REGION',
348: CCCC & A, CHA )
349: CCCC WRITE(IOT,7100) '          <SURVIVED PARTICLE>          '
350: CCCC CALL PRNUWE(IOT,NSURV,WSURV,NGROUP,NREG,'GROUP ', 'REGION',
351: CCCC & A, CHA )
352: c##<2007/03/14:PN3:
353: c##      write(IOT,7100) '          <KILLED PARTICLE BY ENERGY CUTOFF>'
354:      write(IOT,7100) '          <<KILLED PARTICLE BY ENERGY CUTOFF>>'
355: c##>
356:      write(IOT,7120) (I,NECUT(I),WECUT(I),I=1,NREG)
357: 7120 format(/1X,' REGION          NUMBER          WEIGHT ' /
358: c##<2007/03/14:PN3:
359: c##      & (1X,3X,I5,2X,F12.0,5X,E13.6))
360:      & (1X,3X,I5,2X,0P,F12.0,5X,1P,E13.6))
361: c##>
362:      return
363:      end

```

src/mvp/prnspc.f

```
1:      subroutine PRNSPC( IOG, A, CHA,  ISPACE,LEVEL )
2: C=====
3: C PURPOSE: LOST PARTICLE CHECK  ( SUBSPACE NAME )
4: C CALLED IN: LOST PARTICLE PROCESSING.
5: C=====
6:      real A(*)
7:      character*4 CHA(*)
8: C
9:      include '../shared/INC/_PTLSP'
10:     include '../shared/INC/_PGEOM'
11:     include '../shared/INC/_SIZES'
12:     include 'INC/_SIZES2'
13:     include '../shared/INC/_PTALY0'
14:     include 'INC/_PTALY'
15: CCCC include '../shared/INC/_ARRAY'
16: C
17:     call PRNSPD( IOG, ISPACE, LEVEL, A(LISPNM), CHA(LTNAMS), NNAMES,
18: &              NEST, NSPACE )
19:     return
20:     end
```

src/mvp/prsize.f

```

1:      subroutine PRSIZE( IOUT )
2: C=====
3: C  PURPOSE:  PRINTOUT OF COMMON /SIZES/ AND OTHER PARAMETERS
4: C  CALLED IN:  INTRO
5: C=====
6:      include 'INC/_KPID$'
7:      include 'INC/_NGPS'
8: C
9:      include '../shared/INC/_SIZES'
10:     include '../shared/INC/_STALY'
11:     include 'INC/_SIZES2'
12:     include '../shared/INC/_TASKDT'
13:     include 'INC/_FLAGS'
14:     include 'INC/_CXSEC'
15:     include 'INC/_SBANK'
16:     include 'INC/_PTALY2'
17: C
18: C  =====
19: C  call HEADER( IOUT,
20: &'SIZE- OR LIMIT-PARAMETERS (SOME VALUES MAY BE GIVEN BY THE CODE)
21: & )
22: C  =====
23: C
24: 7000 format(/3X,'<< ',A,' >>'//)
25: 7020 format(3X,A,T11,I10,3X,A)
26: 7040 format(3X,A,T11,I10,3X,A/3X,T11,10X,3X,A)
27: 7050 format(3X,A,T11,F10.3,3X,A/3X,T11,10X,3X,A)
28: 7060 format(3X,A,T11,F10.3,3X,A)
29: 7080 format(3X,A,T11,F10.3,3X,A,A,A)
30: 7100 format(3X,A,T11,1P,E10.3,3X,A)
31: 7120 format(3X,T11,10X,3X,A)
32: 7140 format(3X,T11,10X,3X,A,A)
33: C
34: C  -----
35:     write(IOUT,7000) 'HISTORY & BATCH CONTROL'
36: C  -----
37: C
38:     write(IOUT,7020)
39:     & 'NTASK', NTASK, 'Number of tasks in this run'
40:     & 'NPART', NPART, 'Number of histories to run (per task)',
41:     & 'NHIST', NHIST, 'Number of histories per batch',
42:     & 'NHSUB', NHSUB, 'Number of histories per sub-batch',
43:     & 'NTHIST', NTHIST,
44:     & 'Number of histories run so far (>0 in restart mode)',
45:     & 'IRAND', IRAND, 'Initial random number',
46:     & 'NBATCH', NBATCH,
47:     & 'Number of batches run so far (>0 in restart mode)',
48:     & 'NSKIP', NSKIP, 'Skipped batch in eigenvalue calculation'
49:     write(IOUT,7040)
50:     & 'NLENG', NLENG,
51:     & 'Number of batches expected after eigenvalue calculation',
52:     & '(>= NBATCH in restart mode).'
53:     write(IOUT,7020)
54:     & 'NRSKIP', NRSKIP, 'Number of random numbers skipped'
55:     write(IOUT,7120)
56:     & '(< 0 : before each batch, > 0 : before first batch)'
57:     write(IOUT,7020)
58:     & 'NRSINT', NRSINT, 'Restart file output interval'
59:     write(IOUT,7120)
60:     & '(> 0 : after every NRSINT''th batch',
61:     & '(< 0 : after every NRSINT''th history',
62:     & '= 0 : only after the last batch'
63:     write(IOUT,7020)
64:     & 'NBPINT', NBPINT, 'Batch monitor printout interval'
65:     write(IOUT,7060)

```

```

66:     & 'TCPU', TCPU, 'Calculation CPU time limit (minute.)'
67:     write(IOUT,7020)
68:     & 'NPKIND', NPKIND, 'Number of particle type to calculate'
69: C
70: C  -----
71:     write(IOUT,7000) 'PARTICLE BANK'
72: C  -----
73: C
74:     write(IOUT,7020)
75:     & 'NBANK', NBANK, 'Length of particle bank',
76:     & 'NFBANK', NFBANK, 'Length of fission source bank'
77:     write(IOUT,7050)
78:     & 'BANKP', BANKP,
79:     & 'Probability of photon-generation when particle bank',
80:     & 'does not have space in (N,GAMMA) reaction.'
81: c <note> Since PPNBR is an array (PPNBR(NUCPN)), output in this routine
82: c in not appropriate. In addition, PPNBR is not defined in this
83: c routine.
84: c write(IOUT,7050)
85: c & 'PPNBR', PPNBR,
86: c & 'Probability which photo-nuclear reaction is forcedly',
87: c & 'selected in photon collision.'
88: C
89: C  -----
90:     write(IOUT,7000) 'GEOMETRY'
91: C  -----
92: C
93:     write(IOUT,7020)
94:     & 'NZONE', NZONE, 'Number of zones',
95:     & 'NINPZ', NINPZ, 'Number of input zones',
96:     & 'NSURF', NSURF, 'Number of surfaces',
97:     & 'NSDA ', NSDA, 'Length of surface data area',
98:     & 'NZDA ', NZDA, 'Length of zone data area',
99:     & 'NLATT', NLATT, 'Number of lattices',
100:    & 'NCELL', NCELL, 'Number of cells',
101:    & 'NEST', NEST, 'Maximum nesting level of lattice geometry',
102:    & 'NLBZ', NLBZ, 'Number of lattice-buffer zones',
103:    & 'NMEMO', NMEMO, 'Number of next-zone memory per zone',
104:    & 'NMEMOP', NMEMOP,
105:    & 'Optimize KMEMO array while NBATCH .le. NMEMOP (default=2)',
106:    & 'NREFS', NREFS, 'Number of reflection surfaces',
107:    & 'NMEMS', NMEMS,
108:    & 'Number of memorized starting zones for each particle source',
109:    & 'NPIC', NPIC, 'Number of pictures drawn'
110: C
111: C  -----
112:     write(IOUT,7000) 'TALLY'
113: C  -----
114: C
115:     write(IOUT,7020)
116:     & 'NREG', NREG, 'Number of regions',
117:     & 'NTREG', NTREG, 'Number of tally-regions',
118:     & 'NSPACE', NSPACE, 'Number of subspaces excluding level 0',
119:     & 'NUNV', NUNV, 'Number of input-zones which define frames',
120:     & 'NSUZON', NSUZON,
121:     & 'Number of zone in regions in a subspaces.',
122:     & 'NCREG', NCREG,
123:     & 'Number of tally-regions in "#TALLY REGION" input',
124:     & 'NGROUP', NGROUP, 'Number of energy bins (groups)',
125:     & 'NGP1', NGP1, 'Number of neutron energy bins (groups)',
126:     & 'NGP2', NGP2, 'Number of photon energy bins (groups)',
127:     & 'NTIME', NTIME, 'Number of time bins',
128:     & 'NSOUR', NSOUR, 'Number of sources',
129:     & 'NRESP', NRESP, 'Number of responses (multi-group)',
130:     & 'NSTAL', NSTAL, 'Number of responses (continuous energy)',

```

src/mvp/prsize.f

```
131:      & 'NSTALY', NSTALY, 'Number of special tallies'
132: C
133: C -----
134:      write(IOUT,7000) 'MATERIAL & CROSS SECTION'
135: C -----
136: C
137:      write(IOUT,7020)
138:      & 'NMAT', NMAT, 'Number of materials',
139:      & 'NUC', NUC, 'Number of nuclides (neutron reaction)',
140: c##<2007/03/14:PN3:
141: c## & 'NPATOM',NPATOM,'Number of atoms (photon reaction)'
142:      & 'NPATOM',NPATOM,'Number of atoms (photon reaction)',
143:      & 'NUCPN',NUCPN,'Number of nuclides (photonuclear reaction)'
144: c##>
145:      write(IOUT,7040)
146:      & 'MB',MB,'Maximum number of materials whose macro x-sec etc.',
147:      & 'are stored in sigma bank.'
148:      write(IOUT,7020)
149:      & 'NSMAC', NSMAC, 'Number of macro reactions in sigma bank',
150:      & 'NSMIC', NSMIC, 'Number of micro reactions in sigma bank',
151: c##<2007/03/14:PN3:
152:      & 'NSMICPN',NSMICPN,
153:      & 'Number of micro reactions for photonuclear in sigma bank',
154: c##>
155:      & 'NEMAC', NEMAC, 'Number of macro reactions calculated',
156:      & 'NEMIC', NEMIC, 'Number of micro reactions calculated'
157: C
158: C -----
159:      write(IOUT,7000) 'CUTOFF & THRESHOLD'
160: C -----
161:      write(IOUT,7100) 'TCUT', TCUT, 'Cutoff particle time(age) (sec.)'
162: C -----
163:      write(IOUT,7000) 'CUTOFF & THRESHOLD (NEUTRON)'
164: C -----
165: C
166:      write(IOUT,7100)
167:      & 'ETOP', ETOP, 'Upper energy limit (eV)',
168:      & 'EBOT', EBOT, 'Lower energy limit (eV)',
169:      & 'ETHMAX',ETHMAX,'Upper energy limit for thermal scattering',
170:      & 'AMLIM', AMLIM, 'Upper mass limit for free-gas treatment',
171:      & 'EWCUT', EWCUT, 'Threshold energy for analog absorption (eV)',
172:      & 'ETOPL', ETOPL, 'Upper energy limit of x-sec library',
173:      & 'EBOTL', EBOTL, 'Lower energy limit of x-sec library',
174:      & 'ESABL', ESABL, 'Upper energy limit for S(alpha,beta)'
175: C
176: C -----
177:      write(IOUT,7000) 'CUTOFF & THRESHOLD (PHOTON)'
178: C -----
179: C
180:      write(IOUT,7100)
181:      & 'ETOPP', ETOPP, 'Upper energy limit (eV)',
182:      & 'EBOTP', EBOTP, 'Lower energy limit (eV)',
183:      & 'ETOPLP',ETOPLP,'Upper energy limit of x-sec library',
184:      & 'EBOTLP',EBOTLP,'Lower energy limit of x-sec library'
185: C
186: c##<2007/03/14:PN3:
187: C -----
188:      write(IOUT,7000) 'CUTOFF & THRESHOLD (PHOTONUCLEAR)'
189: C -----
190: C
191:      write(IOUT,7100)
192:      & 'ETOPLPN',ETOPLPN,'Upper energy limit of x-sec library',
193:      & 'EBOTLPN',EBOTLPN,'Lower energy limit of x-sec library'
194: C
195: c##>
```

```
196:      return
197:      end
```


src/mvp/prsour.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine PRSOUR( NSOUR, NGROUP,NMAT,  JTIME, JEIGN, KINPZ,
3:      &                  KSOUR, ISZON, SOUR,  PSPAC, PENRG, KENRG,
4:      &                  NSTIM, STIM,  PSTIM, ISTIM, NSANG, SANG,
5:      &                  SAXIS, PSANG, ISANG, IFISM, TEMP )
6: C=<MVP>=====
7: C  PURPOSE: PRINTOUT & PREPROCESSING OF SOURCE DATA.
8: C  CALLED IN:  INTRO
9: C
10: C=====
11:      integer KSOUR(NSOUR), ISZON(2,NSOUR), KENRG(2,NGROUP,NSOUR),
12:      &        NSTIM(NSOUR), ISTIM(*), NSANG(NSOUR), ISANG(NSOUR),
13:      &        KINPZ(*), IFISM(NSOUR)
14:      real*8  SOUR(NSOUR)
15:      real    PSPAC(10,NSOUR), PENRG(NGROUP,NSOUR)
16:      real    STIM(*), PSTIM(*), SANG(*), SAXIS(3,NSOUR), PSANG(*)
17:      real    TEMP(*)
18: C
19:      include '../shared/INC/_IOUNIT'
20: C
21: C =====
22:      call HEADER( IPR, 'SOURCE DATA' )
23: C =====
24: C
25:      write(IPR,'(/1X,' * NUMBER OF SOURCES =',I5)') NSOUR
26: C
27: C *****
28:      call LABEL( IPR, 'SOURCE FRACTION (UNNORMALIZED)' )
29: C *****
30: C
31:      call R8PRNT( 'SOUR ',SOUR,NSOUR, 1, 'SOURCE #', ' ', IPR )
32: C
33: C *****
34:      call LABEL( IPR, 'SOURCE FRACTION (NORMALIZED)' )
35: C *****
36: C
37:      call PNORM8( SOUR, NSOUR )
38:      call R8PRNT( 'SOUR ',SOUR,NSOUR, 1, 'SOURCE #', ' ', IPR )
39: C
40: CM  IF(JEIGN.NE.0) THEN
41: CM      WRITE(IPR,'(/1X,' == MAT # FOR FISSION SPECTRUM ==')')
42: CM      CALL R4PRNT('IFISM ',IFISM,NSOUR, 1,'SOURCE #', ' ')
43: CM  ENDIF
44: C
45: C
46: C *****
47:      call LABEL( IPR, 'DATA FOR EACH SOURCE' )
48: C *****
49: C
50:      do 100 N = 1, NSOUR
51:          write(IPR,'(/1X,' >>>> SOURCE # ',I3,' KSOUR=',I3,
52:          &        ' FRACTION = ',E12.5)') N, KSOUR(N), SOUR(N)
53:          if ( ISZON(1,N).ne.0 ) write(IPR,7000) ISZON(2,N),
54:          &        KINPZ(ISZON(1,N)), ISZON(1,N)
55:          7000 format(/1X,' STARTING ZONE : ',I5,'TH ZONE OF ',I5,
56:          &        'TH INPUT-ZONE ----> ZONE ',I5)
57:          write(IPR,'(/1X,' SPATIAL DATA ' '))
58:          call R4PRNT( 'PSPAC ', PSPAC(1,N), 10, 1, ' DATA #',
59:          &        ' ', IPR )
60: CM
61:          if ( KSOUR(N).eq.1 .or. KSOUR(N).eq.3 .or. KSOUR(N).eq.4 ) then
62:              if ( PSPAC(9,N).eq.0..and.PSPAC(10,N).eq.0. ) then
63:                  PSPAC(9,N) = -1.0
64:                  PSPAC(10,N) = 1.0
65:              end if

```

```

66:          end if
67: C
68:          if ( JEIGN.eq.0 ) then
69:              if ( KSOUR(N).ne.9 ) then
70:                  write(IPR,
71:          &        '(/1X,' ** ENERGY DISTRIBUTION (UNNORMALIZED)')')
72:                  call R4PRNT( 'PENRG ', PENRG(1,N), NGROUP, 1, ' GROUP ',
73:          &        ' ', IPR )
74: C
75: C  TEMPORARY MODIFICATION FOR MONO ENRGY SOURCE      ...
76: C
77:          if ( PENRG(1,N).ge.0.0 ) then
78:              call PNORM( PENRG(1,N), NGROUP )
79:              write(IPR,
80:          &        '(/1X,' ** ENERGY DISTRIBUTION (NORMALIZED)')')
81:              call R4PRNT( 'PENRG ', PENRG(1,N), NGROUP, 1,
82:          &        ' GROUP ', ' ', IPR )
83:              call BMCML( PENRG(1,N), TEMP, KENRG(1,1,N), NGROUP )
84:              call ATRANS( TEMP, PENRG(1,N), NGROUP )
85:          end if
86:          end if
87:      else
88:          write(IPR,7020) IFISM(N)
89:          7020 format(/1X,' ENERGY DISTRIBUTION = FISSION SPECTRUM',
90:          &        ' OF ',I5,'TH NUCLIDE '/')
91:      end if
92: C
93:      100 continue
94: C
95:      return
96:      end

```

src/mvp/qp01.f

```
1:      function QPOL(S,X,Y,N)
2: C=====
3: C  PURPOSE :  INTERPOLATE QUADRATICALLY IN TABULAR FUNCTION Y(X), WHOSE
4: C              LENGTH IS N, TO GET QPOL, WHICH IS THE VALUE OF Y AT X=S.
5: C  CALLED IN:  RANGE
6: C=====
7: C
8:      implicit real(A-H,O-Z)
9: C
10:     real X(N), Y(N)
11: C
12: C
13:     if ( N.eq.2 ) then
14:         QPOL = Y(1) + (S-X(1))*(Y(2)-Y(1)) / (X(2)-X(1))
15: C
16:     else
17:         I = 1
18:         if ( X(1).le.X(N) ) then
19:             if ( S.le.X(2) ) go to 150
20:             if ( S.ge.X(N-1) ) go to 140
21:             do 100 I = 2, N - 2
22:                 if ( S.lt.X(I+1) ) go to 110
23:             100 continue
24:             110 continue
25:             if ( S.le.0.5*(X(I-1)+X(I+2)) ) I = I - 1
26:         else
27:             if ( S.ge.X(2) ) go to 150
28:             if ( S.le.X(N-1) ) go to 140
29:             do 120 I = 2, N - 2
30:                 if ( S.gt.X(I+1) ) go to 130
31:             120 continue
32:             130 continue
33:             if ( S.ge.0.5*(X(I-1)+X(I+2)) ) I = I - 1
34:         end if
35:         go to 150
36:     140 continue
37:         I = N - 2
38:     150 continue
39:         QPOL =
40:         & (S-X(I+1))*(S-X(I+2))*Y(I)/((X(I)-X(I+1))*(X(I)-X(I+2)))
41:         & + (S-X(I))*(S-X(I+2))*Y(I+1)/((X(I)-X(I+1))*(X(I+2)-X(I+1)))
42:         & + (S-X(I))*(S-X(I+1))*Y(I+2)/((X(I+2)-X(I))*(X(I+2)-X(I+1)))
43:     end if
44: C
45:     return
46: end
```

src/mvp/rangee.f

```

1:      subroutine RANGEE( CXE,   MMCXE, KLBE1, XLBE1, NEE,   EEL,   RNG,
2:      &                  NPATOM,NMAT, MLEN,  MPATM, LPDNP, IATMT,
3:      &                  NATMT, DNSTP, FME )
4: C=====
5: C  PURPOSE :  CALCULATE THE RANGE TABLE OF ELECTRON IN MATERIAL
6: C  CALLED IN:  XSECE
7: C=====
8: C
9: C  CXE(*)          R4  ELECTRON CROSS SECTION DATA
10: C  KLBE1(NPATOM,20) I4  SPECIFICATION PARAMETERS OF CXE ARRAY
11: C  XLBE1(NPATOM, 6) R4  ATOMIC PARAMETERS
12: C  NEE              I4  NUMBER OF ELECTRON ENERGY GRIDS
13: C  EEL(NEE)         R4  ELECTRON ENERGY GRIDS
14: C  RNG(NEE,NMAT)    R4  ELECTRON RANGE IN MATERIAL
15: C  NPATOM           I4  NUMBER OF ATOM ELEMENTS IN PROBLEM
16: C  NMAT             I4  NUMBER OF MATERIALS IN PROBLEM
17: C  MLEN             I4  LENGTH OF ATOMIC DATA FOR ALL MATERIALS
18: C  MPATM(NMAT)      I4  NUMBER OF ATOMIC ELEMENTS IN MATERIAL
19: C  LPDNP(NMAT+1)    I4  STARTING POINTER OF MATERIAL DATA
20: C  IATMT(MLEN)      I4  ATOMIC NUMBER TABLE BY MATERIALS
21: C  NATMT(MLEN)      I4  ATOMIC NUMBER TABLE USED IN PROBLEM
22: C  DNSTP(MLEN)      R4  ATOM NUMBER DENSITY TABLE INCLUDED IN EACH
23: C                    MATERIAL
24: C  FME(MLEN)        R4  ATOM FRACTION TABLE INCLUDED IN EACH MATERIAL
25: C
26: C  PV(12)           R4  MEAN IONIZATION POTENTIAL COEFFICIENCY FROM
27: C                    HYDROGEN TO MAGNESIUM BY STERNHEIMER, IN EV.
28: C
29: C=====
30: C
31:      implicit real*8(A-H,O-Z)
32: C
33:      parameter( ZERO = 0.0D0, ONE   = 1.0D0, THIRD = ONE/3.0D0,
34:      &          EIGHT = 8.0D0 )
35:      parameter( EMAS = 0.511008D+6, AVOGAD  = 0.60221367D0 )
36: C
37:      real CXE(MMCXE)
38: C
39:      real XLBE1(NPATOM,6), EEL(NEE), RNG(NEE,NMAT)
40:      integer KLBE1(NPATOM,20)
41: C
42:      real DNSTP(MLEN), FME(MLEN)
43:      integer MPATM(NMAT), LPDNP(NMAT+1), IATMT(MLEN), NATMT(NPATOM)
44: C
45:      real*8 PV(12), EL(10), G(10), BL(10)
46: C
47:      real QPOL
48: C
49:      data PV /18.7D0, 42.0, 38.0, 60.0, 71.0, 78.0, 85.0, 89.0, 92.0,
50:      &      131.0, 146.0, 156.0/
51: C
52: C=====
53: C
54: C-----
55: C  CALCULATE THE AVERAGE Z, AVERAGE ATOMIC WEIGHT, MASS AVERAGES OF
56: C  Z**(4/3) AND Z**2, AND AVERAGE DENSITY.
57: C-----
58: C
59:      EFAC   = 2.0**(-1.0/EIGHT)
60:      NC     = NEE - 1
61: C
62:      do 150 M = 1, NMAT
63: C
64:          AZ   = 0.0
65:          AA   = 0.0

```

```

66:          Q    = 0.0
67:          ZQ   = 0.0
68:          RO   = 0.0
69:          LP   = LPDNP(M) - 1
70: C
71:          do 100 J = 1, MPATM(M)
72: C
73:             IA   = IATMT(LP+J)
74:             IZ   = NATMT(IA)
75:             AZ   = AZ + FME(LP+J)*IZ
76: C
77:             AA   = AA + XLBE1(IA,1)*FME(LP+J)
78:             Q    = Q + XLBE1(IA,1)*FME(LP+J)*IZ**(4.0*THIRD)
79:             ZQ   = ZQ + XLBE1(IA,1)*FME(LP+J)*IZ**2
80: C
81:             RO   = RO + DNSTP(LP+J)
82: C
83:          100 continue
84: C
85:          QA    = 0.00001D0*Q/AA
86: C
87: C-----
88: C  CALCULATE THE MEAN-IONIZATION-POTENTIAL TERM
89: C-----
90: C
91:          SR    = 0.0
92:          do 110 J = 1, MPATM(M)
93: C
94:             IA   = IATMT(LP+J)
95:             IZ   = NATMT(IA)
96: C
97:             if ( IZ.lt.13 ) then
98:                P   = PV(IZ)
99:             else
100:                P   = 9.76D0*IZ + 58.8D0*IZ**(-0.19D0)
101:             end if
102: C
103:             SR    = SR + LOG(P)*IZ*FME(LP+J)
104: C
105:          110 continue
106: C
107:          SR      = EXP(SR/AZ)
108:          C2      = LOG(2.0*(SR/EMAS)**2)
109: C
110: C  .... SET UP THE ELECTRON SCATTERING DENSITY CORRECTION ....
111: C
112:          CB      = 2.0*LOG(SR/(37.1D0*SQRT(AZ*RO))) + 1.0
113: C
114:          XB      = 2 + MIN(1,INT(SR/100.0))
115:          XA      = MAX(0.2D0+ZERO,0.326D0*CB-0.5D0*XB)
116: C
117: C-----
118: C  CALCULATE RANGES FROM RADIATION AND IONIZATION LOSS RATES
119: C-----
120: C
121:          AF      = AVOGAD/AA
122:          YE      = 0.0
123: C
124:          do 140 NZ = 1, NC
125:             N     = NC + 1 - NZ
126:             E     = EEL(N)
127:             DE    = E*(1.0-EFAC)
128: C
129:          do 130 K = 1, 10
130:             S     = (E-(K-1)*DE/9.0D0) /EMAS

```

src/mvp/rangee.f

```

131: C
132: C     .... GET THE RADIATION LOSS BY INTERPOLATION IN CROSS SECTION
133: C     TABLE ....
134: C
135: C         BL(K)   = 0.0
136: C         ES      = EMAS*(S+1.0)
137: C         do 120 J = 1, MPATM(M)
138: C             IA   = IATMT(LP+J)
139: C             SL    = LOG(S)
140: C             BL(K) = BL(K) + ES*FME(LP+J)*QPOL(REAL(SL),
141: C &             CXE(KLBE1(IA,1))),
142: C &             CXE(KLBE1(IA,1))-KLBE1(IA,2)),KLBE1(IA,2))
143: C 120 continue
144: C
145: C     .... GET THE IONIZATION LOSS FROM ELECTRON SCATTERING THEORY ....
146: C
147: C         BS      = S*(S+2.0) / (S+1.0)**2
148: C         X       = 0.4342945*LOG(SQRT(S*(S+2.0)))
149: C         D       = 0.0
150: C         if ( X.gt.XA ) then
151: C             D    =
152: C &             MAX(ZERO,4.606*X-CB+(CB-4.606*XA)*
153: C &             ((XB-MIN(X,XB))/(XB-XA))**3)
154: C         end if
155: C         if ( K.eq.1 ) DD = D
156: C
157: C         EL(K)   = 0.499*EMAS*AZ*
158: C &             (LOG(S**2*(S+2.0))-C2+0.30585-ES+0.81815*
159: C &             (S/(S+1.0))**2-D) /BS
160: C
161: C         G(K)    = 1.0/(EL(K)+BL(K))
162: C
163: C 130 continue
164: C
165: C         if ( N.eq.NC ) then
166: C             RG    = 0.5*(E-DE)*G(10)
167: C             RNG(NEE,M) = RG
168: C         end if
169: C
170: C     .... AVERAGE THE RANGE INCREMENT OVER THE ENERGY INTERVAL ....
171: C
172: C         DR      = (G(1)+3.875*(G(2)+G(5))+2.625*(G(3)+G(4))+4.0*
173: C &             (G(7)+G(9))+2.0*(G(6)+G(8))+G(10))*DE/27.0
174: C
175: C         RG      = RG + DR
176: C
177: C         FE      = (G(1)*BL(1) + 3.875*(G(2)*BL(2) + G(5)*BL(5)) +
178: C &             2.625*(G(3)*BL(3) + G(4)*BL(4)) + 4.0*(G(7)*BL(7)
179: C &             + G(9)*BL(9)) + 2.0*(G(6)*BL(6) + G(8)*BL(8)) +
180: C &             G(10)*BL(10)) * DE / 27.0
181: C
182: C         YE      = YE + FE
183: C
184: C         RNG(N,M) = RG
185: C         DRS(N,M) = DR / NSB(M)
186: C
187: C     .... CALCULATE ENERGY LOSS AND STRAGGLING COEFFICIENTS ....
188: C
189: C         EF      = (EL(1)+EL(10)) / (BL(1)+BL(10)+EL(1)+EL(10))
190: C
191: C         QAV(N,M) = DE * EF / DR
192: C
193: C         IF( ISTRG.EQ.0 ) THEN
194: C             S    = (E - 0.5 * DE) / EMAS
195: C             T2   = (S + 1.0) ** 2
196: C
197: C             ASP(N,M) = 0.5 * EMAS * AZ * T2 / (S * (S + 2.0))
198: C             EAR(N,M) = LOG( (E - 0.5 * DE) / (DR * ASP(N,M)) )
199: C             &
200: C             QCN(N,M) = SQRT( QA * DE * EF ) / DR
201: C         ENDIF
202: C
203: C             BS = E * (E + 2.0 * EMAS) / (E + EMAS)**2
204: C 140 continue
205: C
206: C 150 continue
207: C
208: C         return
209: C         end

```

src/mvp/resti0.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTI0( NTHIST,WSUM,  WLEK,  IRAND, NBATCH,NFISB,
3:   &                  NGROUP,NRESP, NREG,  NTREG, NPKIND,NMEMO,
4:   &                  NZONE, NPART, NLATT, NEST,  NUC,   NEMIC,
5:   &                  NEMAC, NSTAL, NETALY,NETRV )
6: C=====
7: C  PURPOSE: input MVP restart information header (records existing only
8: C            as single record even in multitask mode.) and check validity.
9: C  CALLED IN:  INTR02
10: C  CALLS: (NONE)
11: C
12: C  I/O UNIT OF RESTART INPUT :  FT10F001  (IRIN)
13: C  I/O UNIT OF RESTART OUTPUT:  FT20F001  (IROT)
14: C
15: C-----
16:   include 'INC/_PROGV'
17:   include 'INC/_FLAGS'
18:   include '../shared/INC/_IOUNIT'
19:   include 'INC/_IOUNIT2'
20: C/#IF PARA(PVM)
21:   include '../shared/INC/_PVMPARA'
22: C/#ELSEIF PARA(MPI)
23:   * include 'mpif.h'
24: C/#ENDIF
25:   include '../shared/INC/_TASKDT'
26: C
27:   real*8 WSUM, WLEK
28: C/#IF INTEGER8
29:   * integer*8 NPART, NTHIST
30: C/#ELSE
31:   integer  NPART, NTHIST
32: C/#ENDIF
33: C
34: C ... local variables ...
35: C
36:   character*80 HD
37:   character*72 TP(2), VERSN(40)*60
38:   integer IWRK(32)
39: C/#IF INTEGER8
40:   * integer*8 I8WRK(4)
41: C/#ELSE
42:   integer  I8WRK(4)
43: C/#ENDIF
44: C
45: C-----
46: C
47:   call HEADER( IPR, 'RESTART FILE INPUT' )
48: C
49:   IPERR = 0
50: C
51: C ... in message psssing multitask mode, only main task input header
52: C
53: C/#IF PARA(PVM MPI)
54:   if ( IDTASK.eq.1 ) then
55: C/#ENDIF
56: C
57: C-----
58: C ..... READ FILE HEADER (80 CHARACTERS) .....
59: C-----
60:   read(IRIN,end =100) HD
61:   write(IPR,7000) HD
62: 7000 format(3X,'=== HEADER OF RESTART FILE ==='//5X,A/)
63:   go to 110
64: C
65: 100 continue

```

```

66:   if ( JARST.eq.0 ) then
67:     write(IMG,* ) 'XXX Restart file is empty !!'
68:     call CNTERR( 'FATAL' )
69:     IPERR = IPERR + 1
70:   else
71:     write(IPR,*)
72:     &      '<<MESSAGE>> (Auto-restart) Restart file is empty,',
73:     &      'so calculation is set to non-restart mode.'
74:     call CNTERR( 'MESSAGE' )
75:     JREST = 0
76:   end if
77:   return
78: C
79: 110 continue
80:   JREST = 1
81: C-----
82: C ..... READ PROBLEM TITLE IN RESTART FILE .....
83: C-----
84:   read(IRIN) (TP(I),I=1,2)
85:   write(IPR,7020) (TP(I),I=1,2)
86: 7020 format(/3X,'=== PROBLEM TITLE IN RESTART FILE ==='//5X,A/5X,A/)
87: C
88: C-----
89: C ..... READ PROGRAM VERSION DESCRIPTION. ....
90: C-----
91:   read(IRIN) (VERSN(I),I=1,40)
92:   write(IPR,7040) VERSN(40)
93: 7040 format(3X,'=== TYPE OF RESTART FILE : ',A/)
94: C
95: C/#IF PARA(PVM MPI)
96: C
97: C ... end if of "if( IDTASK.eq.1 ) ..."
98: C
99: end if
100: C
101:   if ( NTASK.gt.1 ) then
102:     MSGNUM = 12345
103:     IT0 = 1
104:     call MVPCOM_BCAST_CH( IT0, MSGNUM, HD, LEN(HD), INFO )
105:     call MVPCOM_BCAST_CH( IT0, MSGNUM, TP(1), LEN(TP(1)), INFO )
106:     call MVPCOM_BCAST_CH( IT0, MSGNUM, TP(2), LEN(TP(2)), INFO )
107:     call MVPCOM_BCAST_CH( IT0, MSGNUM, VERSN(40), LEN(VERSN(40)),
108:     &      INFO )
109:   end if
110: C/#ENDIF
111: C
112: C-----
113: C ..... RESTART FILE DOES NOT MATCH THIS VERSION !! .....
114: C-----
115:   if ( VERSN(40).ne.PROGV(40) ) then
116:     write(IMG,*) 'XXX I cannot read restart file of this type. '
117:     call CNTERR( 'FATAL' )
118:     return
119:   end if
120: C
121: C-----
122: C ... number of histories etc.( sum of all tasks ) ....
123: C
124: C  NRBUF0: read in buffer size of chopped restrt file records.
125: C-----
126: C/#IF PARA(PVM MPI)
127:   if ( IDTASK.eq.1 ) then
128: C/#ENDIF
129:     read(IRIN) NTASK2, NTHIST, NBATCH, NRBUF
130: C

```

src/mvp/resti0.f

```

131: C/#IF PARA(PVM MPI)
132: C
133: C      ... end if of "if( IDTASK.eq.1 ) ..."
134: C
135: C      end if
136: C
137: C      if ( NTASK.gt.1 ) then
138: C          MSGNUM = 345
139: C          IT0 = 1
140: C
141: C          if ( IDTASK.eq.IT0 ) then
142: C              IWRK(1) = NTASK2
143: C/#IF INTEGER8
144: C          * I8WRK(2) = NTHIST
145: C/#ELSE
146: C          IWRK(2) = NTHIST
147: C/#ENDIF
148: C          IWRK(3) = NBATCH
149: C          IWRK(4) = NRBUFF
150: C      end if
151: C
152: C      call MVPCOM_BCAST_I4( IT0, MSGNUM, IWRK, 4, INFO )
153: C/#IF INTEGER8
154: C      * call MVPCOM_BCAST_I8( IT0, MSGNUM+1, I8WRK, 4, INFO )
155: C/#ENDIF
156: C
157: C      if ( IDTASK.ne.IT0 ) then
158: C          NTASK2 = IWRK(1)
159: C/#IF INTEGER8
160: C          * NTHIST = I8WRK(2)
161: C/#ELSE
162: C          NTHIST = IWRK(2)
163: C/#ENDIF
164: C          NBATCH = IWRK(3)
165: C          NRBUFF = IWRK(4)
166: C      end if
167: C      end if
168: C/#ENDIF
169: C
170: C      write(IPR,7060) NTASK2, NTHIST, NBATCH, NRBUFF
171: C      7060 format(3X,'=== STATUS OF RESTART FILE (TASK COMMON) ==='/1X,
172: C      & '      ' NUMBER OF TASKS (NTASK) = ',I10/1X,
173: C      & '      ' NUMBER OF HISTORIES (NTHIST) = ',I10/1X,
174: C      & '      ' NUMBER OF BATCHES (NBATCH) = ',I10/1X,
175: C      & '      ' INPUT BUFFER SIZE (NRBUF) = ',I10/1X)
176: C
177: C      if ( NTASK2.ne.NTASK ) then
178: C          write(IMG,7080) NTASK2, NTASK
179: C      7080 format(/1X,'XXX(RESTI0) Restart file contains information of
180: C      & '      I4,' tasks and it does not match the current one ',I4,
181: C      & '      '.)
182: C          call CNTERR( 'FATAL' )
183: C      end if
184: C
185: C      .... MRBUF is defined in include file ../shared/_TASKDT currently
186: C
187: C      if ( NRBUFF.gt.MRBUF ) then
188: C          write(IMG,7100) NRBUFF, MRBUF
189: C      7100 format(/1X,'XXX(RESTI0) Input buffer size of restart file ',I6,
190: C      & '      ' is greater than that of the program (MRBUF=',I6,')'/
191: C      & '      'Please modify MRBUF and recompile the program.')
192: C          call CNTERR( 'FATAL' )
193: C      end if
194: C
195: C-----

```

```

196: C      ... VARIABLE PARAMETERS ....
197: C-----
198: C      read(IRIN) NTHIST, WSUM, IRAND, NBATCH, NFISB, WLEK
199: C
200: C      write(IPR,7060) NTHIST, WSUM, IRAND, NBATCH, NFISB, WLEK
201: C      7060 format(3X,'=== STATUS OF RESTART FILE ==='/1X,
202: C      & '      ' HISTORY NUMBER (NTHIST) = ',I10/1X,
203: C      & '      ' SUM OF WEIGHT (WSUM) = ',D13.5/1X,
204: C      & '      ' LAST RANDOM NUMBER (IRAND) = ',I10/1X,
205: C      & '      ' BATCH NUMBER (NBATCH) = ',I10/1X,
206: C      & '      ' FISSION SOURCE (NFISB) = ',I10/1X,
207: C      & '      ' LEAKAGE (WLEK) = ',D13.5)
208: C
209: C
210: C
211: C      if ( NPART.gt.0.and.NPART.le.NTHIST ) then
212: C          write(IMG,7120) NPART, NTHIST
213: C      7120 format(1X,'!!! YOU INTEND TO RUN TOTAL OF ',I9,' HISTORIES',
214: C      & '      ', BUT THE RESTART FILE HAS RESULTS OF ',I9,
215: C      & '      ' HISTORIES.'/
216: C      & '      ' SO MVP RUNS NO HISTORY AND ONLY ANALYZES TALLIES.'/
217: C      & '      ' 1X,' TO RUN MORE N HISTORIES INPUT AS NPART( -N ). '/')
218: C      else if ( NPART.lt.0 ) then
219: C          write(IPR,7140) NTHIST + 1, NTHIST + ABS(NPART)
220: C      7140 format(5X,'*** HISTORIES IN CURRENT RUN: FROM ',I10,' TO ',I10)
221: C      else if ( NPART.gt.0 ) then
222: C          write(IPR,7160) NTHIST + 1, NPART, NPART - NTHIST
223: C      7160 format(5X,'*** HISTORIES TO BE RUN: FROM ',I10,' TO ',I10,
224: C      & '      ' (',I10,' HISTORIES)')
225: C      end if
226: C-----
227: C      .... READ NON-VARIABLE PARAMETERS .....
228: C-----
229: C      IPERR = 0
230: C
231: C/#IF PARA(PVM MPI)
232: C      if ( IDTASK.eq.1 ) then
233: C/#ENDIF
234: C
235: C
236: C      ... until file version 3.0
237: C
238: C      read(IRIN) KGROUP, KPKIND, KNREG, KNTREG, KNRESP, KNMEMO, KNZONE,
239: C      & KFBANK, KNLATT, KNEST, KNUC, KNEMIC, KNEMAC, KNSTAL,
240: C      & KNETAL, KJEIGN, KJRESP, KJHLAT, KJTLLT
241: C
242: C      ... from file version 3.1
243: C
244: C      read(IRIN) KGROUP, KPKIND, KNREG, KNTREG, KNRESP, KNMEMO,
245: C      & KNZONE, KFBANK, KNLATT, KNEST, KNUC, KNEMIC, KNEMAC,
246: C      & KNSTAL, KNETAL, KNETRV
247: C      read(IRIN) KJEIGN, KJRESP, KJHLAT, KJTLLT
248: C
249: C/#IF PARA(PVM MPI)
250: C
251: C      ... end if of "if( IDTASK.eq.1 ) ..."
252: C
253: C      end if
254: C
255: C      if ( NTASK.gt.1 ) then
256: C          MSGNUM = 2345
257: C          IT0 = 1
258: C          if ( IDTASK.eq.IT0 ) then
259: C              IWRK(1) = KGROUP
260: C              IWRK(2) = KPKIND

```

src/mvp/resti0.f

```

261:      IWRK(3) = KNREG
262:      IWRK(4) = KNTREG
263:      IWRK(5) = KNRESP
264:      IWRK(6) = KNMEMO
265:      IWRK(7) = KNZONE
266:      IWRK(8) = KFBANK
267:      IWRK(9) = KNLATT
268:      IWRK(10) = KNEST
269:      IWRK(11) = KNUC
270:      IWRK(12) = KNEMIC
271:      IWRK(13) = KNEMAC
272:      IWRK(14) = KNSTAL
273:      IWRK(15) = KNETAL
274:      IWRK(16) = KNETRV
275:      IWRK(17) = KJEIGN
276:      IWRK(18) = KJRESP
277:      IWRK(19) = KJHLAT
278:      IWRK(20) = KJTLLT
279:      end if
280:
281:      call MVPCOM_BCAST_I4( IT0, MSGNUM, IWRK, 20, INFO )
282:
283:      if ( IDTASK.ne.IT0 ) then
284:         KGROUP = IWRK(1)
285:         KPKIND = IWRK(2)
286:         KNREG = IWRK(3)
287:         KNTREG = IWRK(4)
288:         KNRESP = IWRK(5)
289:         KNMEMO = IWRK(6)
290:         KNZONE = IWRK(7)
291:         KFBANK = IWRK(8)
292:         KNLATT = IWRK(9)
293:         KNEST = IWRK(10)
294:         KNUC = IWRK(11)
295:         KNEMIC = IWRK(12)
296:         KNEMAC = IWRK(13)
297:         KNSTAL = IWRK(14)
298:         KNETAL = IWRK(15)
299:         KNETRV = IWRK(16)
300:         KJEIGN = IWRK(17)
301:         KJRESP = IWRK(18)
302:         KJHLAT = IWRK(19)
303:         KJTLLT = IWRK(20)
304:      end if
305:      end if
306: C/#ENDIF
307: C
308:      if ( KJEIGN.ne.JEIGN .or. KJRESP.ne.JRESP .or. KJHLAT.ne.JHLAT
309:         & .or. KJTLLT.ne.JTLLT ) then
310:
311:         write(IMG,7180) KJEIGN, KJRESP, KJHLAT, KJTLLT
312: 7180      format(/1X,'XXX(RESTI0) Calculation options in restart file',
313:         &      ' do not match those from current input! STOP'/
314:         &      ' JEIGN = ',I2,' JRESP = ',I2,' JHLAT = ',I2,
315:         &      ' JTLLT = ',I2)
316:
317:         call CNTERR( 'FATAL' )
318:         IPERR = IPERR + 1
319:      end if
320: C
321:      if ( KGROUP.ne.KGROUP .or. KNREG.ne.NREG .or. KNRESP.ne.NRESP
322:         & .or. KNTREG.ne.NTREG .or. KPKIND.ne.NPKIND .or. KNMEMO.ne.NMEMO
323:         & .or. KNZONE.ne.NZONE .or. KNLATT.ne.NLATT .or. KNEST.ne.NEST
324:         & .or. KNUC.ne.NUC .or. KNEMIC.ne.NEMIC .or. KNEMAC.ne.NEMAC
325:         & .or. KNSTAL.ne.NSTAL .or. KNETAL.ne.NETALY .or. KNETRV.ne.NETRV

```

```

326:         & ) then
327:         write(IMG,7200) KGROUP, NGROUP, KNREG, NREG, KNRESP, NRESP,
328:         &      KNMEMO, NMEMO, KNZONE, NZONE, KNLATT, NLATT, KNEST,
329:         &      NEST, KNUC, NUC, KNEMIC, NEMIC, KNEMAC, NEMAC, KNSTAL,
330:         &      NSTAL, KNTREG, NTREG, KPKIND, NPKIND, KNETAL, NETAL,
331:         &      KNETRV, NETRV
332: 7200      format(/1X,'XXX(RESTI0) One or more size parameters in',
333:         &      ' restart file do not match those from current input!'/
334:         &      ' NGROUP = ',2I7,' NREG = ',2I7,' NRESP = ',2I7/
335:         &      ' NMEMO = ',2I7,' NZONE = ',2I7,' NLATT = ',2I7/
336:         &      ' NEST = ',2I7,' NUC = ',2I7,' NEMIC = ',2I7/
337:         &      ' NEMAC = ',2I7,' NSTAL = ',2I7,' NTREG = ',2I7/
338:         &      ' NPKIND = ',2I7,' NETALY= ',2I7,' NETRV = ',2I7/)
339:         call CNTERR( 'FATAL' )
340:         IPERR = IPERR + 1
341:      end if
342: C
343:      return
344:      end

```

src/mvp/restin.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine RESTIN( IOPR, H, ITSK,TITLE )
3: C=<MVP>=====
4: C PURPOSE: Interface routine to RESTRT which inputs restart file record.
5: C CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C CALLS:
7: C
8: C-----
9: C
10: C arguments ( i = input, o = output, c =constant, w = work )
11: C
12: C i IOPR : message printout I/O unit
13: C io H : task local dynamic memory array
14: C i ITSK : data are stored on local data of task ITSK.
15: C (ITSK may not be same as current IDTASK)
16: C i TITLE : problem titile
17: C-----
18:      real H(*)
19:      character TITLE(2)*72
20: C
21: CCCC include '../shared/INC/_ARRAY'
22:      include '../shared/INC/_TASKDT'
23:      include '../shared/INC/_LISTOFF'
24: C
25:      include 'INC/_KPIDS'
26:      include 'INC/_NGPS'
27: C
28:      include 'INC/_FLAGS'
29:      include '../shared/INC/_SIZES'
30:      include 'INC/_SIZES2'
31:      include 'INC/_CXSEC'
32:      include 'INC/_PXSEC'
33:      include 'INC/_XBANK'
34:      include 'INC/_SBANK'
35:      include 'INC/_STACK'
36:      include 'INC/_XWORK'
37:      include '../shared/INC/_PGEOM'
38:      include '../shared/INC/_PVRED'
39:      include 'INC/_PSOUR'
40:      include '../shared/INC/_PTALY0'
41:      include 'INC/_PTALY'
42:      include 'INC/_PTALY2'
43:      include '../shared/INC/_STALY'
44:      include '../shared/INC/_PTLSP'
45:      include '../shared/INC/_LISTON'
46:      include '../shared/INC/_WORDL'
47:      include 'INC/_PERT'
48:      include 'INC/_WKSOU'
49: C
50: C-----
51: C
52:      call RESTRT( IOPR, ITSK, NTHIST, IRAND, H(LWSUM), H(LWLEK),
53: &                NGROUP, NRESP, NREG, NTREG, NPKIND, NBATCH, NMEMO,
54: &                NEVENT, NZONE, H(LKMEMO), H(LWCNTR), H(LNCNTR),
55: &                H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
56: &                H(LXAVT), H(LAAVT),H(LEAVT),
57: &                NFISB, NFBANK,NFBNK0, NLENG, H(LXXXF), H(LYYYYF),
58: &                H(LZZZF), H(LEEEF), H(LIZZF), H(LINUF), H(LIBRGF),
59: &                H(LIBSPF), H(LXSOC), H(LXSXV), H(LXKEF), NLATT, NEST,
60: &                H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
61: &                H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK, MIFBK,
62: &                NUC, NEMIC,
63: &                NEMAC, H(LSRMIC), H(LRMICR), H(LXMIC), H(LRMAC),
64: &                H(LRMACR), H(LXMAC), H(LSRSTR), H(LSRACL), NSTAL,
65: &                H(LWCXTY), H(LDNFLX), NETALY, H(LETALY), NLETAL,

```

```

66:      &                NETRV, NSKIP, METRV, H(LETRV),
67:      &                NBANK, NPTDS, NPTCS, NGSP, NTPT, NUCPT,
68:      &                H(LWSD), H(LWSC), H(LWD0), H(LWR0),
69:      &                H(P1(23)), H(P1(24)), H(LXKEFP), NOPSDS,
70:      &                H(LIMTF),
71:      &                NGP(KPNEUT),H(LSRMICSM),H(LXMICSM),H(LRMACSM),
72:      &                H(LXMACSM),
73:      &                H(LSMIMU),H(LRMAMU),
74:      c+beff1
75:      &                NEBEF, H(LWCBEF), H(LXBEF),
76:      c-beff1
77:      c+beff2
78:      &                H(LWSB), H(LWB0), H(P1(22)), NPTBE,
79:      &                H(LWSBD), H(LWBD0), H(P1(21)),
80:      &                H(LWSLD), H(LWLD0), H(P1(20)))
81:      c-beff2
82: C
83:      return
84:      end

```


src/mvp/resto0.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine RESTO0( IOPR, H, MONITR, TITLE, KTHIST, KBATCH )
3: C=<MVP>=====
4: C   PURPOSE:  interface routine to restart file header output routine.
5: C   CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C   CALLS: RESTO1
7: C-----
8: C   arguments ( i = input, o = output, c =constant, w = work )
9: C
10: C i IOPR :  message printout I/O unit
11: C io H   :  task local dynamic memory array
12: C i MONITR : do not print any message in RESTO1 if zero
13: C i TITLE : problem titile
14: C i KTHIST : total history number (sum for tasks)
15: C i KBATCH : total batch number (sum for tasks)
16: C=====
17:   real H(*)
18:   character TITLE(2)*72
19: CCCC include '../shared/INC/_ARRAY'
20:   include '../shared/INC/_LISTOFF'
21:   include '../shared/INC/_TASKDT'
22: C
23:   include 'INC/_KPIDS'
24:   include 'INC/_NGPS'
25: C
26:   include 'INC/_FLAGS'
27:   include '../shared/INC/_SIZES'
28:   include 'INC/_SIZES2'
29:   include '../shared/INC/_PGEOM'
30:   include 'INC/_XWORK'
31:   include 'INC/_CXSEC'
32:   include '../shared/INC/_PTALY0'
33:   include 'INC/_PTALY'
34:   include 'INC/_PTALY2'
35:   include '../shared/INC/_STALY'
36:   include '../shared/INC/_PTLSP'
37:   include 'INC/_XBANK'
38:   include 'INC/_STACK'
39:   include 'INC/_SBANK'
40: C
41:   include 'INC/_WKTLO'
42:   include '../shared/INC/_LISTON'
43: C
44:   include '../shared/INC/_WORDL'
45: C
46: C/#IF INTEGER8
47: *      integer*8 KTHIST
48: C/#ELSE
49:   integer      KTHIST
50: C/#ENDIF
51: C
52: C ..... OUTPUT TO RESTART FILE .....
53: C
54:   call RESTO1( IOPR, MONITR, TITLE,
55: &              KTHIST,H(LWSUM), H(LWLEK), KBATCH,
56: &              NGROUP,NRESP, NREG,  NTREG, NPKIND,NMEMO,
57: &              NZONE, NPART, NLATT, NEST,  NUC,   NEMIC,
58: &              NEMAC, NSTAL, NETALY, NETRV )
59: C
60:   return
61: end
```

src/mvp/restol.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTOL( IOPR, MONITR, TITLE,
3:     &                NTHIST,WSUM, WLEK, NBATCH,
4:     &                NGROUP,NRESP, NREG, NTREG, NPKIND,NMEMO,
5:     &                NZONE, NPART, NLATT, NEST, NUC, NEMIC,
6:     &                NEMAC, NSTAL, NETALY, NETRV )
7: C=====
8: C  PURPOSE: output MVP restart information header(records existing only
9: C           as single record even in multitask mode.)
10: C  CALLED IN: RESTO0
11: C  CALLS: (NONE)
12: C
13: C  I/O UNIT OF RESTART INPUT : FT10F001 (IRIN)
14: C  I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
15: C
16: C=====
17:   include 'INC/ PROGV'
18:   include 'INC/ FLAGS'
19:   include '../shared/INC/_IOUNIT'
20:   include 'INC/_IOUNIT2'
21: C/#IF PARA(PVM)
22:   include '../shared/INC/_PVMPARA'
23: C/#ELSEIF PARA(MPI)
24: *   include 'mpif.h'
25: C/#ENDIF
26:   include '../shared/INC/_TASKDT'
27: C
28:   character*72 TITLE(2)
29:   real*8 WSUM, WLEK
30: C/#IF INTEGER8
31: *   integer*8 NPART, NTHIST
32: C/#ELSE
33:   integer NPART, NTHIST
34: C/#ENDIF
35: C
36: C ... local variables ...
37: C
38:   character*80 HD
39:   character*12 DT, TM
40: C
41: C-----
42: C
43: ccc  call HEADER( IPR, 'RESTART FILE OUTPUT' )
44: C
45: C ... in message psssing multitask mode, only main task input header
46: C
47: C/#IF PARA(PVM MPI)
48:   if( IDTASK.eq.1 ) then
49: C/#ENDIF
50: C
51: C
52: C-----
53: C ..... DATE & TIME OF OUTPUT .....
54: C-----
55: C
56:   DT = ' '
57:   TM = ' '
58:   call HIZUKE( DT )
59:   call JIKAN( TM(1:8) )
60: C
61:   HD = ' '
62: C
63: C   HD = 'MVP RESTART FILE V3.0 //'DT/' '/'TM
64: C
65:

```

```

66: C ... version 3.1 (from 2 Oct 1998)
67: C
68: C * size/flag record is separated into two record and NETRV is added.
69: C
70: C   HD = 'MVP RESTART FILE V3.1 //'DT/' '/'TM
71: C
72: C ... version 3.2 (from 1 Feb 1999)
73: C
74: C * output FDBNK and IDBNK if necessary.
75: C
76: C   HD = 'MVP RESTART FILE V3.2 //'DT/' '/'TM
77: C
78: C ... version 3.3 (from 21 Aug 2003)
79: C
80: C * output IRNSU, IRNSM, IRNSL and RNCTR. A seed value of a 63-bit RN
81: C   is split into 3 parts. RNCTR is the number of used RNs.
82: C
83: C   HD = 'MVP RESTART FILE V3.3 //'DT/' '/'TM
84: C
85: C ... version 3.4 (from 03 May 2004)
86: C
87: C * output restart date for perturbation & IMTF for delayed neutrons
88: C
89: C   HD = 'MVP RESTART FILE V3.4 //'DT/' '/'TM
90: C
91:   if( MONITR.ne.0 ) write(IOPR,7020) (TITLE(I),I=1,2)
92: 7020 format(//1X,' ==== OUTPUT TO RESTART FILE ==== '//1X,
93:   &        ' Problem title :',A72/1X,16X,A72/)
94: C
95:   if( MONITR.ne.0 ) write(IOPR,7000) HD
96: 7000 format(3X,'=== Header of restart file ===//5X,A/)
97: C-----
98: C ..... write file header (80 CHARACTERS) .....
99: C-----
100:   write(IROT) HD
101: C-----
102: C ..... write problem title in restart file .....
103: C-----
104:   write(IROT) (TITLE(I),I=1,2)
105: C
106: C-----
107: C ..... write program version description. ....
108: C-----
109:   write(IROT) (PROGV(I),I=1,40)
110: C
111: C-----
112: C ... number of histories etc.( sum of all tasks ) ...
113: C
114: C MRBUF: read in buffer size of chopped restrtr file records.
115: C-----
116:   write(IROT) NTASK, NTHIST, NBATCH, MRBUF
117: C-----
118: C ..... write non-variable parameters .....
119: C-----
120:   IPERR = 0
121: C
122: C .. until v3.0
123: C
124: C   write(IROT) NGROUP, NPKIND, NREG, NTREG, NRESP, NMemo, NZONE,
125: C   & NFBANK, NLATT, NEST, NUC, NEMIC, NEMAC, NSTAL,
126: C   & NETALY, JEIGN, JRESP, JHLAT, JTLT
127: C
128: C .. after v3.1
129: C
130:   write(IROT) NGROUP, NPKIND, NREG, NTREG, NRESP, NMemo, NZONE,

```

src/mvp/restol.f

```
131:      &          NFBANK, NLATT, NEST, NUC, NEMIC, NEMAC, NSTAL,  
132:      &          NETALY, NETRV  
133:      write(IROT) JEIGN, JRESP, JHLAT, JTLLT  
134: C  
135: C/#IF PARA(PVM MPI)  
136:      end if  
137: C/#ENDIF  
138:      return  
139:      end
```

SAFE

src/mvp/resto.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTO( IOPR, MONITR,ITSK, NTHIST,IRAND, WSUM, WLEK,
3:   &                 NGROUP,NRESP, NREG, NTREG, NPKIND,NBATCH,NMEMO,
4:   &                 NEVENT,NZONE, KMEMO, WCNTR, NCNTR, SFLTR, SFLCL,
5:   &                 SRETR, SRECL, XAVT, AAVT, EAVT, NFISB,
6:   &                 NFBANK,NFBNK0,NLENG, XXXF, YYYY, ZZZF, EEEF,
7:   &                 IZZF, INUF, IBRGF, IBSPF, XSOC, XSXV, XKEF,
8:   &                 NLATT, NEST, LEVLF, LZZF, LPOSF, LCRSF, DFBK,
9:   &                 KDFBK, MDFBK, IFBK, KIFBK, MIFBK, NUC, NEMIC,
10:  &                 NEMAC, SRMIC, RMICR, XMIC, RMAC, RMACR, XMAC,
11:  &                 SRSTR, SRSCl, NSTAL, WCXTY, DNFLX, NETALY,ETALY,
12:  &                 NLETAL,NETRV, NSKIF, METRV, ETRV,
13:  &                 NBANK, NPTDS, NPTCS, NGSP, NTPT, NUCPT,
14:  &                 WSD, WSC, WD0, WR0, SWD0, SWR0, XKEFP, NOPSDS,
15:  &                 IMTR,
16:  &                 NGP1,SRMICSM,XMICSM,RMACSM,XMACSM,
17:  &                 SMIMU, RMAMU,
18:  c+beff1
19:  &                 NEBEF, WCBEF, XBEF,
20:  c-beff1
21:  c+beff2
22:  &                 WSB, WB0, SWB0, NPTBE,
23:  &                 WSD, WBD0, SWBD0, WSLD, WLD0, SWLD0)
24:  c-beff2
25:  C=<MVP>=====
26:  C  PURPOSE: Output restart information
27:  C      (header - parameter must be input in RESTI0)
28:  C
29:  C  CALLED IN: RESTOT
30:  C  CALLS: SRECO[I|R|D]
31:  C
32:  C  I/O UNIT OF RESTART INPUT : FT10F001 (IRIN)
33:  C  I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
34:  C
35:  C=====
36:  include 'INC/_FLAGS'
37:  include '../shared/INC/_IOUNIT'
38:  include 'INC/_IOUNIT2'
39:  C/#IF PARA(PVM)
40:  include '../shared/INC/_PVMPARA'
41:  C/#ELSEIF PARA(MPI)
42:  * include 'mpif.h'
43:  C/#ELSEIF PARA(VPP)
44:  include '../shared/INC/_VPPPARA'
45:  C/#ENDIF
46:  include '../shared/INC/_TASKDT'
47:  C
48:  C =<memo>= The seed value is saved as common variables
49:  C      for a restart calculation. This treatment will be
50:  C      modified in the future.
51:  include '../shared/INC/_RAND'
52:  C
53:  real*8 WSUM, WLEK
54:  C/#IF INTEGER8
55:  * integer*8 NTHIST
56:  C/#ELSE
57:  integer NTHIST
58:  C/#ENDIF
59:  C
60:  C      ..... EVENT MONITOR.....
61:  C
62:  real*8 WCNTR(NEVENT,NPKIND), NCNTR(NEVENT,NPKIND)
63:  real XAVT(3), AAVT(3), EAVT
64:  C
65:  C      ..... NEXT-ZONE MEMORY .....

```

```

66:  C
67:  integer KMEMO(NZONE,NMEMO)
68:  C
69:  C      ..... TALLY ARRAYS .....
70:  C
71:  real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
72:  &       SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
73:  C
74:  real*8 SRMIC(NGROUP,NTREG,NUC,NEMIC,2),
75:  &       RMICR(NPKIND,NTREG,NUC,NEMIC,2),
76:  &       XMIC(NGROUP+NPKIND,NTREG,NUC,NEMIC,2),
77:  &       DNFLX(NGROUP,NREG,NUC)
78:  C
79:  real*8 RMAC(NGROUP,NTREG,NEMAC,2), RMACR(NPKIND,NTREG,NEMAC,2),
80:  &       XMAC(NGROUP+NPKIND,NTREG,NEMAC,2)
81:  real*8 WCXTY(NGROUP+NPKIND,NTREG)
82:  real*8 SRSTR(NTREG,NSTAL,2), SRSCl(NTREG,NSTAL,2)
83:  real*8 ETALY(NLETAL,2)
84:  real*8 ETRV(METRV,*)
85:  c%c &       XMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,*),
86:  c%c &       XMACSM(NGP1+1,NGP1+1,NTREG,2,*),
87:  real*8 SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,JSCTM,*),
88:  &       RMACSM(NGP1+1,NGP1+1,NTREG,2,JSCTM,*)
89:  real*8 SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,*),
90:  &       RMAMU(NGP1+1,NGP1+2,NTREG,3,*)
91:  C
92:  C      ... EIGENVALUE TALLY .....
93:  C
94:  c##<2007/03/14:PN4:
95:  c## real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(7,NLENG)
96:  real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(10,NLENG)
97:  c##>
98:  C
99:  C      ... FISSION PARTICLE (SOURCE) BANK ....
100: C
101: real*8 XXXF(NFBNK0), YYYY(NFBNK0), ZZZF(NFBNK0)
102: integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
103: &       LCRSF(NFBNK0,NEST), IZZF(NFBNK0), INUF(NFBNK0)
104: integer IBRGF(NFBNK0), IBSPF(NFBNK0,0:NEST)
105: real EEEF(NFBNK0)
106: integer IMTR(NFBNK0)
107: C
108: real*8 DFBK(NFBNK0,*)
109: integer KDFBK(0:MDFBK)
110: integer IFBK(NFBNK0,*)
111: integer KIFBK(0:MIFBK)
112: C
113: C      ... PERTURBATION .....
114: C
115: real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
116: real*8 WSC(NBANK,0:NGSP,NPTCS)
117: real*8 WD0(NFBNK0,0:NGSP,NPTDS,NOPSDS)
118: real*8 WR0(NFBNK0,0:NGSP,NPTCS)
119: real*8 SWD0(0:NGSP,NPTDS,NOPSDS)
120: real*8 SWR0(0:NGSP,NPTCS)
121: real*8 XKEFP(NUCPT,NTPT,7,NLENG)
122: c+beff1
123: C
124: C      ... BETA EFFECTIVE .....
125: C
126: real*8 XBEF(NEBEF,NLENG), WCBEF(NEBEF)
127: c-beff1
128: c+beff2
129: real*8 WSB(NBANK,0:NGSP)
130: real*8 WB0(NFBNK0,0:NGSP)

```

src/mvp/resto.f

```

131:      real*8 SWB0(0:NGSP)
132:      real*8 WSBD(NBANK,NGSP)
133:      real*8 WBD0(NFBNK0,NGSP)
134:      real*8 SWBD0(NGSP)
135:      real*8 WSLD(NBANK,NGSP)
136:      real*8 WLD0(NFBNK0,NGSP)
137:      real*8 SWLD0(NGSP)
138: c-beff2
139: C
140: C ..... local data .....
141: C
142:      character*32 TAG
143:      integer IWRK(32)
144:      real RWRK(32)
145:      real*8 DWRK(32)
146: C
147: C-----
148: C
149:      IERR = 0
150: C
151: C ... Divide NTHIST into 2 parts for more than (2**31-1) histories.
152: C This is the treatment not to change restart output format.
153: C It should be modified in the future.
154: C
155:      I4MAX = 2147483647 ! 2**31-1
156:      IQNTH = int(NTHIST/I4MAX)
157:      IRNTH = NTHIST - IQNTH*I4MAX
158: C
159: C ... idtask, NTHIST, IRAND, NBATCH, NFISB .....
160: C
161:      IWRK(1) = ITSK
162:      IWRK(2) = NTHIST
163:      IWRK(3) = IRAND
164:      IWRK(4) = NBATCH
165:      IWRK(5) = NFISB
166:      IWRK(6) = IRNSU
167:      IWRK(7) = IRNSM
168:      IWRK(8) = IRNSL
169:      IWRK(9) = IQNTH
170:      IWRK(10) = IRNTH
171:      NPKI = 10
172:      call SRECOI( IROT, ITSK, 'TASKHEADER', IWRK, NPKI, IERR )
173: C
174:      DWRK(1) = WSUM
175:      DWRK(2) = WLEK
176:      DWRK(3) = RNCTR
177:      NPKD = 3
178:      call SRECOD( IROT, ITSK, 'WSUM,WLEK', DWRK, NPKD, IERR )
179: C
180:      if ( MONITR.ne.0 ) write(IOPR,*) ' === Task ', IDTASK,
181:      & ' output task wise restart data header '
182: C
183: c###<2007/03/14:PN3:
184: c7000 format(1X,A,I12)
185: 7000 format(1X,A,I14)
186: c###>
187: 7020 format(1X,A,1P,E14.6)
188: C
189:      if ( MONITR.ne.0.and.IDTASK.eq.1.and.ITSK.eq.1 ) then
190:
191:          write(IOPR,7000) ' IDTASK marked on record: ', ITSK
192:          write(IOPR,7000) ' NTHIST : ', NTHIST
193:          write(IOPR,7000) ' IRAND : ', IRAND
194:          write(IOPR,7000) ' NBATCH : ', NBATCH
195:          write(IOPR,7000) ' NFISB : ', NFISB

```

```

196:          write(IOPR,7020) ' WSUM : ', WSUM
197:          write(IOPR,7020) ' WLEK : ', WLEK
198:          write(IOPR,7000) ' IRNSU : ', IRNSU
199:          write(IOPR,7000) ' IRNSM : ', IRNSM
200:          write(IOPR,7000) ' IRNSL : ', IRNSL
201:          write(IOPR,7020) ' RNCTR : ', RNCTR
202:          write(IOPR,7000) ' IQNTH : ', IQNTH
203:          write(IOPR,7000) ' IRNTH : ', IRNTH
204:
205:      else
206:
207: C/#IF PARA(PVM MPI)
208:      MSGNUM = 111
209:      IT0 = 1
210:      if ( IDTASK.ne.IT0 ) then
211:          call MVPCOM_SEND_I4( IT0, MSGNUM, IWRK, NPKI, INFO )
212:          call MVPCOM_SEND_R8( IT0, MSGNUM, DWRK, NPKD, INFO )
213:      else
214:          call MVPCOM_RECV_I4( ITSK, MSGNUM, IWRK, NPKI, INFO )
215:          call MVPCOM_RECV_R8( ITSK, MSGNUM, DWRK, NPKD, INFO )
216:      end if
217:      write(IOPR,7000) ' IDTASK marked on record: ', ITSK
218:      write(IOPR,7000) ' NTHIST : ', IWRK(2)
219:      write(IOPR,7000) ' IRAND : ', IWRK(3)
220:      write(IOPR,7000) ' NBATCH : ', IWRK(4)
221:      write(IOPR,7000) ' NFISB : ', IWRK(5)
222:      write(IOPR,7020) ' WSUM : ', DWRK(1)
223:      write(IOPR,7020) ' WLEK : ', DWRK(2)
224:      write(IOPR,7000) ' IRNSU : ', IWRK(6)
225:      write(IOPR,7000) ' IRNSM : ', IWRK(7)
226:      write(IOPR,7000) ' IRNSL : ', IWRK(8)
227:      write(IOPR,7020) ' RNCTR : ', DWRK(3)
228:      write(IOPR,7000) ' IQNTH : ', IWRK(9)
229:      write(IOPR,7000) ' IRNTH : ', IWRK(10)
230: C/#ENDIF
231:      end if
232:
233:      call SRECOI( IROT, ITSK, 'KMEMO', KMEMO, NZONE*NMEMO, IERR )
234:      call SRECOD( IROT, ITSK, 'WCNTR', WCNTR, NEVENT*NPKIND, IERR )
235:      call SRECOD( IROT, ITSK, 'NCNTR', NCNTR, NEVENT*NPKIND, IERR )
236:
237:      RWRK(1) = XAVT(1)
238:      RWRK(2) = XAVT(2)
239:      RWRK(3) = XAVT(3)
240:      RWRK(4) = AAVT(1)
241:      RWRK(5) = AAVT(2)
242:      RWRK(6) = AAVT(3)
243:      RWRK(7) = EAVT
244:      call SRECOR( IROT, ITSK, 'XAVT,AAVT,EAVT', RWRK, 7, IERR )
245: C
246: C ..... FLUXES .....
247: C
248: C write(IROT) SFLTR
249: C write(IROT) SFLCL
250:
251:      call SRECOD( IROT, ITSK, 'SFLTR', SFLTR, NGROUP*NTREG*2, IERR )
252:      call SRECOD( IROT, ITSK, 'SFLCL', SFLCL, NGROUP*NTREG*2, IERR )
253: C
254: C ..... REACTION RATES .....
255: C
256:      if ( JRESP.ne.0 ) then
257:          if ( NRESP.gt.0 ) then
258: Cccc          write(IROT) SRETR
259: Cccc          write(IROT) SRECL
260:          call SRECOD( IROT, ITSK, 'SRETR', SRETR, NTREG*NRESP*2, IERR

```

src/mvp/resto.f

```

261:      &      )
262:      call SRECOD( IROT, ITSK, 'SRECL', SRECL, NTREG*NRESP*2, IERR
263:      &      )
264:      end if
265:      if ( NSTAL.gt.0 ) then
266:      Cccc      write(IROT) SRSTR
267:      Cccc      write(IROT) SRSCL
268:      call SRECOD( IROT, ITSK, 'SRSTR', SRSTR, NGROUP*NSTAL*2,
269:      &      IERR )
270:      call SRECOD( IROT, ITSK, 'SRSCL', SRSCL, NGROUP*NSTAL*2,
271:      &      IERR )
272:      end if
273:      end if
274: C
275: C ..... MICROSCOPIC REACTION RATE .....
276: C
277:      if ( NEMIC.gt.0 ) then
278:      Cccc      write(IROT) SRMIC
279:      Cccc      write(IROT) RMICR
280:      Cccc      write(IROT) XMIC
281:      call SRECOD( IROT, ITSK, 'SRMIC', SRMIC, NGROUP*NTREG*NUC*NEMIC
282:      &      *2, IERR )
283:      call SRECOD( IROT, ITSK, 'RMICR', RMICR, NPKIND*NTREG*NUC*NEMIC
284:      &      *2, IERR )
285:      call SRECOD( IROT, ITSK, 'XMIC', XMIC, (NGROUP
286:      &      +NPKIND)*NTREG*NUC*NEMIC*2, IERR )
287:      end if
288: C
289: C ..... MACROSCOPIC REACTION RATE .....
290: C
291:      if ( NEMAC.gt.0 ) then
292:      Cccc      write(IROT) RMAC
293:      Cccc      write(IROT) RMACR
294:      Cccc      write(IROT) XMAC
295:      call SRECOD( IROT, ITSK, 'RMAC', RMAC, NGROUP*NTREG*NEMAC*2,
296:      &      IERR )
297:      call SRECOD( IROT, ITSK, 'RMACR', RMACR, NPKIND*NTREG*NEMAC*2,
298:      &      IERR )
299:      call SRECOD( IROT, ITSK, 'XMAC', XMAC, (NGROUP
300:      &      +NPKIND)*NTREG*NEMAC*2, IERR )
301:      end if
302: C
303:      if ( JSCTM.ne.0 ) then
304: C
305: C ..... MICROSCOPIC REACTION RATE OF SCATTERING MATRIX
306: C
307:      KY = 0
308:      if ( JMICE(4).ne.0 ) KY = KY + 1
309:      if ( JMICE(6).ne.0 ) KY = KY + 1
310:      if ( JMICE(7).ne.0 ) KY = KY + 1
311:      NN = (NGP1+1)**2*NTREG*NUC*2*KY*JSCTM
312: c$$$
313:      if ( NN.gt.0 ) then
314:      call SRECOD( IROT, ITSK, 'SRMICS', SRMICS, NN, IERR )
315: c%c      call SRECOD( IROT, ITSK, 'XMICS', XMICS, NN, IERR )
316:      end if
317: C
318: C ..... MACROSCOPIC REACTION RATE OF SCATTERING MATRIX
319: C
320:      KYM = 0
321:      if ( JMACE(4).ne.0 ) KYM = KYM + 1
322:      if ( JMACE(6).ne.0 ) KYM = KYM + 1
323:      if ( JMACE(7).ne.0 ) KYM = KYM + 1
324:      NN = (NGP1+1)**2*NTREG*2*KYM*JSCTM
325:      if ( NN.gt.0 ) then

```

```

326:      call SRECOD( IROT, ITSK, 'RMACSM', RMACSM, NN, IERR )
327: c%c      call SRECOD( IROT, ITSK, 'XMACSM', XMACSM, NN, IERR )
328:      end if
329:      end if
330: C
331:      if ( JSCMU.ne.0 ) then
332: C
333: C ..... MICROSCOPIC SCATTERING MUBAR
334: C
335:      KY = 0
336:      if ( JMICE(4).ne.0 ) KY = KY + 1
337:      if ( JMICE(6).ne.0 ) KY = KY + 1
338:      if ( JMICE(7).ne.0 ) KY = KY + 1
339:      NN = (NGP1+1)*(NGP1+2)*NTREG*NUC*3*KY
340:      if ( NN.gt.0 ) then
341:      call SRECOD( IROT, ITSK, 'SMIMU', SMIMU, NN, IERR )
342:      end if
343: C
344: C ..... MACROSCOPIC SCATTERING MUBAR
345: C
346:      KYM = 0
347:      if ( JMACE(4).ne.0 ) KYM = KYM + 1
348:      if ( JMACE(6).ne.0 ) KYM = KYM + 1
349:      if ( JMACE(7).ne.0 ) KYM = KYM + 1
350:      NN = (NGP1+1)*(NGP1+2)*NTREG*3*KYM
351:      if ( NN.gt.0 ) then
352:      call SRECOD( IROT, ITSK, 'RMAMU', RMAMU, NN, IERR )
353:      end if
354:      end if
355: C
356: C ..... MONITOR OF CROSS SECTION TALLY
357: C
358:      if ( NEMIC.gt.0 ) then
359:      Cccc      write(IROT) WCXTY
360:      Cccc      write(IROT) DNFLX
361:      call SRECOD( IROT, ITSK, 'WCXTY', WCXTY, (NGROUP
362:      &      +NPKIND)*NTREG, IERR )
363:      call SRECOD( IROT, ITSK, 'DNFLX', DNFLX, NGROUP*NTREG*NUC, IERR
364:      &      )
365:      end if
366: C
367:      if ( JEIGN.ne.0 ) then
368: C
369: C .... EIGEN VALUE .....
370: C
371:      Cccc      write(IROT) (XSOC(N),N=1,NBATCH)
372:      Cccc      write(IROT) (XSXV(N,1),N=1,NBATCH)
373:      Cccc      write(IROT) (XSXV(N,2),N=1,NBATCH)
374:      Cccc      write(IROT) (XSXV(N,3),N=1,NBATCH)
375:      Cccc      write(IROT) ((XKEF(J,N),J=1,7),N=1,NBATCH)
376:      call SRECOD( IROT, ITSK, 'XSOC', XSOC, NBATCH, IERR )
377:      call SRECOD( IROT, ITSK, 'XSXV1', XSXV(1,1), NBATCH, IERR )
378:      call SRECOD( IROT, ITSK, 'XSXV2', XSXV(1,2), NBATCH, IERR )
379:      call SRECOD( IROT, ITSK, 'XSXV3', XSXV(1,3), NBATCH, IERR )
380: c##<2007/03/14:PN4:
381: c##      call SRECOD( IROT, ITSK, 'XKEF', XKEF, 7*NBATCH, IERR )
382:      call SRECOD( IROT, ITSK, 'XKEF', XKEF, 10*NBATCH, IERR )
383: c##>
384: C
385: C .... FISSION SOURCE BANK .....
386: C
387:      call SRECOD( IROT, ITSK, 'XXF', XXF, NFISB, IERR )
388:      call SRECOD( IROT, ITSK, 'YF', YF, NFISB, IERR )
389:      call SRECOD( IROT, ITSK, 'ZZF', ZZF, NFISB, IERR )
390:      call SRECOI( IROT, ITSK, 'IZZF', IZZF, NFISB, IERR )

```

src/mvp/resto.f

```

391:      call SRECOR( IROT, ITSK, 'EEEF', EEEF, NFISB, IERR )
392:      call SRECOI( IROT, ITSK, 'INUF', INUF, NFISB, IERR )
393:      call SRECOI( IROT, ITSK, 'IMTF', IMTF, NFISB, IERR )
394: C
395:      if ( NLATT.ne.0 ) then
396:        call SRECOI( IROT, ITSK, 'LEVL', LEVL, NFISB, IERR )
397:        do 100 J = 1, NEST
398:          call SRECOI( IROT, ITSK, 'LZZF', LZZF(1,J), NFISB, IERR )
399:          call SRECOI( IROT, ITSK, 'LPOSF', LPOSF(1,J), NFISB, IERR
400: &
401:          if ( JHLAT.ne.0 ) then
402:            call SRECOI( IROT, ITSK, 'LCRSF', LCRSF(1,J), NFISB,
403: & IERR )
404:          end if
405:        100 continue
406:      end if
407: C
408:      call SRECOI( IROT, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
409: C
410:      if ( JTLT.ne.0 ) then
411:        do 110 K = 1, NEST
412:          call SRECOI( IROT, ITSK, 'IBSPF', IBSPF(1,K), NFISB, IERR
413: &
414:        110 continue
415:      end if
416: C
417: C ... optional fission bank
418: C
419:      if ( KDFBK(0).gt.0 ) then
420:        do 130 K = 1, MDFBK
421:          if ( KDFBK(K).ne.0 ) then
422:            TAG = ' '
423:            write(TAG,(''DFBK'',I3)) K
424:            call CCOMP( TAG, LEN(TAG), TAG, IFL )
425:
426:            if ( K.eq.4 ) then
427:              ND = NEST + 1
428:            else if ( K.eq.5 ) then
429:              ND = 3*NEST
430:            else if ( K.eq.6 ) then
431:              ND = 6*NEST
432:            else
433:              ND = 1
434:            end if
435:
436:            do 120 J = 0, ND - 1
437:              call SRECOD( IROT, ITSK, TAG(:ICLEN2(TAG)),
438: & DFBK(1,KDFBK(K)+J), NFISB, IERR )
439:        120 continue
440:          end if
441:        130 continue
442:      end if
443:      if ( KIFBK(0).gt.0 ) then
444:        do 150 K = 1, MIFBK
445:          if ( KIFBK(K).ne.0 ) then
446:            TAG = ' '
447:            write(TAG,(''IFBK'',I3)) K
448:            call CCOMP( TAG, LEN(TAG), TAG, IFL )
449:
450:            if ( K.eq.6 ) then
451:              NI = NEST
452:            else
453:              NI = 1
454:            end if
455:

```

```

456:          do 140 J = 0, NI - 1
457:            call SRECOI( IROT, ITSK, TAG(:ICLEN2(TAG)),
458: & IFBK(1,KIFBK(K)+J), NFISB, IERR )
459:        140 continue
460:      end if
461:    150 continue
462:  end if
463: end if
464: C
465: C .... SPECIAL TALLIES ....
466: C
467:      if ( NETALY.gt.0 ) then
468: Cccc write(IROT) ETALY
469:        call SRECOD( IROT, ITSK, 'ETALY', ETALY, NLETAL*2, IERR )
470: C
471: C ... ETRV: added in file version 3.1 ...
472: C
473:      if ( NETRV.gt.0 ) then
474:        call SRECOD( IROT, ITSK, 'ETRV', ETRV,
475: & METRV*(NBATCH-NSKIP), IERR )
476:      end if
477:    end if
478: C
479: C .... PERTURBATION ....
480: C
481:      if ( JEIGN.ne.0 .and. JPert.ne.0 ) then
482:        call SRECOD( IROT, ITSK, 'WSD', WSD,
483: & NBANK*(NGSP+1)*NPTDS*NOPSDS, IERR )
484:        call SRECOD( IROT, ITSK, 'WSC', WSC,
485: & NBANK*(NGSP+1)*NPTCS, IERR )
486:        call SRECOD( IROT, ITSK, 'WDO', WDO,
487: & NFBNK0*(NGSP+1)*NPTDS*NOPSDS, IERR )
488:        call SRECOD( IROT, ITSK, 'WRO', WRO,
489: & NFBNK0*(NGSP+1)*NPTCS, IERR )
490:        call SRECOD( IROT, ITSK, 'SWD0', SWD0,
491: & (NGSP+1)*NPTDS*NOPSDS, IERR )
492:        call SRECOD( IROT, ITSK, 'SWR0', SWR0,
493: & (NGSP+1)*NPTCS, IERR )
494:        call SRECOD( IROT, ITSK, 'XKEFP', XKEFP,
495: & 7*NBATCH*NPT*NUCT, IERR )
496: c+beff2
497:      if ( NPTBE.ne.0 ) then
498:        call SRECOD( IROT, ITSK, 'WSB', WSB,
499: & NBANK*(NGSP+1), IERR )
500:        call SRECOD( IROT, ITSK, 'WB0', WB0,
501: & NFBNK0*(NGSP+1), IERR )
502:        call SRECOD( IROT, ITSK, 'SWB0', SWB0,
503: & (NGSP+1), IERR )
504:        call SRECOD( IROT, ITSK, 'WSBD', WSBD, NBANK*NGSP, IERR )
505:        call SRECOD( IROT, ITSK, 'WBDO', WBDO, NFBNK0*NGSP, IERR )
506:        call SRECOD( IROT, ITSK, 'SWBD0', SWBD0, NGSP, IERR )
507:        call SRECOD( IROT, ITSK, 'WSLD', WSLD, NBANK*NGSP, IERR )
508:        call SRECOD( IROT, ITSK, 'WLD0', WLD0, NFBNK0*NGSP, IERR )
509:        call SRECOD( IROT, ITSK, 'SWLD0', SWLD0, NGSP, IERR )
510:      end if
511: c-beff2
512:    end if
513: c+beff1
514: C
515: C .... BETA EFFECTIVE ....
516: C
517:      if ( JEIGN.ne.0 .and. JBEFF.ne.0 ) then
518:        call SRECOD( IROT, ITSK, 'XBEF', XBEF, NEBEF*NBATCH, IERR )
519:        call SRECOD( IROT, ITSK, 'WCBF', WCBF, NEBEF, IERR )
520:      end if

```

src/mvp/resto.f

```
521: c-beff1
522: C
523: C    .... rewind
524: C
525: Cccc  call RWIND(IROT)
526: C
527:      return
528:      end
```

SAFE

src/mvp/restot.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine RESTOT( IOPR, H, MONITR, ITSK,TITLE )
3: C=<MVP>=====
4: C PURPOSE: Interface routine to RESTO which outputs restart file record.
5: C CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C CALLS:
7: C
8: C-----
9: C
10: C arguments ( i = input, o = output, c =constant, w = work )
11: C
12: C i IOPR : message printout I/O unit
13: C io H : task local dynamic memory area
14: C i MONITR : do not print any message in RESTO if zero.
15: C i ITSK : data are stored on local data of task ITSK.
16: C          (ITSK may not same as current IDTASK)
17: C i TITLE : problem title
18: C-----
19:      real H(*)
20:      character TITLE(2)*72
21: C
22: CCCC include '../shared/INC/_ARRAY'
23:      include '../shared/INC/_TASKDT'
24:      include '../shared/INC/_LISTOFF'
25: C
26:      include 'INC/_KPIDS'
27:      include 'INC/_NGPS'
28: C
29:      include 'INC/_FLAGS'
30:      include '../shared/INC/_SIZES'
31:      include 'INC/_SIZES2'
32:      include 'INC/_CXSEC'
33:      include 'INC/_PXSEC'
34:      include 'INC/_XBANK'
35:      include 'INC/_SBANK'
36:      include 'INC/_STACK'
37:      include 'INC/_XWORK'
38:      include '../shared/INC/_PGEOM'
39:      include '../shared/INC/_PVRED'
40:      include 'INC/_PSOUR'
41:      include '../shared/INC/_PTALY0'
42:      include 'INC/_PTALY'
43:      include 'INC/_PTALY2'
44:      include '../shared/INC/_STALY'
45:      include '../shared/INC/_PTLSP'
46:      include '../shared/INC/_LISTON'
47:      include '../shared/INC/_WORDL'
48:      include 'INC/_PERT'
49:      include 'INC/_WKSOU'
50: C
51: C-----
52: C
53:      call RESTO( IOPR, MONITR, ITSK, NTHIST, IRAND, H(LWSUM), H(LWLEK),
54: &               NGROUP, NRESP, NREG, NTREG, NPKIND, NBATCH, NMEMO,
55: &               NEVENT, NZONE, H(LKMEMO), H(LWCNTR), H(LNCNTR),
56: &               H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
57: &               H(LXAVT), H(LAAVT),H(LEAVT),
58: &               NFISB, NFBANK,NFBNK0, NLENG, H(LXXXF), H(LYYYF),
59: &               H(LZZZF), H(LEEF), H(LIZZF), H(LINUF), H(LIBRGF),
60: &               H(LIBSPF), H(LXSOC), H(LXSXV), H(LXKEF), NLATT, NEST,
61: &               H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
62: &               H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK, MIFBK,
63: &               NUC, NEMIC,
64: &               NEMAC, H(LSRMIC), H(LRMICR), H(LXMIC), H(LRMAC),
65: &               H(LRMACR), H(LXMAC), H(LSRSTR), H(LSRSC), NSTAL,

```

```

66: &               H(LWCXTY), H(LDNFLX), NETALY, H(LETALY), NLETAL,
67: &               NETRV, NSKIP, METRV, H(LETRV),
68: &               NBANK, NPTDS, NPTCS, NGSP, NTPT, NUCPT,
69: &               H(LWSD), H(LWSC), H(LWDO), H(LWR0),
70: &               H(P1(23)), H(P1(24)), H(LXKEFP), NOPSDS,
71: &               H(LIMTF),
72: &               NGP(KPNEUT),H(LSRMICSM),H(LXMICSM),H(LRMACSM),
73: &               H(LXMACSM),
74: &               H(LSMIMU),H(LRMAMU),
75: c+beff1
76: &
77: c-beff1
78: c+beff2
79: &               H(LWSB), H(LWB0), H(P1(22)), NPTBE,
80: &               H(LWSBD), H(LWBD0), H(P1(21)),
81: &               H(LWSLD), H(LWLD0), H(P1(20)))
82: c-beff2
83: C
84:      return
85:      end

```

src/mvp/restrt.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTRT( IOPR, ITSK, NTHIST,IRAND, WSUM, WLEK,
3:   &                  NGROUP,NRESP, NREG,  NTREG, NPKIND,NBATCH,
4:   &                  NMEMO, NEVENT,NZONE, KMEMO, WCNTR, NCNTR,
5:   &                  SFLTR, SFLCL, SRETR, SRECL, XAVT,  AAVT,  EAVT,
6:   &                  NFISB, NFBANK,NFBNK0,NLENG, XXXF,  YYF,  ZZZF,
7:   &                  EEEF,  IZZF,  INUF,  IBRGF,  IBSPF,  XSOC,  XSXV,
8:   &                  XKEF,  NLATT, NEST,  LEVLF,  LZZF,  LPOSF,
9:   &                  LCRSF,  DFBK,  KDFBK,  MDFBK,  IFBK,  KIFBK,
10:  &                  MIFBK, NUC,   NEMIC, NEMAC, SRMIC, RMICR, XMIC,
11:  &                  RMAC,  RMACR, XMAC,  SRSTR, SRSCl, NSTAL,
12:  &                  WCXTY, DNFLX, NETALY, ETALY, NLETAL,NETRV,
13:  &                  NSKIP, METRV, ETRV,
14:  &                  NBANK, NPTDS, NPTCS, NGSP, NTPT, NUCPT,
15:  &                  WSD, WSC, WDO, WR0, SWD0, SWR0, XKEFP, NOPSDS,
16:  &                  IMTF,
17:  &                  NGP1,SRMICSM,XMICSM,RMACSM,XMACSM,
18:  &                  SMIMU, RMAMU,
19: c+beff1
20:   &                  NEBEF, WCBF, XBEF,
21: c-beff1
22: c+beff2
23:   &                  WSB, WB0, SWB0, NPTBE,
24:   &                  WSD, WBD0, SWBD0, WSLD, WLD0, SWLD0)
25: c-beff2
26: C=<MVP>=====
27: C  PURPOSE: Input restart information
28: C          (header - parameter must be input in RESTI0)
29: C
30: C  CALLED IN: RESTIN
31: C  CALLS: SRECI[I|R|D]
32: C
33: C  I/O UNIT OF RESTART INPUT : FT10F001 (IRIN)
34: C  I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
35: C
36: C=====
37:   include 'INC/_FLAGS'
38:   include '../shared/INC/_IOUNIT'
39:   include 'INC/_IOUNIT2'
40: C/#IF PARA(PVM)
41:   include '../shared/INC/_PVM PARA'
42: C/#ELSEIF PARA(MPI)
43: *   include 'mpif.h'
44: C/#ELSEIF PARA(VPP)
45:   include '../shared/INC/_VPP PARA'
46: C/#ENDIF
47:   include '../shared/INC/_TASKDT'
48:
49: C =<memo>= The seed value is saved as common variables
50: C          for a restart calculation. This treatment will be
51: C          modified in the future.
52:   include '../shared/INC/_RAND'
53: C
54:   real*8 WSUM, WLEK
55: C/#IF INTEGER8
56: *   integer*8 NTHIST
57: C/#ELSE
58:   integer NTHIST
59: C/#ENDIF
60: C
61: C  .... EVENT MONITOR.....
62: C
63:   real*8 WCNTR(NEVENT,NPKIND), NCNTR(NEVENT,NPKIND)
64:   real XAVT(3), AAVT(3), EAVT
65: C

```

```

66: C  .... NEXT-ZONE MEMORY .....
67: C
68:   integer KMEMO(NZONE,NMEMO)
69: C
70: C  .... TALLY ARRAYS .....
71: C
72:   real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
73:   &       SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
74: C
75:   real*8 SRMIC(NGROUP,NTREG,NUC,NEMIC,2),
76:   &       RMICR(NPKIND,NTREG,NUC,NEMIC,2),
77:   &       XMIC(NGROUP+NPKIND,NTREG,NUC,NEMIC,2),
78:   &       DNFLX(NGROUP,NREG,NUC)
79: C
80:   real*8 RMAC(NGROUP,NTREG,NEMAC,2), RMACR(NPKIND,NTREG,NEMAC,2),
81:   &       XMAC(NGROUP+NPKIND,NTREG,NEMAC,2)
82:   real*8 WCXTY(NGROUP+NPKIND,NTREG)
83:   real*8 SRSTR(NTREG,NSTAL,2), SRSCl(NTREG,NSTAL,2)
84:   real*8 ETALY(NLETAL,2)
85:   real*8 ETRV(METRV,*)
86: c%&   XMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,*),
87: c%&   XMACSM(NGP1+1,NGP1+1,NTREG,2,*),
88:   real*8 SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,JSCTM,*),
89:   &       RMACSM(NGP1+1,NGP1+1,NTREG,2,JSCTM,*)
90:   real*8 SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,*),
91:   &       RMAMU(NGP1+1,NGP1+2,NTREG,3,*)
92: C
93: C  ... EIGENVALUE TALLY .....
94: C
95: c##<2007/03/14:PN4:
96: c##   real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(7,NLENG)
97:   real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(10,NLENG)
98: c##>
99: C
100: C  ... FISSION PARTICLE (SOURCE) BANK ....
101: C
102:   real*8 XXXF(NFBNK0), YYF(NFBNK0), ZZZF(NFBNK0)
103:   integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
104:   &       LCRSF(NFBNK0,NEST), IZZF(NFBNK0), INUF(NFBNK0)
105:   integer IBRGF(NFBNK0), IBSPF(NFBNK0,0:NEST)
106:   real EEEF(NFBNK0)
107:   integer IMTF(NFBNK0)
108: C
109:   real*8 DFBK(NFBNK0,*)
110:   integer KDFBK(0:MDFBK)
111:   integer IFBK(NFBNK0,*)
112:   integer KIFBK(0:MIFBK)
113: C
114: C  ... PERTURBATION .....
115: C
116:   real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
117:   real*8 WSC(NBANK,0:NGSP,NPTCS)
118:   real*8 WDO(NFBNK0,0:NGSP,NPTDS,NOPSDS)
119:   real*8 WR0(NFBNK0,0:NGSP,NPTCS)
120:   real*8 SWD0(0:NGSP,NPTDS,NOPSDS)
121:   real*8 SWR0(0:NGSP,NPTCS)
122:   real*8 XKEFP(NUCPT,NTPT,7,NLENG)
123: c+beff1
124: C
125: C  ... BETA EFFECTIVE .....
126: C
127:   real*8 XBEF(NEBEF,NLENG), WCBF(NEBEF)
128: c-beff1
129: c+beff2
130:   real*8 WSB(NBANK,0:NGSP)

```

src/mvp/restrt.f

```

131:      real*8 WB0(NFBNK0,0:NGSP)
132:      real*8 SWB0(0:NGSP)
133:      real*8 WSD0(NBANK,NGSP)
134:      real*8 WBD0(NFBNK0,NGSP)
135:      real*8 SWBD0(NGSP)
136:      real*8 WSLD(NBANK,NGSP)
137:      real*8 WLD0(NFBNK0,NGSP)
138:      real*8 SWLD0(NGSP)
139: c-beff2
140: C
141: C ..... local data .....
142: C
143:      character*32 TAG
144:      integer IWRK(32)
145:      real RWRK(32)
146:      real*8 DWRK(32)
147:
148: C/IF INTEGER8
149: *      integer*8 I4MAX
150: C/ELSE
151:      integer I4MAX
152: C/ENDIF
153:      parameter( I4MAX = 2147483647 ) ! 2**31-1
154: C
155: C-----
156: C
157:      IERR = 0
158: C
159:      NPKI = 10
160:      call SRECII( IRIN, ITSK, 'TASKHEADER', IWRK, NPKI, IERR )
161: C
162:      if ( IERR.ne.0 ) then
163:          write(IMG,*) 'XXX(RESTRT) Cannot find an expected',
164:          &          ' task header of restart records. Task-ID: ', IDTASK,
165:          &          ' task-ID on which data are stored:', ITSK
166:      end if
167: C
168: C ... idtask, NTHIST, IRAND, NBATCH, NFISB .....
169: C
170:      if ( IDTASK.eq.ITSK ) then
171:          NTHIST = IWRK(1)
172:          NTHIST = IWRK(2)
173:          IRAND = IWRK(3)
174:          NBATCH = IWRK(4)
175:          NFISB = IWRK(5)
176:          IRNSU = IWRK(6)
177:          IRNSM = IWRK(7)
178:          IRNSL = IWRK(8)
179:          IQNTH = IWRK(9)
180:          IRNTH = IWRK(10)
181: C
182: C ... Restore NTHIST from 2 parts for more than (2**31-1) histories.
183: C This is the treatment not to change restart output format.
184: C It should be modified in the future.
185: C
186:          NTHIST = IQNTH*I4MAX + IRNTH
187:      end if
188: C
189:      NPKD = 3
190:      call SRECID( IRIN, ITSK, 'WSUM,WLEK', DWRK, NPKD, IERR )
191: C
192:      if ( IDTASK.eq.ITSK ) then
193:          WSUM = DWRK(1)
194:          WLEK = DWRK(2)
195:          RNCTR = DWRK(3)

```

```

196:      end if
197: C
198:      write(IOPR,*) ' === Task ', ITSK,
199:      &          ' input task wise restart data header '
200: C
201:      if ( IDTASK.eq.1.and.ITSK.eq.1 ) then
202:          write(IOPR,*) ' IDTASK marked on record: ', ITSK1
203:          write(IOPR,*) ' NTHIST : ', NTHIST
204:          write(IOPR,*) ' IRAND : ', IRAND
205:          write(IOPR,*) ' NBATCH : ', NBATCH
206:          write(IOPR,*) ' NFISB : ', NFISB
207:          write(IOPR,*) ' WSUM : ', WSUM
208:          write(IOPR,*) ' WLEK : ', WLEK
209:          write(IOPR,*) ' IRNSU : ', IRNSU
210:          write(IOPR,*) ' IRNSM : ', IRNSM
211:          write(IOPR,*) ' IRNSL : ', IRNSL
212:          write(IOPR,*) ' RNCTR : ', RNCTR
213:          write(IOPR,*) ' IQNTH : ', IQNTH
214:          write(IOPR,*) ' IRNTH : ', IRNTH
215:      else
216: C/IF PARA(PVM MPI)
217:          MSGNUM = 111
218:          IT0 = 1
219:          if ( IDTASK.ne.IT0 ) then
220:              call MVPCOM_SEND_I4( IT0, MSGNUM, IWRK, NPKI, INFO )
221:              call MVPCOM_SEND_R8( IT0, MSGNUM, DWRK, NPKD, INFO )
222:          else
223:              call MVPCOM_RECV_I4( ITSK, MSGNUM, IWRK, NPKI, INFO )
224:              call MVPCOM_RECV_R8( ITSK, MSGNUM, DWRK, NPKD, INFO )
225:          end if
226:          write(IOPR,*) ' IDTASK marked on record: ', IWRK(1)
227:          write(IOPR,*) ' NTHIST : ', IWRK(2)
228:          write(IOPR,*) ' IRAND : ', IWRK(3)
229:          write(IOPR,*) ' NBATCH : ', IWRK(4)
230:          write(IOPR,*) ' NFISB : ', IWRK(5)
231:          write(IOPR,*) ' WSUM : ', DWRK(1)
232:          write(IOPR,*) ' WLEK : ', DWRK(2)
233:          write(IOPR,*) ' IRNSU : ', IWRK(6)
234:          write(IOPR,*) ' IRNSM : ', IWRK(7)
235:          write(IOPR,*) ' IRNSL : ', IWRK(8)
236:          write(IOPR,*) ' RNCTR : ', DWRK(3)
237:          write(IOPR,*) ' IQNTH : ', IWRK(9)
238:          write(IOPR,*) ' IRNTH : ', IWRK(10)
239: C/ENDIF
240:      end if
241: C
242:      call SRECII( IRIN, ITSK, 'KMEMO', KMEMO, NZONE*NMEMO, IERR )
243:      call SRECID( IRIN, ITSK, 'WCNTR', WCNTR, NEVENT*NPKIND, IERR )
244:      call SRECID( IRIN, ITSK, 'NCNTR', NCNTR, NEVENT*NPKIND, IERR )
245: C
246:      call SRECIR( IRIN, ITSK, 'XAVT,AAVT,EAVT', RWRK, 7, IERR )
247:      if ( IDTASK.eq.ITSK ) then
248:          XAVT(1) = RWRK(1)
249:          XAVT(2) = RWRK(2)
250:          XAVT(3) = RWRK(3)
251:          AAVT(1) = RWRK(4)
252:          AAVT(2) = RWRK(5)
253:          AAVT(3) = RWRK(6)
254:          EAVT = RWRK(7)
255:      end if
256: C
257: C ..... FLUXES .....
258: C
259: C      read(IRIN) SFLTR
260: C      read(IRIN) SFLCL

```

src/mvp/restrt.f

```

261:
262:      call SRECID( IRIN, ITSK, 'SFLTR', SFLTR, NGROUP*NTREG*2, IERR )
263:      call SRECID( IRIN, ITSK, 'SFLCL', SFLCL, NGROUP*NTREG*2, IERR )
264: C
265: C ..... REACTION RATES .....
266: C
267:      if ( JRESP.ne.0 ) then
268:      if ( NRESP.gt.0 ) then
269: Cccc      read(IRIN) SRETR
270: Cccc      read(IRIN) SRECL
271:      call SRECID( IRIN, ITSK, 'SRETR', SRETR, NTREG*NRESP*2, IERR
272: &
273:      call SRECID( IRIN, ITSK, 'SRECL', SRECL, NTREG*NRESP*2, IERR
274: &
275:      end if
276:      if ( NSTAL.gt.0 ) then
277: Cccc      read(IRIN) SRSTR
278: Cccc      read(IRIN) SRSCL
279:      call SRECID( IRIN, ITSK, 'SRSTR', SRSTR, NGROUP*NSTAL*2,
280: &
281:      call SRECID( IRIN, ITSK, 'SRSCL', SRSCL, NGROUP*NSTAL*2,
282: &
283:      end if
284:      end if
285: C
286: C ..... MICROSCOPIC REACTION RATE .....
287: C
288:      if ( NEMIC.gt.0 ) then
289: Cccc      read(IRIN) SRMIC
290: Cccc      read(IRIN) RMICR
291: Cccc      read(IRIN) XMIC
292:      call SRECID( IRIN, ITSK, 'SRMIC', SRMIC, NGROUP*NTREG*NUC*NEMIC
293: &
294:      call SRECID( IRIN, ITSK, 'RMICR', RMICR, NPKIND*NTREG*NUC*NEMIC
295: &
296:      call SRECID( IRIN, ITSK, 'XMIC', XMIC, (NGROUP
297: &
298:      end if
299: C
300: C ..... MACROSCOPIC REACTION RATE .....
301: C
302:      if ( NEMAC.gt.0 ) then
303: Cccc      read(IRIN) RMAC
304: Cccc      read(IRIN) RMACR
305: Cccc      read(IRIN) XMAC
306:      call SRECID( IRIN, ITSK, 'RMAC', RMAC, NGROUP*NTREG*NEMAC*2,
307: &
308:      call SRECID( IRIN, ITSK, 'RMACR', RMACR, NPKIND*NTREG*NEMAC*2,
309: &
310:      call SRECID( IRIN, ITSK, 'XMAC', XMAC, (NGROUP
311: &
312:      end if
313: C
314:      if ( JSCTM.ne.0 ) then
315: C
316: C ..... MICROSCOPIC REACTION RATE OF SCATTERING MATRIX
317: C
318:      KY = 0
319:      if ( JMICE(4).ne.0 ) KY = KY + 1
320:      if ( JMICE(6).ne.0 ) KY = KY + 1
321:      if ( JMICE(7).ne.0 ) KY = KY + 1
322:      NN = (NGP1+1)**2*NTREG*NUC*2*KY*JSCTM
323:      if ( NN.gt.0 ) then
324:      call SRECID( IRIN, ITSK, 'SRMICS', SRMICS, NN, IERR )
325: c%c      call SRECID( IRIN, ITSK, 'XMICS', XMICS, NN, IERR )

```

```

326:      end if
327: C
328: C ..... MACROSCOPIC REACTION RATE OF SCATTERING MATRIX
329: C
330:      KYM = 0
331:      if ( JMACE(4).ne.0 ) KYM = KYM + 1
332:      if ( JMACE(6).ne.0 ) KYM = KYM + 1
333:      if ( JMACE(7).ne.0 ) KYM = KYM + 1
334:      NN = (NGP1+1)**2*NTREG*2*KYM*JSCTM
335:      if ( NN.gt.0 ) then
336:      call SRECID( IRIN, ITSK, 'RMACSM', RMACSM, NN, IERR )
337: c%c      call SRECID( IRIN, ITSK, 'XMACSM', XMACSM, NN, IERR )
338:      end if
339:      end if
340: C
341:      if ( JSCMU.ne.0 ) then
342: C
343: C ..... MICROSCOPIC SCATTERING MUBAR
344: C
345:      KY = 0
346:      if ( JMICE(4).ne.0 ) KY = KY + 1
347:      if ( JMICE(6).ne.0 ) KY = KY + 1
348:      if ( JMICE(7).ne.0 ) KY = KY + 1
349:      NN = (NGP1+1)*(NGP1+2)*NTREG*NUC*3*KY
350:      if ( NN.gt.0 ) then
351:      call SRECID( IRIN, ITSK, 'SMIMU', SMIMU, NN, IERR )
352:      end if
353: C
354: C ..... MACROSCOPIC SCATTERING MUBAR
355: C
356:      KYM = 0
357:      if ( JMACE(4).ne.0 ) KYM = KYM + 1
358:      if ( JMACE(6).ne.0 ) KYM = KYM + 1
359:      if ( JMACE(7).ne.0 ) KYM = KYM + 1
360:      NN = (NGP1+1)*(NGP1+2)*NTREG*3*KYM
361:      if ( NN.gt.0 ) then
362:      call SRECID( IRIN, ITSK, 'RMAMU', RMAMU, NN, IERR )
363:      end if
364:      end if
365: C
366: C ..... MONITOR OF CROSS SECTION TALLY
367: C
368:      if ( NEMIC.gt.0 ) then
369: Cccc      read(IRIN) WCXTY
370: Cccc      read(IRIN) DNFLX
371:      call SRECID( IRIN, ITSK, 'WCXTY', WCXTY, (NGROUP
372: &
373:      call SRECID( IRIN, ITSK, 'DNFLX', DNFLX, NGROUP*NTREG*NUC, IERR
374: &
375:      end if
376: C
377:      if ( JEIGN.ne.0 ) then
378: C
379: C .... EIGEN VALUE .....
380: C
381: Cccc      read(IRIN) (XSOC(N),N=1,NBATCH)
382: Cccc      read(IRIN) (XSXV(N,1),N=1,NBATCH)
383: Cccc      read(IRIN) (XSXV(N,2),N=1,NBATCH)
384: Cccc      read(IRIN) (XSXV(N,3),N=1,NBATCH)
385: Cccc      read(IRIN) ((XKEF(J,N),J=1,7),N=1,NBATCH)
386:      call SRECID( IRIN, ITSK, 'XSOC', XSOC, NBATCH, IERR )
387:      call SRECID( IRIN, ITSK, 'XSXV1', XSXV(1,1), NBATCH, IERR )
388:      call SRECID( IRIN, ITSK, 'XSXV2', XSXV(1,2), NBATCH, IERR )
389:      call SRECID( IRIN, ITSK, 'XSXV3', XSXV(1,3), NBATCH, IERR )
390: c##<2007/03/14:PN4:

```

src/mvp/restrt.f

```

391: c##      call SRECID( IRIN, ITSK, 'XKEF', XKEF, 7*NBATC, IERR )
392:      call SRECID( IRIN, ITSK, 'XKEF', XKEF, 10*NBATC, IERR )
393: c##>
394: C
395: C      .... FISSION SOURCE BANK .....
396: C
397:       if ( NFBANK.lt.NFISB ) then
398:         write(IMG,*) '!!! LENGTH OF FISSION BANK IS LESS THAN ',
399:           &          'THAT IN RESTART FILE (NFBANK=', NFBANK, ',NFISB=',
400:           &          NFISB
401:         call CNERR( 'WARNING' )
402:         NFISB = NFBANK
403:       end if
404: C
405:       call SRECID( IRIN, ITSK, 'XXF', XXF, NFISB, IERR )
406:       call SRECID( IRIN, ITSK, 'YYF', YYF, NFISB, IERR )
407:       call SRECID( IRIN, ITSK, 'ZZF', ZZF, NFISB, IERR )
408:       call SRECII( IRIN, ITSK, 'LZZF', LZZF, NFISB, IERR )
409:       call SRECIR( IRIN, ITSK, 'EEEF', EEEF, NFISB, IERR )
410:       call SRECII( IRIN, ITSK, 'INUF', INUF, NFISB, IERR )
411:       call SRECII( IRIN, ITSK, 'IMTF', IMTF, NFISB, IERR )
412: C
413:       if ( NLATT.ne.0 ) then
414:         call SRECII( IRIN, ITSK, 'LEVLF', LEVLF, NFISB, IERR )
415:         do 100 J = 1, NEST
416:           call SRECII( IRIN, ITSK, 'LZZF', LZZF(1,J), NFISB, IERR )
417:           call SRECII( IRIN, ITSK, 'LPOSF', LPOSF(1,J), NFISB, IERR )
418:         &
419:           if ( JHLAT.ne.0 ) then
420:             call SRECII( IRIN, ITSK, 'LCRSF', LCRSF(1,J), NFISB,
421:             &          IERR )
422:           end if
423:         100 continue
424:       end if
425: C
426:       call SRECII( IRIN, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
427: C
428:       if ( JTLT.ne.0 ) then
429:         do 110 K = 0, NEST
430:           call SRECII( IRIN, ITSK, 'IBSPF', IBSPF(1,K), NFISB, IERR )
431:         &
432:         110 continue
433:       end if
434: C
435: C      ... optional fission bank
436: C
437:       if ( KDFBK(0).gt.0 ) then
438:         do 130 K = 1, MDFBK
439:           if ( KDFBK(K).ne.0 ) then
440:             TAG = ' '
441:             write(TAG,('DFBK',I3)) K
442:             call CCOMP( TAG, LEN(TAG), TAG, IFL )
443:
444:             if ( K.eq.4 ) then
445:               ND = NEST + 1
446:             else if ( K.eq.5 ) then
447:               ND = 3*NEST
448:             else if ( K.eq.6 ) then
449:               ND = 6*NEST
450:             else
451:               ND = 1
452:             end if
453:
454:             do 120 J = 0, ND - 1
455:               call SRECID( IRIN, ITSK, TAG(:ICLEN2(TAG)),
456:               &          DFBK(1,KDFBK(K)+J), NFISB, IERR )
457:             120 continue
458:           end if
459:         130 continue
460:       end if
461:       if ( KIFBK(0).gt.0 ) then
462:         do 150 K = 1, MIFBK
463:           if ( KIFBK(K).ne.0 ) then
464:             TAG = ' '
465:             write(TAG,('IFBK',I3)) K
466:             call CCOMP( TAG, LEN(TAG), TAG, IFL )
467:
468:             if ( K.eq.6 ) then
469:               NI = NEST
470:             else
471:               NI = 1
472:             end if
473:
474:             do 140 J = 0, NI - 1
475:               call SRECII( IRIN, ITSK, TAG(:ICLEN2(TAG)),
476:               &          IFBK(1,KIFBK(K)+J), NFISB, IERR )
477:             140 continue
478:           end if
479:         150 continue
480:       end if
481:
482: C      .... SPECIAL TALLIES .....
483: C
484: C      if ( NETALY.gt.0 ) then
485: Ccccc read(IRIN) ETALY
486:       call SRECID( IRIN, ITSK, 'ETALY', ETALY, NLETAL*2, IERR )
487: C
488: C      ... ETRV: added in file version 3.1 ...
489: C
490: C      if ( NETRV.gt.0 ) then
491: C        call SRECID( IRIN, ITSK, 'ETRV', ETRV, METRV*(NBATCH
492: C          &          -NSKIP), IERR )
493: C      end if
494: C    end if
495:
496: C
497: C      .... PERTURBATION .....
498: C
499: C      if ( JEIGN.ne.0 .and. JPRT.ne.0 ) then
500: C        call SRECID( IRIN, ITSK, 'WSD', WSD,
501: C          &          NBANK*(NGSP+1)*NPTDS*NOPSDS, IERR )
502: C        call SRECID( IRIN, ITSK, 'WSC', WSC,
503: C          &          NBANK*(NGSP+1)*NPTCS, IERR )
504: C        call SRECID( IRIN, ITSK, 'WDO', WDO,
505: C          &          NFBNK0*(NGSP+1)*NPTDS*NOPSDS, IERR )
506: C        call SRECID( IRIN, ITSK, 'WRO', WRO,
507: C          &          NFBNK0*(NGSP+1)*NPTCS, IERR )
508: C        call SRECID( IRIN, ITSK, 'SWD0', SWD0,
509: C          &          (NGSP+1)*NPTDS*NOPSDS, IERR )
510: C        call SRECID( IRIN, ITSK, 'SWR0', SWR0,
511: C          &          (NGSP+1)*NPTCS, IERR )
512: C        call SRECID( IRIN, ITSK, 'XKEFP', XKEFP,
513: C          &          7*NBATC*NPT*NUCPT, IERR )
514: C      c+beff2
515: C      if ( NPTBE.gt.0 ) then
516: C        call SRECID( IRIN, ITSK, 'WSB', WSB,
517: C          &          NBANK*(NGSP+1), IERR )
518: C        call SRECID( IRIN, ITSK, 'WBO', WBO,
519: C          &          NFBNK0*(NGSP+1), IERR )
520: C

```

src/mvp/restrt.f

```
521:          call SRECID( IRIN, ITSK, 'SWB0', SWB0,
522:            &          (NGSP+1), IERR )
523:          call SRECID(IRIN, ITSK, 'WSBD', WSBD, NBANK*NGSP, IERR)
524:          call SRECID(IRIN, ITSK, 'WBD0', WBD0, NFBNK0*NGSP, IERR)
525:          call SRECID(IRIN, ITSK, 'SWBD0', SWBD0, NGSP, IERR)
526:          call SRECID(IRIN, ITSK, 'WSLD', WSLD, NBANK*NGSP, IERR)
527:          call SRECID(IRIN, ITSK, 'WLD0', WLD0, NFBNK0*NGSP, IERR)
528:          call SRECID(IRIN, ITSK, 'SWLD0', SWLD0, NGSP, IERR)
529:        end if
530: c-beff2
531:    end if
532: c+beff1
533: C
534: C      .... BETA EFFECTIVE ....
535: C
536:    if ( JE1EN.ne.0 .and. JBEFF.ne.0 ) then
537:      call SRECID(IRIN, ITSK, 'XBEF', XBEF, NEBEF*NBATCH, IERR)
538:      call SRECID(IRIN, ITSK, 'WCBEF', WCBEF, NEBEF, IERR)
539:    end if
540: c-beff1
541:    return
542:  end
```

src/mvp/seaone.f

```

1: C/#IF ARGSAVE
2: * subroutine SEASON0(IOW, A, CHA,
3: * N IDEFER,NLOST,NDEAD,NEVENT,DEPS,NSMAC,NSMIC,MB,NUC,NMKREG,
4: * B XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
5: * B WWW ,KKP ,IZZ ,IGG ,TTT,ITT, KLSF ,EEE ,XIM ,
6: * B SMAC ,SMIC ,SGTAL ,KMAC ,MMAC ,KSPI ,
7: * B LEVL ,LZZ ,LPOS ,LCRS ,IBREG ,IBSPC ,
8: * B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
9: * S LSFFL ,NFFL ,IZFFL ,LSSRC ,NNXT ,IZNXT ,
10: * S LSDED ,LSREF ,ISREF ,IZREF ,NBREF ,KSREF ,
11: * * LSLAT ,IZLAT ,NXLT ,
12: * G SDA ,KZMAT ,KZREG ,KZDA ,KZAA ,ksfbd ,
13: * * KCELL ,IPCEL ,MLBZZ ,ISUSP ,MKREG ,
14: * T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD ,WKILD ,NSURV ,
15: * T WSURV ,NSPLT ,WSPLT ,NCNTR ,WCNTR ,
16: * T JDTRG ,IDTAL ,JTEVE ,LTEVE ,KTEVE ,NTEVE ,DTALY ,RESP ,
17: * W X ,Y ,Z ,W ,IBP ,IFI ,
18: * W IZI ,IWK ,IFL ,R ,ISRF ,FXYZ ,
19: * & NUCPN ,NSMICPN ,SMICPN ,KPNFG ) ! photonuc
20: C/#ELSE
21: subroutine SEAONE(IOW, A, CHA,
22: N IDEFER,NLOST,NDEAD,NEVENT,DEPS,NSMAC,NSMIC,MB,NUC,NMKREG,
23: B XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
24: B WWW ,KKP ,IZZ ,IGG ,TTT,ITT, KLSF ,EEE ,XIM ,
25: B SMAC ,SMIC ,SGTAL ,KMAC ,MMAC ,KSPI ,
26: B LEVL ,LZZ ,LPOS ,LCRS ,IBREG ,IBSPC ,
27: B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
28: S LSFFL ,NFFL ,IZFFL ,LSSRC ,NNXT ,IZNXT ,
29: S LSDED ,LSREF ,ISREF ,IZREF ,NBREF ,KSREF ,
30: * LSLAT ,IZLAT ,NXLT ,
31: G SDA ,KZMAT ,KZREG ,KZDA ,KZAA ,ksfbd ,
32: * KCELL ,IPCEL ,MLBZZ ,ISUSP ,MKREG ,
33: T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD ,WKILD ,NSURV ,
34: T WSURV ,NSPLT ,WSPLT ,NCNTR ,WCNTR ,
35: T JDTRG ,IDTAL ,JTEVE ,LTEVE ,KTEVE ,NTEVE ,DTALY ,RESP ,
36: W X ,Y ,Z ,W ,IBP ,IFI ,
37: W IZI ,IWK ,IFL ,R ,ISRF ,FXYZ ,
38: % MZONE ,JMEM ,KMEMO ,MEMC,MEMZ ,
39: & NUCPN ,NSMICPN ,SMICPN ,KPNFG , ! photonuc
40: & SMICP ,SMACP ,NPTDS ,NPTCS ,NGSP ,WWD ,WWD2 ,WWR ,WSD , ! pert
41: & WSC ,NORDD ,NOPS , ! pert
42: & WWB ,WSB ,NPTBE ,WWBD ,WSBD ,WWLD ,WSLD , ! beff
43: & DWK1 ,IOTL) ! time-list
44: C/#ENDIF
45:
46: C=<MVP>=====
47: C PURPOSE: FIND ZONES TO ENTER. (ONE ZONE LOGIC)
48: C CALLED IN: ACTION
49: C CALLS: JUDGE,VMNTRL,SPLITB
50: C=====
51: C
52: C === INTER-STACK DATA FLOW ===
53: C
54: C NEXT-ZONE SEARCH STACK----+----> FREE-FLIGHT STACK
55: C (LSSRC,IZNXT,NNXT) | | (LSFFL,IZFFL,NFFL)
56: C DEAD PARTICLE STACK ----+ ----> LATTICE-SEARCH STACK
57: C (LSDED,NDEAD) | | (LSLAT,IZLAT,NXLT)
58: C | | ----> REFLECTION STACK
59: C | | (LSREF,ISREF,IZREF,NBREF)
60: C +----> DEAD-PARTICLE STACK
61: C (LSDED,NDEAD)
62: C
63: C === BANK DATA TO BE UPDATED ===
64: C
65: C IZZ : ZONE #
66: C WWW : WEIGHT (SPLITTING OR RUSSIAN ROULETTE)

```

```

66: C
67: C === BANK DATA ADDED NEWLY (SPLITTING) ===
68: C
69: C (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #
70: C IGG, WWW : ENERGY GROUP & WEIGHT
71: C LEVL,LZZ,LPOS,LCRS : LATTICE-PARAMETER
72: C IBREG, IBSPC
73: C=====
74: C implicit real*8(D)
75: C
76: C real A(*)
77: C character*4 CHA(*)
78: C
79: C include 'INC/_KPIDS'
80: C include 'INC/_NGPS'
81: C
82: C include '../shared/INC/_SIZES'
83: C
84: C include 'INC/_FLAGS'
85: C include '../shared/INC/_TASKDT'
86: C
87: C ..... TALLY .....
88: C
89: C integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
90: C integer IDTAL(*)
91: C real*8 DTALY(*)
92: C real RESP(*)
93: C
94: C ..... BANK .....
95: C
96: C real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
97: C & CCC(NBANK)
98: C real WWW(NBANK), XIM(NBANK)
99: C real*8 TTT(NBANK)
100: C integer ITT(NBANK)
101: C integer KKP(NBANK)
102: C integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
103: C & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
104: C integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
105: C
106: C real*8 DBNK(NBANK,*)
107: C integer KDBNK(0:MDBNK)
108: C integer IBNK(NBANK,*)
109: C integer KIBNK(0:MIBNK)
110: C integer KPNFG(NBANK) ! photonuc
111: C
112: C ... SIGMA BANK .....
113: C
114: C real EEE(NBANK), SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
115: C & SGTAL(NBANK,NSTAL)
116: C real SMICPN(NBANK,NUCPN,NSMICPN) ! photonuc
117: C integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC)
118: C
119: C ... variables for perturbation calculation ...
120: C
121: C real*8 WWD(NBANK,NPTDS,NORDD), WWD2(NBANK,NPTDS)
122: C real*8 WSD(NBANK,0:NGSP,NPTDS,NOPS)
123: C real*8 WWR(NBANK,NPTCS)
124: C real*8 WSC(NBANK,0:NGSP,NPTCS)
125: C real SMICP(NBANK,NUC,NSMIC), SMACP(NBANK,MB,NSMAC)
126: C real*8 WWB(NBANK)
127: C real*8 WSB(NBANK,0:NGSP)
128: C real*8 WWBD(NBANK)
129: C real*8 WSBD(NBANK,NGSP)
130: C real*8 WWLD(NBANK)

```

src/mvp/seaone.f

```

131:      real*8 WSLD(NBANK,NGSP)
132: C
133: C ..... STACK .....
134: C
135:      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSSRC(NBANK),
136:      &      NNXT(NZONE+1), IZNXT(NBANK), LSDDED(NBANK), LSREF(NBANK),
137:      &      ISREF(NBANK), IZREF(NBANK), NBRF(NREFS+1), LSLAT(NBANK),
138:      &      IZLAT(NBANK), NXLT(*)
139: C
140: C ..... GEOMETRY .....
141: C
142:      real*8 SDA(NSDA)
143:      integer KZMAT(NZONE), KZREG(NZONE), KSREF(NZDA), KZDA(2,NZDA),
144:      &      KZAA(NZONE+1), KMEMO(NZONE,NMEMO), MEMC(NZONE,NMEMO),
145:      &      MEMZ(NZONE), KCELL(NZONE), IPCEL(*), MLBZZ(NZONE)
146:      integer KSFBD(NZDA)
147: C
148:      integer ISUSP(NSUZON,NSPACE)
149: C
150: C ..... VARIANCE REDUCTION .....
151: C
152:      real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
153:      real WTIME(NTIME)
154:      real WLLIM
155: C
156: C ..... TALLY (MONITOR) .....
157: C
158:      integer MKREG(NREG,2)
159: C
160:      real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG)
161:      real*8 NSPLT(NGROUP,NREG)
162:      real*8 WKILD(NGROUP,NREG), WSRV(NGROUP,NREG)
163:      real*8 WSPLT(NGROUP,NREG)
164:      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
165: C
166: C ..... WORK AREA .....
167: C
168:      real*8 X(NBANK), Y(NBANK), Z(NBANK), W(NBANK)
169:      real R(NBANK), FXYZ(NBANK)
170:      integer IBP(NBANK), IZI(NBANK), IWK(NBANK), ISRF(NBANK)
171:      logical IFI(NBANK), IFL(NBANK), JMEM
172:      real*8 DWK1(NBANK)
173: C
174:      include '../shared/INC/_COUNTS'
175: C
176: C ..... local variable ...
177: C
178:      parameter( MWARN      = 10 )
179:      logical JRR
180: C
181:      integer KSORT(KPLIM), IKSORT(KPLIM+1)
182: C
183:      include '../shared/INC/_PMLATT'
184:      include '../shared/INC/_SFLATT'
185: C
186: C-----
187: C
188: C/#IF ARGSAVE
189: *      return
190: C
191: C -----
192: C
193: *      entry      SEAONE( MZONE ,JMEM ,KMEMO ,MEMC,MEMZ)
194: C/#ENDIF
195: C

```

```

196: C-----
197: C
198: CM1999-8-31
199:      NDEAD0 = NDEAD
200: C
201:      JVARR = JIMPT + JWWND + JRRLT
202: C
203: C      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
204: C          IPAR = 0
205: C      else if ( JNEUT.ne.0 ) then
206: C          IPAR = 1
207: C      else
208: C          IPAR = 2
209: C      end if
210: C
211:      IPAR = 0
212:      if ( NPKIND.eq.1 ) then
213:          do 100 IK = 1, KPLIM
214:              if ( JKPARIK).ne.0 ) then
215:                  IPAR = IK
216:                  go to 110
217:              end if
218:          100 continue
219:          110 continue
220:      end if
221: C
222: C
223: C .... MZONE: > 0 : SEARCH FOR ZONE MZONE, < 0 : DEFERRED PARTICLES
224: C
225:      if ( MZONE.lt.0 ) go to 440
226: C
227:      if ( JVMNT.ne.0 ) call VMNTRI( 4, NNXT(MZONE) )
228: C
229: C-----
230: C .... GATHER VECTORS .....
231: C-----
232: C
233:      II = 0
234:      do 120 I = 1, NNXT(NZONE+1)
235:          if ( IZNXT(I).eq.MZONE ) then
236:              II = II + 1
237:              IBP(II) = LSSRC(I)
238:          end if
239:      120 continue
240:      do 130 I = 1, NNXT(MZONE)
241:          X(I) = XXX(IBP(I)) + DEPS*AAA(IBP(I))
242:          Y(I) = YYY(IBP(I)) + DEPS*BBB(IBP(I))
243:          Z(I) = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
244:          W(I) = WWW(IBP(I))
245:      130 continue
246: C
247: C-----
248: C .... SEARCH ZONE # KKZ+1 TO MMZ.
249: C-----
250: C
251:      if ( JLATT.ne.0 ) then
252:          ICEL = KCELL(MZONE)
253:          MMZ = IPCEL(ICEL+1) - 1
254:          if ( ICEL.gt.0 ) then
255:              KKZ = IPCEL(ICEL) - 1
256:          else
257:              KKZ = 0
258:          end if
259:      else
260:          ICEL = 0

```


src/mvp/seaone.f

```

261:      KKZ      = 0
262:      MMZ      = NZONE
263:      end if
264:      MMX      = MIN(MMZ-KKZ,NMEMO)
265: C
266: C ..... IMEMO = 1 MEANS THAT ALL ZONES IN KMEMO ARRAY HAS NOT BEEN
267: C SEARCHED.
268: C
269:      IMEMO     = 1
270: C
271: C-----
272: C ..... BEGINS NEXT-ZONE-SEARCH .....
273: C-----
274: C
275:      INX      = NNXT(MZONE)
276:      INFL     = NFFL(NZONE+1)
277:      IREF     = 0
278:      ILAT     = 0
279:      if ( JREFL.ne.0 ) IREF = NBREF(NREFS+1)
280:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
281: C
282:      do 340 M = 1, MMX
283: C
284: C
285: C .... RETURN HERE IF KMEMO(MZONE,M)=0 AND THERE ARE NO PARTICLES WHICH
286: C BELONGS TO ZONE 'KZ' DETERMINE NEXT 'KZ' WITHOUT INCREMENTING
287: C LOOP COUNTER 'M'.
288: C
289: C
290:      140      if ( INX.eq.0 ) go to 350
291: C
292: C-----
293: C .... DETERMINE ZONE TO BE CHECKED (KZ) .....
294: C-----
295: C
296:      if ( KMEMO(MZONE,M).ne.0 ) then
297:      KZ      = KMEMO(MZONE,M)
298:      else
299:      IMEMO    = 0
300:      150      KKZ      = KKZ + 1
301:      if ( KKZ.eq.MZONE ) go to 150
302:      do 160 MM = 1, MMX
303:      if ( KKZ.eq.KMEMO(MZONE,MM) ) go to 150
304:      160      continue
305:      KZ      = KKZ
306:      if ( KZ.gt.MMZ ) go to 350
307:      end if
308:      MAT     = KZMAT(KZ)
309: Ccccccc IF( JREFL .NE. 0 ) JRR = MAT.LE.-1001
310: CCCC      JRR      = JREFL.ne.0.and.MAT.le. - 1001
311:      JRR      = JREFL.ne.0.and.MAT.le. - 1001.and..not.ISLATT(MAT)
312: C
313: C
314: C-----
315: C .... CHECK FOR EACH PARTICLE. IFI(I) = .FALSE./.TRUE. = OUT/IN
316: C-----
317: C
318: C
319:      call JUDGE( 'SEAONE', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
320:      &          NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, JRR, KSREF,
321:      &          ISRF, FXYZ )
322: C
323: C-----
324: C .... COMPRESS PARTICLE DATA & UPDATE FLIGHT STACK .....
325: C-----

```

```

326: C
327:      KK      = 0
328:      do 170 I = 1, INX
329:      if ( IFI(I) ) KK      = KK + 1
330:      170      continue
331: C
332:      if ( KK.ne.0 ) then
333:      II      = 0
334:      IFFL    = 0
335:      IDEAD   = NDEAD
336: C
337: C
338: C-----
339: C ..... SEND PARTICLES TO FLIGHT STACK .....
340: C-----
341: C
342: C
343:      if ( MAT.ge.0 ) then
344:      *VOCL LOOP,NOVREC
345:      do 180 I = 1, INX
346:      if ( IFI(I) ) then
347:      IFFL    = IFFL + 1
348:      LSFFL(INFL+IFFL) = IBP(I)
349:      end if
350:      180      continue
351: C
352: C ... check marker region ...
353: C
354:      if ( NMKREG.ne.0 ) then
355:      IRRG    = KZREG(MZONE)
356:      IRRG2   = KZREG(KZ)
357:      if ( JTLLT.eq.0.and.MKREG(IRRG,1).ne.0
358:      &      .and.IRRG2.ne.IRRG ) then
359:      CC      &      KG      = KIBNK(9) + MKREG(IRRG,1) - 1
360:      CC*VOCL LOOP,NOVREC
361:      CC
362:      do 170 I = 1, IFFL
363:      CC      IP      = LSFFL(INFL+I)
364:      CC      IBNK(IP,KG) = IBNK(IP,KG) + 1
365:      CC      continue
366:      CC*VOCL LOOP,NOVREC
367:      CC
368:      do 180 I = 1, IFFL
369:      CC      IP      = LSFFL(INFL+I)
370:      CC      IRRG    = IBREG(IP)
371:      CC
372:      ILV      = LEVL(IP)
373:      IRRG2    = KZREG(KZ)
374:      if ( ILV.gt.0 ) then
375:      IRRG2    = ISUSP(KZREG(KZ),IBSPC(IP,ILV))
376:      end if
377:      if ( MKREG(IRRG,1).ne.0.and.IRRG.ne.IRRG2 ) then
378:      CC      KG      = KIBNK(9) + MKREG(IRRG,1) - 1
379:      CC      IBNK(IP,KG) = IBNK(IP,KG) + 1
380:      CC
381:      Check %%%%%%%%%
382:      C      write(*,*) ' IRRG IRRG2 KG : ',IRRG,IRRG2, KG
383:      C %%%%%%%%%
384:      CC      end if
385:      CC      continue
386:      CC
387:      C
388:      .... marker region procedure is modified in Jan 2000,
389:      C      to treat marker-region in lattice geometry.
390:      C
391:      ... check marker region : set previous region # in IWK ...

```

src/mvp/seaone.f

```

391: C
392:       if ( NMKREG.ne.0 ) then
393: *VOCL LOOP,NOVREC
394:         do 190 I = 1, IFFL
395:           IP = LSFFL(INFL+I)
396:           IWK(I) = IBREG(IP)
397: C
398: C       ... this is the first flight after birth as source
399: C       particle, so particle has no "previous region".
400: C
401:       if ( IBNK(IP,KIBNK(5)).lt.0 ) then
402:         IWK(I) = 0
403:       end if
404: 190      continue
405:       end if
406: C
407: C
408:       if ( JTLLT.ne.0 ) then
409: *VOCL LOOP,NOVREC
410:         do 200 I = 1, IFFL
411:           IP = LSFFL(INFL+I)
412:           ILV = LEVL(IP)
413:           if ( ILV.gt.0 ) then
414:             IBREG(IP) = ISUSP(KZREG(KZ),IBSPC(IP,ILV))
415:           else
416:             IBREG(IP) = KZREG(KZ)
417:           end if
418: 200      continue
419:       else
420: C
421: C       ... IBREG is set anytime (from Jan 2000)
422: *VOCL LOOP,NOVREC
423:         do 210 I = 1, IFFL
424:           IP = LSFFL(INFL+I)
425:           IBREG(IP) = KZREG(KZ)
426: 210      continue
427:         end if
428: C
429: C       ... check marker region escape : IWK is previous value of IBREG ...
430: C
431:       if ( NMKREG.ne.0 ) then
432: *VOCL LOOP,NOVREC
433:         do 220 I = 1, IFFL
434:           IP = LSFFL(INFL+I)
435:           IRRG = IWK(I)
436:           IRRG2 = IBREG(IP)
437:           if ( IRRG.gt.0.and.MKREG(IRRG,1).ne.0
438:             & .and.IRRG.ne.IRRG2 ) then
439:             KG = KIBNK(9) + MKREG(IRRG,1) - 1
440:             IBNK(IP,KG) = IBNK(IP,KG) + 1
441:           end if
442: 220      continue
443:         end if
444: C
445:         do 230 I = 1, IFFL
446:           IZFFL(INFL+I) = KZ
447: 230      continue
448: C
449:       if ( IPAR.eq.0 ) then
450: *VOCL LOOP, SCALAR
451:         do 240 I = 1, IFFL
452:           KP = KKP(LSFFL(INFL+I))
453:           NCNTR(17,KP) = NCNTR(17,KP) + 1
454: 240      continue
455:         end if

```

```

456: C
457: C       ... set region# and/or zone# on birth here !!!
458: C
459:       if ( JLATT.ne.0.and.KIBNK(5).ne.0
460:         & .and.(KIBNK(2).ne.0.or.KIBNK(3).ne.0) ) then
461:         NMNM = 0
462: *VOCL LOOP,NOVREC
463:         do 250 I = 1, IFFL
464:           IP = LSFFL(INFL+I)
465:           if ( IBNK(IP,KIBNK(5)).lt.0 ) then
466:             NMNM = NMNM + 1
467:           end if
468: 250      continue
469: C
470:       if ( NMNM.gt.0.and.KIBNK(2).ne.0 ) then
471: *VOCL LOOP,NOVREC
472:         do 260 I = 1, IFFL
473:           IP = LSFFL(INFL+I)
474:           if ( IBNK(IP,KIBNK(5)).lt.0 ) then
475:             if ( JTLLT.ne.0 ) then
476:               IBNK(IP,KIBNK(2)) = IBREG(IP)
477:             else
478:               IBNK(IP,KIBNK(2)) = KZREG(KZ)
479:             end if
480:           end if
481: 260      continue
482:         end if
483: C
484:       if ( NMNM.gt.0.and.KIBNK(3).ne.0 ) then
485: *VOCL LOOP,NOVREC
486:         do 270 I = 1, IFFL
487:           IP = LSFFL(INFL+I)
488:           if ( IBNK(IP,KIBNK(5)).lt.0 ) then
489:             IBNK(IP,KIBNK(3)) = KZ
490:           end if
491: 270      continue
492:         end if
493:       end if
494: C
495:       NFFL(KZ) = NFFL(KZ) + IFFL
496:       INFL = INFL + IFFL
497: C
498: C
499: C-----
500: C       ..... LEAKAGE .....
501: C-----
502: C
503: C
504:       else if ( MAT.eq.-1000 ) then
505: *VOCL LOOP,NOVREC
506:         do 280 I = 1, INX
507:           if ( IFI(I) ) then
508:             IDEAD = IDEAD + 1
509:             LSDDED(IDEAD) = IBP(I)
510:             if ( IPAR.ne.0 ) then
511:               WCNTR(7,IPAR) = WCNTR(7,IPAR) + W(I)
512:             end if
513: c##<2007/03/14:PN3:
514:           if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
515:           if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
516:           if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
517: c##>
518: c##<2007/03/14:PN4:
519:           if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
520: c##>

```

src/mvp/seaone.f

```

521:         end if
522: 280      continue
523: C
524:         if ( IPAR.ne.0 ) then
525:             NCNTR(7,IPAR) = NCNTR(7,IPAR) + IDEAD - NDEAD
526:             NCNTR(17,IPAR) = NCNTR(17,IPAR) + IDEAD - NDEAD
527:         else
528:             do 290 I = 1, INX
529:                 if ( IFI(I) ) then
530:                     KP = KKP(IBP(I))
531:                     WCNTR(7,KP) = WCNTR(7,KP) + W(I)
532:                     NCNTR(7,KP) = NCNTR(7,KP) + 1
533:                     NCNTR(17,KP) = NCNTR(17,KP) + 1
534:                 end if
535: 290      continue
536:         end if
537: C
538:         if ( IDEAD.gt.NDEAD ) NDEAD = IDEAD
539: C
540: C
541: C-----
542: C ..... REFLECTION .....
543: C-----
544: C
545:         else if ( MAT.eq.-4000 ) then
546: *VOCL LOOP,NOVREC
547:             do 300 I = 1, INX
548:                 if ( IFI(I) ) then
549:                     IRCF = IRCF + 1
550:                     LSREF(IRCF) = IBP(I)
551:                     ISREF(IRCF) = ISRF(I)
552:                     IZREF(IRCF) = KZ
553:                 end if
554: 300      continue
555: C
556:         else if ( MAT.lt.-1000.and..not.ISLATT(MAT) ) then
557: *VOCL LOOP,NOVREC
558:             do 310 I = 1, INX
559:                 if ( IFI(I) ) then
560:                     IRCF = IRCF + 1
561:                     LSREF(IRCF) = IBP(I)
562:                     ISREF(IRCF) = ISRF(I)
563:                     IZREF(IRCF) = MZONE
564: CCCCCCCCCCCCCCCCCC WCNTR(18) = WCNTR(18) + W(I)
565:                 end if
566: 310      continue
567: C
568: C
569: C-----
570: C ..... ESCAPING LATTICE OR CELL .....
571: C-----
572: C
573: C
574:         else if ( ISLATT(MAT) .or. MAT.eq.-999 ) then
575: *VOCL LOOP,NOVREC
576:             do 320 I = 1, INX
577:                 if ( IFI(I) ) then
578:                     ILAT = ILAT + 1
579:                     LSLAT(ILAT) = IBP(I)
580:                     IZLAT(ILAT) = MLBZZ(KZ)
581:                     IZZ(IBP(I)) = KZ
582:                 end if
583: 320      continue
584:             NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + KK
585:         end if

```

```

586: C
587: C
588: C-----
589: C ..... COMPRESS POINTERS & TEMPORALY ARRAYS .....
590: C-----
591: C
592: C
593: *VOCL LOOP,NOVREC
594:             do 330 I = 1, INX
595:                 if ( .not.IFI(I) ) then
596:                     II = II + 1
597:                     IBP(II) = IBP(I)
598:                     X(II) = X(I)
599:                     Y(II) = Y(I)
600:                     Z(II) = Z(I)
601:                     W(II) = W(I)
602:                 end if
603: 330      continue
604:             INX = II
605: C
606: C
607: C-----
608: C ..... SETTING OF KMEMO ARRAY .....
609: C-----
610: C
611: C
612:             if ( .not.JMEM ) then
613:                 if ( IMEMO.eq.0 ) KMEMO(MZONE,M) = KZ
614:             else
615:                 if ( IMEMO.eq.0 ) then
616:                     KMEMO(MZONE,M) = KZ
617:                     MEMC(MZONE,M) = MEMC(MZONE,M) + KK
618:                     MEMZ(MZONE) = M
619:                 else if ( M.le.NMEMO ) then
620:                     MEMC(MZONE,M) = MEMC(MZONE,M) + KK
621:                 end if
622:             end if
623: C
624:             else if ( IMEMO.eq.0 ) then
625:                 go to 140
626:             end if
627: 340 continue
628: C
629: C
630: C-----
631: C .... UPDATE ZONE # IN BANK
632: C-----
633: C
634: C
635: 350 continue
636: C
637:             do 360 I = NFFL(NZONE+1) + 1, INFL
638:                 IZZ(LSFFL(I)) = IZFFL(I)
639: 360 continue
640: C
641:             if ( IPAR.ne.0 ) then
642:                 NCNTR(17,IPAR) = NCNTR(17,IPAR) + INFL - NFFL(NZONE+1)
643:             end if
644: C
645: C
646: C-----
647: C .... UPDATE OF REFLECTION STACK
648: C-----
649: C
650: C

```

src/mvp/seaone.f

```

651:      if ( JREFL.ne.0.and.IRCF.gt.NBREF(NREFS+1) ) then
652:        do 270 I = NBREF(NREFS+1) + 1, IRCF
653:          CM      IZREF(I) = MZONE
654:        CM270    continue
655:      *VOCL LOOP,SCALAR
656:      do 370 I = NBREF(NREFS+1) + 1, IRCF
657:        NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
658:      370      continue
659:      CCCCCC NCNTR(18) = NCNTR(18) + IRCF - NBREF(NREFS+1)
660:      NBREF(NREFS+1) = IRCF
661:    end if
662:  C
663:  C
664:  C-----
665:  C .... COMPRESS SEARCH STACK .....
666:  C-----
667:  C
668:  C
669:      II = 0
670:  *VOCL LOOP,NOVREC
671:      do 380 I = 1, NNXT(NZONE+1)
672:        if ( IZNXT(I).ne.MZONE ) then
673:          II = II + 1
674:          LSSRC(II) = LSSRC(I)
675:          IZNXT(II) = IZNXT(I)
676:        end if
677:      380      continue
678:  C
679:  CM1999-8-31
680:      NDEAD1 = NDEAD
681:  C
682:  C =====
683:  C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, I TREAT THEM
684:  C AS DEFERRED PARTICLES OR LOST PARTICLES.
685:  C =====
686:  C
687:  C
688:      if ( INX.ne.0 ) then
689:  C
690:  C
691:  C-----
692:  C .... DEFERRED PARTICLES .....
693:  C-----
694:  C
695:  C
696:      CCCC      IF(IMEMO.EQ.1.OR. IMEMO.EQ.0.AND.MMX.LT.NZONE) THEN
697:      if ( IMEMO.eq.1 .or. IMEMO.eq.0.and.KKZ.le.MMZ ) then
698:  CM      &      .AND. MMX.LT.(MMZ-KKZ) ) THEN
699:  C
700:  C .... WARNING FOR DEFERRED PARTICLES ....
701:  C
702:      MDEFR = MDEFR + 1
703:      if ( MDEFR.lt.MWARN ) then
704:  C
705:  C/#IF PARA(SX* CRAY)
706:  *      call MVPSYNC_LOCK(2)
707:  C/#ENDIF
708:  C
709:      write(IOW,7000) INX, MZONE
710:  C
711:  C/#IF PARA(SX* CRAY)
712:  *      call MVPSYNC_UNLOCK(2)
713:  C/#ENDIF
714:  C
715:  7000      format('/' === ',I5,' PARTICLES CAN NOT FIND ',

```

```

716: C##<2007/03/14:PN3:
717: C## &      'NEXT-ENTERING-ZONE IN MEMORIZED DATA (KMEMO) '/'
718: C## &      '      IN ZONE ',I5)
719: &      'NEXT-ENTERING-ZONE IN MEMORIZED DATA (KMEMO) ',
720: &      'IN ZONE ',I5)
721: C##>
722:      end if
723:  C
724:  C
725:      do 390 I = 1, INX
726:  C      LSSRC(NNXT(NZONE+1)+I) = IBP(I)
727:  C      IZNXT(NNXT(NZONE+1)+I) = -MZONE
728:  C      LSSRC(II+I) = IBP(I)
729:  C      IZNXT(II+I) = -MZONE
730:  390      continue
731:      IDEFER = IDEFER + INX
732:      NNXT(NZONE+1) = NNXT(NZONE+1) + INX
733:  C
734:  C
735:  C-----
736:  C ..... LOST PARTICLES OR TANGENTIAL PARTICLES .....
737:  C-----
738:  C
739:  C
740:  CC      ELSE IF(INX.NE.0.AND.IMEMO.EQ.0) THEN
741:      else
742:        if ( KZMAT(MZONE).ge.0 ) then
743:  C
744:  C .... CHECK FOR EACH PARTICLE. IFI(I) = .FALSE./.TRUE. = OUT/IN
745:  C
746:  Ccccc      if( jdebug(1).eq.0 ) then
747:  C      JRR = .FALSE.
748:  C
749:  C      CALL JUDGE('SEAONE',MZONE,INX,X,Y,Z,IFI,
750:  C      &      SDA,KZDA,KZAA,NSDA,NZDA,
751:  C      &      NZONE+1,JVMNT,IFL,JSIMP,JRR,KSREF,ISRF,FXYZ)
752:  Ccccc      else
753:  C      JRR = .true.
754:  C
755:  C ... check the surface nearest to the particle ..
756:  C      ( KSFBD is used in place of KSREF for this case )
757:  C
758:  C      call JUDGE( 'SEAONE', MZONE, INX, X, Y, Z, IFI, SDA,
759:  C      &      KZDA, KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL,
760:  C      &      JSIMP, JRR, KSFBD, ISRF, FXYZ )
761:  Ccccc      endif
762:  C
763:      IDEAD = 0
764:      IFFL = 0
765:  C
766:      do 400 I = 1, INX
767:      if ( IFI(I) ) then
768:  C      IFFL = IFFL + 1
769:  C      LSFFL(INFL+IFFL) = IBP(I)
770:  C      IZFFL(INFL+IFFL) = MZONE
771:  C      XXX(IBP(I)) = X(I)
772:  C      YYY(IBP(I)) = Y(I)
773:  C      ZZZ(IBP(I)) = Z(I)
774:  C      KLSF(IBP(I)) = 0
775:  C
776:  C-----
777:  C ..... LOST
778:  C-----
779:  C
780:      else

```

src/mvp/seaone.f

```

781:      IDEAD = IDEAD + 1
782:      LSDED(NDEAD+IDEAD) = IBP(I)
783: c##<2007/03/14:PN3:
784:      if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
785:      if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
786:      if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
787: c##>
788: c##<2007/03/14:PN4:
789:      if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
790: c##>
791:      end if
792:      400 continue
793: C
794: C .... WARNING FOR TANGENTIAL PARTICLES ....
795: C
796:      MTANG = MTANG + 1
797:      if ( IFFL.gt.0.and.MTANG.lt.MWARN ) then
798:          IP1 = LSFFL(INFL+1)
799: C/#IF PARA(SX* CRAY)
800: *      call MVPSYNC_LOCK(2)
801: C/#ENDIF
802:      write(IOW,7020) IFFL, MZONE, XXX(IP1), YYY(IP1),
803:      &      ZZZ(IP1), AAA(IP1), BBB(IP1), CCC(IP1)
804: C/#IF PARA(SX* CRAY)
805: *      call MVPSYNC_UNLOCK(2)
806: C/#ENDIF
807: C
808: c##<2007/03/14:PN3:
809: c7020      format(/' === TRACKS OF ',I5,' PARTICLES ',
810: c## &      'MAY TOUCH THE BOUNDARY OF ZONE ',I5,'.'/
811: c## &      ' (OTHERWISE JUST ON-EDGE OR AT-CORNER) '
812: c## &      ' ONE OF THEM HAS THE FOLLOWING POSITION &',
813: c## &      ' DIRECTION;.'/ POSITION: ',3E12.5,
814: c## &      ' DIRECTION: ',3E12.5)
815: 7020 format(/' === TRACKS OF ',I5,' PARTICLES MAY TOUCH THE BOUNDARY',
816: & ' OF ZONE ',I5,'. (OTHERWISE JUST ON-EDGE OR AT-CORNER)')
817: &/' ONE OF THEM HAS THE FOLLOWING POSITION & DIRECTION;.'/
818: &/' POSITION: ',1P,3E14.6,' DIRECTION: ',3E13.5)
819: c##>
820:      end if
821: C
822: C
823:      NFFL(MZONE) = NFFL(MZONE) + IFFL
824:      INFL = INFL + IFFL
825:      else
826:      do 410 I = 1, INX
827:          IFI(I) = .false.
828:          LSDED(NDEAD+I) = IBP(I)
829: c##<2007/03/14:PN3:
830:      if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
831:      if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
832:      if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
833: c##>
834: c##<2007/03/14:PN4:
835:      if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
836: c##>
837:      410 continue
838:      IDEAD = INX
839:      end if
840: C
841:      NDEAD = NDEAD + IDEAD
842:      NLOST = NLOST + IDEAD
843: C
844:      if ( IDEAD.gt.0 ) then
845: C/#IF PARA(SX* CRAY)

```

```

846: *      call MVPSYNC_LOCK(2)
847: C/#ENDIF
848:      write(IOW,'(/1x,a,i5,a,i5)') '!!!(SEAONE) ', IDEAD,
849:      &      ' particles are lost !! ZONE # = ', MZONE
850:      do 430 I = 1, INX
851:          if ( .not.IFI(I) ) then
852: C
853:              if ( JDEBG(1).eq.0 ) then
854:                  write(IOW,7040) IBP(I), XXX(IBP(I)),
855:                  &      YYY(IBP(I)), ZZZ(IBP(I)), WWW(IBP(I)),
856:                  &      AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I))
857:              else
858:                  write(IOW,7060) IBP(I), XXX(IBP(I)),
859:                  &      YYY(IBP(I)), ZZZ(IBP(I)), WWW(IBP(I)),
860:                  &      AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I)),
861:                  &      ISRF(I)
862:              end if
863: C
864:              if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 )
865:                  &      write(IOW,7080)
866:                  &      (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,LEVL(IBP(I))
867:                  &      )
868:              if ( JTLLT.ne.0 ) then
869:                  do 420 L = 1, LEVL(IBP(I))
870:                      call PRNSPC( IOW, A, CHA, IBSPC(IBP(I),L), L
871:                      &      )
872:                  420 continue
873:              end if
874:              end if
875:              430 continue
876: C/#IF PARA(SX* CRAY)
877: *      call MVPSYNC_UNLOCK(2)
878: C/#ENDIF
879:      end if
880: C
881:      7040      format(1X,I5,' X,Y,Z=',1P,E12.5,',',E12.5,',',E12.5,' W=',
882:      &      E11.4,' MU=',E11.4,' ETA=',E11.4,' XI=',E11.4)
883:      7060      format(1X,I5,' X,Y,Z=',1P,E12.5,',',E12.5,',',E12.5,' W=',
884:      &      E11.4,' MU=',E11.4,' ETA=',E11.4,' XI=',E11.4,
885:      &      ' BODYID=',I5)
886: C
887:      7080      format(1X,' LEVL = ',I3,' LZZ = ',I5,' LPOS = ',I5)
888:      end if
889:      end if
890:      end if
891: C
892: C
893: C-----
894: C .... ADJUSTMENT OF NUMBER OF PARTICLE IN STACKS .....
895: C-----
896: C
897: C
898:      NNXT(NZONE+1) = NNXT(NZONE+1) - NNXT(MZONE)
899:      NNXT(MZONE) = 0
900:      ISF0 = NFFL(NZONE+1) + 1
901:      NFFL(NZONE+1) = INFL
902:      if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
903: C
904:      go to 640
905: C
906: C-----
907: C .... TAKE special tallies .....
908: C-----
909: C
910: C

```

src/mvp/seaone.f

```

911: C      if ( NTEVE.gt.0.and.JTEVE(7).gt.0.and.NDEAD1-NDEAD0.gt.0 ) then
912: C          ISAFE = NDEAD1 - NDEAD0
913: C
914: C          do 460 J = 0, JTEVE(7) - 1
915: C
916: C      ... pointer to direct tally (idt) & d-tally # (it) ...
917: C          IDT = KTEVE(LTEVE(7)+J) - 1
918: C          IT = IDTAL(IDT+9)
919: C
920: C      .. neutron or photon ?
921: C
922: C          NLEK = 0
923: C          if ( IPAR.eq.0 ) then
924: C              if ( IDTAL(IDT+5).eq.1 ) then
925: C                  do 400 I = 1, ISAFE
926: C                      IBPIII = LSDED(NDEAD0+I)
927: C                      if ( IGG(IBPIII).le.NGP1 ) then
928: C                          NLEK = NLEK + 1
929: C                          IBP(NLEK) = IBPIII
930: C                      end if
931: C                  continue
932: C              else if ( IDTAL(IDT+5).eq.2 ) then
933: C                  do 410 I = 1, ISAFE
934: C                      IBPIII = LSDED(NDEAD0+I)
935: C                      if ( IGG(IBPIII).gt.NGP1 ) then
936: C                          NLEK = NLEK + 1
937: C                          IBP(NLEK) = IBPIII
938: C                      end if
939: C                  continue
940: C              end if
941: C          else if ( IPAR.eq.IDTAL(IDT+5) ) then
942: C              do 420 I = 1, ISAFE
943: C                  IBP(I) = LSDED(NDEAD0+I)
944: C              continue
945: C              NLEK = ISAFE
946: C          end if
947: C
948: C          do 430 I = 1, NLEK
949: C              IWK(I) = IGG(IBP(I))
950: C              W(I) = WWW(IBP(I))
951: C              if ( JTLLT.eq.0 ) then
952: C                  IZI(I) = KZREG(IZZ(IBP(I)))
953: C              else
954: C                  IZI(I) = IBREG(IBP(I))
955: C              end if
956: C              if ( JTIME.gt.0 ) ISRF(I) = ITT(IBP(I))
957: C          continue
958: C
959: C      ... skip if current region does not need this tally ...
960: C
961: C          do 440 I = 1, NLEK
962: C              if ( JDTRG(IZI(I),IT).ne.0 ) go to 450
963: C          continue
964: C          go to 460
965: C      continue
966: C
967: C          if ( NLEK.gt.0 ) then
968: C              call STALN7( IOW, IDTAL(IDT+1), W, IBP, NLEK, IWK, IZI,
969: C          & ISRF, NGROUP, NGP1, NREG, NTIME, NBANK, DTALY,
970: C          & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, X, Y )
971: C          end if
972: C      460 continue
973: C      end if
974: C
975: C-----

```

```

976: C**** RUSSIAN ROULETTE KILL & SPLITTING BASED ON IMPORTANCE, WEIGHT
977: C-----
978: C
979: C
980: C      if ( JVARR.ne.0.and.NFFL(NZONE+1).ge.ISF0 ) then
981: C          NVARI = NFFL(NZONE+1) - ISF0 + 1
982: C          call SPLITB( 0, IPAR, ISF0, NVARI, IRAND, JMNTR, JIMPT,
983: C          & JWWND, JRRLT, JLATT, JHLAT, JTLLT, JTIME, JWTIM, JRWVR,
984: C          & NBANK, NZONE, NGROUP, NGP1, NGP2, NREG, NTIME, NEST,
985: C          & LSFFL, IZFFL, NFFL, LSDED, NDEAD, NSMAC, NSMIC, XIMP,
986: C          & WKIL, WSRV, WTIME, WLLIM, KZREG, NCNTR, WCNTR, NEVENT,
987: C          & XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IZZ, IGG, TTT, ITT,
988: C          & LEVL, LZZ, LPOS, LCRS, XIM, KLSF, EEE, IBREG, IBSPC,
989: C          & MB, NUC, NSTAL, SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL,
990: C          & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, NKILD, WKILD,
991: C          & NSURV, WSURV, NSPLT, WSPLT, X, Y, Z, W, IZI, IWK )
992: C      end if
993: C
994: C      return
995: C
996: C
997: C
998: C =====
999: C ... DEFERRED PARTICLES .....
1000: C =====
1001: C
1002: C
1003: C      440 II = 0
1004: C          ISF0 = NFFL(NZONE+1) + 1
1005: C
1006: C      do 450 I = 1, NNXT(NZONE+1)
1007: C          if ( IZNXT(I).lt.0 ) then
1008: C              II = II + 1
1009: C              IBP(II) = LSSRC(I)
1010: C              IZI(II) = -IZNXT(I)
1011: C          end if
1012: C      450 continue
1013: C
1014: C      if ( II.ne.IDEFER ) then
1015: C          c##<2007/03/14:PN3:
1016: C          c## write(IOW,*) 'XXX(SEAONE) NUMBER OF DEFERRED PARTICLES IN ',
1017: C          write(IOW,'(1X,A,A,2I8)')
1018: C          & 'XXX(SEAONE) NUMBER OF DEFERRED PARTICLES IN',
1019: C          c##>
1020: C          & ' SEARCH STACK IS DIFFERENT FROM IDEFER !! ', II,
1021: C          & IDEFER
1022: C          stop 666
1023: C      end if
1024: C
1025: C      do 460 I = 1, IDEFER
1026: C          X(I) = XXX(IBP(I)) + DEPS*AAA(IBP(I))
1027: C          Y(I) = YYY(IBP(I)) + DEPS*BBB(IBP(I))
1028: C          Z(I) = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
1029: C          W(I) = WWW(IBP(I))
1030: C      460 continue
1031: C
1032: C-----
1033: C ... CHECK ALL ZONES .....
1034: C-----
1035: C
1036: C      INX = IDEFER
1037: C      IDEAD = NDEAD
1038: C      IRCF = 0
1039: C      ILAT = 0
1040: C      if ( JREFL.ne.0 ) IRCF = NBREF(NREFS+1)

```

src/mvp/seaone.f

```

1041:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
1042:      KKZ      = 0
1043:      MMZ      = NZONE
1044: C
1045: C
1046: C
1047:      ITANG    = 0
1048: C
1049:      do 560 KZ = 1, NZONE
1050:      if ( INX.eq.0 ) go to 570
1051: C
1052: C      JRR = .TRUE. IF ZONE KZ IS REFLECTIVE.
1053: C
1054:      JRR      = JREFL.ne.0.and.KZMAT(KZ) .le. - 1001
1055:      &        .and..not.ISLAT(KZMAT(KZ))
1056: C
1057:      call JUDGE( 'SEARCH', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
1058:      &        NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, JRR, KSREF,
1059:      &        ISRF, FXYZ )
1060: C
1061:      II       = 0
1062:      if ( JLATT.ne.0 ) then
1063:      ICEL     = KCELL(KZ)
1064:      MMZ      = IPCEL(ICEL+1) - 1
1065:      if ( ICEL.gt.0 ) then
1066:      KKZ      = IPCEL(ICEL) - 1
1067:      else
1068:      KKZ      = 0
1069:      end if
1070:      end if
1071: C
1072:      if ( KZMAT(KZ).lt.0 ) then
1073:      *VOCL LOOP,NOVREC
1074:      do 470 I = 1, INX
1075:      IBPIII   = IBP(I)
1076:      if ( (.not.IFI(I)) .or. IZI(I).eq.KZ .or. IZI(I).le.KKZ
1077:      &      .or. IZI(I).gt.MMZ ) then
1078:      IFI(I)   = .false.
1079:      else
1080:      IFI(I)   = .true.
1081:      end if
1082:      470      continue
1083:      *VOCL LOOP,NOVREC
1084:      do 480 I = 1, INX
1085:      IBPIII   = IBP(I)
1086:      if ( IFI(I) ) then
1087: C
1088: C      .... FREE-FLIGHT ....
1089: C      IF(KZMAT(KZ).GE.0) THEN
1090: C      IFFL = IFFL + 1
1091: C      LSFFL(NFFL(NZONE+1)+IFFL) = IBP(I)
1092: C      IZZ(IBP(I)) = KZ
1093: C
1094: C-----
1095: C      .... LEAKAGE      ....
1096: C-----
1097: C
1098: C
1099:      if ( KZMAT(KZ).eq.-1000 ) then
1100:      NDEAD    = NDEAD + 1
1101:      LSDDED(NDEAD) = IBPIII
1102: c##<2007/03/14:PN3:
1103:      if ( KIBNK(17).ne.0 ) IBNK(IBPIII,KIBNK(17)) = 0
1104:      if ( KIBNK(18).ne.0 ) IBNK(IBPIII,KIBNK(18)) = 0
1105:      if ( KIBNK(19).ne.0 ) IBNK(IBPIII,KIBNK(19)) = 0

```

```

1106: c##>
1107: c##<2007/03/14:PN4:
1108:      if ( JPHNU.ne.0 ) KPNFG(IBPIII) = 0
1109: c##>
1110: C
1111:      if ( IPAR.ne.0 ) then
1112:      WCNTR(7,IPAR) = WCNTR(7,IPAR) + W(I)
1113:      end if
1114:      end if
1115: C
1116: C-----
1117: C      .... periodic boundary ....
1118: C-----
1119: C
1120:      if ( KZMAT(KZ).eq.-4000 ) then
1121:      IRCF      = IRCF + 1
1122:      LSREF(IRCF) = IBPIII
1123:      ISREF(IRCF) = ISRF(I)
1124:      IZREF(IRCF) = KZ
1125: C
1126: C-----
1127: C      .... REFLECTION ....
1128: C-----
1129: C
1130:      else if ( KZMAT(KZ).lt.-1000
1131:      &      .and..not.ISLAT(KZMAT(KZ)) ) then
1132:      IRCF      = IRCF + 1
1133:      LSREF(IRCF) = IBPIII
1134:      ISREF(IRCF) = ISRF(I)
1135:      IZREF(IRCF) = IZZ(IBPIII)
1136:      CCCCCCCCCC WCNTR(18) = WCNTR(18) + W(I)
1137:      end if
1138: C
1139: C-----
1140: C      .... LATTICE-SEARCH ....
1141: C-----
1142: C
1143: C
1144: C
1145:      if ( ISLAT(KZMAT(KZ)) .or. KZMAT(KZ).eq.-999 ) then
1146:      ILAT      = ILAT + 1
1147:      LSLAT(ILAT) = IBPIII
1148:      IZLAT(ILAT) = MLBZZ(KZ)
1149:      end if
1150:      end if
1151:      480      continue
1152: C
1153:      if ( JLATT.ne.0.and.ILAT.gt.NXLT(NLBZ+1) ) then
1154:      NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + ILAT - NXLT(NLBZ+1)
1155:      NXLT(NLBZ+1) = ILAT
1156:      end if
1157: C
1158:      if ( KZMAT(KZ).eq.-1000.and.IPAR.eq.0 ) then
1159:      *VOCL LOOP,SCALAR
1160:      do 490 I = 1, INX
1161:      if ( IFI(I) ) then
1162:      KP      = KKP(IBP(I))
1163:      WCNTR(7,KP) = WCNTR(7,KP) + W(I)
1164:      NCNTR(7,KP) = NCNTR(7,KP) + 1.0D0
1165:      end if
1166:      490      continue
1167:      end if
1168: C
1169:      *VOCL LOOP,NOVREC
1170:      do 500 I = 1, INX

```

src/mvp/seaone.f

```

1171:      IBPIII = IBP(I)
1172:      if ( .not.IFI(I) ) then
1173:        II = II + 1
1174:        IBP(II) = IBPIII
1175:        X(II) = X(I)
1176:        Y(II) = Y(I)
1177:        Z(II) = Z(I)
1178:        W(II) = W(I)
1179:        IZI(II) = IZI(I)
1180:      end if
1181:      500 continue
1182:
1183: C
1184: C .... KZMAT(KZ) >= 0 ....
1185: C
1186:      else
1187:        IFFL = 0
1188: *VOCL LOOP,NOVREC
1189:      do 510 I = 1, INX
1190:        IBPIII = IBP(I)
1191:        IZIIII = IZI(I)
1192:        if ( (.not.IFI(I)) .or. IZIIII.le.KKZ .or. IZIIII.gt.MMZ
1193:          &
1194:            ) then
1195:          II = II + 1
1196:          IBP(II) = IBPIII
1197:          X(II) = X(I)
1198:          Y(II) = Y(I)
1199:          Z(II) = Z(I)
1200:          W(II) = W(I)
1201:          IZI(II) = IZIIII
1202:        else
1203: C
1204: C -----
1205: C ..... FREE-FLIGHT .....
1206: C -----
1207: C
1208: C
1209:        IFFL = IFFL + 1
1210:        LSFFL(NFFL(NZONE+1)+IFFL) = IBPIII
1211:        IZZ(IBP(I)) = KZ
1212: C
1213: C .... check marker region ...
1214: C
1215: CC      if ( NMKREG.ne.0 ) then
1216: CC        IRRG = KZREG(IZIIII)
1217: CC        if ( JTLT.ne.0 ) then
1218: CC          IRRG = IBREG(IBPIII)
1219: CC        end if
1220: CC
1221: CC        IRRG2 = KZREG(KZ)
1222: CC        if ( JTLT.ne.0 ) then
1223: CC          ILV = LEVL(IBPIII)
1224: CC          if ( ILV.gt.0 ) then
1225: CC            IRRG2 = ISUSP(KZREG(KZ),IBSPC(IBPIII,ILV))
1226: CC          end if
1227: CC        end if
1228: CC        if ( MKREG(IRRG,1).ne.0.and.IRRG.ne.IRRG2 ) then
1229: CC          KG = KIBNK(9) + MKREG(IRRG,1) - 1
1230: CC          IBNK(IBPIII,KG) = IBNK(IBPIII,KG) + 1
1231: CC        end if
1232: CC      end if
1233: C
1234: C .... marker region procedure is modified in Jan 2000,
1235: C      to treat marker-region in lattice geometry.

```

```

1236: C
1237: C
1238: C ... check previous region for MARKER-REGION flag
1239: C
1240: C      if ( NMKREG.ne.0 ) then
1241: C        ... IBREG is set always (from Jan 2000)
1242: C        IRRG = IBREG(IBPIII)
1243: C
1244: C ... this is the first flight after birth as source
1245: C      particle, so particle has no "previous region".
1246: C
1247: C      if ( IBNK(IBPIII,KIBNK(5)).lt.0 ) then
1248: C        IRRG = 0
1249: C      end if
1250: C
1251: C .... SET REGION # FOR UNIVERSE-DEPENDENT TALLY MODE ...
1252: C
1253: C ... IBREG is set anytime (from Jan 2000)
1254: C      IBREG(IBPIII) = KZREG(KZ)
1255: C      if ( JTLT.ne.0 ) then
1256: C        ILV = LEVL(IBPIII)
1257: C        if ( ILV.gt.0.and.KZREG(KZ).gt.0 ) then
1258: C          IBREG(IBPIII) =
1259: C            &
1260: C            ISUSP(KZREG(KZ),IBSPC(IBPIII,ILV))
1261: C        end if
1262: C      end if
1263: C
1264: C .... check marker region escape ...
1265: C
1266: C      if ( NMKREG.ne.0.and.IRRG.ne.0 ) then
1267: C        IRRG2 = IBREG(IBPIII)
1268: C        if ( MKREG(IRRG,1).ne.0.and.IRRG.ne.IRRG2 ) then
1269: C          KG = KIBNK(9) + MKREG(IRRG,1) - 1
1270: C          IBNK(IBPIII,KG) = IBNK(IBPIII,KG) + 1
1271: C        end if
1272: C      end if
1273: C
1274: C -----
1275: C ..... TANGENTIAL PARTICLES
1276: C -----
1277: C
1278: C
1279: C      if ( IZIIII.eq.KZ ) then
1280: C        ITANG = ITANG + 1
1281: C        XXX(IBPIII) = X(I)
1282: C        YYY(IBPIII) = Y(I)
1283: C        ZZZ(IBPIII) = Z(I)
1284: C      end if
1285: C    end if
1286: C  510 continue
1287: C
1288: C      if ( IFFL.gt.0 ) then
1289: C        do 520 I = 1, IFFL
1290: C          IZFFL(NFFL(NZONE+1)+I) = KZ
1291: C        520 continue
1292: C
1293: C ... set region# and/or zone# on birth here !!!
1294: C
1295: C      if ( JLATT.ne.0.and.KIBNK(5).ne.0
1296: C        &
1297: C        .and.(KIBNK(2).ne.0.or.KIBNK(3).ne.0) ) then
1298: C        MNM = 0
1299: C
1300: C      do 530 I = 1, IFFL
1301: C        IP = LSFFL(NFFL(NZONE+1)+I)

```


src/mvp/seaone.f

```

1301:          if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1302:              NMNM      = NMNM + 1
1303:          end if
1304: 530      continue
1305:
1306:          if ( NMNM.gt.0.and.KIBNK(2).ne.0 ) then
1307: *VOCL LOOP,NOVREC
1308:              do 540 I = 1, IFFL
1309:                  IP      = LSFFL(NFFL(NZONE+1)+I)
1310:                  if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1311:                      if ( JTLIT.ne.0 ) then
1312:                          IBNK(IP,KIBNK(2)) = IBREG(IP)
1313:                      else
1314:                          IBNK(IP,KIBNK(2)) = KZREG(KZ)
1315:                      end if
1316:                  end if
1317: 540      continue
1318:          end if
1319:
1320:          if ( NMNM.gt.0.and.KIBNK(3).ne.0 ) then
1321: *VOCL LOOP,NOVREC
1322:              do 550 I = 1, IFFL
1323:                  IP      = LSFFL(NFFL(NZONE+1)+I)
1324:                  if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1325:                      IBNK(IP,KIBNK(3)) = KZ
1326:                  end if
1327: 550      continue
1328:          end if
1329:      end if
1330: C
1331:          NFFL(NZONE+1) = NFFL(NZONE+1) + IFFL
1332:          NFFL(KZ)      = NFFL(KZ) + IFFL
1333:      end if
1334:  end if
1335: C
1336:          INX      = II
1337: 560 continue
1338: C
1339: C .... WARNING FOR TANGENTIAL PARTICLES ....
1340: C
1341:          if ( ITANG.gt.0 ) then
1342:              MTANG = MTANG + 1
1343:          if ( MTANG.lt.MWARN ) then
1344: C/#IF PARA(SX* CRAY)
1345: *      call MVPSYNC_LOCK(2)
1346: C/#ENDIF
1347:          write(IOW,7100) ITANG
1348: C/#IF PARA(SX* CRAY)
1349: *      call MVPSYNC_UNLOCK(2)
1350: C/#ENDIF
1351: C
1352: 7100      format(' == ',I5,' PARTICLES MAY BE TANGENTIAL',
1353: &              ' TO THE BOUNDARIES OF ZONE. ',
1354: &              ' (OTHERWISE JUST ON-EDGE OR AT-CORNER)')
1355:          end if
1356:      end if
1357: C
1358: C
1359: 570 if ( IPAR.ne.0 ) NCNTR(7,IPAR) = NCNTR(7,IPAR) + NDEAD - IDEAD
1360: C
1361: CM1999-8-31
1362:      NDEAD1 = NDEAD
1363: C
1364: C ..... LOST PARTICLES .....
1365: C

```

```

1366: C
1367:          if ( INX.ne.0 ) then
1368:              do 580 I = 1, INX
1369:                  LSDED(NDEAD+I) = IBP(I)
1370: C##<2007/03/14:PN3:
1371:                  if ( KIBNK(17).ne.0 ) IBNK(IBP(I),KIBNK(17)) = 0
1372:                  if ( KIBNK(18).ne.0 ) IBNK(IBP(I),KIBNK(18)) = 0
1373:                  if ( KIBNK(19).ne.0 ) IBNK(IBP(I),KIBNK(19)) = 0
1374: C##>
1375: C##<2007/03/14:PN4:
1376:                  if ( JPHNU.ne.0 ) KPNFG(IBP(I)) = 0
1377: C##>
1378: 580      continue
1379:          NDEAD = NDEAD + INX
1380:          NLOST = NLOST + INX
1381: C/#IF PARA(SX* CRAY)
1382: *      call MVPSYNC_LOCK(2)
1383: C/#ENDIF
1384:          write(IOW,'(/lx,a,i5,a)') '!!!(SEAONE) ', INX,
1385: &          ' particles are LOST in deferred search. '
1386:          do 600 I = 1, INX
1387:              write(IOW,7040) IBP(I), X(I), Y(I), Z(I), WWW(IBP(I)),
1388: &              AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I))
1389:              if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 ) then
1390:                  write(IOW,7080)
1391: &                  (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,LEVL(IBP(I)))
1392:              end if
1393:              if ( JTLIT.ne.0 ) then
1394:                  do 590 L = 1, LEVL(IBP(I))
1395:                      call PRNSPC( IOW, A, CHA, IBSPC(IBP(I),L), L )
1396: 590      continue
1397:              end if
1398:          continue
1399: C/#IF PARA(SX* CRAY)
1400: *      call MVPSYNC_UNLOCK(2)
1401: C/#ENDIF
1402:          end if
1403: C
1404: C
1405: C ..... COUNT THE NUMBER OF BOUNDARY CROSSING
1406: C
1407: C
1408:          if ( IPAR.ne.0 ) then
1409:              NCNTR(17,IPAR) = NCNTR(17,IPAR) + IDEFER - INX
1410:          else
1411: *VOCL LOOP,SCALAR
1412:              do 610 I = 1, IFFL
1413:                  KP      = KKP(LSFFL(NFFL(NZONE+1)+I))
1414:                  NCNTR(17,KP) = NCNTR(17,KP) + 1
1415: 610      continue
1416:              end if
1417: C
1418: C .... UPDATE REFLECTION STACK
1419: C
1420:          if ( JREFL.ne.0.and.IRCF.gt.NBREF(NREFS+1) ) then
1421: CM      do 470 I = NBREF(NREFS+1) + 1, IRCF
1422: CM          IZREF(I) = IZZ(LSREF(I))
1423: CM470      continue
1424: *VOCL LOOP,SCALAR
1425:              do 620 I = NBREF(NREFS+1) + 1, IRCF
1426:                  NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
1427: 620      continue
1428: CCCCCCCC NCNTR(18) = NCNTR(18) + IRCF - NBREF(NREFS+1)
1429:          NBREF(NREFS+1) = IRCF
1430:          end if

```

src/mvp/seaone.f

```

1431: C
1432: C .... COMPRESS SEARCH STACK .....
1433: C
1434:       II      = 0
1435: *VOCL LOOP,NOVREC
1436:       do 630 I = 1, NNXT(NZONE+1)
1437:         if ( IZNXT(I).ge.0 ) then
1438:           II      = II + 1
1439:           LSSRC(II) = LSSRC(I)
1440:           IZNXT(II) = IZNXT(I)
1441:         end if
1442:       630 continue
1443:       NNXT(NZONE+1) = NNXT(NZONE+1) - IDEFER
1444: CM1993-9-8 IF( JLATT .NE. 0 ) NXLT(NLEZ+1) = ILAT
1445:       IDEFER = 0
1446: C
1447: C
1448: C
1449:       640 continue
1450: C
1451: C
1452: C-----
1453: C .... TAKE special tallys .....
1454: C-----
1455: C
1456: C
1457:       if ( NTEVE.gt.0.and.JTEVE(7).gt.0.and.NDEAD1-NDEAD0.gt.0 ) then
1458:         ISAFE = NDEAD1 - NDEAD0
1459:
1460:         do 650 I = 1, ISAFE
1461:           IBP(I) = LSDED(NDEAD0+I)
1462:       650 continue
1463: C
1464:       KS      = 1
1465:       if ( NPKIND.eq.1 ) then
1466:         KS      = 1
1467:         KSORT(1) = KKP(IBP(1))
1468:         IKSORT(1) = 1
1469:         IKSORT(2) = ISAFE + 1
1470:       else
1471:         call KPSORT( KKP, NBANK, KPLIM, IBP, ISAFE, KS, KSORT(1)
1472:       & IKSORT(1), IWK )
1473:       end if
1474: C
1475:       do 710 J = 0, JTEVE(7) - 1
1476: C
1477: C ... pointer to direct tally (idt) & d-tally # (it) ...
1478:       IDT      = KTEVE(LTEVE(7)+J) - 1
1479:       IT        = IDTAL(IDT+9)
1480: C
1481: C .. neutron or photon ?
1482: C
1483:       NLEK      = 0
1484:       if ( IPAR.eq.0 ) then
1485:         if ( IDTAL(IDT+5).eq.1 ) then
1486:           do 640 I = 1, ISAFE
1487:             IBPIII = LSDED(NDEAD0+I)
1488:             if ( IGG(IBPIII).le.NGPI ) then
1489:               NLEK = NLEK + 1
1490:               IBP(NLEK) = IBPIII
1491:             end if
1492:           640 continue
1493:         else if ( IDTAL(IDT+5).eq.2 ) then
1494:           do 650 I = 1, ISAFE
1495:             IBPIII = LSDED(NDEAD0+I)

```

```

1496: C           if ( IGG(IBPIII).gt.NGPI ) then
1497: C             NLEK = NLEK + 1
1498: C             IBP(NLEK) = IBPIII
1499: C           end if
1500: C 650 continue
1501: C       end if
1502: C     else if ( IPAR.eq.IDTAL(IDT+5) ) then
1503: C       do 660 I = 1, ISAFE
1504: C         IBP(I) = LSDED(NDEAD0+I)
1505: C 660 continue
1506: C       NLEK = ISAFE
1507: C     end if
1508: C
1509: C     KK      = 1
1510: C     NLEK     = 0
1511: C     if ( IPAR.eq.IDTAL(IDT+5) ) then
1512: C       NLEK = ISAFE
1513: C     else
1514: C       do 660 K = 1, KS
1515: C         if ( IDTAL(IDT+5).eq.KSORT(K) ) then
1516: C           KK = IKSORT(K)
1517: C           NLEK = IKSORT(K+1) - KK
1518: C           go to 670
1519: C         end if
1520: C 660 continue
1521: C 670 continue
1522: C     end if
1523: C     if ( NLEK.eq.0 ) go to 710
1524: C
1525: C     KK2      = KK + NLEK - 1
1526: C
1527: C     do 680 I = KK, KK2
1528: C       IWK(I) = IGG(IBP(I))
1529: C       W(I)   = WWW(IBP(I))
1530: C       if ( JTLLT.eq.0 ) then
1531: C         IZI(I) = KZREG(IZZ(IBP(I)))
1532: C       else
1533: C         IZI(I) = IBREG(IBP(I))
1534: C       end if
1535: C       if ( JTIME.gt.0 ) ISRF(I) = ITT(IBP(I))
1536: C       if ( JTIME.gt.0 ) DWK1(I) = TTT(IBP(I)) ! time-list
1537: C 680 continue
1538: C
1539: C ... skip if current region does not need this tally ...
1540: C
1541: C     do 690 I = KK, KK2
1542: C       if ( JDTRG(IZI(I),IT).ne.0 ) go to 700
1543: C 690 continue
1544: C       go to 710
1545: C 700 continue
1546: C
1547: C     if ( NLEK.gt.0 ) then
1548: C       call STALN7( IOW, IDTAL(IDT+1), W(KK), IBP(KK), NLEK,
1549: C       & IWK(KK), IZI(KK), ISRF(KK), NGROUP, NREG, NTIME,
1550: C       & NBANK, DTALY, DBNK, KDBNK, MDBNK, IBNK, KIBNK,
1551: C       & MIBNK, X, Y,
1552: C       & DWK1(KK), JTLST, IOTL) ! time-list
1553: C     end if
1554: C 710 continue
1555: C     end if
1556: C
1557: C
1558: C**** RUSSIAN ROULETTE KILL & SPLITTING BASED ON IMPORTANCE, WEIGHT
1559: C
1560: C

```

src/mvp/seaone.f

```
1561:      if ( JVARR.ne.0.and.NFFL(NZONE+1).ge.ISF0 ) then
1562:
1563:          NVARI    = NFFL(NZONE+1) - ISF0 + 1
1564: C
1565:      JJCOL = 0 ! photonuc
1566:      call SPLITB( JJCOL, IPAR, ISF0, NVARI, IRAND, NPKIND, NBANK,
1567: &                NZONE,
1568: &                NGROUP, NREG, NTIME, NEST, LSFFL, IZFFL, NFFL, LSDDED,
1569: &                NDEAD, NSMAC, NSMIC, XIMP, WKIL, WSRV, WTIME, WLLIM,
1570: &                KZREG, NCNTR, WCNTR, NEVENT, XXX, YYY, ZZZ, AAA, BBB,
1571: &                CCC, WWW, KKP, IZZ, IGG, TTT, ITT, LEVL, LZZ, LPOS,
1572: &                LCRS, XIM, KLSF, EEE, IBREG, IBSPC, MB, NUC, NSTAL,
1573: &                SMAC, KMAC, MMAC, SMIC, KSPI, SGTAL, DBNK, KDBNK,
1574: &                MDBNK, IBNK, KIBNK, MIBNK, NKILD, WKILD, NSURV, WSURV,
1575: &                NSPLT, WSPLT, X, Y, Z, W, IZI, IWK, IFL,
1576: &                SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD, ! pert
1577: &                WSC, NORDDS, NOBDDS,
1578: &                NUCPN, NSMICPN, SMICPN, ! photonuc
1579: &                WWB, WSB, NPTBE, ! beff
1580: &                WWBD, WSD, WWLD, WSLD ! beff
1581: &                )
1582: C
1583:      end if
1584: C
1585:      return
1586:      end
```

src/mvp/search.f

```

1: C/IF ARGSAVE
2: * SUBROUTINE SEARCH0(IOW, NBANK,NZONE,NSDA,NZDA,NLOST,NDEAD,
3: * N NREFS,NGROUP,NGP1,NGP2,NEVENT,NREG,NTIME,
4: * N IRAND,NLBZ,NEST,DEPS,NSMAC,NSMIC,MB,NSTAL,NUC,NSPACE,NSUZON,
5: * B XXX,YYY,ZZZ,AAA,BBB,CCC,
6: * B WWW,IZZ,IGG,TTT,ITT,EEE,XIM,KLSF,
7: * B SMAC,SMIC,SGTAL,KMAC,MMAC,KSPI,
8: * B LEVL,LZZ,LPOS,LCRS,IBREG,IBSPC,
9: * S LSFFL,NFFL,IZFFL,LSSRC,NNXT,IZNXT,
10: * S LSDED,LSREF,ISREF,IZREF,NBREF,KSREF,
11: * * LSLAT,IZLAT,NXLT,
12: * G SDA,KZMAT,KZREG,KZDA,KZAA,NKZAA,
13: * * KCELL,IPCEL,MLBZZ,ISUSP,
14: * T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD,WKILD,NSURV,
15: * T WSURV,NSPLT,WSPLT,NCNTR,WCNTR,
16: * W IZT,IPZONE,MEM,JKSF,IWK2,IWK3,
17: * W IWK4,X,Y,Z,W,IWK,
18: * W IFG,LLS,IWK0,IFI,IFL,ISRF,
19: * W FXYZ,R,DSDA0,DSDA1,DSDA2,DSDA3,
20: * W DSDA4,DSDA5,DSDA6,DSDA7,DSDA8,DSDA9,
21: C/ELSE
22: SUBROUTINE SEARCH(IOW, NBANK,NZONE,NSDA,NZDA,NLOST,NDEAD,
23: N NREFS,NGROUP,NGP1,NGP2,NEVENT,NREG,NTIME,
24: N IRAND,NLBZ,NEST,DEPS,NSMAC,NSMIC,MB,NSTAL,NUC,NSPACE,NSUZON,
25: B XXX,YYY,ZZZ,AAA,BBB,CCC,
26: B WWW,IZZ,IGG,TTT,ITT,EEE,XIM,KLSF,
27: B SMAC,SMIC,SGTAL,KMAC,MMAC,KSPI,
28: B LEVL,LZZ,LPOS,LCRS,IBREG,IBSPC,
29: S LSFFL,NFFL,IZFFL,LSSRC,NNXT,IZNXT,
30: S LSDED,LSREF,ISREF,IZREF,NBREF,KSREF,
31: * LSLAT,IZLAT,NXLT,
32: G SDA,KZMAT,KZREG,KZDA,KZAA,NKZAA,
33: * KCELL,IPCEL,MLBZZ,ISUSP,
34: T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD,WKILD,NSURV,
35: T WSURV,NSPLT,WSPLT,NCNTR,WCNTR,
36: W IZT,IPZONE,MEM,JKSF,IWK2,IWK3,
37: W IWK4,X,Y,Z,W,IWK,
38: W IFG,LLS,IWK0,IFI,IFL,ISRF,
39: W FXYZ,R,DSDA0,DSDA1,DSDA2,DSDA3,
40: W DSDA4,DSDA5,DSDA6,DSDA7,DSDA8,DSDA9,
41: * JMEM,NMEMO,KMEMO,MEMC,MEMZ)
42: C/ENDIF
43: C=====
44: C PURPOSE: FIND ZONES TO ENTER. (ALL-ZONE LOGIC)
45: C CALLED IN: ACTION
46: C CALLS: VMNTR0,VMNTR1,VMNTR2
47: C=====
48: implicit real*8(D)
49: include 'INC/_FLAGS'
50: include '../shared/INC/_TASKDT'
51: C
52: C ..... BANK .....
53: C
54: real*8 XXX(NBANK),YYY(NBANK),ZZZ(NBANK)
55: real*8 AAA(NBANK),BBB(NBANK),CCC(NBANK)
56: real WWW(NBANK),XIM(NBANK)
57: real*8 TTT(NBANK)
58: integer ITT(NBANK)
59: integer IZZ(NBANK),IGG(NBANK),LEVL(NBANK),LZZ(NBANK,NEST),
60: & LPOS(NBANK,NEST),LCRS(NBANK,NEST),KLSF(NBANK)
61: integer IBREG(NBANK),IBSPC(NBANK,0:NEST)
62: C
63: C ... SIGMA BANK .....
64: C
65: real EEE(NBANK),SMAC(NBANK,MB,NSMAC),SMIC(NBANK,NUC,NSMIC),

```

```

66: & SGTAL(NBANK,NSTAL)
67: integer KMAC(NBANK,MB,2),MMAC(NBANK,2),KSPI(NBANK,NUC)
68: C
69: C ..... STACK .....
70: C
71: integer LSFFL(NBANK),NFFL(NZONE+1),IZFFL(NBANK),LSSRC(NBANK),
72: & NNXT(NZONE+1),IZNXT(NBANK),LSDED(NBANK),LSREF(NBANK),
73: & ISREF(NBANK),IZREF(NBANK),NBREF(*),LSLAT(NBANK),
74: & IZLAT(NBANK),NXLT(*)
75: C
76: C ..... GEOMETRY .....
77: C
78: real*8 SDA(NSDA)
79: integer KZMAT(NZONE),KZREG(NZONE),KSREF(NZDA),KZDA(2,NZDA),
80: & KZAA(NZONE+1),NKZAA(NZONE),KMEMO(NZONE,NMEMO),
81: & MEMC(NZONE,NMEMO),MEMZ(NZONE),KCELL(NZONE),IPCEL(*),
82: & MLBZZ(NZONE)
83: integer ISUSP(NSUZON,NSPACE)
84: C
85: C ..... VARIANCE REDUCTION .....
86: C
87: real XIMP(NGROUP,NREG),WKIL(NGROUP,NREG),WSRV(NGROUP,NREG)
88: real WTIME(NTIME)
89: real WLLIM
90: C
91: C ..... TALLY (MONITOR) .....
92: C
93: real*8 NKILD(NGROUP,NREG),NSURV(NGROUP,NREG),NSPLT(NGROUP,NREG)
94: real*8 WKILD(NGROUP,NREG),WSURV(NGROUP,NREG),WSPLT(NGROUP,NREG)
95: real*8 WCNTR(NEVENT,2),NCNTR(NEVENT,2)
96: C
97: C ..... WORKING AREA .....
98: C
99: real*8 X(NBANK),Y(NBANK),Z(NBANK),W(NBANK),DSDA0(NBANK),
100: & DSDA1(NBANK),DSDA2(NBANK),DSDA3(NBANK),DSDA4(NBANK),
101: & DSDA5(NBANK),DSDA6(NBANK),DSDA7(NBANK),DSDA8(NBANK),
102: & DSDA9(NBANK)
103: real R(NBANK),FXYZ(NBANK)
104: integer IWK(NBANK),LLS(NBANK),IWK0(NBANK),ISRF(NBANK),
105: & IZT(NZONE+1),IPZONE(NZONE+1),MEM(NZONE+1),
106: & JKSF(NZONE+1),IWK2(NZONE+1),IWK3(NZONE+1),IWK4(NZONE+1)
107: logical IFG(NBANK),IFI(NBANK),IFL(NBANK),JRR,JMEM
108: C
109: CCCCC DATA DEPS,EPS/1.0D-8,0.0/,
110: CCCCC DATA JRR/.FALSE./
111: C
112: CCCCC JVARR = JIMPT + JWWND + JRRLT
113: C
114: C/IF ARGSAVE
115: * return
116: C -----
117: * entry SEARCH( JMEM, NMEMO, KMEMO, MEMC, MEMZ)
118: C/ENDIF
119: JVARR = JIMPT + JWWND + JRRLT
120: NZ1 = NZONE + 1
121: C
122: C
123: if (JVMNT.ne.0) call VMNTR1(4,NNXT(NZ1))
124: C=====
125: C ... GATHER VECTORS .....
126: C * NZK : NUMBER OF ZONES HAVING PARTICLES IN SEARCH STACK.
127: C * JKSF : ZONE # LIST OF ZONES HAVING PARTICLES IN SEARCH STACK.
128: C * IPZONE : ZONE POINTERS IN GATHERED ARRAYS (X,Y,Z ETC.)
129: C=====
130: NZK = 0

```

src/mvp/search.f

```

131:      do 100 K = 1, NZONE
132:      if ( NNXT(K).ne.0 ) then
133:      NZK      = NZK + 1
134:      JKSF(NZK) = K
135:      end if
136:      100 continue
137: C
138:      if ( NZK.gt.20 ) then
139:
140:      IPZONE(1) = 1
141: *VOCL LOOP,SCALAR
142:      do 110 NK = 1, NZK
143:      IPZONE(NK+1) = IPZONE(NK) + NNXT(JKSF(NK))
144:      110 continue
145: *VOCL LOOP,NOVREC
146:      do 120 NK = 1, NZK
147:      MEM(JKSF(NK)) = IPZONE(NK) - 1
148:      120 continue
149: Cs*VOCL LOOP,NOVREC(IWK)
150: Cs      DO 130 I=1, NNXT(NZ1)
151: Cs      MEM(IZNXT(I)) = MEM(IZNXT(I)) + 1
152: Cs      KKK = MEM(IZNXT(I))
153: Cs      IWK(KKK) = LSSRC(I)
154: Cs      120 CONTINUE
155: *VOCL LOOP,SCALAR
156:      do 130 I = 1, NNXT(NZ1)
157:      MEM(IZNXT(I)) = MEM(IZNXT(I)) + 1
158:      IWK(I) = MEM(IZNXT(I))
159:      130 continue
160: *VOCL LOOP,NOVREC
161:      do 140 I = 1, NNXT(NZ1)
162:      IWK(IWK(I)) = LSSRC(I)
163:      140 continue
164:
165:      else
166:
167:      IPZONE(1) = 1
168:      do 150 NK = 1, NZK
169:      IPZONE(NK+1) = IPZONE(NK) + NNXT(JKSF(NK))
170:      150 continue
171:      KKK      = 0
172:      do 170 NK = 1, NZK
173:      do 160 I = 1, NNXT(NZ1)
174:      if ( IZNXT(I).eq.JKSF(NK) ) then
175:      KKK      = KKK + 1
176:      IWK(KKK) = LSSRC(I)
177:      end if
178:      160 continue
179:      170 continue
180:
181:      end if
182: C
183: C ..... GATHER POSITIONS .....
184: C MOVE EACH PARTLES TO THEIR FLIGHT DIRECTIONS BY SMALL DISTANCE (DEPS)
185: C TO PREVENT MISJUDGE. (7/20/1988)
186: C
187:      do 180 I = 1, NNXT(NZ1)
188:      LSSRC(I) = IWK(I)
189:      X(I) = XXX(IWK(I)) + DEPS*AAA(IWK(I))
190:      Y(I) = YYY(IWK(I)) + DEPS*BBB(IWK(I))
191:      Z(I) = ZZZ(IWK(I)) + DEPS*CCC(IWK(I))
192:      W(I) = WWW(IWK(I))
193:      180 continue
194: C=====
195: C      JUDGEMENTS

```

```

196: C ..... * MEM(NK) : CONTROL ARRAYS FOR MEMORANDOM ARRAY TREATMENT
197: C      = 0 : GET CANDIDATES OF NEXT ZONE FROM KMEMO ARRAY.
198: C      > 0 : SET MEM(NK)'S ELEMENT OF KMEMO TO A NEWLY FOUND
199: C      NEXT ZONE#.
200: C=====
201:      do 190 NK = 1, NZK
202:      MEM(NK) = 0
203:      190 continue
204: C
205: C-----
206: C .... * INX : NUMBER OF PARTICLES WHOSE NEXT ZONES ARE SEARCHED.
207: C-----
208:      INX      = NNXT(NZ1)
209:      IFFL      = NFFL(NZ1)
210:      if ( JREFL.ne.0 ) ICREF = NBREF(NREFS+1)
211:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
212:      do 720 M = 1, NZONE
213: C
214: C-----
215: C .... DETERMINE ZONE TO BE SEARCHED .....
216: C-----
217:      LL      = 0
218:      if ( M.le.NMEMO ) then
219:      do 200 NK = 1, NZK
220:      if ( KMEMO(JKSF(NK),M).ne.0 ) then
221:      IZT(NK) = KMEMO(JKSF(NK),M)
222:      LL      = LL + 1
223:      end if
224:      200 continue
225:      end if
226: C
227: C-----
228: C .... NO CANDIDATE OF NEXT ZONE IN KMEMO .....
229: C-----
230:      if ( LL.lt.NZK ) then
231:      do 240 NK = 1, NZK
232:      K      = JKSF(NK)
233:      ICEL      = 0
234:      if ( JLATT.ne.0 ) ICEL = KCELL(K)
235:      if ( M.gt.NMEMO .or. MEM(NK).ne.0 .or. KMEMO(K,M).eq.0 )
236:      &      then
237: C
238: C-----
239: C ..... MEM = 0 : BEGINE SEARCHING OF NEW NEXT-ZONE OTHER THAN
240: C      THOSE IN ARRAY KMEMO. IF A NEW NEXT-ZONE IS FOUND,
241: C      IT IS MEMORIZRED AT KMEMO(JKSF(NK),MEM(NK)).
242: C-----
243: C
244:      if ( MEM(NK).eq.0 ) then
245:      MEM(NK) = M
246:      IZT(NK) = 1
247:      if ( ICEL.ne.0 ) IZT(NK) = IPCEL(ICEL)
248:      else
249:      IZT(NK) = IZT(NK) + 1
250:      end if
251:      MMZ      = NZONE
252:      if ( JLATT.ne.0 ) MMZ = IPCEL(ICEL+1) - 1
253: *VOCL LOOP,SCALAR
254:      do 220 MK = IZT(NK), MMZ
255:      if ( MK.eq.K ) go to 220
256:      do 210 L = 1, NMEMO
257:      if ( MK.eq.KMEMO(K,L) ) go to 220
258:      210 continue
259:      go to 230
260:      220 continue

```

src/mvp/search.f

```

261: C 1988/7/20 CCCCCCCC MK      = NZONE
262: C-----
263: C      .... IF NO NEXT-ZONE WERE FOUND AFTER SEARCHING ALL ZONES EXCEPT
264: C      THAT IN WHICH PARTICLES LIE (JKSF(NK)), THE ZONE JKSF(NK) WOULD
265: C      BE SELECTED AS THE CANDIDATE.  (I HOPE THAT THIS CASE BE RARE.)
266: C-----
267: C
268: C      MK      = K
269: C011691      MK      = MMZ
270: C      230      IZT(NK) = MK
271: C      end if
272: C      240      continue
273: C      end if
274: C
275: C-----
276: C      .... CALCULATE MAXIMUM SURFACE NUMBER OF ZONES .....
277: C-----
278: C      MSF      = 0
279: C      do 250 NK = 1, NZK
280: C      K      = IZT(NK)
281: C      MSF      = MAX(NKZAA(K),MSF)
282: C      250      continue
283: C
284: C-----
285: C      .... SET   FLAGS  ( IFG = T/F : IN/OUT) .....
286: C-----
287: C      do 260 I = 1, INX
288: C      IFG(I) = .true.
289: C      if ( JSIMP.eq.0 ) IFL(I) = .false.
290: C      if ( JREFL.ne.0 ) FXYZ(I) = 1.0E30
291: C      260      continue
292: C
293: C-----
294: C      .... LOOP BY SURFACE NUMBER .....
295: C-----
296: C      do 560 N = 1, MSF
297: C      do 270 I = 1, INX
298: C      IFI(I) = .true.
299: C      IWK(I) = 0
300: C      270      continue
301: C      do 320 NK = 1, NZK
302: C      IWK2(NK) = 0
303: C      K      = IZT(NK)
304: C      KK      = NKZAA(K)
305: C      if ( KK.ge.N ) then
306: C
307: C      .... IWK2 : TYPE OF SURFACE  IWK3 : SIGN * (SDA POINTER)
308: C      IWK4 : LOGICAL TREATMENT FLAG (0/1/2)
309: C
310: C      if ( JSIMP.eq.0 ) LLG = KZDA(2,KZAA(K)+N-1)
311: C      LS      = KZDA(1,KZAA(K)+N-1)
312: C      LSG      = -SIGN(1,LS)
313: C      LSP      = ABS(LS)
314: C      KSF      = INT(ABS(SDA(LSP)))
315: C      IWK2(NK) = KSF
316: C
317: C-----
318: C      .... GATHER SURFACE DATA ....
319: C-----
320: C
321: C      if ( KSF.eq.1 ) then
322: C      do 280 I = IPZONE(NK), IPZONE(NK+1) - 1
323: C      LLS(I) = LSG
324: C      if ( JSIMP.eq.0 ) IWK(I) = LLG
325: C      DSDA0(I) = SDA(LSP+1)

```

```

326: C      DSDA1(I) = SDA(LSP+2)
327: C      DSDA2(I) = SDA(LSP+3)
328: C      DSDA3(I) = SDA(LSP+4)
329: C      DSDA4(I) = SDA(LSP+5)
330: C      280      continue
331: C      else if ( KSF.eq.2 .or. KSF.eq.5 ) then
332: C      do 290 I = IPZONE(NK), IPZONE(NK+1) - 1
333: C      LLS(I) = LSG
334: C      if ( JSIMP.eq.0 ) IWK(I) = LLG
335: C      DSDA0(I) = SDA(LSP+1)
336: C      DSDA1(I) = SDA(LSP+2)
337: C      DSDA2(I) = SDA(LSP+3)
338: C      DSDA3(I) = SDA(LSP+4)
339: C      290      continue
340: C      else if ( KSF.eq.3 ) then
341: C      do 300 I = IPZONE(NK), IPZONE(NK+1) - 1
342: C      LLS(I) = LSG
343: C      if ( JSIMP.eq.0 ) IWK(I) = LLG
344: C      DSDA0(I) = SDA(LSP+1)
345: C      DSDA1(I) = SDA(LSP+2)
346: C      DSDA2(I) = SDA(LSP+3)
347: C      DSDA3(I) = SDA(LSP+4)
348: C      DSDA4(I) = SDA(LSP+5)
349: C      DSDA5(I) = SDA(LSP+6)
350: C      DSDA6(I) = SDA(LSP+7)
351: C      300      continue
352: C      else if ( KSF.eq.4 ) then
353: C      do 310 I = IPZONE(NK), IPZONE(NK+1) - 1
354: C      LLS(I) = LSG
355: C      if ( JSIMP.eq.0 ) IWK(I) = LLG
356: C      DSDA0(I) = SDA(LSP+1)
357: C      DSDA1(I) = SDA(LSP+2)
358: C      DSDA2(I) = SDA(LSP+3)
359: C      DSDA3(I) = SDA(LSP+4)
360: C      DSDA4(I) = SDA(LSP+5)
361: C      DSDA5(I) = SDA(LSP+6)
362: C      DSDA6(I) = SDA(LSP+7)
363: C      DSDA7(I) = SDA(LSP+8)
364: C      DSDA8(I) = SDA(LSP+9)
365: C      DSDA9(I) = SDA(LSP+10)
366: C      310      continue
367: C      end if
368: C      end if
369: C      if ( JREFL.ne.0 ) IWK3(NK) = KSREF(KZAA(IZT(NK))+N-1)
370: C      320      continue
371: C
372: C-----
373: C      .... JUDGE WHETHER PARTICLES ARE IN ZONE IZT(NK) OR NOT.
374: C      ZONES FROM KKK TO KKK2 HAVE SURFACES OF THE SAME TYPE.
375: C-----
376: C
377: C      KKK      = 1
378: C      KSF      = IWK2(KKK)
379: C      330      do 340 NK = KKK, NZK
380: C      if ( IWK2(NK).ne.KSF ) go to 350
381: C      340      continue
382: C
383: C-----
384: C      .... END OF SURFACE LOOP IF ALL OF SURFACE TYPE (IWK2) = 0 ....
385: C-----
386: C      CCC      IF (KSF.EQ.0.AND.KKK.EQ.1) GOTO 282
387: C
388: C      NK      = NZK + 1
389: C      KKK2     = NK
390: C      JRR      = .false.

```

src/mvp/search.f

```

391:         if ( KSF.ne.0.and.JREFL.ne.0 ) then
392:             do 360 NK = KKK, KKK2 - 1
393:                 if ( IWK3(NK).ne.0 ) JRR      = .true.
394:             360         continue
395:         end if
396: C
397: C-----
398: C .... CHECK FOR EACH PARTICLE. IFG(I) = T/F = IN/OUT
399: C-----
400:         go to(370,400,430,460,490) KSF
401:         if ( KSF.ne.0 ) go to 520
402:         go to 530
403: C
404: C ..... SLAB .....
405: CCCCCCCCCCCCC IF(KSF.EQ.1) THEN
406: 370         if ( JRR ) then
407:             do 380 I = IPZONE(KKK), IPZONE(KKK2) - 1
408:                 D1 = DSDA0(I)*X(I) + DSDA1(I)*Y(I) + DSDA2(I)*
409:                 &      Z(I)
410:                 D1 = (D1-DSDA3(I))*(D1-DSDA4(I))*LLS(I)
411:                 IFI(I) = D1.ge.0.0
412:                 D1 = ABS(D1)
413:                 if ( D1.lt.FXYZ(I).and.IFI(I) ) then
414:                     ISRF(I) = N
415:                     FXYZ(I) = D1
416:                 end if
417:             380         continue
418:         else
419:             do 390 I = IPZONE(KKK), IPZONE(KKK2) - 1
420:                 D1 = DSDA0(I)*X(I) + DSDA1(I)*Y(I) + DSDA2(I)*
421:                 &      Z(I)
422:                 IFI(I) = (D1-DSDA3(I))*(D1-DSDA4(I))*LLS(I) .ge.0.0
423:             390         continue
424:         end if
425:         go to 530
426: C
427: C ..... SPHERE .....
428: CCCCCCCCCCCCC ELSE IF(KSF.EQ.2) THEN
429: 400         if ( JRR ) then
430:             do 410 I = IPZONE(KKK), IPZONE(KKK2) - 1
431:                 D1 = ((DSDA0(I)-X(I))**2+(DSDA1(I)-Y(I))**2
432:                 &      +(DSDA2(I)-Z(I))**2-DSDA3(I))*LLS(I)
433:                 IFI(I) = D1.ge.0.0
434:                 D1 = ABS(D1)
435:                 if ( D1.lt.FXYZ(I).and.IFI(I) ) then
436:                     ISRF(I) = N
437:                     FXYZ(I) = D1
438:                 end if
439:             410         continue
440:         else
441:             do 420 I = IPZONE(KKK), IPZONE(KKK2) - 1
442:                 IFI(I) = ((DSDA0(I)-X(I))**2+(DSDA1(I)-Y(I))**2
443:                 &      +(DSDA2(I)-Z(I))**2-DSDA3(I))*LLS(I) .ge.0.0
444:             420         continue
445:         end if
446:         go to 530
447: C
448: C ..... CYLINDER .....
449: CCCCCCCCCCCCC ELSE IF(KSF.EQ.3) THEN
450: 430         if ( JRR ) then
451:             do 440 I = IPZONE(KKK), IPZONE(KKK2) - 1
452:                 DX = X(I) - DSDA0(I)
453:                 DY = Y(I) - DSDA1(I)
454:                 DZ = Z(I) - DSDA2(I)
455:                 D1 = (DX**2+DY**2+DZ**2-DSDA6(I)

```

```

456:         &      -(DSDA3(I)*DX+DSDA4(I)*DY+DSDA5(I)*DZ)**2)*
457:         &      LLS(I)
458:         IFI(I) = D1.ge.0.0
459:         D1 = ABS(D1)
460:         if ( D1.lt.FXYZ(I).and.IFI(I) ) then
461:             ISRF(I) = N
462:             FXYZ(I) = D1
463:         end if
464: 440         continue
465:     else
466:         do 450 I = IPZONE(KKK), IPZONE(KKK2) - 1
467:             DX = X(I) - DSDA0(I)
468:             DY = Y(I) - DSDA1(I)
469:             DZ = Z(I) - DSDA2(I)
470:             IFI(I) = (DX**2+DY**2+DZ**2-DSDA6(I)
471:             &      -(DSDA3(I)*DX+DSDA4(I)*DY+DSDA5(I)*DZ)**2)*
472:             &      LLS(I) .ge.0.0
473:             450         continue
474:         end if
475:         go to 530
476: C
477: C ..... QUADARATIC SURFACE (GENERAL FORM) .....
478: CCCCCCCCCCCCC ELSE IF(KSF.EQ.4) THEN
479: 460         if ( JRR ) then
480:             do 470 I = IPZONE(KKK), IPZONE(KKK2) - 1
481:                 D1 = (X(I)*
482:                 &      (X(I)*DSDA0(I)+Y(I)*DSDA3(I)+Z(I)*DSDA5(I)
483:                 &      +DSDA6(I))+Y(I)*
484:                 &      (Y(I)*DSDA1(I)+Z(I)*DSDA4(I)+DSDA7(I))+Z(I)*
485:                 &      (Z(I)*DSDA2(I)+DSDA8(I))+DSDA9(I))*LLS(I)
486:                 IFI(I) = D1.ge.0.0
487:                 D1 = ABS(D1)
488:                 if ( D1.lt.FXYZ(I).and.IFI(I) ) then
489:                     ISRF(I) = N
490:                     FXYZ(I) = D1
491:                 end if
492:             470         continue
493:         else
494:             do 480 I = IPZONE(KKK), IPZONE(KKK2) - 1
495:                 IFI(I) = (X(I)*
496:                 &      (X(I)*DSDA0(I)+Y(I)*DSDA3(I)+Z(I)*DSDA5(I)
497:                 &      +DSDA6(I))+Y(I)*
498:                 &      (Y(I)*DSDA1(I)+Z(I)*DSDA4(I)+DSDA7(I))+Z(I)*
499:                 &      (Z(I)*DSDA2(I)+DSDA8(I))+DSDA9(I))*LLS(I) .ge.
500:                 &      0.0
501:             480         continue
502:         end if
503:         go to 530
504: C
505: C ..... PLANE (HALF SPACE) .....
506: CCCCCCCCCCCCC ELSE IF(KSF.EQ.5) THEN
507: 490         if ( JRR ) then
508:             do 500 I = IPZONE(KKK), IPZONE(KKK2) - 1
509:                 D1 =
510:                 &      (X(I)*DSDA0(I)+Y(I)*DSDA1(I)+Z(I)*DSDA2(I)
511:                 &      -DSDA3(I))*LLS(I)
512:                 IFI(I) = D1.ge.0.0
513:                 D1 = ABS(D1)
514:                 if ( D1.lt.FXYZ(I).and.IFI(I) ) then
515:                     ISRF(I) = N
516:                     FXYZ(I) = D1
517:                 end if
518:             500         continue
519:         else
520:             do 510 I = IPZONE(KKK), IPZONE(KKK2) - 1

```

src/mvp/search.f

```

521:          IFI(I) =
522:          &      (X(I)*DSDA0(I)+Y(I)*DSDA1(I)+Z(I)*DSDA2(I)
523:          &      -DSDA3(I))*LSS(I) .ge.0.0
524: 510      continue
525:      end if
526:      go to 530
527: C
528: 520      write(IOW,*) ' !! INVALID SURFACE TYPE ', KSF
529:      stop 666
530: C
531: 530      KKK      = KKK2
532:      if ( KKK.le.NZK ) go to 330
533: C
534: C-----
535: C ..... JUDGEMENT .....
536: C-----
537:      if ( JSIMP.eq.1 ) then
538:          do 540 I = 1, INX
539:              IFG(I) = IFG(I) .and. IFI(I)
540:          continue
541:      else
542:          do 550 I = 1, INX
543:              if ( IWK(I).eq.0 ) then
544:                  IFG(I) = IFG(I) .and. IFI(I)
545:              else
546:                  IFL(I) = IFL(I) .or. IFI(I)
547:                  if ( IWK(I).eq.2 ) then
548:                      IFG(I) = IFG(I) .and. IFL(I)
549:                      IFL(I) = .false.
550:                  end if
551:              end if
552:          continue
553:      end if
554: 560      continue
555: C
556: C-----
557: C .... COMPRESS PARTICLE DATA. UPDATE FLIGHT STACK & REFLECTION STACK
558: C      KK IS THE NUMBER OF PARTICLES WHOSE NEXT ZONES ARE DETERMINED.
559: C-----
560: 570      KK      = 0
561:      do 580 I = 1, INX
562:          if ( IFG(I) ) KK      = KK + 1
563: 580      continue
564: C
565:      if ( KK.ne.0 ) then
566:          do 590 NK = 1, NZK
567:              IWK2(NK) = 0
568: 590      continue
569: C
570: C-----
571: C ... SEND PARTICLES WHOSE NEXT ZONES HAVE BEEN FOUND TO FLIGHT STACK,
572: C      IF THE NEXT ZONE IS OUTER VOID, PARTICLES ARE TREATED AS LEAKED.
573: C-----
574:      do 660 NK = 1, NZK
575:          MAT      = KZMAT(IZT(NK))
576:          if ( IZT(NK).eq.JKSF(NK) ) then
577:              if ( MAT.ge.0 ) then
578: C-----
579: C ..... KEEP ON TRACKING .....
580: C-----
581: *VOCL LOOP,NOVREC
582:          do 600 I = IPZONE(NK), IPZONE(NK+1) - 1
583:              if ( IFG(I) ) then
584: C ..... TANGENTIAL PARTICLES
585:                  IWK2(NK) = IWK2(NK) + 1

```

```

586:          IFFL      = IFFL + 1
587:          LSFFL(IFFL) = LSSRC(I)
588:          IZFFL(IFFL) = IZT(NK)
589:          XXX(LSSRC(I)) = X(I)
590:          YYY(LSSRC(I)) = Y(I)
591:          ZZZ(LSSRC(I)) = Z(I)
592:      end if
593: C      ..... IWK0 : OLD ZONE NUMBER (NECESSARY IN SPLITTING)
594: CM      IF(JIMPT.NE.0) IWK0(IFFL) = JKSF(NK)
595: 600      continue
596:      else
597:          do 610 I = IPZONE(NK), IPZONE(NK+1) - 1
598:              IFG(I) = .false.
599:          continue
600:      end if
601: C011691      IF(MAT.ge.0) THEN
602: C
603:      else if ( MAT.ge.0 ) then
604: C-----
605: C ..... KEEP ON TRACKING .....
606: C-----
607: C
608: C
609: *VOCL LOOP,NOVREC
610:      do 620 I = IPZONE(NK), IPZONE(NK+1) - 1
611:          if ( IFG(I) ) then
612:              IWK2(NK) = IWK2(NK) + 1
613:              IFFL      = IFFL + 1
614:              LSFFL(IFFL) = LSSRC(I)
615:              IZFFL(IFFL) = IZT(NK)
616:              if ( JTLLT.ne.0 ) then
617:                  ILV      = LEVL(LSSRC(I))
618:                  if ( ILV.gt.0 ) then
619:                      IBREG(LSSRC(I)) =
620:                      &      ISUSP(KZREG(IZT(NK))),
621:                      &      IBSPC(LSSRC(I),ILV))
622:                  end if
623:              else
624:                  IBREG(LSSRC(I)) = KZREG(IZT(NK))
625:              end if
626: C      ..... IWK0 : OLD ZONE NUMBER (NECESSARY IN SPLITTING)
627: CM      IF(JIMPT.NE.0) IWK0(IFFL) = JKSF(NK)
628:      end if
629: 620      continue
630: C
631: C-----
632: C ..... TERMINATE TRACKING (LEAKAGE) .....
633: C-----
634:      else if ( MAT.eq.-1000 ) then
635:          IDEAD      = NDEAD
636: *VOCL LOOP,NOVREC
637:      do 630 I = IPZONE(NK), IPZONE(NK+1) - 1
638:          if ( IFG(I) ) then
639:              IDEAD      = IDEAD + 1
640:              LSDED(IDEAD) = LSSRC(I)
641:              WCNTR(7,1) = WCNTR(7,1) + W(I)
642:          end if
643: 630      continue
644:          IWK2(NK) = IDEAD - NDEAD
645:          NCNTR(7,1) = NCNTR(7,1) + IWK2(NK)
646:          NCNTR(17,1) = NCNTR(17,1) + IWK2(NK)
647:          NDEAD      = IDEAD
648: C
649: C-----
650: C ..... REFLECTION .....

```


src/mvp/search.f

```

651: C-----
652:       else if ( MAT.lt.-1000 ) then
653:         IW      = KZAA(IZT(NK)) - 1
654:         IRF      = ICREF
655:         do 640 I = IPZONE(NK), IPZONE(NK+1) - 1
656:           if ( IFG(I) ) then
657:             IRF      = IRF + 1
658:             LSREF(IRF) = LSSRC(I)
659:             ISREF(IRF) = KSREF(IW+ISRF(I))
660:             IZREF(IRF) = JKSF(NK)
661:           end if
662:         640 continue
663:         if ( IRF.gt.ICREF ) then
664:           IWK2(NK) = IRF - ICREF
665:           ICREF     = IRF
666:         end if
667: C-----
668: C-----
669: C ..... ESCAPING FROM LATTICE OR CELL .....
670: C-----
671:       else if ( MAT.le.-1.and.MAT.ge.-999 ) then
672:         ILL      = ILAT
673:         do 650 I = IPZONE(NK), IPZONE(NK+1) - 1
674:           if ( IFG(I) ) then
675:             ILL      = ILL + 1
676:             LSLAT(ILL) = LSSRC(I)
677:             IZLAT(ILL) = MLBZZ(IZT(NK))
678:             IZZ(LSSRC(I)) = IZT(NK)
679:           end if
680:         650 continue
681:         if ( ILL.gt.ILAT ) then
682:           IWK2(NK) = ILL - ILAT
683:           NXLT(MLBZZ(IZT(NK))) = NXLT(MLBZZ(IZT(NK)))
684:           &      + (ILL-ILAT)
685:           ILAT      = ILL
686:         end if
687:       end if
688:       660 continue
689: *VOCL LOOP,SCALAR
690:       do 670 NK = 1, NZK
691:         if ( KZMAT(IZT(NK)).ge.0 ) NFFL(IZT(NK)) =
692:           &      NFFL(IZT(NK)) + IWK2(NK)
693:       670 continue
694: C-----
695: C-----
696: C ..... REGISTER THE NEXT-ZONE TO KMEMO ARRAY
697: C-----
698: cccc*VOCL LOOP,NOVREC(KMEMO,MEM)
699: *VOCL LOOP,NOVREC
700:       do 680 NK = 1, NZK
701:         if ( IWK2(NK).ne.0 ) then
702: C880901 ***      IF (MEM(NK).NE.0.AND.MEM(NK).LE.NMEMO) THEN
703: C              KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
704: C              MEM(NK) = MEM(NK) + 1
705: C*****
706:         if ( .not.JMEM ) then
707:           if ( MEM(NK).ne.0.and.MEM(NK).le.NMEMO ) then
708:             KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
709:             MEM(NK) = MEM(NK) + 1
710:           end if
711:         else
712:           if ( MEM(NK).ne.0.and.MEM(NK).le.NMEMO ) then
713:             KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
714:             MEMC(JKSF(NK),MEM(NK)) = MEMC(JKSF(NK),MEM(NK))
715:             &      + IWK2(NK)

```

```

716:             MEMZ(JKSF(NK)) = MEM(NK)
717:             MEM(NK) = MEM(NK) + 1
718:           else if ( MEM(NK).eq.0 ) then
719:             MEMC(JKSF(NK),M) = MEMC(JKSF(NK),M) +
720:             &      IWK2(NK)
721:           end if
722:         end if
723:       end if
724:       680 continue
725: C-----
726: C-----
727: C ..... RE-COUNT THE NUMBER OF NON-ZERO-PARTICLE ZONE. (NZKN - NZK)
728: C ..... CHANGE IWK2 (NUMBER OF PARTICLES WHOSE NEXT-ZONES ARE FOUND)
729: C ..... TO THE NUMBER OF PARTICLES WHOSE NEXT-ZONES ARE NOT FOUND.
730: C-----
731: C-----
732:       NZKN      = 0
733: *VOCL LOOP,NOVREC
734:       do 690 NK = 1, NZK
735:         IWK2(NK) = IPZONE(NK+1) - IPZONE(NK) - IWK2(NK)
736:         if ( IWK2(NK).ne.0 ) then
737:           NZKN      = NZKN + 1
738:           JKSF(NZKN) = JKSF(NK)
739:           IWK2(NZKN) = IWK2(NK)
740:           IZT(NZKN) = IZT(NK)
741:           MEM(NZKN) = MEM(NK)
742:         end if
743:       690 continue
744: C-----
745: C-----
746: C ..... COMPRESS SEARCH STACK .....
747: C-----
748:       KK      = 0
749: *VOCL LOOP,NOVREC
750:       do 700 I = 1, INX
751:         if ( .not.IFG(I) ) then
752:           KK      = KK + 1
753:           LSSRC(KK) = LSSRC(I)
754:           X(KK) = X(I)
755:           Y(KK) = Y(I)
756:           Z(KK) = Z(I)
757:           W(KK) = W(I)
758:         end if
759:       700 continue
760:       INX      = KK
761:       NZK      = NZKN
762:       IPZONE(1) = 1
763: *VOCL LOOP,SCALAR
764:       do 710 NK = 1, NZK
765:         IPZONE(NK+1) = IPZONE(NK) + IWK2(NK)
766:       710 continue
767:       end if
768:       if ( INX.eq.0 ) go to 730
769:       720 continue
770: C-----
771: C-----
772: C ..... COUNT UP OF REFLECTION STACK .....
773: C-----
774:       730 if ( JREFL.ne.0.and.ICREF.gt.NBREF(NREFS+1) ) then
775: *VOCL LOOP,SCALAR
776:       do 740 I = NBREF(NREFS+1) + 1, ICREF
777:         NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
778:       740 continue
779:       NBREF(NREFS+1) = ICREF
780:       end if

```

src/mvp/search.f

```

781: C
782: C =====
783: C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, TREAT THEM
784: C AS LOST PARTICLE.
785: C =====
786: C
787:       if ( INX.ne.0 ) then
788:         do 750 I = 1, INX
789:           LSDED(NDEAD+I) = LSSRC(I)
790:       750 continue
791:         NDEAD = NDEAD + INX
792:         NLOST = NLOST + INX
793:
794: C/#IF PARA(SX* CRAY)
795: *      call MVPSYNC_LOCK(2)
796: C/#ENDIF
797:
798:       write(IOW,*) ' !! (SEARCH) ', INX, ' PARTICLES ARE LOST !! '
799:       write(IOW,7000) (X(I),Y(I),Z(I),AAA(LSSRC(I)),BBB(LSSRC(I)),
800: &      CCC(LSSRC(I)),IZZ(LSSRC(I)),I=1,INX)
801:       7000 format(1X,' X= ',1P,E12.5,' Y= ',E12.5,' Z= ',E12.5,' MU= ',
802: &      E12.5,' ETA=',E12.5,' XI=',E12.5,' ZONE= ',I5)
803:       if ( JLATT.ne.0 ) write(IOW,7020) (LEVL(LSSRC(I)),
804: &      LZZ(LSSRC(I)),LEVL(LSSRC(I)),LPOS(LSSRC(I)),LEVL(LSSRC(I)),
805: &      I=1,INX)
806:       7020 format(1X,' LEVL = ',I3,' LZZ = ',I7,' LPOS = ',I7)
807:
808: C/#IF PARA(SX* CRAY)
809: *      call MVPSYNC_UNLOCK(2)
810: C/#ENDIF
811:
812:       end if
813: C
814: C-----
815: C ..... UPDATING OF ZONE # IN BANK .....
816: C-----
817:       if ( IFFL.gt.NFFL(NZ1) ) then
818:         KFFL = NFFL(NZ1) + 1
819:         do 760 I = KFFL, IFFL
820:           IZZ(LSFFL(I)) = IZFFL(I)
821:       760 continue
822:         NCNTR(17,1) = NCNTR(17,1) + IFFL - NFFL(NZ1)
823:       end if
824: C
825: C =====
826: C = BOUNDARY SPLITTING AND/OR RUSSIAN ROULETTE =
827: C =====
828: C
829: C-----
830: C ..... UPDATING OF PARTICLE NUMBER IN STACKS .....
831: C-----
832:       NFFL(NZ1) = IFFL
833:       if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
834:       do 770 K = 1, NZ1
835:         NNXT(K) = 0
836:       770 continue
837: C
838: C
839: C-----
840: C*** RUSSIAN ROULETTE KILL & SPLITTING BASED ON IMPORTANCE, WEIGHT
841: C-----
842: C
843:       if ( JVARR.ne.0.and.NFFL(NZ1).ge.KFFL ) then
844:         NVARI = NFFL(NZ1) - KFFL + 1
845:         call SPLITB( 0, JNEUT, JPHOT, KFFL, NVARI, IRAND, JMNTR, JIMPT,

```

```

846: &      JWWND, JRRLT, JLATT, JHLAT, JTLLT, JTIME, JWTIM, NBANK, NZONE,
847: &      NGROUP, NGP1, NGP2, NREG, NTIME, NEST, LSFFL, IZFFL, NFFL,
848: &      LSDED, NDEAD, NSMAC, NSMIC,
849: &      XIMP, WKIL, WSRV, WTIME, WLLIM, KZREG,
850: &      NCNTR, WCNTR, NEVENT, XXX, YYY, ZZZ, AAA, BBB, CCC,
851: &      WWW, IZZ, IGG, TTT, ITT, LEVL, LZZ, LPOS, LCRS, XIM,
852: &      KLSF, EEE, IBREG, IBSPC, MB, NUC, NSTAL, SMAC, KMAC,
853: &      MMAC, SMIC, KSPI, SGTAL, NKILD, WKILD, NSURV, WSURV,
854: &      NSPLT, WSPLT, X, Y, Z, W, IFG, IWK )
855:       end if
856: C
857:       return
858:       end

```

src/mvp/sef5n0.f

```

1:      subroutine SEF5N0( IOW,  INUC,  NSS,  EINCD, E,      CX,  ICX,
2:      &                  MCX,  KLIB1, KLIB1, KLIB2, XLIB2, NUC,  NMT,
3:      &                  IRAND, JDEBG,
4:      &                  IWK8,  IWK9,  IWK10, WK3,   WK4,   WK5,R
5:      &                  )
6: C=====
7: C  PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA
8: C    Almost similar to sef5n2, but all particles have the same target
9: C    nuclei and the same incident enrgy.
10: C    and energy variable E is given as real*8.
11: C
12: C    For '#FISSION' type of source sampling function used in initial
13: C    source distribution sampling in eigenvalue problem.
14: C
15: C    called from: NUCFIS
16: C    CALL      : RANU2 BSEDEC3 BSVDEC
17: C-----
18: C    MT = 18,
19: C
20: C    INUC  : NUCLIDE,
21: C    nss   : NUMBER OF PARTICLES
22: C    EINCD : INCIDENT ENERGY
23: C    E(I)  : OUTGOING ENERGY (double precision)
24: C-----
25: C
26: C    implicit real*8(A-H,O-Z)
27: C
28: C    integer JDEBG(*)
29: C
30: C    real*8  E(NSS)
31: C    real    EINCD
32: C
33: C    parameter( PIH  = 3.141592653589793200D0*0.5D0 )
34: C
35: C    ..... CROSS SECTION DATA IN CX ARRAY
36: C
37: C##<2007/03/14:PN3:
38: C##    real    CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,2)
39: C##    real    CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
40: C##>
41: C    integer ICX(MCX)
42: C##<2007/03/14:PN3:
43: C##    integer KLIB1(NUC,21), KLIB2(NUC,NMT,13)
44: C##    integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
45: C##>
46: C
47: C    ..... WORKING AREA
48: C
49: C    integer IWK8(NSS), IWK9(NSS), IWK10(NSS)
50: C    real    WK3(NSS), WK4(NSS), WK5(NSS), R(4*NSS)
51: C
52: C    parameter( MT  = 18 )
53: C
54: C    parameter (SMALL=1.0E-35)
55: C
56: C**** DETERMINE THE SUBSECTION USED
57: C
58: C    ..... MAKE TEMPORARY STACK OF PARTICLE HAVING SUBSECTION > 1
59: C
60: C    NN      = 0
61: C
62: C    nkf5 : no. of partial distribution (subsection)
63: C    lstf5 : starting address of energy distribution data of each MT #
64: C    plus 2 ( -> position of lstf5s(1,mt) )
65: C

```

```

66:      NKF5  = KLIB2(INUC,MT,8)
67:      LSTF5  = KLIB2(INUC,MT,13)
68:      NEF5   = KLIB2(INUC,MT,9)
69: C
70: C    ... CX(I8) == EPROB(1,MT)
71: C    ... energy bin of subsection
72: C    subsection distribution is given for each EPROB bin.
73: C
74: C    I8      = LSTF5 + NKF5
75: C
76: C    ..... THE CASE WHERE MANY SUBSECTIONS ARE GIVEN
77: C
78: C    if ( NKF5.ge.2 ) then
79: C
80: C    ..... SELECTION OF USED SUBSECTION TABLE ----> IWK10
81: C
82: C          t      nt      x      it      nl
83: C          call BSVDEC( CX(I8), NEF5, EINCD, ITT, 1 )
84: C
85: C          call RANU2( IRAND, R, NSS*3, ICON )
86: C          NN2      = NSS*2
87: C
88: C*VOCL LOOP,NOVREC
89: C    do 100 I = 1, NSS
90: C        L2      = I8 + ITT
91: C        L1      = L2 - 1
92: C
93: C    ..... LINEAR INTERPOLATION
94: C
95: C        X1      = (CX(L1)-EINCD) / (CX(L1)-CX(L2))
96: C
97: C/IF ROUNDOFF(NEAREST)
98: C    IT      = MIN(1,INT(R(I)+X1)) + ITT
99: C/ELSE
100: C    *      R1      = R(I) + X1
101: C    *      IT      = ITT + R1
102: C/ENDIF
103: C
104: C
105: C    ..... BMC SAMPLING
106: C
107: C    L1      = I8 + NEF5*2 + 3*NKF5*(ITT-1) - 1
108: C
109: C/IF ROUNDOFF(NEAREST)
110: C    L2      = MIN(INT(NKF5*R(NSS+I))+1,NKF5)
111: C    L3      = L1 + L2
112: C    L4      = 2*L2
113: C    L4      = MIN(L4,INT(L4+R(NN2+I)-CX(L3))) + L1 + NKF5
114: C/ELSE
115: C    *      L2      = NKF5*R(NSS+I) + 1
116: C    *      L3      = L1 + L2
117: C    *      R2      = R(NN2+I) - CX(L3)
118: C    *      L4      = L1 + NKF5 + 2*L2 + R2
119: C/ENDIF
120: C    IWK10(I)  = ICX(L4)
121: C    100      continue
122: C
123: C    end if
124: C
125: C
126: C    .... DETERMINE TABLE USED ----> IWK10(I) = INTERVAL
127: C
128: C    IWK8 <- POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
129: C    ( CX(IWK8(I)) == EENG(1) )
130: C    IWK9 <- NUMBER OF INCIDENT ENERGY (NEF5S)

```

src/mvp/sef5n0.f

```

131: C      IWK10: SUBSECTION NUMBER USED --> interval
132: C      wk3  <- INCIDENT NEUTRON ENERGY
133: C
134: C      do 110 I = 1, NSS
135: C
136: C          ... lstf5s : position of LF data of selected subsection )
137: C
138: C          if ( NKf5.ge.2 ) then
139: C              LSTF5S = ICX(LSTF5+IWK10(I)-1)
140: C          else
141: C              LSTF5S = ICX(LSTF5+NKF5-1)
142: C          end if
143: C
144: C          ... CX(IWK8(I)) == EENG(1) ...
145: C
146: C          IWK8(I) = LSTF5S + 2
147: C
148: C          ... IWK9(I) == NEF5S ...
149: C
150: C          IWK9(I) = ICX(LSTF5S+1)
151: C
152: C          WK3(I) = EINCD
153: C      110 continue
154: C
155: C          t  nt  mw  ls  x  it  ni
156: C      call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NSS )
157: C
158: C      C**** SAMPLING OF OUTGOING ENERGY FROM DATA IN SUBSECTION IWK12(I)
159: C      MAKE TEMPORARY STACK, THESE ARRAYS ARE CHANGED DURING CALCULATION
160: C          IWK8(NN) : TEMPORARY STACK FOR E' > E-U
161: C          IWK9(NN) : LFF5S
162: C          IWK10(NN) :
163: C          WK3  (NN) :
164: C          WK4  (NN) :
165: C          WK5  (NN) :
166: C
167: C      call RANU2( IRAND, R, NSS*4, ICON )
168: C
169: C          N2 = NSS*2
170: C          N3 = NSS*3
171: C
172: C          NN = 0
173: C
174: C      *VOCL LOOP,NOVREC
175: C      do 120 I = 1, NSS
176: C
177: C          LSTF5S = IWK8(I) - 2
178: C          L2 = IWK8(I) + IWK10(I)
179: C          L1 = L2 - 1
180: C          INTE = ICX(L2+IWK9(I))
181: C          INTE0 = ICX(L2+IWK9(I))
182: C          INTE1 = INTE0 / 10
183: C          INTE = INTE0 - 10*INTE1
184: C
185: C          if ( INTE.eq.5 .or. INTE.eq.3 ) then
186: C              LOG INTERPOLATION
187: C              X1 = LOG(CX(L1)/WK3(I)) / LOG(CX(L1)/CX(L2))
188: C          else
189: C              LINEAR INTERPOLATION
190: C              X1 = (CX(L1)-WK3(I)) / (CX(L1)-CX(L2))
191: C          end if
192: C
193: C          LFF5S = ICX(LSTF5S)
194: C          L1 = IWK8(I) + IWK9(I)*2
195: C

```

```

196: C==== DEPEND ON LFF5S-VALUE
197: C
198: C
199: C      C.... TABULATED PROBABILITY (LF=1) USING 3 OR 4 RANDOM NUMBERS
200: C
201: C
202: C
203: C          if ( LFF5S.eq.1 ) then
204: C
205: C      C/#IF ROUNDOFF(NEAREST)
206: C          IT = MIN(1,INT(R(I)+X1)) + IWK10(I)
207: C      C/#ELSE
208: C          R1 = R(I) + X1
209: C          IT = IWK10(I) + R1
210: C      C/#ENDIF
211: C
212: C          NBINE1 = KLIB1(INUC,21)
213: C          if ( NBINE1.gt.1 ) then
214: C
215: C              LLE = NBINE1*2*(IT-1) + L1
216: C              LLE2 = NBINE1*2*(IWK10(I)) + L1
217: C              LLE1 = LLE2 - NBINE1*2
218: C              NEP = NBINE1 - 1
219: C              NEP1 = NEP
220: C              NEP2 = NEP
221: C      C/#IF ROUNDOFF(NEAREST)
222: C          L3 =
223: C              &
224: C              MIN(INT(KLIB1(INUC,8)*R(NSS+I))+1,KLIB1(INUC,8))
225: C      C/#ELSE
226: C          L3 = KLIB1(INUC,8)*R(NSS+I) + 1
227: C      C/#ENDIF
228: C
229: C          L3 = L3 - 1 + LLE
230: C          L4 = L3 + NBINE1
231: C          else
232: C      C..... BMC SAMPLING USING 4 RANDOM NUMBERS
233: C
234: C          LSTF5E = ICX(L1+IT-1)
235: C          NEP = ICX(LSTF5E)
236: C
237: C      C/#IF ROUNDOFF(NEAREST)
238: C          LL2 = MIN(INT(NEP*R(NSS+I))+1,NEP)
239: C          LL3 = LSTF5E + LL2
240: C          LL4 = 2*LL2
241: C          LL4 = LSTF5E + NEP +
242: C              &
243: C              MIN(LL4,INT(LL4+R(N3+I)-CX(LL3)))
244: C      C/#ELSE
245: C          LL2 = NEP*R(NSS+I) + 1
246: C          LL3 = LSTF5E + LL2
247: C          R3 = R(N3+I) - CX(LL3)
248: C          LL4 = LSTF5E + NEP + 2*LL2 + R3
249: C      C/#ENDIF
250: C
251: C          LLL = LSTF5E + 3*NEP
252: C          L3 = LLL + ICX(LL4)
253: C          L4 = L3 + NEP + 1
254: C
255: C          LLE = LLL + 1
256: C          if ( IT.eq.IWK10(I) ) then
257: C              NEP1 = NEP
258: C              LLE1 = LLE
259: C              LSTF5T = ICX(L1+IT)
260: C              NEP2 = ICX(LSTF5T)
261: C              LLE2 = LSTF5T + 3*NEP2 + 1

```

src/mvp/sef5n0.f

```

261:         else
262:             NEP2 = NEP
263:             LLE2 = LLE
264:             LSTF5T = ICX(L1+IT-2)
265:             NEP1 = ICX(LSTF5T)
266:             LLE1 = LSTF5T + 3*NEP1 + 1
267:         end if
268:     end if
269:     E0 = CX(L3)
270:     E1 = CX(L3+1)
271:     E0 = CX(L4)
272:     E1 = CX(L4+1)
273: C
274:     RRR = R(N2+I)*(P0+P1)
275:     E(I) = E1
276:     & + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
277: C
278:     if ( INTE1.eq.2 ) then
279: C ... linear interpolation is assumed
280:         EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
281:         E(I) = E(I) * EMAX / CX(LLE)
282:     else if ( INTE1.eq.1 ) then
283: C ... linear interpolation is assumed
284:         EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
285:         EMIN = CX(LLE1+NEP1)
286:         & + (CX(LLE2+NEP2)-CX(LLE1+NEP1))*X1
287:         E(I) = EMIN + (EMAX-EMIN)*
288:         & (E(I)-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
289:     end if
290: C
291:     end if
292: C
293: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
294: C
295:     if ( LFF5S.eq.5 ) then
296:         L5 = L1 + IWK10(I)
297:         THETA5 = CX(L5)*X1 + CX(L5-1)*(1.-X1)
298:         LL = L1 + IWK9(I)
299:         NBINE = KLIB1(INUC,8)
300: C
301: C..... equiprobability bin ....
302: C
303:     if ( NBINE.gt.0 ) then
304:         NBINE1 = KLIB1(INUC,21)
305:         U5 = CX(LL+NBINE1*2)
306: C
307: C/#IF ROUNDOFF(NEAREST)
308:         L6 = MIN(NBINE-1,INT(NBINE*R(I))) + LL
309: C/#ELSE
310:         R23 = NBINE*R(I)
311:         L6 = R23 + LL
312: C/#ENDIF
313: C
314:         L7 = L6 + NBINE1
315:     else
316: C
317: C..... BMC SAMPLING USING 2 RANDOM NUMBERS
318: C
319:         NBINE = ICX(LL)
320:         NBINE1 = 1
321: C
322: C/#IF ROUNDOFF(NEAREST)
323:         LL5 = MIN(INT(NBINE*R(I))+1,NBINE)
324:         LL6 = LL + LL5
325:         LL7 = 2*LL5

```

```

326:         LL7 = LL + NBINE + MIN(LL7,INT(LL7+R(NSS+I)-CX(LL6)))
327: C/#ELSE
328:         LL5 = NBINE*R(I) + 1
329:         LL6 = LL + LL5
330:         R1 = R(NSS+I) - CX(LL6)
331:         LL7 = LL + NBINE + 2*LL5 + R1
332: C/#ENDIF
333:         L6 = LL + 3*NBINE + ICX(LL7)
334:         L7 = L6 + NBINE + 1
335:         U5 = CX(LL+5*NBINE+3)
336:     end if
337: C
338:     EHI5 = EINCD - U5
339: C
340:     E0 = CX(L6)
341:     E1 = CX(L6+1)
342:     P0 = CX(L7)
343:     P1 = CX(L7+1)
344: C
345:     RRR = R(N2+I)*(P0+P1)
346:     X2 = E1
347:     & + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
348:     & E(I) = X2*THETA5
349: C
350:     if ( E(I).gt.EHI5 ) then
351:         NN = NN + 1
352:         IWK8(NN) = I
353:         IWK9(NN) = LFF5S
354:         IWK10(NN) = LL
355:         WK3(NN) = EHI5
356:         WK4(NN) = THETA5
357:     end if
358:     end if
359: C
360: C.... SIMPLE FISSION SPECTRUN (LF=7) USING 3 RANDOM NUMBERS
361: C
362:     if ( LFF5S.eq.7 ) then
363:         L8 = L1 + IWK10(I)
364:         THETA7 = CX(L8)*X1 + CX(L8-1)*(1.-X1)
365:         U7 = CX(L1+IWK9(I))
366:         EHI7 = EINCD - U7
367:         COSINE = COS(PIH*R(I))
368:         COS2 = COSINE*COSINE
369: C
370:         E(I) = -THETA7*(LOG(R(NSS+I))+LOG(R(N2+I))*COS2)
371: C
372:     if ( E(I).gt.EHI7 ) then
373:         NN = NN + 1
374:         IWK8(NN) = I
375:         IWK9(NN) = LFF5S
376:         WK3(NN) = EHI7
377:         WK4(NN) = THETA7
378:     end if
379:     end if
380: C
381: C.... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
382:     if ( LFF5S.eq.9 ) then
383:         L9 = L1 + IWK10(I)
384:         THETA9 = CX(L9)*X1 + CX(L9-1)*(1.-X1)
385:         U9 = CX(L1+IWK9(I))
386:         EHI9 = EINCD - U9
387: C
388:         E(I) = -THETA9*LOG(R(NSS+I)*R(I))
389: C
390:     if ( E(I).gt.EHI9 ) then

```

src/mvp/sef5n0.f

```

391:      NN      = NN + 1
392:      IWK8(NN) = I
393:      IWK9(NN) = LFF5S
394:      WK3(NN) = EHI9
395:      WK4(NN) = THETA9
396:      end if
397:      end if
398: C
399: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
400:      if ( LFF5S.eq.11 ) then
401:          L10 = L1 + IWK10(I)
402:          L11 = L10 + IWK9(I)
403:          A = CX(L10)*X1 + CX(L10-1)*(1.-X1)
404:          B = CX(L11)*X1 + CX(L11-1)*(1.-X1)
405:          U11 = CX(L1+IWK9(I)*2)
406:          EHI11 = EINGD - U11
407:          COSI1 = COS(PIH*R(I))
408:          COSI2 = COSI*COSI
409:          V2 = -A*(LOG(R(NSS+I))+LOG(R(N2+I))*COSI2)
410:          XMU = 2.*R(N3+I) - 1.
411: C
412:          E(I) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
413: C
414:          if ( E(I).gt.EHI11 ) then
415:              NN = NN + 1
416:              IWK8(NN) = I
417:              IWK9(NN) = LFF5S
418:              WK3(NN) = EHI11
419:              WK4(NN) = A
420:              WK5(NN) = B
421:          end if
422:          end if
423:      120 continue
424: C
425: C
426: C ===== utilize gathered parameters after second stage of energy
427: C rejection.
428: C
429: C
430:      if ( NN.gt.0 ) then
431:          NTEST = 1
432: C
433: C ... rejection loop
434: C
435: C Accepted when E .le. wk3(i)
436: C
437: C
438:      130 continue
439: C
440:      NNN = NN
441: C
442:      call RANU2( IRAND, R, NNN*4, ICON )
443: C
444:      N2 = NNN*2
445:      N3 = NNN*3
446: C
447:      NN = 0
448: C
449: *VOCL LOOP,NOVREC
450:      do 140 I = 1, NNN
451:          LFF5S = IWK9(I)
452: C
453: C===== DEPEND ON LFF5S-VALUE
454: C
455: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS

```

```

456: C
457:      if ( LFF5S.eq.5 ) then
458:          NBINE = KLIB1(INUC,8)
459:          if ( NBINE.gt.0 ) then
460:              NBINE1 = KLIB1(INUC,21)
461:          else
462:              NBINE1 = 1
463:          end if
464:          if ( NBINE1.gt.1 ) then
465: C/#IF ROUNDOFF(NEAREST)
466:              L6 = MIN(NBINE-1,INT(NBINE*R(I))) + IWK10(I)
467: C/#ELSE
468:              R23 = NBINE*R(I)
469:              L6 = R23 + IWK10(I)
470: C/#ENDIF
471:              L7 = L6 + NBINE1
472:          else
473: C/#IF ROUNDOFF(NEAREST)
474:              LL5 = MIN(INT(NBINE*R(I))+1,NBINE)
475:              LL6 = IWK10(I) + LL5
476:              LL7 = 2*LL5
477:              LL7 = MIN(LL7,INT(LL7+R(NNN+I)-CX(LL6))) +
478:                  IWK10(I) + NBINE
479:              &
480: C/#ELSE
481:              LL5 = NBINE*R(I) + 1
482:              LL6 = IWK10(I) + LL5
483:              R3 = R(NNN+I) - CX(LL6)
484:              LL7 = IWK10(I) + NBINE + 2*LL5 + R3
485: C/#ENDIF
486:              L6 = IWK10(I) + 3*NBINE + ICX(LL7)
487:              L7 = L6 + NBINE + 1
488:          end if
489:          E0 = CX(L6)
490:          E1 = CX(L6+1)
491:          P0 = CX(L7)
492:          P1 = CX(L7+1)
493: C
494: C
495: C
496:          RRR = R(N2+I)*(P0+P1)
497:          X2 = E1
498:          &
499:          E(IWK8(I)) = X2*WK4(I)
500:          end if
501: C
502: C..... SIMPLE FISSION SPECTRUN (LF=7) USING THREE RANDOM NUMBERS
503: C
504:      if ( LFF5S.eq.7 ) then
505:          COSINE = COS(PIH*R(I))
506:          COS2 = COSINE*COSINE
507: C
508:          E(IWK8(I)) = -WK4(I)*(LOG(R(NNN+I))+LOG(R(N2+I))*COS2)
509:          end if
510: C
511: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
512: C
513:      if ( LFF5S.eq.9 ) then
514: C
515:          E(IWK8(I)) = -WK4(I)*LOG(R(I)*R(NNN+I))
516:          end if
517: C
518: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
519: C
520:      if ( LFF5S.eq.11 ) then

```

src/mvp/sef5n0.f

```
521:          A      = WK4(I)
522:          B      = WK5(I)
523:          COSI    = COS(PIH*R(I))
524:          COSI2   = COSI*COSI
525:          V2      = -A*(LOG(R(NNN+I))+LOG(R(N2+I))*COSI2)
526:          XMU     = 2.*R(N3+I) - 1.
527: C
528:          E(IWK8(I)) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
529: C
530:          end if
531: C
532:          if ( E(IWK8(I)).gt.WK3(I) ) then
533:              NN      = NN + 1
534:              IWK8(NN) = IWK8(I)
535:              IWK9(NN) = IWK9(I)
536:              WK3(NN) = WK3(I)
537:              WK4(NN) = WK4(I)
538:              if ( LFF5S.eq.11 ) WK5(NN) = WK5(I)
539:              if ( LFF5S.eq.5 ) IWK10(NN) = IWK10(I)
540:          end if
541: C
542: 140      continue
543: C
544:          if ( NN.gt.0 ) then
545:              NTEST = NTEST + 1
546:              if ( NTEST.le.300 ) then
547:                  go to 130
548:              else
549:                  do 150 I = 1, NN
550:                      E(IWK8(I)) = WK3(I)
551: 150              continue
552:                  end if
553:              end if
554:          end if
555: C
556:          return
557:          end
```

src/mvp/sef5n2.f

```

1:      subroutine SEF5N2( IOW,  INUC,  NSEF5, EIN,  EEE,  CX,  ICX,
2:      &                  MCX,  KLIB1, XLIB1, KLIB2, XLIB2, NBANK, NUC,
3:      &                  NMT,  IRAND, JDEBG,
4:      &                  IWK8, IWK9, IWK10, IWK11,
5:      &                  IWK12, IWK13, WK3,  WK4,  WK5,  R )
6:      C=====
7:      C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA
8:      C      MT = 18,
9:      C
10:     C.  INUC(I) : COLLIDING NUCLIDE,
11:     C.  NSEF5 : NUMBER OF PARTICLES
12:     C.  EIN(I) : INCIDENT ENERGY
13:     C.  -----> EEE(I) : OUTGOING ENERGY
14:     C-----
15:     C CALLED IN: FISGEN, FISSRC, PANLGF
16:     C CALL      : RANU2, BSDEC3
17:     C-----
18:     C
19:     C      implicit real*8(A-H,O-Z)
20:     C
21:     C      parameter( PIH = 3.141592653589793200D0*0.5D0 )
22:     C
23:     C      integer JDEBG(*)
24:     C
25:     C**** CROSS SECTION DATA IN CX ARRAY
26:     C
27:     C##<2007/03/14:PN3:
28:     C##      real  CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,2)
29:     C##      real  CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
30:     C##>
31:     C##      integer ICX(MCX)
32:     C##<2007/03/14:PN3:
33:     C##      integer KLIB1(NUC,21), KLIB2(NUC,NMT,13)
34:     C##      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
35:     C##>
36:     C
37:     C      integer INUC(NBANK)
38:     C      real  EIN(NBANK)
39:     C      real  EEE(NBANK)
40:     C
41:     C**** WORKING AREA
42:     C
43:     C      integer IWK8(NBANK), IWK9(NBANK), IWK10(NBANK), IWK11(NBANK),
44:     C      &      IWK12(NBANK), IWK13(NBANK)
45:     C      real  WK3(NBANK), WK4(NBANK), WK5(NBANK)
46:     C      real  R(4*NBANK)
47:     C
48:     C##<2007/03/14:PN3:
49:     C##      parameter (SMALL=1.0E-35)
50:     C##      parameter (SMALL=1.0D-35)
51:     C##>
52:     C
53:     C-----
54:     C
55:     C**** DETERMINE THE SUBSECTION USED
56:     C.... MAKE TEMPORARY STACK OF PARTICLE HAVING SUBSECTION > 1
57:     C      IWK11(J)
58:     C      IWK12(I) : SUBSECTION NO. USED FOR I-TH PARTICLE
59:     C      NN      = 0
60:     C      do 100 I = 1, NSEF5
61:     C          IWK12(I) = KLIB2(INUC(I),18,8)
62:     C          IWK13(I) = KLIB2(INUC(I),18,13)
63:     C          if ( IWK12(I).ge.2 ) then
64:     C              NN      = NN + 1
65:     C          IWK11(NN) = I

```

```

66:          IWK8(NN)      = IWK13(I) + IWK12(I)
67:          IWK9(NN)      = KLIB2(INUC(I),18,9)
68:          WK3(NN) = EIN(I)
69:      end if
70:      100 continue
71:  C
72:  C**** THE CASE WHERE MANY SUBSECTIONS ARE GIVEN
73:  C
74:      if ( NN.gt.0 ) then
75:  C
76:  C.... SELECTION OF USED TABLE ----> IWK10
77:  C      IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
78:  C      IWK9 : NUMBER OF INCIDENT ENERGY
79:  C      WK3 : INCIDENT NEUTRON ENERGY
80:  C      WK4 : WORK AREA IN BSDEC3
81:  C      IWK10: Table position as results
82:  C
83:  C          T,  NT,  MW,  LS,  X,  IT,  N1
84:  C      call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NN )
85:  C
86:  C      call RANU2( IRAND, R, NN*3, ICON )
87:  C      NN2      = NN*2
88:  C
89:  C*VOCL LOOP,NOVREC
90:  C      do 110 J = 1, NN
91:  C          I      = IWK11(J)
92:  C          L2      = IWK8(J) + IWK10(J)
93:  C          L1      = L2 - 1
94:  C          C..... LINEAR INTERPOLATION
95:  C          X1      = (CX(L1)-WK3(J)) / (CX(L1)-CX(L2))
96:  C
97:  C      C/#IF ROUNDOFF(NEAREST)
98:  C          IT      = MIN(1,INT(R(J)+X1)) + IWK10(J)
99:  C      C/#ELSE
100:  C          *      R1      = R(J) + X1
101:  C          *      IT      = IWK10(J) + R1
102:  C      C/#ENDIF
103:  C
104:  C          C..... BMC SAMPLING
105:  C          NKF5      = IWK12(I)
106:  C          L1      = IWK8(J) + IWK9(J)*2 + 3*NKF5*(IT-1) - 1
107:  C
108:  C      C/#IF ROUNDOFF(NEAREST)
109:  C          L2      = MIN(INT(NKF5*R(NN+J))+1,NKF5)
110:  C          L3      = L1 + L2
111:  C          L4      = 2*L2
112:  C          L4      = MIN(L4,INT(L4+R(NN2+J)-CX(L3))) + L1 + NKF5
113:  C      C/#ELSE
114:  C          *      L2      = NKF5*R(NN+J) + 1
115:  C          *      L3      = L1 + L2
116:  C          *      R2      = R(NN2+J) - CX(L3)
117:  C          *      L4      = L1 + NKF5 + 2*L2 + R2
118:  C      C/#ENDIF
119:  C          IWK12(I)      = ICX(L4)
120:  C      110 continue
121:  C
122:  C      end if
123:  C
124:  C
125:  C.... DETERMINE TABLE USED ----> IWK10(I) = INTERVAL
126:  C      IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
127:  C      IWK9 : NUMBER OF INCIDENT ENERGY
128:  C      IWK12: SUBSECTION NUMBER USED
129:  C      EIN : INCIDENT NEUTRON ENERGY
130:  C

```


src/mvp/sef5n2.f

```

131:      do 120 I = 1, NSEF5
132:         IWK13(I) = ICX(IWK13(I)+IWK12(I)-1)
133:         IWK8(I) = IWK13(I) + 2
134:         IWK9(I) = ICX(IWK13(I)+1)
135:      120 continue
136: C
137:      call BSDEC3( CX, IWK9, WK4, IWK8, EIN, IWK10, NSEF5 )
138: C
139: C**** SAMPLING OF OUTGOING ENERGY FROM DATA IN SUBSECTION IWK12(I)
140: C      MAKE TEMPORARY STACK, THESE ARRAYS ARE CHANGED DURING CALCULATION
141: C      IWK8(NN) : TEMPORARY STACK FOR E' > E-U
142: C      IWK9(NN) : LFF5S
143: C      IWK10(NN) :
144: C      IWK11(NN) :
145: C      IWK12(NN) :
146: C      WK3 (NN) :
147: C      WK4 (NN) :
148: C      WK5 (NN) :
149: C
150:      call RAND2( IRAND, R, NSEF5*4, ICON )
151: C
152:      N2 = NSEF5*2
153:      N3 = NSEF5*3
154: C
155:      NN = 0
156: C
157: *VOCL LOOP,NOVREC
158:      do 130 I = 1, NSEF5
159:         L2 = IWK8(I) + IWK10(I)
160:         L1 = L2 - 1
161:         INTE0 = ICX(L2+IWK9(I))
162:         INTE1 = INTE0 / 10
163:         INTE = INTE0 - 10*INTE1
164:         if ( INTE.eq.5 .or. INTE.eq.3 ) then
165: C..... LOG INTERPOLATION
166:            X1 = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
167:         else
168: C..... LINEAR INTERPOLATION
169:            X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
170:         end if
171: C
172:         LFF5S = ICX(IWK13(I))
173:         L1 = IWK8(I) + IWK9(I)*2
174: C
175: C==== DEPEND ON LFF5S-VALUE
176: C
177: C.... TABULATED PROBABILITY (LF=1) USING 3 OR 4 RANDOM NUMBERS
178:      if ( LFF5S.eq.1 ) then
179:
180: C/#IF ROUNDOFF(NEAREST)
181:         IT = MIN(1,INT(R(I)+X1)) + IWK10(I)
182: C/#ELSE
183:         R1 = R(I) + X1
184:         IT = IWK10(I) + R1
185: C/#ENDIF
186:
187:         NBINE1 = KLIB1(INUC(I),21)
188:         if ( NBINE1.gt.1 ) then
189:
190:             LLE = NBINE1*2*(IT-1) + L1
191:             LLE2 = NBINE1*2*(IWK10(I)) + L1
192:             LLE1 = LLE2 - NBINE1*2
193:             NEP = NBINE1 - 1
194:             NEP1 = NEP
195:             NEP2 = NEP

```

```

196: C/#IF ROUNDOFF(NEAREST)
197:         LKL1 = KLIB1(INUC(I),8)
198:         L3 = MIN(INT(LKL1*R(NSEF5+I))+1,LKL1)
199: C/#ELSE
200:         L3 = KLIB1(INUC(I),8)*R(NSEF5+I) + 1
201: C/#ENDIF
202:
203:         L3 = L3 - 1 + LLE
204:         L4 = L3 + NBINE1
205:         else
206: C..... BMC SAMPLING USING 4 RANDOM NUMBERS
207:         LSTF5E = ICX(L1+IT-1)
208:         NEP = ICX(LSTF5E)
209:
210: C/#IF ROUNDOFF(NEAREST)
211:         LL2 = MIN(INT(NEP*R(NSEF5+I))+1,NEP)
212:         LL3 = LSTF5E + LL2
213:         LL4 = 2*LL2
214:         LL4 = LSTF5E + NEP +
215:             & MIN(LL4,INT(LL4+R(N3+I)-CX(LL3)))
216: C/#ELSE
217:         LL2 = NEP*R(NSEF5+I) + 1
218:         LL3 = LSTF5E + LL2
219:         R2 = R(N3+I) - CX(LL3)
220:         LL4 = LSTF5E + NEP + 2*LL2 + R2
221: C/#ENDIF
222:
223:         LLL = LSTF5E + 3*NEP
224:         L3 = LLL + ICX(LL4)
225:         L4 = L3 + NEP + 1
226: C
227:         LLE = LLL + 1
228:         if ( IT.eq.IWK10(I) ) then
229:             NEP1 = NEP
230:             LLE1 = LLE
231:             LSTF5T = ICX(L1+IT)
232:             NEP2 = ICX(LSTF5T)
233:             LLE2 = LSTF5T + 3*NEP2 + 1
234:         else
235:             NEP2 = NEP
236:             LLE2 = LLE
237:             LSTF5T = ICX(L1+IT-2)
238:             NEP1 = ICX(LSTF5T)
239:             LLE1 = LSTF5T + 3*NEP1 + 1
240:         end if
241:         end if
242:         E0 = CX(L3)
243:         E1 = CX(L3+1)
244:         P0 = CX(L4)
245:         P1 = CX(L4+1)
246: C
247:         RRR = R(N2+I)*(P0+P1)
248:         EEE(I) = E1
249:         & + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
250: C
251:         if ( INTE1.eq.2 ) then
252: C ... linear interpolation is assumed
253:             EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
254:             EEE(I) = EEE(I) * EMAX / CX(LLE)
255:         else if ( INTE1.eq.1 ) then
256: C ... linear interpolation is assumed
257:             EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
258:             EMIN = CX(LLE1+NEP1)
259:             & + (CX(LLE2+NEP2)-CX(LLE1+NEP1))*X1
260:             EEE(I) = EMIN + (EMAX-EMIN)*

```

src/mvp/sef5n2.f

```

261:      &      (EEE(I)-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
262:      end if
263: C
264:      end if
265: C
266: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
267: C
268:      if ( LFF5S.eq.5 ) then
269:          L5      = L1 + IWK10(I)
270:          THETA5   = CX(L5)*X1 + CX(L5-1)*(1.-X1)
271:          LL       = L1 + IWK9(I)
272:          NBINE    = KLIB1(INUC(I),8)
273:          if ( NBINE.gt.0 ) then
274:              NBINE1 = KLIB1(INUC(I),21)
275:              U5     = CX(LL+NBINE1*2)
276:
277: C/#IF ROUNDOFF(NEAREST)
278:          L6      = MIN(NBINE-1,INT(NBINE*R(I))) + LL
279: C/#ELSE
280:          R23     = NBINE*R(I)
281:          L6      = R23 + LL
282: C/#ENDIF
283:
284:          L7      = L6 + NBINE1
285:      else
286: C..... BMC SAMPLING USING 2 RANDOM NUMBERS
287:          NBINE    = ICX(LL)
288:          NBINE1   = 1
289:
290: C/#IF ROUNDOFF(NEAREST)
291:          LL5      = MIN(INT(NBINE*R(I))+1,NBINE)
292:          LL6      = LL + LL5
293:          LL7      = 2*LL5
294:          LL7      = LL + NBINE
295:          &
296: C/#ELSE
297:          LL5      = NBINE*R(I) + 1
298:          LL6      = LL + LL5
299:          R3       = R(NSEF5+I) - CX(LL6)
300:          LL7      = LL + NBINE + 2*LL5 + R3
301: C/#ENDIF
302:          L6      = LL + 3*NBINE + ICX(LL7)
303:          L7      = L6 + NBINE + 1
304:          U5      = CX(LL+5*NBINE+3)
305:      end if
306:      EHI5       = EIN(I) - U5
307: C
308:      E0         = CX(L6)
309:      E1         = CX(L6+1)
310:      P0         = CX(L7)
311:      P1         = CX(L7+1)
312: C
313:      RRR        = R(N2+I)*(P0+P1)
314:      X2         = E1
315:      &
316:      + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
317:
318: C      EEE(I) = X2*THETA5
319:
320:      if ( EEE(I).gt.EHI5 ) then
321:          NN      = NN + 1
322:          IWK8(NN) = I
323:          IWK9(NN) = LFF5S
324:          IWK10(NN) = LL
325:          IWK11(NN) = NBINE
326:          IWK12(NN) = NBINE1

```

```

326:          WK3(NN) = EHI5
327:          WK4(NN) = THETA5
328:      end if
329:      end if
330: C
331: C.... SIMPLE FISSION SPECTRUM (LF=7) USING 3 RANDOM NUMBERS
332: C
333:      if ( LFF5S.eq.7 ) then
334:          L8      = L1 + IWK10(I)
335:          THETA7   = CX(L8)*X1 + CX(L8-1)*(1.-X1)
336:          U7       = CX(L1+IWK9(I))
337:          EHI7     = EIN(I) - U7
338:          COSINE   = COS(PIH*R(I))
339:          COS2     = COSINE*COSINE
340: C
341:          EEE(I)   = -THETA7*(LOG(R(NSEF5+I))+LOG(R(N2+I))*COS2)
342: C
343:      if ( EEE(I).gt.EHI7 ) then
344:          NN      = NN + 1
345:          IWK8(NN) = I
346:          IWK9(NN) = LFF5S
347:          WK3(NN) = EHI7
348:          WK4(NN) = THETA7
349:      end if
350:      end if
351: C
352: C.... EVAPORATION SPECTRUM (LF=9) USING TWO RANDOM NUMBERS
353: C
354:      if ( LFF5S.eq.9 ) then
355:          L9      = L1 + IWK10(I)
356:          THETA9   = CX(L9)*X1 + CX(L9-1)*(1.-X1)
357:          U9       = CX(L1+IWK9(I))
358:          EHI9     = EIN(I) - U9
359: C
360:          EEE(I)   = -THETA9*LOG(R(NSEF5+I)*R(I))
361: C
362:      if ( EEE(I).gt.EHI9 ) then
363:          NN      = NN + 1
364:          IWK8(NN) = I
365:          IWK9(NN) = LFF5S
366:          WK3(NN) = EHI9
367:          WK4(NN) = THETA9
368:      end if
369:      end if
370: C
371: C.... WATT FISSION SPECTRUM (LF=11) USING FOUR RANDOM NUMBERS
372: C
373:      if ( LFF5S.eq.11 ) then
374:          L10     = L1 + IWK10(I)
375:          L11     = L10 + IWK9(I)
376:          A       = CX(L10)*X1 + CX(L10-1)*(1.-X1)
377:          B       = CX(L11)*X1 + CX(L11-1)*(1.-X1)
378:          U11     = CX(L1+IWK9(I))*2)
379:          EHI11   = EIN(I) - U11
380:          COSI    = COS(PIH*R(I))
381:          COSI2   = COSI*COSI
382:          V2      = -A*(LOG(R(NSEF5+I))+LOG(R(N2+I))*COSI2)
383:          XMU     = 2.*R(N3+I) - 1.
384: C
385:          EEE(I)  = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
386: C
387:      if ( EEE(I).gt.EHI11 ) then
388:          NN      = NN + 1
389:          IWK8(NN) = I
390:          IWK9(NN) = LFF5S

```

src/mvp/sef5n2.f

```

391:          WK3(NN) = EHI11
392:          WK4(NN) = A
393:          WK5(NN) = B
394:        end if
395:      end if
396:    130 continue
397: C
398:    if ( NN.gt.0 ) then
399:      NTEST = 1
400: C
401:    140 continue
402: C
403:      NNN = NN
404: C
405:      call RANU2( IRAND, R, NNN*4, ICON )
406: C
407:      N2 = NNN*2
408:      N3 = NNN*3
409: C
410:      NN = 0
411: C
412: *VOCL LOOP,NOVREC
413:   do 150 I = 1, NNN
414:     LFF5S = IWK9(I)
415: C
416: C===== DEPEND ON LFF5S-VALUE
417: C
418: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
419:   if ( LFF5S.eq.5 ) then
420:     if ( IWK12(I).gt.1 ) then
421: C
422: C/#IF ROUNDOFF(NEAREST)
423:       L6 = MIN(IWK11(I)-1,INT(IWK11(I)*R(I))+IWK10(I))
424: C/#ELSE
425:       R23 = IWK11(I)*R(I)
426:       L6 = R23 + IWK10(I)
427: C/#ENDIF
428:       L7 = L6 + IWK12(I)
429:     else
430: C
431: C/#IF ROUNDOFF(NEAREST)
432:       LL5 = MIN(INT(IWK11(I)*R(I))+1,IWK11(I))
433:       LL6 = IWK10(I) + LL5
434:       LL7 = 2*LL5
435:       LL7 = MIN(LL7,INT(LL7+R(NNN+I)-CX(LL6)))
436:       &
437: C/#ELSE
438:       LL5 = IWK11(I)*R(I) + 1
439:       LL6 = IWK10(I) + LL5
440:       R3 = R(NNN+I) - CX(LL6)
441:       LL7 = IWK10(I) + IWK11(I) + 2*LL5 + R3
442: C/#ENDIF
443: C
444:       L6 = IWK10(I) + 3*IWK11(I) + ICX(LL7)
445:       L7 = L6 + IWK11(I) + 1
446:     end if
447:     E0 = CX(L6)
448:     E1 = CX(L6+1)
449:     P0 = CX(L7)
450:     P1 = CX(L7+1)
451: C
452:     RRR = R(N2+I)*(P0+P1)
453:     X2 = E1
454:     &
455:     + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)

```

```

456:          EEE(IWK8(I)) = X2*WK4(I)
457:        end if
458: C
459: C..... SIMPLE FISSION SPECTRUN (LF=7) USING THREE RANDOM NUMBERS
460: C
461:   if ( LFF5S.eq.7 ) then
462:     COSINE = COS(PIH*R(I))
463:     COS2 = COSINE*COSINE
464: C
465:     EEE(IWK8(I)) = -WK4(I)*
466:     & (LOG(R(NNN+I))+LOG(R(N2+I))*COS2)
467:   end if
468: C
469: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
470: C
471:   if ( LFF5S.eq.9 ) then
472: C
473:     EEE(IWK8(I)) = -WK4(I)*LOG(R(I)*R(NNN+I))
474:   end if
475: C
476: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
477: C
478:   if ( LFF5S.eq.11 ) then
479:     A = WK4(I)
480:     B = WK5(I)
481:     COSI = COS(PIH*R(I))
482:     COSI2 = COSI*COSI
483:     V2 = -A*(LOG(R(NNN+I))+LOG(R(N2+I))*COSI2)
484:     XMU = 2.*R(N3+I) - 1.
485: C
486:     EEE(IWK8(I)) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
487: C
488:   end if
489: C
490:   if ( EEE(IWK8(I)).gt.WK3(I) ) then
491:     NN = NN + 1
492:     IWK8(NN) = IWK8(I)
493:     IWK9(NN) = IWK9(I)
494:     WK3(NN) = WK3(I)
495:     WK4(NN) = WK4(I)
496:     if ( LFF5S.eq.11 ) WK5(NN) = WK5(I)
497:     if ( LFF5S.eq.5 ) then
498:       IWK10(NN) = IWK10(I)
499:       IWK11(NN) = IWK11(I)
500:       IWK12(NN) = IWK12(I)
501:     end if
502:   end if
503: C
504:   150 continue
505: C
506: C IF( NN.GT.0 ) GOTO 499
507: if ( NN.gt.0 ) then
508:   NTEST = NTEST + 1
509:   if ( NTEST.le.300 ) then
510:     go to 140
511:   else
512:     do 160 I = 1, NN
513:       EEE(IWK8(I)) = WK3(I)
514:     160 continue
515:   end if
516: end if
517: end if
518: C
519: return
520: end

```

src/mvp/sef5n3.f

```

1:      subroutine SEF5N3( IOW, NSEF5, ISEL, EEEJ,
2:      &                  ISF5, ILF, THEA, THEB, EMIU,
3:      &                  CX, ICX, MCX, NBANK,
4:      &                  IWK1, R, IRAND )
5: C-----1-----2-----3-----4-----5-----6-----7---
6: C=====
7: C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA
8: C   using the following data already determined.
9: C
10: C   ISEL(J) : I = ISEL(J)
11: C   NSEF5   : number of particles to be processed (J=1,NSEF5)
12: C   EEEJ(J) : incident energy
13: C   ISF5(I) : pointer to used data in CX array
14: C   ILF(I)  : LF value
15: C   THEA(I) : theta for LF=5,7,9,11
16: C   THEB(I) : theta2 for LF=11
17: C   THEB(I) : minimum energy for LF=1,6,61 (INT>10)
18: C   EMIU(I) : EIN-U for LF=5,7,9,11
19: C
20: C
21: C   -----> EEEJ(J) : OUTGOING ENERGY
22: C
23: C CALLED IN: NXTNR3
24: C CALL      : RANU2
25: C UPDATE    :
26: C=====
27: C
28: C   implicit real*8(A-H,O-Z)
29: C
30: C   integer ISF5(NBANK), ILF(NBANK), ISEL(NSEF5)
31: C   real    EEEJ(NSEF5), THEA(NBANK), THEB(NBANK), EMIU(NBANK)
32: C
33: C**** CROSS SECTION DATA IN CX ARRAY
34: C   real CX(MCX)
35: C   integer ICX(MCX)
36: C
37: C
38: C**** WORK AREA
39: C
40: C   integer IWK1(NBANK)
41: C   real    R(NBANK*4)
42: C
43: C****
44: C   parameter( PI = 3.141592653589793200D+00 )
45: C   parameter( PIH = 0.5D0*PI )
46: C   parameter( SMALL = 1.0E-35 )
47: C
48: C   call RANU2( IRAND, R, NSEF5*4, ICON )
49: C
50: C   N2 = NSEF5*2
51: C   N3 = NSEF5*3
52: C
53: C   NN = 0
54: C
55: C *VOCL LOOP,NOVREC
56: C   do 100 J = 1, NSEF5
57: C     I = ISEL(J)
58: C
59: C==== DEPEND ON ILF-VALUE
60: C
61: C.... TABULATED PROBABILITY (LF=1) USING 3 OR 4 RANDOM NUMBERS
62: C   if ( ILF(I).eq.1 ) then
63: C
64: C..... BMC SAMPLING USING 2 RANDOM NUMBERS
65: C     LSTF5E = ISF5(I)

```

```

66: C     NEP = ICX(LSTF5E)
67: C
68: C /#IF ROUNDOFF(NEAREST)
69: C   LL2 = MIN( INT(NEP*R(J))+1,NEP )
70: C /#ELSE
71: C   LL2 = NEP*R(J) + 1
72: C /#ENDIF
73: C   LL3 = LSTF5E + LL2
74: C
75: C /#IF ROUNDOFF(NEAREST)
76: C   LL4 = 2*LL2
77: C   LL4 = MIN( LL4, INT( LL4+R(NSEF5+J)-CX( LL3 ) ) ) + LSTF5E +
78: C     & NEP
79: C /#ELSE
80: C   R2 = R(NSEF5+J) - CX( LL3 )
81: C   LL4 = LSTF5E + NEP + 2*LL2 + R2
82: C /#ENDIF
83: C   LLL = LSTF5E + 3*NEP
84: C   L3 = LLL + ICX( LL4 )
85: C   L4 = L3 + NEP + 1
86: C
87: C   E0 = CX( L3 )
88: C   E1 = CX( L3+1 )
89: C   P0 = CX( L4 )
90: C   P1 = CX( L4+1 )
91: C
92: C   ... the following formula is valid for both P0 = P1 and P0 >< P1.
93: C
94: C   RRR = (P0+P1)*R(N2+J)
95: C   EEEJ(J) = E1 + (E0-E1)*RRR/
96: C     & (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
97: C
98: C   if ( THEA(I).gt.0. .and. THEB(I).gt.0. ) then
99: C     LLE = LLL + 1
100: C     EEEJ(J) = THEB(I) + (THEA(I)-THEB(I))*
101: C       & (EEEJ(J)-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
102: C   else if ( THEA(I).gt.0. ) then
103: C     LLE = LLL + 1
104: C     EEEJ(J) = THEA(I)*EEEJ(J)/CX(LLE)
105: C   end if
106: C   end if
107: C
108: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
109: C
110: C   if ( ILF(I).eq.5 ) then
111: C     LL = ISF5(I)
112: C..... BMC SAMPLING USING 4 RANDOM NUMBERS
113: C     NBINE = ICX( LL )
114: C
115: C /#IF ROUNDOFF(NEAREST)
116: C   LL5 = MIN( INT(NBINE*R(J))+1,NBINE )
117: C /#ELSE
118: C   LL5 = NBINE*R(J) + 1
119: C /#ENDIF
120: C
121: C   LL6 = LL + LL5
122: C
123: C /#IF ROUNDOFF(NEAREST)
124: C   LL7 = 2*LL5
125: C   LL7 = MIN( LL7, INT( LL7+R(NSEF5+J)-CX( LL6 ) ) ) + LL
126: C     & NBINE
127: C /#ELSE
128: C   R3 = R(NSEF5+J)-CX( LL6 )
129: C   LL7 = LL+NBINE + 2*LL5+R3
130: C /#ENDIF

```

src/mvp/sef5n3.f

```

131:
132:      L6      = LL + 3*NBINE + ICX(LL7)
133:      L7      = L6 + NBINE + 1
134: C
135:      E0      = CX(L6)
136:      E1      = CX(L6+1)
137:      P0      = CX(L7)
138:      P1      = CX(L7+1)
139: C
140:      RRR      = (P0+P1)*R(N2+J)
141:      X2      = E1 + (E0-E1)*RRR/
142:      &      (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
143:      EEEJ(J) = X2*THEA(I)
144:      end if
145: C
146: C..... SIMPLE FISSION SPECTRUN (LF=7) USING 3 RANDOM NUMBERS
147:      if ( ILF(I).eq.7 ) then
148:      COSINE   = COS(PIH*R(J))
149:      COS2     = COSINE*COSINE
150:      EEEJ(J) = -THEA(I)*(LOG(R(NSEF5+J))+LOG(R(N2+J))*COS2)
151:      end if
152: C
153: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
154:      if ( ILF(I).eq.9 ) then
155:      EEEJ(J) = -THEA(I)*LOG(R(NSEF5+J)*R(J))
156:      end if
157: C
158: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
159:      if ( ILF(I).eq.11 ) then
160:      A      = THEA(I)
161:      B      = THEB(I)
162:      COSI   = COS(PIH*R(J))
163:      COSI2  = COSI*COSI
164:      V2     = -A*(LOG(R(NSEF5+J))+LOG(R(N2+J))*COSI2)
165:      XMU    = 2.*R(N3+J) - 1.
166:      EEEJ(J) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
167:      end if
168: C
169:      if ( ILF(I).ne.1.and.EEEJ(J).gt.EMIU(I) ) then
170:      NN      = NN + 1
171:      IWK1(NN) = J
172:      end if
173: 100 continue
174: C
175:      if ( NN.gt.0 ) then
176:      NTEST   = 1
177: C
178: 200      continue
179: C
180:      NNN     = NN
181: C
182:      call RANU2( IRAND, R, NNN*4, ICON )
183: C
184:      N2      = NNN*2
185:      N3      = NNN*3
186: C
187:      NN      = 0
188: C
189: *VOCL LOOP,NOVREC
190:      do 210 JJ = 1, NNN
191:      J      = IWK1(JJ)
192:      I      = ISEL(J)
193:      LFF5S   = ILF(I)
194: C
195: C===== DEPEND ON LFF5S-VALUE

```

```

196: C
197: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
198:      if ( LFF5S.eq.5 ) then
199:      NBINE   = ICX(ISF5(I))
200: C
201: C/#IF ROUNDOFF(NEAREST)
202:      LL5     = MIN(INT(NBINE*R(JJ))+1,NBINE)
203: C/#ELSE
204:      LL5     = NBINE*R(JJ) + 1
205: C/#ENDIF
206: C
207:      LL6     = ISF5(I) + LL5
208: C
209: C/#IF ROUNDOFF(NEAREST)
210:      LL7     = 2*LL5
211:      LL7     = MIN(LL7,INT(LL7+R(NNN+JJ))-CX(LL6)) +
212:      &      ISF5(I) + NBINE
213: C/#ELSE
214:      R3      = R(NNN+JJ) - CX(LL6)
215:      LL7     = ISF5(I) + NBINE + 2*LL5 + R3
216: C/#ENDIF
217:      L6      = ISF5(I) + 3*NBINE + ICX(LL7)
218:      L7      = L6 + NBINE + 1
219:      E0      = CX(L6)
220:      E1      = CX(L6+1)
221:      P0      = CX(L7)
222:      P1      = CX(L7+1)
223: C
224:      RRR      = (P0+P1)*R(N2+JJ)
225:      X2      = E1 + (E0-E1)*RRR/
226:      &      (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
227:      EEEJ(J) = X2*THEA(I)
228:      end if
229: C
230: C
231: C..... SIMPLE FISSION SPECTRUN (LF=7) USING THREE RANDOM NUMBERS
232:      if ( LFF5S.eq.7 ) then
233:      COSINE   = COS(PIH*R(JJ))
234:      COS2     = COSINE*COSINE
235: C
236:      EEEJ(J) = -THEA(I)*
237:      &      (LOG(R(NNN+JJ))+LOG(R(N2+JJ))*COS2)
238:      end if
239: C
240: C
241: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
242:      if ( LFF5S.eq.9 ) then
243: C
244:      EEEJ(J) = -THEA(I)*LOG(R(JJ)*R(NNN+JJ))
245:      end if
246: C
247: C
248: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
249:      if ( LFF5S.eq.11 ) then
250:      A      = THEA(I)
251:      B      = THEB(I)
252:      COSI   = COS(PIH*R(JJ))
253:      COSI2  = COSI*COSI
254:      V2     = -A*(LOG(R(NNN+JJ))+LOG(R(N2+JJ))*COSI2)
255:      XMU    = 2.*R(N3+JJ) - 1.
256: C
257:      EEEJ(J) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
258: C
259:      end if
260: C

```

src/mvp/sef5n3.f

```
261:          if ( EEEJ(J).gt.EMIU(I) ) then
262:              NN      = NN + 1
263:              IWK1(NN) = IWK1(JJ)
264:          end if
265: C
266: 210      continue
267: C
268:          if ( NN.gt.0 ) then
269:              NTEST    = NTEST + 1
270:              if ( NTEST.le.300 ) then
271:                  go to 200
272:              else
273: *VOCL LOOP,NOVREC
274:                  do 220 JJ = 1, NN
275:                      J      = IWK1(JJ)
276:                      I      = ISEL(J)
277:                      EEEJ(J) = EMIU(I)
278: 220              continue
279:                  end if
280:              end if
281:          end if
282: C
283:          return
284:      end
```

src/mvp/sef5n4.f

```

1:      subroutine SEF5N4( IOW,  INUC,  NSEF5, EIN,  EEE,  CX,  ICX,
2:      &                  MCX,  KLIB1, XLIB1, KLIB2, XLIB2, NBANK, NUC,
3:      &                  NMT,  IRAND, JDEBG,
4:      &                  IWK8, IWK9, IWK10, IWK11,
5:      &                  IWK12, IWK13, WK3,  WK4,  WK5,  R,
6:      &                  TOUT, IMT,  JDLYN, JTIME )
7: C=====
8: C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 DATA
9: C      MT = 18, including delayed neutron.
10: C
11: C      i/o : comment
12: C      INUC(I)  i : COLLIDING NUCLIDE
13: C      NSEF5    i : NUMBER OF PARTICLES
14: C      EIN(I)   i : INCIDENT ENERGY
15: C      EEE(I)   o : OUTGOING ENERGY
16: C      TOUT(I)  o : delayed time
17: C      IMT(I)   i : flag of delayed neutron (=98)
18: C      JDLYN    i : option to treat the delayed neutron (0/1=off/on)
19: C      JTIME    i : option to treat the time-dependent (0/1=off/on)
20: C-----
21: C CALLED IN: FISGEN
22: C CALLS      : RANU2, BSDEC3
23: C-----
24: C
25:      implicit real*8(A-H,O-Z)
26: C
27:      parameter( PIH  = 3.141592653589793200d0*0.5d0 )
28:      parameter( SMALL = 1.0d-35 )
29: C
30:      integer JDEBG(*)
31: C
32: C**** CROSS SECTION DATA IN CX ARRAY
33: C
34:      real    CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
35:      integer ICX(MCX)
36:      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
37: C
38:      integer INUC(NBANK)
39:      real    EIN(NBANK)
40:      real    EEE(NBANK)
41: C
42:      real    TOUT(NBANK)
43:      integer IMT(NBANK)
44: C
45: C**** WORKING AREA
46: C
47:      integer IWK8(NBANK), IWK9(NBANK), IWK10(NBANK), IWK11(NBANK),
48:      &        IWK12(NBANK), IWK13(NBANK)
49:      real    WK3(NBANK), WK4(NBANK), WK5(NBANK)
50:      real    R(4*NBANK)
51: C
52: C
53: C-----
54: C
55: C**** DETERMINE THE SUBSECTION USED
56: C.... MAKE TEMPORARY STACK OF PARTICLE HAVING SUBSECTION > 1
57: C      IWK11(J)
58: C      IWK12(I) : SUBSECTION NO. USED FOR I-TH PARTICLE
59: C      NN      = 0
60: C      do 100 I = 1, NSEF5
61: C          MT      = 18
62: C          JJDLYN  = JDLYN
63: C          if ( KLIB1(INUC(I),23).eq.0 ) JJDLYN = 0
64: C          if ( JJDLYN.ne.0.and.IMT(I).eq.98 ) MT = 98
65: C          IWK12(I) = KLIB2(INUC(I),MT,8)

```

```

66:      IWK13(I) = KLIB2(INUC(I),MT,13)
67:      if ( IWK12(I).ge.2 ) then
68:      NN      = NN + 1
69:      IWK11(NN) = I
70:      IWK8(NN)  = IWK13(I) + IWK12(I)
71:      IWK9(NN)  = KLIB2(INUC(I),MT,9)
72:      WK3(NN) = EIN(I)
73:      end if
74:      100 continue
75: C
76: C**** THE CASE WHERE MANY SUBSECTIONS ARE GIVEN
77: C
78:      if ( NN.gt.0 ) then
79: C
80: C.... SELECTION OF USED TABLE ----> IWK10
81: C      IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
82: C      IWK9 : NUMBER OF INCIDENT ENERGY
83: C      WK3 : INCIDENT NEUTRON ENERGY
84: C      WK4 : WORK AREA IN BSDEC3
85: C      IWK10: Table position as results
86: C
87: C          T,  NT,  MW,  LS,  X,  IT,  N1
88: C      call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NN )
89: C
90: C      call RANU2( IRAND, R, NN*3, ICON )
91: C      NN2      = NN*2
92: C
93: C*VOCL LOOP,NOVREC
94: C      do 110 J = 1, NN
95: C          I      = IWK11(J)
96: C          L2     = IWK8(J) + IWK10(J)
97: C          L1     = L2 - 1
98: C..... LINEAR INTERPOLATION
99: C          X1     = (CX(L1)-WK3(J)) / (CX(L1)-CX(L2))
100: C##<2007/03/14:PN3:
101: C          if ( X1.gt.1.0d0 ) X1 = 1
102: C          if ( X1.lt.0.0d0 ) X1 = 0
103: C##>
104: C
105: C/#IF ROUND OFF(NEAREST)
106: C          IT     = MIN(1,INT(R(J)+X1)) + IWK10(J)
107: C/#ELSE
108: C          R1     = R(J) + X1
109: C          IT     = IWK10(J) + R1
110: C/#ENDIF
111: C
112: C..... BMC SAMPLING
113: C          NKF5   = IWK12(I)
114: C          L1     = IWK8(J) + IWK9(J)*2 + 3*NKF5*(IT-1) - 1
115: C
116: C/#IF ROUND OFF(NEAREST)
117: C          L2     = MIN(INT(NKF5*R(NN+J))+1,NKF5)
118: C          L3     = L1 + L2
119: C          L4     = 2*L2
120: C          L4     = MIN(L4,INT(L4+R(NN2+J)-CX(L3))) + L1 + NKF5
121: C/#ELSE
122: C          L2     = NKF5*R(NN+J) + 1
123: C          L3     = L1 + L2
124: C          R2     = R(NN2+J) - CX(L3)
125: C          L4     = L1 + NKF5 + 2*L2 + R2
126: C/#ENDIF
127: C          IWK12(I) = ICX(L4)
128:      110 continue
129: C
130:      end if

```

src/mvp/sef5n4.f

```

131: C
132: C
133: C.... DETERMINE TABLE USED ----> IWK10(I) = INTERVAL
134: C    IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
135: C    IWK9 : NUMBER OF INCIDENT ENERGY
136: C    IWK12: SUBSECTION NUMBER USED
137: C    EIN  : INCIDENT NEUTRON ENERGY
138: C
139: C    do 120 I = 1, NSEF5
140: C        IWK13(I) = ICX(IWK13(I)+IWK12(I)-1)
141: C        IWK8(I) = IWK13(I) + 2
142: C        IWK9(I) = ICX(IWK13(I)+1)
143: C    120 continue
144: C
145: C    call BSDEC3( CX, IWK9, WK4, IWK8, EIN, IWK10, NSEF5 )
146: C
147: C-----
148: C**** SAMPLING OF DELAYED TIME FROM SELECTED SUBSECTION
149: C-----
150: C
151: C    if ( JTIME.ne.0 ) then
152: C        call RANU2( IRAND, R, NSEF5, ICON )
153: C        do 121 I = 1, NSEF5
154: C            MT = 18
155: C            JJDLYN = JDLYN
156: C            if ( KLIB1(INUC(I),23).eq.0 ) JJDLYN = 0
157: C            if ( JJDLYN.ne.0.and.IMT(I).eq.98 ) MT = 98
158: C            if ( MT.eq.98 ) then
159: C                LSRAMDA = KLIB1(INUC(I),25) - 1 +
160: C                    & KLIB1(INUC(I),22) * KLIB1(INUC(I),23)
161: C                NNF = KLIB1(INUC(I),24)
162: C                ISSN = max(1, min(IWK12(I), NNF))
163: C                TOUT(I) = - log(1-R(I)) / CX(LSRAMDA+ISSN)
164: C            else
165: C                TOUT(I) = 0
166: C            end if
167: C        121 continue
168: C    else
169: C        do 122 I = 1, NSEF5
170: C            TOUT(I) = 0
171: C        122 continue
172: C    end if
173: C
174: C**** SAMPLING OF OUTGOING ENERGY FROM DATA IN SUBSECTION IWK12(I)
175: C    MAKE TEMPORARY STACK, THESE ARRAYS ARE CHANGED DURING CALCULATION
176: C    IWK8(NN) : TEMPORARY STACK FOR E' > E-U
177: C    IWK9(NN) : LFF5S
178: C    IWK10(NN) :
179: C    IWK11(NN) :
180: C    IWK12(NN) :
181: C    WK3 (NN) :
182: C    WK4 (NN) :
183: C    WK5 (NN) :
184: C
185: C    call RANU2( IRAND, R, NSEF5*4, ICON )
186: C
187: C    N2 = NSEF5*2
188: C    N3 = NSEF5*3
189: C
190: C    NN = 0
191: C
192: C*VOCL LOOP,NOVREC
193: C    do 130 I = 1, NSEF5
194: C        L2 = IWK8(I) + IWK10(I)
195: C        L1 = L2 - 1

```

```

196: C    INTE0 = ICX(L2+IWK9(I))
197: C    INTE1 = INTE0 / 10
198: C    INTE = INTE0 - 10*INTE1
199: C    if ( INTE.eq.5 .or. INTE.eq.3 ) then
200: C        LOG INTERPOLATION
201: C        X1 = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
202: C    else
203: C        LINEAR INTERPOLATION
204: C        X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
205: C    end if
206: C
207: C    LFF5S = ICX(IWK13(I))
208: C    L1 = IWK8(I) + IWK9(I)*2
209: C
210: C==== DEPEND ON LFF5S-VALUE
211: C
212: C.... TABULATED PROBABILITY (LF=1) USING 3 OR 4 RANDOM NUMBERS
213: C    if ( LFF5S.eq.1 ) then
214: C
215: C/#IF ROUNDOFF(NEAREST)
216: C    IT = MIN(1,INT(R(I)+X1)) + IWK10(I)
217: C/#ELSE
218: C    R1 = R(I) + X1
219: C    IT = IWK10(I) + R1
220: C/#ENDIF
221: C
222: C    NBINE1 = KLIB1(INUC(I),21)
223: C    if ( NBINE1.gt.1 ) then
224: C
225: C        LLE = NBINE1*2*(IT-1) + L1
226: C        LLE2 = NBINE1*2*(IWK10(I)) + L1
227: C        LLE1 = LLE2 - NBINE1*2
228: C        NEP = NBINE1 - 1
229: C        NEP1 = NEP
230: C        NEP2 = NEP
231: C/#IF ROUNDOFF(NEAREST)
232: C##<2007/03/14:PN3:
233: C##    LKL1 = KLIB1(INUC(I),8)
234: C##    L3 = MIN(INT(LKL1*R(NSEF5+I))+1,LKL1)
235: C##    L3 = min(int(NEP*R(NSEF5+I))+1,NEP)
236: C##>
237: C/#ELSE
238: C##<2007/03/14:PN3:
239: C##    L3 = KLIB1(INUC(I),8)*R(NSEF5+I) + 1
240: C##    L3 = NEP*R(NSEF5+I) + 1
241: C##>
242: C/#ENDIF
243: C
244: C    L3 = L3 - 1 + LLE
245: C    L4 = L3 + NBINE1
246: C    else
247: C        BMC SAMPLING USING 4 RANDOM NUMBERS
248: C        LSTF5E = ICX(L1+IT-1)
249: C        NEP = ICX(LSTF5E)
250: C
251: C/#IF ROUNDOFF(NEAREST)
252: C    LL2 = MIN(INT(NEP*R(NSEF5+I))+1,NEP)
253: C    LL3 = LSTF5E + LL2
254: C    LL4 = 2*LL2
255: C    LL4 = LSTF5E + NEP +
256: C        & MIN(LL4,INT(LL4+R(N3+I)-CX(LL3)))
257: C/#ELSE
258: C    LL2 = NEP*R(NSEF5+I) + 1
259: C    LL3 = LSTF5E + LL2
260: C    R2 = R(N3+I) - CX(LL3)

```


src/mvp/sef5n4.f

```

261: *          LL4      = LSTF5E + NEP + 2*LL2 + R2
262: C/#ENDIF
263:
264:          LLL      = LSTF5E + 3*NEP
265:          L3       = LLL + ICX(LL4)
266:          L4       = L3 + NEP + 1
267: C
268:          LLE      = LLL + 1
269:          if ( IT.eq.IWK10(I) ) then
270:            NEP1    = NEP
271:            LLE1    = LLE
272:            LSTF5T  = ICX(L3+IT)
273:            NEP2    = ICX(LSTF5T)
274:            LLE2    = LSTF5T + 3*NEP2 + 1
275:          else
276:            NEP2    = NEP
277:            LLE2    = LLE
278:            LSTF5T  = ICX(L1+IT-2)
279:            NEP1    = ICX(LSTF5T)
280:            LLE1    = LSTF5T + 3*NEP1 + 1
281:          end if
282:        end if
283:        E0      = CX(L3)
284:        E1      = CX(L3+1)
285:        P0      = CX(L4)
286:        P1      = CX(L4+1)
287: C
288:        RRR     = R(N2+I)*(P0+P1)
289:        EEE(I)  = E1
290:        &      + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
291: C
292:        if ( INTE1.eq.2 ) then
293: C ... linear interpolation is assumed
294:          EMAX   = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
295:          EEE(I) = EEE(I) * EMAX / CX(LLE)
296:        else if ( INTE1.eq.1 ) then
297: C ... linear interpolation is assumed
298:          EMAX   = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
299:          EMIN   = CX(LLE1+NEP1)
300:          &      + (CX(LLE2+NEP2)-CX(LLE1+NEP1))*X1
301:          EEE(I) = EMIN + (EMAX-EMIN)*
302:          &      (EEE(I)-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
303:        end if
304: C
305:        end if
306: C
307: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
308: C
309:        if ( LFF5S.eq.5 ) then
310:          L5      = L1 + IWK10(I)
311:          THETA5  = CX(L5)*X1 + CX(L5-1)*(1.-X1)
312:          LL      = L1 + IWK9(I)
313:          NBINE   = KLIB1(INUC(I),8)
314:          if ( NBINE.gt.0 ) then
315:            NBINE1 = KLIB1(INUC(I),21)
316:            U5     = CX(LL+NBINE1*2)
317:          end if
318: C/#IF ROUNDOFF(NEAREST)
319:          L6      = MIN(NBINE-1,INT(NBINE*R(I))) + LL
320: C/#ELSE
321:          *        R23    = NBINE*R(I)
322:          *        L6     = R23 + LL
323: C/#ENDIF
324:
325:          L7      = L6 + NBINE1

```

```

326:          else
327: C..... BMC SAMPLING USING 2 RANDOM NUMBERS
328:          NBINE   = ICX(LL)
329:          NBINE1  = 1
330:
331: C/#IF ROUNDOFF(NEAREST)
332:          LL5     = MIN(INT(NBINE*R(I))+1,NBINE)
333:          LL6     = LL + LL5
334:          LL7     = 2*LL5
335:          LL7     = LL + NBINE
336:          &      + MIN(LL7,INT(LL7+R(NSEF5+I)-CX(LL6)))
337: C/#ELSE
338:          *        LL5    = NBINE*R(I) + 1
339:          *        LL6    = LL + LL5
340:          *        R3     = R(NSEF5+I) - CX(LL6)
341:          *        LL7    = LL + NBINE + 2*LL5 + R3
342: C/#ENDIF
343:          L6      = LL + 3*NBINE + ICX(LL7)
344:          L7      = L6 + NBINE + 1
345:          U5      = CX(LL+5*NBINE+3)
346:        end if
347:        EHI5     = EIN(I) - U5
348: C
349:        E0      = CX(L6)
350:        E1      = CX(L6+1)
351:        P0      = CX(L7)
352:        P1      = CX(L7+1)
353: C
354:        RRR     = R(N2+I)*(P0+P1)
355:        X2      = E1
356:        &      + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
357: C
358:        EEE(I)  = X2*THETA5
359: C
360:        if ( EEE(I).gt.EHI5 ) then
361:          NN     = NN + 1
362:          IWK8(NN) = I
363:          IWK9(NN) = LFF5S
364:          IWK10(NN) = LL
365:          IWK11(NN) = NBINE
366:          IWK12(NN) = NBINE1
367:          WK3(NN) = EHI5
368:          WK4(NN) = THETA5
369:        end if
370:        end if
371: C
372: C.... SIMPLE FISSION SPECTRUM (LF=7) USING 3 RANDOM NUMBERS
373: C
374:        if ( LFF5S.eq.7 ) then
375:          L8      = L1 + IWK10(I)
376:          THETA7  = CX(L8)*X1 + CX(L8-1)*(1.-X1)
377:          U7      = CX(LL+IWK9(I))
378:          EHI7    = EIN(I) - U7
379:          COSINE  = COS(PIH*R(I))
380:          COS2    = COSINE*COSINE
381: C
382:          EEE(I)  = -THETA7*(LOG(R(NSEF5+I))+LOG(R(N2+I)))*COS2)
383: C
384:        if ( EEE(I).gt.EHI7 ) then
385:          NN     = NN + 1
386:          IWK8(NN) = I
387:          IWK9(NN) = LFF5S
388:          WK3(NN) = EHI7
389:          WK4(NN) = THETA7
390:        end if

```

src/mvp/sef5n4.f

```

391:         end if
392: C
393: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
394: C
395:         if ( LFF5S.eq.9 ) then
396:             L9      = L1 + IWK10(I)
397:             THETA9   = CX(L9)*X1 + CX(L9-1)*(1.-X1)
398:             U9       = CX(L1+IWK9(I))
399:             EHI9     = EIN(I) - U9
400: C
401:             EEE(I)   = -THETA9*LOG(R(NSEF5+I))*R(I))
402: C
403:             if ( EEE(I).gt.EHI9 ) then
404:                 NN    = NN + 1
405:                 IWK8(NN) = I
406:                 IWK9(NN) = LFF5S
407:                 WK3(NN) = EHI9
408:                 WK4(NN) = THETA9
409:             end if
410:         end if
411: C
412: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
413: C
414:         if ( LFF5S.eq.11 ) then
415:             L10     = L1 + IWK10(I)
416:             L11     = L10 + IWK9(I)
417:             A       = CX(L10)*X1 + CX(L10-1)*(1.-X1)
418:             B       = CX(L11)*X1 + CX(L11-1)*(1.-X1)
419:             U11     = CX(L1+IWK9(I)*2)
420:             EHI11   = EIN(I) - U11
421:             COSI    = COS(PIH*R(I))
422:             COSI2   = COSI*COSI
423:             V2      = -A*(LOG(R(NSEF5+I))+LOG(R(N2+I))*COSI2)
424:             XMU     = 2.*R(N3+I) - 1.
425: C
426:             EEE(I)  = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
427: C
428:             if ( EEE(I).gt.EHI11 ) then
429:                 NN    = NN + 1
430:                 IWK8(NN) = I
431:                 IWK9(NN) = LFF5S
432:                 WK3(NN) = EHI11
433:                 WK4(NN) = A
434:                 WK5(NN) = B
435:             end if
436:         end if
437: 130 continue
438: C
439:         if ( NN.gt.0 ) then
440:             NTEST = 1
441: C
442: 140 continue
443: C
444:             NNN    = NN
445: C
446:             call RANU2( IRAND, R, NNN*4, ICON )
447: C
448:             N2     = NNN*2
449:             N3     = NNN*3
450: C
451:             NN      = 0
452: C
453: *VOCL LOOP,NOVREC
454:         do 150 I = 1, NNN
455:             LFF5S   = IWK9(I)

```

```

456: C
457: C===== DEPEND ON LFF5S-VALUE
458: C
459: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
460:         if ( LFF5S.eq.5 ) then
461:             if ( IWK12(I).gt.1 ) then
462: C/#IF ROUNDOFF(NEAREST)
463:                 L6      = MIN(IWK11(I)-1,INT(IWK11(I)*R(I)))+IWK10(I)
464: C/#ELSE
465:                 R23     = IWK11(I)*R(I)
466:                 L6      = R23 + IWK10(I)
467: C/#ENDIF
468:                 L7      = L6 + IWK12(I)
469:             else
470: C/#IF ROUNDOFF(NEAREST)
471:                 LL5     = MIN(INT(IWK11(I)*R(I))+1,IWK11(I))
472:                 LL6     = IWK10(I) + LL5
473:                 LL7     = 2*LL5
474:                 LL7     = MIN(LL7,INT(LL7+R(NNN+I)-CX(LL6))) +
475:                     & IWK10(I) + IWK11(I)
476: C/#ELSE
477:                 LL5     = IWK11(I)*R(I) + 1
478:                 LL6     = IWK10(I) + LL5
479:                 R3      = R(NNN+I) - CX(LL6)
480:                 LL7     = IWK10(I) + IWK11(I) + 2*LL5 + R3
481: C/#ENDIF
482:                 L6      = IWK10(I) + 3*IWK11(I) + ICX(LL7)
483:                 L7      = L6 + IWK11(I) + 1
484:             end if
485:             E0         = CX(L6)
486:             E1         = CX(L6+1)
487:             P0         = CX(L7)
488:             P1         = CX(L7+1)
489: C
490:             RRR        = R(N2+I)*(P0+P1)
491:             X2         = E1
492:             &          + (E0-E1)*RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
493: C
494:             EEE(IWK8(I)) = X2*WK4(I)
495:         end if
496: C
497: C..... SIMPLE FISSION SPECTRUN (LF=7) USING THREE RANDOM NUMBERS
498: C
499:         if ( LFF5S.eq.7 ) then
500:             COSINE = COS(PIH*R(I))
501:             COS2   = COSINE*COSINE
502: C
503:             EEE(IWK8(I)) = -WK4(I)*
504:                 & (LOG(R(NNN+I))+LOG(R(N2+I))*COS2)
505:         end if
506: C
507: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
508: C
509:         if ( LFF5S.eq.9 ) then
510:             EEE(IWK8(I)) = -WK4(I)*LOG(R(I)*R(NNN+I))
511:         end if
512: C
513: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
514: C
515:         if ( LFF5S.eq.11 ) then
516:             A      = WK4(I)

```

src/mvp/sef5n4.f

```
521:          B          = WK5(I)
522:          COSI        = COS(PIH*R(I))
523:          COSI2        = COSI*COSI
524:          V2          = -A*(LOG(R(NNN+I))+LOG(R(N2+I)))*COSI2)
525:          XMU          = 2.*R(N3+I) - 1.
526: C
527:          EEE(IWK8(I))  = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
528: C
529:          end if
530: C
531:          if ( EEE(IWK8(I)).gt.WK3(I) ) then
532:              NN        = NN + 1
533:              IWK8(NN)   = IWK8(I)
534:              IWK9(NN)   = IWK9(I)
535:              WK3(NN)    = WK3(I)
536:              WK4(NN)    = WK4(I)
537:              if ( LFF5S.eq.11 ) WK5(NN) = WK5(I)
538:              if ( LFF5S.eq.5 ) then
539:                  IWK10(NN) = IWK10(I)
540:                  IWK11(NN) = IWK11(I)
541:                  IWK12(NN) = IWK12(I)
542:              end if
543:          end if
544: C
545: 150      continue
546: C
547: C      IF( NN.GT.0 ) GOTO 499
548:          if ( NN.gt.0 ) then
549:              NTEST      = NTEST + 1
550:              if ( NTEST.le.300 ) then
551:                  go to 140
552:              else
553:                  do 160 I = 1, NN
554:                      EEE(IWK8(I))  = WK3(I)
555: 160              continue
556:                  end if
557:              end if
558:          end if
559: C
560:          return
561:          end
```

src/mvp/sef5n.f

```

1:      subroutine SEF5N( IOW,  LSEF5, IWK1,  IWK2,  NSEF5, EIN,  LSCOL,
2:      &                 NCOLS, NF60,  NF6,   NF5,   CTH,  EEE,  CX,
3:      &                 ICX,  MCX,  KLIB1,  XLIB1,  KLIB2,  XLIB2,  NBANK,
4:      &                 NUC,  NMT,  IRAND,  JDEBG,  JLAT,  JHLAT,  JTLT,
5:      &                 JIMPT,  JTIME,  NEST,  LSEF5,  NSEF5,  XXX,  YYY,
6:      &                 ZZZ,  AAA,  BBB,  CCC,  WWW,  TTT,  ITT,
7:      &                 KKP,  IZZ,  XIM,  KLSF,  LEVL,  LZZ,  LPOS,  LCRS,
8:      &                 IBREG,  IBSPC,  DBNK,  KDBNK,  MDBNK,  IBNK,  KIBNK,
9:      &                 MIBNK,  NCNTR,  WCNTR,  EWCUT,  IWK6,  IWK7,  IWK8,
10:      &                 IWK9,  IWK10,  IWK11,  IWK12,  IWK13,  WK3,  WK4,
11:      &                 WK5,  R,
12:      &                 JPRT,  NPTDS,  WWD,  WWD2,  NPTCS,  WWR,
13:      &                 NGSP,  WSD,  WSC,  NORDD,  NOPSD,
14:      c##<2007/03/14:PN4:
15:      &                 KPNEG,  JPHNU
16:      c##>
17:      c+beff2
18:      &                 JBEFF,  WWB,  WSB,  NPTBE,
19:      &                 WWBD,  WSBD,  WWLD,  WSLD)
20:      c-beff2
21:      C---5---1---5---2---5---3---5---4---5---5---5---6---5---7---
22:      C=====
23:      C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 5 & 6 DATA
24:      C   LSEF5(I) : STACK POINTER, IWK1(I) : COLLIDING NUCLIDE,
25:      C   IWK2(I) : REACTION TYPE (MF), NSEF5 : NUMBER OF PARTICLES
26:      C   EIN(I)  : INCIDENT ENERGY
27:      C   LSCOL(I) : STACK, NF6 : NUMBER OF PARTICLES BY FILE 6 DATA
28:      C   NF60 (WITHOUT FILE 4 DATA) INCLUDED IN NF6
29:      C   NF5 : NUMBER OF PARTICLES BY FILE 5 DATA   NSEF5 = NF5 + NF6
30:      C   -----> EEE(LSCOL(LSEF5(I))) : OUTGOING ENERGY
31:      C   CTH(LSEF5(I)) : SCATTERING ANGLE COSINE ( NF60 )
32:      C
33:      C CALLED IN: NTHSCT
34:      C CALL      : RANU2,BSDEC3,BSINC3
35:      C=====
36:      C
37:      C   === BANK DATA TO BE UPDATED ===
38:      C
39:      C   EEE : ENERGY
40:      C
41:      C=====
42:      C
43:      C   implicit real*8(A-H,O-Z)
44:      C
45:      C   integer JDEBG(*)
46:      C
47:      C**** CROSS SECTION DATA IN CX ARRAY
48:      c##<2007/03/14:PN3:
49:      c##      real CX(MCX),  XLIB1(NUC,11),  XLIB2(NUC,NMT,2)
50:      c##      real CX(MCX),  XLIB1(NUC,11),  XLIB2(NUC,NMT,4)
51:      c##>
52:      c##      real EWCUT
53:      c##      integer ICX(MCX)
54:      c##<2007/03/14:PN3:
55:      c##      integer KLIB1(NUC,29),  KLIB2(NUC,NMT,19)
56:      c##      integer KLIB1(NUC,31),  KLIB2(NUC,NMT,21)
57:      c##>
58:      C
59:      C**** STACK(TEMPORARY) = IWK13 IN NTHSCT
60:      C   integer LSEF5(NBANK)
61:      C
62:      C**** STACK
63:      C   integer LSCOL(NBANK),  LSEF5(NBANK)
64:      C
65:      C**** OUTPUT : PARTICLE BANK

```

```

66:      real*8 XXX(NBANK),  YYY(NBANK),  ZZZ(NBANK)
67:      real*8 AAA(NBANK),  BBB(NBANK),  CCC(NBANK)
68:      real EEE(NBANK),  WWW(NBANK),  XIM(NBANK)
69:      integer KKP(NBANK)
70:      integer IZZ(NBANK),  LEVL(NBANK),  LZZ(NBANK,NEST),
71:      &                 LPOS(NBANK,NEST),  LCRS(NBANK,NEST),  KLSF(NBANK)
72:      real*8 TTT(NBANK)
73:      integer ITT(NBANK)
74:      integer IBREG(NBANK),  IBSPC(NBANK,0:NEST)
75:      real CTH(NBANK)
76:      C
77:      real*8 DBNK(NBANK,MDBNK)
78:      integer KDBNK(0:MDBNK)
79:      integer IBNK(NBANK,MIBNK)
80:      integer KIBNK(0:MIBNK)
81:      c##<2007/03/14:PN4:
82:      c##>      integer KPNEG(NBANK)
83:      C
84:      C
85:      C.... (TEMPORARY) = WK3 IN NTHSCT
86:      C   real EIN(NBANK)
87:      C   real*8 WWD(NBANK,NPTDS,NORDD),  WWD2(NBANK,NPTDS)
88:      C   real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSD)
89:      C   real*8 WWR(NBANK,NPTCS)
90:      C   real*8 WSC(NBANK,0:NGSP,NPTCS)
91:      c+beff
92:      C   real*8 WWB(NBANK)
93:      C   real*8 WSB(NBANK,0:NGSP)
94:      C   real*8 WWBD(NBANK)
95:      C   real*8 WSBD(NBANK,NGSP)
96:      C   real*8 WWLD(NBANK)
97:      C   real*8 WSLD(NBANK,NGSP)
98:      c-beff
99:      C
100:      C**** TALLY ARRAY
101:      C
102:      C   real*8 WCNTR(*),  NCNTR(*)
103:      C
104:      C**** WORK AREA
105:      C
106:      C   integer IWK1(NBANK),  IWK2(NBANK),  IWK8(NBANK),  IWK9(NBANK),
107:      C   &                 IWK6(NBANK),  IWK7(NBANK),  IWK10(NBANK),  IWK11(NBANK),
108:      C   &                 IWK12(NBANK),  IWK13(NBANK)
109:      C   real WK3(NBANK),  WK4(NBANK),  WK5(NBANK),  R(NBANK*6)
110:      C
111:      C   0...NBANK...2NBANK...3NBANK...4NBANK...5NBANK...6NBANK...7NBANK...8NBANK
112:      C   IWK1  IWK2  IWK3  IWK4  IWK5  IWK6  IWK7  ....
113:      C   IWK13 : R IN NTHSCT
114:      C   R      : R(NBANK+1) IN NTHSCT
115:      C****
116:      C   parameter( PI      = 3.141592653589793200D+00 )
117:      C   parameter( PIH     = 0.5D0*PI )
118:      C   parameter( SMALL   = 1.0E-35 )
119:      C
120:      C
121:      C**** DETERMINE THE SUBSECTION USED
122:      C.... MAKE TEMPORARY STACK OF PARTICLE HAVING SUBSECTION > 1
123:      C   LSEF5( IWK11(J) )
124:      C   IWK12(I) : SUBSECTION NO. USED FOR LSEF5(I) PARTICLE
125:      C   AT FIRST STAGE, THIS IS SET AS NK5
126:      C   IWK13(I) : LSTF5
127:      C   IWK8 (J) : POINTER OF ENERGY GRIDS
128:      C   IWK9 (J) : NEF5
129:      C
130:      NN      = 0

```

src/mvp/sef5n.f

```

131:      N6      = 0
132:      *VOCL LOOP,NOVREC
133:      do 100 I = 1, NSEF5
134:         IWK2I = ABS(IWK2(I))
135:         IWK12(I) = KLIB2(IWK1(I),IWK2I,8)
136:         IWK13(I) = KLIB2(IWK1(I),IWK2I,13)
137:         if ( IWK12(I).ge.2 ) then
138:            NN      = NN + 1
139:            IWK11(NN) = I
140:            IWK8(NN)  = IWK12(I) + IWK13(I)
141:            IWK9(NN)  = KLIB2(IWK1(I),IWK2I,9)
142:            WK3(NN) = EIN(I)
143:         end if
144:         if ( IWK12(I).le.-2 ) N6 = N6 + 1
145:         if ( IWK12(I).lt.0 ) then
146:            IWK12(I) = ABS(IWK12(I))
147: C ..... IWK12(I) should be 1 because this is a used subsection ..
148:         end if
149:      100 continue
150:      check
151:      if ( N6.gt.0 ) then
152:         write(IOW,*) ' I found some nuclides with multiple',
153:         &           ' subsections for file 6 data: stop !!!'
154:         stop
155:      end if
156: C
157: C
158: C**** THE CASE WHERE MANY SUBSECTIONS ARE GIVEN
159: C
160:      if ( NN.gt.0 ) then
161: C
162: C.... SELECTION OF USED TABLE ----> IWK10
163: C IWK8 : POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
164: C IWK9 : NUMBER OF INCIDENT ENERGY (NEF5)
165: C WK3  : INCIDENT NEUTRON ENERGY
166: C WK4  : WORK AREA IN BSDEC3
167: C
168:         call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NN )
169: C
170:         call RANU2( IRAND, R, NN*3, ICON )
171:         NN2      = NN*2
172: C
173:      *VOCL LOOP,NOVREC
174:      do 110 J = 1, NN
175:         I      = IWK11(J)
176:         if ( IWK10(J).eq.0 ) IWK10(J) = 1
177:         L2     = IWK8(J) + IWK10(J)
178:         L1     = L2 - 1
179: C..... LINEAR INTERPOLATION
180:         X1     = (CX(L1)-WK3(J)) / (CX(L1)-CX(L2))
181:         if ( X1.lt.0.0 ) X1 = 0.0
182:         if ( X1.gt.1.0 ) X1 = 1.0
183:
184: C/#IF ROUNDOFF(NEAREST)
185:         IT     = MIN(1,INT(R(J)+X1)) + IWK10(J)
186: C/#ELSE
187:         *      R1      = R(J) + X1
188:         *      IT     = IWK10(J) + R1
189: C/#ENDIF
190:
191: C..... BMC SAMPLING
192:         NKF5    = IWK12(I)
193:         L1     = IWK8(J) + IWK9(J)*2 + 3*NKF5*(IT-1) - 1
194:
195: C/#IF ROUNDOFF(NEAREST)

```

```

196:         L2     = MIN( INT(NKF5*R(NN+J)) + 1, NKF5 )
197: C/#ELSE
198:         *      L2     = NKF5*R(NN+J) + 1
199: C/#ENDIF
200:         L3     = L1 + L2
201:
202: C/#IF ROUNDOFF(NEAREST)
203:         L4     = 2*L2
204:         L4     = MIN(L4,INT(L4+R(NN2+J)-CX(L3))) + L1 + NKF5
205: C/#ELSE
206:         *      R2     = R(NN2+J) - CX(L3)
207:         *      L4     = L1 + NKF5 + 2*L2 + R2
208: C/#ENDIF
209:
210:         IWK12(I) = ICX(L4)
211:      110 continue
212: C
213:      end if
214: C
215: C.... INCREASE NEUTRONS BY (N,MN),M>4 REACTIONS .....
216: C YIELD DATA TAKEN FROM FILE 6 OF ENDF-B6
217: C
218:      if ( NF6.gt.0 ) then
219:         NN      = 0
220:         do 120 I = NF5 + 1, NSEF5
221:            if ( IWK2(I).lt.0 ) then
222:               NN      = NN + 1
223:               IWK11(NN) = I
224:               IWK2(I) = ABS(IWK2(I))
225:               IWK8(NN) = IWK12(I) + IWK13(I)
226:               IWK9(NN) = KLIB2(IWK1(I),IWK2(I),9)
227:               WK3(NN) = EIN(I)
228:            end if
229:         120 continue
230: C
231:         if ( NN.gt.0 ) then
232: C
233:             call BSDEC3( CX, IWK9, WK4, IWK8, WK3, IWK10, NN )
234:             call RANU2( IRAND, R, NN, ICON )
235: C
236:             NF67T = NSEF5 - NF60
237: C
238:             NSPLTT = 0
239:             ND60   = 0
240:             ND6T   = 0
241:             NDEAD0 = NDEAD
242:      *VOCL LOOP,NOVREC
243:             do 130 J = 1, NN
244:                if ( IWK10(J).eq.0 ) IWK10(J) = 1
245:                L2     = IWK8(J) + IWK10(J)
246:                L1     = L2 - 1
247: C..... LINEAR INTERPOLATION
248:                X1     = (CX(L1)-WK3(J)) / (CX(L1)-CX(L2))
249:                if ( X1.lt.0.0 ) X1 = 0.0
250:                if ( X1.gt.1.0 ) X1 = 1.0
251:                L2     = L2 + IWK9(J)*2
252:                L1     = L2 - 1
253:                YIELD  = CX(L1)*(1.0-X1) + CX(L2)*X1
254:                IWK9(J) = YIELD + R(J)
255:                IWK9(J) = IWK9(J) - 1
256:                if ( YIELD.lt.1.0 ) then
257:                   LSEF5J = LSEF5(IWK11(J))
258:                   IP     = LSOL(LSEF5J)
259: C ..... WEIGHT REDUCTION .....
260:                   if ( WK3(J).ge.EWCUT ) then

```

src/mvp/sef5n.f

```

261:      WWW(IP) = WWW(IP)*YIELD
262:      IWK9(J) = 0
263: C ..... ANALOG ABSORPTION .....
264:      else if ( IWK9(J).lt.0 ) then
265: C          ND6T = ND6T + 1
266:          NDEAD = NDEAD + 1
267: C ..... PREPARATION TO REMOVE DEAD PARTICLES FROM .....
268: C          LSEF5, IWK1, IWK2, EIN, CTH, LSCOL
269: C          IWK12, IWK13, IWK9, IWK11
270:          LSEF5(IWK11(J)) = -1
271:          LSCOL(LSEF5(J)) = -1
272:          WCNTR(23) = WCNTR(23) + WWW(IP)
273:          if ( IWK11(J).gt.NF67T ) ND60 = ND60 + 1
274: C###<2007/03/14:PN3:
275:      if ( KIBNK(17).ne.0 ) IBNK(IP,KIBNK(17)) = 0
276:      if ( KIBNK(18).ne.0 ) IBNK(IP,KIBNK(18)) = 0
277:      if ( KIBNK(19).ne.0 ) IBNK(IP,KIBNK(19)) = 0
278: C###>
279: C###<2007/03/14:PN4:
280:      if ( JPHNU.ne.0 ) KPNFG(IP) = 0
281: C###>
282:      end if
283:      else
284:          NSPLTT = NSPLTT + IWK9(J)
285:      end if
286:      continue
287: 130
288: C          ND6T = NDEAD - NDEAD0
289: C          NCNTR(23) = NCNTR(23) + ND6T
290: C          NN67 = 0
291: C          NNN = 0
292: C          *VOCL LOOP,NOVREC
293:          do 140 J = 1, NN
294:              if ( IWK9(J).gt.0 ) then
295:                  NNN = NNN + 1
296:                  IWK9(NNN) = IWK9(J)
297:                  IWK11(NNN) = IWK11(J)
298:                  if ( IWK11(NNN).le.NF67T ) NN67 = NN67 + 1
299:              end if
300:          continue
301:          NN = NNN
302: 140
303: C      .... REMOVE DEAD PARTICLES FROM
304: C          LSCOL, LSEF5, IWK1, IWK2, EIN, CTH, IWK12, IWK13
305: C          if ( ND6T.gt.0 ) then
306:              do 150 I = 1, NCOLS
307:                  IWK6(I) = 0
308:              continue
309:          *VOCL LOOP,NOVREC
310:              do 160 J = 1, NSEF5
311:                  if ( LSEF5(J).gt.0 ) then
312:                      IWK6(LSEF5(J)) = J
313:                  end if
314:              continue
315:              N = 0
316:          *VOCL LOOP,NOVREC
317:              do 170 I = 1, NCOLS
318:                  if ( LSCOL(I).gt.0 ) then
319:                      N = N + 1
320:                      LSCOL(N) = LSCOL(I)
321:                      CTH(N) = CTH(I)

```

```

322:      IWK7(N) = I
323:      end if
324:      continue
325: 170
326:      NCOLS = N
327:      *VOCL LOOP,NOVREC
328:          do 180 N = 1, NCOLS
329:              I = IWK7(N)
330:              if ( IWK6(I).gt.0 ) LSEF5(IWK6(I)) = N
331:              continue
332:          180
333: C          do 190 I = NF5 + 1, NSEF5
334:              IWK6(I) = 0
335:              continue
336:          *VOCL LOOP,NOVREC
337:              do 200 J = 1, NN
338:                  IWK6(IWK11(J)) = J
339:              continue
340:          C          N = NF5
341:          *VOCL LOOP,NOVREC
342:              do 210 I = NF5 + 1, NSEF5
343:                  if ( LSEF5(I).lt.0 ) then
344:                      N = N + 1
345:                      LSEF5(N) = LSEF5(I)
346:                      IWK1(N) = IWK1(I)
347:                      IWK2(N) = IWK2(I)
348:                      IWK12(N) = IWK12(I)
349:                      IWK13(N) = IWK13(I)
350:                      EIN(N) = EIN(I)
351:                      IWK7(N) = I
352:                  end if
353:              continue
354:          CHECK
355:          if ( N.ne.NSEF5-ND6T ) then
356:              write(IOW,*) ' something wrong in treatment of',
357:                  ' yield data in SEF5N'
358:          end if
359:          NSEF5 = N
360:          *VOCL LOOP,NOVREC
361:              do 220 N = NF5 + 1, NSEF5
362:                  I = IWK7(N)
363:                  if ( IWK6(I).gt.0 ) IWK11(IWK6(I)) = N
364:                  continue
365:          C          NF6 = NF6 - ND6T
366:          NF60 = NF60 - ND60
367:          NF67T = NSEF5 - NF60
368:          end if
369:          C          if ( NSPLTT.gt.0 ) then
370:              if ( NSPLTT.le.NDEAD ) then
371:                  N67AD = 0
372:                  if ( NN67.gt.0 ) then
373:                      MA = 0
374:                      do 230 J = 1, NN67
375:                          MA = MAX(MA,IWK9(J))
376:                      continue
377:                      do 240 K = 1, MA
378:                          do 250 J = 1, NN67
379:                              if ( IWK9(J).ge.K ) then
380:                                  N67AD = N67AD + 1
381:                                  IWK10(N67AD) = IWK11(J)
382:                              end if
383:                          continue
384:                      240
385:                  250

```

src/mvp/sef5n.f

```

391: 250      continue
392:      end if
393:      if ( NN67.lt.NN ) then
394:          MA      = 0
395:          do 260 J = NN67 + 1, NN
396:              MA      = MAX(MA,IWK9(J))
397: 260      continue
398:          N      = N67AD
399:          do 280 K = 1, MA
400:              do 270 J = NN67 + 1, NN
401:                  if ( IWK9(J).ge.K ) then
402:                      N      = N + 1
403:                      IWK10(N) = IWK11(J)
404:                  end if
405:              continue
406: 280      continue
407:      end if
408: C
409: C ..... INSUFFICIENT BANK SIZE FOR (N,MN) REACTIONS
410: C
411: C
412:      else
413:          NSPLTT = 0
414:          N      = 0
415:          J1     = 0
416:          N67AD  = 0
417:          do 300 J = 1, NN
418:              K      = IWK9(J)
419:              NSPLTT = NSPLTT + K
420:              if ( NSPLTT.gt.NDEAD ) go to 310
421:              J1     = J
422:              do 290 II = 1, K
423:                  N      = N + 1
424:                  IWK10(N) = IWK11(J)
425: 290      continue
426:              if ( J.le.NN67 ) N67AD = N
427: 300      continue
428:              go to 330
429: C
430: C
431: 310      NSPLTT = N
432:          if ( J1.lt.NN ) then
433: *VOCL LOOP,NOVREC
434:          do 320 J = J1 + 1, NN
435:              I      = LSEF5(IWK11(J))
436:              IP      = LSCOL(I)
437: C
438:              WCNTR(25) = WCNTR(25) + WWW(IP)*IWK9(J)
439: C
440:              WWW(IP) = WWW(IP)*(IWK9(J)+1)
441: 320      continue
442:          end if
443: C
444: 330      continue
445:      end if
446: C
447:      NDEAD  = NDEAD - NSPLTT
448: *VOCL LOOP,NOVREC
449:      do 340 J = 1, NSPLTT
450:          J0      = IWK10(J)
451:          I0      = LSEF5(J0)
452:          IWK8(J) = LSCOL(I0)
453:          IWK9(J) = LSDED(NDEAD+J)
454:          NN      = NCOLS + J
455:          LSCOL(NN) = IWK9(J)

```

```

456:          if ( J0.gt.NF67T ) IWK10(J) = IWK10(J) + N67AD
457: 340      continue
458: C
459:          if ( N67AD.gt.0 ) then
460: *VOCL LOOP,NOVREC
461:          do 350 I = NSEF5, NF67T + 1, -1
462:              LSEF5(I+N67AD) = LSEF5(I)
463:              IWK1(I+N67AD) = IWK1(I)
464:              IWK2(I+N67AD) = IWK2(I)
465:              IWK12(I+N67AD) = IWK12(I)
466:              IWK13(I+N67AD) = IWK13(I)
467:              EIN(I+N67AD) = EIN(I)
468: 350      continue
469: *VOCL LOOP,NOVREC
470:          do 360 J = 1, N67AD
471:              LSEF5(J+NF67T) = NCOLS + J
472:              IWK1(J+NF67T) = IWK1(IWK10(J))
473:              IWK2(J+NF67T) = IWK2(IWK10(J))
474:              IWK12(J+NF67T) = IWK12(IWK10(J))
475:              IWK13(J+NF67T) = IWK13(IWK10(J))
476:              EIN(J+NF67T) = EIN(IWK10(J))
477: C
478:              IWK6(J) = KLIB2(IWK1(J+NF67T),IWK2(J+NF67T),12)
479:              IWK7(J) = KLIB2(IWK1(J+NF67T),IWK2(J+NF67T),7)
480: 360      continue
481: C
482: C .....SCATTERING ANGLE COSINE .....
483: C
484: C      IWK6 : POINTER OF INCIDENT ENERGY FOR WHICH A.D. ARE GIVEN
485: C      IWK7 : NUMBER OF INCIDENT ENERGY
486: C      EIN  : INCIDENT NEUTRON ENERGY
487: C
488:          call BSDEC3( CX, IWK7, WK4, IWK6, EIN(NF67T+1), IWK11,
489:              &
490:              call RANU2( IRAND, R, N67AD*3, ICON )
491:              N2      = N67AD*2
492: C
493: *VOCL LOOP,NOVREC
494:          do 370 J = 1, N67AD
495:              if ( IWK11(J).eq.0 ) IWK11(J) = 1
496:              L2      = IWK6(J) + IWK11(J)
497:              L1      = L2 - 1
498: C
499:              INTE     = ICX(L2+IWK7(J))
500:              INTE0    = ICX(L2+IWK7(J))
501: C
502:              if ( INTE.eq.5 .or. INTE.eq.3 ) then
503: C..... LOG      INTERPOLATION
504:                  X1      = LOG(CX(L1)/EIN(J+NF67T)) /
505:                  &
506:                  LOG(CX(L1)/CX(L2))
507: C..... LINEAR INTERPOLATION
508:                  X1      = (CX(L1)-EIN(J+NF67T)) / (CX(L1)-CX(L2))
509:              end if
510:              if ( X1.lt.0.0 ) X1 = 0.0
511:              if ( X1.gt.1.0 ) X1 = 1.0
512: C/#IF ROUNDOFF(NEAREST)
513:              IT      = MIN(1,INT(R(J)+X1)) + IWK11(J)
514:              L3      = MIN(
515:                  &
516:                  INT(KLIB1( IWK1(J+NF67T),7)*R(N67AD
517:                  &
518:                  +J))+1,KLIB1(IWK1(J
519:                  &
520:                  +NF67T),7)) + IWK6(J) + IWK7(J)*2
521:                  + KLIB1(IWK1(J+NF67T),20)*(IT-1)
522:                  XMU    = (CX(L3-1)-CX(L3))*R(N2+J) + CX(L3)

```

src/mvp/sef5n.f

```

521: C/#ELSE
522: *          R1      = R(J) + X1
523: *          IT      = IWK11(J) + R1
524: *          R2      = R(N67AD+J)*KLIB1(IWK1(J+N67T),7)
525: *          L3      = R2
526: *          &      + IWK6(J)+IWK7(J)*2
527: *          &      + KLIB1(IWK1(J+N67T),20)*(IT-1)
528: *          XMU     = (CX(L3+1)-CX(L3))*R(N2+J) + CX(L3)
529: C/#ENDIF
530:          CTH(LSEF5(J+N67T)) = XMU
531: 370      continue
532: C
533:      end if
534: C
535:      if ( N67AD.lt.NSPLTT ) then
536: *VOCL LOOP,NOVREC
537:      do 380 J = N67AD + 1, NSPLTT
538:          LSEF5(J+NSEF5) = NCOLS + J
539:          IWK1(J+NSEF5) = IWK1(IWK10(J))
540:          IWK2(J+NSEF5) = IWK2(IWK10(J))
541:          IWK12(J+NSEF5) = IWK12(IWK10(J))
542:          IWK13(J+NSEF5) = IWK13(IWK10(J))
543:          EIN(J+NSEF5) = EIN(IWK10(J))
544: 380      continue
545:      end if
546: C
547:          NSEF5 = NSEF5 + NSPLTT
548:          NF6 = NF6 + NSPLTT
549:          NF60 = NF60 + NSPLTT - N67AD
550: C
551:          NCOLS = NCOLS + NSPLTT
552: C
553: C
554: *VOCL LOOP,NOVREC
555:      do 390 I = 1, NSPLTT
556:          AAA(IWK9(I)) = AAA(IWK8(I))
557:          BBB(IWK9(I)) = BBB(IWK8(I))
558:          CCC(IWK9(I)) = CCC(IWK8(I))
559:          XXX(IWK9(I)) = XXX(IWK8(I))
560:          YYY(IWK9(I)) = YYY(IWK8(I))
561:          ZZZ(IWK9(I)) = ZZZ(IWK8(I))
562:          WWW(IWK9(I)) = WWW(IWK8(I))
563:          EEE(IWK9(I)) = EEE(IWK8(I))
564:          KKP(IWK9(I)) = KKP(IWK8(I))
565:          IZZ(IWK9(I)) = IZZ(IWK8(I))
566:          TTT(IWK9(I)) = TTT(IWK8(I))
567:          KLSF(IWK9(I)) = 0
568:
569:          WCNTR(25) = WCNTR(25) + WWW(IWK9(I))
570:
571:          if ( JLATT.ne.0 ) LEVL(IWK9(I)) = LEVL(IWK8(I))
572:          if ( JIMPT.ne.0 ) XIM(IWK9(I)) = XIM(IWK8(I))
573: CCCCC      if ( JTLLT.ne.0 ) IBREG(IWK9(I)) = IBREG(IWK8(I))
574:          IBREG(IWK9(I)) = IBREG(IWK8(I))
575:          if ( JTIME.ne.0 ) ITT(IWK9(I)) = ITT(IWK8(I))
576:
577:          if ( JPERT.ne.0 ) then
578:
579:              do IPT = 1, NPTDS
580:                  do IOD = 1, NORDD
581:                      WWD(IWK9(I),IPT,IOD) = WWD(IWK8(I),IPT,IOD)
582:                  end do
583:                  WWD2(IWK9(I),IPT) = WWD2(IWK8(I),IPT)
584:                  do ISP = 0, NGSP
585:                      do IOP = 1, NOPS

```

```

586:                      WSD(IWK9(I),ISP,IPT,IOP) =
587:                      &      WSD(IWK8(I),ISP,IPT,IOP)
588:                  end do
589:              end do
590:          WWT(IWK9(I),IPT) = WWT(IWK8(I),IPT)
591:      end do
592:
593:      do IPT = 1, NPTCS
594:          WWR(IWK9(I),IPT) = WWR(IWK8(I),IPT)
595:          do ISP = 0, NGSP
596:              WSC(IWK9(I),ISP,IPT) = WSC(IWK8(I),ISP,IPT)
597:          end do
598:      end do
599: c+beff2
600:      if (NPTBE.gt.0) then
601:          WWB(IWK9(I)) = WWB(IWK8(I))
602:          do ISP = 0, NGSP
603:              WSB(IWK9(I),ISP) = WSB(IWK8(I),ISP)
604:          end do
605:          WWBD(IWK9(I)) = WWBD(IWK8(I))
606:          WWLD(IWK9(I)) = WWLD(IWK8(I))
607:          do ISP = 1, NGSP
608:              WSD(IWK9(I),ISP) = WSD(IWK8(I),ISP)
609:              WSLD(IWK9(I),ISP) = WSLD(IWK8(I),ISP)
610:          end do
611:      end if
612: c-beff2
613:      end if
614:
615: 390      continue
616:
617:          NCNTR(25) = NCNTR(25) + NSPLTT
618: C
619:          if ( JLATT.ne.0.and.NSPLTT.gt.0 ) then
620:              do 410 K = 1, NEST
621: *VOCL LOOP,NOVREC
622:              do 400 I = 1, NSPLTT
623:                  LZZ(IWK9(I),K) = LZZ(IWK8(I),K)
624:                  LPOS(IWK9(I),K) = LPOS(IWK8(I),K)
625:                  if ( JHLAT.ne.0 ) LCRS(IWK9(I),K) =
626:                  &      LCRS(IWK8(I),K)
627:              continue
628:          410      continue
629:          end if
630:          if ( JTLLT.ne.0.and.NSPLTT.gt.0 ) then
631:              do 430 K = 0, NEST
632: *VOCL LOOP,NOVREC
633:              do 420 I = 1, NSPLTT
634:                  IBSPC(IWK9(I),K) = IBSPC(IWK8(I),K)
635:              420      continue
636:          430      continue
637:          end if
638: C
639: C .... optional bank parameters
640: C
641:          do 450 K = 1, KDBNK(0)
642: *VOCL LOOP,NOVREC
643:          do 440 I = 1, NSPLTT
644:              DBNK(IWK9(I),K) = DBNK(IWK8(I),K)
645:          440      continue
646:          450      continue
647:          do 470 K = 1, KIBNK(0)
648: *VOCL LOOP,NOVREC
649:          do 460 I = 1, NSPLTT
650:              IBNK(IWK9(I),K) = IBNK(IWK8(I),K)

```


src/mvp/sef5n.f

```

651: 460          continue
652: 470          continue
653: C
654:         end if
655:     end if
656: end if
657: C
658: C.... DETERMINE TABLE USED ----> IWK10(I) = INTERVAL
659: C IWK8 : POINTER OF INCIDENT ENERGIES IN SUBSECTION
660: C IWK9 : NUMBER OF INCIDENT ENERGIES
661: C IWK12: SUBSECTION NUMBER USED
662: C IWK13: POINTER TO SUBSECTION USED
663: C EIN  : INCIDENT NEUTRON ENERGY
664: C
665: do 480 I = 1, NSEF5
666:     IWK13(I) = ICX(IWK13(I)+IWK12(I)-1)
667:     IWK8(I) = IWK13(I) + 2
668:     IWK9(I) = ICX(IWK13(I)+1)
669: 480 continue
670: C
671:     call BSDEC3( CX, IWK9, WK4, IWK8, EIN, IWK10, NSEF5 )
672: C
673: C..... FILE 6 without FILE 4 DATA ( NF60 )
674: C
675:     N61 = 0
676:     N66 = 0
677: *VOCL LOOP,NOVREC
678: do 490 I = NSEF5 - NF60 + 1, NSEF5
679:     if ( ICX(IWK13(I)).eq.61 ) then
680:         N61 = N61 + 1
681:         IWK11(N61) = I
682:     else if ( ICX(IWK13(I)).eq.66 ) then
683:         N66 = N66 + 1
684:         IWK12(N66) = I
685:     end if
686: 490 continue
687: C
688: C..... LF=61 : Kalbach-87 DENSITY FUNCTION
689: C
690:     if ( N61.gt.0 ) then
691:         call RANU2( IRAND, R, N61*6, ICON )
692:         N2 = N61*2
693:         N3 = N61*3
694:         N4 = N61*4
695:         N5 = N61*5
696: C
697: *VOCL LOOP,NOVREC
698: do 500 J = 1, N61
699:     I = IWK11(J)
700:     IP = LSCOL(LSEF5(I))
701: C
702:     if ( IWK10(I).eq.0 ) IWK10(I) = 1
703:     L2 = IWK8(I) + IWK10(I)
704:     L1 = L2 - 1
705: C     INTE = ICX(L2+IWK9(I))
706: C
707:     INTE0 = ICX(L2+IWK9(I))
708:     INTE1 = INTE0 / 10
709:     INTE = INTE0 - INTE1*10
710: C
711: C..... LOG INTERPOLATION
712: C
713:     if ( INTE.eq.5 .or. INTE.eq.3 ) then
714:         X1 = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
715: C

```

```

716: C..... LINEAR INTERPOLATION
717: C
718:     else
719:         X1 = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
720:     end if
721:     if ( X1.lt.0.0 ) X1 = 0.0
722:     if ( X1.gt.1.0 ) X1 = 1.0
723: C
724:         L1 = IWK8(I) + IWK9(I)*2
725: C
726: C/#IF ROUNDOFF(NEAREST)
727:     IT = MIN(1,INT(R(J)+X1)) + IWK10(I)
728: C/#ELSE
729:     R1 = R(J) + X1
730:     IT = IWK10(I) + R1
731: C/#ENDIF
732: C
733: C..... BMC SAMPLING USING 3 RANDOM NUMBERS
734:     LSTF5E = ICX(L1+IT-1)
735:     NEP = ICX(LSTF5E)
736:     LSTF5E = LSTF5E + 1
737:     ND = ICX(LSTF5E)
738: C
739: C/#IF ROUNDOFF(NEAREST)
740:     LL2 = MIN( INT(NEP*R(N61+J)) + 1, NEP )
741: C/#ELSE
742:     LL2 = NEP*R(N61+J) + 1
743: C/#ENDIF
744: C
745:     LL3 = LSTF5E + LL2
746: C
747: C/#IF ROUNDOFF(NEAREST)
748:     LL4 = 2*LL2
749:     LL4 = MIN(LL4,INT(LL4+R(N3+J)-CX(LL3))) + LSTF5E + NEP
750: C/#ELSE
751:     R2 = R(N3+J) - CX(LL3)
752:     LL4 = LSTF5E + NEP + 2*LL2 + R2
753: C/#ENDIF
754: C
755:     LL5 = ICX(LL4)
756:     LLL = LSTF5E + 3*NEP
757:     L3 = LLL + LL5
758:     L4 = L3 + NEP + 1
759:     L5 = L4 + NEP + 1
760:     L6 = L5 + NEP + 1
761: C
762:     E0 = CX(L3)
763:     E1 = CX(L3+1)
764:     P0 = CX(L4)
765:     P1 = CX(L4+1)
766:     R0 = CX(L5)
767:     R1 = CX(L5+1)
768:     A0 = CX(L6)
769:     A1 = CX(L6+1)
770: C
771:     if ( LL5.le.ND ) then
772: C..... CONTINUOUS SPECTRA
773:         RRR = (P0+P1)*R(N2+J)
774:         RRR = RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
775: C
776:         PRECF = R1 + (R0-R1)*RRR
777:         ANGDS = A1 + (A0-A1)*RRR
778: C..... DISCRETE SPECTRA
779:         RRR = 0.0
780:         PRECF = R1

```

src/mvp/sef5n.f

```

781:      ANGDS = A1
782:      end if
783: C..... SECONDARY ENERGY AND SCATTERING ANGLE COSINE IN CM SYSTEM
784:      EEE(IP) = E1 + (E0-E1)*RRR
785:      if ( R(N4+J).gt.PRECF ) then
786:          T = (2.*R(N5+J)-1.0)*SINH(ANGDS)
787:          CTH(LSEF5(I)) = LOG(T+SQRT(T*T+1.)) /ANGDS
788:      else
789:          CTH(LSEF5(I)) =
790:      &      LOG(R(N5+J)*EXP(ANGDS)+(1.-R(N5
791:      &      +J))*EXP(-ANGDS)) /ANGDS
792:      end if
793: C
794:      if ( INTE1.gt.0 .and. LL5.le.ND ) then
795:          LLE = LLE + 1
796:          if ( IT.eq.IWK10(I) ) then
797:              NEP1 = NEP
798:              ND1 = ND
799:              LLE1 = LLE
800:              LSTF5T = ICX(L1+IT)
801:              NEP2 = ICX(LSTF5T)
802:              ND2 = ICX(LSTF5T+1)
803:              LLE2 = LSTF5T + 3*NEP2 + 2
804:          else
805:              NEP2 = NEP
806:              ND2 = ND
807:              LLE2 = LLE
808:              LSTF5T = ICX(L1+IT-2)
809:              NEP1 = ICX(LSTF5T)
810:              ND1 = ICX(LSTF5T+1)
811:              LLE1 = LSTF5T + 3*NEP1 + 2
812:          end if
813:          if ( INTE1.eq.2 ) then
814: C ... linear interpolation is assumed
815:              EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
816:              EEE(IP) = EEE(IP) * EMAX / CX(LLE)
817:          else if ( INTE1.eq.1 ) then
818: C ... linear interpolation is assumed
819:              EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
820:              EMIN = CX(LLE1+ND1)
821:      &      + (CX(LLE2+ND2)-CX(LLE1+ND1))*X1
822:              EEE(IP) = EMIN + (EMAX-EMIN)*
823:      &      (EEE(IP)-CX(LLE+ND))/(CX(LLE)-CX(LLE+ND))
824:          end if
825:      end if
826: C
827: 500 continue
828:      end if
829: C
830: C
831: C..... LF=66 : N-BODY PHASE SPACE DISTRIBUTION
832: C
833: C      IWK11 : NPSX
834: C      WK3 : EIMAX
835:      if ( N66.gt.0 ) then
836:          call RANU2( IRAND, R, N66*5, ICON )
837:          N2 = N66*2
838:          N3 = N66*3
839:          N4 = N66*4
840:      *VOCL LOOP,NOVREC
841:      do 510 J = 1, N66
842:          I = IWK12(J)
843:          LL = IWK8(I) + IWK9(I)*2
844:          APSX = CX(LL)
845:          IWK11(J) = ICX(LL+IWK9(I))

```

```

846:          ATW = XLIB1(IWK1(I),1)
847:          EIMAX = ATW*EIN(I) /(ATW+1.0) - XLIB2(IWK1(I),IWK2(I),1)
848:          WK3(J) = EIMAX*(APSX-1.0) /APSX
849: C
850:          R1 = SQRT(R(J))
851:          T1 = PIH*R(N66+J)
852:          X1 = (R1*COS(T1))**2
853:          S1 = (R1*SIN(T1))**2 + X1
854:          WK4(J) = -X1*LOG(S1) /S1 - LOG(R(N2+J))
855:          R2 = SQRT(R(N3+J))
856:          T2 = PIH*R(N4+J)
857:          X2 = (R2*COS(T2))**2
858:          S2 = (R2*SIN(T2))**2 + X2
859:          WK5(J) = -X2*LOG(S2) /S2
860: 510 continue
861: C
862:      call RANU2( IRAND, R, N66*5, ICON )
863:          N2 = N66*2
864:          N3 = N66*3
865:          N4 = N66*4
866: C
867:      *VOCL LOOP,NOVREC
868:      do 520 J = 1, N66
869:          I = IWK12(J)
870:          IP = LSCOL(LSEF5(I))
871:          if ( IWK11(J).eq.3 ) then
872:              P = R(J)
873:          else if ( IWK11(J).eq.4 ) then
874:              P = R(J)*R(N66+J)
875:          else
876:              P = R(J)*R(N66+J)*R(N2+J)*R(N3+J)
877:          end if
878: C..... SECONDARY ENERGY AND SCATTERING ANGLE COSINE IN CM SYSTEM
879:          EEE(IP) = WK3(J)*WK4(J) /(WK4(J)+WK5(J)-LOG(P))
880:          CTH(LSEF5(I)) = 2.*R(N4+J) - 1.0
881: 520 continue
882:      end if
883: C
884: C*** CONVERSION OF SECONDARY ENERGY AND ANGLE fro CM TO LAB SYSTEM
885: C
886:      *VOCL LOOP,NOVREC
887:      do 530 I = NSEF5 - NF60 + 1, NSEF5
888:          IP = LSCOL(LSEF5(I))
889:          A1 = XLIB1(IWK1(I),1) + 1.0
890:          ECM = EEE(IP)
891:          XCM = CTH(LSEF5(I))
892:          EEE(IP) = ECM + (EIN(I)+2.*A1*XCM*SQRT(EIN(I)*ECM)) /A1/A1
893:          CTH(LSEF5(I)) = XCM*SQRT(ECM/EEE(IP)) + SQRT(EIN(I)/EEE(IP))
894:      &      /A1
895: 530 continue
896: C
897: C..... LF=67 : TABULATED ANGLE-ENERGY DISTRIBUTION
898: C
899:          N67 = NF6 - NF60
900:          if ( N67.gt.0 ) then
901:              N67I = 0
902:              call RANU2( IRAND, R, N67, ICON )
903:      *VOCL LOOP,NOVREC
904:      do 540 J = 1, N67
905:          I = NF5 + J
906:          if ( IWK10(I).eq.0 ) IWK10(I) = 1
907:          L2 = IWK8(I) + IWK10(I)
908:          L1 = L2 - 1
909: C
910:          INTE0 = ICX(L2+IWK9(I))

```

src/mvp/sef5n.f

```

911:          INTE1  = INTE0 / 10
912:          INTE   = INTE0 - INTE1*10
913: C
914: C..... LOG    INTERPOLATION
915: C
916:          if ( INTE.eq.5 .or. INTE.eq.3 ) then
917:              X1   = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
918: C
919: C..... LINEAR INTERPOLATION
920: C
921:          else
922:              X1   = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
923:          end if
924:          if ( X1.lt.0.0 ) X1 = 0.0
925:          if ( X1.gt.1.0 ) X1 = 1.0
926: C
927:          L1      = IWK8(I) + IWK9(I)*2
928: C/#IF ROUNDOFF(NEAREST)
929:          IT      = MIN(1,INT(R(J)+X1)) + IWK10(I)
930: C/#ELSE
931: *              R1      = R(J) + X1
932: *              IT      = IWK10(I) + R1
933: C/#ENDIF
934:          LSTF5E = ICX(L1+IT-1)
935:          IWK8(I) = LSTF5E + KL1B1(IWK1(I),20) + 2
936:          IWK9(I) = ICX(IWK8(I)-1)
937:          WK3(I)  = CTH(LSEF5(I))
938: C
939:          IWK12(I) = INTE1
940:          if ( INTE1.gt.0 ) then
941:              N67I = N67I + 1
942:              IWK11(N67I) = I
943:              WK4(N67I) = WK3(I)
944:              if ( IT.eq.IWK10(I) ) then
945:                  LSTF5T = ICX(L1+IT)
946:                  IWK6(N67I) = LSTF5T + KL1B1(IWK1(I),20) + 2
947:                  IWK7(N67I) = ICX(IWK6(N67I)-1)
948:                  WK5(N67I) = X1
949:              else
950:                  LSTF5T = ICX(L1+IT-2)
951:                  IWK6(N67I) = LSTF5T + KL1B1(IWK1(I),20) + 2
952:                  IWK7(N67I) = ICX(IWK6(N67I)-1)
953:                  WK5(N67I) = 1. - X1
954:              end if
955:          end if
956: 540 continue
957: C
958:          NS      = NF5 + 1
959:          call BSINC3( CX, IWK9(NS), R, IWK8(NS), WK3(NS), IWK10(NS),
960: &                  N67 )
961: C
962:          call RANU2( IRAND, R, N67*4, ICON )
963:          N2      = N67*2
964:          N3      = N67*3
965: C
966: *VOCL LOOP,NOVREC
967:          do 550 J = 1, N67
968:              I    = NF5 + J
969:              if ( IWK10(I).eq.0 ) IWK10(I) = 1
970:              L2   = IWK8(I) + IWK10(I)
971:              L1   = L2 - 1
972:              L3   = IWK8(I) + IWK9(I) - 1
973:              INTA = ICX(IWK8(I)-2)
974: C
975: C..... STEP INTERPOLATION

```

```

976: C
977:          if ( INTA.eq.1 ) then
978:              LST2A = ICX(IWK10(I)+L3)
979: C
980: C..... LINEAR INTERPOLATION
981: C
982:          else
983:              X1   = (CX(L1)-WK3(I)) / (CX(L1)-CX(L2))
984:              if ( X1.lt.0.0 ) X1 = 0.0
985:              if ( X1.gt.1.0 ) X1 = 1.0
986: C
987: C/#IF ROUNDOFF(NEAREST)
988:              IT   = MIN(1,INT(R(J)+X1)) + IWK10(I)
989: C/#ELSE
990: *              R1   = R(J) + X1
991: *              IT   = IWK10(I) + R1
992: C/#ENDIF
993: C
994:              LST2A = ICX(IT+L3)
995:          end if
996: C
997:              NEP   = ICX(LST2A)
998: C
999: C/#IF ROUNDOFF(NEAREST)
1000:              LL2   = MIN( INT(NEP*R(N67+J))+1,NEP)
1001: C/#ELSE
1002: *              LL2   = NEP*R(N67+J) + 1
1003: C/#ENDIF
1004: C
1005:              LL3   = LST2A + LL2
1006: C
1007: C/#IF ROUNDOFF(NEAREST)
1008:              LL4   = 2*LL2
1009:              LL4   = MIN(LL4,INT(LL4+R(N2+J)-CX(LL3))) + LST2A + NEP
1010: C/#ELSE
1011: *              R2   = R(N3+J) - CX(LL3)
1012: *              LL4   = LST2A + NEP + 2*LL2 + R2
1013: C/#ENDIF
1014: C
1015:              L3     = LST2A + 3*NEP + ICX(LL4)
1016:              LLL     = LST2A + 3*NEP
1017:              L3      = LLL + ICX(LL4)
1018:              L4      = L3 + NEP + 1
1019: C
1020:              E0     = CX(L3)
1021:              E1     = CX(L3+1)
1022:              P0     = CX(L4)
1023:              P1     = CX(L4+1)
1024: C
1025:              IP     = LSCOL(LSEF5(I))
1026:              RRR     = (P0+P1)*R(N3+J)
1027:              &       = E1 + (E0-E1)*RRR/
1028:              &       (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1029: C ... INT>10, WK3:EMAX, R:EMIN
1030:              if ( IWK12(I).gt.0 ) then
1031:                  WK3(I) = CX(LL4+1)
1032:                  R(I)   = CX(LL4+NEP+1)
1033:              end if
1034: 550 continue
1035: C ... INT(E)>10
1036: C
1037:          if ( N67I.gt.0 ) then
1038:              NS      = NF5 + 1
1039:              call BSINC3( CX, IWK7, R(NBANK+1), IWK6, WK4,
1040: &                      IWK10(NS),N67I )

```

src/mvp/sef5n.f

```

1041:      call RANU2( IRAND, R(NBANK+1), N67I, ICON )
1042: C
1043: *VOCL LOOP,NOVREC
1044:      do 560 J = 1, N67I
1045:          JJ      = NF5 + J
1046:          if ( IWK10(JJ).eq.0 ) IWK10(JJ) = 1
1047:          L2      = IWK6(J) + IWK10(JJ)
1048:          L1      = L2 - 1
1049:          L3      = IWK6(J) + IWK7(J) - 1
1050:          INTA    = ICX(IWK6(J)-2)
1051: C
1052: C..... STEP INTERPOLATION
1053: C
1054:          if ( INTA.eq.1 ) then
1055:              LST2A = ICX(IWK10(JJ)+L3)
1056: C
1057: C..... LINEAR INTERPOLATION
1058: C
1059:          else
1060:              X1    = (CX(L1)-WK4(J)) / (CX(L1)-CX(L2))
1061:              if ( X1.lt.0.0 ) X1 = 0.0
1062:              if ( X1.gt.1.0 ) X1 = 1.0
1063: C
1064: C/#IF ROUNDOFF(NEAREST)
1065:              IT    = MIN(1,INT(R(NBANK+J)+X1)) + IWK10(JJ)
1066: C/#ELSE
1067:              R1    = R(NBANK+J) + X1
1068:              IT    = IWK10(JJ) + R1
1069: C/#ENDIF
1070: C
1071:              LST2A = ICX(IT+L3)
1072:          end if
1073: C
1074:          NEP      = ICX(LST2A)
1075:          LLE      = LST2A + 3*NEP + 1
1076: C
1077:          I        = IWK11(J)
1078:          IP       = LSCOL(LSEF5(I))
1079: C
1080:          if ( IWK12(I).eq.2 ) then
1081: C ... linear interpolation is assumed
1082:              EMAX = WK3(I) + (CX(LLE)-WK3(I))*WK5(J)
1083:              EEE(IP) = EEE(IP) * EMAX / WK3(I)
1084:          else
1085: C ... linear interpolation is assumed
1086:              EMAX = WK3(I) + (CX(LLE)-WK3(I))*WK5(J)
1087:              EMIN = R(I)
1088:              &    + (CX(LLE+NEP)-R(I))*WK5(J)
1089:              EEE(IP) = EMIN + (EMAX-EMIN)*
1090:              &    (EEE(IP)-R(I))/(WK3(I)-R(I))
1091:          end if
1092:      560      continue
1093:          end if
1094: C
1095:      end if
1096: C
1097: C**** SAMPLING OF OUTGOING ENERGY FROM DATA IN SUBSECTION IWK12(I)
1098: C MAKE TEMPORARY STACK, THESE ARRAYS ARE CHANGED DURING CALCULATION
1099: C      IWK8(NN) : TEMPORARY STACK FOR E' > E-U
1100: C      IWK9(NN) : LFF5S
1101: C      IWK10(NN) :
1102: C      IWK11(NN) :
1103: C      IWK12(NN) :
1104: C      WK3 (NN) :
1105: C      WK4 (NN) :

```

```

1106: C      WK5 (NN) :
1107: C
1108:      NSEF50 = NSEF5
1109:      NSEF5  = NF5
1110: C
1111:      call RANU2( IRAND, R, NSEF5*4, ICON )
1112: C
1113:      N2      = NSEF5*2
1114:      N3      = NSEF5*3
1115: C
1116:      NN      = 0
1117: C
1118: C ... "loop 480" in update list means the following loop.
1119: *VOCL LOOP,NOVREC
1120:      do 570 I = 1, NSEF5
1121:          IP    = LSCOL(LSEF5(I))
1122: C
1123:          if ( IWK10(I).eq.0 ) IWK10(I) = 1
1124:          L2    = IWK8(I) + IWK10(I)
1125:          L1    = L2 - 1
1126:          INTE  = ICX(L2+IWK9(I))
1127: C
1128:          INTE0 = ICX(L2+IWK9(I))
1129:          INTE1 = INTE0 / 10
1130:          INTE  = INTE0 - 10*INTE1
1131: C
1132: C..... LOG INTERPOLATION
1133: C
1134:          if ( INTE.eq.5 .or. INTE.eq.3 ) then
1135:              X1    = LOG(CX(L1)/EIN(I)) / LOG(CX(L1)/CX(L2))
1136: C
1137: C..... LINEAR INTERPOLATION
1138: C
1139:          else
1140:              X1    = (CX(L1)-EIN(I)) / (CX(L1)-CX(L2))
1141:          end if
1142:          if ( X1.lt.0.0 ) X1 = 0.0
1143:          if ( X1.gt.1.0 ) X1 = 1.0
1144: C
1145:          LFF5S    = ICX(IWK13(I))
1146:          L1        = IWK8(I) + IWK9(I)*2
1147: C
1148: C==== DEPEND ON LFF5S-VALUE
1149: C
1150: C.... TABULATED PROBABILITY (LF=1) USING 3 OR 4 RANDOM NUMBERS
1151:          if ( LFF5S.eq.1 ) then
1152: C
1153: C/#IF ROUNDOFF(NEAREST)
1154:              IT    = MIN(1,INT(R(I)+X1)) + IWK10(I)
1155: C/#ELSE
1156:              R1    = R(I) + X1
1157:              IT    = IWK10(I) + R1
1158: C/#ENDIF
1159: C
1160:              NBINE1 = KLIB1(IWK1(I),21)
1161: *VOCL STMT,IF(0)
1162:          if ( NBINE1.gt.1 ) then
1163: C
1164:              LLE    = NBINE1*2*(IT-1) + L1
1165:              LLE2    = NBINE1*2*IWK10(I) + L1
1166:              LLE1    = LLE2 - NBINE1*2
1167:              NEP     = NBINE1 - 1
1168:              NEP1    = NEP
1169:              NEP2    = NEP
1170: C/#IF ROUNDOFF(NEAREST)

```

src/mvp/sef5n.f

```

1171:          LKL1    = KLIB1(IWK1(I),8)
1172:          L3       = MIN(LKL1-1,INT(LKL1*R(NSEF5
1173:          &        +I))) + LLE
1174: C/#ELSE
1175: *          R22     = KLIB1(IWK1(I),8)*R(NSEF5+I)
1176: *          L3      = R22
1177: *          &      + LLE
1178: C/#ENDIF
1179:
1180:          L4       = L3 + NBINE1
1181:          else
1182: C..... BMC SAMPLING USING 4 RANDOM NUMBERS
1183:          LSTF5E   = ICX(L1+IT-1)
1184:          NEP      = ICX(LSTF5E)
1185:
1186: C/#IF ROUNDOFF(NEAREST)
1187:          LL2      = MIN(INT(NEP*R(NSEF5+I))+1,NEP)
1188: C/#ELSE
1189: *          LL2     = NEP*R(NSEF5+I) + 1
1190: C/#ENDIF
1191:
1192:          LL3      = LSTF5E + LL2
1193:
1194: C/#IF ROUNDOFF(NEAREST)
1195:          LL4      = 2*LL2
1196:          LL4      = MIN(LL4,INT(LL4+R(N3+I)-CX(LL3))) + LSTF5E +
1197:          &        NEP
1198: C/#ELSE
1199: *          R2      = R(N3+I) - CX(LL3)
1200: *          LL4     = LSTF5E + NEP + 2*LL2 + R2
1201: C/#ENDIF
1202:          LLL      = LSTF5E + 3*NEP
1203:          L3       = LLL + ICX(LL4)
1204:          L4       = L3 + NEP + 1
1205: C
1206:          LLE      = LLL + 1
1207:          if ( IT.eq.IWK10(I) ) then
1208:              NEP1   = NEP
1209:              LLE1   = LLE
1210:              LSTF5T = ICX(L1+IT)
1211:              NEP2   = ICX(LSTF5T)
1212:              LLE2   = LSTF5T + 3*NEP2 + 1
1213:          else
1214:              NEP2   = NEP
1215:              LLE2   = LLE
1216:              LSTF5T = ICX(L1+IT-2)
1217:              NEP1   = ICX(LSTF5T)
1218:              LLE1   = LSTF5T + 3*NEP1 + 1
1219:          end if
1220:          end if
1221:          E0       = CX(L3)
1222:          E1       = CX(L3+1)
1223:          P0       = CX(L4)
1224:          P1       = CX(L4+1)
1225:
1226: C
1227: C950719          IF( P0.EQ.P1 ) THEN
1228: C              EEE(IP) = E1 + (E0-E1)*R(N2+I)
1229: C          ELSE
1230: C              EEE(IP) = E1 + (E0-E1)/(P0-P1)
1231: C          +          * ( SQRT( P1*P1+(P0*P0-P1*P1)*R(N2+I) ) - P1 )
1232: C          ENDIF
1233: C
1234: C          ... the following formula is valid for both P0 = P1 and P0 >> P1.
1235: C

```

```

1236:          RRR      = (P0+P1)*R(N2+I)
1237:          EEE(IP) = E1 + (E0-E1)*RRR/
1238:          &        (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1239: C
1240:          if ( INTEL.eq.2 ) then
1241: C          ... linear interpolation is assumed
1242:          EMAX     = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
1243:          EEE(IP) = EEE(IP) * EMAX / CX(LLE)
1244:          else if ( INTEL.eq.1 ) then
1245: C          ... linear interpolation is assumed
1246:          EMAX     = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
1247:          EMIN     = CX(LLE1+NEP1)
1248:          &        + (CX(LLE2+NEP2)-CX(LLE1+NEP1))*X1
1249:          EEE(IP) = EMIN + (EMAX-EMIN)*
1250:          &        (EEE(IP)-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
1251:          end if
1252: C
1253:          end if
1254: C
1255: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
1256: C
1257:          if ( LFF5S.eq.5 ) then
1258:              L5     = L1 + IWK10(I)
1259:              THETA5 = CX(L5)*X1 + CX(L5-1)*(1.-X1)
1260:              LL      = L1 + IWK9(I)
1261:              NBINE   = KLIB1(IWK1(I),8)
1262: *VOCL STMT,IF(0)
1263:          if ( NBINE.gt.0 ) then
1264:              NBINE1 = KLIB1(IWK1(I),21)
1265:              U5     = CX(LL+NBINE1*2)
1266:
1267: C/#IF ROUNDOFF(NEAREST)
1268:          L6     = MIN(NBINE-1,INT(NBINE*R(I))) + LL
1269: C/#ELSE
1270: *          R23    = NBINE*R(I)
1271: *          L6     = R23 + LL
1272: C/#ENDIF
1273:
1274:          L7     = L6 + NBINE1
1275:          else
1276: C..... BMC SAMPLING USING 4 RANDOM NUMBERS
1277:          NBINE   = ICX(LL)
1278:          NBINE1  = 1
1279:
1280: C/#IF ROUNDOFF(NEAREST)
1281:          LL5     = MIN(INT(NBINE*R(I))+1,NBINE)
1282: C/#ELSE
1283: *          LL5    = NBINE*R(I) + 1
1284: C/#ENDIF
1285:
1286:          LL6     = LL + LL5
1287:
1288: C/#IF ROUNDOFF(NEAREST)
1289:          LL7     = 2*LL5
1290:          LL7     = MIN(LL7,INT(LL7+R(NSEF5+I)-CX(LL6))) + LL
1291:          &        + NBINE
1292: C/#ELSE
1293: *          R3     = R(NSEF5+I)-CX(LL6)
1294: *          LL7    = LL+NBINE + 2*LL5+R3
1295: C/#ENDIF
1296:
1297:          L6      = LL + 3*NBINE + ICX(LL7)
1298:          L7      = L6 + NBINE + 1
1299:          U5      = CX(LL+5*NBINE+3)
1300:          end if

```

src/mvp/sef5n.f

```

1301:      EHI5      = EIN(I) - U5
1302: C
1303:      E0        = CX(L6)
1304:      E1        = CX(L6+1)
1305:      P0        = CX(L7)
1306:      P1        = CX(L7+1)
1307: C
1308:      RRR       = (P0+P1)*R(N2+I)
1309:      X2        = E1 + (E0-E1)*RRR/
1310: &              (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1311: C
1312:      EEE(IP) = X2*THETA5
1313: C
1314:      IWK10(I)  = LL
1315:      IWK11(I)  = NBINE
1316:      IWK12(I)  = NBINE1
1317:      WK3(I)    = EHI5
1318:      WK4(I)    = THETA5
1319:      end if
1320: C
1321: C..... SIMPLE FISSION SPECTRUN (LF=7) USING 3 RANDOM NUMBERS
1322:      if ( LFF5S.eq.7 ) then
1323:      L8        = L1 + IWK10(I)
1324:      THETA7    = CX(L8)*X1 + CX(L8-1)*(1.-X1)
1325:      U7        = CX(L1+IWK9(I))
1326:      EHI7      = EIN(I) - U7
1327:      COSINE    = COS(PIH*R(I))
1328:      COS2      = COSINE*COSINE
1329: C
1330:      EEE(IP) = -THETA7*(LOG(R(NSEF5+I))+LOG(R(N2+I))*COS2)
1331: C
1332:      WK3(I)    = EHI7
1333:      WK4(I)    = THETA7
1334:      end if
1335: C
1336: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
1337:      if ( LFF5S.eq.9 ) then
1338:      L9        = L1 + IWK10(I)
1339:      THETA9    = CX(L9)*X1 + CX(L9-1)*(1.-X1)
1340:      U9        = CX(L1+IWK9(I))
1341:      EHI9      = EIN(I) - U9
1342: C
1343:      EEE(IP) = -THETA9*LOG(R(NSEF5+I)*R(I))
1344: C
1345:      WK3(I)    = EHI9
1346:      WK4(I)    = THETA9
1347:      end if
1348: C
1349: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
1350:      if ( LFF5S.eq.11 ) then
1351:      L10       = L1 + IWK10(I)
1352:      L11       = L10 + IWK9(I)
1353:      A         = CX(L10)*X1 + CX(L10-1)*(1.-X1)
1354:      B         = CX(L11)*X1 + CX(L11-1)*(1.-X1)
1355:      U11       = CX(L1+IWK9(I))*2)
1356:      EHI11     = EIN(I) - U11
1357:      COSI      = COS(PIH*R(I))
1358:      COSI2     = COSI*COSI
1359:      V2        = -A*(LOG(R(NSEF5+I))+LOG(R(N2+I))*COSI2)
1360:      XMU       = 2.*R(N3+I) - 1.
1361: C
1362:      EEE(IP) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
1363: C
1364:      WK3(I)    = EHI11
1365:      WK4(I)    = A
1366:      WK5(I)    = B
1367:      end if
1368: C
1369:      if ( LFF5S.ne.1.and.EEE(IP).gt.WK3(I) ) then
1370:      NN        = NN + 1
1371:      IWK8(NN)  = IP
1372:      IWK9(NN)  = LFF5S
1373:      if ( LFF5S.eq.5 ) then
1374:      IWK10(NN) = IWK10(I)
1375:      IWK11(NN) = IWK11(I)
1376:      IWK12(NN) = IWK12(I)
1377:      end if
1378:      WK3(NN)   = WK3(I)
1379:      WK4(NN)   = WK4(I)
1380:      if ( LFF5S.eq.11 ) then
1381:      WK5(NN)   = WK5(I)
1382:      end if
1383:      end if
1384:      570 continue
1385: C
1386:      if ( NN.gt.0 ) then
1387:      NTEST     = 1
1388: C
1389:      580 continue
1390: C
1391:      NNN       = NN
1392: C
1393:      call RANU2( IRAND, R, NNN*4, ICON )
1394: C
1395:      N2        = NNN*2
1396:      N3        = NNN*3
1397: C
1398:      NN        = 0
1399: C
1400: *VOCL LOOP,NOVREC
1401:      do 590 I = 1, NNN
1402:      LFF5S     = IWK9(I)
1403: C
1404: C===== DEPEND ON LFF5S-VALUE
1405: C
1406: C.... LF=5 USING 2 OR 3 RANDOM NUMBERS
1407:      if ( LFF5S.eq.5 ) then
1408:      if ( IWK12(I).gt.1 ) then
1409: C/#IF ROUNDOFF(NEAREST)
1410:      L6        = MIN(IWK11(I)-1,INT(IWK11(I)*R(I))) +
1411:      &          IWK10(I)
1412: C/#ELSE
1413:      R23       = IWK11(I)*R(I)
1414:      L6        = IWK10(I) + R23
1415: C/#ENDIF
1416:      L7        = L6 + IWK12(I)
1417:      else
1418: C/#IF ROUNDOFF(NEAREST)
1419:      LL5       = MIN(INT(IWK11(I)*R(I))+1,IWK11(I))
1420: C/#ELSE
1421:      LL5       = IWK11(I)*R(I) + 1
1422: C/#ENDIF
1423:      LL6       = IWK10(I) + LL5
1424:      LL7       = 2*LL5
1425: C/#IF ROUNDOFF(NEAREST)
1426:      LL7       = 2*LL5
1427: C/#IF ROUNDOFF(NEAREST)
1428:      LL7       = 2*LL5
1429: C/#IF ROUNDOFF(NEAREST)
1430:      LL7       = 2*LL5

```

src/mvp/sef5n.f

```

1431:          LL7      = MIN(LL7,INT(LL7+R(NNN+I)-CX(LL6))) +
1432:          &          IWK10(I) + IWK11(I)
1433: C/#ELSE
1434: *          R3      = R(NNN+I) - CX(LL6)
1435: *          LL7      = IWK10(I) + IWK11(I) + 2*LL5 + R3
1436: C/#ENDIF
1437:          L6      = IWK10(I) + 3*IWK11(I) + ICX(LL7)
1438:          L7      = L6 + IWK11(I) + 1
1439:          end if
1440:          E0      = CX(L6)
1441:          E1      = CX(L6+1)
1442:          P0      = CX(L7)
1443:          P1      = CX(L7+1)
1444: C
1445:          RRR      = (P0+P1)*R(N2+I)
1446:          X2      = E1 + (E0-E1)*RRR/
1447:          &          (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
1448:          EEE(IWK8(I)) = X2*WK4(I)
1449:          end if
1450: C
1451: C
1452: C..... SIMPLE FISSION SPECTRUN (LF=7) USING THREE RANDOM NUMBERS
1453:          if ( LFF5S.eq.7 ) then
1454:              COSINE = COS(PIH*R(1))
1455:              COS2    = COSINE*COSINE
1456: C
1457:              EEE(IWK8(I)) = -WK4(I)*
1458:              &          (LOG(R(NNN+I))+LOG(R(N2+I))*COS2)
1459:          end if
1460: C
1461: C
1462: C..... EVAPORATION SPECTRUN (LF=9) USING TWO RANDOM NUMBERS
1463:          if ( LFF5S.eq.9 ) then
1464: C
1465:              EEE(IWK8(I)) = -WK4(I)*LOG(R(I)*R(NNN+I))
1466:          end if
1467: C
1468: C
1469: C..... WATT FISSION SPECTRUN (LF=11) USING FOUR RANDOM NUMBERS
1470:          if ( LFF5S.eq.11 ) then
1471:              A      = WK4(I)
1472:              B      = WK5(I)
1473:              COSI    = COS(PIH*R(I))
1474:              COSI2    = COSI*COSI
1475:              V2      = -A*(LOG(R(NNN+I))+LOG(R(N2+I))*COSI2)
1476:              XMU      = 2.*R(N3+I) - 1.
1477: C
1478:              EEE(IWK8(I)) = V2 + A*XMU*SQRT(V2*B) + A*A*B*0.25
1479: C
1480:          end if
1481: C
1482:          if ( EEE(IWK8(I)).gt.WK3(I) ) then
1483:              NN      = NN + 1
1484:              IWK8(NN) = IWK8(I)
1485:              IWK9(NN) = IWK9(I)
1486:              WK3(NN) = WK3(I)
1487:              WK4(NN) = WK4(I)
1488:              if ( LFF5S.eq.11 ) WK5(NN) = WK5(I)
1489:              if ( LFF5S.eq.5 ) then
1490:                  IWK10(NN) = IWK10(I)
1491:                  IWK11(NN) = IWK11(I)
1492:                  IWK12(NN) = IWK12(I)
1493:              end if
1494:          end if
1495: C
1496: 590      continue
1497: C
1498:          if ( NN.gt.0 ) then
1499:              NTEST = NTEST + 1
1500:              if ( NTEST.le.300 ) then
1501:                  go to 580
1502:              else
1503:                  do 600 I = 1, NN
1504:                      EEE(IWK8(I)) = WK3(I)
1505:                  600      continue
1506:              end if
1507:          end if
1508:          end if
1509: C
1510:          NSEF5 = NSEF50
1511:          return
1512:          end

```

src/mvp/sef6n3.f

```

1:      subroutine SEF6N3( IOW, NF6 , ISEL , EEEJ , WW , COSL ,
2:      &                ISF5 , ILF , ICLN , IMT ,
3:      &                ISF4 , THEA , THEB ,
4:      &                CX , ICX , MCX , KLIB1, XLIB1, XLIB2,
5:      &                NUC , NMT , NBANK,
6:      &                IWK1 , IWK2 , IWK3 , IWK4 , IWK5 , IWK6 ,
7:      &                WK1 , WK2 , WK3 , WK4 , WK5 , R ,
8:      &                IRAND )
9: C-----1-----5-----2-----3-----5-----4-----5-----5-----6-----5-----7---
10: C=====
11: C PURPOSE: SAMPLING OF OUTGOING ENERGY FROM FILE 6 DATA
12: C using the following data already determined.
13: C
14: C ISEL(J) : I = ISEL(J)
15: C NF6 : number of particles to be processed (J=1,NF6)
16: C EEEJ(J) : incident energy
17: C WW(J) :
18: C COSL(J) : scattering angle cosine in Lab system
19: C ISF5(I) : pointer to used data in CX array
20: C ILF(I) : LF value
21: C ICLN(I) : collision nuclide
22: C IMT(I) : reaction MT
23: C ISF4(I) : another pointer to used data (LF=67,INT>10)
24: C THEA(I) : maximum energy for LF=1,6,61 (INT>10)
25: C : interpolation ratio for LF=67
26: C THEB(I) : minimum energy for LF=1,6,61 (INT>10)
27: C
28: C -----> EEEJ(J) : OUTGOING ENERGY
29: C WW(J)
30: C
31: C CALLED IN: NXTNR3
32: C CALL : RANU2
33: C=====
34: C
35:      implicit real*8(A-H,O-Z)
36: C
37:      integer ISF5(NBANK), ILF(NBANK), ICLN(NBANK), IMT(NBANK),
38:      &        ISF4(NBANK), ISEL(NF6)
39:      real THEA(NBANK), THEB(NBANK), EEEJ(NF6), WW(NF6), COSL(NF6)
40: C
41: C**** CROSS SECTION DATA IN CX ARRAY
42:      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
43:      integer ICX(MCX)
44: c##<2007/03/14:PN3:
45: c## integer KLIB1(NUC,27)
46: integer KLIB1(NUC,31)
47: c##>
48: C
49: C
50: C**** WORK AREA
51: C
52:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK),IWK4(NBANK),
53:      &        IWK5(NBANK), IWK6(NBANK)
54:      real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK),
55:      &        WK5(NBANK), R(NBANK*5)
56: C
57: C****
58:      parameter( PI = 3.141592653589793200D+00 )
59:      parameter( PIH = 0.5D0*PI )
60:      parameter( PI2 = 2.0D0*PI )
61:      parameter( PI4I = 1.0D0/(4.0D0*PI ) )
62:      parameter( SMALL = 1.0E-35 )
63: C
64: C
65:      N61 = 0

```

```

66:      N66 = 0
67:      N67 = 0
68:      do 100 J = 1, NF6
69:          I = ISEL(J)
70:          if ( ILF(I).eq.61 ) then
71:              N61 = N61 + 1
72:              IWK1(N61) = J
73:          else if ( ILF(I).eq.66 ) then
74:              N66 = N66 + 1
75:              IWK2(N66) = J
76:          else if ( ILF(I).eq.67 ) then
77:              N67 = N67 + 1
78:              IWK3(N67) = J
79:          end if
80:      100 continue
81: C
82: C..... LF=61 : Kalbach-87 DENSITY FUNCTION
83: C
84:      if ( N61.gt.0 ) then
85:          call RANU2( IRAND, R, N61*4, ICON )
86:          N2 = N61*2
87:          N3 = N61*3
88:          NN = 0
89: C
90: *VOCL LOOP,NOVREC
91:      do 500 JJ = 1, N61
92:          J = IWK1(JJ)
93:          I = ISEL(J)
94: C
95: C..... BMC SAMPLING USING 3 RANDOM NUMBERS
96:          LSTF5E = ISF5(I)
97:          NEP = ICX(LSTF5E)
98:          LSTF5E = LSTF5E + 1
99:          ND = ICX(LSTF5E)
100: C
101: C/#IF ROUND OFF(NEAREST)
102:          LL2 = MIN( INT(NEP*R(JJ))+1,NEP)
103: C/#ELSE
104:          *          LL2 = NEP*R(JJ) + 1
105: C/#ENDIF
106: C
107:          LL3 = LSTF5E + LL2
108: C
109: C/#IF ROUND OFF(NEAREST)
110:          LL4 = 2*LL2
111:          LL4 = MIN(LL4,INT(LL4+(R(N61+JJ)-CX(LL3))) + LSTF5E + NEP)
112: C/#ELSE
113:          *          R2 = R(N61+JJ) - CX(LL3)
114:          *          LL4 = LSTF5E + NEP + 2*LL2 + R2
115: C/#ENDIF
116:          LL5 = ICX(LL4)
117:          LLL = LSTF5E + 3*NEP
118:          L3 = LLL + LL5
119:          L4 = L3 + NEP + 1
120:          L5 = L4 + NEP + 1
121:          L6 = L5 + NEP + 1
122: C
123:          E0 = CX(L3)
124:          E1 = CX(L3+1)
125:          P0 = CX(L4)
126:          P1 = CX(L4+1)
127:          R0 = CX(L5)
128:          R1 = CX(L5+1)
129:          A0 = CX(L6)
130:          A1 = CX(L6+1)

```


src/mvp/sef6n3.f

```

131: C
132:       if ( LL5.le.ND ) then
133: C..... CONTINUOUS SPECTRA
134:         RRR      = (P0+P1)*R(N2+JJ)
135:         RRR      = RRR/(SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
136: C
137:         PRECF    = R1 + (R0-R1)*RRR
138:         ANGDS    = A1 + (A0-A1)*RRR
139:       else
140: C..... DISCRETE SPECTRA
141:         RRR      = 0.0
142:         PRECF    = R1
143:         ANGDS    = A1
144:       end if
145: C..... SECONDARY ENERGY AND SCATTERING ANGLE COSINE IN CM SYSTEM
146:       ECM      = E1 + (E0-E1)*RRR
147: C
148:       if ( LL5.le.ND ) then
149:         LLE      = LLL + 1
150:       if ( THEA(I).gt.0. .and. THEB(I).gt.0. ) then
151:         ECM      = THEB(I) + (THEA(I)-THEB(I))*
152: &               (ECM-CX(LLE+ND))/(CX(LLE)-CX(LLE+ND))
153:       else if ( THEA(I).gt.0. ) then
154:         ECM      = THEA(I)*ECM/CX(LLE)
155:       end if
156:     end if
157: C
158: C .... Conversion to Lab system ( taken from MCNP4b )
159:       A1        = XLIB1(ICLN(ISEL(J)),1) + 1.0
160: C .... T3 = 1-ge**2
161:       T3        = 1. - ECM*A1**2/EEEJ(J)
162:       P         = 0.0
163:       if ( T3.lt.0.0 ) then
164:         T1       = 1. / A1
165:         EIN      = EEEJ(J)
166:         EEEJ(J)  = EIN *
167: &               (T1*(COSL(J)+SQRT(COSL(J)**2-T3)))*2
168:         T2       = SQRT( EEEJ(J)/ECM )
169:         T4       = T1 * SQRT( EIN/EEEJ(J) )
170:         XCM      = T2 * ( COSL(J)-T4 )
171:         T5       = SINH(ANGDS) * ( 1.-COSL(J)*T4 )
172:         if ( T5.gt.0.0 )
173: &         P      = ( COSH( ANGDS*XCM ) + PRECF*SINH( ANGDS*XCM ) )
174: &               * ANGDS*T2/T5 * PI4I
175:       else if ( COSL(J).gt.SQRT(T3) ) then
176:         T1       = 1. / A1
177:         EIN      = EEEJ(J)
178:         EEEJ(J)  = EIN *
179: &               (T1*(COSL(J)+SQRT(COSL(J)**2-T3)))*2
180:         EEE2     = EIN *
181: &               (T1*(COSL(J)-SQRT(COSL(J)**2-T3)))*2
182:         T21      = SQRT( EEEJ(J)/ECM )
183:         T41      = T1 * SQRT( EIN/EEEJ(J) )
184:         XCM1     = T21 * ( COSL(J)-T41 )
185:         T51      = SINH(ANGDS) * ( 1.-COSL(J)*T41 )
186:         P1       = 0.0
187:         P2       = 0.0
188:         if ( T51.gt.0.0 )
189: &         P1     = ( COSH( ANGDS*XCM1 ) + PRECF*SINH( ANGDS*XCM1 ) )
190: &               * ANGDS*T21/T51 * PI4I
191:         T22      = SQRT( EEE2/ECM )
192:         T42      = T1 * SQRT( EIN/EEE2 )
193:         XCM2     = T22 * ( COSL(J)-T42 )
194:         T52      = SINH(ANGDS) * ( 1.-COSL(J)*T42 )
195:         if ( T52.gt.0.0 )

```

```

196: &         P2 = ( COSH( ANGDS*XCM2 ) + PRECF*SINH( ANGDS*XCM2 ) )
197: &               * ANGDS*T22/T52 * PI4I
198:       P        = P1 + P2
199:       if ( P*R(N3+JJ).gt.P1 ) EEEJ(J) = EEE2
200:     end if
201:     WW(J)      = WW(J) * P
202:   500 continue
203: end if
204: C
205: C
206: C..... LF=66 : N-BODY PHASE SPACE DISTRIBUTION
207: C
208: C   IWK4 : NPSX
209: C   WK1  : EIMAX
210:       if ( N66.gt.0 ) then
211:         call RANU2( IRAND, R, N66*5, ICON )
212:         N2        = N66*2
213:         N3        = N66*3
214:         N4        = N66*4
215: *VOCL LOOP,NOVREC
216:       do 510 JJ = 1, N66
217:         J         = IWK2(JJ)
218:         I         = ISEL(J)
219:         LL        = ISF5(I)
220:         APSX      = CX(LL)
221:         IWK4(JJ)  = ICX(LL+2)
222:         ATW       = XLIB1(ICLN(I),1)
223:         EIMAX     = ATW*EEEJ(J) /(ATW+1.0) - XLIB2(ICLN(I),IMT(I),1)
224:         WK1(JJ)   = EIMAX*(APSX-1.0) /APSX
225: C
226:         R1        = SQRT(R(JJ))
227:         T1        = PIH*R(N66+JJ)
228:         X1        = (R1*COS(T1))**2
229:         S1        = (R1*SIN(T1))**2 + X1
230:         WK2(JJ)   = -X1*LOG(S1) /S1 - LOG(R(N2+JJ))
231:         R2        = SQRT(R(N3+JJ))
232:         T2        = PIH*R(N4+JJ)
233:         X2        = (R2*COS(T2))**2
234:         S2        = (R2*SIN(T2))**2 + X2
235:         WK3(JJ)   = -X2*LOG(S2) /S2
236:   510 continue
237: C
238:       call RANU2( IRAND, R, N66*4, ICON )
239:       N2        = N66*2
240:       N3        = N66*3
241: C
242: *VOCL LOOP,NOVREC
243:       do 520 JJ = 1, N66
244:         J         = IWK2(JJ)
245:         if ( IWK4(JJ).eq.3 ) then
246:           P        = R(JJ)
247:         else if ( IWK4(JJ).eq.4 ) then
248:           P        = R(JJ)*R(N66+JJ)
249:         else
250:           P        = R(JJ)*R(N66+JJ)*R(N2+JJ)*R(N3+JJ)
251:         end if
252: C..... SECONDARY ENERGY AND SCATTERING ANGLE COSINE IN CM SYSTEM
253:         ECM      = WK1(JJ)*WK2(JJ) /(WK2(JJ)+WK3(JJ)-LOG(P))
254: C .... Conversion to Lab system
255:         A1       = XLIB1(ICLN(ISEL(J)),1) + 1.0
256:         EEEJ(J)  = ECM +
257: &               (EEEJ(J)+2.*A1*XCM*SQRT(EEEJ(J)*ECM)) /A1/A1
258: C .... T3 = 1-ge**2
259:         T3       = 1. - ECM*A1**2/EEEJ(J)
260:         P        = 0.0

```

src/mvp/sef6n3.f

```

261:         if ( T3.lt.COSL(J)**2 ) then
262:             T1      = 1. / A1
263:             EIN      = EEEJ(J)
264:             EEEJ(J) = EIN *
265:             &        (T1*(COSL(J)+SQRT(COSL(J)**2-T3)))*2
266:             T2      = SQRT( EEEJ(J)/ECM )
267:             T4      = T1 * SQRT( EIN/EEEJ(J) )
268:             XCM      = T2 * ( COSL(J)-T4 )
269:             T5      = 1. - COSL(J)*T4
270:             if ( T5.gt.0.0 ) P = T2 / T5 * PI4I
271:         end if
272:         WW(J)      = WW(J) * P
273: 520 continue
274: end if
275: C
276: C..... LF=67 : TABULATED ANGLE-ENERGY DISTRIBUTION
277: C
278:         if ( N67.gt.0 ) then
279: C
280:             N      = 0
281: C
282: *VOCL LOOP,NOVREC
283:         do 540 JJ = 1, N67
284:             J      = IWK3(JJ)
285:             I      = ISEL(J)
286:             NBINA1 = KLIB1(ICLN(I),20)
287:             if ( COSL(J).ge.CX(ISF5(I)).and.
288:             &        COSL(J).le.CX(ISF5(I)+NBINA1-1) ) then
289:                 N      = N + 1
290:                 IWK3(N) = J
291:                 IWK4(N) = ISF5(I)
292:                 IWK5(N) = NBINA1
293:                 WK1(N)  = COSL(J)
294:             else
295:                 WW(J)   = 0.0
296:             end if
297: 540 continue
298:             N67 = N
299: C
300:             call BSINC3( CX, IWK5, WK4, IWK4, WK1, IWK6, N67 )
301: C
302: *VOCL LOOP,NOVREC
303:         do 550 JJ = 1, N67
304:             J      = IWK3(JJ)
305:             LL      = IWK4(JJ) + IWK6(JJ)
306:             DMUPI2  = PI2 * (CX(LL) - CX(LL-1)) * (IWK5(JJ)-1)
307:             WW(J)   = WW(J) / DMUPI2
308: 550 continue
309: C
310: *VOCL LOOP,NOVREC
311:         do 560 JJ = 1, N67
312:             IWK4(JJ) = IWK4(JJ) + IWK5(JJ) + 2
313:             IWK5(JJ) = ICX(IWK4(JJ)-1)
314: 560 continue
315: C
316:             call BSINC3( CX, IWK5, WK4, IWK4, WK1, IWK6, N67 )
317: C
318:             call RANU2( IRAND, R, N67*4, ICON )
319:             N2      = N67*2
320:             N3      = N67*3
321: C
322:             N67I    = 0
323: C
324: *VOCL LOOP,NOVREC
325:         do 570 JJ = 1, N67

```

```

326:         if ( IWK6(JJ).eq.0 ) IWK6(JJ) = 1
327:             L2      = IWK4(JJ) + IWK6(JJ)
328:             L1      = L2 - 1
329:             L3      = IWK4(JJ) + IWK5(JJ) - 1
330:             INTA     = ICX(IWK4(JJ)-2)
331: C
332: C..... STEP INTERPOLATION
333: C
334:         if ( INTA.eq.1 ) then
335:             LST2A    = ICX(IWK6(JJ)+L3)
336: C
337: C..... LINEAR INTERPOLATION
338: C
339:         else
340:             X1      = (CX(L1)-WK1(JJ)) / (CX(L1)-CX(L2))
341:             if ( X1.lt.0.0 ) X1 = 0.0
342:             if ( X1.gt.1.0 ) X1 = 1.0
343: C
344: C/#IF ROUNDOFF(NEAREST)
345:             IT      = MIN(1,INT(R(JJ)+X1)) + IWK6(JJ)
346: C/#ELSE
347:             *        R1      = R(JJ) + X1
348:             *        IT      = IWK6(JJ) + R1
349: C/#ENDIF
350: C
351:             LST2A    = ICX(IT+L3)
352:         end if
353: C
354:             NEP      = ICX(LST2A)
355: C
356: C/#IF ROUNDOFF(NEAREST)
357:             LL2      = MIN(INT(NEP*R(N67+JJ))+1,NEP)
358: C/#ELSE
359:             *        LL2      = NEP*R(N67+JJ) + 1
360: C/#ENDIF
361: C
362:             LL3      = LST2A + LL2
363: C
364: C/#IF ROUNDOFF(NEAREST)
365:             LL4      = 2*LL2
366:             LL4      = MIN(LL4,INT(LL4+R(N2+JJ)-CX(LL3))) + LST2A + NEP
367: C/#ELSE
368:             *        R2      = R(N3+JJ) - CX(LL3)
369:             *        LL4      = LST2A + NEP + 2*LL2 + R2
370: C/#ENDIF
371:             LLL      = LST2A + 3*NEP
372:             L3       = LLL + ICX(LL4)
373:             L4       = L3 + NEP + 1
374: C
375:             E0      = CX(L3)
376:             E1      = CX(L3+1)
377:             P0      = CX(L4)
378:             P1      = CX(L4+1)
379: C
380:             RRR      = (P0+P1)*R(N3+JJ)
381:             J        = IWK3(JJ)
382:             EEEJ(J) = E1 + (E0-E1)*RRR/
383:             &        (SQRT(P1*P1+(P0-P1)*RRR)+P1+SMALL)
384: C
385: C .... INT > 10, WK2:EMIN, WK3:EMAX
386:             I        = ISEL(J)
387:             if ( THEA(I).ge.0. ) then
388:                 N67I  = N67I + 1
389:                 IWK3(N67I) = J
390:                 IWK4(N67I) = ISF4(I) + KLIB1(ICLN(I),20) + 2

```

src/mvp/sef6n3.f

```

391:          IWK5(N67I) = ICX(IWK4(N67I)-1)
392:          WK1(N67I)  = WK1(JJ)
393:          LLE        = LLL + 1
394:          WK2(N67I)  = CX(LLE+NEP)
395:          WK3(N67I)  = CX(LLE)
396:          end if
397: 570      continue
398: C
399: C ... INT(E)>10
400: C
401:      if ( N67I.gt.0 ) then
402: C
403:      call BSINC3( CX, IWK5, WK4, IWK4, WK1, IWK6, N67I )
404:      call RANU2( IRAND, R, N67I, ICON )
405: C
406: *VOCL LOOP,NOVREC
407:      do 580 JJ = 1, N67I
408:          if ( IWK6(JJ).eq.0 ) IWK6(JJ) = 1
409:          L2      = IWK4(JJ) + IWK6(JJ)
410:          L1      = L2 - 1
411:          L3      = IWK4(JJ) + IWK5(JJ) - 1
412:          INTA    = ICX(IWK4(JJ)-2)
413: C
414: C..... STEP INTERPOLATION
415: C
416:          if ( INTA.eq.1 ) then
417:              LST2A = ICX(IWK6(JJ)+L3)
418: C
419: C..... LINEAR INTERPOLATION
420: C
421:          else
422:              X1    = (CX(L1)-WK1(JJ)) / (CX(L1)-CX(L2))
423:              if ( X1.lt.0.0 ) X1 = 0.0
424:              if ( X1.gt.1.0 ) X1 = 1.0
425: C
426: C/#IF ROUND OFF(NEAREST)
427:              IT    = MIN(1,INT(R(JJ)+X1)) + IWK6(JJ)
428: C/#ELSE
429:              R1    = R(JJ) + X1
430:              IT    = IWK6(JJ) + R1
431: C/#ENDIF
432: C
433:              LST2A = ICX(IT+L3)
434:          end if
435: C
436:          NEP      = ICX(LST2A)
437:          LLE      = LST2A + 3*NEP + 1
438: C
439:          J        = IWK3(JJ)
440:          I        = ISEL(J)
441: C
442:          if ( THEB(I).le.0. ) then
443:              EMAX  = WK3(JJ) + (CX(LLE)-WK3(JJ))*THEA(I)
444:              EEEJ(J) = EEEJ(J) * EMAX / WK3(JJ)
445:          else
446:              EMAX  = WK3(JJ) + (CX(LLE)-WK3(JJ))*THEA(I)
447:              EMIN  = WK2(JJ) + (CX(LLE+NEP)-WK2(JJ))*THEA(I)
448:              EEEJ(J) = EMIN + (EMAX-EMIN)*
449:              &      (EEEJ(J)-WK2(JJ))/(WK3(JJ)-WK2(JJ))
450:          end if
451: C
452: 580      continue
453: C
454:      end if
455: C
456:          end if
457: C
458:          return
459:          end

```

src/mvp/select.f

```
1:      subroutine SELALZ( MACT, NZONE, NBANK, NHIST, NTGEN, NPART,
2:      &                  NGENE, NFFL, NNXT, NCOLS, NDEAD, NBREF,
3:      &                  NREFS, JREFL, JLATT, NXLT, NLBZ, NCOLP,
4:      &                  JNEUT, JPHOT, JTERM )
5: C=====
6: C  PURPOSE: SELECT NEXT ACTION  (ALL ZONE GEOMETRY TRACKING)
7: C  CALLED IN: ACTION
8: C=====
9:      integer NFFL(1), NNXT(1), NBREF(1), NXLT(1), MMM(6)
10: C/#IF INTEGER8
11: *      integer*8 NPART, NTGEN
12: C/#ELSE
13:      integer  NPART, NTGEN
14: C/#ENDIF
15: cccc DATA  MMM / 6*0 /
16: C
17: C .... IF ALL PARTICLES ARE DEAD, GENERATE NEXT PARTICLES ....
18:      if ( JTERM.eq.0.and.NBANK-NDEAD.le.0 ) then
19:          NGENE = MIN(NPART,NTGEN,NHIST)
20:          if ( NGENE.eq.0 ) go to 100
21:          MACT = 0
22:          MZONE = 0
23:          if ( NTGEN+NGENE.eq.NPART ) JTERM = 1
24:          return
25:      end if
26: C
27:      100 continue
28: C
29: C .... CHECK FLIGHT STACK .....
30:      MMM(1) = NFFL(NZONE+1)
31: C
32: C .... CHECK NEXT ZONE SEARCH STACK .....
33: C
34:      MMM(2) = NNXT(NZONE+1)
35: C
36:      do 110 I = 3, 6
37:          MMM(3) = 0
38:      110 continue
39: C
40: C .... CHECK COLLISION STACK .....
41: C
42:      if ( JNEUT.ne.0 ) MMM(3) = NCOLS
43: C
44: C .... CHECK REFLECTION STACK .....
45: C
46:      if ( JREFL.ne.0 ) MMM(4) = NBREF(NREFS+1)
47: C
48: C .... CHECK LATTICE SEARCH STACK .....
49: C
50:      if ( JLATT.ne.0 ) MMM(5) = NXLT(NLBZ+1)
51: C
52: C .... CHECK PHOTON COLLISION STACK .....
53: C
54:      if ( JPHOT.ne.0 ) MMM(6) = NCOLP
55: C
56: C ....
57:      MMAX = 0
58: *VOCL LOOP, SCALAR
59:      do 120 M = 1, 6
60:          if ( MMM(M).gt.MMAX ) then
61:              MMAX = MMM(M)
62:              MACT = M
63:          end if
64:      120 continue
65:      if ( JTERM.eq.1.and.MMAX.eq.0 ) MACT = 99
66:
67:      return
end
```

src/mvp/selone.f

```

1:      subroutine SELONE( MACT, JTERM, MZONE, NTGEN,NGENE,NGENE0,NGBAT,
2:      &                  H, NFFL,NNXT,NBREF,NXLT,
3:      &                  IMMODE,IMNFL, IMNNX, IMNLT,
4:      &                  NCNTR, WCNTR, NEVENT)
5: C=====
6: C PURPOSE: SELECT NEXT ACTION (ZONE-SELECTION GEOMETRY TRACKING)
7: C CALLED IN: ACTION
8: C
9: C <arguments: i=input o=output, w=work >
10: C
11: C o MACT : next action selected by this routine.
12: C   = 0 : GENERATE NEXT BATCH OF PARTICLES.
13: C   = 1 : FREE FLIGHT
14: C   = 2 : NEXT ZONE SEARCH
15: C   = 3 : COLLISION (NEUTRON)
16: C   = 5 : REFLECTION
17: C   = 6 : LATTICE SEARCH
18: C   = 7 : COLLISION (PHOTON)
19: C   = 8 : NEXT EVENT ESTIMATOR
20: C   = 88 : End of batch
21: C   = 99 : END OF RUN
22: C o MZONE : target zone of next action ( =0 : all zones )
23: C o NTGEN : number of generated source so far in calculation
24: C o NGENE : number of source particle to be generated when positive
25: C o NGBAT : remaining number of histories in current batch.
26: C o JTERM : set to 1 from 0 when all histories will be done after
27: C           the next batch.
28: C i JREPR : running in "NO-REPRODUCIBLE-RUN" mode if = 0
29: C           else running in "REPRODUCIBLE-RUN" mode.
30: C o IMMODE : action mode when MACT=8.
31: C
32: C
33: C (for other arguments, see description after common statements)
34: C
35: C=====
36: C/#IF INTEGER8
37: *      integer*8 NTGEN
38: C/#ELSE
39:      integer      NTGEN
40: C/#ENDIF
41: C
42:      real H(*)
43:      integer NFFL(NZONE+1), NNXT(NZONE+1), NBREF(NREFS+1), NXLT(NLBZ+1)
44:      real*8  NCNTR(NEVENT,2), WCNTR(NEVENT,2)
45: C
46:      integer IMNFL(NZONE+1), IMNNX(NZONE+1), IMNLT(NLBZ+1)
47: C
48:      include 'INC/_FLAGS'
49:      include '../shared/INC/_SIZES'
50:      include 'INC/_STACK'
51:      include 'INC/_XBANK'
52:      include '../shared/INC/_IOUNIT'
53:      include '../shared/INC/_TASKDT'
54: C
55: C ..... DATA FOR ENDLESS RANDOM-WALK DETECTION ....
56: C
57:      include '../shared/INC/_COUNTS'
58: Ccccc REAL*8  DESUM,DESUM2
59: Ccccc INTEGER JLOOP,NLOOP
60: Ccccc DATA   DESUM /0.0D0/, DESUM2 /0.0D0/, JLOOP /0/, NLOOP /0/
61: C
62: C ... number of events for endless loop printout
63: C parameter ( MXEDLC = 30 )
64: C
65: C ..... data for debugging .....

```

```

66: C
67:      parameter( NSAVEV = 30 )
68: C/#IF PARA( SX* )
69:      local common /SELDBG/
70: C/#ELSEIF PARA( CRAY* )
71:      task common /SELDBG/
72: C/#ELSE
73:      common /SELDBG/
74: C/#ENDIF
75:      &          MACTSV, MZONSV, MMAXSV, MMMSV(7),          NFFLSV,
76:      &          NNXTSV, IDEFSV, NBREFSV,          NXLTSV, NCOLSSV,
77:      &          NCOLPSV,
78:      &          IMNFLSV, IMNNXSV,          IMNLTSV
79: C
80: C .... LOCAL DATA
81: C
82:      integer MMM(8), MZZ(8)
83:      real*8 WENDL
84: C
85: C-----
86: C
87: C =====
88: C .... Check consistency of stacks & bank ....
89: C =====
90: C
91:      if ( JDEBG(5).ne.0 ) then
92: C
93:          NSERR = 0
94:          NALIVE = 0
95: C
96: C ---- FLIGHT STACK ----
97: C
98:          NALIVE = NALIVE + NFFL(NZONE+1)
99:          NKKKK = 0
100:          do 100 IZ = 1, NZONE
101:              NKKKK = NKKKK + NFFL(IZ)
102:          100 continue
103:          if ( NFFL(NZONE+1).ne.NKKKK ) then
104:              NSERR = NSERR + 1
105:              write(IPR,7000) 'FLIGHT', NKKKK, NFFL(NZONE+1),
106:              &          (NFFL(IZ),IZ=1,NZONE)
107:          end if
108: C
109: C ---- SEARCH STACK ----
110: C
111:          NALIVE = NALIVE + NNXT(NZONE+1)
112:          NKKKK = IDEFER
113:          do 110 IZ = 1, NZONE
114:              NKKKK = NKKKK + NNXT(IZ)
115:          110 continue
116:          if ( NNXT(NZONE+1).ne.NKKKK ) then
117:              NSERR = NSERR + 1
118:              write(IPR,7000) 'SEARCH', NKKKK, NNXT(NZONE+1),
119:              &          (NNXT(IZ),IZ=1,NZONE), IDEFER
120:          end if
121: C
122: C ---- REFLECTION STACK ----
123: C
124:      if ( JREFL.ne.0 ) then
125:          NALIVE = NALIVE + NBREF(NREFS+1)
126:          NKKKK = 0
127:          do 120 IZ = 1, NREFS
128:              NKKKK = NKKKK + NBREF(IZ)
129:          120 continue
130:          if ( NBREF(NREFS+1).ne.NKKKK ) then

```

src/mvp/selone.f

```

131:      NSERR = NSERR + 1
132:      write(IPR,7000) 'REFLECTION', NKKKK, NBREF(NREFS+1),
133:      &      (NBREF(IZ),IZ=1,NREFS)
134:      end if
135:      end if
136: C
137: C ---- LATTICE-SEARCH STACK ----
138: C
139:      if ( JLATT.ne.0 ) then
140:      NALIVE = NALIVE + NXLT(NLBZ+1)
141:      NKKKK = 0
142:      do 130 IZ = 1, NLBZ
143:      NKKKK = NKKKK + NXLT(IZ)
144: 130 continue
145:      if ( NXLT(NLBZ+1).ne.NKKKK ) then
146:      NSERR = NSERR + 1
147:      write(IPR,7000) 'REFLECTION', NKKKK, NXLT(NLBZ+1),
148:      &      (NXLT(IZ),IZ=1,NLBZ)
149:      end if
150:      end if
151: C
152: C --- collision stack(neutron) ---
153: C
154:      if ( JNEUT.ne.0 ) NALIVE = NALIVE + NCOLP
155: C
156: C --- collision stack(photon) ---
157: C
158:      if ( JPHOT.ne.0 ) NALIVE = NALIVE + NCOLP
159: C
160:      if ( NALIVE+NDEAD.ne.NBANK ) then
161:      NSERR = NSERR + 1
162:      write(IPR,7020) NALIVE, NDEAD, NBANK
163:      end if
164: C
165: C
166: C ---- imaginary particle for next event estimator ----
167: C
168:      if ( JPTDT.ne.0 ) then
169:      NALIVE = IMNFL(NZONE+1)
170:      NKKKK = 0
171:      do 140 IZ = 1, NZONE
172:      NKKKK = NKKKK + IMNFL(IZ)
173: 140 continue
174:      if ( IMNFL(NZONE+1).ne.NKKKK ) then
175:      NSERR = NSERR + 1
176:      write(IPR,7000) 'NES-FLIGHT', NKKKK, IMNFL(NZONE+1),
177:      &      (IMNFL(IZ),IZ=1,NZONE)
178:      end if
179: C
180:      NALIVE = NALIVE + IMNNX(NZONE+1)
181:      NKKKK = 0
182:      do 150 IZ = 1, NZONE
183:      NKKKK = NKKKK + IMNNX(IZ)
184: 150 continue
185:      if ( IMNNX(NZONE+1).ne.NKKKK ) then
186:      NSERR = NSERR + 1
187:      write(IPR,7000) 'NES-SEARCH', NKKKK, IMNNX(NZONE+1),
188:      &      (IMNNX(IZ),IZ=1,NZONE)
189:      end if
190:      if ( JLATT.ne.0 ) then
191:      NALIVE = NALIVE + IMNLT(NLBZ+1)
192:      NKKKK = 0
193:      do 160 IZ = 1, NLBZ
194:      NKKKK = NKKKK + IMNLT(IZ)
195: 160 continue

```

```

196:      if ( IMNLT(NLBZ+1).ne.NKKKK ) then
197:      NSERR = NSERR + 1
198:      write(IPR,7000) 'NES-LATTICE', NKKKK, IMNLT(NLBZ+1),
199:      &      (IMNLT(IZ),IZ=1,NLBZ)
200:      end if
201:      end if
202: C
203:      if ( NALIVE+NDIMPT.ne.IMPMAX ) then
204:      NSERR = NSERR + 1
205:      write(IPR,7020) NALIVE, NDIMPT, IMPMAX
206:      end if
207: C
208:      end if
209: C
210:      if ( NSERR.gt.0 ) then
211:      write(IPR,7040) MACTSV, MZONSV, MMAXSV, MMSV
212:      write(IPR,7060) 'STACKS '
213:      write(IPR,7080) 'SAVED ', NFFLSV, NNXTSV, IDEFSV, NBREFSV,
214:      &      NXLTSV, NCOLSSV, NCOLPSV
215:      write(IPR,7080) 'CURRENT ', NFFL(NZONE+1), NNXT(NZONE+1),
216:      &      IDEFER, NBREF(NREFS+1), NXLT(NLBZ+1), NCOLS, NCOLP
217:      end if
218: C
219: 7000 format(1X,'XXX(SELONE) ',A,' stack inconsistency : sum ',I8,
220:      &      ' logged sum ',I8/(:1X,3X,10(:I7)))
221: 7020 format(1X,'XXX(SELONE) Sum of number of alive particles(=',I8,
222:      &      ') and NDEAD (=',I10,') does not match NBANK (=',I8)
223: 7040 format(1X,'XXX(SELONE) Particle stack inconsistency detected: ',
224:      &      ' the last action ID is ',I3,'. saved parameters :'/1X,
225:      &      ' MZONE ',I6,' MMAX ',I8/1X,' MMM ',10(:I8))
226: 7060 format(1X,A,3X,' NFFL ',I8,' NNXT ',I8,' IDEFER ',I8,' NBREF ',
227:      &      ' NXLT ',I8,' NCOLS ',I8,' NCOLP ',I8)
228: 7080 format(1X,A,1X,7I8)
229: C
230:      end if
231: C
232: C =====
233: C .... IF ALL PARTICLES ARE DEAD, GENERATE NEXT PARTICLES ....
234: C =====
235: C
236: 1000 continue
237: C
238: CM1993-9 IF( jterm.EQ.0 .AND. NBANK-NDEAD.LE.NHIST*SUPPLY ) THEN
239: CCCC if ( JTERM.eq.0.and.NBANK-NDEAD.le.0 ) then
240:      if ( NBANK-NDEAD.le.0 ) then
241:      C
242:      C ... process remaining next event estimator task until finish ...
243:      C
244:      if ( JPTDT.ne.0.and.NIMPT.ne.0 ) then
245:      MACT = 8
246:      MZONE = 0
247:      IMMODE = 1
248:      go to 230
249:      end if
250: C
251: C ... end of all histories
252: C
253:      if ( JTERM.eq.1 ) then
254:      MACT = 99
255:      goto 230
256:      end if
257: C
258:      if ( JREPR.eq.1 ) then
259:      C
260:      C ... Sub-batch (whose history size is NHSUB) is introduced in

```

src/mvp/selone.f

```

261: C          Feb 2000.  NBANK may be smaller than NHIST.
262: C
263: CCCCCC      NGENE      = MIN(NPART-NTGEN,NHIST-NBANK+NDEAD)
264: C          NGENE      = MIN(NGBAT,NHSUB)
265: CCCCCC      NGBAT      = NGBAT - NGENE
266: C
267: CCCCCC      if ( NGENE.eq.0 ) go to 170
268: C
269: C          ... MACT = 88: end of batch
270: C
271: C          if ( NGENE.eq.0 ) then
272: C              MACT = 88
273: C          else
274: C              MACT      = 0
275: C              MZONE      = 0
276: C              if ( NTGEN+NGENE.eq.NPART ) JTERM      = 1
277: C          end if
278: C
279: C          else
280: C              call REQHST( NGENE, JTSKR )
281: C              if ( NGENE.eq.0 ) then
282: C                  MACT      = 99
283: C                  MZONE      = 0
284: C                  JTERM      = 1
285: C              else
286: C                  MACT      = 0
287: C                  MZONE      = 0
288: C              end if
289: C          end if
290: C
291: C          go to 230
292: C
293: C          end if
294: C
295: C
296: C =====
297: C ... check number of particles in event stacks and decide action.
298: C =====
299: C
300: C 170 MM      = 0
301: C      MZ      = 0
302: C
303: C .... FLIGHT STACK .....
304: C
305: C          if ( NFFL(NZONE+1).gt.0 ) then
306: C *VOCL LOOP,NOVREC
307: C              do 180 KZ = 1, NZONE
308: C                  if ( NFFL(KZ).gt.MM ) then
309: C                      MM      = NFFL(KZ)
310: C                      MZ      = KZ
311: C                  end if
312: C              180 continue
313: C          end if
314: C          MMM(1) = MM
315: C          MZZ(1) = MZ
316: C
317: C .... NEXT ZONE SEARCH STACK .....
318: C
319: C          MM      = 0
320: C          MZ      = 0
321: C          if ( NNXT(NZONE+1).gt.0 ) then
322: C *VOCL LOOP,NOVREC
323: C              do 190 KZ = 1, NZONE
324: C                  if ( NNXT(KZ).gt.MM ) then
325: C                      MM      = NNXT(KZ)

```

```

326: C          MZ      = KZ
327: C          end if
328: C          190 continue
329: C          end if
330: C          MMM(2) = MM
331: C          MZZ(2) = MZ
332: C
333: C          do 170 I = 3, 7
334: C              MMM(I) = 0
335: C              MZZ(I) = 0
336: C          170 continue
337: C
338: C          MMM(3) = 0
339: C          MMM(4) = 0
340: C          MMM(5) = 0
341: C          MMM(6) = 0
342: C          MMM(7) = 0
343: C          MMM(8) = 0
344: C          MZZ(3) = 0
345: C          MZZ(4) = 0
346: C          MZZ(5) = 0
347: C          MZZ(6) = 0
348: C          MZZ(7) = 0
349: C          MZZ(8) = 0
350: C
351: C
352: C          FCOL      = 0.5
353: C          FCOL      = 0.2
354: C          MLL0      = (NBANK-NDEAD)*FCOL
355: C
356: C .... COLLISION STACK .....
357: C
358: C          if ( JNEUT.ne.0 ) then
359: C              if ( NCOLS.gt.(NBANK-NDEAD)*FCOL .or. MMM(1).eq.0
360: C                  if ( NCOLS.gt.MLL0 .or. MMM(1).eq.0
361: C                      & .and.MMM(2).eq.0 ) MMM(3)      = NCOLS
362: C                  end if
363: C
364: C .... PHOTON-COLLISION STACK .....
365: C
366: C          if ( JPHOT.ne.0 ) then
367: C              if ( NCOLP.gt.(NBANK-NDEAD)*FCOL .or. MMM(1).eq.0
368: C                  if ( NCOLP.gt.MLL0 .or. MMM(1).eq.0
369: C                      & .and.MMM(2).eq.0 ) MMM(7)      = NCOLP
370: C                  end if
371: C
372: C          MLL      = MAX(MMM(3),MMM(7))
373: C
374: C .... DEFERRED PARTICLE (NEXT-ZONE SEARCH) .....
375: C
376: C          MMM(4) = IDEFER
377: C          MZZ(4) = -1
378: C
379: C .... REFLECTION STACK .....
380: C
381: C          if ( JREFL.ne.0 ) then
382: C              if ( NBREF(NREFS+1).gt.MLL ) MMM(5) = NBREF(NREFS+1)
383: C          end if
384: C
385: C .... LATTICE-SEARCH STACK .....
386: C
387: C          if ( JLATT.ne.0 ) then
388: C              if ( NXLT(NLBZ+1).gt.MLL ) MMM(6)      = NXLT(NLBZ+1)
389: C          end if
390: C

```

src/mvp/selone.f

```

391: C .... CHECK NEXT EVENT ESTIMATOR STACK .....
392: C
393:       if ( JPTDT.ne.0 ) then
394:         MM      = 0
395: *VOCL LOOP,NOVREC
396:       do 200 KZ = 1, NZONE
397:         if ( IMNFL(KZ).gt.MM ) then
398:           MM      = IMNFL(KZ)
399:         end if
400:       200 continue
401: *VOCL LOOP,NOVREC
402:       do 210 KZ = 1, NZONE
403:         if ( IMNNX(KZ).gt.MM ) then
404:           MM      = IMNNX(KZ)
405:         end if
406:       210 continue
407:       if ( JLATTT.ne.0 ) then
408:         if ( IMNLT(NLBZ+1).gt.MLL0.or.MM.eq.0 )
409:         &      MM = IMNLT(NLBZ+1)
410:       end if
411:       IMMODE = -1
412:       MMM(8) = MM
413:     end if
414: C
415: C ....
416: C
417:       MMAX      = 0
418:       MACT      = 0
419: C
420: *VOCL LOOP,SCALAR
421:       do 220 M = 1, 8
422:         if ( MMM(M).gt.MMAX ) then
423:           MMAX   = MMM(M)
424:           MACT   = M
425:           MZONE  = MZZ(M)
426:         end if
427:       220 continue
428: C
429: C ... this condition should not be satisfied here ...
430:       if ( JTERM.eq.1.and.MMAX.eq.0 ) MACT = 99
431: C
432:       if ( MACT.eq.4 ) MACT = 2
433: C
434: C =====
435: C DETECTION OF ENDLESS RANDOM-WALK
436: C =====
437: C
438: C JLOOP = 0 : NON-DETECTION MODE
439: C
440: C       Number of remaining particles are more than 1% of 'NHIST'.
441: C
442: C JLOOP = 1 : DETECTION MODE
443: C
444: C       in a batch
445: C       number of remaining particles are less than 2 or 1% of 'NHIST'.
446: C
447: C       Sum of processed-particles in detection mode
448: C       When ----- > ELOOP
449: C       Sum of processed-particles in the batch
450: C
451: C       enter monitoring-mode.
452: C
453: C JLOOP = 2 : MONITORING MODE
454: C
455: C       Print selected event, zone etc. upto MXEDLC times.

```

```

456: C       and terminate the batch.
457: C
458: C
459: C
460:       if ( MACT.ne.0.and.MACT.ne.99.and.MACT.ne.8 ) then
461: C
462: C       .... SUMMATION OF NUMBER OF PROCESSED PARTICLES IN THIS BATCH ..
463: C
464: C       DESUM      = DESUM + MMAX
465: C
466: C
467: CCCC      if ( JLOOP.eq.0.and.NBANK-NDEAD.le.MAX(INT(NHIST*0.01),2) )
468: C
469: C       ... NGBAT is remaining number of histories for this batch.
470: C
471: C <<comment>> when NHSUB < NHIST, a batch includes more than
472: C one sub-batches (new history control layer introduced in
473: C Feb 2000) and this endless-loop detection might not
474: C work for earlier sub-batches in a batch ...
475: C
476:       if ( JLOOP.eq.0.and.
477:         &      NGBAT+(NBANK-NDEAD).le.MAX(INT(NHIST*0.01),2) )
478:         &      then
479:           DESUM2 = 0.0
480:           JLOOP  = 1
481:         end if
482: C
483:       if ( JLOOP.eq.1 ) then
484:         DESUM2 = DESUM2 + MMAX
485:         if ( ELOOP.gt.0.0.and.DESUM2/DESUM.gt.ELOOP ) then
486: C/#IF PARA(SX* CRAY)
487: *       call MVPSYNC_LOCK(2)
488: C/#ENDIF
489:         write(IOW,7100) NBANK - NDEAD, MXEDLC
490:       7100      format(/1X,'!!! Probably ',I8,' particles are ',
491:         &      'in endless loop. ( monitor ',I3,' events )')
492: C/#IF PARA(SX* CRAY)
493: *       call MVPSYNC_UNLOCK(2)
494: C/#ENDIF
495:         JLOOP  = 2
496:         NLOOP  = 0
497:       end if
498:     end if
499: C
500: C ..... PROBABLE ENDLESS LOOP .....
501: C
502: C       ( monitor print upto MXEDLC events )
503: C
504:       if ( JLOOP.eq.2 ) then
505:         NLOOP = NLOOP + 1
506:         if ( NLOOP.lt.MXEDLC.and.MACT.ne.8 ) then
507:           call ELMNTR( IOW, NLOOP, MACT, MZONE, MMAX,
508:             &      NZONE, NREFS, NLBZ, NBANK,
509:             &      NFFL, H(LLSFFL), H(LIZFFL),
510:             &      NNXT, IDEFER, H(LLSSRC), H(LIZNXT),
511:             &      NCOLS, H(LLSCOL),
512:             &      NBREF, H(LLSREF),
513:             &      NXLT,  H(LLSLAT),
514:             &      NCOLP, H(LLSCLP),
515:             &      H(LIZZ), H(LXXX), H(LYYY), H(LZZZ),
516:             &      H(LAAA), H(LBBB), H(LCCC),
517:             &      H(LEEE), H(LTTT), H(LWWW) )
518: C
519: C       write(IOW,*) ' EVENT :', MACT, ' MZONE ', MZONE,
520: C       &      ' MMAX ', MMAX

```


src/mvp/selone.f

```

521:
522:         else if ( MMM(8).gt.0.and.NLOOP.eq.MXEDLC ) then
523:             MACT = 8
524:             MZONE = 0
525:             IMMODE = 1
526:         else
527: C
528: C         .... MORE THAN 30 EVENTS MONITORED. TERMINATE BATCH. ....
529: C
530: C/#IF PARA(SX* CRAY)
531: *      call MVPSYNC_LOCK(2)
532: C/#ENDIF
533:         write(IOW,'(/1X,A,I5,A/)') 'XXX Probably ', NBANK
534:         &      - NDEAD,
535:         &      ' particles are in ENDLESS LOOP and KILLED.'
536: C/#IF PARA(SX* CRAY)
537: *      call MVPSYNC_UNLOCK(2)
538: C/#ENDIF
539: C
540: C         .... CLEAR EVENT STACKS HERE ....
541: C
542: C         NDEAD = NBANK
543: C
544: C         call PUTVI( NFFL, NZONE+1, 0 )
545: C
546: C         call PUTVI( NNXT, NZONE+1, 0 )
547: C
548: C         if ( JREFL.ne.0 ) call PUTVI( NBREF, NREFS+1, 0 )
549: C         if ( JLATT.ne.0 ) call PUTVI( NXLT, NLBZ+1, 0 )
550: C
551: C         NCOLS = 0
552: C         NCOLP = 0
553: C         IDEFER = 0
554: C
555: C         call CLRSTK( IOW, NENDL, WENDL, H(LWWW),
556: C         &      NZONE, NREFS, NLBZ, NBANK,
557: C         &      NDEAD,
558: C         &      NFFL, H(LLSFFL),
559: C         &      NNXT, IDEFER, H(LLSSRC),
560: C         &      NCOLS, H(LLSCOL),
561: C         &      NBREF, H(LLSREF),
562: C         &      NXLT, H(LLSLAT),
563: C         &      NCOLP, H(LLSCLP) )
564: C
565: C         NCNTR(28,1) = NCNTR(28,1) + NENDL
566: C         WCNTR(28,1) = WCNTR(28,1) + WENDL
567: C
568: C         if ( JPTDT.ne.0 ) then
569: C             call PUTVI( IMNFL, NZONE+1, 0 )
570: C             call PUTVI( IMNNX, NZONE+1, 0 )
571: C             if ( JLATT.ne.0 ) call PUTVI( IMNLT, NLBZ+1, 0 )
572: C             NIMPT = 0
573: C         end if
574: C
575: C         if ( JTERM.eq.1 ) then
576: C             MACT = 99
577: C         else
578: C             go to 1000
579: C         end if
580: C     end if
581: C end if
582: C end if
583: C
584: C =====
585: C     .... END OF SELONE ...

```

```

586: C =====
587: C
588: C     230 continue
589: C
590: C     .... RESET ENDLESS-LOOP MONITORING PARAMETERS ....
591: C
592: CCCC if ( ( MACT.eq.0.and.NGBAT.eq.0 ) .or. MACT.eq.99 ) then
593: C     if ( MACT.eq.88 .or. MACT.eq.99 ) then
594: C         DESUM = 0.0
595: C         DESUM2 = 0.0
596: C         JLOOP = 0
597: C         NLOOP = 0
598: C     end if
599: C
600: C ... print selection information for debugging ...
601: C
602: C     if ( JDEBG(4).ge.2 ) then
603: C         write(IOW,7120) MACT, MZONE, MMAX, MMM
604: C         7120 format(1X,'(SELONE) mact ',I2,' mzone',I5,' mmax',I5,' mmm',10
605: C         &      (:I5))
606: C     end if
607: C
608: C     if ( JDEBG(4).ge.3 ) then
609: C         write(IOW,'(1x,3(a,i7))') 'NCOLS', NCOLS, 'NCOLP', NCOLP,
610: C         &      'NDEAD', NDEAD
611: C         write(IOW,7140) 'NFFL', NFFL
612: C         write(IOW,7140) 'NNXT', NNXT
613: C         if ( JLATT.ne.0 ) write(IOW,7140) 'NXLT', NXLT
614: C         if ( JREFL.ne.0 ) write(IOW,7140) 'NBREF', NBREF
615: C         7140 format(1X,'<',A,'>',10I7/(5X,10I7))
616: C     end if
617: C
618: C ... save MACT etc. for debug output ....
619: C
620: C     if ( JDEBG(5).ne.0 ) then
621: C         MACTSV = MACT
622: C         MZONSV = MZONE
623: C         MMAXSV = MMAX
624: C         do 240 K = 1, 7
625: C             MMSV(K) = MMM(K)
626: C         240 continue
627: C
628: C         NFFLSV = NFFL(NZONE+1)
629: C         NNXTSV = NNXT(NZONE+1)
630: C         IDEFSV = IDEFER
631: C         CCC if ( JREFS.ne.0 ) then
632: C             if ( JREFL.ne.0 ) then
633: C                 NBREFSV = NBREF(NREFS+1)
634: C             else
635: C                 NBREFSV = 0
636: C             end if
637: C             if ( JLATT.ne.0 ) then
638: C                 NXLTSV = NXLT(NLBZ+1)
639: C             else
640: C                 NXLTSV = 0
641: C             end if
642: C             if ( JNEUT.ne.0 ) then
643: C                 NCOLSSV = NCOLS
644: C             else
645: C                 NCOLSSV = 0
646: C             end if
647: C             if ( JPHOT.ne.0 ) then
648: C                 NCOLPSV = NCOLP
649: C             else
650: C                 NCOLPSV = 0

```

src/mvp/selone.f

```
651:         end if
652:     end if
653: C
654:     return
655: end
```

SAE

src/mvp/source.f

```

1: C/#IF .NOT.SOURCE(NEW)
2:   subroutine SDUMMY( )
3:     return
4: C/#ELSE
5:
6: C/#IF ARGSAVE
7: *   subroutine SOURCE( IOW,A,H,DH,
8: *   N NTGEN, NDEAD, NFISB, JBPRT, NLOST, NEVENT, NMKREG,
9: *   X MCX,MCXP, NUC,NPATOM,NSMIC,
10: *   B XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,KKP,IZZ,IGG,TTT,ITT,KLSF,EEE,IBREG,
11: *   B IBSPC,SMIC,LMIC,KSPI,MMAC,XXYF,YYYF,ZZZF,ZZZF,EEEF,INUF,
12: *   H LEVL,LZZ,LPOS,LCRS,LEVLF,LZZF,LPOSF,LCRSF,IBRGF,IBSPF,KLSFF,XIM,
13: *   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
14: *   B DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
15: *   G MLBZZ, KZMAT, KZREG, MEMZN, SDA,
16: *   G KZDA,KZAA, IPCEL, LTYP, XIMP,
17: *   S LSFFL,NFFL,IZFFL,LSDED,LSLAT,IZLAT,NXLT,
18: *   * SRCSP, SOUR,ENGYB,TIMEB,WGTF,
19: *   T WSUM,XSOC,NCNTR,WCNTR,EBOTX,ETOPX,
20: *   T XSXV, XAVT, AAVT, EAVT, NNCST, INCST,
21: *   X CX,ICX, KLIB1, XLIB1, KLIB2, KLB2S, XLIB2,
22: *   X NNK,NMT, EINCD, CRES, KCRES, SGTAL, MCRES,
23: *   X CXP,ICXP,KLBP1, KLBP2, XLBP2, NMTP,
24: *   D NPDET, NPLEN, JPUSD, XPDET, IPDET, IPDT2,
25: *   S JSPDT, NIMPT, NDMPT, IMSFL, IMNFL,
26: *   S IMZFL, IMDED, IMSET, IMNLT, IMZLT,
27: *   B LZZI, LPOSI, LCRSI, OPTI, PATH, KDET,
28: *   B KLSFI, SMACI, MMACI,
29: *   W LXYZ,LPWRK,LVSTK,MWVEC,MVSTK, MXREJ,
30: c##<2007/03/14:PN3:PN4:
31: c##*   W R,X,Y,Z,IWK1,IWK2,IWK3, EW1, RWK1,IFL,IFI,NNSRC,XSOUR,SMCW,RWF)
32: *   W R,X,Y,Z,IWK1,IWK2,IWK3, EW1, RWK1,IFL,IFI,NNSRC,XSOUR,SMCW,RWF,
33: *   & NUCPN, NUCPN1, NSMICPN, MCXPN, NUC_MAX, SMICPN, NMTPN, CXP,
34: *   & ICXPN, KLBPN1, KLBPN2, XLBP1, EBOTLPN, KPNFG )
35: c##>
36: C/#ELSE
37:   subroutine SOURCE( IOW,A,H,DH,
38:   N NTGEN, NDEAD, NFISB, JBPRT, NLOST, NEVENT, NMKREG,
39:   X MCX,MCXP, NUC,NPATOM,NSMIC,
40:   B XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,KKP,IZZ,
41:   B IGG,TTT,ITT,KLSF,EEE,IBREG,IBSPC, SMIC,LMIC,
42:   B KSPI,MMAC,XXYF,YYYF,ZZZF,ZZZF,EEEF,
43:   B INUF,LEVL,LZZ,LPOS,LCRS,LEVLF,
44:   B LZZF,LPOSF,LCRSF,IBRGF,IBSPF,KLSFF,XIM,
45:   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
46:   B DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
47:   G MLBZZ, KZMAT, KZREG, MEMZN, SDA,
48:   G KZDA,KZAA, IPCEL, LTYP, XIMP,
49:   S LSFFL,NFFL,IZFFL,LSDED,LSLAT,IZLAT,NXLT,
50:   * SRCSP, SOUR,ENGYB,TIMEB,WGTF,
51:   T WSUM,XSOC,NCNTR,WCNTR,EBOTX,ETOPX,
52:   T XSXV, XAVT, AAVT, EAVT, NNCST, INCST,
53:   X CX,ICX, KLIB1, XLIB1, KLIB2, KLB2S, XLIB2,
54:   X NNK,NMT, EINCD, CRES, KCRES, SGTAL, MCRES,
55:   X CXP,ICXP,KLBP1, KLBP2, XLBP2, NMTP,
56:   D NPDET, NPLEN, JPUSD, XPDET, IPDET, IPDT2,
57:   S JSPDT, NIMPT, NDMPT, IMSFL, IMNFL,
58:   S IMZFL, IMDED, IMSET, IMNLT, IMZLT,
59:   B LZZI, LPOSI, LCRSI, OPTI, PATH, KDET,
60:   B KLSFI, SMACI, MMACI,
61:   W LXYZ,LPWRK,LVSTK,MWVEC,MVSTK, MXREJ,
62:   W R,X,Y,Z,IWK1,IWK2,IWK3, EW1, RWK1,IFL,IFI,NNSRC,XSOUR,SMCW,RWF,
63:   % NGENE, NGENE0, NHIST1, NGBAT, IFISB, WFFACT, RNU, WSUMB,
64:   & SMCWP,RP,E2P,DEP,
65:   & CXPI,SMICP,

```

```

66:   & NPTDS,WWD, WD0, WCTRP, WWD2, NPTCS, WWR, WR0,
67:   & WWT, ,WT0, ,WT0A, ,WT0B, ,NUCPT, ,
68:   & NGSP, WSD, NTPT, SWD0, WSC, SWR0, NORDD, NOPS,
69:   & IMTF, RWK2, IWK4,
70: c##<2007/03/14:PN3:PN4:
71:   & NUCPN, NUCPN1, NSMICPN, MCXPN, NUC_MAX, SMICPN, NMTPN, CXP,
72:   & ICXPN, KLBPN1, KLBPN2, XLBP1, EBOTLPN, KPNFG,
73: c##>
74: c+beff2
75:   & WWB, WB0, WSB, SWB0, NPTBE,
76:   & WWBD, WBD0, WSD, SWBD0, WWLD, WLD0, WSLD, SWLD0)
77: c-beff2
78: C/#ENDIF
79: C
80: C=<MVP>=====
81: C PURPOSE: GENERATE PARTICLES (new type)
82: C CALLED IN: ACTION          CALLS: RANU2,VMNTR1,BSVDEC,FISSRC
83: C CALLS:
84: C BSVDEC FISSRC GETMIC GTMICP JUDGE RANU2 SYSSRC VMNT00 VMNT22
85: C VMNTR1 SUNCL GMICN1 GMICP1 GTMICPN, GMICPN1
86: C=====
87: C
88: C IMPLICIT REAL*8 (D)
89: C implicit real*8(A-H,O-Q,S-Z)
90: C implicit real(R)
91: C
92: C parameter( KNUTRN = 1, KPHOTN = 2 )
93: C
94: C include 'INC/_KPIDS'
95: C include 'INC/_NGPS'
96: C
97: C include 'INC/_FLAGS'
98: C include '../shared/INC/_TASKDT'
99: C include '../shared/INC/_WORDL'
100: C
101: C include '../shared/INC/_SIZES'
102: C
103: C ... include _FBANK2 to access XXXF2, YYYF2 etc. in FISSRC
104: C
105: C include 'INC/_FBANK2'
106: C
107: C ... head position of dynamic memory area ....
108: C
109: C real A(*)
110: C real H(*)
111: C real*8 DH(*)
112: C
113: C/#IF INTEGER8
114: * integer*8 NTGEN
115: C/#ELSE
116: integer NTGEN
117: C/#ENDIF
118: integer IFISB(NHIST)
119: real*8 WFFACT
120: C
121: C .... PARTICLE BANK .....
122: C
123: C real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
124: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
125: C real WWW(NBANK), EEE(NBANK), XIM(NBANK)
126: C real*8 TTT(NBANK)
127: C integer ITT(NBANK)
128: C real SMIC(NBANK,NUC,NSMIC), SGTAL(NBANK,NSTAL)
129: c##<2007/03/14:PN3:
130: c## real RNU(NBANK,NUC,3)

```

src/mvp/source.f

```
131:      real SMICPN(NBANK,NUCPN,NSMICPN)
132:      real RNU(NBANK,NUC_MAX,3)
133: c##>
134:      integer KKP(NBANK)
135:      integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
136: &          LPOS(NBANK,NEST), LCRS(NBANK,NEST), KSPI(NBANK,NUC),
137: &          MMAC(NBANK,2), KLSF(NBANK)
138: C
139:      integer IBSPC(NBANK,0:NEST), IBREG(NBANK)
140:      integer LMIC(8)
141: C
142:      real*8 DBNK(NBANK,*)
143:      integer KDBNK(0:MDBNK)
144:      integer IBNK(NBANK,*)
145:      integer KIBNK(0:MIBNK)
146: c##<2007/03/14:PN4:
147:      integer KPNEG(NBANK)
148: c##>
149: C
150: C .... FISSION BANK .....
151: C
152:      real*8 XXXF(NFBNK0), YYYF(NFBNK0), ZZZF(NFBNK0)
153:      real EEEF(NFBNK0)
154:      integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
155: &          LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST), LNUF(NFBNK0)
156: &          KLSFF(NFBNK0)
157:      integer IBSPF(NFBNK0,NEST), IBRGF(NFBNK0)
158:      integer IMTF(NFBNK0)
159:
160:      real*8 DFBK(NFBNK0,*)
161:      integer KDFBK(0:MDFBK)
162:      integer IFBK(NFBNK0,*)
163:      integer KIFBK(0:MIFBK)
164: C
165: C .... TALLY .....
166: C
167:      real*8 WSUM, XSOC(NLENG), XSXV(NLENG,3), WSUMB, RSRC
168:      real XAVT(3), AAVT(3), EAVT
169:      real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
170:      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
171: CCCC real ENGYB(NGP1+1), ENGPB(NGP2+1), TIMEB(NTIME+1)
172:      real ENGYB(NGP1+NPKIND)
173:      real TIMEB(NTIME+1)
174:      integer INCST(NUC,*)
175: C
176: C .... VARIANCE REDUCTION DATA
177: C
178:      real XIMP(NGROUP,NREG)
179: C
180: C .... STACKS ...(FREE FLIGHT & MORGUE) .....
181: C
182:      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSDED(NBANK)
183: C
184: C .... STACKS ...(LATTICE SEARCH) .....
185: C
186:      integer LSLAT(NBANK), NXLT(*), IZLAT(NBANK)
187: C
188: C .... SOURCE DATA .....
189: C
190:      integer SRCSP(*)
191:      integer MEMZN(NMEMS,NSOUR)
192:      real EINCD
193:      real*8 SOUR(NSOUR)
194: C
195: C .... GEOMETRY ARRAY DATA
196: C
197:      integer KZMAT(NZONE), KZREG(NZONE), KZDA(*), KZAA(*),
198: &          MLBZZ(NZONE), IPCEL(*), LTYP(*)
199:      real*8 SDA(*)
200: C
201: C CROSS SECTION ..... (NEUTRON)
202: C
203: CC      real EBOT, ETOP
204: CC      real EBOTP, ETOPP
205:      real*8 ETOPX(KPLIM), EBOTX(KPLIM)
206:      real CX(MCX)
207:      integer ICX(MCX)
208: c##<2007/03/14:PN3:
209: c##      real XLIB1(NUC,9), XLIB2(NUC,NMT,4)
210: c##      integer KLIB1(NUC,27), KLIB2(NUC,NMT,21)
211:      real XLIB1(NUC,11), XLIB2(NUC,NMT,4)
212:      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
213: c##>
214: C
215: C .... CROSS SECTION & POINTERS ( PHOTON ).....
216: C
217:      real CXP(MCXP)
218:      integer ICXP(MCXP), KLBP1(NPATOM,20), KLBP2(NPATOM,NMTP,6)
219:      real XLBP2(NPATOM,NMTP,2)
220: c##<2007/03/14:PN3:
221:      real CXPN(MCXPN), XLBPN1(NUCPNI,6)
222:      integer ICXPN(MCXPN), KLBP1(NUCPNI,16), KLBP2(NUCPNI,NMTPN,13)
223: c##>
224: C
225:      real CRES(MCRES)
226:      integer KCRES(NSTAL,4)
227: C
228: c##<2007/03/14:PN3:
229: c##      real WGTF(NREG)
230:      real WGTF(NREG,2)
231: c##>
232: C
233: C .... PERTURBATION .....
234: C
235: c ... particle bank
236: c
237: c
238:      real SMICP(NBANK,NUC,NSMIC)
239: c
240: c ... cross section
241: c
242:      real CXPl(MCX)
243: c
244: c ... working arrays
245: c
246:      real SMCWP(NBANK,NSMIC)
247:      real RP(7*NBANK), E2P(NBANK)
248:      integer DEP(NBANK)
249: c
250: c ... perturbation
251: c
252:      real*8 WWD(NBANK,NPTDS,NORDD), WWD2(NBANK,NPTDS)
253:      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
254:      real*8 WWR(NBANK,NPTCS)
255:      real*8 WSC(NBANK,0:NGSP,NPTCS)
256:      real*8 WD0(NFBNK0,0:NGSP,NPTDS,NOPSDS)
257:      real*8 WCTRP(NUCPT,7,NTPT)
258:      real*8 WR0(NFBNK0,0:NGSP,NPTCS)
259:      real*8 WWT(NBANK,NPTDS), WT0(NFBNK0)
260:      real*8 WTOA(NFBNK0), WTOB(NFBNK0)
```

src/mvp/source.f

```

261:      real*8 SWD0(0:NGSP,NPTDS,NOPSDS)
262:      real*8 SWR0(0:NGSP,NPTCS)
263: c+beff2
264:      real*8 WWB(NBANK)
265:      real*8 WB0(NFBNK0,0:NGSP)
266:      real*8 WSB(NBANK,0:NGSP)
267:      real*8 SWB0(0:NGSP)
268:      real*8 WWBD(NBANK)
269:      real*8 WBD0(NFBNK0,NGSP)
270:      real*8 WBSD(NBANK,NGSP)
271:      real*8 SWBD0(NGSP)
272:      real*8 WWLD(NBANK)
273:      real*8 WLD0(NFBNK0,NGSP)
274:      real*8 WSLD(NBANK,NGSP)
275:      real*8 SWLD0(NGSP)
276: c-beff2
277: C
278: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR
279: C
280:      real*8 XPDET(NPLEN,NPDET)
281:      integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
282:      integer JPUSD(NPDET), JSPDT(NPDET)
283: C
284: C**** STACK FOR IMAGINARY PARTICLE
285: C
286:      integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFI(*), IMDED(IMPMAX),
287:      &      IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
288: C
289: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
290: C
291:      integer LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
292: C****
293:      real*8 OPTI(IMPMAX), PATH(IMPMAX)
294:      integer KDETP(IMPMAX), KLSFI(IMPMAX)
295: C
296: C**** cross section and geometry data
297: C
298:      real      SMACI(*)
299:      integer   MMACI(*)
300: C
301: C ... working arrays ....
302: C
303: C      .... LENGTH OF R MUST BE 7*NBANK
304: C      R, EW1 and IFL can be used as real*8 data
305: C
306:      real R(7*NBANK), EW1(NBANK), RWK1(NBANK)
307:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
308:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK)
309:      logical IFL(NBANK), IFI(NBANK)
310:      integer NNSRC(NSOUR)
311:      real*8 XSOUR(NSOUR)
312:      real SMCW(NBANK,NSMIC)
313:      real RWF(NHIST)
314:      real RWK2(NBANK)
315:      integer IWK4(NBANK)
316: C
317: C ... pointers to working array
318: C
319:      integer LXYZ(10)
320: C
321: C ---> integer      lx,ly,lz,la,lb,lc, le,lg,lt,lw
322: C
323:      integer LPWRK(MWVEC), LVSTK(MVSTK)
324: C
325:      include '../shared/INC/_PMLATT'

```

```

326: C
327: C --- local data ----
328: C
329:      integer JVSMP(10)
330: C
331:      integer IKSORT(KPLIM+1), KSORT(KPLIM)
332: C
333: C ... local data whose value must be saved call to call
334: C
335: C/#IF PARA(SX*)
336: *      LOCAL COMMON /LSSOUR/ XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN
337: C/#ELSEIF PARA(CRAY)
338: *      TASK COMMON /LSSOUR/ XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN
339: C/#ENDIF
340:      data XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN /8*0.0d0/
341: C
342: C --- statement functions
343: C
344:      include '../shared/INC/_SFLATT'
345: C
346: C --- statement function to convert pointer to real data to
347: C      pointer to real*8 data
348: C
349: C/#IF WORD(64)
350: *      LDBLE(L)      = L
351: C/#ELSE
352: *      LDBLE(L)      = L/MWORD + MWORD - 1
353: C/#ENDIF
354: C
355: C/#IF ARGSAVE
356: *      return
357: C
358: C -----
359: C
360: *      entry SOURCE( NGENE, NGENE0, NHIST1, NGBAT, IFISB, WFFACT, RNU,
361: *      &      WSUMB )
362: C/#ENDIF
363: C
364: C -----
365: C .... IF ALL PARTICLES ARE DEAD, CLEAR UP DEAD PARTICLE STACK ....
366: C -----
367:      if ( NDEAD.eq.NBANK ) then
368:      do 100 I = 1, NBANK
369:      LSDED(I)      = I
370:      continue
371: C
372:      NDIMPT = IMPMAX
373:      if ( JPTDT.ne.0 ) then
374:      do 105 I = 1, IMPMAX
375:      IMDED(I)      = I
376:      continue
377:      end if
378:      else
379: c##<2007/03/14:PN3:
380: c##      write(IOW,*)
381: c## &      'XXX(SOURCE) Program error?: NDEAD is not equal to NBANK !!!',
382: c## &      ' NDEAD=', NDEAD, ' NBANK=', NBANK, ' NTGEN= ', NTGEN
383: c##      write(IOW,*) '      Something wrong happened, STOP !!!'
384: c##      write(IOW,'(1X,A,A,I8,A,I8,A,I8)')
385: c## &      'XXX(SOURCE) Program error?: NDEAD is not equal to',
386: c## &      ' NBANK !! NDEAD=', NDEAD, ' NBANK=', NBANK,
387: c## &      ' NTGEN=', NTGEN
388: c##      write(IOW,'(1X,A)') '      Something wrong happened, STOP !!!'
389: c##>
390:      stop 666

```

src/mvp/source.f

```

391:      end if
392:
393: C-----
394: C .... Print number of histories processed in a batch for first
395: C      sub-batch only.
396: C-----
397:      if ( NGENE0.eq.0 ) then
398: C/#IF PARA
399:
400: C/#IF PARA(SX* CRAY)
401: *      call MVPSYNC_LOCK(2)
402: C/#ENDIF
403:      if ( JBPRNT.ne.0 ) then
404: cc      write(IOW,7000) IDTASK, NHIST1, IRAND
405:      write(IOW,7000) IDTASK, NHIST1
406:      end if
407: c7000   format(1X,' TASK ',I5,' GENERATE ',I6,' SOURCES.  IRAND = ',
408: c      &      I11)
409: 7000   format(1X,' TASK ',I5,' GENERATE ',I6,' SOURCES.')
410:
411: C/#IF PARA(SX* CRAY)
412: *      call MVPSYNC_UNLOCK(2)
413: C/#ENDIF
414:
415: C/#ELSE
416:
417:      if ( JBPRNT.ne.0 ) then
418: c      write(IOW,7020) NHIST1, IRAND
419: c7020   format(1X,' GENERATE ',I6,' SOURCES.  IRAND = ',I11)
420:      write(IOW,7020) NHIST1
421: 7020   format(1X,' GENERATE ',I6,' SOURCES.')
422:      end if
423: C/#ENDIF
424: C
425: C      ... select fission source of batch for eigenvalue problem ...
426: C
427:      if ( JEIGN.ne.0.and.NBATCH.gt.0 ) then
428:      call FISSET( IOW, IFISB, WFACT, IRAND, NFISB, NHIST,
429:      &      NHIST1, NHSUB, NBANK, NZONE, NFBANK, NFBK0, NBATCH,
430:      &      NREG, NEST, WGTF, KZREG, XSOC, NLENG, XXXF, YYYF,
431:      &      ZZZF, EEEF, INUF, IZZF, LEVLF, LZZF, LPOSF, LCRSF,
432:      &      IBRGF, IBSPF, KLSFF, DFBK, KDFBK, MDFBK, IFBK,
433:      &      KIFBK, MIFBK, H(LXXXF2), H(LYYYF2), H(LZZZF2),
434:      &      H(LEEF2), H(LINUF2), H(LIZZF2), H(LLEVLF2),
435:      &      H(LLZZF2), H(LLPOSF2), H(LLCRSF2), H(LIBRGF2),
436:      &      H(LIBSPF2), H(LKLSFF2), H(LDFBK2), KDFBK2, MDFBK2,
437:      &      H(LIFBK2), KIFBK2, MIFBK2, RWF(1) )
438:      end if
439: C
440: C      ... initialize variables for batchwise source parameter average
441: C
442:      XAV      = 0.0D0
443:      YAV      = 0.0D0
444:      ZAV      = 0.0D0
445:      AAV      = 0.0D0
446:      BAV      = 0.0D0
447:      CAV      = 0.0D0
448:      EAV      = 0.0D0
449: C      DWSUMN = 0.0D0
450:      end if
451: C-----
452: C ..... NGENE0 : sum of NGENE for use in TALSUM .....
453: C      NFB0 : use fission bank for index IFISB(NFB0+1:NFB0+NGENE)
454: C      for eigenvalue problem.
455: C-----

```

```

456: CCCC  NGENE0 = NGENE
457:      NFB0 = NGENE0
458:      NGENE0 = NGENE0 + NGENE
459:      NGBAT = NGBAT - NGENE
460: C
461:      if ( JVMNT.ne.0 ) call VMNTRI( 1, NGENE )
462: C
463:      JMP      = 0
464:      KKPID     = 0
465: C
466: C
467: C =====
468: C FIXED SOURCE PROBLEM OR INITIAL SOURCE FOR EIGENVALUE PROBLEM
469: C =====
470: C
471: C
472:      if ( JEIGN.eq.0 .or. NBATCH.le.0 ) then
473: C-----
474: C .... DETERMINE PARTICLE NUMBERS FROM EACH SOURCE IF NSOUR > 1 .....
475: C-----
476:      if ( NSOUR.gt.1 ) then
477:          NSS      = 0
478: C
479: C      NNSRC is integer part of NGENE*SOUR(N), and
480: C      XSOUR is cumulative sum of decimal part of it.
481: C
482: *VOCL LOOP,SCALAR
483:      do 110 N = 1, NSOUR
484:          NNSRC(N) = NGENE*SOUR(N)
485:          XSOUR(N) = DBLE(NGENE)*SOUR(N) - DBLE(NNSRC(N))
486:          if ( N.gt.1 ) XSOUR(N) = XSOUR(N-1) + XSOUR(N)
487:          NSS      = NSS + NNSRC(N)
488:      110      continue
489: C-----
490: C .... If the sum of NNSRC is not equal to NGENE, number of source
491: C      particles are corrected with probabilities proportional to the
492: C      decimal parts of NGENE*SOUR of each source. (XSOUR).
493: C
494: C      NSS may be greater than NGENE for machines int(N*R) (R<1) may be N
495: C      in roundoff treatment (should be rare case).
496: C      in such a case NNSRC is reduced.
497: C-----
498:      NSS1      = NGENE - NSS
499:      if ( NSS1.gt.0 ) then
500:          call RANU2( IRAND, R, NSS1, ICON )
501: *VOCL LOOP,SCALAR
502:      do 140 K = 1, NSS1
503:          R(K) = R(K)*XSOUR(NSOUR)
504:      do 120 L = 1, NSOUR
505:          if ( XSOUR(L).gt.R(K) ) go to 130
506:      120      continue
507:          L      = NSOUR
508:      130      NNSRC(L) = NNSRC(L) + 1
509:      140      continue
510: C
511: C      ... sum of NNSRC > NGENE: reject NSS-NGENE sources
512: C      else
513:      call RANU2( IRAND, R, ABS(NSS1), ICON )
514:      NSS2      = NSS
515:      do 170 K = 1, ABS(NSS1)
516:          IKIL    = MIN(INT(R(K)*NSS2+1),NSS2)
517:          NSS3     = 0
518:      do 150 N = 1, NSOUR
519:          NSS3     = NSS3 + NNSRC(N)
520:          if ( NSS3.ge.IKIL ) then

```

src/mvp/source.f

```

521:          NNSRC(N) = NNSRC(N) - 1
522:          go to 160
523:        end if
524:      150    continue
525:      160    NSS2 = NSS2 - 1
526:      170    continue
527:    end if
528:  else
529:    NNSRC(1) = NGENE
530:  end if
531: C-----
532: C ... GENERATE PARTICLES .....
533: C-----
534:    NSS = 0
535:    IST = 0
536:    IPP = NDEAD
537:    ILOST = 0
538: C
539:    LXD = LDBLE(LXYZ(1)) - 1
540:    LYD = LDBLE(LXYZ(2)) - 1
541:    LZD = LDBLE(LXYZ(3)) - 1
542:    LAD = LDBLE(LXYZ(4)) - 1
543:    LBD = LDBLE(LXYZ(5)) - 1
544:    LCD = LDBLE(LXYZ(6)) - 1
545:    LED = LDBLE(LXYZ(7)) - 1
546:    LGD = LDBLE(LXYZ(8)) - 1
547:    LTD = LDBLE(LXYZ(9)) - 1
548:    LWD = LDBLE(LXYZ(10)) - 1
549: C
550:    do 390 N = 1, NSOUR
551:      NN = NNSRC(N)
552: Ccc      IF(JMNTR.NE.0)
553: Ccc &      WRITE(IOW,*) ' SOURCE # ',N,' : ',NN,' PARTICLES'
554:      IPP = IPP - NN
555: C
556: Ccccc    WXREJ = 300
557: Ccccc    MXREJ = 300
558: C
559: C ... subroutines under 'SYSSRC' can use working array other than
560: C nnsrc,xsour & iwk1,iwk2
561: C
562:    NRWK = 7
563: C
564:    call SYSSRC( 'MVP', A, N, NN, NPK, JDEBG, JLATT, JTLT,
565: &      JHLAT, SRCSP, SRCSP, SRCSP, MXREJ, H, H, DH, IRAND,
566: &      SDA, NSDA, LXYZ(1), LXYZ(2), LXYZ(3), LXYZ(4),
567: &      LXYZ(5), LXYZ(6), LXYZ(7), LXYZ(8), LXYZ(9),
568: &      LXYZ(10), JVSMP, MWVEC, MVSTK, LPWRK, LVSTK, R, R,
569: &      NRWK, NERROR )
570: C
571: C
572:    if ( JLATT.ne.0 ) then
573:      do 180 I = 1, NN
574:        LEVL(LSDED(IPP+I)) = 0
575:      180    continue
576:    end if
577:    do 190 I = 1, NN
578:      KLSF(LSDED(IPP+I)) = 0
579:    190    continue
580: C
581: C .... transfer sampled data to bank ....
582: C
583:    DWSUMN = 0.0D0
584:    do 200 I = 1, NN
585:      IWK1(NSS+I) = LSDED(IPP+I)

```

```

586: C
587:    KKP(IWK1(NSS+I)) = NPK
588: C
589:    XXX(IWK1(NSS+I)) = DH(LXD+I)
590:    YYY(IWK1(NSS+I)) = DH(LYD+I)
591:    ZZZ(IWK1(NSS+I)) = DH(LZD+I)
592: C
593:    AAA(IWK1(NSS+I)) = DH(LAD+I)
594:    BBB(IWK1(NSS+I)) = DH(LBD+I)
595:    CCC(IWK1(NSS+I)) = DH(LCD+I)
596: C
597:    WWW(IWK1(NSS+I)) = DH(LWD+I)
598: C
599: C
600:    DWSUMN = DWSUMN + WWW(IWK1(NSS+I))
601: C
602:    if ( JTIME.ne.0 ) TTT(IWK1(NSS+I)) = DH(LTD+I)
603: C
604:    if ( JPERT.ne.0 ) then
605: C ... initialize additional weights ...
606: C
607:      do IPT = 1, NPSTD
608:        do IOD = 1, NORDDS
609:          WWD(IWK1(NSS+I),IPT,IOD) = 0.0d0
610:        end do
611:        WWD2(IWK1(NSS+I),IPT) = 0.0d0
612:        WWT(IWK1(NSS+I),IPT) = 0.0d0
613:        do ISP = 0, NGSP
614:          do IOP = 1, NOPSDS
615:            WSD(IWK1(NSS+I),ISP,IPT,IOP) = 0.0d0
616:          end do
617:        end do
618:      end do
619:    end do
620: C
621:    do IPT = 1, NPSTCS
622:      WWR(IWK1(NSS+I),IPT) = 1.0d0
623:      do ISP = 0, NGSP
624:        WSC(IWK1(NSS+I),ISP,IPT) = 0.0d0
625:      end do
626:    end do
627: C
628:    end if
629:  200    continue
630: C
631:    if ( JMNTR.ne.0.and.JBPRNT.ne.0 ) then
632:      write(IOW,'(1x,a,i4,a,i7,a,e12.5,a,e12.5,a)')
633: &      ' SOURCE # ', N, ' : ', NN,
634: &      ' PARTICLES. WEIGHT SUM ', DWSUMN, ' (AVERAGE ',
635: &      DWSUMN/NN, ' )'
636:    end if
637: C
638: C ... energy value is sampled ...
639: C
640:    if ( JVSMP(7).ne.0 ) then
641: C ... neutron ...
642: C      if ( NPK.eq.2**0 ) then
643: C        EBT = EBOT
644: C        ETP = ETOP
645: C
646: C ... photon ...
647: C      else if ( NPK.eq.2**1 ) then
648: C        EBT = EBOTP
649: C        ETP = ETOPP
650: C      end if

```

src/mvp/source.f

```

651:      EBT      = EBOTX(NPK)
652:      ETP      = ETOPX(NPK)
653:
654:      do 210 I = 1, NN
655:         EW1(I) = MAX(EBT,MIN(DH(LED+I),ETP))
656:      210      continue
657: C
658: C      ... determine energy bins ....
659: C
660: C      if ( NPK.eq.2**0 ) then
661: C         call BSVDEC( ENGYB, NGP1+1, EW1, IWK2, NN )
662: C      else
663: C         call BSVDEC( ENGPB, NGP2+1, EW1, IWK2, NN )
664: C         if ( JNEUT.ne.0 ) then
665: C            do 220 I = 1, NN
666: C               IWK2(I) = IWK2(I) + NGP1
667: C            220      continue
668: C         end if
669: C      end if
670: C
671: C      call BSVDEC( ENGYB(KENGP(NPK)), NGP(NPK)+1, EW1, IWK2, NN
672: C      &
673: C      NNNN1 = KNGP(NPK) - 1
674: C      do 220 I = 1, NN
675: C         IWK2(I) = MIN(NGP(NPK),MAX(1,IWK2(I))) + NNNN1
676: C      220      continue
677: C
678: C      do 230 I = 1, NN
679: C         EEE(IWK1(NSS+I)) = EW1(I)
680: C         IGG(IWK1(NSS+I)) = IWK2(I)
681: C      230      continue
682: C
683: C      ... energy group is sampled (for backward compatibility) ...
684: C
685: C      else if ( JVSMP(8).ne.0 ) then
686: C
687: C         call RANU2( IRAND, R, NN, ICON )
688: C
689: C      ... neutron ...
690: C      if ( NPK.eq.2**0 ) then
691: C         do 240 I = 1, NN
692: C            IIG = DH(LGD+I)
693: C            IGG(IWK1(NSS+I)) = IIG
694: C            EEE(IWK1(NSS+I)) = ENGYB(IIG)*
695: C              (ENGYB(IIG+1)/ENGYB(IIG))**R(I)
696: C      240      &      continue
697: C      ... photon ...
698: C      else if ( NPK.eq.2**1 ) then
699: C         do 250 I = 1, NN
700: C            IIG = DH(LGD+I)
701: C            IGG(IWK1(NSS+I)) = IIG + NGP1
702: C            EEE(IWK1(NSS+I)) = ENGPB(IIG)*
703: C              (ENGPB(IIG+1)/ENGPB(IIG))**R(I)
704: C      250      &      continue
705: C      end if
706: C
707: C      do 240 I = 1, NN
708: C         IIG = DH(LGD+I)
709: C         IGG(IWK1(NSS+I)) = IIG + NGP(NPK) - 1
710: C         IJ = KENGP(NPK) - 1 + IIG
711: C         EEE(IWK1(NSS+I)) = ENGYB(IJ)*
712: C           (ENGYB(IJ+1)/ENGYB(IJ))**R(I)
713: C      240      &      continue
714: C      end if
715: C

```

```

716: C      ... determine time bin ....
717: C
718: C      if ( JTIME.ne.0 ) then
719: C         call BS0ISD( TIMEB, NTIME+1, DH(LTD+1), IWK2(1), NN )
720: C      *VOCL LOOP,NOVREC
721: C      do 250 I = 1, NN
722: C         ITT(IWK1(NSS+I)) = MAX(1,MIN(NTIME,IWK2(I)))
723: C      250      continue
724: C      end if
725: C
726: C-----
727: C      === DETERMINE STARTING ZONE NUMBERS =====
728: C-----
729: C
730: C      <COMMENT> 'NSST' INDICATES THE NUMBER OF REALLY GENERATED SOURCE
731: C      PARTICLES ( NSST - NSS IS THE NUMBER OF 'REAL' SOURCE
732: C      PARTICLES (I.E. UNLOST SOURCE PARTICLES)).
733: C      TO BE USED IN THE UNCOLLIDED FLUX CALCULATION.
734: C
735: C      NSST = NSS
736: C
737: C      do 260 I = 1, NN
738: C         IWK2(I) = IWK1(NSS+I)
739: C         X(I) = XXX(IWK2(I)) + 1.0D-8*AAA(IWK2(I))
740: C         Y(I) = YYY(IWK2(I)) + 1.0D-8*BBB(IWK2(I))
741: C         Z(I) = ZZZ(IWK2(I)) + 1.0D-8*CCC(IWK2(I))
742: C      260      continue
743: C      II = NN
744: C      KKZ = 1
745: C      MEM = 1
746: C      MMZ = NZONE
747: C      if ( JLATT.ne.0 ) MMZ = IPCEL(1) - 1
748: C      c##<2007/03/14:PN3:
749: C      IM = 0
750: C      c##>
751: C      do 340 M = 1, MMZ
752: C         if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
753: C            if ( MEM.le.NMEMS.and.MEMZN(MEM,N).ne.0 ) then
754: C               IZ = MEMZN(MEM,N)
755: C               IM = 1
756: C            else
757: C               IM = 0
758: C               do 280 KZ = KKZ, MMZ
759: C                  do 270 K = 1, NMEMS
760: C                     if ( MEMZN(K,N).ne.0.and.KZ.eq.MEMZN(K,N) )
761: C                        &      go to 280
762: C                  270      continue
763: C                  go to 290
764: C                  continue
765: C                  KZ = MMZ
766: C                  IZ = KZ
767: C                  KKZ = IZ + 1
768: C               end if
769: C            else
770: C               IZ = M
771: C            end if
772: C
773: C      .... JUDGE WHETHER PARTICLES BELONG TO ZONE IZ OR NOT. ....
774: C
775: C      call JUDGE( 'SOURCE', IZ, II, X, Y, Z, IFI, SDA, KZDA,
776: C      &      KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
777: C      &      .false., DUMMY, DUMMY, DUMMY )
778: C
779: C-----
780: C      ..... SEND PARTICLES TO FLIGHT STACK .....

```


src/mvp/source.f

```

781: C-----
782: C
783:       INN      = 0
784:       if ( KZMAT(IZ).ge.0 ) then
785:         IFFL     = NFFL(NZONE+1)
786: *VOCL LOOP,NOVREC
787:       do 300 I = 1, II
788:         if ( IFI(I) ) then
789:           IZZ(IWK2(I)) = IZ
790:           INN      = INN + 1
791:           LSFFL(IFFL+INN) = IWK2(I)
792:           IZFFL(IFFL+INN) = IZ
793:           IREG      = KZREG(IZ)
794: C       ... set IBREG anytime (from Jan 2000)
795:           IBREG(IWK2(I)) = IREG
796:           if ( JTLT.ne.0 ) then
797: CCCCCC           IBREG(IWK2(I)) = IREG
798:           IBSPC(IWK2(I),0) = 0
799:           end if
800: C .....
801:           if ( JIMPT.ne.0 ) then
802:             XIM(IWK2(I)) = XIMP(IGG(IWK2(I)),IREG)
803:           end if
804:           if ( JPTDT.ne.0 ) IWK1(NSST+INN) = IWK2(I)
805:         end if
806:       300 continue
807:
808:       NFFL(IZ) = NFFL(IZ) + INN
809:       NFFL(NZONE+1) = NFFL(NZONE+1) + INN
810: C
811:       if ( JPTDT.ne.0 ) NSST = NSST + INN
812: C
813: C-----
814: C ..... IF PARTICLES ARE IN A LATTICE, SEND THEM TO LATTICE STACK.
815: C-----
816: C
817: CCCCCC       else if ( KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998 ) then
818:       else if ( ISLATT(KZMAT(IZ)) ) then
819:         ILAT      = NXLT(NLBZ+1)
820: *VOCL LOOP,NOVREC
821:       do 310 I = 1, II
822:         if ( IFI(I) ) then
823:           IZZ(IWK2(I)) = IZ
824:           INN      = INN + 1
825:           LSLAT(ILAT+INN) = IWK2(I)
826:           IZLAT(ILAT+INN) = MLBZZ(IZ)
827:           IREG      = KZREG(IZ)
828: C       ... set IBREG anytime (from Jan 2000)
829:           IBREG(IWK2(I)) = IREG
830:           if ( JTLT.ne.0 ) then
831:             IBSPC(IWK2(I),0) = 0
832:           end if
833:           if ( JIMPT.ne.0 ) then
834:             if ( IREG.gt.0 ) then
835:               XIM(IWK2(I)) = XIMP(IGG(IWK2(I)),IREG)
836:             else
837:               XIM(IWK2(I)) = 1.
838:             end if
839:           end if
840:           if ( JPTDT.ne.0 ) IWK1(NSST+INN) = IWK2(I)
841:         end if
842:       310 continue
843:       NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + INN
844:       NXLT(NLBZ+1) = ILAT + INN
845: C

```

```

846:       if ( JPTDT.ne.0 ) NSST = NSST + INN
847: C
848: C-----
849: C ..... GENERATING PARTICLES IN INVALID MATERIAL ? .....
850: C-----
851: C
852:       else
853:       do 320 I = 1, II
854:         if ( IFI(I) ) then
855:           INN      = INN + 1
856:           ILOST     = ILOST + 1
857: C1999/8/5           LSDED(IPP+I) = IWK2(I)
858:           LSDED(IPP+INN) = IWK2(I)
859:         end if
860:       320 continue
861:       IPP      = IPP + INN
862: C
863: C/#IF PARA(SX* CRAY)
864: *       call MVPSYNC_LOCK(2)
865: C/#ENDIF
866:       if ( INN.gt.0 ) write(IOW,7040) N, KZMAT(IZ), IZ, INN
867:       7040 format(/1X,' !!! CAUTION: YOU ARE GOING TO',
868:         &          ' GENERATE PARTICLES IN A ZONE WHICH ',
869:         &          ' PARTICLES SHOULD NOT EXIST !!! '/1X,
870:         &          ' SOURCE NO. ',I3,' MAT = ',I5,' ZONE = ',I5,
871:         &          ', ',I8,
872:         &          ' PARTICLES. THEY ARE TREATED AS LOST!!')
873: C/#IF PARA(SX* CRAY)
874: *       call MVPSYNC_UNLOCK(2)
875: C/#ENDIF
876:       end if
877: C
878: C-----
879: C ..... MEMORIZE NEWLY FOUND ZONE .....
880: C-----
881: C
882:       if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
883:       if ( IM.ne.0 ) then
884:         MEM      = MEM + 1
885:       else if ( INN.ne.0.and.MEM.le.NMEMS
886:         &          .and.MEMZN(MEM,N).eq.0 ) then
887:         MEMZN(MEM,N) = IZ
888:         MEM      = MEM + 1
889:       end if
890:       end if
891: C
892: C-----
893: C ..... COMPRESS .....
894: C-----
895: C
896:       if ( INN.lt.II ) then
897:         INNN      = 0
898: *VOCL LOOP,NOVREC
899:       do 330 I = 1, II
900:         if ( .not.IFI(I) ) then
901:           INNN      = INNN + 1
902:           IWK2(INNN) = IWK2(I)
903:           X(INNN) = X(I)
904:           Y(INNN) = Y(I)
905:           Z(INNN) = Z(I)
906:         end if
907:       330 continue
908:       II      = INNN
909:       else
910:       II      = 0

```

src/mvp/source.f

```

911:          go to 350
912:          end if
913: 340      continue
914: C
915: C-----
916: C ..... ZONES NOT FOUND !! (LOST) .....
917: C-----
918: C
919: 350      continue
920: C
921:          if ( II.ne.0 ) then
922: C/#IF PARA(SX* CRAY)
923: *          call MVPSYNC_LOCK(2)
924: C/#ENDIF
925:          write(IOW,7060) II, N, (XXX(IWK2(I)),YYY(IWK2(I)),
926: &          ZZ2(IWK2(I)),AAA(IWK2(I)),BBB(IWK2(I)),
927: &          CCC(IWK2(I)),I=1,II)
928: 7060      format(/1X,' *** ',I5,
929: &          ' PARTICLES ARE LOST IN SOURCE ROUTINE!'/1X,
930: &          ' SOURCE NUMBER = ',I5/(1X,' X = ',1PE12.5,
931: &          ' Y = ',E12.5,' Z = ',E12.5,' MU = ',E12.5,
932: &          ' ETA = ',1PE12.5,' XI = ',E12.5))
933: C/#IF PARA(SX* CRAY)
934: *          call MVPSYNC_UNLOCK(2)
935: C/#ENDIF
936:          do 360 I = 1, II
937:              LSDED(IPP+I) = IWK2(I)
938: 360      continue
939:          ILOST = ILOST + II
940:          IPP = IPP + II
941:          end if
942: C
943: C-----
944: C ..... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS ....
945: C-----
946: C
947: C      .... KSOUR2 : =0   ISOTROPIC SOURCE
948: C      .NE.0   SOURCE TYPE
949: C
950:          if ( JPTDT.ne.0 ) then
951:              KSOUR2 = 0
952:              KS = 0
953:              if ( KS.eq.2 .or. KS.eq.9 ) KSOUR2 = KS
954:              DUMAX = 1.0D0
955:              DUMIN = -1.0D0
956:              NSORS = NSST-NSS
957:              if ( NPK.eq.2**0 ) then
958:                  JNP = 1
959:              else
960:                  JNP = 2
961:              end if
962: C
963:          call SUNCL( IOW, A, H, NSORS, IWK1(NSS+1), KSOUR2,
964: &          RDUMMY, DUMAX, DUMIN, JNP,
965: &          JPUSD, JSPDT, XPDET, IPDET, IPDT2,
966: &          IMSFL, IMNFL, IMZFL,
967: &          IMDED, IMSLT, IMNLT, IMZLT,
968: &          KDBNK(0),KIBNK(0), TIMEB, KZMAT,
969: &          XXX , YYY , ZZZ , AAA , BBB , CCC ,
970: &          EEE , WWW , IZZ , IGG , TTT , ITT ,
971: &          LEVL , LZZ , LPOS , LCRS ,
972: &          DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
973: &          LZZI , LPOSI, LCRSI,
974: &          OPTI , PATH , KDETP, KLSFI, SMACI, MMACI,
975: &          X, Y, Z, EW1, IFL, R,

```

```

976: &          H(LXYZ(1)),H(LXYZ(2)),H(LXYZ(3)),H(LXYZ(4)),
977: &          H(LXYZ(5)),H(LXYZ(6)),H(LXYZ(7)),H(LXYZ(8)),
978: &          H(LXYZ(9)),H(LXYZ(10)),
979: &          IWK2, IWK3, R(2*NBANK+1), R(3*NBANK+1),
980: &          R(4*NBANK+1), SMCW )
981: C
982:          end if
983: C
984: C ... save source set # if necessary as optional bank parameter
985: C
986:          if ( KIBNK(1).ne.0 ) then
987: *VOCL LOOP,NOVREC
988:          do 370 I = 1, NN
989:              IBNK(IWK1(NSS+I),KIBNK(1)) = N
990: 370      continue
991:          end if
992: C
993: C ... set flight path counter to negative for source particle
994: C
995:          if ( KIBNK(5).ne.0 ) then
996: *VOCL LOOP,NOVREC
997:          do 380 I = 1, NN
998:              IBNK(IWK1(NSS+I),KIBNK(5)) = -1
999: 380      continue
1000:          end if
1001: C
1002:          NSS = NSS + NN
1003: 390      continue
1004: C
1005: C-----
1006: C .... UPDATE SUM OF WEIGHTS .....
1007: C-----
1008: C
1009:          do 400 I = 1, NGENE
1010:              WSUM = WSUM + WWW(IWK1(I))
1011: 400      continue
1012: C
1013: C ... save source weight sum of a batch (first batch only)...
1014: C
1015:          if ( JEIGN.ne.0.and.NGBAT.eq.0 ) XSOC(NBATCH+1) = WSUM
1016: C
1017:          JMP = 0
1018:          KKPID = KKP(IWK1(1))
1019:          do 410 I = 2, NGENE
1020:              if ( KKP(IWK1(I)).ne.KKPID ) then
1021:                  JMP = 1
1022:                  go to 420
1023:              end if
1024: 410      continue
1025: 420      continue
1026:          if ( JMP.eq.0 ) then
1027:              do 430 I = 1, NGENE
1028:                  WCNTR(1,KKPID) = WCNTR(1,KKPID) + WWW(IWK1(I))
1029: 430      continue
1030:                  NCNTR(1,KKPID) = NCNTR(1,KKPID) + NGENE
1031:          else
1032:              do 450 IK = 1, KPLIM
1033:                  if ( JKPAR(IK).ne.0 ) then
1034:                      do 440 I = 1, NGENE
1035:                          if ( KKP(IWK1(I)).eq.IK ) then
1036:                              NCNTR(1,IK) = NCNTR(1,IK) + 1
1037:                              WCNTR(1,IK) = WCNTR(1,IK) + WWW(IWK1(I))
1038:                          end if
1039: 440      continue
1040:                      end if

```

src/mvp/source.f

```

1041: 450      continue
1042:      end if
1043:
1044: C      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
1045: C          do 420 I = 1, NGENE
1046: C              if ( IGG(IWK1(I)).le.NGPI ) then
1047: C                  NCNTR(1,KNUTRN) = NCNTR(1,KNUTRN) + 1
1048: C                  WCNTR(1,KNUTRN) = WCNTR(1,KNUTRN) + WWW(IWK1(I))
1049: C              else
1050: C                  NCNTR(1,KPHOTN) = NCNTR(1,KPHOTN) + 1
1051: C                  WCNTR(1,KPHOTN) = WCNTR(1,KPHOTN) + WWW(IWK1(I))
1052: C              end if
1053: C 420      continue
1054: C      else if ( JNEUT.ne.0 ) then
1055: C          do 430 I = 1, NGENE
1056: C              WCNTR(1,KNUTRN) = WCNTR(1,KNUTRN) + WWW(IWK1(I))
1057: C 430      continue
1058: C              NCNTR(1,KNUTRN) = NCNTR(1,KNUTRN) + NGENE
1059: C      else if ( JPHOT.ne.0 ) then
1060: C          do 440 I = 1, NGENE
1061: C              WCNTR(1,KPHOTN) = WCNTR(1,KPHOTN) + WWW(IWK1(I))
1062: C 440      continue
1063: C              NCNTR(1,KPHOTN) = NCNTR(1,KPHOTN) + NGENE
1064: C          end if
1065: C
1066: C =====
1068: C SOURCES FROM FISSION NEUTRON BANK FOR EIGENVALUE PROBLEM
1069: C =====
1070: C
1071: C
1072: C      else
1073: C
1074: C          if ( NHSUB.lt.NHIST1 ) then
1075: C              call FISSRC( IOW, IRAND, NFISB, NGENE, NFB0, WFACT, IFISB,
1076: C                  &      NHIST, IWK1, NHIST, NBANK, NUC, NZONE, NFBANK,
1077: C                  &      NBATCH, NGROUP, NGP(KPNEUT), NREG, NEST, NLENG,
1078: C                  &      WSUM, KZREG, KZMAT, LTP, WGT, REAL(EBOTX(KPNEUT)),
1079: C                  &      REAL(ETOPX(KPNEUT)), ENGYB, XIMP, XSOC, NCNTR,
1080: C                  &      WCNTR, NEVENT, CX, ICX, MCX, NMT, KLBI1, XLBI1,
1081: C                  &      KLBI2, XLBI2, NFFL, LSFFL, IZFFL, NDEAD, LSDED, XXX,
1082: C                  &      YYY, ZZZ, AAA, BBB, CCC, EEE, WWW, IGS, IZZ, TTT,
1083: C                  &      XIM, IBREG, IBSPC, KLSF, LEVL, LZZ, LPOS, LCRS,
1084: C                  &      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
1085: C                  &      H(LXXXF2), H(LYYYF2), H(LZZZF2), H(LEEEF2),
1086: C                  &      H(LINUF2), H(LIZZF2), H(LLEVL2), H(LLZZF2),
1087: C                  &      H(LLPOSF2), H(LLCRSF2), H(LIBRGF2), H(LIBSPF2),
1088: C                  &      H(LKLSFF2), H(LDFBK2), KDFBK2, MDFBK2, H(LIFBK2),
1089: C                  &      KIFBK2, MIFBK2, R, IWK2, IWK3, IFL, IFI, X, Y, RWK1,
1090: C                  &      EW1, IMTF, RWK2, IWK4)
1091: C
1092: C              if (JPRT.ne.0) write(IOW, '(1x,a,a)')
1093: C                  &      'XXX(SOURCE) Perturbation capability not implemented',
1094: C                  &      'for sub-batch methods.'
1095: C          else
1096: C              call FISSRC( IOW, IRAND, NFISB, NGENE, NFB0, WFACT, IFISB,
1097: C                  &      NFBK0, IWK1, NHIST, NBANK, NUC, NZONE, NFBANK,
1098: C                  &      NBATCH, NGROUP, NGP(KPNEUT), NREG, NEST, NLENG,
1099: C                  &      WSUM, KZREG, KZMAT, LTP, WGT, REAL(EBOTX(KPNEUT)),
1100: C                  &      REAL(ETOPX(KPNEUT)), ENGYB, XIMP, XSOC, NCNTR,
1101: C                  &      WCNTR, NEVENT, CX, ICX, MCX, NMT, KLBI1, XLBI1,
1102: C                  &      KLBI2, XLBI2, NFFL, LSFFL, IZFFL, NDEAD, LSDED, XXX,
1103: C                  &      YYY, ZZZ, AAA, BBB, CCC, EEE, WWW, IGG, IZZ, TTT,
1104: C                  &      XIM, IBREG, IBSPC, KLSF, LEVL, LZZ, LPOS, LCRS,
1105: C                  &      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
1106: C                  &      XXXF, YYYF, ZZZF, EEEF,
1107: C                  &      INUF, IZZF, LEVL, LZZF,
1108: C                  &      LPOSF, LCRSF, IBRGF, IBSPF,
1109: C                  &      KLSFF, DFBK, KDFBK, MDFBK, IFBK,
1110: C                  &      KIFBK, MIFBK, R, IWK2, IWK3, IFL, IFI, X, Y, RWK1,
1111: C                  &      EW1, IMTF, RWK2, IWK4)
1112: C              call FISSRCPT(NGENE, IWK1, NHIST, IFISB, NFB0,
1113: C                  &      NFBK0, IZZF, IBRGF, IMTF,
1114: C                  &      NBANK, LSDED, IBREG, IBNK, KIBNK, MIBNK,
1115: C                  &      NZONE, KZREG, NREG, WGT, NBATCH, NDEAD, IWK2, RWK1,
1116: C                  &      NPTDS, WWD, WD0, WCTR, WWD2, NPTCS, WWR, WR0,
1117: C                  &      NUCPT,
1118: C                  &      NGSP, WSD, NTPT, SWD0, WSC, SWR0, NSKIP, NORDDS,
1119: C                  &      NOPS,
1120: C                  &      RWK2, IWK4,
1121: C                  &      WWB, WB0, WSB, SWB0, NPTBE,
1122: C                  &      WWBD, WBD0, WSD, SWBD0, WWLD, WLD0, WSLD, SWLD0)
1123: C          end if
1124: C
1125: C          ... fission source is always neutron!! is it OK?
1126: C
1127: C          JMP      = 0
1128: C          KKPID      = KPNEUT
1129: C          *VOCL LOOP,NOVREC
1130: C              do 460 I = 1, NGENE
1131: C                  KKP(IWK1(I)) = KKPID
1132: C 460      continue
1133: C
1134: C          ... determine time bin ....
1135: C
1136: C          if ( JTIME.ne.0 ) then
1137: C              do 470 I = 1, NGENE
1138: C                  X(I) = TTT(IWK1(I))
1139: C 470      continue
1140: C              call BS0ISD( TIMEB, NTIME+1, X(1), IWK2(1), NGENE )
1141: C          *VOCL LOOP,NOVREC
1142: C              do 480 I = 1, NGENE
1143: C                  ITT(IWK1(I)) = MAX(1,MIN(NTIME,IWK2(I)))
1144: C 480      continue
1145: C          end if
1146: C
1147: C          ILOST      = 0
1148: C
1149: C          =====
1150: C          .... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS ....
1151: C          =====
1152: C
1153: C          .... KSOUR2 : =0      ISOTROPIC SOURCE
1154: C                  .NE.0      SOURCE TYPE
1155: C
1156: C          .... GENERATE IMAGINARY PARTICLES TO CALCULATE UNCLLIDED
1157: C                  CONTRIBUTION TO NEXT EVENT DETECTORS .....
1158: C
1159: C
1160: C          if ( JPTDT.ne.0 ) then
1161: C              LLLLL      = NFFL(NZONE+1) - NGENE + 1
1162: C              KSOUR2      = 0
1163: C              DUMAX      = +1.D0
1164: C              DUMIN      = -1.D0
1165: C              JNP      = 1
1166: C              call SUNCL( IOW, A, H, NGENE, LSFFL(LLLLL),
1167: C                  &      KSOUR2, RDUMMY, DUMAX, DUMIN, JNP,
1168: C                  &      JPUSD, JSPDT, XPDET, IPDET, IPDT2,
1169: C                  &      IMSFL, IMNFL, IMZFL,
1170: C                  &      IMDED, IMSLT, IMNLT, IMZLT ,

```

src/mvp/source.f

```

1171:      &          KDBNK(0),KIBNK(0), TIMEB, KZMAT,
1172:      &          XXX ,   YY ,   ZZZ ,   AAA ,   BBB ,   CCC ,
1173:      &          EEE ,   WWW ,   IZZ ,   IGG ,   TTT ,   ITT ,
1174:      &          LEVL ,   LZZ ,   LPOS ,   LCRS ,
1175:      &          DBNK ,   KDBNK,   MDBNK,   IBNK ,   KIBNK,   MIBNK,
1176:      &          LZZI ,   LPOSI,   LCRSI,
1177:      &          OPTI ,   PATH ,   KDETP,   KLSFI,   SMACI,   MMACI,
1178:      &          X, Y, Z, EW1, IFL, R,
1179:      &          H(LXYZ(1)),H(LXYZ(2)),H(LXYZ(3)),H(LXYZ(4)),
1180:      &          H(LXYZ(5)),H(LXYZ(6)),H(LXYZ(7)),H(LXYZ(8)),
1181:      &          H(LXYZ(9)),H(LXYZ(10)),
1182:      &          IWK2, IWK3, R(2*NBANK+1), R(3*NBANK+1),
1183:      &          R(4*NBANK+1), SMCN )
1184:      end if
1185: C
1186:      end if
1187: C
1188:      ... treatment for optional bank parameters ....
1189: C
1190:      ... weight on birth
1191: C
1192:      if ( KDBNK(1).ne.0 ) then
1193: *VOCL LOOP,NOVREC
1194:      do 490 I = 1, NGENE
1195:          DBNK(IWK1(I),KDBNK(1)) = WWW(IWK1(I))
1196:      490      continue
1197:      end if
1198: C
1199:      ... time on birth
1200: C
1201:      if ( KDBNK(2).ne.0 ) then
1202: *VOCL LOOP,NOVREC
1203:      do 500 I = 1, NGENE
1204:          DBNK(IWK1(I),KDBNK(2)) = TTT(IWK1(I))
1205:      500      continue
1206:      end if
1207: C
1208:      ... energy on birth
1209: C
1210:      if ( KDBNK(3).ne.0 ) then
1211: *VOCL LOOP,NOVREC
1212:      do 510 I = 1, NGENE
1213:          DBNK(IWK1(I),KDBNK(3)) = EEE(IWK1(I))
1214:      510      continue
1215:      end if
1216: C
1217: C      ... region # and/or zone # on birth
1218: C      but the region # and zone # may not be the real one
1219: C      in lattice geometry when TALLY=LATTICE or STG region is used.
1220: C
1221: C      Actual region/zone # on birth should be determined in zone
1222: C      search routine checking value of IBNK(*,KIBNK(5)) < 0.
1223: C
1224:      if ( KIBNK(2).ne.0 ) then
1225: *VOCL LOOP,NOVREC
1226:      do 520 I = 1, NGENE
1227:          if ( JTLT.ne.0 ) then
1228:              IBNK(IWK1(I),KIBNK(2)) = IBREG(IWK1(I))
1229:          else
1230:              IBNK(IWK1(I),KIBNK(2)) = KZREG(IZZ(IWK1(I)))
1231:          end if
1232:      520      continue
1233:      end if
1234: C
1235:      if ( KIBNK(3).ne.0 ) then

```

```

1236: *VOCL LOOP,NOVREC
1237:      do 530 I = 1, NGENE
1238:          IBNK(IWK1(I),KIBNK(3)) = IZZ(IWK1(I))
1239:      530      continue
1240:      end if
1241: C
1242: C      ... generation of particle (1 for primary particles)
1243: C
1244:      if ( KIBNK(4).ne.0 ) then
1245: *VOCL LOOP,NOVREC
1246:      do 540 I = 1, NGENE
1247:          IBNK(IWK1(I),KIBNK(4)) = 1
1248:      540      continue
1249:      end if
1250: C
1251: C      ... Marker region flag
1252: C
1253:      if ( KIBNK(9).ne.0 ) then
1254:          do 560 J = 1, NMKREG
1255:              KI9 = KIBNK(9) + J - 1
1256: *VOCL LOOP,NOVREC
1257:          do 550 I = 1, NGENE
1258:              IBNK(IWK1(I),KI9) = 0
1259:          550      continue
1260:          560      continue
1261:          end if
1262: C##<2007/03/14:PN3:
1263: C
1264: C      ... initialization of produce region
1265: C
1266:      if ( KIBNK(17).ne.0 ) then
1267: *VOCL LOOP,NOVREC
1268:          do I = 1, NGENE
1269:              IBNK(IWK1(I),KIBNK(17)) = 0
1270:          end do
1271:          end if
1272: C
1273: C      ... initialization of produce nuclide
1274: C
1275:      if ( KIBNK(18).ne.0 ) then
1276: *VOCL LOOP,NOVREC
1277:          do I = 1, NGENE
1278:              IBNK(IWK1(I),KIBNK(18)) = 0
1279:          end do
1280:          end if
1281: C
1282: C      ... initialization of produce reaction
1283: C
1284:      if ( KIBNK(19).ne.0 ) then
1285: *VOCL LOOP,NOVREC
1286:          do I = 1, NGENE
1287:              IBNK(IWK1(I),KIBNK(19)) = 0
1288:          end do
1289:          end if
1290: C##>
1291: C##<2007/03/14:PN4:
1292:      if ( JPHNU.ne.0 ) then
1293:          do I = 1, NGENE
1294:              KPNFG(IWK1(I)) = 0
1295:          end do
1296:          end if
1297: C##>
1298: C
1299: C      .... PARTICLES IN THE REST OF BANK ARE DEAD PARTICLES ! .....
1300: C

```

src/mvp/source.f

```

1301:      NDEAD = NDEAD - NGENE + ILOST
1302:      NLOST = NLOST + ILOST
1303: C
1304: C .... CALCULATE & PRINT AVERAGED PARAMETERS .....
1305: C
1306: C      XAV = 0.0
1307: C      YAV = 0.0
1308: C      ZAV = 0.0
1309: C      AAV = 0.0
1310: C      BAV = 0.0
1311: C      CAV = 0.0
1312: C      EAV = 0.0
1313: C      DWSUM = 0.0D0
1314: C
1315: C      do 570 I = 1, NGENE
1316: C          XAV = XAV + XXX(IWK1(I))*WWW(IWK1(I))
1317: C          YAV = YAV + YYY(IWK1(I))*WWW(IWK1(I))
1318: C          ZAV = ZAV + ZZZ(IWK1(I))*WWW(IWK1(I))
1319: C          AAV = AAV + AAA(IWK1(I))*WWW(IWK1(I))
1320: C          BAV = BAV + BBB(IWK1(I))*WWW(IWK1(I))
1321: C          CAV = CAV + CCC(IWK1(I))*WWW(IWK1(I))
1322: C          EAV = EAV + EEE(IWK1(I))*WWW(IWK1(I))
1323: C          WSUMB = WSUMB + WWW(IWK1(I))
1324: C      570 continue
1325: C
1326: C      NTGEN = NTGEN + NGENE
1327: C
1328: C      ... generated all history in a batch ...
1329: C
1330: C      if ( NGBAT.eq.0 ) then
1331: C
1332: C          RSRC = DBLE(NGENE0) / WSUMB
1333: C          XAV = XAV * RSRC
1334: C          YAV = YAV * RSRC
1335: C          ZAV = ZAV * RSRC
1336: C          AAV = AAV * RSRC
1337: C          BAV = BAV * RSRC
1338: C          CAV = CAV * RSRC
1339: C          EAV = EAV * RSRC
1340: C
1341: C          XAVT(1) = XAVT(1) + XAV
1342: C          XAVT(2) = XAVT(2) + YAV
1343: C          XAVT(3) = XAVT(3) + ZAV
1344: C          AAVT(1) = AAVT(1) + AAV
1345: C          AAVT(2) = AAVT(2) + BAV
1346: C          AAVT(3) = AAVT(3) + CAV
1347: C          EAVT = EAVT + EAV
1348: C
1349: C          XAV = XAV/NGENE0
1350: C          YAV = YAV/NGENE0
1351: C          ZAV = ZAV/NGENE0
1352: C
1353: C          if ( JPRTS(6).ne.1 ) then
1354: C              C/#IF PARA(SX* CRAY)
1355: C              * call MVPSYNC_LOCK(2)
1356: C              C/#ENDIF
1357: C              if ( JBPRNT.ne.0 ) then
1358: C
1359: C              C/#IF PARA
1360: C                  write(IOW,'(1X,' TASK ',I5)') IDTASK
1361: C              C/#ENDIF
1362: C              write(IOW,7080) '(BATCH) ', XAV, YAV, ZAV, AAV/
1363: C              & NGENE0, BAV/NGENE0, CAV/NGENE0, EAV/NGENE0
1364: C              write(IOW,7080) '(CUMULATIVE)', XAVT(1) /NTGEN, XAVT(2) /
1365: C              & NTGEN, XAVT(3) /NTGEN, AAVT(1) /NTGEN, AAVT(2) /
1366: C              & NTGEN, AAVT(3) /NTGEN, EAVT/NTGEN
1367: C              write(IOW,*) ' GENERATED PARTICLES IN THIS RUN = ', NTGEN
1368: C              end if
1369: C              C/#IF PARA(SX* CRAY)
1370: C              * call MVPSYNC_UNLOCK(2)
1371: C              C/#ENDIF
1372: C              end if
1373: C              end if
1374: C
1375: C      7080 format(1X,' AVERAGE ',A12,2X,' X=',1PE11.4,' Y=',E11.4,' Z=',
1376: C              & E11.4,' MU=',E11.4,' ETA=',E11.4,' XI=',E11.4,' E =',E11.4
1377: C              & )
1378: C
1379: C      ... debug dump
1380: C
1381: C      if ( MOD(JDEBG(3),2).eq.1 ) then
1382: C          write(IOW,7100) (I,XXX(IWK1(I)),YYY(IWK1(I)),ZZZ(IWK1(I)),
1383: C              & AAA(IWK1(I)),BBB(IWK1(I)),CCC(IWK1(I)),WWW(IWK1(I)),
1384: C              & IZZ(IWK1(I)),EEE(IWK1(I)),IGG(IWK1(I)),TTT(IWK1(I)),I=
1385: C              & 1,NGENE)
1386: C      7100 format(/1X,' GENERATED SOURCE PARAMETERS '/1X,
1387: C              & ' XXX YYY ZZZ',
1388: C              & ' AAA BBB',
1389: C              & ' CCC WWW',
1390: C              & ' IZZ EEE IGG TTT'/
1391: C              & (1X,I5,1P,6E12.4,E12.4,I5,E12.4,I3,E12.4))
1392: C      end if
1393: C
1394: C      if ( JEIGN.ne.0 ) then
1395: C
1396: C          XVA = 0.0
1397: C          YVA = 0.0
1398: C          ZVA = 0.0
1399: C          do 580 I = 1, NGENE
1400: C              XVA = XVA + (XXX(IWK1(I))-XAV)*(XXX(IWK1(I))-XAV)*
1401: C              & WWW(IWK1(I))
1402: C              YVA = YVA + (YYY(IWK1(I))-YAV)*(YYY(IWK1(I))-YAV)*
1403: C              & WWW(IWK1(I))
1404: C              ZVA = ZVA + (ZZZ(IWK1(I))-ZAV)*(ZZZ(IWK1(I))-ZAV)*
1405: C              & WWW(IWK1(I))
1406: C              XVA = XVA + XXX(IWK1(I))*XXX(IWK1(I))*WWW(IWK1(I))
1407: C              YVA = YVA + YYY(IWK1(I))*YYY(IWK1(I))*WWW(IWK1(I))
1408: C              ZVA = ZVA + ZZZ(IWK1(I))*ZZZ(IWK1(I))*WWW(IWK1(I))
1409: C      580 continue
1410: C          XSXV(NBATCH+1,1) = XSXV(NBATCH+1,1) + XVA
1411: C          XSXV(NBATCH+1,2) = XSXV(NBATCH+1,2) + YVA
1412: C          XSXV(NBATCH+1,3) = XSXV(NBATCH+1,3) + ZVA
1413: C
1414: C          if ( NGBAT.eq.0 ) then
1415: C
1416: C              XSXV(NBATCH+1,1) = XSXV(NBATCH+1,1)/(NGENE0-1)
1417: C              XSXV(NBATCH+1,2) = XSXV(NBATCH+1,2)/(NGENE0-1)
1418: C              XSXV(NBATCH+1,3) = XSXV(NBATCH+1,3)/(NGENE0-1)
1419: C              XSXV(NBATCH+1,1) = XSXV(NBATCH+1,1) * RSRC
1420: C              XSXV(NBATCH+1,2) = XSXV(NBATCH+1,2) * RSRC
1421: C              XSXV(NBATCH+1,3) = XSXV(NBATCH+1,3) * RSRC
1422: C
1423: C              XSXV(NBATCH+1,1) = (XSXV(NBATCH+1,1)-NGENE0*XAV**2) /
1424: C              & (NGENE0-1)
1425: C              XSXV(NBATCH+1,2) = (XSXV(NBATCH+1,2)-NGENE0*YAV**2) /
1426: C              & (NGENE0-1)
1427: C              XSXV(NBATCH+1,3) = (XSXV(NBATCH+1,3)-NGENE0*ZAV**2) /
1428: C              & (NGENE0-1)
1429: C              C/#IF PARA(SX* CRAY)
1430: C              * call MVPSYNC_LOCK(2)

```

src/mvp/source.f

```

1431: C/#ENDIF
1432:       if ( JBRPNT.ne.0 ) then
1433: C/#IF PARA
1434:       if ( JPRTS(6).ne.1 ) then
1435: c##<2007/03/14:PN3:
1436: c##      write(IOW,7120) IDTASK, '(BATCH) ',
1437:      write(IOW,7120) IDTASK, '(BATCH) ',
1438: c##>
1439:       &      XSXV(NBATCH+1,1), XSXV(NBATCH+1,2),
1440:       &      XSXV(NBATCH+1,3)
1441:       end if
1442: c##<2007/03/14:PN3:
1443: c7120      format(1X,' TASK ',I5,' VARIANCE ',A10,2X,' X= ',
1444: c## &      1PE11.4,' Y= ',E11.4,' Z= ',E11.4)
1445: c7120      format(1X,' TASK ',I5,' VARIANCE ',A10,3X,' X=',
1446: &      1P,E11.4,' Y= ',E11.4,' Z=',E11.4)
1447: c##>
1448: C/#ELSE
1449:       if ( JPRTS(6).ne.1 ) then
1450: c##<2007/03/14:PN3:
1451: c##      write(IOW,7140) '(BATCH) ', XSXV(NBATCH+1,1),
1452:      write(IOW,7140) '(BATCH) ', XSXV(NBATCH+1,1),
1453: c##>
1454:       &      XSXV(NBATCH+1,2), XSXV(NBATCH+1,3)
1455:       end if
1456: c##<2007/03/14:PN3:
1457: c7140      format(1X,' VARIANCE ',A10,2X,' X= ',1PE11.4,' Y= ',
1458: c## &      E11.4,' Z= ',E11.4)
1459: c7140      format(1X,' VARIANCE ',A10,3X,' X=',1P,E11.4,' Y= ',
1460: &      E11.4,' Z=',E11.4)
1461: c##>
1462: C/#ENDIF
1463:       end if
1464: C/#IF PARA(SX* CRAY)
1465: *      call MVPSYNC_UNLOCK(2)
1466: C/#ENDIF
1467:       end if
1468:       end if
1469: C
1470: C      IF(JPRTS(6).NE.1)
1471: C      & WRITE(IOW,*) ' GENERATED PARTICLES IN THIS RUN = ' NTGEN
1472: C
1473: C-----
1474: C      MAKE MICRO CROSS SECTIONS FOR EACH PARTICLE
1475: C-----
1476: C
1477: C      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
1478: C      INEUT = 0
1479: C      do 570 I = 1, NGENE
1480: C      IGII = IGG(IWK1(I))
1481: C      if ( IGII.le.NGP1 ) then
1482: C      INEUT = INEUT + 1
1483: C      IWK2(INEUT) = IWK1(I)
1484: C      end if
1485: C 570      continue
1486: C      IPHOT = INEUT
1487: C      do 580 I = 1, NGENE
1488: C      IGII = IGG(IWK1(I))
1489: C      if ( IGII.gt.NGP1 ) then
1490: C      IPHOT = IPHOT + 1
1491: C      IWK2(IPHOT) = IWK1(I)
1492: C      end if
1493: C 580      continue
1494: C      do 590 I = 1, NGENE
1495: C      IWK1(I) = IWK2(I)

```

```

1496: C 590      continue
1497: C
1498: C      else
1499: C      if ( JNEUT.ne.0 ) INEUT = NGENE
1500: C      if ( JPHOT.ne.0 ) INEUT = 0
1501: C      end if
1502: C
1503: C      ... sort IWK1(*) by particle species
1504: C
1505: C      if ( JMP.eq.0 ) then
1506: C      KS = 1
1507: C      KSORT(1) = KKPID
1508: C      IKSORT(1) = NGENE
1509: C      else
1510: C      KS = 0
1511: C      KK = 0
1512: C      do 600 IK = 1, KPLIM
1513: C      KK2 = KK
1514: C      do 590 I = 1, NGENE
1515: C      if ( KKP(IWK1(I)).eq.IK ) then
1516: C      KK2 = KK2 + 1
1517: C      IWK2(KK2) = IWK1(I)
1518: C      end if
1519: C 590      continue
1520: C      if ( KK2.gt.KK ) then
1521: C      KS = KS + 1
1522: C      KSORT(KS) = IK
1523: C      IKSORT(KS) = KK2 - KK
1524: C      end if
1525: C      if ( KK2.ge.NGENE ) go to 610
1526: C      KK = KK2
1527: C 600      continue
1528: C 610      continue
1529: C      do 620 I = 1, NGENE
1530: C      IWK1(I) = IWK2(I)
1531: C 620      continue
1532: C      end if
1533: C
1534: C      ..... NEUTRON MICRO CROSS SECTION .....
1535: C
1536: C
1537: C      ... calculate micro for all nuclides here ...
1538: C
1539: C      if ( JAMXC.eq.0 ) then
1540: C
1541: C      KK = 1
1542: C      do 630 K = 1, KS
1543: C      IKPID = KSORT(K)
1544: C
1545: C      ... neutron
1546: C
1547: C      if ( IKPID.eq.KPNEUT ) then
1548: C      if ( JVMNT.ne.0 ) call VMNT00( 8, 1 )
1549: C      if ( JVMNT.ne.0 ) call VMNTR1( 8, IKSORT(K) )
1550: C      if ( JPRTS.ne.0 .and. JPTMP.ne.0 ) then
1551: C      call GMICPT( INEUT, IWK1, IRAND, JEIGN, NUC, NMT, NBANK,
1552: C      &      NSMIC,
1553: C      &      EEE, SMIC, LMIC, KSPI, CX, ICX, MCX, KLIB1, KLIB2,
1554: C      &      XLIB1, SGTAL, CRES, KCRES, NSTAL, MCRES, EWL, RWK1,
1555: C      &      IWK2, IWK3, IFI, IFL, R(1), R(NBANK+1), SMCW,
1556: C      &      SMICP, CXPl, SMCWP, E2P, DEP, RP )
1557: C      else
1558: C      call GETMIC( IKSORT(K), IWK1(KK), IRAND, NUC, NMT,
1559: C      &      NBANK, NSMIC, EEE, SMIC, LMIC, KSPI, CX, ICX,
1560: C      &      MCX, KLIB1, KLIB2, XLIB1, EWL, RWK1, IWK2, IWK3,

```

src/mvp/source.f

```

1561: c##<2007/03/14:PN3:
1562: c## &      IFI, IFL, R(1), R(NBANK+1), SMCW, RNU )
1563: c## &      IFI, IFL, R(1), R(NBANK+1), SMCW, RNU, NUC_MAX )
1564: c##>
1565:       end if
1566:
1567:       if ( JVMNT.ne.0 ) call VMNT22( 8, 1 )
1568: C
1569: C      ..... PHOTON MICRO CROSS SECTION .....
1570: C
1571:       else if ( IKPID.eq.KPPHOT ) then
1572:
1573:       if ( JVMNT.ne.0 ) call VMNT00( 8, 1 )
1574:       if ( JVMNT.ne.0 ) call VMNTR1( 8, IKSORT(K) )
1575:
1576:       call GTMICP( JGAMM, IKSORT(K), IWK1(KK), NUC, NMTP,
1577: &      NPATOM, NBANK, NSMIC, EEE, SMIC, KSPI, CXP, ICXP,
1578: &      MCXP, KLBP1, KLBP2, XLBP2, EW1, IWK2, R, IWK3 )
1579:
1580: C &      SGIAL, CRES, KCRES, NSTAL, MCRES,
1581: c##<2007/03/14:PN3:
1582: C
1583: C      .... photo-nuclear micro cross section
1584:       if ( JPHNU.ne.0 ) then
1585:       call GTMICPN( IKSORT(K), IWK1(KK), NNC, NUCPN, NUCPNI,
1586: &      NMTPN, NBANK, NSMICPN, NUC_MAX, EEE, SMICPN,
1587: &      CXP, ICXP, MCXPN, KLBP1, KLBP2, XLBP1,
1588: &      EBOTLPN, EW1, RWK1, IWK2, IWK3, R, SMCW, RNU )
1589:       end if
1590: c##>
1591:
1592:       if ( JVMNT.ne.0 ) call VMNT22( 8, 1 )
1593:       else
1594: c##<2007/03/14:PN3:
1595: c##      write(IOW,*)
1596:       write(IOW,'(1X,A,A,I6)')
1597: c##>
1598: &      'XXX(SOURCE) Program error: Particle other than',
1599: &      ' neutron/photon is not supported!!! IKPID=', IKPID
1600:       stop 666
1601:       end if
1602: C
1603:       KK      = KK + IKSORT(K)
1604: 630      continue
1605: C
1606: C      ... ADAPTIVE-MICRO-CALCULATION mode
1607: C      calculate micro for micro reaction rate tally here
1608: C
1609:       else
1610: C
1611: C      ... KSPI is used to judge that micro for the particle has
1612: C      been calculated or not
1613: C
1614:       do 650 N = 1, NUC
1615: *VOCL LOOP,NOVREC
1616:       do 640 I = 1, NGENE
1617:       KSPI(IWK1(I),N) = 0
1618: 640       continue
1619: 650       continue
1620: C
1621:       if ( NNCST.gt.0 ) then
1622: C
1623:       KK      = 1
1624:       do 700 K = 1, KS
1625:       IKPID   = KSORT(K)

```

```

1626:       KK2      = KK + IKSORT(K) - 1
1627: C
1628: C      ... neutron
1629: C
1630: CCC      if ( JNEUT.ne.0.and.INEUT.gt.0 ) then
1631:       if ( IKPID.eq.KPNEUT ) then
1632:       if ( JVMNT.ne.0 ) call VMNT00( 8, 1 )
1633:       if ( JVMNT.ne.0 ) call VMNTR1( 8, IKSORT(K) )
1634:       do 660 I = KK, KK2
1635:       EW1(I) = EEE(IWK1(I))
1636: 660       continue
1637:       do 670 NC = 1, NNCST
1638:       IN      = INCST(NC,1)
1639:       if ( IN.ne.0 ) then
1640:       call GMICN1( IN, IKSORT(K), IWK1(KK), EW1(KK),
1641: &      IRAND, NUC, NMTP, NBANK, NSMIC,
1642: &      SMIC, LMIC, KSPI, CX, ICX, MCX, KLIB1,
1643: &      KLIB2, XLIB1, RWK1, IWK2, IWK3, IFI,
1644: &      IFL, R, SMCW, RNU, NUC_MAX)
1645:       end if
1646: 670       continue
1647:       if ( JVMNT.ne.0 ) call VMNT22( 8, 1 )
1648: C
1649: c##<2007/03/14:PN3:
1650: C      ... photon
1651: c##>
1652: CC      if ( JPHOT.ne.0.and.NGENE-INEUT.gt.0 ) then
1653:       else if ( IKPID.eq.KPPHOT ) then
1654:       if ( JVMNT.ne.0 ) call VMNT00( 8, 1 )
1655:       if ( JVMNT.ne.0 ) call VMNTR1( 8, IKSORT(K) )
1656:
1657:       do 680 I = KK, KK2
1658:       EW1(I) = LOG(EEE(IWK1(I)))
1659: c##<2007/03/14:PN3:
1660:       R(I) = EEE(IWK1(I))
1661: c##>
1662: 680       continue
1663:
1664:       do 690 NC = 1, NNCST
1665:       IN      = INCST(NC,2)
1666:       if ( IN.ne.0 ) then
1667:       call GMICP1( IN, IKSORT(K), IWK1(KK), EW1(KK),
1668: &      JGAMM, NUC, NMTP, NPATOM, NBANK, NSMIC,
1669: &      SMIC, KSPI, CXP, ICXP, MCXP, KLBP1,
1670: c##<2007/03/14:PN3:
1671: c## &      KLBP2, XLBP2, IWK2, R, IWK3 )
1672: c## &      KLBP2, XLBP2, IWK2, R(NBANK+1), IWK3 )
1673: c##>
1674:       end if
1675: 690       continue
1676: c##<2007/03/14:PN3:
1677:       if ( JPHNU.ne.0 ) then
1678:       do NC = 1, NUCPN
1679:       call GMICPN1( NC, IKSORT(K), IWK1(KK), R, NUC,
1680: &      NUCPN, NUCPNI, NMTPN, NBANK, NUC_MAX,
1681: &      NSMICPN, SMICPN, CXP, ICXP, MCXPN,
1682: &      KLBP1, KLBP2, XLBP1, EBOTLPN, RWK1,
1683: &      IWK2, IWK3, R(NBANK+1),
1684: &      SMCW, RNU )
1685:       end do
1686:       end if
1687: c##>
1688:
1689:       if ( JVMNT.ne.0 ) call VMNT22( 8, 1 )
1690:       else

```

src/mvp/source.f

```
1691: c##<2007/03/14:PN3:
1692: c##          write(IOW,*)
1693:          write(IOW,'(1X,A,A,I6)')
1694: c##>
1695:      &          'XXX(SOURCE) Program error: Particle other than',
1696:      &          ' neutron/photon is not supported!!! IKPID=', IKPID
1697:      stop 666
1698:      end if
1699: C
1700:      KK          = KK2 + 1
1701: 700      continue
1702:      end if
1703:      end if
1704: C
1705: C
1706:      if ( NSTAL.gt.0 ) then
1707:      KK          = 1
1708:      do 710 K = 1, KS
1709:      call GTSGTL( KSORT(K), IKSORT(K), IWK1(KK), NBANK, EEE,
1710:      &          SGTAL, CRES, KCRES, NSTAL, MCRES, EW1, IWK2 )
1711:      KK          = KK + IKSORT(K)
1712: 710      continue
1713:      end if
1714: C
1715: C      .... NO MACRO CROSS SECTION PREPARED HERE ....
1716: C
1717: *VOCL LOOP,NOVREC
1718: do 720 I = 1, NGENE
1719:      MMAC(IWK1(I),2) = 0
1720: 720 continue
1721: C
1722:      return
1723:
1724: C/#ENDIF
1725:
1726:      end
```


src/mvp/splitb.f

```

1:      subroutine SPLITB( JJCOL, IPAR0, ISF0, NSF, IRAND,
2:      & NPKIND,NBANK,NZONE,NGROUP,NREG,NTIME,NEST,
3:      S LSFFL, IZFFL, NFFL, LSDED, NDEAD, NSMAC, NSMIC,
4:      V XIMP, WKIL, WSRV, WTIME, WLLIM, KZREG, NCNTR, WCNTR, NEVENT,
5:      B XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,KKP,IZZ,IGG,TTT,ITT,LEVL,LZZ,
6:      B LPOS,LCRS,XIM,KLSF,EEE, IBREG,IBSPC, MB, NUC, NSTAL,
7:      B SMAC, KMAC, MMAC,
8:      B SMIC, KSPI,SGTAL,
9:      B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
10:     T NKILD, WKILD, NSURV, WSURV, NSPLT, WSPLT,
11:     W R, W1, W2, P1, ISP, ISP2, IPT,
12:     & SMICP, SMACP, NPTDS, NPTCS, NGSP, WWD, WWD2, WWR, WSD, WSC,
13:     & NORDD, NOPS,
14:     c##<2007/03/14:PN3:
15:     & NUCPN, NSMICPN, SMICPN,
16:     c##>
17:     c+beff2
18:     & WWB, WSB, NPTBE,
19:     & WWBD, WSD, WWLD, WSLD)
20:     c-beff2
21:     C=<MVP>=====
22:     C PURPOSE: SPLITTING & RUSSIAN ROULETTE
23:     C CALLED IN: SEADONE, SEARCH, NEUTR, PHOTR
24:     C
25:     C=====
26:     C DO NOT CALL IF JRRIT = JIMPT = IWWND = 0 !!
27:     C
28:     C JJCOL : 1/0 = CALLED AFTER COLLISION / NO
29:     C
30:     C ISF0 : START POINT OF PARTICLES IN FREE-FLIGHT STACK TO BE SPLITTED
31:     C OR KILLED.
32:     C NSF : NUMBER OF PARTICLES IN FREE-FLIGHT STACK TO BE SPLITTED OR
33:     C COPIED.
34:     C X IGOLD: OLD GROUP NUMBER NECESSARY TO SPLIT BY IMPORTANCE.
35:     C X IROLD: OLD REGION NUMBER NECESSARY TO SPLIT BY IMPORTANCE.
36:     C -----
37:     C CONTENTS OF FREE-FLIGHT STACK & ISF0 & NSF WILL BE CHANGED.
38:     C -----
39:     C
40:     include 'INC/_KPIDS'
41:     include 'INC/_NGPS'
42:     C
43:     include 'INC/_FLAGS'
44:     C
45:     C ... VARIANCE REDUCTION PARAMETERS ....
46:     C
47:     real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
48:     real WTIME(NTIME)
49:     real WLLIM
50:     C
51:     C ... FREE-FLIGHT STACK .....
52:     C
53:     integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
54:     C
55:     C ... DEAD PARTICLE STACK.....
56:     C
57:     integer LSDED(NBANK), NDEAD
58:     C
59:     C ... COUNTER .....
60:     C
61:     real*8 WKILD(NGROUP,NREG), WSURV(NGROUP,NREG), WSPLT(NGROUP,NREG)
62:     real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG), NSPLT(NGROUP,NREG)
63:     CCCC real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
64:     real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
65:     C
66:     C ... BANK .....
67:     C
68:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
69:     real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
70:     real WWW(NBANK), XIM(NBANK)
71:     real*8 TTT(NBANK)
72:     integer ITT(NBANK)
73:     integer KKP(NBANK)
74:     integer IGG(NBANK), IZZ(NBANK), LEVL(NBANK), LPOS(NBANK,NEST),
75:     & LZZ(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
76:     integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
77:     C
78:     C ... SIGMA BANK .....
79:     C
80:     real EEE(NBANK), SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
81:     & SGTAL(NBANK,NSTAL)
82:     c##<2007/03/14:PN3:
83:     real SMICPN(NBANK,NUCPN,NSMICPN)
84:     c##>
85:     integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC)
86:     C
87:     C ... variables for perturbation calculation ...
88:     C
89:     real*8 WWD(NBANK,NPTDS,NORDD), WWD2(NBANK,NPTDS)
90:     real*8 WSD(NBANK,0:NGSP,NPTDS,NOPS)
91:     real*8 WWR(NBANK,NPTCS)
92:     real*8 WSC(NBANK,0:NGSP,NPTCS)
93:     real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
94:     c+beff2
95:     real*8 WWB(NBANK)
96:     real*8 WSB(NBANK,0:NGSP)
97:     real*8 WWBD(NBANK)
98:     real*8 WSD(NBANK,NGSP)
99:     real*8 WWLD(NBANK)
100:    real*8 WSLD(NBANK,NGSP)
101:    c-beff2
102:    C
103:    C ... optional bank parameters
104:    C
105:    real*8 DBNK(NBANK,*)
106:    integer KDBNK(0:MDBNK)
107:    integer IBNK(NBANK,*)
108:    integer KIBNK(0:MIBNK)
109:    C
110:    C ... REGION NUMBERS .....
111:    C
112:    integer KZREG(NZONE)
113:    C
114:    C ... WORKING ARRAYS .....
115:    C
116:    integer ISP(NBANK), ISP2(NBANK), IPT(NBANK)
117:    real R(NBANK), W1(NBANK), W2(NBANK), P1(NBANK)
118:    C
119:    C-----
120:    C
121:    WMIN = WLLIM
122:    ISF1 = ISF0 + NSF - 1
123:    C
124:    C ... IF IPAR > 0 : TALLY NCNTR(I,IPAR) & WCNTR(I,IPAR)
125:    C
126:    IPAR = 0
127:    C
128:    if ( JNEU.eq.1.and.JPHT.eq.0 ) IPAR = 1
129:    C
130:    if ( JNEU.eq.0.and.JPHT.eq.1 ) IPAR = 2
131:    C
132:    IPAR = 0

```

src/mvp/splitb.f

```

131:      if ( IPAR0.ne.0 ) then
132:        IPAR = IPAR0
133:      else
134:        if ( NPKIND.eq.1 ) then
135:          do 100 IK = 1, KPLIM
136:            if ( JKPAR(IK).ne.0 ) then
137:              IPAR = IK
138:              go to 110
139:            end if
140:          100 continue
141:        110 continue
142:      end if
143:    end if
144: C
145: C
146: C    === CASE 1 : Russian roulette only ===
147: C
148: C
149: C    if ( JRLT.ne.0.and.JIMPT.eq.0 ) then
150: C
151: C      call RANU2( IRAND, R(ISF0), NSF, ICON )
152: C      II = ISF0 - 1
153: C
154: C    *VOCL LOOP,NOVREC
155: C      do 120 I = ISF0, ISF1
156: C        IGT = IGG(LSFFL(I))
157: C        NRG = KZREG(IZFFL(I))
158: C        if ( JTLT.ne.0 ) NRG = IBREG(LSFFL(I))
159: C        WK = WKIL(IGT,NRG)
160: C        WS = WSRV(IGT,NRG)
161: C        if ( JWTIM.ne.0 ) then
162: C          ITTII = ITT(LSFFL(I))
163: C          WK = WK*WTIME(ITTII)
164: C          WS = WS*WTIME(ITTII)
165: C        end if
166: C        if ( JRWVR.ne.0 ) then
167: C          WK = WK*DBNK(LSFFL(I),KDBNK(1))
168: C          WS = WS*DBNK(LSFFL(I),KDBNK(1))
169: C        end if
170: C        WS = MAX(WMIN,WS)
171: C        WK = MAX(WMIN,WK)
172: C        W = WWW(LSFFL(I))
173: C        W1(I) = WS
174: C        W2(I) = W
175: C        ISP(I) = 2
176: C        if ( W.lt.WK ) ISP(I) = W/WS + R(I)
177: C
178: C      C ..... SURVIVED .....
179: C
180: C        if ( ISP(I).eq.1 ) then
181: C          WWW(LSFFL(I)) = WS
182: C          CCCCCC NCNTR(12,IPAR) = NCNTR(12,IPAR) + 1
183: C          CCCCCC WCNTR(12,IPAR) = WCNTR(12,IPAR) + WS
184: C        end if
185: C
186: C      C ..... KILLED .....
187: C
188: C        if ( ISP(I).eq.0 ) then
189: C          NDEAD = NDEAD + 1
190: C          LSDED(NDEAD) = LSFFL(I)
191: C          CCCCCC NCNTR(11,IPAR) = NCNTR(11,IPAR) + 1
192: C          CCCCCC WCNTR(11,IPAR) = WCNTR(11,IPAR) + W
193: C        end if
194: C      120 continue
195: C

```

```

196: C      .... take monitor ....
197: C
198: C      if ( IPAR.ne.0 ) then
199: C    *VOCL LOOP,NOVREC
200: C      do 130 I = ISF0, ISF1
201: C        if ( ISP(I).eq.1 ) then
202: C          NCNTR(12,IPAR) = NCNTR(12,IPAR) + 1.0D0
203: C          WCNTR(12,IPAR) = WCNTR(12,IPAR) + W1(I)
204: C        else if ( ISP(I).eq.0 ) then
205: C          NCNTR(11,IPAR) = NCNTR(11,IPAR) + 1.0D0
206: C          WCNTR(11,IPAR) = WCNTR(11,IPAR) + W2(I)
207: C        end if
208: C      130 continue
209: C    else
210: C    *VOCL LOOP,SCALAR
211: C      do 140 I = ISF0, ISF1
212: C        KP = KKP(LSFFL(I))
213: C        if ( ISP(I).eq.1 ) then
214: C          NCNTR(12,KP) = NCNTR(12,KP) + 1.0D0
215: C          WCNTR(12,KP) = WCNTR(12,KP) + W1(I)
216: C        else if ( ISP(I).eq.0 ) then
217: C          NCNTR(11,KP) = NCNTR(11,KP) + 1.0D0
218: C          WCNTR(11,KP) = WCNTR(11,KP) + W2(I)
219: C        end if
220: C      140 continue
221: C    end if
222: C
223: C    if ( JMNTR.ne.0 ) then
224: C    *VOCL LOOP,SCALAR
225: C      do 150 I = ISF0, ISF1
226: C        IGT = IGG(LSFFL(I))
227: C        NRG = KZREG(IZFFL(I))
228: C        if ( JTLT.ne.0 ) NRG = IBREG(LSFFL(I))
229: C
230: C        W = WWW(LSFFL(I))
231: C        if ( ISP(I).eq.0 ) then
232: C          NKILD(IGT,NRG) = NKILD(IGT,NRG) + 1.0D0
233: C          WKILD(IGT,NRG) = WKILD(IGT,NRG) + W
234: C        else
235: C          NSURV(IGT,NRG) = NSURV(IGT,NRG) + 1.0D0
236: C          WSURV(IGT,NRG) = WSURV(IGT,NRG) + W
237: C        end if
238: C      150 continue
239: C    end if
240: C
241: C    .... CORRECT NFFL .... (UNVECTORIZABLE) ....
242: C
243: C    *VOCL LOOP,SCALAR
244: C      do 160 I = ISF0, ISF1
245: C        if ( ISP(I).eq.0 ) NFFL(IZFFL(I)) = NFFL(IZFFL(I)) - 1
246: C      160 continue
247: C
248: C    .... COMPRESS FLIGHT STACK ....
249: C
250: C    *VOCL LOOP,NOVREC
251: C      do 170 I = ISF0, ISF1
252: C        if ( ISP(I).gt.0 ) then
253: C          II = II + 1
254: C          LSFFL(II) = LSFFL(I)
255: C          IZFFL(II) = IZFFL(I)
256: C        end if
257: C      170 continue
258: C      NFFL(NZONE+1) = II
259: C
260: C      go to 720

```

src/mvp/splitb.f

```

261:      end if
262: C
263: C
264: C ==== WEIGHT WINDOW OR SPLITTING =====
265: C
266: C
267: C ISP(I) : A PARTICLE SPLITS INTO ISP(I) PARTICLES (ISP(I) > 1)
268: C         OR DIES (ISP(I)=0) OR NEITHER SPLITS NOR DIES (ISP(I)=1).
269: C W1(I) : WEIGHT BEFORE SPLITTING.
270: C W2(I) : WEIGHT THAT IS GIVEN TO PARTICLES AFTER WEIGHT-
271: C        WINDOW OR IMPORTANCE CHECK IF SPLITTING IS ALLOWED.
272: C
273: C      NSPL = 0
274: C      call RANU2( IRAND, R(ISF0), NSF, ICON )
275: C
276: C      if ( JWWND.ne.0 ) then
277: C *VOCL LOOP,NOVREC
278: C      do 180 I = ISF0, ISF1
279: C        IGT = IGG(LSFFL(I))
280: C        if ( JTLLT.eq.0 ) then
281: C          IPT(I) = KZREG(IZFFL(I))
282: C        else
283: C          IPT(I) = IBREG(LSFFL(I))
284: C        end if
285: C        W1(I) = WWW(LSFFL(I))
286: C        W2(I) = W1(I)
287: C        WK = WKIL(IGT,IPT(I))
288: C        WS = WSRV(IGT,IPT(I))
289: C        if ( JWTIM.ne.0 ) then
290: C          ITTII = ITT(LSFFL(I))
291: C          WK = WK*WTIME(ITTII)
292: C          WS = WS*WTIME(ITTII)
293: C        end if
294: C        if ( JRWVR.ne.0 ) then
295: C          WK = WK*DBNK(LSFFL(I),KDBNK(1))
296: C          WS = WS*DBNK(LSFFL(I),KDBNK(1))
297: C        end if
298: C        WS = MAX(WMIN,WS)
299: C        WK = MAX(WMIN,WK)
300: C        ISP(I) = 1
301: C        if ( W1(I).gt.2.0*WS .or. W1(I).lt.WK ) then
302: C          ISP(I) = W1(I) /WS + R(I)
303: C          W2(I) = WS
304: C        end if
305: C      180 continue
306: C
307: C      else if ( JIMPT.ne.0 ) then
308: C *VOCL LOOP,NOVREC
309: C      do 190 I = ISF0, ISF1
310: C        W1(I) = WWW(LSFFL(I))
311: C        if ( JTLLT.eq.0 ) then
312: C          IPT(I) = KZREG(IZFFL(I))
313: C        else
314: C          IPT(I) = IBREG(LSFFL(I))
315: C        end if
316: C        XIMN = XIMP(IGG(LSFFL(I)),IPT(I))
317: C        XLS = XIMN/XIM(LSFFL(I))
318: C        ISP(I) = XLS + R(I)
319: C        W2(I) = W1(I) /XLS
320: C        XIM(LSFFL(I)) = XIMN
321: C      190 continue
322: C    end if
323: C
324: C
325: C-----

```

```

326: C      REMOVE KILLED PARTICLES FROM FLIGHT STACK ====
327: C-----
328: C
329: C      ND = NDEAD
330: C *VOCL LOOP,NOVREC
331: C      do 200 I = ISF0, ISF1
332: C        if ( ISP(I).eq.0 ) then
333: C          ND = ND + 1
334: C          LSDED(ND) = LSFFL(I)
335: C        end if
336: C      200 continue
337: C
338: C      .... monitor
339: C
340: C      if ( IPAR.ne.0 ) then
341: C *VOCL LOOP,NOVREC
342: C      do 210 I = ISF0, ISF1
343: C        if ( ISP(I).eq.0 ) then
344: C          WCNTR(9,IPAR) = WCNTR(9,IPAR) + W1(I)
345: C        else if ( W1(I).lt.W2(I) ) then
346: C          WCNTR(10,IPAR) = WCNTR(10,IPAR) + W2(I)
347: C          NCNTR(10,IPAR) = NCNTR(10,IPAR) + 1.0D0
348: C        end if
349: C      210 continue
350: C          NCNTR(9,IPAR) = NCNTR(9,IPAR) + ND - NDEAD
351: C        else
352: C *VOCL LOOP,SCALAR
353: C      do 220 I = ISF0, ISF1
354: C        KP = KKP(LSFFL(I))
355: C        if ( ISP(I).eq.0 ) then
356: C          WCNTR(9,KP) = WCNTR(9,KP) + W1(I)
357: C          NCNTR(9,KP) = NCNTR(9,KP) + 1.0D0
358: C        else if ( W1(I).lt.W2(I) ) then
359: C          WCNTR(10,KP) = WCNTR(10,KP) + W2(I)
360: C          NCNTR(10,KP) = NCNTR(10,KP) + 1.0D0
361: C        end if
362: C      220 continue
363: C    end if
364: C
365: C      if ( JMNTR.ne.0 ) then
366: C *VOCL LOOP,SCALAR
367: C      do 230 I = ISF0, ISF1
368: C        IGT = IGG(LSFFL(I))
369: C        if ( ISP(I).eq.0 ) then
370: C          NKILD(IGT,IPT(I)) = NKILD(IGT,IPT(I)) + 1.0D0
371: C          WKILD(IGT,IPT(I)) = WKILD(IGT,IPT(I)) + W1(I)
372: C        else if ( W1(I).lt.W2(I) ) then
373: C          NSURV(IGT,IPT(I)) = NSURV(IGT,IPT(I)) + 1.0D0
374: C          WSURV(IGT,IPT(I)) = WSURV(IGT,IPT(I)) + W2(I)
375: C        end if
376: C      230 continue
377: C    end if
378: C
379: C
380: C
381: C
382: C      if ( JRRLT.ne.0 ) then
383: C
384: C        call RANU2( IRAND, R(ISF0), NSF, ICON )
385: C
386: C      CCC      if ( JMNTR.eq.0 ) then
387: C      CCC      if ( IPAR.ne.0 ) then
388: C *VOCL LOOP,NOVREC
389: C      do 240 I = ISF0, ISF1
390: C        IGT = IGG(LSFFL(I))

```

src/mvp/splitb.f

```

391:      WK      = WKIL(IGT,IPT(I))
392:      WS      = WSRV(IGT,IPT(I))
393:      if ( JVTIM.ne.0 ) then
394:        ITTII  = ITT(LSFFL(I))
395:        WK      = WK*WTIME(ITTII)
396:        WS      = WS*WTIME(ITTII)
397:      end if
398:      if ( JRWVR.ne.0 ) then
399:        WK      = WK*DBNK(LSFFL(I),KDBNK(1))
400:        WS      = WS*DBNK(LSFFL(I),KDBNK(1))
401:      end if
402:      WS      = MAX(WMIN,WS)
403:      WK      = MAX(WMIN,WK)
404:      IS      = 2
405:      if ( W2(I).lt.WK.and.ISP(I).gt.0 ) then
406:        IS      = W2(I) / WS + R(I)
407:      end if
408: C
409:      ISP2(I) = IS
410: C
411: CCCC      DD      = W2(I)*ISP(I)
412: C
413:      if ( IS.eq.0 ) then
414:        ND      = ND + 1
415:        LSDDED(ND) = LSFFL(I)
416: CCCC      WCNTR(11,IPAR) = WCNTR(11,IPAR) + DD
417: CCCC      NCNTR(11,IPAR) = NCNTR(11,IPAR) + ISP(I)
418: CCCC      WCNTR(5,IPAR)  = WCNTR(5,IPAR) + DD - W2(I)
419: CCCC      NCNTR(5,IPAR)  = NCNTR(5,IPAR) + ISP(I) - 1
420: CCCC      ISP(I)        = -IZFFL(I)
421:      else if ( IS.eq.1 ) then
422:        WSV      = WS*ISP(I)
423:        WWW(LSFFL(I)) = WSV
424: CCCC      WCNTR(12,IPAR) = WCNTR(12,IPAR) + WSV
425: CCCC      NCNTR(12,IPAR) = NCNTR(12,IPAR) + ISP(I)
426:        W2(I)    = WS
427:      end if
428: 240      continue
429: C
430: C      ... monitor
431: C
432:      if ( IPAR.ne.0 ) then
433: *VOCL LOOP,NOVREC
434:        do 250 I = ISF0, ISF1
435:          if ( ISP2(I).eq.0 ) then
436:            DD      = W2(I)*ISP(I)
437:            WCNTR(11,IPAR) = WCNTR(11,IPAR) + DD
438:            NCNTR(11,IPAR) = NCNTR(11,IPAR) + ISP(I)
439:            WCNTR(5,IPAR)  = WCNTR(5,IPAR) + DD - W2(I)
440:            NCNTR(5,IPAR)  = NCNTR(5,IPAR) + ISP(I) - 1
441:            ISP(I)        = 0
442:          else if ( ISP2(I).eq.1 ) then
443:            WCNTR(12,IPAR) = WCNTR(12,IPAR) + WWW(LSFFL(I))
444:            NCNTR(12,IPAR) = NCNTR(12,IPAR) + ISP(I)
445:          end if
446: 250      continue
447:        else
448: *VOCL LOOP,SCALAR
449:          do 260 I = ISF0, ISF1
450:            KP      = KKP(LSFFL(I))
451:            if ( ISP2(I).eq.0 ) then
452:              DD      = W2(I)*ISP(I)
453:              WCNTR(11,KP) = WCNTR(11,KP) + DD
454:              NCNTR(11,KP) = NCNTR(11,KP) + ISP(I)
455:              WCNTR(5,KP) = WCNTR(5,KP) + DD - W2(I)

```

```

456:            NCNTR(5,KP) = NCNTR(5,KP) + ISP(I) - 1
457:            ISP(I)      = 0
458:          else if ( ISP2(I).eq.1 ) then
459:            WCNTR(12,KP) = WCNTR(12,KP) + WWW(LSFFL(I))
460:            NCNTR(12,KP) = NCNTR(12,KP) + ISP(I)
461:          end if
462: 260      continue
463:          end if
464: C
465:          if ( JMNTR.ne.0 ) then
466: *VOCL LOOP,SCALAR
467:            do 270 I = ISF0, ISF1
468:              IGT      = IGG(LSFFL(I))
469:              if ( ISP2(I).eq.0 ) then
470:                NKILD(IGT,IPT(I)) = NKILD(IGT,IPT(I)) + ISP(I)
471:                WKILD(IGT,IPT(I)) = WKILD(IGT,IPT(I)) + W2(I)*ISP(I)
472:              else if ( ISP2(I).eq.1 ) then
473:                NSURV(IGT,IPT(I)) = NSURV(IGT,IPT(I)) + ISP(I)
474:                WSURV(IGT,IPT(I)) = WSURV(IGT,IPT(I)) +
475:                  & WWW(LSFFL(I))
476:              end if
477: 270      continue
478:            end if
479:          end if
480: C
481: C-----
482: C COMPRESS FREE-FLIGHT STACK IF THERE ARE ANY KILLED PARTICLES.
483: C-----
484: C
485:          if ( ND.gt.NDEAD ) then
486:            II      = ISF0 - 1
487:            NDEAD   = ND
488:            do 280 I = ISF0, ISF1
489:              if ( ISP(I).eq.0 ) NFFL(IZFFL(I)) = NFFL(IZFFL(I)) - 1
490: 280      continue
491: *VOCL LOOP,NOVREC
492:            do 290 I = ISF0, ISF1
493:              if ( ISP(I).gt.0 ) then
494:                II      = II + 1
495:                ISP(II) = ISP(I)
496:                LSFFL(II) = LSFFL(I)
497:                IZFFL(II) = IZFFL(I)
498:                W1(II)   = W1(I)
499:                W2(II)   = W2(I)
500:              end if
501: 290      continue
502: C
503:            NSF      = II - ISF0 + 1
504:            NFFL(NZONE+1) = II
505:            if ( NSF.eq.0 ) go to 720
506: C
507:            ISF1      = II
508:          end if
509: C
510: C ISP(I) : NUMBER OF PARTICLES GENERATED BY SPLITTING
511: C
512:          do 300 I = ISF0, ISF1
513:            ISP(I)    = ISP(I) - 1
514:            NSPL      = NSPL + ISP(I)
515: 300      continue
516: C-----
517: C CHECK WHETHER ALL SPLITTINGS ARE ALLOWED OR NOT.
518: C-----
519: C
520: C-----

```

src/mvp/splitb.f

```

521: C      NSPL = 0 : NO NEW PERTICLE IS BORN ! CHANGE WEIGHTS ONLY
522: C      AND RETURN.
523: C-----
524: C
525: C      if ( NSPL.eq.0 ) then
526: C      *VOCL LOOP,NOVREC
527: C      do 310 I = ISF0, ISF1
528: C      WWW(LSFFL(I)) = W2(I)
529: C      310 continue
530: C      go to 720
531: C      end if
532: C-----
533: C      NSPL > NDEAD : TOO MANY PARTICLES ARE BORN! SOME SPLITTINGS
534: C      ARE PREVENTED
535: C-----
536: C      if ( NSPL.gt.NDEAD ) then
537: C-----
538: C      ALL SPLITTINGS ARE PREVENTED ! MONITOR ONLY ! AND RETURN
539: C-----
540: C      if ( NDEAD.eq.0 ) then
541: C      *VOCL LOOP,NOVREC
542: C      do 320 I = ISF0, ISF1
543: C      if ( ISP(I).le.0.and.W1(I).lt.W2(I) ) then
544: C      WWW(LSFFL(I)) = W2(I)
545: C      end if
546: C      320 continue
547: C
548: C      if ( IPAR.ne.0 ) then
549: C      *VOCL LOOP,NOVREC
550: C      do 330 I = ISF0, ISF1
551: C      if ( ISP(I).gt.0 ) then
552: C      WCNTR(6,IPAR) = WCNTR(6,IPAR) + W1(I)
553: C      NCNTR(6,IPAR) = NCNTR(6,IPAR) + 1.0D0
554: C      end if
555: C      330 continue
556: C      else
557: C      *VOCL LOOP,SCALAR
558: C      do 340 I = ISF0, ISF1
559: C      if ( ISP(I).gt.0 ) then
560: C      KP = KKP(LSFFL(I))
561: C      WCNTR(6,KP) = WCNTR(6,KP) + W1(I)
562: C      NCNTR(6,KP) = NCNTR(6,KP) + 1
563: C      if ( IGG(LSFFL(I)).le.NGP1 ) then
564: C      WCNTR(6,1) = WCNTR(6,1) + W1(I)
565: C      NCNTR(6,1) = NCNTR(6,1) + 1.0D0
566: C      else
567: C      WCNTR(6,2) = WCNTR(6,2) + W1(I)
568: C      NCNTR(6,2) = NCNTR(6,2) + 1.0D0
569: C      end if
570: C      end if
571: C      340 continue
572: C      end if
573: C      go to 720
574: C      end if
575: C
576: C-----
577: C      SELECT PARTICLES TO SPLIT.
578: C-----
579: C      FS = NDEAD / FLOAT(NSPL)**2
580: C      FS = NDEAD/FLOAT(NSPL)
581: C      NS = 0
582: C      do 350 I = ISF0, ISF1
583: C      Pl(I) = FS**ISP(I)
584: C      ISP(I) = -ISP(I)
585: C      350 continue

```

```

586: C
587: C      ..... REPEAT RANDOM SAMPLING UPTO 10 TIMES .....
588: C
589: C      do 370 K = 1, 10
590: C      call RANU2( IRAND, R(ISF0), NSF, ICON )
591: C      do 360 I = ISF0, ISF1
592: C      CCCCCCCCCCCCCCCC IF( R(I) .LT. -ISP(I)*FS ) THEN
593: C      if ( R(I).lt.-ISP(I)*Pl(I) ) then
594: C      NS = NS - ISP(I)
595: C
596: C      ..... RECOVER PLUS SIGN OF ISP FOR SELECTED PARTICLES....
597: C
598: C      if ( NS.le.NDEAD ) ISP(I) = -ISP(I)
599: C      end if
600: C      360 continue
601: C      if ( NS.ge.NDEAD ) go to 380
602: C      370 continue
603: C      380 continue
604: C
605: C-----
606: C      PREVENT SPLITTING FOR PARTICLES WHOSE ISP < 0
607: C-----
608: C
609: C      NSPL = 0
610: C      *VOCL LOOP,NOVREC
611: C      do 390 I = ISF0, ISF1
612: C      if ( ISP(I).lt.0 ) then
613: C
614: C      ... restore weight
615: C      WWW(LSFFL(I)) = W2(I)*(1-ISP(I))
616: C      else
617: C      NSPL = NSPL + ISP(I)
618: C      end if
619: C      390 continue
620: C
621: C      if ( IPAR.ne.0 ) then
622: C      *VOCL LOOP,NOVREC
623: C      do 400 I = ISF0, ISF1
624: C      if ( ISP(I).lt.0 ) then
625: C      WCNTR(6,IPAR) = WCNTR(6,IPAR) + W1(I)
626: C      NCNTR(6,IPAR) = NCNTR(6,IPAR) + 1.0D0
627: C      end if
628: C      400 continue
629: C      else
630: C      *VOCL LOOP,SCALAR
631: C      do 410 I = ISF0, ISF1
632: C      if ( ISP(I).lt.0 ) then
633: C      KP = KKP(LSFFL(I))
634: C      WCNTR(6,KP) = WCNTR(6,KP) + W1(I)
635: C      NCNTR(6,KP) = NCNTR(6,KP) + 1.0D0
636: C      if ( IGG(LSFFL(I)).le.NGP1 ) then
637: C      WCNTR(6,1) = WCNTR(6,1) + W1(I)
638: C      NCNTR(6,1) = NCNTR(6,1) + 1.0D0
639: C      else
640: C      WCNTR(6,2) = WCNTR(6,2) + W1(I)
641: C      NCNTR(6,2) = NCNTR(6,2) + 1.0D0
642: C      end if
643: C      end if
644: C      410 continue
645: C      end if
646: C      end if
647: C
648: C-----
649: C      RENEW WEIGHTS IN THE BANK & MONITOR FOR CREATED PARTICLES
650: C-----

```

src/mvp/splitb.f

```

651: C
652: *VOCL LOOP,NOVREC
653:   do 420 I = ISF0, ISF1
654:     if ( ISP(I).ge.0 ) WWW(LSFFL(I)) = W2(I)
655:     420 continue
656:
657:     if ( IPAR.ne.0 ) then
658:       *VOCL LOOP,NOVREC
659:       do 430 I = ISF0, ISF1
660:         if ( ISP(I).ge.0 ) then
661:           WCNTR(5,IPAR) = WCNTR(5,IPAR) + ISP(I)*W2(I)
662:         end if
663:       430 continue
664:       NCNTR(5,IPAR) = NCNTR(5,IPAR) + NSPL
665:     else
666:       *VOCL LOOP,SCALAR
667:       do 440 I = ISF0, ISF1
668:         if ( ISP(I).ge.0 ) then
669:           DD = ISP(I)*W2(I)
670:           IKPID = KKP(LSFFL(I))
671:           WCNTR(5,IKPID) = WCNTR(5,IKPID) + DD
672:           NCNTR(5,IKPID) = NCNTR(5,IKPID) + ISP(I)
673:         C   if ( IGG(LSFFL(I)).le.NCPI ) then
674:         C     WCNTR(5,1) = WCNTR(5,1) + DD
675:         C     NCNTR(5,1) = NCNTR(5,1) + ISP(I)
676:         C   else
677:         C     WCNTR(5,2) = WCNTR(5,2) + DD
678:         C     NCNTR(5,2) = NCNTR(5,2) + ISP(I)
679:         C   end if
680:         end if
681:       440 continue
682:     end if
683: C
684:   if ( NSPL.eq.0 ) go to 720
685: C
686: C-----
687: C MX: THE MAXIMUM NUMBER OF NEWLY GENERATED PARTICLES FROM ONE PARTICLE
688: C   IF MX = 0, THERE ARE NO NEED TO SPLIT !!
689: C-----
690:   MX = 0
691:   do 450 I = ISF0, ISF1
692:     MX = MAX(MX,ISP(I))
693:   450 continue
694: C   IF(MX.EQ.0) GOTO 9000
695: C-----
696: C DETERMINE BANK POINTERS FOR EACH GENERATED PARTICLE,
697: C REGISTER THEM TO FREE-FLIGHT STACK
698: C AND MEMORIZE BANK-POINTERS OF THEIR MOTHER-PARTICLES (IPT).
699: C-----
700:   NSP = ISF1
701:   NDEAD = NDEAD - NSPL
702:   NDD = NDEAD - ISF1
703:   do 470 M = 1, MX
704:     *VOCL LOOP,NOVREC
705:     do 460 I = ISF0, ISF1
706:       if ( ISP(I).ge.M ) then
707:         NSP = NSP + 1
708:         IPT(NSP) = LSFFL(I)
709:         LSFFL(NSP) = LSDED(NDD+NSP)
710:         IZFFL(NSP) = IZFFL(I)
711:       end if
712:     460 continue
713:   470 continue
714:   NFFL(NZONE+1) = NSP
715:   do 480 I = ISF0, ISF1

```

```

716:     if ( ISP(I).ge.1 ) NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + ISP(I)
717:   480 continue
718: C
719: C=====
720: C=== SPLITTING STARTS HERE !! =====
721: C=====
722: C
723:   IS = ISF1 + 1
724:   IE = NFFL(NZONE+1)
725: C-----
726: C POSITION, DIRECTION, ENERGY, ZONE & TIME
727: C-----
728: *VOCL LOOP,NOVREC
729:   do 490 I = IS, IE
730:     XXX(LSFFL(I)) = XXX(IPT(I))
731:     YYY(LSFFL(I)) = YYY(IPT(I))
732:     ZZZ(LSFFL(I)) = ZZZ(IPT(I))
733:     AAA(LSFFL(I)) = AAA(IPT(I))
734:     BBB(LSFFL(I)) = BBB(IPT(I))
735:     CCC(LSFFL(I)) = CCC(IPT(I))
736:     WWW(LSFFL(I)) = WWW(IPT(I))
737:     KKP(LSFFL(I)) = KKP(IPT(I))
738:     IZZ(LSFFL(I)) = IZZ(IPT(I))
739:     IGG(LSFFL(I)) = IGG(IPT(I))
740:     TTT(LSFFL(I)) = TTT(IPT(I))
741:     KLSF(LSFFL(I)) = KLSF(IPT(I))
742:     EEE(LSFFL(I)) = EEE(IPT(I))
743:     if ( JIMPT.ne.0 ) XIM(LSFFL(I)) = XIM(IPT(I))
744:     IBREG(LSFFL(I)) = IBREG(IPT(I))
745:     if ( JTIME.ne.0 ) ITT(LSFFL(I)) = ITT(IPT(I))
746:   490 continue
747: C
748:   if ( JPERT.ne.0 ) then
749:     *VOCL LOOP,NOVREC
750:     do I = IS, IE
751:       do IPTX = 1, NPTDS
752:         do IOD = 1, NORDDS
753:           WWD(LSFFL(I),IPTX,IOD) = WWD(IPT(I),IPTX,IOD)
754:         end do
755:         WWD2(LSFFL(I),IPTX) = WWD2(IPT(I),IPTX)
756:         do ISPX = 0, NGSP
757:           do IOP = 1, NOPSDS
758:             WSD(LSFFL(I),ISPX,IPTX,IOP) =
759:             & WSD(IPT(I),ISPX,IPTX,IOP)
760:           end do
761:         end do
762:         WWT(LSFFL(I),IPTX) = WWT(IPT(I),IPTX)
763:       end do
764:     end do
765:     do IPTX = 1, NPTCS
766:       WWR(LSFFL(I),IPTX) = WWR(IPT(I),IPTX)
767:     do ISPX = 0, NGSP
768:       WSC(LSFFL(I),ISPX,IPTX) = WSC(IPT(I),ISPX,IPTX)
769:     end do
770:   end do
771: c+beff2
772:   if (NPTBE.gt.0) then
773:     WWB(LSFFL(I)) = WWB(IPT(I))
774:     do ISPX = 0, NGSP
775:       WSB(LSFFL(I),ISPX) = WSB(IPT(I),ISPX)
776:     end do
777:     WWBD(LSFFL(I)) = WWBD(IPT(I))
778:     WWLD(LSFFL(I)) = WWLD(IPT(I))
779:     do ISPX = 1, NGSP
780:       WSD(LSFFL(I),ISPX) = WSD(IPT(I),ISPX)

```

src/mvp/splitb.f

```

781:          WSLD(LSFFL(I),ISPX) = WSLD(IPT(I),ISPX)
782:        end do
783:      end if
784: c-beff2
785:    end do
786:  end if
787:
788: C-----
789: C   LATTICE GEOMETRY PARAMETER
790: C-----
791:   if ( JLAT.ne.0 ) then
792: *VOCL LOOP,NOVREC
793:   do 500 I = IS, IE
794:     LEVL(LSFFL(I)) = LEVL(IPT(I))
795:   continue
796:   do 520 K = 1, NEST
797: *VOCL LOOP,NOVREC
798:   do 510 I = IS, IE
799:     LZZ(LSFFL(I),K) = LZZ(IPT(I),K)
800:     LPOS(LSFFL(I),K) = LPOS(IPT(I),K)
801:     if ( JHLAT.ne.0 ) LCRS(LSFFL(I),K) = LCRS(IPT(I),K)
802:     if ( JTLT.ne.0 ) IBSPC(LSFFL(I),K) = IBSPC(IPT(I),K)
803:   510 continue
804:   520 continue
805:   end if
806: C-----
807: C   SIGMA BANK EXCEPT EEE (ENERGY)
808: C   ( NOT PERFORMED WHEN CALLED FROM COLLISION ROUTINE ! )
809: C-----
810:   if ( JJC.COL.eq.0 ) then
811:     do 540 K = 1, MB
812: *VOCL LOOP,NOVREC
813:     do 530 I = IS, IE
814:       KMAC(LSFFL(I),K,1) = KMAC(IPT(I),K,1)
815:       KMAC(LSFFL(I),K,2) = KMAC(IPT(I),K,2)
816:     530 continue
817:     540 continue
818:     do 570 K = 1, MB
819:       do 560 L = 1, NSMAC
820: *VOCL LOOP,NOVREC
821:       do 550 I = IS, IE
822:         SMAC(LSFFL(I),K,L) = SMAC(IPT(I),K,L)
823:         if ( JPERT.ne.0 )
824:           & SMACP(LSFFL(I),K,L) = SMACP(IPT(I),K,L)
825:       550 continue
826:       560 continue
827:       570 continue
828: C
829: *VOCL LOOP,NOVREC
830:       do 580 I = IS, IE
831:         MMAC(LSFFL(I),1) = MMAC(IPT(I),1)
832:         MMAC(LSFFL(I),2) = MMAC(IPT(I),2)
833:       580 continue
834:       else
835: C-----
836: C   CLEAR MMAC ARRAY AFTER COLLISION
837: C-----
838: *VOCL LOOP,NOVREC
839:       do 590 I = IS, IE
840:         MMAC(LSFFL(I),2) = 0
841:       590 continue
842:       end if
843: C
844:       do 620 K = 1, NUC
845:         do 610 L = 1, NSMIC

```

```

846: *VOCL LOOP,NOVREC
847:       do 600 I = IS, IE
848:         SMIC(LSFFL(I),K,L) = SMIC(IPT(I),K,L)
849:         if ( JPERT.ne.0 )
850:           & SMICP(LSFFL(I),K,L) = SMICP(IPT(I),K,L)
851:       600 continue
852:       610 continue
853:       620 continue
854: c##<2007/03/14:PN3:
855: C
856:       if ( JPHNU.ne.0 ) then
857:         do K = 1, NUCPN
858:           do L = 1, NSMICPN
859: *VOCL LOOP,NOVREC
860:           do I = IS, IE
861:             SMICPN(LSFFL(I),K,L) = SMICPN(IPT(I),K,L)
862:           end do
863:         end do
864:       end do
865:     end if
866: c##>
867: C
868:     do 640 K = 1, NUC
869: *VOCL LOOP,NOVREC
870:     do 630 I = IS, IE
871:       KSPI(LSFFL(I),K) = KSPI(IPT(I),K)
872:     630 continue
873:     640 continue
874:     do 660 K = 1, NSTAL
875: *VOCL LOOP,NOVREC
876:     do 650 I = IS, IE
877:       SGTAL(LSFFL(I),K) = SGTAL(IPT(I),K)
878:     650 continue
879:     660 continue
880: C-----
881: C   optional bank parameters
882: C-----
883:     do 680 K = 1, KDBNK(0)
884: *VOCL LOOP,NOVREC
885:     do 670 I = IS, IE
886:       DBNK(LSFFL(I),K) = DBNK(IPT(I),K)
887:     670 continue
888:     680 continue
889:     do 700 K = 1, KIBNK(0)
890: *VOCL LOOP,NOVREC
891:     do 690 I = IS, IE
892:       IBNK(LSFFL(I),K) = IBNK(IPT(I),K)
893:     690 continue
894:     700 continue
895: C
896: C..... MONITOR ON
897: C
898:       if ( JMNTR.ne.0 ) then
899: *VOCL LOOP,SCALAR
900:       do 710 I = IS, IE
901:         if ( JTLT.eq.0 ) then
902:           IRT = KZREG(IZFFL(I))
903:         else
904:           IRT = IBREG(LSFFL(I))
905:         end if
906:         IGT = IGG(LSFFL(I))
907:         NSPLT(IGT,IRT) = NSPLT(IGT,IRT) + 1.0D0
908:         WSPLT(IGT,IRT) = WSPLT(IGT,IRT) + WWW(LSFFL(I))
909:       710 continue
910:       end if

```

src/mvp/splitb.f

```
911: C
912: C =====
913: C
914:   720 continue
915: C
916:   return
917:   end
```

SAFE

src/mvp/srcold.f

```

1: C/IF SOURCE(NEW)
2:   subroutine sdummy()
3:     return
4: C/ELSE
5:
6: C/IF ARGSAVE
7: *   SUBROUTINE SOURCE( IOW,A,RH,DH, IRAND,NTGEN,NBANK,NDEAD,NGROUP,
8: *   N NGP1,NGP2,NZONE,NFBANK,NFBNK0,NFISB,NEST,JBPRT,NLOST,NEVENT,
9: *   N NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,MCX,MCXP,NUC,NPATOM,NSMIC,
10: *   B NMKREG,XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,IZZ,IGG,TTT,ITT,KLSF,EEE,
11: *   B IBREG,IBSPC,SMIC,LMIC,KSPI,MMAC,XXXF,YYYF,ZZZF,IZZF,EEEF,INUF,
12: *   F LEVL,LZZ,LPOS,LCRS,LEVLF,IZZF,LPOSF,LCRSF,IBRGF,IBSPF,KLSFF,XIM,
13: *   B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
14: *   G MLBZZ,KZMAT,KZREG, MEMZN, SDA,
15: *   G KZDA,KZAA, IPCEL,XIMP,
16: *   S LSFFL,NFFL,IZFFL,LSDED,LSLAT,IZLAT,NXLT,
17: *   * KSOUR,ISZON,SOUR,PSPAC,PENRG,KENRG,
18: *   * ENGYB,ENGPB,NSTIM,STIM,PSTIM,ISTIM,NSANG,
19: *   * SANG,SAXIS,PSANG,ISANG,IFISM, WGTF,
20: *   T WSUM,XSOC,NCNTR,WCNTR,EBOT,ETOP,
21: *   T NLENG,XSXV,XAVT,AAVT,EAVT,
22: *   X CX,ICX,KLIB1,XLIB1,KLIB2,KLB2S,XLIB2,
23: *   X NNK,NMT,EINCD,CRES,KCRES,SGTAL,MCRES,NSTAL,
24: *   X CXP,ICXP,KLBP1,KLBP2,XLBP2,NMTP,EBOTP,ETOPP,
25: *   W R,X,Y,Z,IWK1,IWK2,IWK3,EW1,RWK1,IFL,IFI,NNSRC,XSOUR,SMC,
26: C/ELSE
27: *   SUBROUTINE SOURCE( IOW,A,RH,DH, IRAND,NTGEN,NBANK,NDEAD,NGROUP,
28: *   N NGP1,NGP2,NZONE,NFBANK,NFBNK0,NFISB,NEST,JBPRT,NLOST,NEVENT,
29: *   N NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,MCX,MCXP,NUC,NPATOM,
30: *   N NSMIC, NMKREG, XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,IZZ,
31: *   B IGG,TTT,ITT,KLSF,EEE,IBREG,IBSPC, SMIC,LMIC,
32: *   B KSPI,MMAC,XXXF,YYYF,ZZZF,IZZF,EEEF,
33: *   B INUF,LEVL,LZZ,LPOS,LCRS,LEVLF,
34: *   B LZZF,LPOSF,LCRSF,IBRGF,IBSPF,KLSFF,XIM,
35: *   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
36: *   G MLBZZ,KZMAT,KZREG, MEMZN, SDA,
37: *   G KZDA,KZAA, IPCEL,XIMP,
38: *   S LSFFL,NFFL,IZFFL,LSDED,LSLAT,IZLAT,NXLT,
39: *   * KSOUR,ISZON,SOUR,PSPAC,PENRG,KENRG,
40: *   * ENGYB,ENGPB,NSTIM,STIM,PSTIM,ISTIM,NSANG,
41: *   * SANG,SAXIS,PSANG,ISANG,IFISM, WGTF,
42: *   T WSUM,XSOC,NCNTR,WCNTR,EBOT,ETOP,
43: *   T NLENG,XSXV,XAVT,AAVT,EAVT,
44: *   X CX,ICX,KLIB1,XLIB1,KLIB2,KLB2S,XLIB2,
45: *   X NNK,NMT,EINCD,CRES,KCRES,SGTAL,MCRES,NSTAL,
46: *   X CXP,ICXP,KLBP1,KLBP2,XLBP2,NMTP,EBOTP,ETOPP,
47: *   W R,X,Y,Z,IWK1,IWK2,IWK3,EW1,RWK1,IFL,IFI,NNSRC,XSOUR,SMC,
48: *   %   NGENE, NGENE0,NBATCH,WSUMB )
49: C/ENDIF
50: C
51: C=<MVP>=====
52: C PURPOSE: GENERATE PARTICLES
53: C   (old type. maybe obsolescent in a future version)
54: C CALLED IN: ACTION          CALLS: RANU2,VMNTR1,BSVDEC,FISSRC
55: C
56: C=====
57: C   IMPLICIT REAL*8 (D)
58: C   IMPLICIT REAL*8 (A-H,O-Q,S-Z)
59: C   IMPLICIT REAL (R)
60: C
61: C   PARAMETER ( KNUTRN = 1, KPHOTN = 2 )
62: C
63: C   include 'INC/_FLAGS'
64: C   include '../shared/INC/_TASKDT'
65: C

```

```

66:   real A(*)
67:   real RH(*)
68:   real*8 DH(*)
69: C
70: C/IF INTEGER8
71: *   integer*8 NTGEN
72: C/ELSE
73:   integer NTGEN
74: C/ENDIF
75: C
76: C .... PARTICLE BANK .....
77: C
78:   REAL*8      XXX(NBANK),ZZZ(NBANK),YYY(NBANK),AAA(NBANK),
79:   &            BBB(NBANK),CCC(NBANK)
80:   REAL        WWW(NBANK),EEE(NBANK),XIM(NBANK)
81:   real*8      TTT(NBANK)
82:   REAL        WGT(NREG)
83:   real        SMIC(NBANK,NUC,NSMIC),SGTAL(NBANK,NSTAL)
84:   INTEGER      IZZ(NBANK),IGG(NBANK),
85:   &            LEVL(NBANK),LZZ(NBANK,NEST),LPOS(NBANK,NEST),
86:   &            LCRS(NBANK,NEST),KSPI(NBANK,NUC),MMAC(NBANK,2),
87:   &            KLSF(NBANK)
88:   INTEGER      ITT(NBANK)
89: C
90:   INTEGER      IBSPC(NBANK,0:NEST),IBREG(NBANK)
91:   INTEGER      LMIC(8)
92: C
93:   real*8 DBNK(NBANK,*)
94:   integer KDBNK(0:MDBNK)
95:   integer IBNK(NBANK,*)
96:   integer KIBNK(0:MIBNK)
97: C
98: C .... FISSION BANK .....
99: C
100:   REAL*8      XXXF(NFBNK0),YYYF(NFBNK0),ZZZF(NFBNK0)
101:   REAL        EEEF(NFBNK0)
102:   INTEGER      IZZF(NFBNK0),LEVLF(NFBNK0),LZZF(NFBNK0,NEST),
103:   &            LPOSF(NFBNK0,NEST),LCRSF(NFBNK0,NEST),
104:   &            INUF(NFBNK0),KLSFF(NFBNK0)
105:   INTEGER      IBSPF(NFBNK0,NEST),IBRGF(NFBNK0)
106: C
107: C .... TALLY .....
108: C
109:   REAL*8      WSUM,XSOC(NLENG),XSXV(NLENG,3),WSUMB
110:   REAL        XAVT(3),AAVT(3),EAVT
111:   REAL*8      WCNTR(NEVENT,2),NCNTR(NEVENT,2)
112: C
113: C .... VARIANCE REDUCTION DATA
114: C
115:   REAL        XIMP(NGROUP,NREG)
116: C
117: C .... STACKS ... (FREE FLIGHT & MORGUE) .....
118: C
119:   INTEGER      LSFFL(NBANK),NFFL(NZONE+1),IZFFL(NBANK),LSDED(NBANK)
120: C
121: C .... STACKS ... (LATTICE SEARCH) .....
122: C
123:   INTEGER      LSLAT(NBANK),NXLT(*),IZLAT(NBANK)
124: C
125: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 4*NBANK FOR USE
126: C   IN FNSSOC )
127: C
128:   REAL        R(5*NBANK),EW1(NBANK),RWK1(NBANK)
129:   INTEGER      IWK1(NBANK),IWK2(NBANK),IWK3(NBANK)
130:   LOGICAL      IFI(NBANK),IFL(NBANK)

```

src/mvp/srcold.f

```

131:      REAL*8      X(NBANK),Y(NBANK),Z(NBANK)
132:      real        SMCW(NBANK,NSMIC)
133: C
134: C .... SOURCE DATA .....
135: C
136:      INTEGER      KSOUR(NSOUR),ISZON(2,NSOUR),KENRG(2,NGROUP,NSOUR),
137: &                 NSTIM(NSOUR),ISTIM(NSOUR),NSANG(NSOUR),ISANG(NSOUR),
138: &                 IFISM(NSOUR),MEMZN(NMEMS,NSOUR)
139:      REAL*8      SOUR(NSOUR)
140:      real        PSPAC(10,NSOUR),
141: &                 PENRG(NGROUP,NSOUR),ENGYB(NGP1+1),ENGPB(NGP2+1),
142: &                 STIM(*),PSTIM(*),SANG(*),PSANG(*),SAXIS(3,NSOUR),
143: &                 EINCd
144: C
145: C .... GEOMETRY ARRAY DATA
146: C
147:      INTEGER      KZMAT(NZONE),KZREG(NZONE),KZDA(*),KZAA(*),
148: &                 MLBZZ(NZONE),IPCEL(*)
149:      REAL*8      SDA(*)
150: C
151: C CROSS SECTION ..... (NEUTRON)
152: C
153:      REAL        EBOT,ETOP
154:      REAL        CX(MCX)
155:      INTEGER      ICX(MCX)
156:      REAL        XLIB1(NUC,11),XLIB2(NUC,NMT,4)
157:      INTEGER      KLIB1(NUC,27),KLIB2(NUC,NMT,21)
158: C
159: C .... CROSS SECTION & POINTERS ( PHOTON ).....
160: C
161:      REAL        CXP(MCXP)
162:      INTEGER      ICXP(MCXP),KLBp1(NPATOM,20),KLBp2(NPATOM,NMTP,6)
163:      REAL        XLBP2(NPATOM,NMTP,2)
164: C
165:      INTEGER      NNSRC(NSOUR)
166:      REAL*8      XSOUR(NSOUR)
167: C
168:      REAL        CRES(MCRES)
169:      INTEGER      KCRES(NSTAL,4)
170: C
171:      include './shared/INC/_PMLATT'
172:      include './shared/INC/_SFLATT'
173: C
174: C/#IF ARGSAVE
175:      RETURN
176: C
177: C -----
178: C
179:      ENTRY        SOURCE( NGENE , NGENEO ,NBATCH, WSUMB )
180: C/#ENDIF
181: C
182: C -----
183: C .... IF ALL PARTICLES ARE DEAD, CLEAR UP DEAD PARTICLE STACK .....
184: C -----
185:      IF(NDEAD.EQ.NBANK) THEN
186:          DO 100 I=1,NBANK
187:              LSDED(I) = I
188:          100 CONTINUE
189:      ELSE
190:          WRITE(IOW,*) ' !!! NDEAD IS NOT EQUAL TO NBANK !!!'
191:          WRITE(IOW,*) '          SOMETHING WRONG HAPPENS, STOP !!!'
192:          STOP 666
193:      ENDIF
194: C -----
195: C ..... NGENEO, WSUMB : MEMORIZE NGENE FOR USE IN TALSUM .....

```

```

196: C-----
197:      NGENEO = NGENE
198:      WSUMB = DBLE( NGENEO )
199: C/#IF PARA
200: C/#IF PARA(SX* CRAY)
201: *      call MVPSYNC_LOCK(2)
202: C/#ENDIF
203:      if ( JBPRNT.ne.0 ) then
204:          WRITE(IOW,
205: &              '(1x,'' TASK '' ,I5,'' GENERATE '' ,I6,'' SOURCES.   IRAND = '' ,
206: &              I11)')
207:          &      idtask,NGENE,IRAND
208:          end if
209: C/#IF PARA(SX* CRAY)
210: *      call MVPSYNC_UNLOCK(2)
211: C/#ENDIF
212: C/#ELSE
213: *      if ( JBPRNT.ne.0 ) then
214: *          WRITE(IOW,
215: * &              '(1x,'' GENERATE '' ,I6,'' SOURCES.   IRAND = '' ,I11)')
216: *          &      NGENE,IRAND
217: *          end if
218: C/#ENDIF
219: C
220:      IF(JVMNT.NE.0) CALL VMNTRI(1,NGENE)
221: C
222: C
223: C =====
224: C FIXED SOURCE PROBLEM OR INITIAL SOURCE FOR EIGENVALUE PROBLEM
225: C =====
226: C
227: C
228:      IF( JEIGN.EQ.0.OR.NBATCH.LE.0 ) THEN
229: C-----
230: C .... DETERMINE PARTICLE NUMBERS FROM EACH SOURCE IF NSOUR > 1 .....
231: C-----
232:      IF(NSOUR.GT.1) THEN
233:          NSS = 0
234: *VOCL LOOP,SCALAR
235:          DO 110 N=1,NSOUR
236:              NNSRC(N) = NGENE*SOUR(N)
237:              XSOUR(N) = FLOAT(NGENE)*SOUR(N) - NNSRC(N)
238:              IF(N.GT.1) XSOUR(N) = XSOUR(N-1) + XSOUR(N)
239:              NSS = NSS + NNSRC(N)
240:          110 CONTINUE
241: C-----
242: C .... IF THE SUM OF NNSRC IS LESS THAN NGENE, THE REST OF SOURCE
243: C PARTICLES ARE GENERATED WITH PROBABILITIES PROPORTIONAL TO THE
244: C DECIMAL PARTS OF NGENE*SOUR OF EACH SOURCE. (XSOUR).
245: C-----
246:          NSS = NGENE - NSS
247:          IF(NSS.ne.0.0) THEN
248:              CALL RANU2( IRAND,R,abs(NSS),ICON)
249: *VOCL LOOP,SCALAR
250:              DO 140 K=1,abs(NSS)
251:                  R(K) = R(K)*XSOUR(NSOUR)
252:                  DO 120 L=1,NSOUR
253:                      IF(XSOUR(L).GT.R(K)) GOTO 130
254:                  120 CONTINUE
255:                  L = NSOUR
256:                  NNSRC(L) = NNSRC(L) + sign(1,NSS)
257:                  140 CONTINUE
258:              ENDIF
259:          ELSE
260:              NNSRC(1) = NGENE

```

src/mvp/srcold.f

```

261:          ENDIF
262: C-----
263: C ..... GENERATE PARTICLES .....
264: C-----
265:          NSS      = 0
266:          IST      = 0
267:          IPP      = NDEAD
268:          ILOST    = 0
269:
270:          DO 400 N=1,NSOUR
271:             NN     = NNSRC(N)
272:             KS     = KSOUR(N)
273:
274: CC          IF(JMNTR.NE.0)
275: CM &        WRITE(IOW,*) ' SOURCE # ',N,' : ',NN,' PARTICLES'
276: CCCCCCCCCC IF(JVMNT.NE.0) CALL VMNTR1(1,NN)
277:
278:          IPP      = IPP - NN
279:          IF(JLATT.NE.0) THEN
280:             DO 150 I=1,NN
281:                LEVL(LSDED(IPP+I)) = 0
282:          150          CONTINUE
283:          ENDIF
284:          DO 132 I=1,NN
285:             KLSF(LSDED(IPP+I)) = 0
286:          132          CONTINUE
287: C
288: CM1994-6-1
289: C          IF(KS.GT.0) THEN
290: CM1994-6-1
291: C
292: C          ===== SAMPLING OF POSITIONS =====
293: C
294: C-----
295: C TYPE 1 ..... ISOTROPIC POINT SOURCE .....
296: C
297: C TYPE 2 ..... MONO DIRECTIONAL POINT SOURCE .....
298: C          PSPAC(1:3) ... COORDINATE, PSPAC(4:6) ... DIRECTION
299: C-----
300:          IF(KS.EQ.1.OR.KS.EQ.2) THEN
301:             DO 160 I=1,NN
302:                IWK1(NSS+I) = LSDED(IPP+I)
303:                IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I))=ISZON(1,N)
304:                XXX(LSDED(IPP+I)) = PSPAC(1,N)
305:                YYY(LSDED(IPP+I)) = PSPAC(2,N)
306:                ZZZ(LSDED(IPP+I)) = PSPAC(3,N)
307:          160          CONTINUE
308: C-----
309: C TYPE 3 ..... RECTANGULAR SOURCE .....
310: C          PSPAC(1)<X<PSPAC(2), PSPAC(3)<Y<PSPAC(4), PSPAC(5)<Z<PSPAC(6)
311: C TYPE 4 ..... SPHERICAL SOURCE .....
312: C          PSPAC(1:3) = CENTER, PSPAC(4) = RADIUS
313: C-----
314:          ELSE IF(KS.EQ.3.OR.KS.EQ.4) THEN
315:             CALL RANU2(IRAND,R,3*NN,ICON)
316: *VOCL LOOP,NOVREC
317:          DO 170 I=1,NN
318:             IWK1(NSS+I) = LSDED(IPP+I)
319:             IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I)) = ISZON(1,N)
320:             IF(KS.EQ.3) THEN
321:                XXX(LSDED(IPP+I)) = PSPAC(1,N) +
322:                &                R(I)*( PSPAC(2,N) - PSPAC(1,N) )
323:                YYY(LSDED(IPP+I)) = PSPAC(3,N) +
324:                &                R(NN+I)*( PSPAC(4,N) - PSPAC(3,N) )
325:                ZZZ(LSDED(IPP+I)) = PSPAC(5,N) +

```

```

326:          &                R(2*NN+I)*( PSPAC(6,N) - PSPAC(5,N) )
327:          ELSE IF(KS.EQ.4) THEN
328:             DR      = PSPAC(4,N)*(R(I)**(1/3.0D0))
329:             D1      = 2.0*R(NN+I) - 1.0
330:             D2      = SQRT(1.0-D1**2)
331:             DD      = 6.283185307179586D0*R(2*NN+I)
332:             XXX(LSDED(IPP+I)) = DR*D1 + pspac(1,n)
333:             YYY(LSDED(IPP+I)) = DR*D2*COS(DD) + pspac(2,n)
334:             ZZZ(LSDED(IPP+I)) = DR*D2*SIN(DD) + pspac(3,n)
335:          ENDIF
336:          170          CONTINUE
337: C TYPE 5 ..... CIRCULAR SOURCE MONO-DIRECTIONAL TEMPORARY.....
338: C          PSPAC(1:3) = CENTER, PSPAC(4) = RADIUS
339: C
340:          ELSE IF(KS.EQ.5) THEN
341:             CALL RANU2(IRAND,R,2*NN,ICON)
342:             DO 161 I=1,NN
343:                IWK1(NSS+I)= LSDED(IPP+I)
344:                IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I)) = ISZON(1,N)
345:                IF(PSPAC(5,N).EQ.0.) THEN
346:                   DAX = 1.0
347:                   DAY = 0.0
348:                   DAZ = 0.0
349:                ELSE IF(PSPAC(6,N).EQ.0.) THEN
350:                   DAX = 0.0
351:                   DAY = 1.0
352:                   DAZ = 0.0
353:                ELSE IF(PSPAC(7,N).EQ.0.) THEN
354:                   DAX = 0.0
355:                   DAY = 0.0
356:                   DAZ = 1.0
357:                ELSE
358:                   DD1 = SQRT( PSPAC(6,N)**2 + PSPAC(7,N)**2 )
359:                   DAX = 0.0
360:                   DAY = PSPAC(7,N) / DD1
361:                   DAZ = -PSPAC(6,N) / DD1
362:                ENDIF
363:                DBX = PSPAC(6,N)*DAZ - PSPAC(7,N)*DAY
364:                DBY = PSPAC(7,N)*DAX - PSPAC(5,N)*DAZ
365:                DBZ = PSPAC(5,N)*DAY - PSPAC(6,N)*DAX
366:                DD2 = SQRT(DBX**2 + DBY**2 + DBZ**2)
367:                DBX = DBX / DD2
368:                DBY = DBY / DD2
369:                DBZ = DBZ / DD2
370: C
371:                DR = PSPAC(4,N) * SQRT(R(I))
372:                DD = 6.283185307179586D0 *R(NN+I)
373:                D1 = COS(DD) * DR
374:                D2 = SIN(DD) * DR
375:                XXX(LSDED(IPP+I)) = DAX*D1 + DBX*D2 + PSPAC(1,N)
376:                YYY(LSDED(IPP+I)) = DAY*D1 + DBY*D2 + PSPAC(2,N)
377:                ZZZ(LSDED(IPP+I)) = DAZ*D1 + DBZ*D2 + PSPAC(3,N)
378:          161          CONTINUE
379: C TYPE 6 ..... SQUARE PLANE SOURCE MONO-DIRECTIONAL TEMPORARY.....
380: C          PSPAC(1:3) = VERTEX, PSPAC(4:6) = VECTOR OF 1 PENN
381: C          PSPAC(7 ) = LENGTH OF OTHER :PSPAC(4:6)XPSPAC(8,10)
382: C          PSPAC(8:10)= MONO-DIRECTION PERPENDICULAR TO PSPAC(4:6)
383: C
384:          ELSE IF(KS.EQ.6) THEN
385:             CALL RANU2(IRAND,R,2*NN,ICON)
386:             DO 162 I=1,NN
387:                IWK1(NSS+I)= LSDED(IPP+I)
388:                IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I)) = ISZON(1,N)
389:                DBX = PSPAC(5,N)*PSPAC(10,N)-PSPAC(6,N)*PSPAC(9,N)
390:                DBY = PSPAC(6,N)*PSPAC(8,N)-PSPAC(4,N)*PSPAC(10,N)

```

src/mvp/srcold.f

```

391:          DBZ = PSPAC(4,N)*PSPAC(9,N)-PSPAC(5,N)*PSPAC(8,N)
392:          DD2 = SQRT(DBX**2 + DBY**2 + DBZ**2)
393:          DBX = DBX / DD2
394:          DBY = DBY / DD2
395:          DBZ = DBZ / DD2
396: C
397:          D1 = R(I)
398:          D2 = PSPAC(7,N) * R(I+NN)
399:          XXX(LSDED(IPP+I))=PSPAC(4,N)*D1+DBX*D2+PSPAC(1,N)
400:          YYY(LSDED(IPP+I))=PSPAC(5,N)*D1+DBY*D2+PSPAC(2,N)
401:          ZZZ(LSDED(IPP+I))=PSPAC(6,N)*D1+DBZ*D2+PSPAC(3,N)
402: 162          CONTINUE
403:          ENDIF
404: C-----
405: C SAMPLING OF DIRECTIONS
406: C-----
407:          IF(KS.GT.0.and.
408:          & KS.NE.2.AND.KS.NE.9.AND.KS.NE.5.AND.KS.NE.6) THEN
409:          CALL RANU2(IRAND,R,2*NN,ICON)
410: C.....AUG.27/1991, ISOTROPIC BETWEEN PSPAC(9) AND PSPAC(10)
411: C          PSPAC(9)<= CCC <= PSPAC(10)
412: C          PSPAC(9,N) : MINIMUM ANGLE COSINE
413: C          PSPAC(10,N) : MAXIMUM ANGLE COSINE
414: C          DUMIN = PSPAC(9,N)
415: C          DUMAX = PSPAC(10,N)
416: C          DO 180 I = 1,NN
417: C              D = (DUMAX-DUMIN)*R(I) + DUMIN
418: C              CCC(LSDED(IPP+I)) = D
419: C              D = SQRT(1.0-D**2)
420: C              D2 = 6.283185307179586D0*R(NN+I)
421: C              BBB(LSDED(IPP+I)) = COS(D2)*D
422: C              AAA(LSDED(IPP+I)) = SIN(D2)*D
423: 180          CONTINUE
424: C
425: C ..... MONO DIRECTIONAL .....
426: C
427:          ELSE IF(KS.EQ.2) THEN
428:          DO 190 I = 1,NN
429:          AAA(LSDED(IPP+I)) = PSPAC(4,N)
430:          BBB(LSDED(IPP+I)) = PSPAC(5,N)
431:          CCC(LSDED(IPP+I)) = PSPAC(6,N)
432: 190          CONTINUE
433:          ELSE IF(KS.EQ.5) THEN
434:          DO 181 I = 1,NN
435:          AAA(LSDED(IPP+I)) = PSPAC(5,N)
436:          BBB(LSDED(IPP+I)) = PSPAC(6,N)
437:          CCC(LSDED(IPP+I)) = PSPAC(7,N)
438: 181          CONTINUE
439:          ELSE IF(KS.EQ.6) THEN
440:          DO 182 I = 1,NN
441:          AAA(LSDED(IPP+I)) = PSPAC(8,N)
442:          BBB(LSDED(IPP+I)) = PSPAC(9,N)
443:          CCC(LSDED(IPP+I)) = PSPAC(10,N)
444: 182          CONTINUE
445:          ENDIF
446: C-----
447: C SAMPLING OF ENERGY, WEIGHT & TIME
448: C-----
449:          IF(KS.gt.0.and.KS.NE.9) THEN
450: C-----
451: C SAMPLING FROM FISSION SPECTRUM (GATHER INCIDENT ENERGY & NUCLIDE#)
452: C-----
453:          IF(JEIGN.NE.0.AND.NBATCH.EQ.0) THEN
454:          DO 200 I=1,NN
455:          EW1(I) = EINCD

```

```

456:          IWK3(I) = IFISM(N)
457: 200          CONTINUE
458:          CALL SEF5N2(
459:          & IOW, IWK3, NN, EW1, EW1,
460:          & CX, ICX, MCX, KLIB1, XLIB1, KLIB2, XLIB2,
461:          & NBANK, NUC, NMT, IRAND, JDEBG,
462:          & R(1), R(NN+1), R(2*NN+1), R(3*NN+1), RWK1,
463:          & IWK2, IFI, IFL, X, Y)
464: C
465:          DO 210 I=1,NN
466:          EW1(I) = MAX(EBOT, MIN(EW1(I), ETOP))
467: 210          CONTINUE
468: C
469:          CALL BSVDEC(ENGYB,NGP1+1,EW1, IWK3, NN)
470:          DO 220 I=1,NN
471:          EEE(LSDED(IPP+I)) = EW1(I)
472:          IGG(LSDED(IPP+I)) = IWK3(I)
473: 220          CONTINUE
474: C
475: C-----
476: C SAMPLING FROM PENRG
477: C-----
478: C
479:          ELSE
480: C
481: C ... undocumented option ...
482: C FIRST ENTRY OF PENRG < 0.0 --> MONO ENERGY
483: C SECOND ENTRY OF PENRG 1/2 --> NEUTRON/PHOTON
484: C
485:          IF(PENRG(1,N).LT.0.0) THEN
486:          EE00 = -PENRG(1,N)
487:          IF(PENRG(2,N).EQ.2.0) THEN
488:          CALL BSVDEC(ENGPB,NGP2+1,
489:          & EE00, IG00, 1 )
490:          IG00 = IG00 + NGP1
491:          ELSE
492:          CALL BSVDEC(ENGYB,NGP1+1,
493:          & EE00, IG00, 1 )
494:          ENDIF
495: C
496:          DO 229 I=1,NN
497:          EEE(LSDED(IPP+I)) = -PENRG(1,N)
498:          IGG(LSDED(IPP+I)) = IG00
499: 229          CONTINUE
500: C
501:          ELSE
502:          CALL RANU2(IRAND,R,3*NN,ICON)
503:          DO 230 I=1,NN
504:          C/#IF ROUNDOFF(NEAREST)
505:          * KK = MIN(INT(NGROUP*R(I))+1,NGROUP)
506:          * KKK = min(2,int(2+R(NN+I)-PENRG(KK,N)))
507:          C/#ELSE
508:          * KK = NGROUP*R(I) + 1
509:          * KKK = 2.0+R(NN+I)-PENRG(KK,N)
510:          C/#ENDIF
511:          C/#ENDIF
512:          IGO = KENRG(KKK,KK,N)
513:          IGG(LSDED(IPP+I)) = IGO
514: C
515: C ..... FOR PHOTON ENERGY
516: C
517:          IF( IG0 .GT. NGP1 ) THEN
518:          IGOP = IG0 - NGP1
519:          EEE(LSDED(IPP+I)) = ENGPB(IGOP)
520:          & + R(2*NN+I)*

```

src/mvp/srcold.f

```

521:      &                                (ENGPB(IG0P+1)-ENGPB(IG0P))
522: C
523: C          .... FOR NEUTRON ENERGY
524: C
525: C          ELSE
526: C..... FLAT IN ENERGY WITHIN EACH GROUP
527: C          EEE(LSDED(IPP+I)) = ENGYB(IG0)
528: C      &          +R(2*NN+I)*(ENGYB(IG0+1)-ENGYB(IG0))
529: C..... 1/E IN EACH GROUP
530: C          EEE(LSDED(IPP+I)) = ENGYB(IG0)
531: C      &          *(ENGYB(IG0+1)/ENGYB(IG0))*R(2*NN+I)
532: C          ENDIF
533: 230      CONTINUE
534: C          ENDIF
535: C          ENDIF
536: C
537: C          DO 240 I=1,NN
538: C              WWW(LSDED(IPP+I)) = 1.0
539: C              TTT(LSDED(IPP+I)) = 0.0D0
540: 240      CONTINUE
541: C          ENDIF
542: C
543: C
544: C ===== SPECIAL CASES : POSITION, DIRECTION & ENERGY ARE SAMPLED
545: C          IN A SPECIAL ROUTINE.
546: C
547: C -----
548: C TYPE 9 ..... FNS NEUTRON SOURCE ( AUG. 1988 M.SASAKI ) .....
549: C   PSPAC(1:3) ... COORDINATE,      PSPAC(4) ... MAXIMUM RADIUS
550: C   PSPAC(5) ... ROTATION INDEX  PSPAC(6:8) ... ROTATION ANGLES(DEGREE)
551: C -----
552: C
553: C          IF(KS.EQ.9) THEN
554: C              DO 250 I=1,NN
555: C                  IWK1(NSS+I) = LSDED(IPP+I)
556: C                  IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I))=ISZON(1,N)
557: 250      CONTINUE
558: C
559: C              CALL FNSSOC(IRAND,NN,IGG,XXX,YYY,ZZZ,
560: C      &              AAA,BBB,CCC,WWW,TTT,R,RWK1,IWK2,ENGYB,NGP1 ,
561: C      &              IWK1(NSS+1),
562: C      &              PSPAC(5,N),PSPAC(1,N),PSPAC(2,N),PSPAC(3,N),
563: C      &              PSPAC(4,N), PSPAC(6,N),PSPAC(7,N),PSPAC(8,N))
564: C -----
565: C          SET ENERGY ( RWK1 --> EEE )
566: C -----
567: C              DO 260 I=1,NN
568: C                  EEE(LSDED(IPP+I)) = RWK1(I)
569: 260      CONTINUE
570: C          ENDIF
571: C
572: C CM1994-6-1
573: C          ELSE
574: C -----
575: C          user defined source
576: C -----
577: C
578: C          IF( KS.LT.0 ) THEN
579: C              DO 5000 I=1,NN
580: C                  IWK1(NSS+I) = LSDED(IPP+I)
581: 5000      CONTINUE
582: C              ISTIM1 = ISTIM(N)
583: C              ISTIM2 = ISTIM(N) - N + 1
584: C              ISANG1 = ISANG(N)
585: C              ISANG2 = ISANG(N) - N + 1

```

```

586:      IF(KS.EQ.-1) THEN
587:      CALL SRCU1( IOW, NN, N, IRAND, IWK1(NSS+1) ,
588:      NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
589:      XXX , YYY , ZZZ , AAA , BBB , CCC ,
590:      EEE , IGG , WWW , TTT ,
591:      KSOUR(N), PSPAC(1,N), PENRG(1,N), KENRG(1,1,N),
592:      NSTIM, STIM(ISTIM1) , PSTIM(ISTIM2),
593:      NSANG, SANG(ISANG1) , PSANG(ISANG2), SAXIS(1,N),
594:      MCX , NUC , EBOT , ETOP , EBOTP, ETOPP,
595:      CX , KLIB1, XLIB1, KLIB2, XLIB2, NMT ,
596:      X,Y,Z,IWK2,IWK3, IFI, IFL, EW1, RWK1, R )
597:      ELSE IF(KS.EQ.-2) THEN
598:      CALL SRCU2( IOW, NN, N, IRAND, IWK1(NSS+1) ,
599:      NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
600:      XXX , YYY , ZZZ , AAA , BBB , CCC ,
601:      EEE , IGG , WWW , TTT ,
602:      KSOUR(N), PSPAC(1,N), PENRG(1,N), KENRG(1,1,N),
603:      NSTIM, STIM(ISTIM1) , PSTIM(ISTIM2),
604:      NSANG, SANG(ISANG1) , PSANG(ISANG2), SAXIS(1,N),
605:      MCX , NUC , EBOT , ETOP , EBOTP, ETOPP,
606:      CX , KLIB1, XLIB1, KLIB2, XLIB2, NMT ,
607:      X,Y,Z,IWK2,IWK3, IFI, IFL, EW1, RWK1, R )
608:      ENDIF
609:      ENDIF
610: C -----
611: C
612: C      Zone determination
613: C -----
614: C -----
615: C
616: C
617: C ===== WHEN ZONE NUMBER IS SPECIFIED =====
618: C
619: C          IF(ISZON(1,N).NE.0) THEN
620: C              IZ = ISZON(1,N)
621: C
622: C          .... FREE FLIGHT STACK .....
623: C
624: C          IF(KZMAT(IZ).GE.0) THEN
625: C              DO 270 I=NFFL(NZONE+1)+1,NFFL(NZONE+1)+NN
626: C                  LSFFL(I) = IWK1(NSS+I-NFFL(NZONE+1))
627: C                  IZFFL(I) = IZ
628: C                  IZZ(LSFFL(I)) = IZ
629: C                  IREG = KZREG(IZ)
630: C 27 MAY 92 M.S. ....
631: C          ... set IBREG anytime (from Jan 2000)
632: C          IBREG(LSFFL(I)) = IREG
633: C          IF( JTLLT .NE. 0 ) THEN
634: C              IBSPC(LSFFL(I),0) = 0
635: C          ENDIF
636: C          IF( JIMPT.NE.0 .AND. IREG .NE. 0 ) THEN
637: C              XIM(LSFFL(I)) =
638: C      &              XIMP( IGG(LSFFL(I)), IREG )
639: C          ENDIF
640: C -----
641: 270      CONTINUE
642: C          NFFL(IZ) = NFFL(IZ) + NN
643: C          NFFL(NZONE+1) = NFFL(NZONE+1) + NN
644: C
645: C          .... INVALID MATERIAL ZONE ! .... (TREATED AS LOST)....
646: C
647: CCCCC      ELSE IF( KZMAT(IZ) .LE. -1000 ) THEN
648: C          ELSE IF( KZMAT(IZ).LE.-1000
649: C      &          .and..not.ISLATT(KZMAT(IZ)) ) THEN
650: C              WRITE(IOW,7000) N,KZMAT(IZ),IZ,NN

```

src/mvp/srcold.f

```

651:          ILOST      = ILOST + NN
652:          IPP        = IPP  + NN
653: C
654: C      .... LATTICE SEARCH STACK ....
655: C
656: C          ELSE
657: C              DO 280 I=NXLT(NLBZ+1)+1,NXLT(NLBZ+1)+NN
658: C                  LSLAT(I)      = IWK1(NSS+I-NXLT(NLBZ+1))
659: C                  IZLAT(I)      = MLBZZ(IZ)
660: C                  IZZ(LSLAT(I)) = IZ
661: C                  IREG          = KZREG(IZ)
662: C          ... set IBREG anytime (from Jan 2000)
663: C          IBREG(LSLAT(I)) = IREG
664: C          IF(JTLT.NE.0) THEN
665: C              IBSPC(LSLAT(I),0) = 0
666: C          ENDIF
667: C          IF( JIMPT.NE.0 ) THEN
668: C              IF( IREG.GT.0 ) THEN
669: C                  XIM(LSLAT(I)) =
670: C                  XIMP( IGG(LSLAT(I)), IREG )
671: C              ELSE
672: C                  XIM(LSLAT(I)) = 1.
673: C              ENDIF
674: C          ENDIF
675: C      280      CONTINUE
676: C              NXLT(NLBZZ(IZ)) = NXLT(NLBZZ(IZ)) + NN
677: C              NXLT(NLBZ+1)    = NXLT(NLBZ+1)    + NN
678: C          ENDIF
679: C
680: C
681: C      ==== DETERMINE STARTING ZONE NUMBERS IF NOT SPECIFIED =====
682: C
683: C
684: C          ELSE IF(ISZON(1,N).EQ.0) THEN
685: C              DO 290 I=1,NN
686: C                  IWK2(I) = LSDDED(IPP+I)
687: C                  X(I)    = XXX(IWK2(I)) + 1.0D-8*AAA(IWK2(I))
688: C                  Y(I)    = YYY(IWK2(I)) + 1.0D-8*BBB(IWK2(I))
689: C                  Z(I)    = ZZZ(IWK2(I)) + 1.0D-8*CCC(IWK2(I))
690: C      290      CONTINUE
691: C              II      = NN
692: C              KKZ     = 1
693: C              MEM      = 1
694: C              MMZ      = NZONE
695: C              IF(JLATT.NE.0) MMZ = IPCEL(1) - 1
696: C              DO 370 M=1,MMZ
697: C                  IF(JEIGN.EQ.0.OR.NBATCH.GT.0) THEN
698: C                      IF(MEM.LE.NMEMS.AND.MEMZN(MEM,N).NE.0) THEN
699: C                          IZ = MEMZN(MEM,N)
700: C                          IM = 1
701: C                      ELSE
702: C                          IM = 0
703: C                          DO 310 KZ=KKZ,MMZ
704: C                              DO 300 K=1,NMEMS
705: C                                  IF(MEMZN(K,N).NE.0.AND.
706: C                                  KZ.EQ.MEMZN(K,N)) GOTO 310
707: C                              CONTINUE
708: C                              GO TO 320
709: C      310      CONTINUE
710: C                          KZ = MMZ
711: C      320      IZ = KZ
712: C                          KKZ = IZ + 1
713: C                      ENDIF
714: C                  ELSE
715: C                      IZ = M

```

```

716:          ENDIF
717: C
718: C      .... JUDGE WHETHER PARTICLES BELONG TO ZONE IZ OR NOT. ....
719: C
720: C          CALL JUDGE('SOURCE',IZ,II,X,Y,Z,IFI,SDA,KZDA,KZAA,
721: C          &          NSDA,NZDA,NZONE+1,JVMNT, IFL,JSIMP, .FALSE. ,
722: C          &          DUMMY, DUMMY, DUMMY )
723: C
724: C      -----
725: C      ..... SEND PARTICLES TO FLIGHT STACK .....
726: C      -----
727: C
728: C          INN      = 0
729: C          IF(KZMAT(IZ).GE.0) THEN
730: C              IFFL  = NFFL(NZONE+1)
731: C              DO 330 I=1,II
732: C                  IF(IFI(I)) THEN
733: C                      IZZ(IWK2(I)) = IZ
734: C                      INN = INN + 1
735: C                      LSFFL(IFFL+INN) = IWK2(I)
736: C                      IZFFL(IFFL+INN) = IZ
737: C      27 MAY 92 M.S. ....
738: C
739: C          ... set IBREG anytime (from Jan 2000)
740: C          IBREG(IWK2(I)) = IREG
741: C          IF( JTLT.NE.0 ) THEN
742: C              IBSPC(IWK2(I),0) = 0
743: C          ENDIF
744: C      .....
745: C
746: C          IF( JIMPT.NE.0 ) THEN
747: C              XIM(IWK2(I)) =
748: C              XIMP( IGG(IWK2(I)), IREG )
749: C          ENDIF
750: C      330      CONTINUE
751: C
752: C          NFFL(IZ) = NFFL(IZ) + INN
753: C          NFFL(NZONE+1) = NFFL(NZONE+1) + INN
754: C
755: C      -----
756: C      ..... IF PARTICLES ARE IN A LATTICE, SEND THEM TO LATTICE STACK.
757: C      -----
758: C
759: C      CCCCC      ELSE IF(KZMAT(IZ).LE.-1.AND.KZMAT(IZ).GE.-998)THEN
760: C      ELSE IF( ISLATT(KZMAT(IZ)) ) THEN
761: C          ILAT = NXLT(NLBZ+1)
762: C      *VOCL LOOP,NOVREC
763: C          DO 340 I=1,II
764: C              IF(IFI(I)) THEN
765: C                  IZZ(IWK2(I)) = IZ
766: C                  INN = INN + 1
767: C                  LSLAT(ILAT+INN) = IWK2(I)
768: C                  IZLAT(ILAT+INN) = MLBZZ(IZ)
769: C
770: C      27 MAY 92 M.S. ....
771: C
772: C          IREG          = KZREG(IZ)
773: C          ... set IBREG anytime (from Jan 2000)
774: C          IBREG(IWK2(I)) = IREG
775: C          IF( JTLT.NE.0 ) THEN
776: C              IBSPC(IWK2(I),0) = 0
777: C          ENDIF
778: C          IF( JIMPT.NE.0 ) THEN
779: C              IREG          = KZREG(IZ)
780: C
781: C          IF( IREG.GT.0 ) THEN

```

```

781:                                XIM(IWK2(I)) =
782:                                & XIMP( IGG(IWK2(I)), IREG )
783:                                ELSE
784:                                XIM(IWK2(I)) = 1.
785:                                ENDIF
786:                                ENDIF
787: C
788:                                ENDIF
789: 340 CONTINUE
790:                                NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + INN
791:                                NXLT(NLBZZ+1) = ILAT + INN
792: C
793: C-----
794: C ..... GENERATING PARTICLES IN INVALID MATERIAL ? .....
795: C-----
796: C                                ELSE
797:                                DO 350 I=1,II
798:                                IF(IFI(I)) THEN
799:                                INN = INN + 1
800:                                ILOST = ILOST + 1
801:                                CM1993-9-30 LSED(IPP+I) = IWK2(I)
802:                                LSED(IPP+INN) = IWK2(I)
803:                                ENDIF
804:                                CONTINUE
805:                                IPP = IPP + INN
806: C
807: C C/#IF PARA(SX* CRAY)
808: C * call MVPSYNC_LOCK(2)
809: C/#ENDIF
810:                                IF(INN.GT.0)
811:                                & WRITE(IOW,7000) N,KZMAT(IZ),IZ,INN
812: C
813: 7000 FORMAT(/1X,' !!! CAUTION: YOU ARE GOING TO',
814:                                & ' GENERATE PARTICLES IN A ZONE WHICH ',
815:                                & ' PARTICLES SHOULD NOT EXIST !!! /1X ,
816:                                & ' SOURCE NO. ',I3,' MAT = ',I5,' ZONE = ',I5,
817:                                & ' ',I8,' PARTICLES. THEY ARE TREATED AS LOST!!'
818: C
819: C C/#IF PARA(SX* CRAY)
820: C * call MVPSYNC_UNLOCK(2)
821: C/#ENDIF
822:                                ENDIF
823: C
824: C-----
825: C ..... MEMORIZE NEWLY FOUND ZONE .....
826: C-----
827: C
828: C                                IF(JEIGN.EQ.0.OR.NBATCH.GT.0) THEN
829:                                IF(IM.NE.0) THEN
830:                                MEM = MEM + 1
831:                                ELSE IF (INN.NE.0.AND.MEM.LE.NMEMS.AND.
832:                                & MEMZN(MEM,N).EQ.0) THEN
833:                                MEMZN(MEM,N) = IZ
834:                                MEM = MEM + 1
835:                                ENDIF
836:                                ENDIF
837: C
838: C-----
839: C ..... COMPRESS .....
840: C-----
841: C
842: C                                IF(INN.LT.II) THEN
843:                                INNN = 0
844:                                *VOCL LOOP,NOVREC

```

```

846: DO 360 I=1,II
847: IF(.NOT.IFI(I)) THEN
848:     INNN = INNN + 1
849:     IWK2(INNN) = IWK2(I)
850:     X(INNN) = X(I)
851:     Y(INNN) = Y(I)
852:     Z(INNN) = Z(I)
853: ENDIF
854: 360 CONTINUE
855:     II = INNN
856: ELSE
857:     II = 0
858:     GOTO 380
859: ENDIF
860: 370 CONTINUE
861: C
862: C-----
863: C ..... ZONES NOT FOUND !! (LOST) .....
864: C-----
865: C
866: 380 CONTINUE
867: C
868: IF(II.NE.0) THEN
869: C/#IF PARA(SX* CRAY)
870: * call MVPSYNC_LOCK(2)
871: C/#ENDIF
872:
873: WRITE(IOW,7010) II,N,(XXX(IWK2(I)),YYY(IWK2(I)),
874: & ZZZ(IWK2(I)),AAA(IWK2(I)),BBB(IWK2(I)),CCC(IWK2(I)),I=1,II)
875: 7010 FORMAT(/1X,' **** ',I5,' PARTICLES ARE LOST IN SOURCE ROUTINE!'/
876: & 1X,' SOURCE NUMBER = ',I5/
877: & (1X,' X = ',1PE12.5,' Y = ',E12.5,' Z = ',E12.5,' MU = ',E12.5,
878: & ' ETA = ',1PE12.5,' XI = ',E12.5))
879:
880: C/#IF PARA(SX* CRAY)
881: * call MVPSYNC_UNLOCK(2)
882: C/#ENDIF
883: DO 390 I=1,II
884:     LSDED(IPP+I) = IWK2(I)
885: 390 CONTINUE
886:     ILOST = ILOST + II
887:     IPP = IPP + II
888: ENDIF
889: ENDIF
890: C
891: C ... save source set # if necessary as optional bank parameter
892: C
893: if ( KIBNK(1).ne.0 ) then
894:     do 392 I = 1, NN
895:         IBNK(IWK1(NSS+I),KIBNK(1)) = N
896: 392 continue
897:     end if
898: C
899:     NSS = NSS + NN
900: 400 CONTINUE
901: C
902: C-----
903: C .... UPDATE SUM OF WEIGHTS .....
904: C-----
905: C
906: DO 410 I=1,NGENE
907:     WSUM = WSUM + WWW(IWK1(I))
908: 410 CONTINUE
909: C
910: IF(JEIGN.NE.0) XSOC(NBATCH+1) = WSUM

```

src/mvp/srcold.f

```

911: C
912: IF( JNEUT.NE.0 .AND. JPHOT.NE.0 ) THEN
913: DO 412 I=1,NGENE
914: IF( IGG(IWK1(I)).LE.NGP1 ) THEN
915: NCNTR(1,KNUTRN) = NCNTR(1,KNUTRN) + 1
916: WCNTR(1,KNUTRN) = WCNTR(1,KNUTRN) + WWW(IWK1(I))
917: ELSE
918: NCNTR(1,KPHOTN) = NCNTR(1,KPHOTN) + 1
919: WCNTR(1,KPHOTN) = WCNTR(1,KPHOTN) + WWW(IWK1(I))
920: ENDIF
921: 412 CONTINUE
922: ELSE IF( JNEUT.NE.0 ) THEN
923: DO 1412 I=1,NGENE
924: WCNTR(1,KNUTRN) = WCNTR(1,KNUTRN) + WWW(IWK1(I))
925: 1412 CONTINUE
926: NCNTR(1,KNUTRN) = NCNTR(1,KNUTRN) + NGENE
927: ELSE IF( JPHOT.NE.0 ) THEN
928: DO 2412 I=1,NGENE
929: WCNTR(1,KPHOTN) = WCNTR(1,KPHOTN) + WWW(IWK1(I))
930: 2412 CONTINUE
931: NCNTR(1,KPHOTN) = NCNTR(1,KPHOTN) + NGENE
932: ENDIF
933: C
934: C
935: C =====
936: C SOURCES FROM FISSION NEUTRON BANK FOR EIGENVALUE PROBLEM
937: C =====
938: C
939: C
940: ELSE
941: C
942: CALL FISSRC( IOW, IRAND, NFISB, NGENE, NBANK, NUC, NZONE,
943: & NFBANK,NFBNK0,NBATCH,NGROUP,NGP1,NREG,NEST,NLENG, WSUM,
944: & KZREG, WGTFF, EBOT, ETOP, ENGYB,
945: & XIMP, XSOC, NCNTR, WCNTR, NEVENT,
946: & CX, ICX, MCX, NMT, KLIB1, XLIB1, KLIB2, XLIB2,
947: & NFFFL, LSFFFL, IZFFFL, NDEAD, LSDDED,
948: & IWK1,
949: & XXX, YYY, ZZZ, AAA, BBB, CCC, EEE, WWW, IGG, IZZ, TTT, XIM,
950: Q IBREG, IBSPC, KLSF,
951: & XXXF, YYF, ZZZF, EEEF, INUF, IZZF,
952: & LEVL, LEVLF, LZZ, LZZF, LPOS, LPOSF, LCRS, LCRSF,
953: Q IBRGF, IBSPF, KLSFF,
954: & R, IWK2,IWK3, IFL,IFI, X, Y, RWK1, EW1 )
955: CM1993-9-30
956: ILOST = 0
957: C
958: ENDIF
959: C
960: C ... treatment for optional bank parameters ....
961: C
962: C ... weight on birth
963: C
964: if ( KDBNK(1).ne.0 ) then
965: do 461 I = 1, NGENE
966: DBNK(IWK1(I),KDBNK(1)) = WWW(IWK1(I))
967: 461 continue
968: end if
969: C
970: C ... time on birth
971: C
972: if ( KDBNK(2).ne.0 ) then
973: do 471 I = 1, NGENE
974: DBNK(IWK1(I),KDBNK(2)) = TTT(IWK1(I))
975: 471 continue

```

```

976: end if
977: C
978: C ... energy on birth
979: C
980: if ( KDBNK(3).ne.0 ) then
981: do 481 I = 1, NGENE
982: DBNK(IWK1(I),KDBNK(3)) = EEE(IWK1(I))
983: 481 continue
984: end if
985: C
986: C ... region # on birth (but the region # may not be the real one
987: C in lattice geometry ...)
988: if ( KIBNK(2).ne.0 ) then
989: do 491 I = 1, NGENE
990: IBNK(IWK1(I),KIBNK(2)) = IBREG(IWK1(I))
991: 491 continue
992: end if
993: C
994: C ... zone # on birth (but the region # may not be the real one
995: C in lattice geometry ...)
996: if ( KIBNK(3).ne.0 ) then
997: do 501 I = 1, NGENE
998: IBNK(IWK1(I),KIBNK(3)) = IZZ(IWK1(I))
999: 501 continue
1000: end if
1001: C
1002: C ... generation of particle (1 for primary particles)
1003: C
1004: if ( KIBNK(4).ne.0 ) then
1005: do 511 I = 1, NGENE
1006: IBNK(IWK1(I),KIBNK(4)) = 1
1007: 511 continue
1008: end if
1009: C
1010: C ... Marker region flag
1011: C
1012: if ( KIBNK(9).ne.0 ) then
1013: do 514 J = 1, NMKREG
1014: KI9 = KIBNK(9)+J-1
1015: do 512 I = 1, NGENE
1016: IBNK(IWK1(I),KI9) = 0
1017: 512 continue
1018: 514 continue
1019: end if
1020: C
1021: C .... PARTICLES IN THE REST OF BANK ARE DEAD PARTICLES ! .....
1022: C
1023: NDEAD = NDEAD - NGENE + ILOST
1024: NLOST = NLOST + ILOST
1025: C* NDEAD = NBANK - NGENE
1026: C* DO 400 I=1,NDEAD
1027: C 400 LSDDED(I) = NGENE + I
1028: C
1029: C .... CALCULATE & PRINT AVERAGED PARAMETERS .....
1030: C XAV = 0.0
1031: C YAV = 0.0
1032: C ZAV = 0.0
1033: C AAV = 0.0
1034: C BAV = 0.0
1035: C CAV = 0.0
1036: C EAV = 0.0
1037: DO 530 I=1,NGENE
1038: XAV = XAV + XXX(IWK1(I)) * WWW(IWK1(I))
1039: YAV = YAV + YYY(IWK1(I)) * WWW(IWK1(I))
1040: ZAV = ZAV + ZZZ(IWK1(I)) * WWW(IWK1(I))

```


src/mvp/srcold.f

```

1041:      AAV = AAV + AAA(IWK1(I)) * WWW(IWK1(I))
1042:      BAV = BAV + BBB(IWK1(I)) * WWW(IWK1(I))
1043:      CAV = CAV + CCC(IWK1(I)) * WWW(IWK1(I))
1044:      CCCCCCCC EAV = EAV + FLOAT(IGG(IWK1(I))) * WWW(IWK1(I))
1045:      EAV = EAV + EEE(IWK1(I)) * WWW(IWK1(I))
1046:      530 CONTINUE
1047:      XAVT(1) = XAVT(1) + XAV
1048:      XAVT(2) = XAVT(2) + YAV
1049:      XAVT(3) = XAVT(3) + ZAV
1050:      AAVT(1) = AAVT(1) + AAV
1051:      AAVT(2) = AAVT(2) + BAV
1052:      AAVT(3) = AAVT(3) + CAV
1053:      EAVT = EAVT + EAV
1054:      NTGEN = NTGEN + NGENE
1055: C
1056:      XAV = XAV/NGENE
1057:      YAV = YAV/NGENE
1058:      ZAV = ZAV/NGENE
1059:      IF(JPRTS(6).NE.1) THEN
1060: C/#IF PARA(SX* CRAY)
1061:      * call MVPSYNC_LOCK(2)
1062: C/#ENDIF
1063:      if ( JBPRNT.ne.0 ) then
1064:      WRITE(IOW,7030) '(BATCH) ',XAV,YAV,ZAV,
1065:      & AAV/NGENE ,BAV/NGENE ,CAV/NGENE ,EAV/NGENE
1066:      WRITE(IOW,7030) '(CUMULATIVE)',XAVT(1)/NTGEN,XAVT(2)/NTGEN,
1067:      & XAVT(3)/NTGEN,
1068:      & AAVT(1)/NTGEN,AAVT(2)/NTGEN,AAVT(3)/NTGEN,EAVT/NTGEN
1069:      IF(JPRTS(6).NE.1)
1070:      & WRITE(IOW,*) ' GENERATED PARTICLES IN THIS RUN = ',NTGEN
1071:      end if
1072: C/#IF PARA(SX* CRAY)
1073:      * call MVPSYNC_UNLOCK(2)
1074: C/#ENDIF
1075:      ENDIF
1076:      7030 FORMAT(1X , ' AVERAGE ',A12,2X, ' X= ',1PE11.4, ' Y= ',E11.4, ' Z= ',
1077:      & E11.4, ' MU= ',E11.4, ' ETA= ',E11.4, ' XI= ',E11.4, ' E = ',
1078:      & E11.4)
1079: C
1080: C IF(JDEBG.EQ.1) THEN
1081: C WRITE(6,7030) (I,XXX(I),YYY(I),ZZZ(I),AAA(I),BBB(I),CCC(I))
1082: C & WWW(I),IZZ(I),IGG(I),TTT(I),I=1,NGENE)
1083: C 7030 FORMAT(/1X, ' GENERATED SOURCE PARAMETERS ' /
1084: C & 1X , ' XXX YYY ZZZ AAA BBB ' /
1085: C & ' CCC WWW IZZ IGG TTT ' /
1086: C & (1X ,I6,1P6D11.4,E11.4,2I7,E11.4))
1087: C ENDIF
1088: CC
1089: IF(JEIGN.NE.0) THEN
1090:      XVA = 0.0
1091:      YVA = 0.0
1092:      ZVA = 0.0
1093:      DO 540 I=1,NGENE
1094:      XVA=XVA+(XXX(IWK1(I))-XAV)*(XXX(IWK1(I))-XAV)*WWW(IWK1(I))
1095:      YVA=YVA+(YYY(IWK1(I))-YAV)*(YYY(IWK1(I))-YAV)*WWW(IWK1(I))
1096:      ZVA=ZVA+(ZZZ(IWK1(I))-ZAV)*(ZZZ(IWK1(I))-ZAV)*WWW(IWK1(I))
1097:      540 CONTINUE
1098:      XSXV(NBATCH+1,1) = XVA/(NGENE-1)
1099:      XSXV(NBATCH+1,2) = YVA/(NGENE-1)
1100:      XSXV(NBATCH+1,3) = ZVA/(NGENE-1)
1101: C/#IF PARA(SX* CRAY)
1102:      * call MVPSYNC_LOCK(2)
1103: C/#ENDIF
1104:      if ( JBPRNT.ne.0 ) then
1105:      IF(JPRTS(6).NE.1) then

```

```

1106:      & WRITE(IOW,7040) '(BATCH) ',XSXV(NBATCH+1,1),
1107:      & XSXV(NBATCH+1,2), XSXV(NBATCH+1,3)
1108:      end if
1109:      end if
1110:      7040 FORMAT(1X , ' VARIANCE ',A10,2X, ' X= ',1PE11.4, ' Y= ',E11.4,
1111:      & ' Z= ',E11.4)
1112: C/#IF PARA(SX* CRAY)
1113:      * call MVPSYNC_UNLOCK(2)
1114: C/#ENDIF
1115:      ENDIF
1116: C
1117: C IF(JPRTS(6).NE.1)
1118: C & WRITE(IOW,*) ' GENERATED PARTICLES IN THIS RUN = ',NTGEN
1119: C
1120: C-----
1121: C MAKE MICRO CROSS SECTIONS FOR EACH PARTICLE
1122: C-----
1123: C
1124:      IF( JNEUT.NE. 0 .AND. JPHOT.NE.0 ) THEN
1125:      INEUT = 0
1126:      DO 550 I=1,NGENE
1127:      IGII = IGG(IWK1(I))
1128:      IF( IGII .LE. NGP1 ) THEN
1129:      INEUT = INEUT + 1
1130:      IWK2(INEUT) = IWK1(I)
1131:      ENDIF
1132:      550 CONTINUE
1133:      IPHOT = INEUT
1134:      DO 560 I=1,NGENE
1135:      IGII = IGG(IWK1(I))
1136:      IF( IGII .GT. NGP1 ) THEN
1137:      IPHOT = IPHOT + 1
1138:      IWK2(IPHOT) = IWK1(I)
1139:      ENDIF
1140:      560 CONTINUE
1141:      DO 570 I=1,NGENE
1142:      IWK1(I) = IWK2(I)
1143:      570 CONTINUE
1144: C
1145:      ELSE
1146:      IF(JNEUT.NE.0) INEUT = NGENE
1147:      IF(JPHOT.NE.0) INEUT = 0
1148:      ENDIF
1149: C
1150: C ..... NEUTRON MICRO CROSS SECTION .....
1151: C
1152:      IF(JNEUT.NE.0) THEN
1153:      IF(JVMNT.NE.0) CALL VMNT00(8,1)
1154:      IF(JVMNT.NE.0) CALL VMNTR1(8,INEUT)
1155:      CALL GETMIC( INEUT, IWK1, IRAND, JEIGN, NUC, NMT, NBANK, NSMIC,
1156:      B EEE, SMIC, LMIC, KSPI,
1157:      X CX,ICX,MCX, KL1B1, KL1B2, XL1B1,
1158:      W EW1,RWK1, IWK2, IWK3 ,IFI ,IFL, R(1),R(NBANK+1),
1159:      W SMCW)
1160: C
1161: CC X SGTAL, CRES, KCRES, NSTAL, MCRES,
1162: C
1163:      IF(JVMNT.NE.0) CALL VMNT22(8,1)
1164:      ENDIF
1165: C
1166: C
1167: C ..... PHOTON MICRO CROSS SECTION .....
1168: C
1169:      IF(JPHOT.NE.0) THEN
1170:      NN = NGENE - INEUT

```

src/mvp/srcold.f

```
1171:          IF( NN.GT.0 ) THEN
1172:              IF(JVMNT.NE.0) CALL VMNT00(8,1)
1173:              IF(JVMNT.NE.0) CALL VMNTR1(8,NN)
1174:              CALL GTMICP( JGAMM, NN,IWK1(INEUT+1), NUC,NMTP,NPATOM,
1175:                  N          NBANK,NSMIC,
1176:                  B          EEE, SMIC, KSPI,
1177:                  X          CXP, ICXP, MCXP,  KLBP1, KLBP2, XLBP2,
1178:                  W          EW1,IWK2, R, IWK3 )
1179:
1180: CCC  X          SGTAL, CRES, KCRES, NSTAL, MCRES,
1181:          IF(JVMNT.NE.0) CALL VMNT22(8,1)
1182:      ENDIF
1183:  ENDIF
1184: C
1185:      if ( NSTAL.gt.0 ) then
1186:          if ( INEUT.gt.0 ) then
1187:              JNP      = 1
1188:              call GTSGTL( JNP, INEUT, IWK1, NBANK, EEE,
1189:                  &      SGTAL, CRES, KCRES, NSTAL, MCRES, EW1, IWK2 )
1190:          end if
1191:          if ( NGENE.gt.INEUT ) then
1192:              JNP      = 2
1193:              IPHOT     = NGENE - INEUT
1194:              call GTSGTL( JNP, IPHOT, IWK1(INEUT+1), NBANK, EEE,
1195:                  &      SGTAL, CRES, KCRES, NSTAL, MCRES, EW1, IWK2 )
1196:          end if
1197:      end if
1198: C
1199: C      .... NO MACRO CROSS SECTION PREPARED HERE ....
1200: C
1201:      DO 580 I=1, NGENE
1202:          MMAC(IWK1(I),2) = 0
1203:      580 CONTINUE
1204: C
1205:      if ( JTIME.ne.0 ) then
1206:          DO 590 I=1, NGENE
1207:              ITT(IWK1(I)) = 1
1208:      590 continue
1209:      endif
1210: C
1211: CC      IF(JVMNT.NE.0) CALL VMNTR2('SOURCE')
1212:      RETURN
1213:
1214: C/#ENDIF
1215:
1216:      END
```

src/mvp/srcul.f

```

1:      SUBROUTINE SRCU1
2:      N ( IOW , NN , ID , IRAND, IBP ,
3:      N   NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
4:      B   XXX , YYY , ZZZ , AAA , BBB , CCC ,
5:      B   EEE , IGG , WWW , TTT ,
6:      S   KSOUR, PSPAC, PENRG, KENRG,
7:      S   NSTIM, STIM , PSTIM,
8:      S   NSANG, SANG , PSANG, SAXIS,
9:      X   MCX , NUC , EBOT , ETOP , ETOPP,
10:     X   CX , KLIB1, KLIB2, KLIB2, NMT ,
11:     W   X,Y,Z,IWK1,IWK2,IWK3,IWK4, WK1, WK2, R )
12: C
13: C=<MVP>=====
14: C PURPOSE: GENERATE PARTICLES (USER SOURCE ROUTINE #1 FOR MVP)
15: C CALLED IN: SOURCE (KSOUR=1)
16: C-----
17: C VARIABLES      TYPE      MEANINGS
18: C IOW            I4        LOGICAL UNIT FOR PRINTOUT
19: C NN             I4        NUMBER OF PARTICLES BEING GENERATED FROM NOW
20: C ID             I4        ID NUMBER OF THIS SOURCE ( 1 <= ID <= NSOUR )
21: C IRAND          I4        INITIAL RANDOM NUMBER
22: C IBP(NN)        I4        BANK POINTER FOR PARTICLES TO BE GENERATED
23: C               I4        DESCRIPTERS FOR NTH PARTICLE ARE STORED IN
24: C               I4        IBP(N)'TH POSITION IN THE BANK.
25: C NGROUP         I4        NUMBER OF ENERGY GROUPS FOR TALLY AND SOURCE
26: C NGP1           I4        NUMBER OF ENERGY GROUPS OF NEUTRONS
27: C NGP2           I4        NUMBER OF ENERGY GROUPS OF PHOTONS
28: C NBANK          I4        SIZE OF PARTICLE BANK
29: C-----
30: C-----< PARTICLE BANK >: THESE ARRAYS SHOULD BE ASSIGNED VALUES. -----
31: C XXX(NBANK)     R8        X-POSITION (CM)
32: C YYY(NBANK)     R8        Y-POSITION (CM)
33: C ZZZ(NBANK)     R8        Z-POSITION (CM)
34: C AAA(NBANK)     R8        X-DIRECTION COSINE
35: C BBB(NBANK)     R8        Y-DIRECTION COSINE
36: C CCC(NBANK)     R8        Z-DIRECTION COSINE
37: C WWW(NBANK)     R4        PARTICLE WEIGHT
38: C EEE(NBANK)     R4        ENERGY (EV)
39: C IGG(NBANK)     I4        ENERGY GROUP
40: C TTT(NBANK)     R8        TIME (SEC) MEANINGLESS IN THE CODE.
41: C-----< SOURCE DATA >-----
42: C KSOUR          I4        TYPE OF THE SOURCE.
43: C PSPAC(10)      R4        SPATIAL DISTRIBUTION. (INPUT VALUES)
44: C
45: C PENRG(NGROUP)  R4        ENERGY DISTRIBUTIONS (MODIFIED DISCRETE SAMPLING)
46: C KENRG(2,NGROUP) I4        ENERGY BIN # (FOR DISCRETE SAMPLING)
47: C
48: C NSTIM          I4        TIME BIN NUMBER OF SOURCE. (INPUT VALUE)
49: C STIM(NSTIM+1)  R4        TIME BINS OF SOURCE. (INPUT VALUES)
50: C PSTIM(NSTIM)   R4        TIME DISTRIBUTION. (INPUT VALUES)
51: C
52: C NSANG          I4        ANGLE NUMBERS OF SOURCE. (INPUT VALUES)
53: C SANG(NSANG+1)  R4        ANGLES (COSINE) BINS OF SOURCE. (INPUT VALUES)
54: C PSANG(NSANG)   R4        ANGULAR DISTRIBUTION. (INPUT VALUES)
55: C SAXIS(3)       R4        DIRECTION COSINES TO SOURCE GENERATION AXES.
56: C*
57: C=====
58: C
59: C .... BANK POINTER .....
60: C
61:      INTEGER      IBP(NN)
62: C
63: C .... PARTICLE BANK .....
64: C
65:      REAL*8        XXX(NBANK) , ZZZ(NBANK) , YYY(NBANK) , AAA(NBANK) ,

```

```

66:      @            BBB(NBANK) , CCC(NBANK)
67:      REAL          WWW(NBANK) , EEE(NBANK)
68:      real*8        TTT(NBANK)
69:      INTEGER       IGG(NBANK)
70: C
71: C .... SOURCE DATA .....
72: C
73:      INTEGER       KSOUR, KENRG(2,NGROUP) , NSTIM, NSANG
74:      REAL          PSPAC(10) , PENRG(NGROUP) , ENGYB(NGP1+1) , ENGPB(NGP2+1) ,
75:      @            STIM(NSTIM+1) , PSTIM(NSTIM) ,
76:      @            SANG(NSANG+1) , PSANG(NSANG) , SAXIS(3)
77: C
78: C CROSS SECTION ..... (NEUTRON)
79: C
80:      REAL          CX(MCX)
81:      REAL          XLIB1(NUC,11) , XLIB2(NUC,NMT,4)
82: C##<2007/03/14:PN3:
83: C##              INTEGER       KLIB1(NUC,27) , KLIB2(NUC,NMT,21)
84: C##              INTEGER       KLIB1(NUC,31) , KLIB2(NUC,NMT,21)
85: C##>
86: C
87:      REAL          EBOT,ETOP,ETOPP,EBOTP
88: C
89: C .... WORKING ARRAY FOR
90: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 4*NBANK FOR USE
91:      REAL          R(5*NBANK) , WK1(NBANK) , WK2(NBANK)
92:      INTEGER       IWK1(NBANK) , IWK2(NBANK) , IWK3(NBANK) , IWK4(NBANK)
93:      REAL*8        X(NBANK) , Y(NBANK) , Z(NBANK)
94: C
95: C
96: C
97:      WRITE(IOW,*) ' '
98:      WRITE(IOW,*)
99:      @ ' XXX YOU ARE USING USER SOURCE ROUTINE #1 (SRC1) XXX'
100:      WRITE(IOW,*)
101:      @ ' XXX THIS JOB HAS BEEN TERMINATED BY THE CODE XXX'
102:      WRITE(IOW,*)
103:      @ ' XXX PLEASE REPLACE SRCU1 WITH YOURS XXX'
104: C
105:      STOP 888
106: C
107: C=====
108: CC      EXAMPLE : SAMPLING OF ENERGY FROM FISSION SPECTRUM OF
109: CC      IWK1(1:NN) : NUCLIDE #
110: CC      WK1(1:NN)  : INCIDENT NEUTRON ENERGY (EV)
111: CC      WK2(1:NN)  : FISSION NEUTRON ENERGY (EV)
112: C-----
113: C      CALL SEP5N2(
114: C      @ IOW , IWK1 , NN , WK1 , WK2 ,
115: C      @ CX,CX, MCX , KLIB1, XLIB1, KLIB2, XLIB2,
116: C      @ NBANK, NUC , NMT , IRAND,
117: C      W R(1) ,R(NN+1), R(2*NN+1), R(3*NN+1), R(4*NN+1),
118: C      W IWK2 , IWK3 , IWK4 , X , Y )
119: C-----
120: CC      EW1 MUST SATISFY EBOT =< WK2 <= ETOP
121: C-----
122: C      DO 100 I=1,NN
123: C          EW2(I) = MAX(EBOT, MIN(WK2(I), ETOP))
124: C 100 CONTINUE
125: C-----
126: CC      STORE ENERGY EEE AND DETERMINE ENERGY GROUP # IGG
127: C-----
128: C      CALL BSVDEC(ENGYB,NGP1+1,EW2, IWK2, NN)
129: C      DO 200 I=1,NN
130: C          EEE(IBP(I)) = EW2(I)

```

src/mvp/srcul.f

```
131: C          IGG(IBP(I)) = IWK2(I)
132: C 200 CONTINUE
133: C=====
134: C
135: C          SAMPLING FROM PENRG : EXAMPLE OF DISCRETE CONDITIONAL SAMPLING
136: C-----
137: C
138: CC          SAMPLING OF ENERGY GROUP # BY DISCRETE CONDITIONAL SAMPLING
139: C          CALL RANU2(IRAND,R,3*NN,ICON)
140: C          DO 300 I=1,NN
141: C              KK          = NGROUP*R(I) + 1
142: C              KKK         = 2.0+R(NN-I)-PENRG(KK)
143: C              IGO         = KENRG(KKK,KK)
144: C              IGG(IBP(I)) = IGO
145: C
146: CC..... FOR PHOTON ENERGY
147: C
148: C          IF( IGO .GT. NGP1 ) THEN
149: C              IGOP = IGO - NGP1
150: C          ... FLAT IN ENERGY WITHIN EACH GROUP
151: C              EEE(IBP(I)) = ENGPB(IGOP)
152: C          @          + R(2*NN+I)*(ENGPB(IGOP+1)-ENGPB(IGOP))
153: C
154: C..... FOR NEUTRON ENERGY
155: C
156: C          ELSE
157: C              ... 1/E IN EACH GROUP
158: C              EEE(IBP(I)) = ENGYB(IGO)
159: C          @          * (ENGYB(IGO+1)/ENGYB(IGO))*R(2*NN+I)
160: C          ENDIF
161: C 300 CONTINUE
162: C
163: CC---- SET DEFAULT VALUES ( ISOTROPIC SOURCE AT THE ORIGIN )
164: C
165: C          CALL RANU2(IRAND,R,2*NN,ICON)
166: C          DO 500 I=1,NN
167: C              XXX(IBP(I)) = 0.0
168: C              YYY(IBP(I)) = 0.0
169: C              ZZZ(IBP(I)) = 0.0
170: C              AAA(IBP(I)) = 1.0 - 2.0*R(I)
171: C              AA          = SQRT( 1.0D0 - AAA(IBP(I))**2 )
172: C              FAI         = 6.2831853D0 * R(NN+I)
173: C              BBB(IBP(I)) = AA * COS( FAI )
174: C              CCC(IBP(I)) = AA * SIN( FAI )
175: C 500 CONTINUE
176: CC
177: C          DO 600 I=1,NN
178: C              WWW(IBP(I)) = 1.0
179: C              TTT(IBP(I)) = 0.0
180: C 600 CONTINUE
181: CC
182: C          RETURN
183: C          END
```

src/mvp/srcu2.f

```

1:      SUBROUTINE SRCU2
2:      N ( IOW , NN , ID , IRAND, IBP ,
3:      N   NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
4:      B   XXX , YYY , ZZZ , AAA , BBB , CCC ,
5:      B   EEE , IGG , WWW , TTT ,
6:      S   KSOUR, PSPAC, PENRG, KENRG,
7:      S   NSTIM, STIM , PSTIM,
8:      S   NSANG, SANG , PSANG, SAXIS,
9:      X   MCX , NUC , EBOT , ETOP , ETOPP,
10:     X   CX , KLIB1, XLIB1, KLIB2, XLIB2, NMT ,
11:     W   X,Y,Z,IWK1,IWK2,IWK3,IWK4, WK1, WK2, R )
12: C
13: C=<MVP>=====
14: C PURPOSE: GENERATE PARTICLES (USER SOURCE ROUTINE #2 FOR MVP)
15: C CALLED IN: SOURCE (KSOUR=-2)
16: C-----
17: C VARIABLES      TYPE      MEANINGS
18: C IOW            I4        LOGICAL UNIT FOR PRINTOUT
19: C NN             I4        NUMBER OF PARTICLES BEING GENERATED FROM NOW
20: C ID             I4        ID NUMBER OF THIS SOURCE ( 1 <= ID <= NSOUR )
21: C IRAND          I4        INITIAL RANDOM NUMBER
22: C IBP(NN)        I4        BANK POINTER FOR PARTICLES TO BE GENERATED
23: C                I4        DESCRIPTERS FOR NTH PARTICLE ARE STORED IN
24: C                I4        IBP(N)'TH POSITION IN THE BANK.
25: C NGROUP          I4        NUMBER OF ENERGY GROUPS FOR TALLY AND SOURCE
26: C NGP1            I4        NUMBER OF ENERGY GROUPS OF NEUTRONS
27: C NGP2            I4        NUMBER OF ENERGY GROUPS OF PHOTONS
28: C NBANK           I4        SIZE OF PARTICLE BANK
29: C-----
30: C-----< PARTICLE BANK >: THESE ARRAYS SHOULD BE ASSIGNED VALUES. -----
31: C XXX(NBANK)      R8        X-POSITION (CM)
32: C YYY(NBANK)      R8        Y-POSITION (CM)
33: C ZZZ(NBANK)      R8        Z-POSITION (CM)
34: C AAA(NBANK)      R8        X-DIRECTION COSINE
35: C BBB(NBANK)      R8        Y-DIRECTION COSINE
36: C CCC(NBANK)      R8        Z-DIRECTION COSINE
37: C WWW(NBANK)      R4        PARTICLE WEIGHT
38: C EEE(NBANK)      R4        ENERGY (EV)
39: C IGG(NBANK)      I4        ENERGY GROUP
40: C TTT(NBANK)      R8        TIME (SEC) MEANINGLESS IN THE CODE.
41: C-----< SOURCE DATA >-----
42: C KSOUR           I4        TYPE OF THE SOURCE.
43: C PSPAC(10)       R4        SPATIAL DISTRIBUTION. (INPUT VALUES)
44: C
45: C PENRG(NGROUP)   R4        ENERGY DISTRIBUTIONS (MODIFIED DISCRETE SAMPLING)
46: C KENRG(2,NGROUP) I4        ENERGY BIN # (FOR DISCRETE SAMPLING)
47: C
48: C NSTIM           I4        TIME BIN NUMBER OF SOURCE. (INPUT VALUE)
49: C STIM(NSTIM+1)   R4        TIME BINS OF SOURCE. (INPUT VALUES)
50: C PSTIM(NSTIM)    R4        TIME DISTRIBUTION. (INPUT VALUES)
51: C
52: C NSANG           I4        ANGLE NUMBERS OF SOURCE. (INPUT VALUES)
53: C SANG(NSANG+1)   R4        ANGLES (COSINE) BINS OF SOURCE. (INPUT VALUES)
54: C PSANG(NSANG)    R4        ANGULAR DISTRIBUTION. (INPUT VALUES)
55: C SAXIS(3)        R4        DIRECTION COSINES TO SOURCE GENERATION AXES.
56: C*
57: C=====
58: C
59: C .... BANK POINTER .....
60: C
61:      INTEGER      IBP(NN)
62: C
63: C .... PARTICLE BANK .....
64: C
65:      REAL*8        XXX(NBANK) , ZZZ(NBANK) , YYY(NBANK) , AAA(NBANK) ,

```

```

66:      @            BBB(NBANK) , CCC(NBANK)
67:      REAL          WWW(NBANK) , EEE(NBANK)
68:      real*8        TTT(NBANK)
69:      INTEGER       IGG(NBANK)
70: C
71: C .... SOURCE DATA .....
72: C
73:      INTEGER       KSOUR, KENRG(2,NGROUP) , NSTIM, NSANG
74:      REAL          PSPAC(10) , PENRG(NGROUP) , ENGYB(NGP1+1) , ENGPB(NGP2+1) ,
75:      @            STIM(NSTIM+1) , PSTIM(NSTIM) ,
76:      @            SANG(NSANG+1) , PSANG(NSANG) , SAXIS(3)
77: C
78: C CROSS SECTION ..... (NEUTRON)
79: C
80:      REAL          CX(MCX)
81:      REAL          XLIB1(NUC,11) , XLIB2(NUC,NMT,4)
82: C##<2007/03/14:PN3:
83: C##              INTEGER       KLIB1(NUC,27) , KLIB2(NUC,NMT,21)
84: C##              INTEGER       KLIB1(NUC,31) , KLIB2(NUC,NMT,21)
85: C##>
86: C
87:      REAL          EBOT,ETOP,ETOPP,EBOTP
88: C
89: C .... WORKING ARRAY FOR
90: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 4*NBANK FOR USE
91:      REAL          R(5*NBANK) , WK1(NBANK) , WK2(NBANK)
92:      INTEGER       IWK1(NBANK) , IWK2(NBANK) , IWK3(NBANK) , IWK4(NBANK)
93:      REAL*8        X(NBANK) , Y(NBANK) , Z(NBANK)
94: C
95: C
96: C
97:      WRITE(IOW,*) ' '
98:      WRITE(IOW,*)
99:      @ ' XXX YOU ARE USING USER SOURCE ROUTINE #2 (SRC2) XXX'
100:      WRITE(IOW,*)
101:      @ ' XXX THIS JOB HAS BEEN TERMINATED BY THE CODE XXX'
102:      WRITE(IOW,*)
103:      @ ' XXX PLEASE REPLACE SRCU2 WITH YOURS XXX'
104: C
105:      STOP 888
106: C
107: C=====
108: C
109:      RETURN
110:      END

```

src/mvp/staln1.f

```

1:      subroutine STALN1( IOW, JTIME, JPTIM, MZONE, IDTALY,
2:      I          FLX, NN, IZN,
3:      T          IEG, IRG, IBP, IMB, ITP, TIM, TI,
4:      N          NGROUP, NREG, NRESP, NUC, NBANK, NSMIC, NSMAC,
5:      N          NSTAL, MB, DNZON, NZONE, NTIME, NEMIC,
6:      D          RESP, TIMEB, NMKREG, MKREG,
7:      R          DTALY,
8:      X          SMIC, LMIC, SMAC, LMAC, SGTAL, TCUT,
9:      B          DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
10:     W          DNF, IPP, IPMK, JPTIM2,
11:     &          EEE, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, CXPN, ! ph-nu
c-3
12:     &          MCXPN, INCSTPN, MTMPN, NMTMPN, KLBPN1, KLBPN2, ! ph-nu
c-3
13:     &          XLBPN1, EBOTLPN, NUCPNA, MNEUTPN, DNSTPN, ! ph-nu
c-3
14:     &          MNUCPN, LPIDPN, DE, IEP, SMCN, E ) ! ph-nu
c-3
15: C=<MVP>=====
16: C PURPOSE: Take special tallies for track length or collision tally
17: C          (neutron/photon & multiple time)
18: C
19: C (can treat neutron & photon, but cannot treat both particles at once)
20: C
21: C CALLED IN: FLIONE, NEUTR, PHOTR
22: C=====
23: C
24: C arguments ( i = input, o=output, w=work )
25: C
26: C i IOW : message printout I/O unit
27: C i JTIME : problem is time dependent if non zero.
28: C i JPTIM : periodic time is used if non zero.
29: C i MZONE : zone # if all particles have the same zone #.
30: C          when MZONE = 0, array IZN(*) holds zone # for each particle
31: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
32: C          routine.
33: C i FLX(NN) : flux or contribution to tally
34: C i NN : number of particles
35: C i IZN(NN) : zone # (used when MZONE=0 and tally requires zone#)
36: C i IEG(NN) : energy group #
37: C i IRG(NN) : region #
38: C i IBP(NN) : bank pointers for each particle (necessary only when
39: C          JREAC=1 or JEIGN.ne.0)
40: C i IMB(NN) : value of MMAC(IBP,1) for each particle
41: C i ITP(NN) : current time bin of each particle
42: C i TIM(NN) : current time
43: C i TI(NN) : flight time tallied in track length tally
44: C          used only for track length tally (IDTALY(4)=2)
45: C o DTALY(*) : tally
46: C
47: C w DNF(NN) : working area
48: C w IPP(NN) : working area
49: C w IPMK(NN) : working area
50: C i/o JPTIM2 : set unity in this routine when periodic time is applied
51: C-----
52: C
53: C Structure of IDTAL(*)
54: C
55: C +----- direct tally loop ( NDTALY )
56: C |
57: C | (1) ID
58: C | (2) (dummy for debug: -999)
59: C | (3) pointer to DTALY(*)
60: C | (4) event #
61: C | (5) particle type
62: C | (6) number of dimensions
63: C | (7) multiplication factor type
64: C | (8) nuclide/atom # & reaction #
65: C | (9) direct tally #
66: C | (10) custom tally # (non-zero means user customized treatment)
67: C | (11) detector # (for surface event positive tally-surface #
68: C |          means particle current tally, and negative one
69: C |          means flux tally)
70: C | ( IDTOFF data so far )
71: C
72: C +----- tally dimension loop
73: C | (1) dimension type
74: C | (2) number of tallied bins
75: C | (3) size of common bin -> tallied bin conversion table
76: C | ( IDMOFF data so far )
77: C | +----- common bin loop
78: C | | tallied bin # for each common bin
79: C | +-----
80: C | +-----
81: C | +-----
82: C=====
83: C          implicit real*8(A-H,O-Z)
84: C
85: C          include 'INC/_KPIDS'
86: C          include 'INC/_NGPS'
87: C
88: C          integer IDTALY(*)
89: C
90: C          real*8 FLX(NN)
91: C          real EEE(NN) ! ph-nuc-3
92: C          integer IZN(NN), IEG(NN), IBP(NN), IMB(NN), IRG(NN)
93: C          integer ITP(NN)
94: C
95: C          real*8 TIM(NN), TI(NN)
96: C
97: C          real*8 DTALY(*)
98: C
99: C          real CXPN(MCXPN), XLBPN1(NUCPNI,6), EBOTLPN ! ph-nuc-3
100: C          integer KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13) ! ph-nuc-3
101: C          real DNSTPN(NUCPNA) ! ph-nuc-3
102: C          integer MNEUTPN(NUCPNA), MNUCPN(NMAT+1), LPIDPN(NUCPNA), ! ph-nuc-3
103: C          & KZMAT(NZONE), INCSTPN(NSTALY,2), MTMPN(NMTMPN) ! ph-nuc-3
104: C
105: C          real DNZON(NUC,NZONE,2)
106: C          real SMIC(NBANK,NUC,NSMIC), SMAC(NBANK,MB,NSMAC)
107: C          integer LMIC(8), LMIC(8)
108: C          real SGTAL(NBANK,NSTAL)
109: C
110: C          real*8 DBNK(NBANK,*)
111: C          integer KDBNK(0:MDBNK)
112: C          integer IBNK(NBANK,*)
113: C          integer KIBNK(0:MIBNK)
114: C
115: C          real RESP(NGROUP,NRESP)
116: C          real TIMEB(NTIME+1)
117: C          real TCUT
118: C          integer MKREG(NREG,2)
119: C
120: C ... working array ...
121: C
122: C          integer IPP(NN)
123: C          integer IPMK(NN)
124: C          real*8 DNF(NN)
125: C          real DE(NN), E(NN), SMCN(NN) ! ph-nuc-3
126: C          integer IEP(NN) ! ph-nuc-3

```

src/mvp/staln1.f

```
127: C
128: C ... offset of direct-tally data in IDTAL(*)
129: C
130: CC parameter( IDTOFF = 12 )
131: C
132: C ... offset of tally dimension dependent data in IDTAL(*)
133: C
134: CC parameter( IDMOFF = 3 )
135: C
136: include '../shared/INC/_IDXOFF'
137: C
138: parameter(ZERO = 0.0d0) ! ph-nuc-3
139: data ID0 /0/
140: C
141: C -----
142: C
143: do 100 I = 1, NN
144:   IPP(I) = 0
145: 100 continue
146: C
147:   IGOFF = KNGP(IDTALY(5)) - 1
148: C
149: C ... this flag shows that time dependent tally is really necessary
150: C
151:   JTIME2 = 0
152: C
153: C ... IDO is to check multiple time tally is finished or not
154: C   in time dependent case.
155: C
156:   IDO = NN
157:   ICYCLE = 0
158: C
159: C ... check marker region and marker region tally bin in IPMK(I) ...
160: C
161:   if ( NMKREG.gt.0 ) then
162:     IDT = IDTOFF
163:     do 110 IB = 1, IDTALY(6)
164:       KBIN = IDTALY(IDT+1)
165: C
166: C ... this tally has marker region
167: C   if ( KBIN.eq.8 ) then
168: C     IDTM = IDT
169: C     go to 120
170: C   end if
171: C     IDT = IDT + IDMOFF + IDTALY(IDT+3)
172: 110 continue
173:   go to 160
174: 120 continue
175: C
176: C ... set non-marked as default
177: C   do 130 I = 1, NN
178: C     IPMK(I) = 2
179: 130 continue
180: C
181: C   do 150 IM = 1, NMKREG
182: C     KG = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
183: C
184: C ... this marker region is used in current tally ...
185: C
186: C   if ( KG.eq.1 ) then
187: C     KI9 = KIBNK(9) + IM - 1
188: C     do 140 I = 1, NN
189: C       if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I) = 1
190: 140 continue
191:   end if
```

```
192: 150 continue
193: C
194: 160 continue
195:   end if
196: C
197: C ... for track length tally of a time dependent case,
198: C   more than one cycles for a loop
199: C   that starts from the following label.
200: C
201: 170 continue
202: C
203: c##<2007/03/14:PN3:
204: c##   if ( IDO.ge.0 ) then
205: c##     if ( IDO.gt.0 ) then
206: c##>
207: C
208: C ... calculate position in dtaly(*)
209: C
210:   IDT = IDTOFF
211:   do 190 IB = 1, IDTALY(6)
212:     KBIN = IDTALY(IDT+1)
213: c##<2007/03/14:PN3:
214:   if ( KBIN.eq.3 ) JTIME2 = 1
215: c##>
216:   do 180 I = 1, NN
217:     if ( IPP(I).ge.0 ) then
218:       K = 0
219: C
220: C ... only 1 dtaly bin
221: C
222:   if ( IDTALY(IDT+3).eq.0 ) then
223:     K = 1
224: C
225: C .. marker region
226: C
227:   else if ( KBIN.eq.8 ) then
228:     K = IPMK(I)
229: C
230:   else
231: C
232: C .. region
233: C
234:   if ( KBIN.eq.1 ) then
235:     K = IRG(I)
236: C
237: C .. energy
238: C
239:   else if ( KBIN.eq.2 ) then
240:     K = IEG(I)
241: C
242: C ... for photon ...
243: C   if ( IDTALY(5).eq.2 ) K = K - NGP1
244: C
245: C     K = IEG(I) - IGOFF
246: C
247: C .. time
248: C
249:   else if ( KBIN.eq.3 ) then
250:     K = ITP(I) + ICYCLE
251: c##<2007/03/14:PN3:
252:   JTIME2 = 1
253: c##>
254: C
255: C   else if ( kbin.eq.4 ) then
256: C     k = ???
```


src/mvp/staln1.f

```

387: 300      continue
388:      if ( DNZON(MNEUTPN(L1),MZONE,1).gt.0.0 ) then
389:        do I = 1, NN
390:          DNF(I) = FLX(I)*SMCPN(I)*DNSTPN(L1)
391:        end do
392:      end if
393:    end if
394:  else
395:    do I = 1, NN
396:      KZM = KZMAT(IZN(I))
397:      do L1 = MNUCPN(KZM), MNUCPN(KZM+1)-1
398:        if ( INUC.eq.LPIDEN(L1) ) go to 310
399:      end do
400:      go to 320
401:    continue
402:  310      if ( DNZON(MNEUTPN(L1),IZN(I),1).gt.0.0 ) then
403:        DNF(I) = FLX(I)*SMCPN(I)*DNSTPN(L1)
404:      end if
405:    320      continue
406:  end do
407: end if
408: 330      continue
409: C
410: C ... smic for photonuclear ... (per atom)
411: else if ( IDTALY(7).eq.1004 ) then
412:   IMIC = IDTALY(8) / 1000
413:   INUC = IDTALY(8) - IMIC * 1000
414:   do I = 1, NN
415:     E(I) = EEE(IBP(I))
416:   end do
417:   call GMICPN2(INUC, NN, E, NUCPN1, NMTPN, CXPN, MCXPN,
418: & INCSTPN(IDTALY(9),2), MTMPN, NMTMPN, KLBPN1,
419: & KLBPN2, XLBPN1, EBOTLPN, IEP, DE, SMEPN )
420:   do I = 1, NN
421:     DNF(I) = FLX(I)*SMCPN(I)
422:   end do
423: c##>
424: C
425: C ... sgtal ...
426: else if ( IDTALY(7).eq.2000 ) then
427:   do 230 I = 1, NN
428:     DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
429:   230   continue
430: C
431: C ... smac ...
432: else if ( IDTALY(7).eq.3000 ) then
433:   IREA = IDTALY(8)
434:   if (IDTALY(5).eq.1 .or. IDTALY(5).eq.2) IREA = LMAC(IREA)
435:   do 240 I = 1, NN
436:     if ( IMB(I).gt.0 ) then
437:       DNF(I) = FLX(I)*
438: & SMAC(IBP(I),IMB(I),IREA)
439:     else
440:       DNF(I) = 0.0D0
441:     end if
442:   240   continue
443: C
444: C ... flux ...
445: else
446:   do 250 I = 1, NN
447:     DNF(I) = FLX(I)
448:   250   continue
449: end if
450: end if
451: C

```

```

452: C --- if tally event type is not track length,
453: C no need to multi-cycle tally for time bin
454: C
455:   if ( IDTALY(4).ne.2 ) then
456:     JTIME2 = 0
457:   end if
458: C
459: C --- weight by time-bin occupation rate ---
460: C
461:   IP0 = IDTALY(3)
462:   if ( JTIME2.eq.0 ) then
463:     do 260 I = 1, NN
464:       if ( IPP(I).ge.0 ) then
465:         IP = IP0 + IPP(I)
466:         DTALY(IP) = DTALY(IP) + DNF(I)
467:       end if
468:     260   continue
469:   else
470:     do 270 I = 1, NN
471:       if ( IPP(I).ge.0 ) then
472:         ITPP = ITP(I) + ICYCLE
473: c##<2007/03/14:PN3:
474: c##   if ( ABS(TI(I)).ne.0.0 ) then
475: c##     DT = MIN(TI(I),DBLE(TIMEB(ITPP+1))-TIM(I))
476: c## & - MAX(0.0D0,DBLE(TIMEB(ITPP))-TIM(I))
477: c##     DDD = DNF(I)*DT/TI(I)
478: c##   else
479: c##     DDD = 0.0
480: c##   end if
481: c##C
482: c##   IP = IP0 + IPP(I)
483: c##   DTALY(IP) = DTALY(IP) + DDD
484: c##   if ( ICYCLE.eq.0.and.ITPP.ge.NTIME.and.
485: & TIM(I).ge.TIMEB(NTIME+1) ) then
486: & IPP(I) = -2
487: & IDO = IDO - 1
488:   else
489:     if ( ABS(TI(I)).ne.ZERO ) then
490:       DT = min(TI(I),dble(TIMEB(ITPP+1))-TIM(I))
491: & - max(ZERO,dble(TIMEB(ITPP))-TIM(I))
492:       DDD = DNF(I)*DT/TI(I)
493:     else
494:       DDD = 0
495:     end if
496:     IP = IP0 + IPP(I)
497:     DTALY(IP) = DTALY(IP) + DDD
498:   end if
499: c##>
500:   end if
501:   270   continue
502: end if
503: C
504: C ... judge whether further tallying is required
505: C
506:   if ( JTIME2.eq.0 ) then
507:     return
508:   else if ( JPTIM.eq.0 ) then
509:     do 280 I = 1, NN
510:       if ( IPP(I).ge.0 ) then
511:         ITPP = ITP(I) + ICYCLE + 1
512:         if ( ITPP.gt.NTIME
513: & .or. TIMEB(ITP(I)+ICYCLE+1)-TIM(I).ge.TI(I) ) then
514: & IPP(I) = -2
515: & IDO = IDO - 1
516:       else

```

src/mvp/staln1.f

```

517:      IPP(I) = 0
518:    end if
519:  end if
520:  280    continue
521:    ICYCLE = ICYCLE + 1
522:  c##<2007/03/14:PN3:
523:  c##    if ( IDO.gt.0 ) then
524:  c##      go to 170
525:  c##    else if ( IDO.lt.0 ) then
526:  c##      write(IOW,*)
527:  c##    if ( IDO.lt.0 ) then
528:  c##      write(IOW,'(1X,A,A,I7)')
529:  c##>
530:  &      'XXX(STALN1) Program error?: IDO is negative !!',
531:  &      ' IDO=',IDO
532:  &      write(IOW,*) ' Something wrong happened, STOP !!!'
533:  &      stop 666
534:  end if
535:  else
536:  do 290 I = 1, NN
537:    if ( IPP(I).ge.0 ) then
538:      ITPP = ITP(I) + ICYCLE
539:      TII  = TI(I)
540:      TT   = TIM(I) + TII
541:      IPP(I) = 0
542:      TT1 = TIMEB(1)
543:      if ( ITPP.le.NTIME ) TT1 = TIMEB(ITPP+1)
544:      if ( TT.le.TT1 ) then
545:        IPP(I) = -2
546:        IDO    = IDO - 1
547:      else if ( ITPP.ge.NTIME ) then
548:        TI(I) = TT - TCUT
549:        TIM(I) = 0.0D0
550:        DNF(I) = DNF(I) * TI(I) / TII
551:        ITP(I) = -ICYCLE
552:        JPTIM2 = 1
553:      end if
554:    end if
555:  290    continue
556:    ICYCLE = ICYCLE + 1
557:  c##<2007/03/14:PN3:
558:  c##    if ( IDO.gt.0 ) then
559:  c##      go to 170
560:  c##    else if ( IDO.lt.0 ) then
561:  c##      write(IOW,*)
562:  c##    if ( IDO.lt.0 ) then
563:  c##      write(IOW,'(1X,A,A,I7)')
564:  c##>
565:  &      'XXX(STALN1) Program error?: IDO is negative !!',
566:  &      ' IDO=',IDO
567:  &      write(IOW,*) ' Something wrong happened, STOP !!!'
568:  &      stop 666
569:  end if
570:  end if
571:  c##<2007/03/14:PN3:
572:  C
573:  390    continue
574:
575:  C      ... check the closed lower time boundary
576:
577:  if ( JTIME2.ne.0 ) then
578:    if ( ID0.eq.0 ) then
579:      if ( ICYCLE.eq.0 ) then
580:        ICYCLE = 1
581:      else if ( ICYCLE.gt.1 ) then

```

```

582:      go to 400
583:    end if
584:  end if
585:  do I = 1, NN
586:    if ( IPP(I).eq.-1 ) then
587:      ITPP = ITP(I) + ICYCLE
588:      TT   = TI(I) + TIM(I)
589:      if ( ITPP.le.NTIME.and.TT.gt.TIMEB(ITPP).and.
590:        &      TIM(I).le.TIMEB(NTIME+1) ) then
591:        IPP(I) = 0
592:        ID0    = ID0 + 1
593:      end if
594:    end if
595:  end do
596:  end if
597:  400    continue
598:
599:  if ( JTIME2.ne.0.and.ID0.gt.0 ) go to 170
600:  c##>
601:  C
602:  end if ! ... if IDO > 0
603:  C
604:  return
605:  end

```

src/mvp/staln3.f

```

1:      subroutine STALN3( IOW,  ISURF, NTSRF,
2:      &                  JTIME, MZONE, IDTALY, FLX,  NN,
3:      &                  IBP,  IBP0,  IEG,  IRG,  IMB,  ITP,
4:      &                  NGROUP, NREG,  NRESP, NUC,  NBANK,
5:      &                  NSMIC, NSMAC, NSTAL, MB,  DNZON, NZONE,
6:      &                  NTIME, NEMIC, RESP,  NMKREG, MKREG, DTALY, SMIC,
7:      &                  LMIC,  SMAC,  LMAC,  SGTAL, DBNK,  KDBNK,
8:      &                  MDBNK, IBNK,  KIBNK, MIBNK, DNF,  IPP,  IPMK
9:      &                  )
10: C=====
11: C PURPOSE: Take special tallies for surface crossing tally
12: C          (Surface ISURF)
13: C
14: C (can treat neutron & photon but cannot treat both particles at once)
15: C
16: C CALLED IN: FLIONE
17: C-----
18: C
19: C arguments ( i = input, o=output, w=work )
20: C
21: C i IOW : message printout I/O unit
22: C i ISURF : tally surface #.
23: C i NTSRF : total number of tally surface.
24: C i JTIME : problem is time dependent if non zero.
25: C i MZONE : zone # if all particles have the same zone#.
26: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
27: C               routine.
28: C i FLX(NN) : current/flux or any other contribution to tally
29: C i NN      : number of particles
30: C i IBP(NN) : bank pointers for each particle
31: C i IBP0(NN) : pointers to point IEG, IRG and IMB.
32: C i IEG(*)  : energy group #
33: C i IRG(*)  : region #
34: C i IMB(*)  : value of MMAC(IBP,1) for each particle
35: C i ITP(NN) : time bin on surface crossing
36: C o DTALY(*) : tally
37: C
38: C w DNF(NN) : working area
39: C w IPP(NN) : working area
40: C w IPMK(NN) : working area
41: C-----
42: C
43: C Structure of IDTAL(*)
44: C
45: C +----- direct tally loop ( NDTALY )
46: C |
47: C | (1) ID
48: C | (2) (dummy for debug: -999)
49: C | (3) pointer to DTALY(*)
50: C | (4) event #
51: C | (5) particle type
52: C | (6) number of dimensions
53: C | (7) multiplication factor type
54: C | (8) nuclide/atom # & reaction #
55: C | (9) direct tally #
56: C | (10) custom tally # (non-zero means user customized treatment)
57: C | (11) detector # (for surface event positive tally-surface #
58: C |           means particle current tally, and negative one
59: C |           means flux tally)
60: C | ( IDTOFF data so far )
61: C |
62: C | +----- tally dimension loop
63: C | | (1) dimension type
64: C | | (2) number of tallied bins
65: C | | (3) size of common bin -> tallied bin conversion table

```

```

66: C | | ( IDMOFF data so far )
67: C | | +----- common bin loop
68: C | | | tallied bin # for each common bin
69: C | | +-----
70: C | +-----
71: C +-----
72: C=====
73:      implicit real*8(A-H,O-Z)
74: C
75:      include 'INC/_KPIDS'
76:      include 'INC/_NGPS'
77: C
78:      integer IDTALY(*)
79: C
80:      real*8 FLX(NN)
81: C
82:      integer IBP(NN), IBP0(NN)
83:      integer IEG(*), IMB(*), IRG(*)
84:      integer ITP(NN)
85: C
86:      real*8 DTALY(*)
87: C
88:      real DNZON(NUC,NZONE,2)
89:      real SMIC(NBANK,NUC,NSMIC), SMAC(NBANK,MB,NSMAC)
90:      integer LMAC(8), LMIC(8)
91:      real SGTAL(NBANK,NSTAL)
92: C
93:      real*8 DBNK(NBANK,*)
94:      integer KDBNK(0:MDBNK)
95:      integer IBNK(NBANK,*)
96:      integer KIBNK(0:MIBNK)
97: C
98:      real RESP(NGROUP,NRESP)
99:      integer MKREG(NREG,2)
100: C
101: C ... working array ...
102: C
103:      integer IPP(NN)
104:      integer IPMK(NN)
105:      real*8 DNF(NN)
106: C
107: C ... offset of direct-tally data in IDTAL(*)
108: C
109: CC      parameter( IDTOFF = 12 )
110: C
111: C ... offset of tally dimension dependent data in IDTAL(*)
112: C
113: CC      parameter( IDMOFF = 3 )
114: C
115:      include '../shared/INC/_IDXOFF'
116: C
117: C-----
118: C
119:      do 100 I = 1, NN
120:          IPP(I) = 0
121:      100 continue
122: C
123:          IGOFF = KNGP(IDTALY(5)) - 1
124: C
125: C ... check marker region and marker region tally bin in IPMK(I) ...
126: C
127:      if ( NMKREG.gt.0 ) then
128:          IDT = IDTOFF
129:          do 110 IB = 1, IDTALY(6)
130:              KBIN = IDTALY(IDT+1)

```

src/mvp/staln3.f

```

131: C
132: C      ... this tally has marker region
133: C      if ( KBIN.eq.8 ) then
134: C          IDTM = IDT
135: C          go to 120
136: C      end if
137: C      IDT = IDT + IDMOFF + IDTALY(IDT+3)
138: 110 continue
139: C      go to 160
140: 120 continue
141: C
142: C      ... set non-marked as default
143: C      do 130 I = 1, NN
144: C          IPMK(I) = 2
145: 130 continue
146: C
147: C      do 150 IM = 1, NMKREG
148: C          KG = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
149: C      ... this marker region is used in current tally ...
150: C
151: C      if ( KG.eq.1 ) then
152: C          KI9 = KIBNK(9) + IM - 1
153: C          do 140 I = 1, NN
154: C              if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I) = 1
155: 140 continue
156: C          end if
157: C      continue
158: 150 continue
159: C
160: 160 continue
161: C      end if
162: C
163: C      ... calculate position in dtaly(*)
164: C
165: C      IDT = IDTOFF
166: C      do 180 IB = 1, IDTALY(6)
167: C          KBIN = IDTALY(IDT+1)
168: C          do 170 I = 1, NN
169: C              if ( IPP(I).ge.0 ) then
170: C                  K = 0
171: C
172: C      ... only 1 dtaly bin
173: C
174: C          if ( IDTALY(IDT+3).eq.0 ) then
175: C              K = 1
176: C
177: C      .. marker region
178: C
179: C          else if ( KBIN.eq.8 ) then
180: C              K = IPMK(I)
181: C
182: C          else
183: C
184: C      .. region
185: C
186: C          if ( KBIN.eq.1 ) then
187: C              K = IRG(IBP0(I))
188: C
189: C      .. energy
190: C
191: C          else if ( KBIN.eq.2 ) then
192: C              K = IEG(IBP0(I))
193: C      ... for photon ...
194: C      if ( IDTALY(5).eq.2 ) K = K - NGP1
195: C

```

```

196: C          K = IEG(IBP0(I)) - IGOFF
197: C
198: C      .. time
199: C
200: C          else if ( KBIN.eq.3 ) then
201: C              K = ITP(I)
202: C              JTIME2 = 1
203: C
204: C      .. angle
205: C
206: C          else if ( KBIN.eq.4 ) then
207: C              K = IBNK(IBP(I),KIBNK(10)+NTSRF+ISURF-1)
208: C
209: C      .. generation
210: C
211: C          else if ( KBIN.eq.5 ) then
212: C              K = MIN(IBNK(IBP(I),KIBNK(4)),IDTALY(IDT+3))
213: C
214: C      .. source set
215: C
216: C          else if ( KBIN.eq.6 ) then
217: C              K = IBNK(IBP(I),KIBNK(1))
218: C
219: C      .. source region
220: C
221: C          else if ( KBIN.eq.7 ) then
222: C              K = IBNK(IBP(I),KIBNK(2))
223: C
224: C      c##<2007/03/14:PN3:
225: C
226: C      ... produce region
227: C          else if ( KBIN.eq.10 ) then
228: C              K = IBNK(IBP(I),KIBNK(17))
229: C
230: C      ... produce nuclide
231: C          else if ( KBIN.eq.11 ) then
232: C              K = IBNK(IBP(I),KIBNK(18))
233: C
234: C      ... produce reaction
235: C          else if ( KBIN.eq.12 ) then
236: C              K = IBNK(IBP(I),KIBNK(19))
237: C
238: C      c##>
239: C          end if
240: C
241: C      ... convert common bin to d-tally bin
242: C          K = IDTALY(IDT+IDMOFF+K)
243: C      end if
244: C
245: C      ... IPP(I) = -1 : not in tally range !!!
246: C
247: C      if ( K.eq.0 ) then
248: C          IPP(I) = -1
249: C      else
250: C          IPP(I) = IPP(I)*IDTALY(IDT+2) + K - 1
251: C      end if
252: C      end if
253: 170 continue
254: C      IDT = IDT + IDMOFF + IDTALY(IDT+3)
255: 180 continue
256: C
257: C      INUC = IDTALY(8) /1024
258: C      IREA = IDTALY(8) - INUC*1024
259: C
260: C

```

src/mvp/staln3.f

```
261: C      ... response ...
262:      if ( IDTALY(7).lt.0 ) then
263:      do 190 I = 1, NN
264:          DNF(I) = FLX(I)*RESP(IEG(IBP0(I)),-IDTALY(7))
265:      190 continue
266: C
267: C      ... ???? ...
268: Ccccccc      else if( idtaly(7).lt.1000 ) then
269: Ccccccc          dnf(i) = dnf(i) *
270: C
271: C      ... smic ...
272:      else if ( IDTALY(7).eq.1001 ) then
273:          if ( IDTALY(5).eq.1 ) IREA = LMIC(IREA)
274:          do 200 I = 1, NN
275:              DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)*
276:              &      DNZON(INUC,MZONE,IDTALY(5))
277:      200 continue
278: C
279: C      ... smic ... (per atom)
280:      else if ( IDTALY(7).eq.1002 ) then
281:          if ( IDTALY(5).eq.1 ) IREA = LMIC(IREA)
282:          do 210 I = 1, NN
283:              DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)
284:      210 continue
285: C
286: C      ... sgtal ...
287:      else if ( IDTALY(7).eq.2000 ) then
288:          do 220 I = 1, NN
289:              DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
290:      220 continue
291: C
292: C      ... smac ...
293:      else if ( IDTALY(7).eq.3000 ) then
294:          IREA = IDTALY(8)
295:          if ( IDTALY(5).eq.1 ) IREA = LMAC(IREA)
296:          do 230 I = 1, NN
297:              if ( IMB(IBP0(I)).gt.0 ) then
298:                  DNF(I) = FLX(I)*
299:                  &      SMAC(IBP(I),IMB(IBP0(I)),IREA)
300:              else
301:                  DNF(I) = 0.0D0
302:              end if
303:      230 continue
304: C
305: C      ... simple current/flux ...
306:      else
307:          do 240 I = 1, NN
308:              DNF(I) = FLX(I)
309:      240 continue
310:      end if
311: C
312: C
313:      IP0 = IDTALY(3)
314:      do 250 I = 1, NN
315:          if ( IPP(I).ge.0 ) then
316:              IP = IP0 + IPP(I)
317:              DTALY(IP) = DTALY(IP) + DNF(I)
318:          end if
319:      250 continue
320: C
321:      return
322:      end
```

src/mvp/staln7.f

```

1:      subroutine STALN7(IOW, IDTALY,
2:      I          FLX,   IBP,   NN,
3:      T          IEG,   IRG,   ITP,
4:      N          NGROUP,NREG,  NTIME, NBANK,
5:      R          DTALY,
6:      B          DBNK,  KDBNK, MDBNK,  IBNK,  KIBNK, MIBNK,
7:      W          DNF,   IPP,
8:      &          TTP, JTLST, IOTL ! time-list
9:      &)
10: C=<MVP>=====
11: C PURPOSE: Take special tallies for noise analysis
12: C
13: C (can treat neutron & photon, but cannot treat both particles at once)
14: C
15: C CALLED IN: PANLGF, SEAONE
16: C=====
17: C
18: C arguments ( i = input, o=output, w=work )
19: C
20: C i IOW : message printout I/O unit
21: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
22: C      routine.
23: C i FLX(NN) : flux or contri bution to tally
24: C i NN : NUMBER OF PARTICLES
25: C i IEG(NN) : GROUP #
26: C i IRG(NN) : REGION #
27: C i IBP(NN) : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
28: C      JREAC=1 OR JEIGN.NE.0
29: C i ITP(NN) : current time bin of each particle
30: C i TTP(NN) : current time of each particle
31: C o DTALY(*) : tally
32: C
33: C w DNF(NN) : WORK AREA
34: C w IPP(NN) : WORK AREA
35: C-----
36: C
37: C Structure of IDTAL(*)
38: C
39: C +----- direct tally loop ( NDTALY )
40: C |
41: C | (1) ID
42: C | (2) (dummy for debug: -999)
43: C | (3) pointer to DTALY(*)
44: C | (4) event #
45: C | (5) particle type
46: C | (6) number of dimensions
47: C | (7) multiplication factor type
48: C | (8) nuclide/atom # & reaction #
49: C | (9) direct tally #
50: C | (10) custom tally # (non-zero means user customized treatment)
51: C | (11) detector # (for surface event positive tally-surface #
52: C |      means particle current tally, and negative one
53: C |      means flux tally)
54: C | ( IDTOFF data so far )
55: C |
56: C | +----- tally dimension loop
57: C | | (1) dimension type
58: C | | (2) number of tallied bins
59: C | | (3) size of common bin -> tallied bin conversion table
60: C | | ( IDMOFF data so far )
61: C | | +----- common bin loop
62: C | | | tallied bin # for each common bin
63: C | | +-----
64: C | +-----
65: C +-----

```

```

66: C=====
67:      implicit real*8(A-H,O-Z)
68: C
69:      include 'INC/_KPIDS'
70:      include 'INC/_NGPS'
71: C
72:      integer IDTALY(*)
73: C
74:      real*8 FLX(NN)
75:      integer IEG(NN), IBP(NN), IRG(NN), ITP(NN)
76:      real*8 TTP(NN) ! time-list
77: C
78:      real*8 DTALY(*)
79: C
80:      real*8 DBNK(NBANK,*)
81:      integer KDBNK(0:MDBNK)
82:      integer IBNK(NBANK,*)
83:      integer KIBNK(0:MIBNK)
84: C
85:      integer IPP(NN)
86:      real*8 DNF(NN)
87: C
88: C
89: C ... offset of direct-tally data in IDTAL(*)
90: C
91: CC      parameter( IDTOFF = 12 )
92: C
93: C ... offset of tally dimension dependent data in IDTAL(*)
94: C
95: CC      parameter( IDMOFF = 3 )
96: C
97:      include '../shared/INC/_IDXOFF'
98: C
99: C-----
100:      do 100 I = 1, NN
101:          IPP(I) = 0
102:      100 continue
103: C
104:      IGOFF = KNGP(IDTALY(5)) - 1
105: C
106: C ... this flag shows that time dependent tally is really necessary
107: C
108: C ... calculate position in dtaly(*)
109: C
110:      IDT = IDTOFF
111:      NTT = 0
112:      do 130 IB = 1, IDTALY(6)
113:          KBIN = IDTALY(IDT+1)
114:          do 120 I = 1, NN
115:              if ( IPP(I).ge.0 ) then
116:                  K = 0
117: C
118: C ... only 1 dtaly bin
119: C
120:              if ( IDTALY(IDT+3).eq.0 ) then
121:                  K = 1
122:              else
123: C
124: C .. region
125: C
126:                  if ( KBIN.eq.1 ) then
127:                      K = IRG(I)
128: C
129: C .. energy
130: C

```

src/mvp/staln7.f

```
131:           else if ( KBIN.eq.2 ) then
132: CC          K          = IEG(I)
133: C          ... for photon ...
134: CC          if ( IDTALY(5).eq.2 ) K = K - NGP1
135:
136:           K          = IEG(I) - IGOFF
137: C
138: C          .. time
139: C
140:           else if ( KBIN.eq.3 ) then
141: C          K          = ITP(I)
142: C
143: CCCCCCCCCC CC      else if( kbin.eq.4 ) then
144: CCCCCCCCCC CC      k = ???
145: C
146: C
147: C          .. generation
148: C
149:           else if ( KBIN.eq.5 ) then
150: C          K          = MIN( IBNK( IBP(I), KIBNK(4)), IDTALY( IDT+3) )
151:           end if
152: C
153: C          ... convert common bin to d-tally bin
154: C
155:           K          = IDTALY( IDT+IDMOFF+K )
156:           end if
157: C
158: C          ... IPP(I) = -1 : not in tally range !!!
159: C
160:           if ( K.eq.0 ) then
161: C          IPP(I) = -1
162:           else
163: C          IPP(I) = IPP(I)*IDTALY( IDT+2 ) + K - 1
164: C          NTT      = NTT + 1
165:           end if
166:           end if
167: 120      continue
168:           IDT      = IDT + IDMOFF + IDTALY( IDT+3 )
169: 130      continue
170: C
171: C
172:           if ( NTT.gt.0 ) then
173: C          IP0      = IDTALY(3)
174: C          do 140 I = 1, NN
175: C          DNF(I) = FLX(I)
176: C          if ( IPP(I).ge.0 ) then
177: C          IP      = IP0 + IPP(I)
178: C          DTALY(IP) = DTALY(IP) + DNF(I)
179: C          if( JTLST.ne.0 ) write( IOTL, IDTALY(1), TTP(I), DNF(I) ! time-lis
t
180:           end if
181: 140      continue
182:           end if
183: C
184:           return
185:           end
```

src/mvp/staln9.f

```

1:      subroutine STALN9( IOW, IDTALY, FLX , IBP, NN,
2:      T                  DTALY,
3:      N                  IMPMAX, NGROUP, NREG , NRESP , NSTAL ,
4:      N                  NZONE, NTIME, NMKREG, MKREG,
5:      T                  SGTAL, RESP,
6:      N                  NUC, NSMIC,
7: c##<2007/03/14:PN4:
8: c## X                  SMIC, LMIC,
9:      X                  SMIC, LMIC, SMAC, LMAC, NSMAC, MB,
10:     &                  KMACI,
11: c##>
12:     E                  KDET, IGGI, ITTI,
13:     E                  DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
14: c##<2007/03/14:PN4:
15: c## W                  DNF, IPP, IPMK )
16:     W                  DNF, IPP, IPMK,
17:     &                  NPATOM, NMT, NMTP, KPNPRD, DNZON, IZZI, EEEI,
18:     &                  E, NMAT, KZMAT, NUCPNI, NMTPN, NSTALY, CXPN,
19:     &                  MCXPN, INCSTPN, MTMPN, NMTPN, KLBPN1, KLBPN2,
20:     &                  KLBPN1, EBOTLPN, NUCPNA, MNEUTPN, DNSTPN,
21:     &                  MNUCPN, LPIDPN, DE, IEP, SMCN, MATZ, IMATZ )
22: c##>
23: C=<MVP>=====
24: C PURPOSE: Take special tallies (neutron/photon)
25: C
26: C (can treat neutron & photon, but cannot treat both particles at once)
27: C
28: C CALLED IN: NXTFLI
29: C=====
30: C
31: C arguments ( i = input, o=output, w=work )
32: C
33: C i IOW : message printout I/O unit
34: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
35: C routine.
36: C i FLX(NN) : flux or contribution to tally
37: C i NN : NUMBER OF PARTICLES
38: C i IBP(NN) : BANK POINTERS FOR EACH PARTICLE
39: C i IMB(NN) : value of MMAC(IBP,1) for each particle
40: C o DTALY(*) : tally
41: C
42: C w DNF(NN) : working area
43: C w IPP(NN) : working area
44: C w IPMK(NN) : working area
45: C-----
46: C
47: C Structure of IDTAL(*)
48: C
49: C +----- direct tally loop ( NDTALY )
50: C |
51: C | (1) ID
52: C | (2) (dummy for debug: -999)
53: C | (3) pointer to DTALY(*)
54: C | (4) event #
55: C | (5) particle type
56: C | (6) number of dimensions
57: C | (7) multiplication factor type
58: C | (8) nuclide/atom # & reaction #
59: C | (9) direct tally #
60: C | (10) custom tally # (non-zero means user customized treatment)
61: C | (11) detector # (for surface event positive tally-surface #
62: C | means particle current tally, and negative one
63: C | means flux tally)
64: C | ( IDTOFF data so far )
65: C |

```

```

66: C | +----- tally dimension loop
67: C | | (1) dimension type
68: C | | (2) number of tallied bins
69: C | | (3) size of common bin -> tallied bin conversion table
70: C | | ( IDMOFF data so far )
71: C | | +----- common bin loop
72: C | | | tallied bin # for each common bin
73: C | | +-----
74: C | +-----
75: C +-----
76: C=====
77:     implicit real*8(A-H,O-Z)
78: C
79:     integer IDTALY(*)
80: C
81:     real*8 FLX(NN)
82:     integer IBP(NN)
83:     integer IMB(NN)
84: C
85:     real*8 DTALY(*)
86: C
87:     real RESP(NGROUP,NRESP)
88:     real SGTAL(IMPMAX,NSTAL)
89: c##<2007/03/14:PN4:
90:     real DNZON(NUC,NZONE,2)
91:     real SMIC(IMPMAX,NUC,NSMIC), SMAC(IMPMAX,MB,NSMAC)
92:     integer LMAC(8), LMIC(8), KMACI(IMPMAX,MB)
93: c## real SMIC(IMPMAX,NUC,NSMIC)
94: c## integer LMIC(8)
95: C
96:     real CXPN(MCXPN), XLBPN1(NUCPNI,6), EBOTLPN
97:     integer KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
98:     real DNSTPN(NUCPNA)
99:     integer MNEUTPN(NUCPNA), MNUCPN(NMAT+1), LPIDPN(NUCPNA),
100:    & KZMAT(NZONE), INCSTPN(NSTALY,2), MTMPN(NMTPN)
101:     integer MATZ(*)
102: c##>
103: C
104: C ... bank ...
105:     integer KDET(IMPMAX), IGGI(IMPMAX), ITTI(IMPMAX)
106: c##<2007/03/14:PN4:
107:     integer IZZI(IMPMAX)
108:     real EEEI(IMPMAX)
109: c##>
110:     real*8 DBNK(IMPMAX,*)
111:     integer KDBNK(0:MDBNK)
112:     integer IBNK(IMPMAX,*)
113:     integer KIBNK(0:MIBNK)
114: C
115:     integer MKREG(NREG,2)
116: C
117: C ... working array ...
118: C
119:     integer IPP(NN)
120:     integer IPMK(NN)
121:     real*8 DNF(NN)
122: c##<2007/03/14:PN4:
123:     real E(NN), DE(NN), SMCN(NN)
124:     integer IEP(NN), IMATZ(NN)
125: c##>
126: C
127: C ... offset of direct-tally data in IDTAL(*)
128: C
129: C parameter( IDTOFF = 12 )
130: C

```


src/mvp/staln9.f

```
131: C ... offset of tally dimension dependent data in IDTAL(*)
132: C
133: C      parameter( IDMOFF  = 3 )
134: C
135: c##<2007/03/14:PN4:
136:       include 'INC/_FLAGS'
137: c##>
138:       include 'INC/_KPIDS'
139:       include 'INC/_NGPS'
140:       include '../shared/INC/_IDXOFF'
141: C
142: C-----
143: C
144:       do 100 I = 1, NN
145:         IPP(I) = 0
146: c##<2007/03/14:PN4:
147:         IMATZ(I) = 0
148: c##>
149:       100 continue
150: C
151:         IGOFF = KNGP(IDTALY(5)) - 1
152: c##<2007/03/14:PN4:
153: C
154:         ID0   = NN
155: c##>
156: C
157: C ... check marker region and marker region tally bin in IPMK(I) ...
158: C
159:       if ( NMKREG.gt.0 ) then
160:         IDT   = IDTOFF
161:         do 110 IB = 1, IDTALY(6)
162:           KBIN   = IDTALY(IDT+1)
163: C
164: C ... this tally has marker region
165:       if ( KBIN.eq.8 ) then
166:         IDTM   = IDT
167:         go to 120
168:       end if
169:         IDT   = IDT + IDMOFF + IDTALY(IDT+3)
170: 110   continue
171:       go to 160
172: 120   continue
173: C
174: C ... set non-marked as default
175:       do 130 I = 1, NN
176:         IPMK(I) = 2
177: 130   continue
178: C
179:       do 150 IM = 1, NMKREG
180:         KG     = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
181: C
182: C ... this marker region is used in current tally ...
183: C
184:       if ( KG.eq.1 ) then
185:         KI9    = KIBNK(9) + IM - 1
186:         do 140 I = 1, NN
187:           if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I)   = 1
188: 140       continue
189:         end if
190: 150   continue
191: C
192: 160   continue
193:       end if
194: C
195: C ... calculate position in dtaly(*)
196: C
197: c##<2007/03/14:PN4:
198:       NUCLM9 = 0
199:       MTSLM9 = 0
200:       if ( KPNPRD.eq.0 ) then
201:         NUCLM9 = NUC + NPATOM
202:         MTSLM9 = NMT + NMTP
203:       end if
204: C
205: 170 continue
206: c##>
207:       IDT   = IDTOFF
208:       do 190 IB = 1, IDTALY(6)
209:         KBIN   = IDTALY(IDT+1)
210:         NTT     = 0
211:         do 180 I = 1, NN
212:           if ( IPP(I).ge.0 ) then
213: c##<2007/03/14:PN3:
214:             K = 0
215: c##>
216: C
217: C ... only 1 dtaly bin
218: C
219:       if ( IDTALY(IDT+3).eq.0 ) then
220:         K     = 1
221: C
222: C .. marker region
223: C
224:       else if ( KBIN.eq.8 ) then
225:         K     = IPMK(I)
226: C
227:       else
228: C
229: C .. region : no region for point detector
230: C
231:       if ( KBIN.eq.1 ) then
232:         K     = IRG(I)
233: C
234: C .. energy
235: C
236:       if ( KBIN.eq.2 ) then
237:         K     = IGGI(IBP(I)) - IGOFF
238: C
239: C .. time
240: C
241:       else if ( KBIN.eq.3 ) then
242:         K     = ITTI(IBP(I))
243: C
244: Ccccccccccccccc cc   else if( kbin.eq.4 ) then
245: Ccccccccccccccc cc   k = ???
246: C
247: C
248: C .. generation
249: C
250:       else if ( KBIN.eq.5 ) then
251:         K     =
252:         &      MIN(IBNK( IBP(I),KIBNK(4)),IDTALY(IDT+3))
253: C
254: C .. source set
255: C
256:       else if ( KBIN.eq.6 ) then
257:         K     = IBNK( IBP(I),KIBNK(1))
258: C
259: C .. source region
260: C
```

src/mvp/staln9.f

```

261:         else if ( KBIN.eq.7 ) then
262:             K = IBNK(IBP(I),KIBNK(2))
263: C
264: C .. spatial point
265: C
266:         else if ( KBIN.eq.9 ) then
267:             K = KDETP(IBP(I))
268: c##<2007/03/14:PN3:PN4:
269: c## end if
270:         do IM = 1, MB
271:             if ( MATZ(K).eq.KMACI(IBP(I),IM) ) then
272:                 IMATZ(I) = IM
273:                 go to 173
274:             end if
275:         end do
276: 173 continue
277: C
278: C .. produce region
279: C
280:         else if ( KBIN.eq.10 ) then
281:             K = IBNK(IBP(I),KIBNK(17))
282:             if ( K.lt.0 ) K = 0
283: C
284: C .. produce nuclide
285: C
286:         else if ( KBIN.eq.11 ) then
287:             K = IBNK(IBP(I),KIBNK(18))
288:             if ( JPHNU.ne.0 ) then
289:                 if ( KPNPRD.eq.0 ) then
290:                     if ( K.le.NUCLM9 ) K = 0
291:                 end if
292:             end if
293: C
294: C .. produce reaction
295: C
296:         else if ( KBIN.eq.12 ) then
297:             K = IBNK(IBP(I),KIBNK(19))
298:             if ( JPHNU.ne.0 ) then
299:                 if ( KPNPRD.eq.0 ) then
300:                     if ( K.le.MTSLM9 ) K = 0
301:                 end if
302:             end if
303:         end if
304:         if ( KBIN.ge.10.and.KBIN.le.12.and.K.eq.0 ) go to 173
305: c##>
306: C
307: C .. convert common bin to d-tally bin
308: C
309:         K = IDTALY(IDT+IDMOFF+K)
310:         end if
311: c##<2007/03/14:PN3:
312: 175 continue
313: c##>
314: C
315: C ... IPP(I) = -1 : not in tally range !!!
316: C
317:         if ( K.eq.0 ) then
318:             IPP(I) = -1
319: c##<2007/03/14:PN4:
320:         ID0 = ID0 - 1
321: c##>
322:         else
323:             IPP(I) = IPP(I)*IDTALY(IDT+2) + K - 1
324:             NTT = NTT + 1
325:         end if

```

```

326:         end if
327: 180 continue
328:         IDT = IDT + IDMOFF + IDTALY(IDT+3)
329: 190 continue
330: C
331: c##<2007/03/14:PN4:
332:         if ( ID0.le.0 ) go to 390
333: C
334: c## INUC = IDTALY(8) /1024
335: c## IREA = IDTALY(8) - INUC*1024
336: c##>
337: C
338: C ... response ...
339:         if ( IDTALY(7).lt.0 ) then
340:             do 200 I = 1, NN
341:                 DNF(I) = FLX(I)*RESP(IGGI(IBP(I)),-IDTALY(7))
342: 200 continue
343: C
344: C ... ??? ...
345: ccccc else if( idtaly(7).lt.1000 ) then
346: ccccc dnf(i) = dnf(i) *
347: C
348: C ... smic ...
349:         else if ( IDTALY(7).eq.1001 ) then
350: c##<2007/03/14:PN4:
351:             INUC = IDTALY(8) /1024
352:             IREA = IDTALY(8) - INUC*1024
353: c##>
354:             if ( IDTALY(5).eq.1 ) IREA = LMIC(IREA)
355: c##<2007/03/14:PN4:
356: c## do 210 I = 1, NN
357: c## DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)*
358: c## & DNZON(INUC,MZONE,IDTALY(5))
359: c##10 continue
360:             if ( MZONE.ne.0 ) then
361:                 do 210 I = 1, NN
362:                     DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)*
363:                     & DNZON(INUC,MZONE,IDTALY(5))
364: 210 continue
365:             else
366:                 do 212 I = 1, NN
367:                     DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)*
368:                     & DNZON(INUC,IZZ(I),IDTALY(5))
369: 212 continue
370:             end if
371: c##>
372: C
373: C ... smic ... (per atom)
374:         else if ( IDTALY(7).eq.1002 ) then
375: c##<2007/03/14:PN4:
376:             INUC = IDTALY(8) /1024
377:             IREA = IDTALY(8) - INUC*1024
378: c##>
379:             if ( IDTALY(5).eq.1 ) IREA = LMIC(IREA)
380:             do 220 I = 1, NN
381:                 DNF(I) = FLX(I)*SMIC(IBP(I),INUC,IREA)
382: 220 continue
383: c##<2007/03/14:PN4:
384: C
385: C ... smic for photonuclear ...
386:         else if ( IDTALY(7).eq.1003 ) then
387:             IMIC = IDTALY(8) / 1000
388:             INUC = IDTALY(8) - IMIC * 1000
389:             do I = 1, NN
390:                 DNF(I) = 0

```

src/mvp/staln9.f

```

391:      E(I) = EEEI(IBP(I))
392:    end do
393:    call GMICPN2(INUC, NN, E, NUCPNI, NMTPN, CXPN, MCXPN,
394:      &      INCSTPN(IDTALY(9),2), MTMPN, NMTPMN, KLBPN1, KLBPN2,
395:      &      XLBPN1, EBOTLPN, IEP, DE, SMCPN )
396:    if ( MZONE.ne.0 ) then
397:      KZM = KZMAT(MZONE)
398:      if ( KZM.gt.0 ) then
399:        do L1 = MNUCPN(KZM), MNUCPN(KZM+1)-1
400:          if ( INUC.eq.LPIDPN(L1) ) go to 300
401:        end do
402:        go to 330
403:      continue
404:      if ( DNZON(MNEUTPN(L1),MZONE,1).gt.0.0 ) then
405:        do I = 1, NN
406:          DNF(I) = FLX(I) * SMCPN(I) * DNSTPN(L1)
407:        end do
408:      end if
409:    end if
410:  else
411:    do I = 1, NN
412:      KZM = KZMAT(IZZI(IBP(I)))
413:      do L1 = MNUCPN(KZM), MNUCPN(KZM+1)-1
414:        if ( INUC.eq.LPIDPN(L1) ) go to 310
415:      end do
416:      go to 320
417:    continue
418:    if ( DNZON(MNEUTPN(L1),IZZI(IBP(I)),1).gt.0.0 ) then
419:      DNF(I) = FLX(I) * SMCPN(I) * DNSTPN(L1)
420:    end if
421:  continue
422:    end do
423:  end if
424: 330 continue
425: C
426: C      ... smic for photonuclear ... (per atom)
427: else if ( IDTALY(7).eq.1004 ) then
428:   IMIC = IDTALY(8) / 1000
429:   INUC = IDTALY(8) - IMIC * 1000
430:   do I = 1, NN
431:     E(I) = EEEI(IBP(I))
432:   end do
433:   call GMICPN2(INUC, NN, E, NUCPNI, NMTPN, CXPN, MCXPN,
434:     &      INCSTPN(IDTALY(9),2), MTMPN, NMTPMN, KLBPN1, KLBPN2,
435:     &      XLBPN1, EBOTLPN, IEP, DE, SMCPN )
436:   do I = 1, NN
437:     DNF(I) = FLX(I) * SMCPN(I)
438:   end do
439: c##>
440: C
441: C      ... sgtal ...
442: else if ( IDTALY(7).eq.2000 ) then
443:   do 230 I = 1, NN
444:     DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
445:   230 continue
446: C
447: C      ... smac ...
448: else if ( IDTALY(7).eq.3000 ) then
449:   IREA = IDTALY(8)
450:   if ( IDTALY(5).eq.1 ) IREA = LMAC(IREA)
451:   do 240 I = 1, NN
452:     if ( IMATZ(I).gt.0 ) then
453:       DNF(I) = FLX(I) * SMAC(IBP(I),IMATZ(I),IREA)
454:     else
455:       DNF(I) = 0.0D0

```

```

456:       end if
457: 240 continue
458: C
459: C      ... flux ...
460:   else
461:     do 250 I = 1, NN
462:       DNF(I) = FLX(I)
463: 250 continue
464:   end if
465: C
466:   IP0 = IDTALY(3)
467:   do 260 I = 1, NN
468:     if ( IPP(I).ge.0 ) then
469:       IP = IP0 + IPP(I)
470:       DTALY(IP) = DTALY(IP) + DNF(I)
471:     end if
472: 260 continue
473: c##<2007/03/14:PN4:
474: C
475: 390 continue
476: c##>
477: C
478:   return
479: end

```

src/mvp/stkidt.f

```
1: C
2: C
3: C
4:     subroutine STKIDT( NAME, IOW,
5: &   ISTK, NSTK, NL,     IWRK, NBANK, NB )
6: C
7:     character*(*) NAME
8:     integer ISTK(*), NSTK(NL)
9:     character*4 IWRK(NBANK)
10: C
11:     write(IOW,*) ' --- CHECKING <', NAME, '> STACK --- ', NSTK(NL)
12: C
13:     if ( NL.gt.1 ) then
14:         NSUM      = 0
15:         do 100 I = 1, NL - 1
16:             NSUM    = NSUM + NSTK(I)
17:         100 continue
18:         if ( NSUM.ne.NSTK(NL) ) then
19:             write(IOW,*) ' (', NAME, ') DATA NUMBERS IN ZONES ARE ',
20: &             'INCONSISTENT WITH TOTAL ( SUM ', NSUM, ' TOTAL ',
21: &             NSTK(NL)
22:             write(IOW,*) ' NSTK: ', NSTK
23:         end if
24:     else
25:         NSUM      = NSTK(NL)
26:     end if
27: C
28:     NB          = NB + NSUM
29: C
30:     NI          = 0
31:     do 120 I = 1, NSUM
32:         if ( ISTK(I).le.0 .or. ISTK(I).gt.NBANK ) then
33:             write(IOW,*) ' (', NAME, ') RANGE OVER I=', I, ' DATA=',
34: &             ISTK(I)
35:             NI      = NI + 1
36:         end if
37: C
38:         do 110 J = I + 1, NSUM
39:             if ( ISTK(J).eq.ISTK(I) ) then
40:                 write(IOW,*) ' (', NAME, ') DATA ', I, ' AND ', J,
41: &                 ' IS THE ', 'SAME BANK INDEX = ', ISTK(I)
42:                 NI      = NI + 1
43:             end if
44:         110 continue
45:     120 continue
46:     if ( NI.ne.0 ) then
47:         write(IOW,*) ' INVALID STACK: ', (ISTK(I),I=1,NSUM)
48:     end if
49: C
50: C .... OVERLAP CHECK WITH OTHER STACK ...>
51: C
52:     do 130 I = 1, NSUM
53:         if ( ISTK(I).gt.0.and.ISTK(I).le.NBANK ) then
54:             if ( IWRK(ISTK(I)).ne.' ' ) then
55:                 write(IOW,*) ' -- DATA(', I, ') IN ', NAME,
56: &                 ' STACK OVERLAPS', ' (', IWRK(ISTK(I)),
57: &                 ') BANK INDEX ', ISTK(I)
58:             else
59:                 IWRK(ISTK(I)) = NAME(1:4)
60:             end if
61:         end if
62:     130 continue
63: C
64:     return
65: end
```

src/mvp/suncl.f

```

1:      subroutine SUNCL( IOW, A, H, NSORS, LSORS, KSRC, PSPAC,
2:      N          DUMAX, DUMIN, JNP ,
3:      D          JPUSD, JSPDT, XPDET, IPDET, IPDT2,
4:      S          IMSFL, IMNFL, IMZFL,
5:      S          IMDED, IMSLT, IMNLT, IMZLT ,
6:      G          NDBNK, NIBNK, TIMEB, KZMAT,
7:      B          XXX , YYY , ZZZ , AAA , BBB , CCC ,
8:      B          EEE , WWW , IZZ , IGG , TTT , ITT ,
9:      B          LEVL , LZZ , LPOS , LCRS ,
10:     B          DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
11:     B          LZZI , LPOSI, LCRSI,
12:     F          OPTI , PATH , KDETP, KLSFI, SMACI, MMACI,
13:     W          X , Y , Z , A0 , B0 , C0 , AA , BB , CC , DI ,
14:     W          DW1, DW2, DW3, DW4, DW5, TI ,
15:     W          IGI, IBP, IT0, W , R , SMCW )
16: C=====
17: C PURPOSE:  Calculation of uncollided contribution to next event
18: C            detectors.
19: C      <COMMENT>
20: C      Some contributions may not be calculated in
21: C      this routine, because the 'NXTEE' routine, that controls
22: C      tracking of imaginary test-particles, is called with
23: C      MODE=2 (cease processing when the number of particles
24: C      drops below a threshold value), and 'NXTEE' is not
25: C      called until the bank for imaginary test-particles is
26: C      full.
27: C
28: C CALLED IN: SOURCE
29: C CALLS:    NXTEE, LATUP2, LATDW2, GMACNX
30: C UPDATE :
31: C=====
32: C      NSORS : NUMBER OF SOURCE PARTICLES WHOSE UNCLLIDED CONTRIBUTIONS
33: C              ARE CALCULATED.
34: C      LSORS : BANK INDEX OF SOURCE PARTICLES.
35: C      KSRC  : TYPE OF SOURCE
36: C              KSRC = 0 : UNIFORM SOURCE.
37: C              > 0 : OTHER NON UNIFORM SOURCE.
38: C=====
39: CCC      implicit real*8 (D)
40: CCC      implicit real*8 (A-H,O-Z)
41: C
42: C      real A(*)
43: C      real H(*)
44: C
45: C**** BANK INDEX FOR SOURCE PARTICLES. *****
46: C
47: C      integer LSORS(NSORS)
48: C
49: C**** SOURCE DATA *****
50: C
51: C      real    PSPAC(10)
52: C
53: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR *****
54: C
55: C      real*8  XPDET(NPLEN,NPDET)
56: C      integer IPDET(NPDET,2), IPDT2(NEST,3,NPDET)
57: C      integer JPUSD(NPDET), JSPDT(NPDET)
58: C
59: C**** TALLY BIN DATA *****
60: C
61: C      real    TIMEB(*)
62: C
63: C**** STACK FOR IMAGINARY PARTICLE *****
64: C
65: C      integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(NZONE+1),

```

```

66: C      &      IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
67: C
68: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE *****
69: C
70: C      real*8  XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2)
71: C      real*8  AAA(NBANK,2), BBB(NBANK,2), CCC(NBANK,2)
72: C
73: C      real    WWW(NBANK,2), EEE(NBANK,2)
74: C      real*8  TTT(NBANK,2)
75: C
76: C      integer IZZ(NBANK,2), IGG(NBANK,2), ITT(NBANK,2)
77: C      integer LEVL(NBANK,2), LZZ(NBANK,NEST),
78: C      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), LZZI(IMPMAX,NEST),
79: C      &      LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
80: C
81: C      real*8  DBNK(NBANK,NDBNK,2)
82: C      integer KDBNK(0:MDBNK)
83: C      integer IBNK(NBANK,NIBNK,2)
84: C      integer KIBNK(0:MIBNK)
85: C
86: C****
87: C
88: C      real*8  OPTI(IMPMAX), PATH(IMPMAX)
89: C      integer KDETP(IMPMAX), KLSFI(IMPMAX)
90: C
91: C**** cross section and geometry data
92: C
93: C      real    SMACI(IMPMAX,MB,NSMACI)
94: C      integer MMACI(IMPMAX,2)
95: C
96: C      integer KZMAT(NZONE)
97: C
98: C**** WORKING AREA *****
99: C
100: C      integer IBP(NBANK), IT0(NBANK), IGI(NBANK)
101: C      real*8  X(NBANK), Y(NBANK), Z(NBANK)
102: C      real*8  AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
103: C      real*8  A0(NBANK), B0(NBANK), C0(NBANK)
104: C      real*8  DW1(NBANK), DW2(NBANK), DW3(NBANK), DW4(NBANK),
105: C      &      DW5(NBANK), TI(NBANK)
106: C      real    W(NBANK)
107: C      real    R(2*NBANK), SMCW(NBANK,NSMICI)
108: C****
109: C      include '../shared/INC/_SIZES'
110: C      include '../shared/INC/_PGEOM'
111: C      include '../shared/INC/_PTALY0'
112: C      include 'INC/_SIZES2'
113: C      include 'INC/_FLAGS'
114: C      include 'INC/_STACK'
115: C      include 'INC/_PXSEC'
116: C      include 'INC/_KPIDS'
117: C      include 'INC/_CXSEC'
118: C      include 'INC/_SBANK'
119: C
120: C      .... constants ....
121: C
122: C      --- light speed --- (cm/s)
123: C
124: C      real*8 CLIGHT
125: C      parameter( CLIGHT = 2.99792458D10 )
126: C
127: C      --- neutron mass --- (gram)
128: C
129: C      real*8 NMASS
130: C      parameter( NMASS = 1.6749286D-24 )

```

src/mvp/suncl.f

```

131: C
132: C --- erg / eV --- electron charge
133: C
134:      real*8 EECHRG
135:      parameter( EECHRG = 1.60217733D-12 )
136: C
137: C --- neutron's mc**2 in eV
138: C
139:      real*8 MC2
140:      parameter( MC2 = NMASS*CLIGHT**2/EECHRG )
141: C
142:      DPI2 = 2.0D0*ACOS(-1.0D0)
143:      DPI4I = 1.0D0/(4*ACOS(-1.0D0))
144: C=====
145: C
146: C**** GATHER POSITION,ENERGY GROUP,ETC ****
147: C
148: check
149: c      write(6,*) ' Nothing is done in SUNCL'
150: c      return
151: check
152:      if ( JLATT.eq.0.or.JEIGN.eq.0.or.(JEIGN.ne.0.and.NBATCH.le.0) )
153:      & then
154: C
155: C ..... NO LATTICE
156: C
157:      do 100 I = 1, NSORS
158:      X(I) = XXX(LSORS(I),1)
159:      Y(I) = YYY(LSORS(I),1)
160:      Z(I) = ZZZ(LSORS(I),1)
161:      A0(I) = AAA(LSORS(I),1)
162:      B0(I) = BBB(LSORS(I),1)
163:      C0(I) = CCC(LSORS(I),1)
164: 100 continue
165: C
166:      else
167: C
168: C ..... LATTICE
169: C ***** TRANSFORMATION OF CORDINATES AND DIRECTION
170: C FROM CELL-ORDINATE SYSTEM TO LABORATORY SYSTEM
171: C (XXX,YYY,ZZZ) --> (X,Y,Z)
172: C
173:      JDIR = 1
174:      JLS = 0
175:      call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
176:      & NZONE, NSORS, NBANK, JDIR, JLS, LSORS,
177:      & X, Y, Z, A0, B0, C0,
178:      & XXX, YYY, ZZZ,
179:      & AAA, BBB, CCC,
180:      & LEVL, LZZ, LPOS, LCRS,
181:      & DBNK,KDBNK,MDBNK,
182:      & A(LKZMAT),A(LKDALT),A(LDALT),A(LNVLAT),A(LSZLAT),
183:      & A(LCELSZ),A(LIPLAT),A(LKLATT),A(LKSLAT),A(LLTYP),
184:      & A(LICTYP),A(LMLBZZ),
185:      & IBP )
186: C
187:      end if
188: C
189: C
190: C**** LOOP OVER DETECTORS *****
191: C
192:      do 200 ND = 1, NPDET
193: C
194:      if ( JPUSD(ND).eq.0 ) go to 200
195: C

```

```

196: C**** OBTAIN THE DISTANCE TO THE DETECTOR
197:      DXDA = XPDET(3,ND)
198:      DYDA = XPDET(4,ND)
199:      DZDA = XPDET(5,ND)
200:      RMIN = XPDET(1,ND)
201:      RMIN2 = RMIN*RMIN
202:      RMAX = XPDET(2,ND)
203:      RMA2 = RMAX*RMAX
204: C
205:      IZD = IPDET(ND,1)
206:      MAT = KZMAT(IZD)
207:      NBS = 0
208: C
209:      call RANU2( IRAND, R, NSORS,ICON )
210: C
211: *VOCL LOOP,NOVREC
212:      do 210 I = 1, NSORS
213:      DAT = DXDA - X(I)
214:      DBT = DYDA - Y(I)
215:      DCT = DZDA - Z(I)
216:      DDI = DAT*DAT + DBT*DBT + DCT*DCT
217: C
218: C
219: C Calculate contrirbution with probability RMIN2/DDI
220: C when DDI > RMIN**2
221: C
222:      if( DDI*R(I).gt.RMIN2 ) then
223:      W(I) = 0.0
224:      elseif ( DDI.gt.RMIN2 ) then
225:      W(I) = WWW(LSORS(I),1) / RMIN2 * DPI4I
226: C
227: C Bounded sphere approximation when DDI < RMAX**2
228: C
229:      elseif ( DDI.lt.RMA2.and.MAT.gt.0 ) then
230:      NBS = NBS + 1
231:      W(I) = WWW(LSORS(I),1) * DPI4I
232:      else
233: C
234: C WEIGHT / (DISTANCE)**2 / (4*PAI)
235: C
236:      W(I) = WWW(LSORS(I),1) / DDI * DPI4I
237:      endif
238: C
239:      DI(I) = SQRT(DDI)
240: C
241:      if ( DI(I).gt.0. ) then
242:      AA(I) = -DAT/DI(I)
243:      BB(I) = -DBT/DI(I)
244:      CC(I) = -DCT/DI(I)
245:      else
246:      AA(I) = -A0(I)
247:      BB(I) = -B0(I)
248:      CC(I) = -C0(I)
249:      end if
250: 210 continue
251: C
252: C ..... APPLY ANGULAR DISTRIBUTION FUNCTION TO WEIGHT WW .....
253: C
254:      KSRC = JSPDT(ND)
255: C
256: C *** KSRC = 0 : ISOTROPIC FACTOR = 1.0 OR 2./(DUMAX-DUMIN)
257: C *** KSRC > 0 : NON ISOTROPIC
258: C *** KSRC < 0 : NO CONTRIBUTION
259: C
260:      if ( KSRC.eq.0 ) then

```

src/mvp/suncl.f

```

261:      DFACT = 2./(DUMAX-DUMIN)
262: C
263:      do 220 I = 1, NSORS
264:        if ( -CC(I).le.DUMAX.and.-CC(I).ge.DUMIN ) then
265:          W(I) = W(I)*DFACT
266:        else
267:          W(I) = 0.0
268:        end if
269:      220 continue
270: C
271: C
272: C CALCULATE FACTOR BY USING AA,BB,CC
273: C else if ( KSRC.gt.0 ) then
274:   DELMU = 1.D-1
275:   CONT = 1.D0 - DELMU / DPI2
276:   DFACT = 2.D0 * DPI2 / DELMU
277: C
278:   do 230 I = 1, NSORS
279:     XMU = A0(I)*AA(I)+B0(I)*BB(I)+C0(I)*CC(I)
280:     XMU = -XMU
281:     if ( XMU.ge.CONT ) then
282:       W(I) = W(I)*DFACT
283:     else
284:       W(I) = 0.0
285:     end if
286:   230 continue
287: C
288: C
289:   else
290:     do 240 I = 1, NSORS
291:       W(I) = 0.0
292:     240 continue
293:   end if
294: C
295: C .... TIME CUT OFF .....
296: C
297:   if ( JTIME.ne.0 ) then
298:     do 250 I = 1, NSORS
299:       DDE = EEE(LSORS(I),1)
300:       VEL = CLIGHT*SQRT(DDE*(DDE+2*MC2)) / (DDE+MC2)
301:       TI(I) = TTT(LSORS(I),1) + DI(I) / VEL
302:     250 continue
303:     if ( JPTIM.eq.0 ) then
304:       do 260 I = 1, NSORS
305:         if ( TI(I).gt.TCUT ) then
306:           W(I) = 0.0
307:           TI(I) = TCUT
308:         end if
309:       260 continue
310:     else
311:       do 270 I = 1, NSORS
312:         if ( TI(I).ge.TCUT ) then
313:           ITREP = TI(I) / TCUT
314:           TI(I) = TI(I) - TCUT*ITREP
315:         end if
316:       270 continue
317:     end if
318: C
319:     call BS0ISD( TIMEB, NTIME+1, TI, IT0, NSORS )
320: C
321:     do 280 I = 1, NSORS
322:       IT0(I) = MAX(1,MIN(NTIME,IT0(I)))
323:     280 continue
324:   end if
325: C

```

```

326: C
327: C **** SEND PARTICLES TO FREE-FLIGHT OR LATIC STACK
328: C                                     FOR IMAGINARY PARTICLES ****
329: C
330: C
331: C
332: C ..... CHECK NUMBER OF PARTICLES HAVING NON-ZERO ANGULAR FACTOR ...
333: C
334: C       IBP : BANK INDEX
335: C
336:       NN = 0
337: *VOCL LOOP,NOVREC
338:       do 290 I = 1, NSORS
339:         if ( W(I).ne.0.0 ) then
340:           NN = NN + 1
341:           IBP(NN) = LSORS(I)
342:           DI(NN) = DI(I)
343:           AA(NN) = AA(I)
344:           BB(NN) = BB(I)
345:           CC(NN) = CC(I)
346:           W(NN) = W(I)
347:           if ( JTIME.ne.0 ) IT0(NN) = IT0(I)
348:         end if
349:       290 continue
350: C
351: C
352: C=====
353: C
354: C       SEND PARTICLES TO STACK FOR IMAGINARY PARTICLES
355: C
356: C=====
357: C
358: C
359:       LVL = IPDET(ND,2)
360:       if ( JLATT.ne.0.and.LVL.gt.0 ) then
361:         if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
362:         LP = 2 + 3*LVL
363:         DXDA = XPDET(LP+1,ND)
364:         DYDA = XPDET(LP+2,ND)
365:         DZDA = XPDET(LP+3,ND)
366:       end if
367: C
368:       300 continue
369: C
370:       if ( NN.gt.0 ) then
371: C
372:         NNN = MIN(NN,NDIMPT)
373: C
374:         if ( NNN.gt.0 ) then
375:           NDIMPT = NDIMPT - NNN
376: C
377: C **** SEND IMAGINARY PARTICLES TO FREE-FLIGHT STACK
378: C
379:           N = IMNFL(NZONE+1)
380:           N1 = NN - NNN
381: *VOCL LOOP,NOVREC
382:           do 310 J = 1, NNN
383:             J1 = N1 + J
384:             IMSFL(N+J) = IMDED(NDIMPT+J)
385:             IMZFL(N+J) = IZD
386:             XXX(IMSFL(N+J),2) = DXDA
387:             YYY(IMSFL(N+J),2) = DYDA
388:             ZZZ(IMSFL(N+J),2) = DZDA
389:             AAA(IMSFL(N+J),2) = AA(J1)
390:             BBB(IMSFL(N+J),2) = BB(J1)

```

src/mvp/suncl.f

```

391:      CCC(IMSFL(N+J),2) = CC(J1)
392:      WWW(IMSFL(N+J),2) = W(J1)
393:      EEE(IMSFL(N+J),2) = EEE(IBP(J1),1)
394:      IGG(IMSFL(N+J),2) = IGG(IBP(J1),1)
395:      TTT(IMSFL(N+J),2) = TTT(IBP(J1),1)
396:      PATH(IMSFL(N+J)) = DI(J1)
397:      OPTI(IMSFL(N+J)) = 0.0
398:      KDETP(IMSFL(N+J)) = ND
399:      KLSFI(IMSFL(N+J)) = 0
400:      if ( JTIME.ne.0 ) then
401:        ITT(IMSFL(N+J),2) = IT0(J1)
402:      end if
403:      if ( KIBNK(5).ne.0 ) then
404:        IBNK(IMSFL(N+J),KIBNK(5),2) = -1
405:      end if
406: 310 continue
407: C
408: C      NN2 = NNN
409: C      JSTG = 0
410: C
411: C
412: C-----
413: C
414: C      In LATTICE geometry, the following parameters should be set.
415: C      LEVL, LZZI, LPOSI, LCRSI
416: C      DBNK, IBNK (JFISX.ne.0)
417: C      JSTG : flag to indicate the detector is located in STG region.
418: C      NN2 : number of remaining particles when JFISX.ne.0
419: C
420: C-----
421: C
422:      if ( JLATT.ne.0 ) then
423:        call LATDW2(IOW, JDEBG, JHLAT, JFISX,
424:          N      IMPMAX,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
425:          N      DINF ,DEPS ,
426:          S      IMSFL(N+1),NNN ,JSTG ,NN2 ,
427:          I      IPDET(ND,2),IPDT2(1,1,ND),IPDT2(1,2,ND),
428:          I      IPDT2(1,3,ND),XPDET(3,ND),
429:          B      AAA(1,2),BBB(1,2),CCC(1,2),
430:          B      LEVL(1,2),LZZI ,LPOSI ,LCRSI ,
431:          B      DBNK(1,1,2),KDBNK ,MDBNK ,IBNK(1,1,2),KIBNK ,MIBNK ,
432:          S      IMDED ,NDIMPT,
433:          G      A(LSDA) ,A(LKZDA) ,A(LKZAA) ,
434:          G      A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LMLBZZ),A(LCELSZ),
435:          G      A(LSZLAT),A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),
436:          G      A(LNLVLT),A(LIPCEL),A(LLTYP) ,A(LICTYP),A(LKDALT),
437:          G      A(LDALT) ,A(LKZREG),
438:          W      DW1 ,DW2 ,DW3 ,AA(N1+1),BB(N1+1),CC(N1+1),
439:          W      DI(N1+1),DW4 ,DW5 ,W(N1+1),IGI(N1+1) )
440:      end if
441: C
442: C      .... ADJUST PARTICLE NUMBERS IN STACK ....
443: C      AND NUMBER OF IMAGINARY PARTICLE
444: C
445:      if ( JSTG.eq.0 ) then
446:        IMNFL(IZD) = IMNFL(IZD) + NN2
447:        IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
448:      else
449: C
450: C
451:      NL      = IMNLT(NLBZ+1)
452: *VOCL LOOP,NOVREC
453:      do 320 J = 1, NN2
454:        IMSLT(NL+J) = IMSFL(N+J)
455:        IMZLT(NL+J) = IZD

```

```

456: 320      continue
457:        IMNLT(IZD) = IMNLT(IZD) + NN2
458:        IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
459:      end if
460: C
461:      NN      = NN - NNN
462:      NIMPT   = NIMPT + NN2
463:    end if
464: C
465: C *** prepare cross sections *****
466: C
467:      JAMXCT = 0
468:      JMIC   = 1
469:      if ( JNP.eq.1 ) then
470:        call GMACNX( IRAND, NN2, IMSFL(N+1), MAT,
471:          &      JMIC, JAMXCT, JNP, JGAMM, JDEBG, IMPMAX,
472:          &      EEE(1,2), MB, NUC, NSTAL,
473:          &      NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
474:          &      NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTLI),
475:          &      A(LLPDEN), A(LINUCT), A(LDENST), NMAT, NUC, NMT,
476:          &      A(LCX), MCX, A(LKLIB1), A(LKLIB2), A(LXLIB1),
477:          &      A(LCRES), A(LKCRSI), MCRES, NNCSI, A(LINCSI),
478:          W      DW1, DW2, DW3, DW4, DW5, AA(N1+1),
479:          W      BB(N1+1), CC(N1+1), R, SMCW )
480:      end if
481:      if ( JNP.eq.2 ) then
482:        call GMACNX( IRAND, NN2, IMSFL(N+1), MAT,
483:          &      JMIC, JAMXCT, JNP, JGAMM, JDEBG, IMPMAX,
484:          &      EEE(1,2), MB, NUC, NSTAL,
485:          &      NSMACI, SMACI, A(LLMACI), H(LKMACI), MMACI ,
486:          &      NSMICI, H(LSMICI), A(LLMICI), H(LKSPII), H(LSGTLI),
487:          &      A(LLPDNP), A(LIATMT), A(LDNSTP), NMAT, NPATOM, NMTP,
488:          &      A(LCXP), MCXP, A(LKLB1), A(LKLB2), A(LXLB1),
489:          &      A(LCRES), A(LKCRSI), MCRES, NNCSI, A(LINCSI),
490:          W      DW1, DW2, DW3, DW4, DW5, AA(N1+1),
491:          W      BB(N1+1), CC(N1+1), R, SMCW )
492:      end if
493: C
494: C .... BOUNDED SPHERE APPROXIMATION
495: C
496:      if ( MAT.gt.0.and.NBS.gt.0 ) then
497:        *VOCL LOOP,NOVREC
498:        do 330 J = 1, NN2
499:          IP      = IMSFL(N+J)
500:          if ( PATH(IP).le.RMAX ) then
501:            RMD   = SMACI(IP,MMACI(IP,1),1) * RMAX
502:            RMI   = 1./RMD
503:            C2    = RMA2*
504:              (RMI*RMI*2.+(1.+2.0*RMI)/(1.-EXP(RMD)))
505:            &
506:            WWW(IP,2) = WWW(IP,2) /C2
507:          end if
508: 330      continue
509:        end if
510: C
511: C
512: C *** CALL TRACKING CONTROL ROUTINE FOR IMAGINARY PARTICLES
513: C      ( VALUES OF NIMPT,NDIMPT, IMNFL MAY BE CHANGED )
514: C
515:      if ( NDIMPT.eq.0 ) then
516:        call NXTEE( 2, A, H, NDIMPT, NIMPT )
517:      end if
518:      go to 300
519: C
520:      end if

```


src/mvp/suncl.f

```
521: C
522: 200 continue
523: C
524:     return
525:     end
```

SAFE

src/mvp/talkp.f

```

1:      subroutine TALKP( IOW,  JEIGN, JREAC, JREGN, JZONE, JRESP, FLX,
2:      &                 NN,    IEG,   IRG,   IBP,   IMB,   DNF,  NGROUP,
3:      &                 NGP1, NREG,  NRESP, NUC,   NBANK, NSMIC, NSMAC,
4:      &                 NSTAL, MB,    IZN,   DNZON, NZONE, LEMIC, NEMIC,
5:      &                 RESP, SMIC, LMIC, SMAC, LMAC, SGTAL, FLXX,
6:      &                 REXX, RMIC, DNFLX, RSTAL, WC1,  WC2,
7:      &                 NPTDS, WWD,   WCTRP, JPTTR, JPTNU, KZREG,
8:      &                 WWD2, NPTCS, WWR,   MAXDA, DELA ,
9:      &                 SMACP,
10:     &                 KZMAT, NUCPT, NMAT,  MNUC, LPDEN, INUCT, DENST,
11:     &                 WKPN, WWT,  WKPT, WKPD, JPTRT, JPTMP, JPDEN,
12:     &                 NGSP, WSD,   WSC,  NOCS, NTPT,  NORDDS, NOPSDS,
13: c+beff2
14:     &                 JBEFF, WWB,  WSB,  RNU,  NPTBE,
15:     &                 WWBD, WSBD, WWLD, WSLD)
16: c-beff2
17: C=<MVP>=====
18: C PURPOSE: Tally for keff perturbation (NEUTRON)
19: C CALLED IN: NEUTR
20: C Copied from tally.f and modified by Y.Nagaya (1999/11/24)
21: C UPDATE:
22: C 06 Nov 2006: Add higher-order differential operator sampling method.
23: C               (Y.Nagaya)
24: C 11 Oct 2007: Add dummy arguments JBEFF, WWB, WSB, RNU.
25: C               Add beta-effective calculation with correlated sampling.
26: C               (Y.Nagaya)
27: C 12 Oct 2007: Fix a bug for perturbation calculation. (Y.Nagaya)
28: C               JPTTR(NREG,NPTDS) -> JPTTR(NREG,NPTDS+NPTCS)
29: C 23 Apr 2008: (Y.Nagaya) Modified the calculation scheme for
30: C               beta-effective calculation with correlated sampling.
31: C               The source weight for delayed neutrons are reduced.
32: C 01 Jun 2009: (Y.Nagaya) Implemented beta-effective calculation
33: C               capability with differential operator sampling.
34: C 30 Aug 2010: (Y.Nagaya) Extend the order of differential operator
35: C               sampling up to 8th.
36: C 28 Nov 2012: (Y.Nagaya) Implemented Lambda calculation capability
37: C               with differential operator sampling.
38: C=====
39: C
40: C JREAC : 1 = TAKE TALLYS OF REACTION RATES OTHER THAN FLUX,
41: C           TOTAL & PRODUCTION
42: C           0 = TAKE TALLYS OF ONLY FLUX, TOTAL & PRODUCTION
43: C* JZONE : 1 = ALL ZONES (FROM FLIGHT)
44: C*         -1 = ALL ZONES (FROM NEUTR)
45: C*         0 = SINGLE ZONE (FROM FLIONE)
46: C JREGN : 1 = MULTI REGION (FROM FLIGHT & FLIONE )
47: C         -1 = MULTI REGION (FROM NEUTR)
48: C         0 = SINGLE REGION (FROM FLIONE)
49: C JZONE : 1 = MULTI ZONE
50: C         0 = SINGLE ZONE
51: C FLX   : FLUX      NN : NUMBER OF PARTICLES
52: C IEG   : GROUP #
53: C IRG   : REGION #
54: C IBP   : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
55: C         JREAC=1 OR JEIGN.NE.0
56: C IMB   : VALUE OF MMAC(IBP,1) FOR EACH PARTICLE (NECESSARY ONLY WHEN
57: C         JREAC=1 OR JEIGN.NE.0
58: C IZN   : ZONE #
59: C DNF   : WORK AREA (ADDED FEB.19,1993)
60: C=====
61:      implicit real*8(A-H,O-Z)
62: C
63:      real*8 FLX(NN), DNF(NN)
64:      integer IEG(NN), IBP(NN), IMB(NN), IRG(NN), IZN(NN)
65: C

```

```

66: C ... FLXX : FLUX TALLY BY ARABITRARY ESTIMATOR (FLTR /FLCL) .....
67: C ... REXX : FLUX TALLY BY ARABITRARY ESTIMATOR (RETR /RECL) .....
68: C WC1 : Production for dk/da
69: C WC2 : LOSS
70: C
71: C .... TEMPORARY NAMES FOR FLUX & REACTION RATE ARRAY ....
72: C & EIGENVALUE TALLY
73: C
74:      real*8 FLXX(NGROUP,NREG), REXX(NREG,NRESP), WC1, WC2
75: C
76: C
77: C
78:      real*8 RMIC(NGROUP,NREG,NUC,NEMIC), RSTAL(NREG,NSTAL),
79:      & DNFLX(NGROUP,NREG,NUC)
80:      real RESP(NGROUP,NRESP), SGTAL(NBANK,NSTAL)
81:      real SMIC(NBANK,NUC,NSMIC), SMAC(NBANK,MB,NSMAC)
82:      real DNZON(NUC,NZONE)
83:      real SMACP(NBANK,MB,NSMAC)
84:      integer LMAC(8), LMIC(8), LEMIC(8,2,2)
85: C
86: C
87: c**** GEOMETRY ARRAY DATA
88: C
89:      integer KZREG(NZONE)
90:      integer KZMAT(NZONE)
91: C
92:      real*8 WWD(NBANK,NPTDS,NORDDS), WWD2(NBANK,NPTDS)
93:      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
94:      real*8 WWR(NBANK,NPTCS)
95:      real*8 WSC(NBANK,0:NGSP,NPTCS)
96:      real*8 WCTRP(NUCPT,7,NTPT)
97:      integer JPTTR(NREG,NPTDS+NPTCS), JPTNU(NUC,NPTDS)
98:      real DELA(MAXDA,NPTDS+NPTCS)
99:      real*8 WKPD(NBANK,NPTDS)
100:      real*8 WKPN(NUCPT,NBANK,NPTDS)
101:      real*8 WWT(NBANK,NPTDS)
102:      real*8 WKPT(NBANK,NPTDS)
103: c+beff2
104:      real*8 WWB(NBANK)
105:      real*8 WSB(NBANK,0:NGSP)
106:      real RNU(NBANK,NUC,3)
107:      real*8 WWBD(NBANK)
108:      real*8 WSBD(NBANK,NGSP)
109:      real*8 WWLD(NBANK)
110:      real*8 WSLD(NBANK,NGSP)
111: C
112:      real*8 RRNU
113: c-beff2
114: C
115: C
116: C **** MATERIAL DATA *****
117: C
118:      real DENST(*)
119:      integer MNUC(NMAT), INUCT(*), LPDEN(NMAT+1)
120: C
121: C
122:      ISCSB = NOCS + (NGSP+2)*NPTCS
123:      IPBDS = ISCSB + (NGSP+2) + 1 ! pointer of beff with diff-sampling
124:      IPLDS = IPBDS + (NGSP+1) ! pointer of lambda with diff. samp.
125: C
126: C***** TALLYS FOR ONE REGION & ONE ZONE *****
127: C
128: C
129:      if ( JREGN.eq.0 ) then
130: C

```

src/mvp/talkp.f

```

131:      IR1      = IRG(1)
132:      IZ1      = IZN(1)
133: C
134: C      .... FLUX TALLY ....
135: C
136: C      if ( NGP1.gt.1 ) then
137: C          do 100 I = 1, NN
138: C              FLXX(IEG(I),IR1) = FLXX(IEG(I),IR1) + FLX(I)
139: C          continue
140: C      else
141: C          IE      = IEG(1)
142: C          do 110 I = 1, NN
143: C              FLXX(IE,IR1) = FLXX(IE,IR1) + FLX(I)
144: C          continue
145: C      end if
146: C
147: C      .... EIGENVALUE PROBLEM AND NON VOID REGION ....
148: C
149: C      if ( JEIGN.ne.0.and.IMB(1).gt.0 ) then
150: C          do 120 I = 1, NN
151: C              SNUF = SMAC(IBP(I),IMB(I),LMAC(2))
152: C              WC1  = WC1 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
153: C              WC2  = WC2 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(8))
154: C
155: C -----
156: C TEMPERATURE
157: C -----
158: C      if( JPTMP.ne.0 ) then
159: C          if( SNUF.gt.0.d0 ) then
160: C
161: C              do 126 II = 1, NUCPT
162: C                  WKPT(IBP(I),1) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
163: C                  WCTRP(II,2,1) = WCTRP(II,2,1)
164: C                  &      + ( WKPT(IBP(I),1)+WWT(IBP(I),1) ) * FLX(I) * SNUF
165: C
166: C                  WKPT(IBP(I),2) = 0.0d0
167: C                  WCTRP(II,2,2) = WCTRP(II,2,2)
168: C                  &      + ( WKPT(IBP(I),2)+WWT(IBP(I),2) ) * FLX(I) * SNUF
169: C
170: C                  WKPT(IBP(I),3) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
171: C                  WCTRP(II,2,3) = WCTRP(II,2,3)
172: C                  &      + ( WKPT(IBP(I),3)+WWT(IBP(I),3) ) * FLX(I) * SNUF
173: C
174: C                  WKPT(IBP(I),4) = 0.0d0
175: C                  WCTRP(II,2,4) = WCTRP(II,2,4)
176: C                  &      + ( WKPT(IBP(I),4)+WWT(IBP(I),4) ) * FLX(I) * SNUF
177: C
178: C                  WKPT(IBP(I),5) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
179: C                  WCTRP(II,2,5) = WCTRP(II,2,5)
180: C                  &      + ( WKPT(IBP(I),5)+WWT(IBP(I),5) ) * FLX(I) * SNUF
181: C
182: C                  WKPT(IBP(I),6) = 0.0d0
183: C                  WCTRP(II,2,6) = WCTRP(II,2,6)
184: C                  &      + ( WKPT(IBP(I),6)+WWT(IBP(I),6) ) * FLX(I) * SNUF
185: C
186: C                  WKPT(IBP(I),7) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
187: C                  WCTRP(II,2,7) = WCTRP(II,2,7)
188: C                  &      + ( WKPT(IBP(I),7)+WWT(IBP(I),7) ) * FLX(I) * SNUF
189: C          c 126      continue
190: C          end if
191: C      end if
192: C -----
193: C DENSITY & NUMBER DENSITY
194: C -----
195: C      if( JPDEN.ne.0 ) then

```

```

196:
197:      if( SNUF.gt.0.d0 ) then
198:          do 121 INC = 1, NUCPT
199:
200:              do 221 IPT = 1, NPTDS
201: C          ... density perturbation ...
202: C              DELTA = DBLE(JPTTR(KZREG(IZN(I)),IPT))
203: C
204: C              DELTA = 0.0d0
205: C              MM = KZMAT(IZN(I))
206: C              do 151 J = 1, MNUC(MM)
207: C                  IDN = LPDEN( MM ) - 1 + J
208: C                  KN  = INUCT( IDN )
209: C                  RHO = DENST( IDN )
210: C                  DELTA = DELTA
211: C                  &      + dble(JPTTR(IRG(I),IPT))
212: C                  &      * dble(JPTNU(KN,IPT))
213: C                  &      * RHO*SMIC( IBP(I),KN,LMIC(2) ) / SNUF
214: C          151      continue
215: C
216: C          ... 1st order ...
217: C              WCTRP( INC,2,IPT ) = WCTRP( INC,2,IPT )
218: C              &      + ( DELTA + WWD( IBP(I),IPT,1 ) )
219: C              &      * FLX(I) * SNUF
220: C          ... 2nd order ...
221: C              WCTRP( INC,2,NPTDS+IPT ) =
222: C              WCTRP( INC,2,NPTDS+IPT )
223: C              &      + ( ( DELTA + WWD( IBP(I),IPT,1 ) )**2
224: C              &      - DELTA + WWD2( IBP(I),IPT ) )
225: C              &      * FLX(I) * SNUF
226: C          ... 3rd & 4th order ...
227: C              CONT1 = WWD( IBP(I),IPT,1 ) + DELTA
228: C              D1CT1 = WWD( IBP(I),IPT,2 ) - DELTA
229: C              D2CT1 = WWD( IBP(I),IPT,3 ) + DELTA*2.d0
230: C              D3CT1 = WWD( IBP(I),IPT,4 ) - DELTA*6.d0
231: C              D4CT1 = WWD( IBP(I),IPT,5 ) + DELTA*24.d0
232: C              D5CT1 = WWD( IBP(I),IPT,6 ) - DELTA*120.d0
233: C              D6CT1 = WWD( IBP(I),IPT,7 ) + DELTA*720.d0
234: C              D7CT1 = WWD( IBP(I),IPT,8 ) - DELTA*5040.d0
235: C
236: C              CONT2 = D1CT1 + CONT1*CONT1
237: C              D1CT2 = D2CT1 + 2.d0*CONT1*D1CT1
238: C              D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
239: C              &      + 2.d0*CONT1*D2CT1
240: C              D3CT2 = D4CT1 + D3CT1*CONT1
241: C              &      + 3.d0*D2CT1*D1CT1
242: C              &      + 3.d0*D1CT1*D2CT1
243: C              &      + CONT1*D3CT1
244: C              D4CT2 = D5CT1 + D4CT1*CONT1
245: C              &      + 4.d0*D3CT1*D1CT1
246: C              &      + 6.d0*D2CT1*D2CT1
247: C              &      + 4.d0*D1CT1*D3CT1
248: C              &      + CONT1*D4CT1
249: C              D5CT2 = D6CT1 + D5CT1*CONT1
250: C              &      + 5.d0*D4CT1*D1CT1
251: C              &      +10.d0*D3CT1*D2CT1
252: C              &      +10.d0*D2CT1*D3CT1
253: C              &      + 5.d0*D1CT1*D4CT1
254: C              &      + CONT1*D5CT1
255: C              D6CT2 = D7CT1 + D6CT1*CONT1
256: C              &      + 6.d0*D5CT1*D1CT1
257: C              &      +15.d0*D4CT1*D2CT1
258: C              &      +20.d0*D3CT1*D3CT1
259: C              &      +15.d0*D2CT1*D4CT1
260: C              &      + 6.d0*D1CT1*D5CT1

```

src/mvp/talkp.f

```

261:      &                + CONT1*D6CT1
262:
263:      CONT3 = D1CT2 + CONT2*CONT1
264:      D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
265:      D2CT3 = D3CT2 + D2CT2*CONT1
266:      &                + 2.d0*D1CT2*D1CT1
267:      &                + CONT2*D2CT1
268:      D3CT3 = D4CT2 + D3CT2*CONT1
269:      &                + 3.d0*D2CT2*D1CT1
270:      &                + 3.d0*D1CT2*D2CT1
271:      &                + CONT2*D3CT1
272:      D4CT3 = D5CT2 + D4CT2*CONT1
273:      &                + 4.d0*D3CT2*D1CT1
274:      &                + 6.d0*D2CT2*D2CT1
275:      &                + 4.d0*D1CT2*D3CT1
276:      &                + CONT2*D4CT1
277:      D5CT3 = D6CT2 + D5CT2*CONT1
278:      &                + 5.d0*D4CT2*D1CT1
279:      &                +10.d0*D3CT2*D2CT1
280:      &                +10.d0*D2CT2*D3CT1
281:      &                + 5.d0*D1CT2*D4CT1
282:      &                + CONT2*D5CT1
283:
284:      CONT4 = D1CT3 + CONT3*CONT1
285:      D1CT4 = D2CT3 + D1CT3*CONT1 + CONT3*D1CT1
286:      D2CT4 = D3CT3 + D2CT3*CONT1
287:      &                + 2.d0*D1CT3*D1CT1
288:      &                + CONT3*D2CT1
289:      D3CT4 = D4CT3 + D3CT3*CONT1
290:      &                + 3.d0*D2CT3*D1CT1
291:      &                + 3.d0*D1CT3*D2CT1
292:      &                + CONT3*D3CT1
293:      D4CT4 = D5CT3 + D4CT3*CONT1
294:      &                + 4.d0*D3CT3*D1CT1
295:      &                + 6.d0*D2CT3*D2CT1
296:      &                + 4.d0*D1CT3*D3CT1
297:      &                + CONT3*D4CT1
298:
299:      CONT5 = D1CT4 + CONT4*CONT1
300:      D1CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
301:      D2CT5 = D3CT4 + D2CT4*CONT1
302:      &                + 2.d0*D1CT4*D1CT1
303:      &                + CONT4*D2CT1
304:      D3CT5 = D4CT4 + D3CT4*CONT1
305:      &                + 3.d0*D2CT4*D1CT1
306:      &                + 3.d0*D1CT4*D2CT1
307:      &                + CONT4*D3CT1
308:
309:      CONT6 = D1CT5 + CONT5*CONT1
310:      D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
311:      D2CT6 = D3CT5 + D2CT5*CONT1
312:      &                + 2.d0*D1CT5*D1CT1
313:      &                + CONT5*D2CT1
314:
315:      CONT7 = D1CT6 + CONT6*CONT1
316:      D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1
317:
318:      CONT8 = D1CT7 + CONT7*CONT1
319:
320:      WCTRP( INC,2,2*NPTDS+IPT) =
321:      &      WCTRP( INC,2,2*NPTDS+IPT)
322:      &      + CONT3*FLX(I)*SNUF
323:
324:      WCTRP( INC,2,3*NPTDS+IPT) =
325:      &      WCTRP( INC,2,3*NPTDS+IPT)

```

```

326:      &                + CONT4*FLX(I)*SNUF
327:
328:      WCTRP( INC,2,4*NPTDS+IPT) =
329:      &      WCTRP( INC,2,4*NPTDS+IPT)
330:      &      + CONT5*FLX(I)*SNUF
331:
332:      WCTRP( INC,2,5*NPTDS+IPT) =
333:      &      WCTRP( INC,2,5*NPTDS+IPT)
334:      &      + CONT6*FLX(I)*SNUF
335:
336:      WCTRP( INC,2,6*NPTDS+IPT) =
337:      &      WCTRP( INC,2,6*NPTDS+IPT)
338:      &      + CONT7*FLX(I)*SNUF
339:
340:      WCTRP( INC,2,7*NPTDS+IPT) =
341:      &      WCTRP( INC,2,7*NPTDS+IPT)
342:      &      + CONT8*FLX(I)*SNUF
343: c ... source perturbation ...
344:
345:      do ISP = 0, NGSP
346:      WCTRP( INC,2,(ISP+NORDDS)*NPTDS+IPT) =
347:      &      WCTRP( INC,2,(ISP+NORDDS)*NPTDS+IPT)
348:      &      + WSD( IBP(I),ISP,IPT,1)*FLX(I)*SNUF
349:      end do
350:
351:      do ISP = 1, NGSP
352:      IP = (NORDDS+NGSP+ISP)*NPTDS+IPT
353:      WCTRP( INC,2,IP) = WCTRP( INC,2,IP)
354:      &      + (2.d0*CONT1*WSD( IBP(I),ISP,IPT,1)
355:      &      + WSD( IBP(I),ISP,IPT,2))
356:      &      *FLX(I)*SNUF
357:      end do
358:
359:      do ISP = 1, NGSP
360:      IP = (NORDDS+2*NGSP+ISP)*NPTDS+IPT
361:      WCTRP( INC,2,IP) = WCTRP( INC,2,IP)
362:      &      + ( WSD( IBP(I),ISP,IPT,3)
363:      &      + 3.d0*CONT1*WSD( IBP(I),ISP,IPT,2)
364:      &      + 3.d0*CONT2*WSD( IBP(I),ISP,IPT,1)
365:      &      )*FLX(I)*SNUF
366:      end do
367:
368:      do ISP = 1, NGSP
369:      IP = (NORDDS+3*NGSP+ISP)*NPTDS+IPT
370:      WCTRP( INC,2,IP) = WCTRP( INC,2,IP)
371:      &      + ( WSD( IBP(I),ISP,IPT,4)
372:      &      + 4.d0*CONT1*WSD( IBP(I),ISP,IPT,3)
373:      &      + 6.d0*CONT2*WSD( IBP(I),ISP,IPT,2)
374:      &      + 4.d0*CONT3*WSD( IBP(I),ISP,IPT,1)
375:      &      )*FLX(I)*SNUF
376:      end do
377:
378:      do ISP = 1, NGSP
379:      IP = (NORDDS+4*NGSP+ISP)*NPTDS+IPT
380:      WCTRP( INC,2,IP) = WCTRP( INC,2,IP)
381:      &      + ( WSD( IBP(I),ISP,IPT,5)
382:      &      + 5.d0*CONT1*WSD( IBP(I),ISP,IPT,4)
383:      &      +10.d0*CONT2*WSD( IBP(I),ISP,IPT,3)
384:      &      +10.d0*CONT3*WSD( IBP(I),ISP,IPT,2)
385:      &      + 5.d0*CONT4*WSD( IBP(I),ISP,IPT,1)
386:      &      )*FLX(I)*SNUF
387:      end do
388:
389:      do ISP = 1, NGSP
390:      IP = (NORDDS+5*NGSP+ISP)*NPTDS+IPT

```

src/mvp/talkp.f

```

391:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
392:      &      + ( WSD(IBP(I),ISP,IPT,6)
393:      &      + 6.d0*CONT1*WSD(IBP(I),ISP,IPT,5)
394:      &      +15.d0*CONT2*WSD(IBP(I),ISP,IPT,4)
395:      &      +20.d0*CONT3*WSD(IBP(I),ISP,IPT,3)
396:      &      +15.d0*CONT4*WSD(IBP(I),ISP,IPT,2)
397:      &      + 6.d0*CONT5*WSD(IBP(I),ISP,IPT,1)
398:      &      )*FLX(I)*SNUF
399:      end do
400:
401:      do ISP = 1, NGSP
402:      IP = (NORDD5+6*NGSP+ISP)*NPTDS+IPT
403:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
404:      &      + ( WSD(IBP(I),ISP,IPT,7)
405:      &      + 7.d0*CONT1*WSD(IBP(I),ISP,IPT,6)
406:      &      +21.d0*CONT2*WSD(IBP(I),ISP,IPT,5)
407:      &      +35.d0*CONT3*WSD(IBP(I),ISP,IPT,4)
408:      &      +35.d0*CONT4*WSD(IBP(I),ISP,IPT,3)
409:      &      +21.d0*CONT5*WSD(IBP(I),ISP,IPT,2)
410:      &      + 7.d0*CONT6*WSD(IBP(I),ISP,IPT,1)
411:      &      )*FLX(I)*SNUF
412:      end do
413:
414:      do ISP = 1, NGSP
415:      IP = (NORDD5+7*NGSP+ISP)*NPTDS+IPT
416:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
417:      &      + ( WSD(IBP(I),ISP,IPT,8)
418:      &      + 8.d0*CONT1*WSD(IBP(I),ISP,IPT,7)
419:      &      +28.d0*CONT2*WSD(IBP(I),ISP,IPT,6)
420:      &      +56.d0*CONT3*WSD(IBP(I),ISP,IPT,5)
421:      &      +70.d0*CONT4*WSD(IBP(I),ISP,IPT,4)
422:      &      +56.d0*CONT5*WSD(IBP(I),ISP,IPT,3)
423:      &      +28.d0*CONT6*WSD(IBP(I),ISP,IPT,2)
424:      &      + 8.d0*CONT7*WSD(IBP(I),ISP,IPT,1)
425:      &      )*FLX(I)*SNUF
426:      end do
427: 221      continue
428:
429: c === correlated sampling ===
430:
431:      do 115 IPT = 1, NPTCS
432:      DLT01 = dble(JPTTR(KZREG(IZN(I)),NPTDS+IPT))
433:      DLT01 = dble(JPTTR(IRG(I),NPTDS+IPT))
434:      DELA1 = dble(DELA(1,NPTDS+IPT))
435:      WCTRP(INC,2,IPT+N0CS) = WCTRP(INC,2,IPT+N0CS)
436:      &      + ( (1.d0+DELA1*DLT01)*WWR(IBP(I),IPT)
437:      &      - 1.d0 )*FLX(I)*SNUF
438: c ... source perturbation ...
439:      do 215 ISP = 0, NGSP
440:      WCTRP(INC,2,N0CS+(ISP+1)*NPTCS+IPT) =
441:      &      WCTRP(INC,2,N0CS+(ISP+1)*NPTCS+IPT)
442:      &      + ( (1.d0+DELA1*DLT01)
443:      &      *WSC(IBP(I),ISP,IPT) )
444:      &      *FLX(I)*SNUF
445: 215      continue
446: 115      continue
447:
448: 121      continue
449:      end if
450:      end if
451: c+beff2
452:      if (NPTBE.gt.0) then
453:      if (SNUF.gt.0.d0) then
454:      do 124 INC = 1, NUCPT
455:      RRNU = 0.d0

```

```

456:      SNUFD = 0.d0
457:      MM = KZMAT(IZN(I))
458:      do J = 1, MNUC(MM)
459:      IDN = LPDEN(MM) - 1 + J
460:      KN = INUCT(IDN)
461:      RHO = DENST(IDN)
462:      if (RRNU(IBP(I),KN,1).gt.0.0) then
463:      RRNU = RRNU
464:      &      + RRNU(IBP(I),KN,3)
465:      &      /RRNU(IBP(I),KN,1)
466:      &      *RHO*SMIC(IBP(I),KN,LMIC(2))/SNUF
467:      SNUFD = SNUFD + RRNU(IBP(I),KN,2)
468:      &      *RHO*SMIC(IBP(I),KN,LMIC(3))
469:      end if
470:      end do
471:
472: c --- scheme 1 ---
473:      WCTRP(INC,2,ISCSB+1) = WCTRP(INC,2,ISCSB+1)
474:      &      + ( WWB(IBP(I)) - 1.d0 )*FLX(I)*SNUF
475: c --- scheme 2 ---
476:      WCTRP(INC,2,ISCSB+1) = WCTRP(INC,2,ISCSB+1)
477:      &      + ( RRNU*WWB(IBP(I)) - 1.d0 )*FLX(I)*SNUF
478: c ... source perturbation ...
479:      do ISP = 0, NGSP
480: c --- scheme 1 ---
481:      WCTRP(INC,2,ISCSB+(ISP+1)+1) =
482:      &      WCTRP(INC,2,ISCSB+(ISP+1)+1)
483:      &      + ( WSB(IBP(I),ISP) )
484:      &      *FLX(I)*SNUF
485: c --- scheme 2 ---
486:      WCTRP(INC,2,ISCSB+(ISP+1)+1) =
487:      &      WCTRP(INC,2,ISCSB+(ISP+1)+1)
488:      &      + ( RRNU*WSB(IBP(I),ISP) )
489:      &      *FLX(I)*SNUF
490:      end do
491:
492: c ... beta-effective with differential operator sampling
493:
494:      WCTRP(INC,2,IPBDS) = WCTRP(INC,2,IPBDS)
495:      &      + WWBD(IBP(I))*FLX(I)*SNUF
496:      WCTRP(INC,2,IPLDS) = WCTRP(INC,2,IPLDS)
497:      &      + WWLD(IBP(I))*FLX(I)*SNUF
498:      do ISP = 1, NGSP
499:      WCTRP(INC,2,IPBDS+ISP) = WCTRP(INC,2,IPBDS+ISP)
500:      &      + WSB(IBP(I),ISP)*FLX(I)*SNUF
501:      WCTRP(INC,2,IPLDS+ISP) = WCTRP(INC,2,IPLDS+ISP)
502:      &      + WSLD(IBP(I),ISP)*FLX(I)*SNUF
503:      end do
504:
505: 124      continue
506:      end if
507:      end if
508: c-beff2
509: 120      continue
510:      end if
511: c
512: c if (JRESP.ne.0) then
513: c do 140 N = 1, NSTAL
514: c do 130 I = 1, NN
515: c RSTAL(IR1,N) = RSTAL(IR1,N) + SGAL(IBP(I),N)*FLX(I)
516: c 130      continue
517: c 140      continue
518: c endif
519: c
520: c if (JREAC.ne.0.and.NEMIC.ne.0) then

```

src/mvp/talkp.f

```

521: C
522: C      do 220 N = 1, NUC
523: C          DN      = DNZON(N,IZ1)
524: C          if ( DN.ne.0.0 ) then
525: C              if ( NGP1.gt.1 ) then
526: C                  do 150 I = 1, NN
527: C                      DNF(I) = DN*FLX(I)
528: C                  continue
529: C          *VOCL LOOP,SCALAR
530: C              do 160 I = 1, NN
531: C                  DNFLX(IEG(I),IR1,N) = DNFLX(IEG(I),IR1,N)
532: C                  &      + DNF(I)
533: C              continue
534: C              do 180 J = 1, NEMIC
535: C                  *VOCL LOOP,SCALAR
536: C                      do 170 I = 1, NN
537: C                          IE      = IEG(I)
538: C                          RMIC(IE,IR1,N,J) = RMIC(IE,IR1,N,J)
539: C                          &      + SMIC(IBP(I),N,LMIC(LEMIC(J,2,1)))
540: C                          &      + DNF(I)
541: C                      continue
542: C                  continue
543: C              continue
544: C              .... NGP1 = 1 (vectorizable)
545: C          else
546: C              IE      = IEG(1)
547: C              do 190 I = 1, NN
548: C                  DNF(I) = DN*FLX(I)
549: C                  DNFLX(IE,IR1,N) = DNFLX(IE,IR1,N) + DNF(I)
550: C              continue
551: C              do 210 J = 1, NEMIC
552: C                  do 200 I = 1, NN
553: C                      RMIC(IE,IR1,N,J) = RMIC(IE,IR1,N,J)
554: C                      &      + SMIC(IBP(I),N,LMIC(LEMIC(J,2,1)))
555: C                      &      + DNF(I)
556: C                  continue
557: C              continue
558: C          end if
559: C      end if
560: C      continue
561: C      end if
562: C      continue
563: C      end if
564: C      continue
565: C      end if
566: C      end if
567: C      end if
568: C      end if
569: C      end if
570: C
571: C**** TALLY INCLUDING MULTIPLE REGIONS ****
572: C      ( JREGN = 1 OR -1 )
573: C
574: C      else
575: C
576: C      if ( JEIGN.ne.0 ) then
577: C
578: C          ( VOID REGION ( IMB(I) = 0 ) MAY APPEAR.)
579: C
580: C          if ( JREGN.eq.1
581: C              &      .and.(JZONE.eq.0.and.IMB(1).ne.0.or.JZONE.eq.1) ) then
582: C
583: C              do 230 I = 1, NN

```

```

586: C          if ( IMB(I).gt.0 ) then
587: C              SNUF      = SMAC(IBP(I),IMB(I),LMAC(2))
588: C              WC1      = WC1 + FLX(I)*SNUF
589: C              WC2      = WC2 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(8))
590: C
591: C-----
592: C TEMPERATURE
593: C-----
594: C          if( JPTMP.ne.0 ) then
595: C
596: C              if( SNUF.gt.0.d0 ) then
597: C                  do 236 II = 1, NUCPT
598: C                      WKPT(IBP(I),1) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
599: C                      WCTRP(II,2,1) = WCTRP(II,2,1)
600: C                      &      + ( WKPT(IBP(I),1)+WWT(IBP(I),1) )*FLX(I)*SNUF
601: C
602: C                      WKPT(IBP(I),2) = 0.0d0
603: C                      WCTRP(II,2,2) = WCTRP(II,2,2)
604: C                      &      + ( WKPT(IBP(I),2)+WWT(IBP(I),2) )*FLX(I)*SNUF
605: C
606: C                      WKPT(IBP(I),3) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
607: C                      WCTRP(II,2,3) = WCTRP(II,2,3)
608: C                      &      + ( WKPT(IBP(I),3)+WWT(IBP(I),3) )*FLX(I)*SNUF
609: C
610: C                      WKPT(IBP(I),4) = 0.0d0
611: C                      WCTRP(II,2,4) = WCTRP(II,2,4)
612: C                      &      + ( WKPT(IBP(I),4)+WWT(IBP(I),4) )*FLX(I)*SNUF
613: C
614: C                      WKPT(IBP(I),5) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
615: C                      WCTRP(II,2,5) = WCTRP(II,2,5)
616: C                      &      + ( WKPT(IBP(I),5)+WWT(IBP(I),5) )*FLX(I)*SNUF
617: C
618: C                      WKPT(IBP(I),6) = 0.0d0
619: C                      WCTRP(II,2,6) = WCTRP(II,2,6)
620: C                      &      + ( WKPT(IBP(I),6)+WWT(IBP(I),6) )*FLX(I)*SNUF
621: C
622: C                      WKPT(IBP(I),7) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
623: C                      WCTRP(II,2,7) = WCTRP(II,2,7)
624: C                      &      + ( WKPT(IBP(I),7)+WWT(IBP(I),7) )*FLX(I)*SNUF
625: C                  continue
626: C              end if
627: C          end if
628: C
629: C-----
630: C DENSITY & NUMBER DENSITY
631: C-----
632: C          if( JPDEN.ne.0 ) then
633: C
634: C              if( SNUF.gt.0.d0 ) then
635: C                  do 231 INC = 1, NUCPT
636: C
637: C                      do 321 IPT = 1, NPPTS
638: C                          ... density perturbation ...
639: C                          DELTA = DBLE(JPTTR(KZREG(IZN(I)),IPT))
640: C
641: C                          DELTA = 0.0d0
642: C                          MM = KZMAT(IZN(I))
643: C                          do 251 J = 1, MNUC(MM)
644: C                              IDN = LPDEN( MM ) - 1 + J
645: C                              KN = INUCT( IDN )
646: C                              RHO = DENST( IDN )
647: C                              DELTA = DELTA
648: C                              &      + dble(JPTTR(IRG(I),IPT))
649: C                              &      * dble(JPTNU(KN,IPT))
650: C                              &      * RHO*SMIC( IBP(I),KN,LMIC(2) )/SNUF

```

src/mvp/talkp.f

```

651: 251 continue
652:
653: c ... 1st order ...
654: WCTRP( INC,2,IPT) = WCTRP( INC,2,IPT)
655: & + ( DELTA+WWD( IBP(I),IPT,1))*FLX(I)*SNUF
656: c ... 2nd order ...
657: WCTRP( INC,2,NPTDS+IPT) =
658: & WCTRP( INC,2,NPTDS+IPT)
659: & + ( ( DELTA + WWD( IBP(I),IPT,1) )**2
660: & - DELTA + WWD2( IBP(I),IPT) )
661: & *FLX(I)*SNUF
662: c ... 3rd & 4th order ...
663: CONT1 = WWD( IBP(I),IPT,1) + DELTA
664: D1CT1 = WWD( IBP(I),IPT,2) - DELTA
665: D2CT1 = WWD( IBP(I),IPT,3) + DELTA*2.d0
666: D3CT1 = WWD( IBP(I),IPT,4) - DELTA*6.d0
667: D4CT1 = WWD( IBP(I),IPT,5) + DELTA*24.d0
668: D5CT1 = WWD( IBP(I),IPT,6) - DELTA*120.d0
669: D6CT1 = WWD( IBP(I),IPT,7) + DELTA*720.d0
670: D7CT1 = WWD( IBP(I),IPT,8) - DELTA*5040.d0
671:
672: CONT2 = D1CT1 + CONT1*CONT1
673: D1CT2 = D2CT1 + 2.d0*CONT1*D1CT1
674: D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
675: & + 2.d0*CONT1*D2CT1
676: D3CT2 = D4CT1 + D3CT1*CONT1
677: & + 3.d0*D2CT1*D1CT1
678: & + 3.d0*D1CT1*D2CT1
679: & + CONT1*D3CT1
680: D4CT2 = D5CT1 + D4CT1*CONT1
681: & + 4.d0*D3CT1*D1CT1
682: & + 6.d0*D2CT1*D2CT1
683: & + 4.d0*D1CT1*D3CT1
684: & + CONT1*D4CT1
685: D5CT2 = D6CT1 + D5CT1*CONT1
686: & + 5.d0*D4CT1*D1CT1
687: & +10.d0*D3CT1*D2CT1
688: & +10.d0*D2CT1*D3CT1
689: & + 5.d0*D1CT1*D4CT1
690: & + CONT1*D5CT1
691: D6CT2 = D7CT1 + D6CT1*CONT1
692: & + 6.d0*D5CT1*D1CT1
693: & +15.d0*D4CT1*D2CT1
694: & +20.d0*D3CT1*D3CT1
695: & +15.d0*D2CT1*D4CT1
696: & + 6.d0*D1CT1*D5CT1
697: & + CONT1*D6CT1
698:
699: CONT3 = D1CT2 + CONT2*CONT1
700: D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
701: D2CT3 = D3CT2 + D2CT2*CONT1
702: & + 2.d0*D1CT2*D1CT1
703: & + CONT2*D2CT1
704: D3CT3 = D4CT2 + D3CT2*CONT1
705: & + 3.d0*D2CT2*D1CT1
706: & + 3.d0*D1CT2*D2CT1
707: & + CONT2*D3CT1
708: D4CT3 = D5CT2 + D4CT2*CONT1
709: & + 4.d0*D3CT2*D1CT1
710: & + 6.d0*D2CT2*D2CT1
711: & + 4.d0*D1CT2*D3CT1
712: & + CONT2*D4CT1
713: D5CT3 = D6CT2 + D5CT2*CONT1
714: & + 5.d0*D4CT2*D1CT1
715: & +10.d0*D3CT2*D2CT1
716: & +10.d0*D2CT2*D3CT1
717: & + 5.d0*D1CT2*D4CT1
718: & + CONT2*D5CT1
719:
720: CONT4 = D1CT3 + CONT3*CONT1
721: D1CT4 = D2CT3 + D1CT3*CONT1 + CONT3*D1CT1
722: D2CT4 = D3CT3 + D2CT3*CONT1
723: & + 2.d0*D1CT3*D1CT1
724: & + CONT3*D2CT1
725: D3CT4 = D4CT3 + D3CT3*CONT1
726: & + 3.d0*D2CT3*D1CT1
727: & + 3.d0*D1CT3*D2CT1
728: & + CONT3*D3CT1
729: D4CT4 = D5CT3 + D4CT3*CONT1
730: & + 4.d0*D3CT3*D1CT1
731: & + 6.d0*D2CT3*D2CT1
732: & + 4.d0*D1CT3*D3CT1
733: & + CONT3*D4CT1
734:
735: CONT5 = D1CT4 + CONT4*CONT1
736: D1CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
737: D2CT5 = D3CT4 + D2CT4*CONT1
738: & + 2.d0*D1CT4*D1CT1
739: & + CONT4*D2CT1
740: D3CT5 = D4CT4 + D3CT4*CONT1
741: & + 3.d0*D2CT4*D1CT1
742: & + 3.d0*D1CT4*D2CT1
743: & + CONT4*D3CT1
744:
745: CONT6 = D1CT5 + CONT5*CONT1
746: D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
747: D2CT6 = D3CT5 + D2CT5*CONT1
748: & + 2.d0*D1CT5*D1CT1
749: & + CONT5*D2CT1
750:
751: CONT7 = D1CT6 + CONT6*CONT1
752: D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1
753:
754: CONT8 = D1CT7 + CONT7*CONT1
755:
756: WCTRP( INC,2,2*NPTDS+IPT) =
757: & WCTRP( INC,2,2*NPTDS+IPT)
758: & + CONT3*FLX(I)*SNUF
759:
760: WCTRP( INC,2,3*NPTDS+IPT) =
761: & WCTRP( INC,2,3*NPTDS+IPT)
762: & + CONT4*FLX(I)*SNUF
763:
764: WCTRP( INC,2,4*NPTDS+IPT) =
765: & WCTRP( INC,2,4*NPTDS+IPT)
766: & + CONT5*FLX(I)*SNUF
767:
768: WCTRP( INC,2,5*NPTDS+IPT) =
769: & WCTRP( INC,2,5*NPTDS+IPT)
770: & + CONT6*FLX(I)*SNUF
771:
772: WCTRP( INC,2,6*NPTDS+IPT) =
773: & WCTRP( INC,2,6*NPTDS+IPT)
774: & + CONT7*FLX(I)*SNUF
775:
776: WCTRP( INC,2,7*NPTDS+IPT) =
777: & WCTRP( INC,2,7*NPTDS+IPT)
778: & + CONT8*FLX(I)*SNUF
779: c ... source perturbation ...
780: do ISP = 0, NGSP

```

src/mvp/talkp.f

```

781:      WCTRP( INC, 2, (ISP+NORDDS)*NPTDS+IPT) =
782:      &      WCTRP( INC, 2, (ISP+NORDDS)*NPTDS+IPT)
783:      &      + WSD( IBP( I), ISP, IPT, 1)*FLX( I)*SNUF
784:      end do
785:
786:      do ISP = 1, NGSP
787:      IP = (NORDDS+NGSP+ISP)*NPTDS+IPT
788:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
789:      &      + ( 2.d0*CONT1*WSD( IBP( I), ISP, IPT, 1)
790:      &      + WSD( IBP( I), ISP, IPT, 2))
791:      &      *FLX( I)*SNUF
792:      end do
793:
794:      do ISP = 1, NGSP
795:      IP = (NORDDS+2*NGSP+ISP)*NPTDS+IPT
796:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
797:      &      + ( WSD( IBP( I), ISP, IPT, 3)
798:      &      + 3.d0*CONT1*WSD( IBP( I), ISP, IPT, 2)
799:      &      + 3.d0*CONT2*WSD( IBP( I), ISP, IPT, 1)
800:      &      )*FLX( I)*SNUF
801:      end do
802:
803:      do ISP = 1, NGSP
804:      IP = (NORDDS+3*NGSP+ISP)*NPTDS+IPT
805:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
806:      &      + ( WSD( IBP( I), ISP, IPT, 4)
807:      &      + 4.d0*CONT1*WSD( IBP( I), ISP, IPT, 3)
808:      &      + 6.d0*CONT2*WSD( IBP( I), ISP, IPT, 2)
809:      &      + 4.d0*CONT3*WSD( IBP( I), ISP, IPT, 1)
810:      &      )*FLX( I)*SNUF
811:      end do
812:
813:      do ISP = 1, NGSP
814:      IP = (NORDDS+4*NGSP+ISP)*NPTDS+IPT
815:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
816:      &      + ( WSD( IBP( I), ISP, IPT, 5)
817:      &      + 5.d0*CONT1*WSD( IBP( I), ISP, IPT, 4)
818:      &      +10.d0*CONT2*WSD( IBP( I), ISP, IPT, 3)
819:      &      +10.d0*CONT3*WSD( IBP( I), ISP, IPT, 2)
820:      &      + 5.d0*CONT4*WSD( IBP( I), ISP, IPT, 1)
821:      &      )*FLX( I)*SNUF
822:      end do
823:
824:      do ISP = 1, NGSP
825:      IP = (NORDDS+5*NGSP+ISP)*NPTDS+IPT
826:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
827:      &      + ( WSD( IBP( I), ISP, IPT, 6)
828:      &      + 6.d0*CONT1*WSD( IBP( I), ISP, IPT, 5)
829:      &      +15.d0*CONT2*WSD( IBP( I), ISP, IPT, 4)
830:      &      +20.d0*CONT3*WSD( IBP( I), ISP, IPT, 3)
831:      &      +15.d0*CONT4*WSD( IBP( I), ISP, IPT, 2)
832:      &      + 6.d0*CONT5*WSD( IBP( I), ISP, IPT, 1)
833:      &      )*FLX( I)*SNUF
834:      end do
835:
836:      do ISP = 1, NGSP
837:      IP = (NORDDS+6*NGSP+ISP)*NPTDS+IPT
838:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
839:      &      + ( WSD( IBP( I), ISP, IPT, 7)
840:      &      + 7.d0*CONT1*WSD( IBP( I), ISP, IPT, 6)
841:      &      +21.d0*CONT2*WSD( IBP( I), ISP, IPT, 5)
842:      &      +35.d0*CONT3*WSD( IBP( I), ISP, IPT, 4)
843:      &      +35.d0*CONT4*WSD( IBP( I), ISP, IPT, 3)
844:      &      +21.d0*CONT5*WSD( IBP( I), ISP, IPT, 2)
845:      &      + 7.d0*CONT6*WSD( IBP( I), ISP, IPT, 1)
846:      &
847:      &
848:      &
849:      do ISP = 1, NGSP
850:      IP = (NORDDS+7*NGSP+ISP)*NPTDS+IPT
851:      WCTRP( INC, 2, IP) = WCTRP( INC, 2, IP)
852:      &      + ( WSD( IBP( I), ISP, IPT, 8)
853:      &      + 8.d0*CONT1*WSD( IBP( I), ISP, IPT, 7)
854:      &      +28.d0*CONT2*WSD( IBP( I), ISP, IPT, 6)
855:      &      +56.d0*CONT3*WSD( IBP( I), ISP, IPT, 5)
856:      &      +70.d0*CONT4*WSD( IBP( I), ISP, IPT, 4)
857:      &      +56.d0*CONT5*WSD( IBP( I), ISP, IPT, 3)
858:      &      +28.d0*CONT6*WSD( IBP( I), ISP, IPT, 2)
859:      &      + 8.d0*CONT7*WSD( IBP( I), ISP, IPT, 1)
860:      &      )*FLX( I)*SNUF
861:      end do
862:      321
863:      continue
864:      c === correlated sampling ===
865:
866:      do 235 IPT = 1, NPTCS
867:      cc
868:      cc &
869:      DLT01 =
870:      &      dble( JPTTR( KZREG( IZN( I)), NPTDS+IPT))
871:      DLT01 =
872:      &      dble( JPTTR( IRG( I), NPTDS+IPT))
873:      &
874:      &
875:      &
876:      &
877:      c ... source perturbation ...
878:      do 335 ISP = 0, NGSP
879:      WCTRP( INC, 2, NOCS+(ISP+1)*NPTCS+IPT)=
880:      &      WCTRP( INC, 2, NOCS+(ISP+1)*NPTCS+IPT)
881:      &      + ( (1.d0+DELA1*DLT01)
882:      &      *WSC( IBP( I), ISP, IPT) )
883:      &      *FLX( I)*SNUF
884:      continue
885:      235
886:      continue
887:      231
888:      end if
889:      end if
890:      c+beff2
891:      if (NPTBE.gt.0) then
892:      if (SNUF.gt.0.d0) then
893:      do 234 INC = 1, NUCPT
894:      SNUFD = 0.d0
895:      RRNU = 0.d0
896:      MM = KZMAT( IZN( I))
897:      do J = 1, MNUC( MM)
898:      IDN = LPDEN( MM) - 1 + J
899:      KN = INUCT( IDN)
900:      RHO = DENST( IDN)
901:      if (RNU( IBP( I), KN, 1).gt.0.0) then
902:      RRNU = RRNU
903:      &
904:      &
905:      &
906:      &
907:      &
908:      &
909:      &
910:      &
911:      &
912:      &
913:      &
914:      &
915:      &
916:      &
917:      &
918:      &
919:      &
920:      &
921:      &
922:      &
923:      &
924:      &
925:      &
926:      &
927:      &
928:      &
929:      &
930:      &
931:      &
932:      &
933:      &
934:      &
935:      &
936:      &
937:      &
938:      &
939:      &
940:      &
941:      &
942:      &
943:      &
944:      &
945:      &
946:      &
947:      &
948:      &
949:      &
950:      &
951:      &
952:      &
953:      &
954:      &
955:      &
956:      &
957:      &
958:      &
959:      &
960:      &
961:      &
962:      &
963:      &
964:      &
965:      &
966:      &
967:      &
968:      &
969:      &
970:      &
971:      &
972:      &
973:      &
974:      &
975:      &
976:      &
977:      &
978:      &
979:      &
980:      &
981:      &
982:      &
983:      &
984:      &
985:      &
986:      &
987:      &
988:      &
989:      &
990:      &
991:      &
992:      &
993:      &
994:      &
995:      &
996:      &
997:      &
998:      &
999:      &
1000:      &

```


src/mvp/talkp.f

```

911:
912: c --- scheme 1 ---
913:           WCTRP(INC,2, ISCSB+1) = WCTRP(INC,2, ISCSB+1)
914:           & + ( WWB(IBP(I)) - 1.d0 ) * FLX(I) * SNUF
915: c --- scheme 2 ---
916:           WCTRP(INC,2, ISCSB+1) = WCTRP(INC,2, ISCSB+1)
917:           & + ( RRNU * WWB(IBP(I)) - 1.d0 ) * FLX(I) * SNUF
918: c ... source perturbation ...
919:           do ISP = 0, NGSP
920: c --- scheme 1 ---
921:           WCTRP(INC,2,ISCSB+(ISP+1)+1) =
922:           & WCTRP(INC,2,ISCSB+(ISP+1)+1)
923:           & + ( WSB(IBP(I),ISP) )
924:           & * FLX(I) * SNUF
925: c --- scheme 2 ---
926:           WCTRP(INC,2,ISCSB+(ISP+1)+1) =
927:           & WCTRP(INC,2,ISCSB+(ISP+1)+1)
928:           & + ( RRNU * WSB(IBP(I),ISP) )
929:           & * FLX(I) * SNUF
930:           end do
931:
932: c ... beta-effective with differential operator sampling
933:
934:           WCTRP(INC,2,IPBDS) = WCTRP(INC,2,IPBDS)
935:           & + WWBD(IBP(I)) * FLX(I) * SNUF
936:           WCTRP(INC,2,IPLDS) = WCTRP(INC,2,IPLDS)
937:           & + WWLD(IBP(I)) * FLX(I) * SNUF
938:           do ISP = 1, NGSP
939:           WCTRP(INC,2,IPBDS+ISP) =
940:           & WCTRP(INC,2,IPBDS+ISP)
941:           & + WSB(D,IBP(I),ISP) * FLX(I) * SNUF
942:           WCTRP(INC,2,IPLDS+ISP) =
943:           & WCTRP(INC,2,IPLDS+ISP)
944:           & + WSLD(IBP(I),ISP) * FLX(I) * SNUF
945:           end do
946:
947: 234           continue
948:
949:           end if
950:
951:           end if
952: c-beff2
953:           end if
954: 230           continue
955:           end if
956: c
957: c ( VOID ZONE ( IMB(I) = 0 ) DOES NOT APPEAR WHEN CALLED
958: c IN 'NEUTR' )
959: c
960:           if ( JREGN.eq.-1 ) then
961:           do 240 I = 1, NN
962:           SNUF = SMAC(IBP(I),IMB(I),LMAC(2))
963: c WC1 = WC1 + FLX(I)*SNUF
964: c WC2 = WC2 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(8))
965:
966: c -----
967: c TEMPERATURE
968: c -----
969: c           if( JPTMP.ne.0 ) then
970:
971: c           if( SNUF.gt.0.d0 ) then
972: c           do 246 II = 1, NUCPT
973: c           WKPT(IBP(I),1) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
974: c           WCTRP(II,2,1) = WCTRP(II,2,1)
975: c           & + ( WKPT(IBP(I),1)+WWT(IBP(I),1) ) * FLX(I) * SNUF

```

```

976:
977: c           WKPT(IBP(I),2) = 0.0d0
978: c           WCTRP(II,2,2) = WCTRP(II,2,2)
979: c           & + ( WKPT(IBP(I),2)+WWT(IBP(I),2) ) * FLX(I) * SNUF
980:
981: c           WKPT(IBP(I),3) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
982: c           WCTRP(II,2,3) = WCTRP(II,2,3)
983: c           & + ( WKPT(IBP(I),3)+WWT(IBP(I),3) ) * FLX(I) * SNUF
984:
985: c           WKPT(IBP(I),4) = 0.0d0
986: c           WCTRP(II,2,4) = WCTRP(II,2,4)
987: c           & + ( WKPT(IBP(I),4)+WWT(IBP(I),4) ) * FLX(I) * SNUF
988:
989: c           WKPT(IBP(I),5) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
990: c           WCTRP(II,2,5) = WCTRP(II,2,5)
991: c           & + ( WKPT(IBP(I),5)+WWT(IBP(I),5) ) * FLX(I) * SNUF
992:
993: c           WKPT(IBP(I),6) = 0.0d0
994: c           WCTRP(II,2,6) = WCTRP(II,2,6)
995: c           & + ( WKPT(IBP(I),6)+WWT(IBP(I),6) ) * FLX(I) * SNUF
996:
997: c           WKPT(IBP(I),7) = SMACP(IBP(I),IMB(I),LMAC(2))/SNUF
998: c           WCTRP(II,2,7) = WCTRP(II,2,7)
999: c           & + ( WKPT(IBP(I),7)+WWT(IBP(I),7) ) * FLX(I) * SNUF
1000: c 246           continue
1001: c           end if
1002:
1003: c           end if
1004: c -----
1005: c DENSITY & NUMBER DENSITY
1006: c -----
1007: c           if( JPDEN.ne.0 ) then
1008:
1009: c           if( SNUF.gt.0.d0 ) then
1010: c           do 241 INC = 1, NUCPT
1011:
1012: c           do 421 IPT = 1, NPTDS
1013: c ... density perturbation ...
1014: c           DELTA = DBLE(JPTTR(KZREG(IZN(I)),IPT))
1015:
1016: c           DELTA = 0.0d0
1017: c           MM = KZMAT(IZN(I))
1018: c           do 351 J = 1, MNUC(MM)
1019: c           IDN = LPDEN( MM ) - 1 + J
1020: c           KN = INUCT( IDN )
1021: c           RHO = DENST( IDN )
1022: c           DELTA = DELTA
1023: c           & + dble(JPTTR(IRG(I),IPT))
1024: c           & * dble(JPTNU(KN,IPT))
1025: c           & * RHO * SMIC(IBP(I),KN,LMIC(2))/SNUF
1026: c 351           continue
1027:
1028: c ... 1st order ...
1029: c           & + ( DELTA + WWD(IBP(I),IPT,1) )
1030: c           & * FLX(I) * SNUF
1031:
1032: c ... 2nd order ...
1033: c           WCTRP( INC,2,NPTDS+IPT) =
1034: c           & WCTRP( INC,2,NPTDS+IPT)
1035: c           & + ( ( DELTA + WWD(IBP(I),IPT,1) ) ** 2
1036: c           & - DELTA + WWD2(IBP(I),IPT) )
1037: c           & * FLX(I) * SNUF
1038: c ... 3rd & 4th order ...
1039: c           CONT1 = WWD(IBP(I),IPT,1) + DELTA
1040: c           D1CT1 = WWD(IBP(I),IPT,2) - DELTA

```

src/mvp/talkp.f

```

1041:      D2CT1 = WWD(IBP(I),IPT,3) + DELTA*2.d0
1042:      D3CT1 = WWD(IBP(I),IPT,4) - DELTA*6.d0
1043:      D4CT1 = WWD(IBP(I),IPT,5) + DELTA*24.d0
1044:      D5CT1 = WWD(IBP(I),IPT,6) - DELTA*120.d0
1045:      D6CT1 = WWD(IBP(I),IPT,7) + DELTA*720.d0
1046:      D7CT1 = WWD(IBP(I),IPT,8) - DELTA*5040.d0
1047:
1048:      CONT2 = D1CT1 + CONT1*CONT1
1049:      D1CT2 = D2CT1 + 2.d0*CONT1*D1CT1
1050:      D2CT2 = D3CT1 + 2.d0*D1CT1*D1CT1
1051:      &      + 2.d0*CONT1*D2CT1
1052:      D3CT2 = D4CT1 + D3CT1*CONT1
1053:      &      + 3.d0*D2CT1*D1CT1
1054:      &      + 3.d0*D1CT1*D2CT1
1055:      &      + CONT1*D3CT1
1056:      D4CT2 = D5CT1 + D4CT1*CONT1
1057:      &      + 4.d0*D3CT1*D1CT1
1058:      &      + 6.d0*D2CT1*D2CT1
1059:      &      + 4.d0*D1CT1*D3CT1
1060:      &      + CONT1*D4CT1
1061:      D5CT2 = D6CT1 + D5CT1*CONT1
1062:      &      + 5.d0*D4CT1*D1CT1
1063:      &      +10.d0*D3CT1*D2CT1
1064:      &      +10.d0*D2CT1*D3CT1
1065:      &      + 5.d0*D1CT1*D4CT1
1066:      &      + CONT1*D5CT1
1067:      D6CT2 = D7CT1 + D6CT1*CONT1
1068:      &      + 6.d0*D5CT1*D1CT1
1069:      &      +15.d0*D4CT1*D2CT1
1070:      &      +20.d0*D3CT1*D3CT1
1071:      &      +15.d0*D2CT1*D4CT1
1072:      &      + 6.d0*D1CT1*D5CT1
1073:      &      + CONT1*D6CT1
1074:
1075:      CONT3 = D1CT2 + CONT2*CONT1
1076:      D1CT3 = D2CT2 + D1CT2*CONT1 + CONT2*D1CT1
1077:      D2CT3 = D3CT2 + D2CT2*CONT1
1078:      &      + 2.d0*D1CT2*D1CT1
1079:      &      + CONT2*D2CT1
1080:      D3CT3 = D4CT2 + D3CT2*CONT1
1081:      &      + 3.d0*D2CT2*D1CT1
1082:      &      + 3.d0*D1CT2*D2CT1
1083:      &      + CONT2*D3CT1
1084:      D4CT3 = D5CT2 + D4CT2*CONT1
1085:      &      + 4.d0*D3CT2*D1CT1
1086:      &      + 6.d0*D2CT2*D2CT1
1087:      &      + 4.d0*D1CT2*D3CT1
1088:      &      + CONT2*D4CT1
1089:      D5CT3 = D6CT2 + D5CT2*CONT1
1090:      &      + 5.d0*D4CT2*D1CT1
1091:      &      +10.d0*D3CT2*D2CT1
1092:      &      +10.d0*D2CT2*D3CT1
1093:      &      + 5.d0*D1CT2*D4CT1
1094:      &      + CONT2*D5CT1
1095:
1096:      CONT4 = D1CT3 + CONT3*CONT1
1097:      D1CT4 = D2CT3 + D1CT3*CONT1 + CONT3*D1CT1
1098:      D2CT4 = D3CT3 + D2CT3*CONT1
1099:      &      + 2.d0*D1CT3*D1CT1
1100:      &      + CONT3*D2CT1
1101:      D3CT4 = D4CT3 + D3CT3*CONT1
1102:      &      + 3.d0*D2CT3*D1CT1
1103:      &      + 3.d0*D1CT3*D2CT1
1104:      &      + CONT3*D3CT1
1105:      D4CT4 = D5CT3 + D4CT3*CONT1
1106:      &      + 4.d0*D3CT3*D1CT1
1107:      &      + 6.d0*D2CT3*D2CT1
1108:      &      + 4.d0*D1CT3*D3CT1
1109:      &      + CONT3*D4CT1
1110:
1111:      CONT5 = D1CT4 + CONT4*CONT1
1112:      D1CT5 = D2CT4 + D1CT4*CONT1 + CONT4*D1CT1
1113:      D2CT5 = D3CT4 + D2CT4*CONT1
1114:      &      + 2.d0*D1CT4*D1CT1
1115:      &      + CONT4*D2CT1
1116:      D3CT5 = D4CT4 + D3CT4*CONT1
1117:      &      + 3.d0*D2CT4*D1CT1
1118:      &      + 3.d0*D1CT4*D2CT1
1119:      &      + CONT4*D3CT1
1120:
1121:      CONT6 = D1CT5 + CONT5*CONT1
1122:      D1CT6 = D2CT5 + D1CT5*CONT1 + CONT5*D1CT1
1123:      D2CT6 = D3CT5 + D2CT5*CONT1
1124:      &      + 2.d0*D1CT5*D1CT1
1125:      &      + CONT5*D2CT1
1126:
1127:      CONT7 = D1CT6 + CONT6*CONT1
1128:      D1CT7 = D2CT6 + D1CT6*CONT1 + CONT6*D1CT1
1129:
1130:      CONT8 = D1CT7 + CONT7*CONT1
1131:
1132:      WCTRP( INC,2,2*NPTDS+IPT) =
1133:      &      WCTRP( INC,2,2*NPTDS+IPT)
1134:      &      + CONT3*FLX(I)*SNUF
1135:
1136:      WCTRP( INC,2,3*NPTDS+IPT) =
1137:      &      WCTRP( INC,2,3*NPTDS+IPT)
1138:      &      + CONT4*FLX(I)*SNUF
1139:
1140:      WCTRP( INC,2,4*NPTDS+IPT) =
1141:      &      WCTRP( INC,2,4*NPTDS+IPT)
1142:      &      + CONT5*FLX(I)*SNUF
1143:
1144:      WCTRP( INC,2,5*NPTDS+IPT) =
1145:      &      WCTRP( INC,2,5*NPTDS+IPT)
1146:      &      + CONT6*FLX(I)*SNUF
1147:
1148:      WCTRP( INC,2,6*NPTDS+IPT) =
1149:      &      WCTRP( INC,2,6*NPTDS+IPT)
1150:      &      + CONT7*FLX(I)*SNUF
1151:
1152:      WCTRP( INC,2,7*NPTDS+IPT) =
1153:      &      WCTRP( INC,2,7*NPTDS+IPT)
1154:      &      + CONT8*FLX(I)*SNUF
1155: c ... source perturbation ...
1156:      do ISP = 0, NGSP
1157:      &      WCTRP( INC,2,(ISP+NORDDS)*NPTDS+IPT) =
1158:      &      WCTRP( INC,2,(ISP+NORDDS)*NPTDS+IPT)
1159:      &      + WSD( IBP(I),ISP,IPT,1)*FLX(I)*SNUF
1160:      end do
1161:
1162:      do ISP = 1, NGSP
1163:      &      IP = (NORDDS+NGSP+ISP)*NPTDS+IPT
1164:      &      WCTRP( INC,2,IP) = WCTRP( INC,2,IP)
1165:      &      + (2.d0*CONT1*WSD( IBP(I),ISP,IPT,1)
1166:      &      + WSD( IBP(I),ISP,IPT,2))
1167:      &      *FLX(I)*SNUF
1168:      end do
1169:
1170:      do ISP = 1, NGSP

```

src/mvp/talkp.f

```

1171:      IP = (NORDDDS+2*NGSP+ISP)*NPTDS+IPT
1172:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1173:      &      + ( WSD(IBP(I),ISP,IPT,3)
1174:      &      + 3.d0*CONT1*WSD(IBP(I),ISP,IPT,2)
1175:      &      + 3.d0*CONT2*WSD(IBP(I),ISP,IPT,1)
1176:      &      )*FLX(I)*SNUF
1177:      end do
1178:
1179:      do ISP = 1, NGSP
1180:      IP = (NORDDDS+3*NGSP+ISP)*NPTDS+IPT
1181:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1182:      &      + ( WSD(IBP(I),ISP,IPT,4)
1183:      &      + 4.d0*CONT1*WSD(IBP(I),ISP,IPT,3)
1184:      &      + 6.d0*CONT2*WSD(IBP(I),ISP,IPT,2)
1185:      &      + 4.d0*CONT3*WSD(IBP(I),ISP,IPT,1)
1186:      &      )*FLX(I)*SNUF
1187:      end do
1188:
1189:      do ISP = 1, NGSP
1190:      IP = (NORDDDS+4*NGSP+ISP)*NPTDS+IPT
1191:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1192:      &      + ( WSD(IBP(I),ISP,IPT,5)
1193:      &      + 5.d0*CONT1*WSD(IBP(I),ISP,IPT,4)
1194:      &      +10.d0*CONT2*WSD(IBP(I),ISP,IPT,3)
1195:      &      +10.d0*CONT3*WSD(IBP(I),ISP,IPT,2)
1196:      &      + 5.d0*CONT4*WSD(IBP(I),ISP,IPT,1)
1197:      &      )*FLX(I)*SNUF
1198:      end do
1199:
1200:      do ISP = 1, NGSP
1201:      IP = (NORDDDS+5*NGSP+ISP)*NPTDS+IPT
1202:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1203:      &      + ( WSD(IBP(I),ISP,IPT,6)
1204:      &      + 6.d0*CONT1*WSD(IBP(I),ISP,IPT,5)
1205:      &      +15.d0*CONT2*WSD(IBP(I),ISP,IPT,4)
1206:      &      +20.d0*CONT3*WSD(IBP(I),ISP,IPT,3)
1207:      &      +15.d0*CONT4*WSD(IBP(I),ISP,IPT,2)
1208:      &      + 6.d0*CONT5*WSD(IBP(I),ISP,IPT,1)
1209:      &      )*FLX(I)*SNUF
1210:      end do
1211:
1212:      do ISP = 1, NGSP
1213:      IP = (NORDDDS+6*NGSP+ISP)*NPTDS+IPT
1214:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1215:      &      + ( WSD(IBP(I),ISP,IPT,7)
1216:      &      + 7.d0*CONT1*WSD(IBP(I),ISP,IPT,6)
1217:      &      +21.d0*CONT2*WSD(IBP(I),ISP,IPT,5)
1218:      &      +35.d0*CONT3*WSD(IBP(I),ISP,IPT,4)
1219:      &      +35.d0*CONT4*WSD(IBP(I),ISP,IPT,3)
1220:      &      +21.d0*CONT5*WSD(IBP(I),ISP,IPT,2)
1221:      &      + 7.d0*CONT6*WSD(IBP(I),ISP,IPT,1)
1222:      &      )*FLX(I)*SNUF
1223:      end do
1224:
1225:      do ISP = 1, NGSP
1226:      IP = (NORDDDS+7*NGSP+ISP)*NPTDS+IPT
1227:      WCTRP(INC,2,IP) = WCTRP(INC,2,IP)
1228:      &      + ( WSD(IBP(I),ISP,IPT,8)
1229:      &      + 8.d0*CONT1*WSD(IBP(I),ISP,IPT,7)
1230:      &      +28.d0*CONT2*WSD(IBP(I),ISP,IPT,6)
1231:      &      +56.d0*CONT3*WSD(IBP(I),ISP,IPT,5)
1232:      &      +70.d0*CONT4*WSD(IBP(I),ISP,IPT,4)
1233:      &      +56.d0*CONT5*WSD(IBP(I),ISP,IPT,3)
1234:      &      +28.d0*CONT6*WSD(IBP(I),ISP,IPT,2)
1235:      &      + 8.d0*CONT7*WSD(IBP(I),ISP,IPT,1)

```

```

1236:      &      )*FLX(I)*SNUF
1237:      end do
1238:      421      continue
1239:
1240:      c === correlated sampling ===
1241:
1242:      do 245 IPT = 1, NPTCS
1243:      cc      DLT01 =
1244:      cc      &      dble(JPTTR(KZREG(IZN(I)),NPTDS+IPT))
1245:      DLT01 =
1246:      &      dble(JPTTR(IRG(I),NPTDS+IPT))
1247:      DELA1 = dble(DELA(1,NPTDS+IPT))
1248:      WCTRP(INC,2,IPT+N0CS) =
1249:      &      WCTRP(INC,2,IPT+N0CS)
1250:      &      + ( (1.d0+DELA1*DLT01)*WWR(IBP(I),IPT)
1251:      &      - 1.d0 )*FLX(I)*SNUF
1252:      c ... source perturbation ...
1253:      do 345 ISP = 0, NGSP
1254:      WCTRP(INC,2,N0CS+(ISP+1)*NPTCS+IPT) =
1255:      &      WCTRP(INC,2,N0CS+(ISP+1)*NPTCS+IPT)
1256:      &      + ( (1.d0+DELA1*DLT01)
1257:      &      *WSC(IBP(I),ISP,IPT) )
1258:      &      *FLX(I)*SNUF
1259:      345      continue
1260:      245      continue
1261:
1262:      241      continue
1263:      end if
1264:      end if
1265:      c+beff2
1266:      if (NPTBE.gt.0) then
1267:
1268:      if (SNUF.gt.0.d0) then
1269:
1270:      do 244 INC = 1, NUCPT
1271:      SNUFD = 0.d0
1272:      RRNU = 0.d0
1273:      MM = KZMAT(IZN(I))
1274:      do J = 1, MNUC(MM)
1275:      IDN = LPDEN(MM) - 1 + J
1276:      KN = INUCT(IDN)
1277:      RHO = DENST(IDN)
1278:      if (RNU(IBP(I),KN,1).gt.0.0) then
1279:      RRNU = RRNU
1280:      &      + RNU(IBP(I),KN,3)
1281:      &      /RNU(IBP(I),KN,1)
1282:      &      *RHO*SMIC(IBP(I),KN,LMIC(2))/SNUF
1283:      SNUFD = SNUFD + RNU(IBP(I),KN,2)
1284:      &      *RHO*SMIC(IBP(I),KN,LMIC(3))
1285:      end if
1286:      end do
1287:
1288:      c --- scheme 1 ---
1289:      WCTRP(INC,2,ISCSB+1) = WCTRP(INC,2,ISCSB+1)
1290:      &      + ( WWB(IBP(I)) - 1.d0 )*FLX(I)*SNUF
1291:      c --- scheme 2 ---
1292:      c      WCTRP(INC,2,ISCSB+1) = WCTRP(INC,2,ISCSB+1)
1293:      c      &      + ( RRNU*WWB(IBP(I)) - 1.d0 )*FLX(I)*SNUF
1294:      c ... source perturbation ...
1295:      do ISP = 0, NGSP
1296:      c --- scheme 1 ---
1297:      WCTRP(INC,2,ISCSB+(ISP+1)+1) =
1298:      &      WCTRP(INC,2,ISCSB+(ISP+1)+1)
1299:      &      + ( WSB(IBP(I),ISP) )
1300:      &      *FLX(I)*SNUF

```

src/mvp/talkp.f

```

1301: c    --- scheme 2 ---
1302: c          WCTRP( INC,2,ISCSB+(ISP+1)+1) =
1303: c      &          WCTRP( INC,2,ISCSB+(ISP+1)+1)
1304: c      &          + ( RRNU*WSB( IBP(I),ISP) )
1305: c      &          *FLX(I)*SNUF
1306: c          end do
1307:
1308: c ... beta-effective with differential operator sampling
1309:
1310:          WCTRP( INC,2,IPBDS) = WCTRP( INC,2,IPBDS)
1311: c      &          + WWBD( IBP(I))*FLX(I)*SNUF
1312:          WCTRP( INC,2,IPLDS) = WCTRP( INC,2,IPLDS)
1313: c      &          + WWLD( IBP(I))*FLX(I)*SNUF
1314:          do ISP = 1, NGSP
1315:              WCTRP( INC,2,IPBDS+ISP) =
1316: c      &              WCTRP( INC,2,IPBDS+ISP)
1317: c      &              + WSB( IBP(I),ISP)*FLX(I)*SNUF
1318:              WCTRP( INC,2,IPLDS+ISP) =
1319: c      &              WCTRP( INC,2,IPLDS+ISP)
1320: c      &              + WSLD( IBP(I),ISP)*FLX(I)*SNUF
1321:          end do
1322:
1323: 244          continue
1324:
1325:          end if
1326:
1327:          end if
1328: c-beff2
1329: 240          continue
1330:          end if
1331:          end if
1332: c
1333: c      if( JRESP.ne.0 ) then
1334: c          do 260 N = 1, NSTAL
1335: c              do 250 I = 1, NN
1336: c                  RSTAL( IRG(I),N) = RSTAL( IRG(I),N) + SGTAL( IBP(I),N)*
1337: c      &                  FLX(I)
1338: c      &          continue
1339: c      &          continue
1340: c          endif
1341: c
1342: c      ..... REFERENCE BY ( IEG,IRG ) --> ( IEG, 1 )
1343: c
1344: c          do 270 I = 1, NN
1345: c              IEG(I) = IEG(I) + ( IRG(I)-1)*NGROUP
1346: c      270          continue
1347: c
1348: c
1349: c
1350: c          do 280 I = 1, NN
1351: c              FLXX( IEG(I),1) = FLXX( IEG(I),1) + FLX(I)
1352: c      280          continue
1353: c
1354: c          if ( JREAC.ne.0.and.NEMIC.ne.0 ) then
1355: c              if ( JZONE.eq.1 ) then
1356: c                  do 320 N = 1, NUC
1357: c                      do 290 I = 1, NN
1358: c                          DNF(I) = DNZON(N,IZN(I))*FLX(I)
1359: c                          if ( DNF(I).ne.0.0 ) DNFLX( IEG(I),1,N) =
1360: c      &                          DNFLX( IEG(I),1,N) + DNF(I)
1361: c      290          continue
1362: c          do 310 J = 1, NEMIC
1363: c              do 300 I = 1, NN
1364: c                  if ( DNF(I).ne.0.0 ) then
1365: c                      DFS = DNF(I)*

```

```

1366: c      &          SMIC( IBP(I),N,LMIC( LEMIC(J,2,1)))
1367: c          RMIC( IEG(I),1,N,J) = RMIC( IEG(I),1,N,J) +
1368: c      &          DFS
1369: c          end if
1370: c
1371: c      300          continue
1372: c      310          continue
1373: c      320          continue
1374: c      else
1375: c          do 360 N = 1, NUC
1376: c              DN = DNZON(N,IZN(1))
1377: c              if ( DN.ne.0.0 ) then
1378: c                  do 330 I = 1, NN
1379: c                      DNF(I) = DN*FLX(I)
1380: c                      DNFLX( IEG(I),1,N) = DNFLX( IEG(I),1,N) + DNF(I)
1381: c      330          continue
1382: c          do 350 J = 1, NEMIC
1383: c              do 340 I = 1, NN
1384: c                  DFS = DNF(I)*
1385: c      &                  SMIC( IBP(I),N,LMIC( LEMIC(J,2,1)))
1386: c      &                  RMIC( IEG(I),1,N,J) = RMIC( IEG(I),1,N,J) +
1387: c      &                  DFS
1388: c
1389: c          continue
1390: c      350          continue
1391: c          end if
1392: c      360          continue
1393: c          end if
1394: c          end if
1395: c          end if
1396: c
1397: c      return
1398: c      end

```

src/mvp/tally.f

```

1:      subroutine TALLY( IOW,  JEIGN, JREAC, JREGN, JZONE, JRESP, FLX,
2:      &                  NN,    IEG,  IRG,  IBP,  IMB,  DNF,  NGROUP,
3:      &                  NGP1, NREG, NRESP, NUC,  NBANK, NSMIC, NSMAC,
4:      &                  NSTAL, MB,   IZN,  DNZON, NZONE, LEMIC, NEMIC,
5:      &                  RESP, SMIC, LMIC, SMAC, LMAC, SGTAL, FLXX,
6:      &                  REXX, RMIC, DNFLX, RSTAL, WC1,  WC2,
7:      c+beff1
8:      &                  JBEFF, RNU,  IBNK,  KIBNK, MIBNK, WC3, WC4,
9:      &                  WC6, WC7,
10: c-beff1
11: c##<2007/03/14:PN4:
12: c      &                  JPHNU, KPMFG, WC5,  NUC_MAX )
13: c##>
14: C=<MVP>=====
15: C PURPOSE: TAKE TALLIES (NEUTRON)
16: C CALLED IN: FLONE, FLIGHT, NEUTR
17: C=====
18: C
19: C JREAC : 1 = TAKE TALLIES OF REACTION RATES OTHER THAN FLUX,
20: C          TOTAL & PRODUCTION
21: C          0 = TAKE TALLIES OF ONLY FLUX, TOTAL & PRODUCTION
22: C* JZONE : 1 = ALL ZONES (FROM FLIGHT)
23: C*          -1 = ALL ZONES (FROM NEUTR)
24: C*          0 = SINGLE ZONE (FROM FLONE)
25: C JREGN : 1 = MULTI REGION (FROM FLIGHT & FLONE )
26: C          -1 = MULTI REGION (FROM NEUTR)
27: C          0 = SINGLE REGION (FROM FLONE)
28: C JZONE : 1 = MULTI ZONE
29: C          0 = SINGLE ZONE
30: C FLX   : FLUX      NN : NUMBER OF PARTICLES
31: C IEG   : energy GROUP # (value is changed in this routine!!!)
32: C IRG   : REGION #
33: C IBP   : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
34: C          JREAC=1 OR JEIGN.NE.0
35: C IMB   : VALUE OF MMAC(IBP,1) FOR EACH PARTICLE (NECESSARY ONLY WHEN
36: C          JREAC=1 OR JEIGN.NE.0
37: C IZN   : ZONE #
38: C DNF   : WORK AREA (ADDED FEB.19,1993)
39: C=====
40:      implicit real*8(A-H,O-Z)
41: C
42:      real*8 FLX(NN)
43:      integer IEG(NN), IBP(NN), IMB(NN), IRG(NN), IZN(NN)
44: C
45: C ... FLXX : FLUX TALLY BY ARABITRARY ESTIMATOR (FLTR /FLCL) .....
46: C ... REXX : FLUX TALLY BY ARABITRARY ESTIMATOR (RETR /RECL) .....
47: C WC1 : PRODUCTION
48: C WC2 : LOSS
49: C
50: C .... TEMPORARY NAMES FOR FLUX & REACTION RATE ARRAY ....
51: C          & EIGENVALUE TALLY
52: C
53:      real*8 FLXX(NGROUP,NREG), REXX(NREG,NRESP), WC1, WC2
54: C
55:      real*8 RMIC(NGROUP,NREG,NUC,NEMIC), RSTAL(NREG,NSTAL),
56:      &      DNFLX(NGROUP,NREG,NUC)
57:      real RESP(NGROUP,NRESP), SGTAL(NBANK,NSTAL)
58:      real SMIC(NBANK,NUC,NSMIC), SMAC(NBANK,MB,NSMAC)
59: c##<2007/03/14:PN3:
60: c##      real DNZON(NUC,NZONE)
61: c##      real DNZON(NUC,NZONE,2)
62: c##>
63: c##<2007/03/14:PN4:
64: c##      integer KPNFG(NBANK)
65: c##      real*8 WC5

```

```

66: c##>
67:      integer LMAC(8), LMIC(8), LEMIC(8,2,2)
68: c+beff1
69: c##<2007/03/14:PN3:
70: c##      real RNU(NBANK,NUC,3)
71: c##      real RNU(NBANK,NUC_MAX,3)
72: c##>
73:      integer IBNK(NBANK,*)
74:      integer KIBNK(0:MIBNK)
75: c-beff1
76: C
77: C ... working array
78: C
79:      real*8 DNF(NN)
80: C
81: C-----
82: C
83: c+beff1
84: c
85: c ... beta-effective ...
86: c WC3 : constant weight function
87: c WC4 : Kobayashi's impotance function
88: c WC6 : fission reaction weight by delayed neutrons
89: c WC7 : fission reaction weight by total neutrons
90: c
91: c-beff1
92: C
93: C***** TALLIES FOR ONE REGION & ONE ZONE *****
94: C
95: C
96:      if ( JREGN.eq.0 ) then
97: C
98:          IR1 = IRG(1)
99:          IZ1 = IZN(1)
100: C
101: C .... FLUX TALLY ....
102: C
103:      if ( NGP1.gt.1 ) then
104:          do 100 I = 1, NN
105:              FLXX(IEG(I),IR1) = FLXX(IEG(I),IR1) + FLX(I)
106:          100 continue
107:      else
108:          IE = IEG(1)
109:          do 110 I = 1, NN
110:              FLXX(IE,IR1) = FLXX(IE,IR1) + FLX(I)
111:          110 continue
112:      end if
113: C
114: C .... EIGENVALUE PROBLEM AND NON VOID REGION ....
115: C
116:      if ( JEIGN.ne.0.and.IMB(1).gt.0 ) then
117:          do 120 I = 1, NN
118:              WC1 = WC1 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
119:              WC2 = WC2 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(8))
120: c##<2007/03/14:PN4:
121: c##      if ( JPHNU.ne.0 ) then
122: c##          if ( KPNFG(IBP(I)).gt.0 ) then
123: c##              WC5 = WC5 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
124: c##          end if
125: c##      end if
126: c##>
127:      120 continue
128: c+beff1
129:      if( JBEFF.ne.0 ) then
130:          do I = 1, NN

```

src/mvp/tally.f

```

131:      WC4 = WC4 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
132:      &      *dble( IBNK( IBP(I), KIBNK(20)))
133:      WC6 = WC6 + FLX(I)*SMAC( IBP(I), IMB(I), LMAC(3))
134:      &      *dble( IBNK( IBP(I), KIBNK(20)))
135:      WC7 = WC7 + FLX(I)*SMAC( IBP(I), IMB(I), LMAC(3))
136: c      WC6 = WC6 + FLX(I)*SMAC( IBP(I), IMB(I), LMAC(2))
137: c      &      *dble( IBNK( IBP(I), KIBNK(20)))
138: c      WC7 = WC7 + FLX(I)*SMAC( IBP(I), IMB(I), LMAC(2))
139:      end do
140:      end if
141: c-beff1
142:      end if
143: c
144:      if( JRESP.ne.0 ) then
145:      do 140 N = 1, NSTAL
146:      do 130 I = 1, NN
147:      RSTAL( IRL,N) = RSTAL( IRL,N) + SGTAL( IBP(I),N)*FLX(I)
148:      130      continue
149:      140      continue
150:      end if
151: c
152: c+beff1
153:      if( JEIGN.ne.0 .and. JBEFF.ne.0 ) then
154:      do 500 N = 1, NUC
155: c##<2007/03/14:PN3:
156: c##      DN      = DNZON(N,IZ1)
157:      DN      = DNZON(N,IZ1,1)
158: c##>
159:      if( DN.ne.0.0 ) then
160:      do 510 I = 1, NN
161:      DNF(I) = DN*FLX(I)
162:      510      continue
163: *VOCL LOOP, SCALAR
164:      do 520 I = 1, NN
165:      WC3 = WC3 + RNU( IBP(I),N,2)
166:      &      *SMIC( IBP(I),N,LMIC(3))*DNF(I)
167:      520      continue
168:      end if
169:      500      continue
170:      end if
171: c-beff1
172: c
173:      if ( JREAC.ne.0.and.NEMIC.ne.0 ) then
174: c
175:      do 220 N = 1, NUC
176: c##<2007/03/14:PN3:
177: c##      DN      = DNZON(N,IZ1)
178:      DN      = DNZON(N,IZ1,1)
179: c##>
180:      if ( DN.ne.0.0 ) then
181:      if ( NGP1.gt.1 ) then
182:      do 150 I = 1, NN
183:      DNF(I) = DN*FLX(I)
184:      150      continue
185: *VOCL LOOP, SCALAR
186:      do 160 I = 1, NN
187:      DNFLX( IEG(I), IRL,N) = DNFLX( IEG(I), IRL,N)
188:      &      + DNF(I)
189:      160      continue
190: c
191:      do 180 J = 1, NEMIC
192: *VOCL LOOP, SCALAR
193:      do 170 I = 1, NN
194:      IE      = IEG(I)
195:      RMIC( IE, IRL,N,J) = RMIC( IE, IRL,N,J)

```

```

196:      &      + SMIC( IBP(I),N,LMIC( LEMIC(J,2,1))) *
197:      &      DNF(I)
198:      170      continue
199:      180      continue
200: c
201: c      .... NGP1 = 1 (vectorizable)
202: c
203:      else
204:      IE      = IEG(1)
205:      do 190 I = 1, NN
206:      DNF(I) = DN*FLX(I)
207:      DNFLX( IE, IRL,N) = DNFLX( IE, IRL,N) + DNF(I)
208:      190      continue
209: c
210:      do 210 J = 1, NEMIC
211:      do 200 I = 1, NN
212:      RMIC( IE, IRL,N,J) = RMIC( IE, IRL,N,J)
213:      &      + SMIC( IBP(I),N,LMIC( LEMIC(J,2,1))) *
214:      &      DNF(I)
215:      200      continue
216:      210      continue
217:      end if
218: c
219:      end if
220:      220      continue
221: c
222: c
223:      end if
224: c
225: c
226: c
227: c**** TALLY INCLUDING MULTIPLE REGIONS ****
228: c      ( JREGN = 1 OR -1 )
229: c
230: c
231:      else
232: c
233: c
234:      if ( JEIGN.ne.0 ) then
235: c
236: c      ( VOID REGION ( IMB(I) = 0 ) MAY APPEAR.)
237: c
238:      if ( JREGN.eq.1
239:      &      .and.( JZONE.eq.0.and.IMB(1).ne.0.or.JZONE.eq.1 ) ) then
240: c
241:      do 230 I = 1, NN
242:      if ( IMB(I).gt.0 ) then
243:      SNUF      = SMAC( IBP(I),IMB(I),LMAC(2))
244:      WC1      = WC1 + FLX(I)*SNUF
245:      WC2      = WC2 + FLX(I)*SMAC( IBP(I),IMB(I),LMAC(8))
246: c##<2007/03/14:PN4:
247:      if ( JPHNU.ne.0 ) then
248:      if ( KPNFG( IBP(I)).gt.0 ) then
249:      WC5 = WC5 + FLX(I)*SNUF
250:      end if
251:      end if
252: c##>
253:      end if
254:      230      continue
255: c+beff1
256:      if( JBEFF.ne.0 ) then
257:      do I = 1, NN
258:      if( IMB(I).gt.0 ) then
259:      WC4 = WC4 + FLX(I)*SMAC( IBP(I),IMB(I),LMAC(2))
260:      &      *dble( IBNK( IBP(I), KIBNK(20)))

```

src/mvp/tally.f

```

261:          WC6 = WC6 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(3))
262:      &          *dble(IBNK(IBP(I),KIBNK(20)))
263:          WC7 = WC7 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(3))
264: c          WC6 = WC6 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
265: c      &          *dble(IBNK(IBP(I),KIBNK(20)))
266: c          WC7 = WC7 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
267:      end if
268:      end do
269:      end if
270: c-beff1
271:      end if
272: c
273: c      ( VOID ZONE ( IMB(I) = 0 ) DOES NOT APPEAR WHEN CALLED
274: c      IN 'NEUTR' )
275: c
276:      if ( JREGN.eq.1 ) then
277:          do 240 I = 1, NN
278:              SNUF = SMAC(IBP(I),IMB(I),LMAC(2))
279:              WC1 = WC1 + FLX(I)*SNUF
280:              WC2 = WC2 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(8))
281: c##<2007/03/14:PN4:
282:              if ( JPHNU.ne.0 ) then
283:                  if ( KPNFG(IBP(I)).gt.0 ) then
284:                      WC5 = WC5 + FLX(I)*SNUF
285:                  end if
286:              end if
287: c##>
288:              240      continue
289: c+beff1
290:              if( JBEFF.ne.0 ) then
291:                  do I = 1, NN
292:                      WC4 = WC4 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
293:                      &          *dble(IBNK(IBP(I),KIBNK(20)))
294:                      WC6 = WC6 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(3))
295:                      &          *dble(IBNK(IBP(I),KIBNK(20)))
296:                      WC7 = WC7 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(3))
297: c                      WC6 = WC6 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
298: c      &          *dble(IBNK(IBP(I),KIBNK(20)))
299: c                      WC7 = WC7 + FLX(I)*SMAC(IBP(I),IMB(I),LMAC(2))
300:                  end do
301:              end if
302: c-beff1
303:              end if
304:          end if
305: c
306: c      DO 1200 N=1,NRESP
307: c      DO 1200 I=1,NN
308: c          REXX(IRG(I),N) = REXX(IRG(I),N) + FLX(I)*RESP(IEG(I),N)
309: c      C1200      CONTINUE
310: c
311:      if( JRESP.ne.0 ) then
312:          do 260 N = 1, NSTAL
313:              do 250 I = 1, NN
314:                  RSTAL(IRG(I),N) = RSTAL(IRG(I),N) + SGTAL(IBP(I),N)*
315:                  &          FLX(I)
316:              250      continue
317:              260      continue
318:          end if
319: c+beff1
320:      if( JEIGN.ne.0 .and. JBEFF.ne.0 ) then
321:          if( JZONE.eq.1 ) then
322:              do 530 N = 1, NUC
323:                  do 540 I = 1, NN
324: c##<2007/03/14:PN3:
325: c##          DNF(I) = DNZON(N,IZN(I))*FLX(I)

```

```

326:          DNF(I) = DNZON(N,IZN(I))*FLX(I)
327: c##>
328:      540      continue
329:
330:      do 550 I = 1, NN
331:          if( DNF(I).ne.0.0 ) then
332:              WC3 = WC3 + DNF(I)*RNU(IBP(I),N,2)
333:              &          *SMIC(IBP(I),N,LMIC(3))
334:          end if
335:      550      continue
336:      530      continue
337:      else
338:          do 560 N = 1, NUC
339: c##<2007/03/14:PN3:
340: c##          DN = DNZON(N,IZN(1))
341: c##          DN = DNZON(N,IZN(1),1)
342: c##>
343:          if( DN.ne.0.0 ) then
344:              do 570 I = 1, NN
345:                  DNF(I) = DN*FLX(I)
346:              570      continue
347:
348:              do 590 I = 1, NN
349:                  WC3 = WC3 + DNF(I)*RNU(IBP(I),N,2)
350:                  &          *SMIC(IBP(I),N,LMIC(3))
351:              590      continue
352:          end if
353:      560      continue
354:          end if
355:      end if
356: c-beff1
357: c
358: c      ..... REFERENCE BY (IEG,IRG) --> (IEG, 1)
359: c
360: c      << do not destroy IEG(*) : since Apr 2000 >>
361: c
362:      do 270 I = 1, NN
363:          IEG(I) = IEG(I) + (IRG(I)-1)*NGROUP
364:      270      continue
365: c
366: c
367:      do 280 I = 1, NN
368:          FLXX(IEG(I),1) = FLXX(IEG(I),1) + FLX(I)
369: c      FLXX(IEG(I),IRG(I)) = FLXX(IEG(I),IRG(I)) + FLX(I)
370:      280      continue
371: c
372:      if ( JREAC.ne.0.and.NEMIC.ne.0 ) then
373:          if ( JZONE.eq.1 ) then
374:              do 320 N = 1, NUC
375:                  do 290 I = 1, NN
376: c##<2007/03/14:PN3:
377: c##          DNF(I) = DNZON(N,IZN(I))*FLX(I)
378: c##          DNF(I) = DNZON(N,IZN(I),1)*FLX(I)
379: c##>
380: c
381:          if ( DNF(I).ne.0.0 ) DNFLX(IEG(I),1,N) =
382:          &          DNFLX(IEG(I),1,N) + DNF(I)
383: c
384: c      if ( DNF(I).ne.0.0 ) then
385: c          DNFLX(IEG(I),IRG(I),N) =
386: c      &          DNFLX(IEG(I),IRG(I),N) + DNF(I)
387: c      end if
388: c
389:      290      continue
390:      do 310 J = 1, NEMIC

```

src/mvp/tally.f

```
391:          do 300 I = 1, NN
392:            if ( DNF(I).ne.0.0 ) then
393:              DFS      = DNF(I)*
394:              &          SMIC(IBP(I),N,LMIC(LEMIC(J,2,1)))
395:
396:              RMIC(IEG(I),1,N,J) = RMIC(IEG(I),1,N,J) +
397:              &          DFS
398:
399:              RMIC(IEG(I),IRG(I),N,J) =
400:              CC &          RMIC(IEG(I),IRG(I),N,J) + DFS
401:
402:            end if
403:          C
404:          300      continue
405:          310      continue
406:          320      continue
407:        else
408:          do 360 N = 1, NUC
409:            c##<2007/03/14:PN3:
410:            c##      DN      = DNZON(N,IZN(1))
411:            DN      = DNZON(N,IZN(1),1)
412:            c##>
413:            if ( DN.ne.0.0 ) then
414:              do 330 I = 1, NN
415:                DNF(I) = DN*FLX(I)
416:
417:                DNFLX(IEG(I),1,N) = DNFLX(IEG(I),1,N) + DNF(I)
418:
419:                DNFLX(IEG(I),IRG(I),N) =
420:                CC &          DNFLX(IEG(I),IRG(I),N) + DNF(I)
421:
422:              330      continue
423:              do 350 J = 1, NEMIC
424:                do 340 I = 1, NN
425:                  DFS      = DNF(I)*
426:                  &          SMIC(IBP(I),N,LMIC(LEMIC(J,2,1)))
427:
428:                  RMIC(IEG(I),1,N,J) = RMIC(IEG(I),1,N,J) +
429:                  &          DFS
430:
431:                  RMIC(IEG(I),IRG(I),N,J) =
432:                  CC &          RMIC(IEG(I),IRG(I),N,J) + DFS
433:
434:                C
435:                340      continue
436:                350      continue
437:              end if
438:            360      continue
439:          end if
440:        end if
441:      end if
442:    C
443:    return
444:  end
```


src/mvp/tallyo.f

```

1:      subroutine TALLYO( TITLE, A, CHA,
2:      &                  NTHIST,NGROUP,NGP1,  NGP2,  NPKIND,NREG,
3:      &                  NRESP, NBATCH,NUC,   NPATOM,NEMIC, NEMAC,
4:      &                  NTREG, NSTAL, NSKIP, NHIST, NTASK, NEVENT,WSUM,
5:      &                  WLEK,  SFLTR, SFLCL, SRETR, SRECL, ENGYB,
6:      &                  ENGPB, RVOL,  TRVOL, WCNTR, XSXV, DNREG,
7:      &                  WCXTY, SRMIC, RMICR, XMIC,  RMAC,  RMACR, XMAC,
8:      &                  LEMIC, LEMAC, NUCID, TEMPN,
9:      &                  SRSTR, SRSCl, TFLHS, ITRNM,
10:     &                  TNAMS, NNames, NSTALY, TIMEB, NTIME,
11:     &                  SRMICS, XMICSM, RMACSM, XMACSM,
12:     &                  SMIMU, RMAMU,
13:     &                  UU,      WWK )
14: C
15: C=<MVP>=====
16: C PURPOSE: CALCULATE FLUX, REACTION RATE AND THEIR ERRORS, AND
17: C PRINT OUT THEM, AND OUTPUT TO FILE (FT30F001)
18: C CALLED IN: CADENZ
19: C CALL      : KEFF
20: C=====
21:      implicit real*8(A-H,O-Z)
22: C
23:      real A(*)
24:      character*4 CHA(*)
25: C
26:      include 'INC/_KPIDS'
27:      include 'INC/_NGPS'
28: C
29:      include 'INC/_FLAGS'
30:      include '../shared/INC/_IOUNIT'
31:      include 'INC/_IOUNIT2'
32: Ccccc include '../shared/INC/_TASKDT'
33: C
34:      parameter( NL = 8 )
35:      character*72 TITLE(2)
36: C
37: C/#IF INTEGER8
38: *      integer*8 NTHIST
39: C/#ELSE
40:      integer  NTHIST
41: C/#ENDIF
42: C
43:      real*8 WCNTR(NEVENT,KPLIM)
44:      real*8 WSUM, WLEK
45:      real*8 SFLTR(NGROUP,NTREG,2),
46:      &      SFLCL(NGROUP,NTREG,2), SRETR(NTREG,NRESP,2),
47:      &      SRECL(NTREG,NRESP,2), XSXV(NBATCH,3)
48: C
49:      real*8 SRMIC(NGROUP,NTREG,NUC,NEMIC,2),
50:      &      RMICR(NPKIND,NTREG,NUC,NEMIC,2),
51:      &      XMIC(NGROUP+NPKIND,NTREG,NUC,NEMIC,2),
52:      &      RMAC(NGROUP,NTREG,NEMAC,2), RMACR(NPKIND,NTREG,NEMAC,2),
53:      &      XMAC(NGROUP+NPKIND,NTREG,NEMAC,2), SRSTR(NTREG,NSTAL,2),
54:      &      SRSCl(NTREG,NSTAL,2)
55: c%c &      XMICS(NGP1+1,NGP1+1,NTREG,NUC,2,*),
56: c%c &      XMACS(NGP1+1,NGP1+1,NTREG,2,*),
57:      real*8 SRMICS(NGP1+1,NGP1+1,NTREG,NUC,2,JSCTM,*),
58:      &      RMACSM(NGP1+1,NGP1+1,NTREG,2,JSCTM,*)
59:      real*8 SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,*),
60:      &      RMAMU(NGP1+1,NGP1+2,NTREG,3,*)
61: C
62:      real*8 WCXTY(NGROUP+NPKIND,NTREG), WWK(NGROUP+NPKIND,NTREG)
63: C
64: C      real*8 TFLHS(NPKIND,2)
65: C      real*8 TFLHS(KPLIM,2)

```

```

66: C
67:      character*16 NUCID(NUC)
68:      real*8 TEMPN(NUC)
69: C*****INTEGER      JMICE(8),JMACE(8),LEMIC(16),LEMAC(16),JPRTS(20)
70:      integer LEMIC(16), LEMAC(16)
71: C
72:      real RVOL(NREG), DNREG(NUC,NREG,2), UU(NGROUP)
73:      real ENGYB(NGP1+1)
74:      real ENGPB(NGP2+1)
75:      real TIMEB(NTIME+1)
76:      real TRVOL(NTREG)
77: C
78: C
79:      integer ITRNM(NTREG)
80:      include '../shared/INC/_LNAME'
81:      character*(LNAME) TNAMS(NNames)
82: C
83: C
84: C***** LOCAL VARIABLES *****
85: C
86:      character*8 TIM
87:      character*12 DAT
88:      CHARACTER*128 NAME
89:      character*32 TAG
90:      character*64 FILEV
91: C
92:      include 'INC/_REACC'
93: C
94: C
95: C -----
96: C
97:      NG1 = NGROUP + NPKIND
98:      NSNEU = NGROUP
99: C
100: C .... ACCESS INDICES FOR PARTICLE SUM ...
101: C
102: C      NSNEU & NSPHT FOR (... , NGROUP+NPKIND) ARRAYS
103: C      ISNEU & ISPHT FOR (... , NPKIND) ARRAYS
104: C
105: C      if ( JNEUT.ne.0 ) NSNEU = NGROUP + 1
106: C      ISNEU = NSNEU - NGROUP
107: C
108: C      NSPHT = 1
109: C      ISPHT = 0
110: C      if ( JPHOT.ne.0 ) then
111: C          NSPHT = NSNEU + 1
112: C          ISPHT = NSPHT - NGROUP
113: C      end if
114: C
115: C      KY = 0
116: C      KYM = 0
117: C      DZERO = 0
118: C
119: C
120: C -----
121: C ... OUTPUT TO BINARY-OUTPUT FILE (I)
122: C -----
123: C      >>>>>> VERSION DESRIPTION >>>>>
124: C      write(IORS)
125: C      &'MVP OUTPUT FILE TYPE 3.0 (FROM 01/MAY/1994)
126: C      write(IORS)
127: C      &'MVP OUTPUT FILE TYPE 3.01 (FROM 14/MAY/1995)
128: C      write(IORS)
129: C      &'MVP OUTPUT FILE TYPE 3.03 (FROM 3/AUG/1998)
130: C      write(IORS)

```

src/mvp/tallyo.f

```

131: C      &'MVP OUTPUT FILE TYPE 3.04 (FROM 27/OCT/1998)
132: C
133: C      version 3.05 : add 'NUCLIDE ID' and 'NUCLIDE TEMPERATURE' record
134: C
135: C      FILEV = 'MVP OUTPUT FILE TYPE 3.05 (FROM 3/MAR/1999)'
136: C
137: C      version 3.06 : add additional tags for 'MICRO & MACRO DATA',
138: C      'MACROSCOPIC REACTION RATE' and 'MICROSCOPIC REACTION RATE' record.
139: C
140: C
141: C      version 3.07 : add additional tags for 'MICRO & MACRO SCATTERING
142: C      MUBAR' record
143: C      FILEV = 'MVP OUTPUT FILE TYPE 3.07 (FROM 20/JUL/2005)'
144: C
145: C      version 3.08 : change NTHIST to I8TOI4(NTHIST) and
146: C      add IQNTH, IRNTH
147: C
148: C      FILEV = 'MVP OUTPUT FILE TYPE 3.08 (FROM 11/AUG/2006)'
149: C
150: C      write(IORS) FILEV
151: C
152: C
153: C      >>>>>> DATE & TIME >>>>>
154: C
155: C      call HIZUKE( DAT )
156: C      call JIKAN( TIM )
157: C      write(IORS) 'DATE ', DAT, ' TIME ', TIM
158: C
159: C      >>>>>> PROBLEM TITLE >>>>
160: C
161: C      write(IORS) TITLE
162: C
163: C      >>>>>> WEIGHT , HISTORY ETC. >>>>
164: C
165: C      NPDET = 0
166: C      JADJM = 0
167: C
168: C
169: C      ... Divide NTHIST into 2 parts for more than (2**31-1) histories.
170: C      This is the treatment not to change binary output format.
171: C      It should be modified in the future.
172: C
173: C      I4MAX = 2147483647 ! 2**31-1
174: C      IQNTH = int(NTHIST/I4MAX)
175: C      IRNTH = NTHIST - IQNTH*I4MAX
176: C
177: C      0-----1-----2-----3--
178: C      TAG = 'PROBLEM PARAMETERS
179: C      0-----1-----2-----3--
180: C      write(IORS) TAG, WSUM, I8TOI4(NTHIST), NBATCH, NGROUP, NREG,
181: C      & NRESP, NSKIP,
182: C      & NSTAL, NGP1, NGP2, NTREG, NPDET, NTIME, NSTALY,
183: C      & JNEUT, JPHOT, JRESP,
184: C      & JEIGN, JADJM, JTIME, IQNTH, IRNTH
185: C
186: C
187: C      >>>>>> ENERGY-BIN BOUNDARY & TALLY REGION VOLUME >>>>>
188: C
189: C      0-----1-----2-----3--
190: C      TAG = 'ENERGY & REG.VOLUME
191: C      -----
192: C
193: C      write(IORS) TAG,
194: C      & (ENGYB(I),I=KENG(PKNEUT),KENG(PKNEUT)+NGP(KPNEUT)),
195: C      & (ENGYB(I),I=KENG(KPPHOT),KENG(KPPHOT)+NGP(KPPHOT)),

```

```

196: C      & (TRVOL(I),I=1,NTREG), (RVOL(I),I=1,NREG),
197: C      & (TIMEB(I),I=1,NTIME+1)
198: C
199: C      >>>>>> TALLY REGION NAMES >>>>>
200: C
201: C      0-----1-----2-----3--
202: C      TAG = 'TALLY REGION NAME
203: C      -----
204: C      do 90 KR = 1, NTREG
205: C      if ( ITRNM(KR).lt.0 ) then
206: C      write(IORS) TAG, KR, len(tnams(1)),TNAMS(ABS(ITRNM(KR)))
207: C      else
208: C      call REGNM0( ITRNM(KR), '>', NAME, LNM, A, CHA )
209: C      write(IORS) TAG, KR, LNM, NAME(:LNM)
210: C      end if
211: C      90 continue
212: C-----
213: C      ... UU : LETHARGY WIDTH
214: C-----
215: C      do 100 IG = 1, NGP1
216: C      UU(IG) = LOG(ENGYB(IG)/ENGYB(IG+1))
217: C      100 continue
218: C      if ( NGP2.gt.0 ) then
219: C      do 110 IG = NGP1 + 1, NGP1 + NGP2
220: C      UU(IG) = LOG(ENGPB(IG-NGP1)/ENGPB(IG-NGP1+1))
221: C      110 continue
222: C      end if
223: C      do 110 IK=1,KPLIM
224: C      if ( JKPAR(IK).ne.0 ) then
225: C      KK = KENGP(IK)
226: C      do 100 IG = KNGP(IK),KNGP(IK+1)-1
227: C      UU(IG) = LOG(ENGYB(KK)/ENGYB(KK+1))
228: C      KK = KK + 1
229: C      100 continue
230: C      end if
231: C      110 continue
232: C
233: C-----
234: C      ... CALCULATE AVERAGE & ESTIMATED STANDARD DEVIATION (FLUX) ...
235: C-----
236: C
237: C      if ( (NBATCH-NSKIP).lt.2 ) then
238: C      write(IOT,*) '!!! CANNOT CALCULATE VARIANCE BECAUSE BATCH ',
239: C      & 'NUMBER (NBATCH-NSKIP) IS LESS THAN 2. '
240: C      DNB = 0.0
241: C      else
242: C      if ( JEIGN.eq.0 ) then
243: C      DNB = 1.0/DBLE(NBATCH-1-NSKIP)
244: C      else
245: C      DNB = 1.0/DBLE(NTASK*(NBATCH-NSKIP)-1)
246: C      end if
247: C      end if
248: C
249: C      DN = 1.0/WSUM
250: C
251: C
252: C      do 130 KR = 1, NTREG
253: C      do 120 IG = 1, NGROUP
254: C
255: C      SFLTR(IG,KR,2) =
256: C      & Sqrt(ABS(SFLTR(IG,KR,2)-(SFLTR(IG,KR,1)**2)*DN)*DN*
257: C      & DNB)
258: C      SFLTR(IG,KR,1) = SFLTR(IG,KR,1)*DN
259: C      if ( SFLTR(IG,KR,1).ne.0. ) SFLTR(IG,KR,2) = SFLTR(IG,KR,2)
260: C      & /SFLTR(IG,KR,1)*100.0

```

src/mvp/tallyo.f

```

261: C
262:         SFLCL(IG,KR,2) =
263:         &         SQRT(ABS(SFLCL(IG,KR,2)-(SFLCL(IG,KR,1)**2)*DN)*DN*
264:         &         DNB)
265:         SFLCL(IG,KR,1) = SFLCL(IG,KR,1)*DN
266:         if ( SFLCL(IG,KR,1).ne.0. ) SFLCL(IG,KR,2) = SFLCL(IG,KR,2)
267:         &         /SFLCL(IG,KR,1)*100.0
268: 120 continue
269: 130 continue
270: C
271: C ..... TO UNIT LETHARGY & UNIT VOLUME .....
272: C
273: do 150 KR = 1, NTREG
274: do 140 IG = 1, NGROUP
275:         SFLTR(IG,KR,1) = SFLTR(IG,KR,1) /UU(IG) /TRVOL(KR)
276:         SFLCL(IG,KR,1) = SFLCL(IG,KR,1) /UU(IG) /TRVOL(KR)
277: 140 continue
278: 150 continue
279: C
280: C-----
281: C .... OUTPUT TO BINARY FILE (II)   FLUXES & THEIR ERROR
282: C-----
283: C
284: C -----0-----1-----2-----3-----
285: TAG = 'FLUX & ERROR: TRACK LENGTH'
286: C -----
287: C
288: do 160 KR = 1, NTREG
289: write(IORS) TAG, (SFLTR(IG,KR,1),IG=1,NGROUP),
290: & (SFLTR(IG,KR,2),IG=1,NGROUP)
291: 160 continue
292: C
293: C -----0-----1-----2-----3-----
294: TAG = 'FLUX & ERROR: COLLISION'
295: C -----
296: C
297: do 170 KR = 1, NTREG
298: write(IORS) TAG, (SFLCL(IG,KR,1),IG=1,NGROUP),
299: & (SFLCL(IG,KR,2),IG=1,NGROUP)
300: 170 continue
301: C
302: C-----
303: C .... REACTION RATES USING RESPONSE FUNCTION
304: C-----
305: C
306: if ( JRESP.ne.0 ) then
307: do 190 N = 1, NRESP
308: do 180 KR = 1, NTREG
309: C
310:         SRETR(KR,N,2) =
311:         &         SQRT(ABS(SRETR(KR,N,2)-(SRETR(KR,N,1)**2)*DN)*DN*
312:         &         DNB)
313:         SRETR(KR,N,1) = SRETR(KR,N,1)*DN
314:         if ( SRETR(KR,N,1).ne.0. ) SRETR(KR,N,2) =
315:         &         SRETR(KR,N,2) /SRETR(KR,N,1)*100.
316: C
317:         SRECL(KR,N,2) =
318:         &         SQRT(ABS(SRECL(KR,N,2)-(SRECL(KR,N,1)**2)*DN)*DN*
319:         &         DNB)
320:         SRECL(KR,N,1) = SRECL(KR,N,1)*DN
321:         if ( SRECL(KR,N,1).ne.0. ) SRECL(KR,N,2) =
322:         &         SRECL(KR,N,2) /SRECL(KR,N,1)*100.
323: 180 continue
324: 190 continue
325: C

```

```

326: do 210 N = 1, NRESP
327: do 200 KR = 1, NTREG
328:         SRETR(KR,N,1) = SRETR(KR,N,1) /TRVOL(KR)
329:         SRECL(KR,N,1) = SRECL(KR,N,1) /TRVOL(KR)
330: 200 continue
331: 210 continue
332: C
333: do 230 I = 1, NSTAL
334: do 220 KR = 1, NTREG
335:         SRSTR(KR,I,2) =
336:         &         SQRT(ABS(SRSTR(KR,I,2)-(SRSTR(KR,I,1)**2)*DN)*DN*
337:         &         DNB)
338:         SRSTR(KR,I,1) = SRSTR(KR,I,1)*DN
339:         if ( SRSTR(KR,I,1).ne.0. ) SRSTR(KR,I,2) =
340:         &         SRSTR(KR,I,2) /SRSTR(KR,I,1)*100.
341:         SRSCCL(KR,I,2) =
342:         &         SQRT(ABS(SRSCCL(KR,I,2)-(SRSCCL(KR,I,1)**2)*DN)*DN*
343:         &         DNB)
344:         SRSCCL(KR,I,1) = SRSCCL(KR,I,1)*DN
345:         if ( SRSCCL(KR,I,1).ne.0. ) SRSCCL(KR,I,2) =
346:         &         SRSCCL(KR,I,2) /SRSCCL(KR,I,1)*100.
347: 220 continue
348: 230 continue
349: do 250 I = 1, NSTAL
350: do 240 KR = 1, NTREG
351:         SRSTR(KR,I,1) = SRSTR(KR,I,1) /TRVOL(KR)
352:         SRSCCL(KR,I,1) = SRSCCL(KR,I,1) /TRVOL(KR)
353: 240 continue
354: 250 continue
355: end if
356: C
357: C-----
358: C .... MICROSCOPIC REACTION RATES & CROSS SECTION
359: C-----
360: C
361: if ( NEMIC.gt.0 ) then
362:         DNHIST = 1./DBLE(NHIST)
363:         D1 = DBLE(NHIST-1)
364:         if ( DNB.eq.0. ) then
365:                 DNB1 = 0.
366:         else
367:                 DNB1 = 1.D10
368:         end if
369: C
370: do 270 KR = 1, NTREG
371: do 260 IG = 1, NGROUP + NPKIND
372:         NB1 = (WCXTY(IG,KR)+D1)*DNHIST*NTASK
373:         if ( NB1.gt.1 ) then
374:                 WWK(IG,KR) = 1./DBLE(NB1-1)
375:         else
376:                 WWK(IG,KR) = DNB1
377:         end if
378:         if ( WCXTY(IG,KR).gt.0.0 ) WCXTY(IG,KR) = 1./WCXTY(IG,KR)
379: 260 continue
380: 270 continue
381: end if
382: C
383: C*VOCL LOOP, SCALAR
384: if ( JNEUT.ne.0 ) then
385: do 310 L = 1, NEMIC
386: do 300 N = 1, NUC
387: do 290 KR = 1, NTREG
388: C
389: C ..... REACTION RATE (GROUP SUM) ....
390: C

```

```

391:      RMICR(ISNEU,KR,N,L,2) = SQRT(
392:      &      ABS(RMICR(ISNEU,KR,N,L,2)
393:      &      -(RMICR(ISNEU,KR,N,L,1)**2)*DN)*DN*DNB)
394:      RMICR(ISNEU,KR,N,L,1) = RMICR(ISNEU,KR,N,L,1)*DN
395:      if ( RMICR(ISNEU,KR,N,L,1).ne.0.0 )
396:      &      RMICR(ISNEU,KR,N,L,2) = RMICR(ISNEU,KR,N,L,2) /
397:      &      RMICR(ISNEU,KR,N,L,1)*100.0
398: C
399: C   The 'WCXTY' here is inverse of the number of the effective histories
400: C   for cross section tally.
401: C
402:      D1 = WCXTY(NSNEU,KR)
403:      D1B = WWK(NSNEU,KR)
404: C
405:      XMIC(NSNEU,KR,N,L,2) = SQRT(
406:      &      ABS(XMIC(NSNEU,KR,N,L,2)
407:      &      -(XMIC(NSNEU,KR,N,L,1)**2)*D1)*D1*D1B)
408:      XMIC(NSNEU,KR,N,L,1) = XMIC(NSNEU,KR,N,L,1)*D1
409: C
410:      if ( XMIC(NSNEU,KR,N,L,1).ne.0.0 )
411:      &      XMIC(NSNEU,KR,N,L,2) = XMIC(NSNEU,KR,N,L,2) /
412:      &      XMIC(NSNEU,KR,N,L,1)*100.0
413:      if ( XMIC(NSNEU,KR,N,L,2).gt.100. )
414:      &      XMIC(NSNEU,KR,N,L,2) = 100.
415: C
416:      do 280 IG = 1, NGP1
417: C
418: C   ..... REACTION RATE (GROUPWISE) ....
419: C
420:      SRMIC(IG,KR,N,L,2) =
421:      &      SQRT(
422:      &      ABS(SRMIC(IG,KR,N,L,2)
423:      &      -(SRMIC(IG,KR,N,L,1)**2)*DN)*DN*DNB)
424:      SRMIC(IG,KR,N,L,1) = SRMIC(IG,KR,N,L,1)*DN
425:      if ( SRMIC(IG,KR,N,L,1).ne.0.0 )
426:      &      SRMIC(IG,KR,N,L,2) = SRMIC(IG,KR,N,L,2) /
427:      &      SRMIC(IG,KR,N,L,1)*100.0
428: C
429: C   ..... CROSS SECTION ....
430: C
431:      D1 = WCXTY(IG,KR)
432:      D1B = WWK(IG,KR)
433: C
434:      XMIC(IG,KR,N,L,2) =
435:      &      SQRT(
436:      &      ABS(XMIC(IG,KR,N,L,2)
437:      &      -(XMIC(IG,KR,N,L,1)**2)*D1)*D1*D1B)
438: C
439:      XMIC(IG,KR,N,L,1) = XMIC(IG,KR,N,L,1)*D1
440: C
441:      if ( XMIC(IG,KR,N,L,1).ne.0.0 ) XMIC(IG,KR,N,L,2) =
442:      &      XMIC(IG,KR,N,L,2) / XMIC(IG,KR,N,L,1)*100.0
443:      if ( XMIC(IG,KR,N,L,2).gt.100. ) XMIC(IG,KR,N,L,2)
444:      &      = 100.
445: C
446:      280          continue
447:      290          continue
448:      300          continue
449:      310          continue
450: C
451:      if ( JSCTM.ne.0 ) then
452:      &      if ( JMICE(4).ne.0 ) KY = KY + 1
453:      &      if ( JMICE(6).ne.0 ) KY = KY + 1
454:      &      if ( JMICE(7).ne.0 ) then
455:      &      &      KY = KY + 1

```

```

456:         KYN2N = KY
457:     end if
458:     do K = 1, KY
459:         do N = 1, NUC
460:             do KR = 1, NTREG
461:                 do IG2 = 1, NGP1
462:                     do IG1 = 1, NGP1
463: C ..... reaction rate .....
464:                     SRMICSM(IG1,IG2,KR,N,2,1,K) = sqrt(abs(
465:                         & SRMICSM(IG1,IG2,KR,N,2,1,K) -
466:                         & (SRMICSM(IG1,IG2,KR,N,1,1,K)**2)*DN)*
467:                         & DN*DNB)
468:                     SRMICSM(IG1,IG2,KR,N,1,1,K) =
469:                         & SRMICSM(IG1,IG2,KR,N,1,1,K)*DN
470:                     if ( SRMICSM(IG1,IG2,KR,N,1,1,K).gt.DZERO )
471:                         & SRMICSM(IG1,IG2,KR,N,2,1,K) =
472:                         & SRMICSM(IG1,IG2,KR,N,2,1,K)*100/
473:                         & SRMICSM(IG1,IG2,KR,N,1,1,K)
474:                     if ( JSCTM.gt.1 ) then
475:                         do IM = 2, JSCTM
476:                             SRMICSM(IG1,IG2,KR,N,2,IM,K)=sqrt(abs(
477:                                 & SRMICSM(IG1,IG2,KR,N,2,IM,K) -
478:                                 & (SRMICSM(IG1,IG2,KR,N,1,IM,K)**2)*DN)
479:                                 & *DN*DNB)
480:                             SRMICSM(IG1,IG2,KR,N,1,IM,K)=
481:                                 & SRMICSM(IG1,IG2,KR,N,1,IM,K)*DN
482:                             if ( SRMICSM(IG1,IG2,KR,N,1,IM,K).ne.
483:                                 & DZERO ) SRMICSM(IG1,IG2,KR,N,2,IM,K)=
484:                                 & 100*SRMICSM(IG1,IG2,KR,N,2,IM,K)/
485:                                 & abs(SRMICSM(IG1,IG2,KR,N,1,IM,K))
486:                         end do
487:                     end if
488: C ..... probability .....
489:                     c%c XMICSM(IG1,IG2,KR,N,2,K) = sqrt(abs(
490:                         & XMICSM(IG1,IG2,KR,N,2,K) -
491:                         & (XMICSM(IG1,IG2,KR,N,1,K)**2)*DN)*DN*DNB)
492:                     c%c XMICSM(IG1,IG2,KR,N,1,K) =
493:                         & XMICSM(IG1,IG2,KR,N,1,K)*DN
494:                     c%c if ( XMICSM(IG1,IG2,KR,N,1,K).gt.DZERO )
495:                         & XMICSM(IG1,IG2,KR,N,2,K) =
496:                         & XMICSM(IG1,IG2,KR,N,2,K)*100/
497:                         & XMICSM(IG1,IG2,KR,N,1,K)
498:                     c%c if ( XMICSM(IG1,IG2,KR,N,2,K).gt.100.D+0 )
499:                         & XMICSM(IG1,IG2,KR,N,2,K) = 100
500:                     end do
501:                 end do
502: C ..... normalization to 1 .....
503:                 c do IG1 = 1, NGP1
504:                 c RT = 0.d0
505:                 c do IG2 = 1, NGP1
506:                 c RT = RT + SRMICSM(IG1,IG2,KR,N,1,K)
507:                 c end do
508:                 c if ( RT.gt.DZERO ) then
509:                 c do IG2 = 1, NGP1
510:                 c SRMICSM(IG1,IG2,KR,N,1,K) =
511:                 c & SRMICSM(IG1,IG2,KR,N,1,K) / RT
512:                 c end do
513:                 c else
514:                 c do IG2 = 1, NGP1
515:                 c SRMICSM(IG1,IG2,KR,N,1,K) = 0.d0
516:                 c end do
517:                 c end if
518:                 c end do
519:             end do
520:         end do

```

src/mvp/tallyo.f

```

521:          end do
522:        end do
523:
524: c ... Since SRMICSM for N2N = 2*Sigma_s(g->g')*phi, divide by 2 for
525: c   binary output (ft30) and standard output (ft06).
526:
527:        if ( JMICE(7).gt.0 ) then
528:          do N = 1, NUC
529:            do KR = 1, NTREG
530:              do IG2 = 1, NGP1
531:                do IG1 = 1, NGP1
532:                  SRMICSM(IG1,IG2,KR,N,1,1,KYN2N) =
533:                  SRMICSM(IG1,IG2,KR,N,1,1,KYN2N)/2.d0
534:                end do
535:              end do
536:            end do
537:          end do
538:        end if
539:
540:      end if
541: c
542:      if ( JSCMU.ne.0 ) then
543:        if ( KY.le.0 ) then
544:          if ( JMICE(4).ne.0 ) KY = KY + 1
545:          if ( JMICE(6).ne.0 ) KY = KY + 1
546:          if ( JMICE(7).ne.0 ) KY = KY + 1
547:        end if
548:        do K = 1, KY
549:          do N = 1, NUC
550:            do KR = 1, NTREG
551:              do IG2 = 1, NGP1+1
552:                do IG1 = 1, NGP1
553:                  if ( SMIMU(IG1,IG2,KR,N,3,K).gt.0ZERO ) then
554:                    WMU = SMIMU(IG1,IG2,KR,N,1,K)
555:                    WM2 = SMIMU(IG1,IG2,KR,N,2,K)
556:                    WM3 = SMIMU(IG1,IG2,KR,N,3,K)
557:                    XMU = WMU / WM3
558:                    WMA = abs( WM2 - (WMU**2) / WM3 )
559:                    if ( WMA.lt.1.0d-12 ) then
560:                      SMIMU(IG1,IG2,KR,N,2,K) = 100
561:                    else
562:                      WMQ = sqrt( WMA / WM3 )
563:                      WMP = WMQ*100 / XMU
564:                      if ( WMP.gt.100. ) then
565:                        SMIMU(IG1,IG2,KR,N,2,K) = 100
566:                      else
567:                        SMIMU(IG1,IG2,KR,N,2,K) = WMP
568:                      end if
569:                    end if
570:                    SMIMU(IG1,IG2,KR,N,1,K) = XMU - 1
571:                  end if
572:                end do
573:              end do
574:            end do
575:          end do
576:        end do
577:      end if
578:    end if
579: c
580: CTEMP ****
581: NEMICP = 0
582: C*VOCL LOOP, SCALAR
583: if ( JPHOT.ne.0 ) then
584:   do 350 L = 1, NEMICP
585:     do 340 N = 1, NPATOM

```

```

586:       do 330 KR = 1, NTREG
587:     C
588:     C ..... REACTION RATE (GROUP SUM) ....
589:     C
590:       RMICR(ISPHT,KR,N,L,2) = SQRT(
591:         & ABS(RMICR(ISPHT,KR,N,L,2)
592:         & -(RMICR(ISPHT,KR,N,L,1)**2)*DN)*DN*DNB)
593:     C
594:       RMICR(ISPHT,KR,N,L,1) = RMICR(ISPHT,KR,N,L,1)*DN
595:     C
596:       if ( RMICR(ISPHT,KR,N,L,1).ne.0.0 )
597:         & RMICR(ISPHT,KR,N,L,2) = RMICR(ISPHT,KR,N,L,2) /
598:         & RMICR(ISPHT,KR,N,L,1)*100.0
599:     C
600:     C ..... CROSS SECTION (GROUP AVERAGE) ....
601:     C
602:       D1 = WCXTY(NSPHT,KR)
603:       D1B = WWK(NSPHT,KR)
604:     C
605:       XMIC(NSPHT,KR,N,L,2) = SQRT(
606:         & ABS(XMIC(NSPHT,KR,N,L,2)
607:         & -(XMIC(NSPHT,KR,N,L,1)**2)*D1)*D1*D1B)
608:     C
609:       XMIC(NSPHT,KR,N,L,1) = XMIC(NSPHT,KR,N,L,1)*D1
610:     C
611:       if ( XMIC(NSPHT,KR,N,L,1).ne.0.0 )
612:         & XMIC(NSPHT,KR,N,L,2) = XMIC(NSPHT,KR,N,L,2) /
613:         & XMIC(NSPHT,KR,N,L,1)*100.0
614:     C
615:       if ( XMIC(NSPHT,KR,N,L,2).gt.100. )
616:         & XMIC(NSPHT,KR,N,L,2) = 100.
617:     C
618:       do 320 IG = NGP1 + 1, NGP1 + NGP2
619:     C
620:     C ..... REACTION RATE (GROUPWISE) ....
621:     C
622:       SRMIC(IG,KR,N,L,2) =
623:         & SQRT(
624:         & ABS(SRMIC(IG,KR,N,L,2)
625:         & -(SRMIC(IG,KR,N,L,1)**2)*DN)*DN*DNB)
626:     C
627:       SRMIC(IG,KR,N,L,1) = SRMIC(IG,KR,N,L,1)*DN
628:     C
629:       if ( SRMIC(IG,KR,N,L,1).ne.0.0 )
630:         & SRMIC(IG,KR,N,L,2) = SRMIC(IG,KR,N,L,2) /
631:         & SRMIC(IG,KR,N,L,1)*100.0
632:     C
633:     C ..... CROSS SECTION ....
634:     C
635:       D1 = WCXTY(IG,KR)
636:       D1B = WWK(IG,KR)
637:     C
638:       XMIC(IG,KR,N,L,2) =
639:         & SQRT(
640:         & ABS(XMIC(IG,KR,N,L,2)
641:         & -(XMIC(IG,KR,N,L,1)**2)*D1)*D1*D1B)
642:     C
643:       XMIC(IG,KR,N,L,1) = XMIC(IG,KR,N,L,1)*D1
644:     C
645:       if ( XMIC(IG,KR,N,L,1).ne.0.0 ) XMIC(IG,KR,N,L,2) =
646:         & XMIC(IG,KR,N,L,2) /XMIC(IG,KR,N,L,1)*100.0
647:     C
648:       if ( XMIC(IG,KR,N,L,2).gt.100. ) XMIC(IG,KR,N,L,2)
649:         & = 100.
650:     C

```

src/mvp/tallyo.f

```

651: 320          continue
652: 330          continue
653: 340          continue
654: 350          continue
655:      end if
656: C
657: C-----
658: C .... MACROSCOPIC REACTION RATES & CROSS SECTION
659: C-----
660: C
661: C      if ( JNEUT.ne.0 ) then
662: C          do 380 L = 1, NEMAC
663: C              do 370 KR = 1, NTREG
664: C
665: C          ..... REACTION RATE (GROUP SUM) ....
666: C
667: C              RMACR(ISNEU,KR,L,2) =
668: C              &          SQRT(
669: C              &          ABS(RMACR(ISNEU,KR,L,2)-(RMACR(ISNEU,KR,L,1)**2)*
670: C              &          DN)*DN*DNB)
671: C
672: C              RMACR(ISNEU,KR,L,1) = RMACR(ISNEU,KR,L,1)*DN
673: C
674: C              if ( RMACR(ISNEU,KR,L,1).ne.0.0 ) RMACR(ISNEU,KR,L,2) =
675: C              &          RMACR(ISNEU,KR,L,2) /RMACR(ISNEU,KR,L,1)*100.0
676: C
677: C          ..... CROSS SECTION (GROUP AVERAGE) ....
678: C
679: C              D1      = WCXTY(NSNEU,KR)
680: C              D1B     = WWK(NSNEU,KR)
681: C
682: C              XMAC(NSNEU,KR,L,2) =
683: C              &          SQRT(
684: C              &          ABS(XMAC(NSNEU,KR,L,2)-(XMAC(NSNEU,KR,L,1)**2)*D1
685: C              &          )*D1*D1B)
686: C
687: C              XMAC(NSNEU,KR,L,1) = XMAC(NSNEU,KR,L,1)*D1
688: C
689: C              if ( XMAC(NSNEU,KR,L,1).ne.0.0 ) XMAC(NSNEU,KR,L,2) =
690: C              &          XMAC(NSNEU,KR,L,2) /XMAC(NSNEU,KR,L,1)*100.0
691: C
692: C              if ( XMAC(NSNEU,KR,L,2).gt.100. ) XMAC(NSNEU,KR,L,2) =
693: C              &          100.
694: C
695: C              do 360 IG = 1, NGP1
696: C
697: C          ..... REACTION RATE (GROUPWISE) ....
698: C
699: C              RMAC(IG,KR,L,2) =
700: C              &          SQRT(
701: C              &          ABS(RMAC(IG,KR,L,2)-(RMAC(IG,KR,L,1)**2)*DN)*
702: C              &          DN*DNB)
703: C
704: C              RMAC(IG,KR,L,1) = RMAC(IG,KR,L,1)*DN
705: C
706: C              if ( RMAC(IG,KR,L,1).ne.0.0 ) RMAC(IG,KR,L,2) =
707: C              &          RMAC(IG,KR,L,2) /RMAC(IG,KR,L,1)*100.0
708: C
709: C          ..... CROSS SECTION ....
710: C
711: C              D1      = WCXTY(IG,KR)
712: C              D1B     = WWK(IG,KR)
713: C
714: C              XMAC(IG,KR,L,2) =
715: C              &          SQRT(

```

```

716:      &          ABS(XMAC(IG,KR,L,2)-(XMAC(IG,KR,L,1)**2)*D1)*
717:      &          D1*D1B)
718: C
719: C              XMAC(IG,KR,L,1) = XMAC(IG,KR,L,1)*D1
720: C
721: C              if ( XMAC(IG,KR,L,1).ne.0.0 ) XMAC(IG,KR,L,2) =
722: C              &          XMAC(IG,KR,L,2) /XMAC(IG,KR,L,1)*100.0
723: C
724: C              if ( XMAC(IG,KR,L,2).gt.100. ) XMAC(IG,KR,L,2) = 100.
725: C          360          continue
726: C          370          continue
727: C          380          continue
728: C
729: C      if ( JSCTM.ne.0 ) then
730: C          if ( JMACE(4).gt.0 ) KYM = KYM + 1
731: C          if ( JMACE(6).gt.0 ) KYM = KYM + 1
732: C          if ( JMACE(7).gt.0 ) then
733: C              KYM = KYM + 1
734: C              KYN2N = KYM
735: C          end if
736: C          do K = 1, KYM
737: C              do KR = 1, NTREG
738: C                  do IG2 = 1, NGP1
739: C                      do IG1 = 1, NGP1
740: C          ..... reaction rate .....
741: C                      RMACSM(IG1,IG2,KR,2,1,K) = sqrt(abs(
742: C                      &          RMACSM(IG1,IG2,KR,2,1,K) -
743: C                      &          (RMACSM(IG1,IG2,KR,1,1,K)**2)*DN)*DN*DNB)
744: C                      RMACSM(IG1,IG2,KR,1,1,K) =
745: C                      &          RMACSM(IG1,IG2,KR,1,1,K)*DN
746: C                      if ( RMACSM(IG1,IG2,KR,1,1,K).gt.DZERO )
747: C                      &          RMACSM(IG1,IG2,KR,2,1,K) =
748: C                      &          RMACSM(IG1,IG2,KR,2,1,K)*100/
749: C                      &          RMACSM(IG1,IG2,KR,1,1,K)
750: C                      if ( JSCTM.gt.1 ) then
751: C                          do IM = 2, JSCTM
752: C                              RMACSM(IG1,IG2,KR,2,IM,K) = sqrt(abs(
753: C                              &          RMACSM(IG1,IG2,KR,2,IM,K) -
754: C                              &          (RMACSM(IG1,IG2,KR,1,IM,K)**2)*DN)*
755: C                              &          DN*DNB)
756: C                              RMACSM(IG1,IG2,KR,1,IM,K) =
757: C                              &          RMACSM(IG1,IG2,KR,1,IM,K)*DN
758: C                              if ( RMACSM(IG1,IG2,KR,1,IM,K).ne.DZERO )
759: C                              &          RMACSM(IG1,IG2,KR,2,IM,K) =
760: C                              &          100*RMACSM(IG1,IG2,KR,2,IM,K)/
761: C                              &          abs(RMACSM(IG1,IG2,KR,1,IM,K))
762: C                              end do
763: C                          end if
764: C          ..... probability .....
765: C                      c%c          XMACSM(IG1,IG2,KR,2,K) = sqrt(abs(
766: C                      c%c          &          XMACSM(IG1,IG2,KR,2,K) -
767: C                      c%c          &          (XMACSM(IG1,IG2,KR,1,K)**2)*DN)*DN*DNB)
768: C                      c%c          XMACSM(IG1,IG2,KR,1,K)=XMACSM(IG1,IG2,KR,1,K)*DN
769: C                      c%c          if ( XMACSM(IG1,IG2,KR,1,K).gt.DZERO )
770: C                      c%c          &          XMACSM(IG1,IG2,KR,2,K) =
771: C                      c%c          &          XMACSM(IG1,IG2,KR,2,K)*100/
772: C                      c%c          &          XMACSM(IG1,IG2,KR,1,K)
773: C                      c%c          if ( XMACSM(IG1,IG2,KR,2,K).gt.100.D+0 )
774: C                      c%c          &          XMACSM(IG1,IG2,KR,2,K) = 100
775: C                      end do
776: C                      end do
777: C          ..... normalization to 1 .....
778: C          c          do IG1 = 1, NGP1
779: C          c          RT = 0.d0
780: C          c          do IG2 = 1, NGP1

```

src/mvp/tallyo.f

```

781: c          RT = RT + RMACSM(IG1,IG2,KR,1,K)
782: c        end do
783: c        if ( RT.gt.DZERO ) then
784: c          do IG2 = 1, NGP1
785: c            RMACSM(IG1,IG2,KR,1,K) =
786: c      &          RMACSM(IG1,IG2,KR,1,K) / RT
787: c        end do
788: c        else
789: c          do IG2 = 1, NGP1
790: c            RMACSM(IG1,IG2,KR,1,K) = 0.d0
791: c          end do
792: c        end if
793: c      end do
794:
795: c    end do
796: c  end do
797:
798: c ... Since RMACSM for N2N = 2*Sigma_s(g->g')*phi, divide by 2 for
799: c binary output (ft30) and standard output (ft06).
800:
801: c    if ( JMACE(7).gt.0 ) then
802: c      do KR = 1, NTREG
803: c        do IG2 = 1, NGP1
804: c          do IG1 = 1, NGP1
805: c            RMACSM(IG1,IG2,KR,1,1,KYN2N) =
806: c      &            RMACSM(IG1,IG2,KR,1,1,KYN2N)/2.d0
807: c          end do
808: c        end do
809: c      end do
810: c    end if
811:
812: c  end if
813: c
814: c  if ( JSCMU.ne.0 ) then
815: c    if ( KYM.le.0 ) then
816: c      if ( JMACE(4).gt.0 ) KYM = KYM + 1
817: c      if ( JMACE(6).gt.0 ) KYM = KYM + 1
818: c      if ( JMACE(7).gt.0 ) KYM = KYM + 1
819: c    end if
820: c    do K = 1, KYM
821: c      do KR = 1, NTREG
822: c        do IG2 = 1, NGP1+1
823: c          do IG1 = 1, NGP1
824: c            if ( RMAMU(IG1,IG2,KR,3,K).gt.DZERO ) then
825: c              WMU = RMAMU(IG1,IG2,KR,1,K)
826: c              WM2 = RMAMU(IG1,IG2,KR,2,K)
827: c              WM3 = RMAMU(IG1,IG2,KR,3,K)
828: c              XMU = WMU / WM3
829: c              WMA = abs( WM2 - (WMU**2) / WM3 )
830: c              if ( WMA.lt.1.0d-12 ) then
831: c                RMAMU(IG1,IG2,KR,2,K) = 100
832: c              else
833: c                WMQ = sqrt( WMA / WM3 )
834: c                WMP = WMQ * 100 / XMU
835: c                if ( WMP.gt.100. ) then
836: c                  RMAMU(IG1,IG2,KR,2,K) = 100
837: c                else
838: c                  RMAMU(IG1,IG2,KR,2,K) = WMP
839: c                end if
840: c              end if
841: c              RMAMU(IG1,IG2,KR,1,K) = XMU - 1
842: c            end if
843: c          end do
844: c        end do
845: c      end do

```

```

846: c        end do
847: c      end if
848: c    end if
849: c
850: c  CTEMP *****
851: c  NEMACP = 0
852: c
853: c  if ( JPHOT.ne.0 ) then
854: c    do 410 L = 1, NEMACP
855: c      do 400 KR = 1, NTREG
856: c
857: c      ..... REACTION RATE (GROUP SUM) ....
858: c
859: c        RMACR(ISPHT,KR,L,2) =
860: c      &          SQRT(
861: c      &          ABS(RMACR(ISPHT,KR,L,2)-(RMACR(ISPHT,KR,L,1)**2)*
862: c      &          DN)*DN*DNB)
863: c
864: c        RMACR(ISPHT,KR,L,1) = RMACR(ISPHT,KR,L,1)*DN
865: c
866: c        if ( RMACR(ISPHT,KR,L,1).ne.0.0 ) RMACR(ISPHT,KR,L,2) =
867: c      &          RMACR(ISPHT,KR,L,2) /RMACR(ISPHT,KR,L,1)*100.0
868: c
869: c      ..... CROSS SECTION (GROUP AVERAGE) ....
870: c
871: c        D1 = WCXTY(NSPHT,KR)
872: c        D1B = WWK(NSPHT,KR)
873: c
874: c        XMAC(NSPHT,KR,L,2) =
875: c      &          SQRT(
876: c      &          ABS(XMAC(NSPHT,KR,L,2)-(XMAC(NSPHT,KR,L,1)**2)*D1
877: c      &          )*D1*D1B)
878: c
879: c        XMAC(NSPHT,KR,L,1) = XMAC(NSPHT,KR,L,1)*D1
880: c
881: c        if ( XMAC(NSPHT,KR,L,1).ne.0.0 ) XMAC(NSPHT,KR,L,2) =
882: c      &          XMAC(NSPHT,KR,L,2) /XMAC(NSPHT,KR,L,1)*100.0
883: c
884: c        if ( XMAC(NSPHT,KR,L,2).gt.100. ) XMAC(NSPHT,KR,L,2) =
885: c      &          100.
886: c
887: c        do 390 IG = NGP1 + 1, NGP1 + NGP2
888: c
889: c        ..... REACTION RATE (GROUPWISE) ....
890: c
891: c          RMAC(IG,KR,L,2) =
892: c        &          SQRT(
893: c        &          ABS(RMAC(IG,KR,L,2)-(RMAC(IG,KR,L,1)**2)*DN)*
894: c        &          DN*DNB)
895: c
896: c          RMAC(IG,KR,L,1) = RMAC(IG,KR,L,1)*DN
897: c
898: c          if ( RMAC(IG,KR,L,1).ne.0.0 ) RMAC(IG,KR,L,2) =
899: c        &          RMAC(IG,KR,L,2) /RMAC(IG,KR,L,1)*100.0
900: c
901: c      ..... CROSS SECTION ....
902: c
903: c        D1 = WCXTY(IG,KR)
904: c        D1B = WWK(IG,KR)
905: c
906: c        XMAC(IG,KR,L,2) =
907: c      &          SQRT(
908: c      &          ABS(XMAC(IG,KR,L,2)-(XMAC(IG,KR,L,1)**2)*D1)*
909: c      &          D1*D1B)
910: c

```

src/mvp/tallyo.f

```

911:          XMAC(IG,KR,L,1) = XMAC(IG,KR,L,1)*D1
912: C
913:          if ( XMAC(IG,KR,L,1).ne.0.0 ) XMAC(IG,KR,L,2) =
914: &          XMAC(IG,KR,L,2) /XMAC(IG,KR,L,1)*100.0
915: C
916:          if ( XMAC(IG,KR,L,2).gt.100. ) XMAC(IG,KR,L,2) = 100.
917: 390      continue
918: 400      continue
919: 410      continue
920:      end if
921: C
922: C-----
923: C ... weighted time of flight
924: C-----
925: C
926:      if( JTIME.ne.0 ) then
927: CCCC      do 412 K = 1, NPKIND
928:          do 412 K = 1, KPLIM
929: C
930:              TFLHS(K,2) =
931: &              SQRT(ABS(TFLHS(K,2)-(TFLHS(K,1)**2)*DN)*DN*
932: &              DNB)
933:              TFLHS(K,1) = TFLHS(K,1)*DN
934:          if ( TFLHS(K,1).ne.0. ) TFLHS(K,2) = TFLHS(K,2)
935: &          /TFLHS(K,1)*100.0
936: C
937: 412      continue
938:      endif
939: C-----
940: C OUTPUT TO BINARY FILE (III) REACTION RATES & THEIR ERROR
941: C-----
942:      if ( JRESP.ne.0 ) then
943: C
944: C          -----0-----1-----2-----3--
945: C          TAG = 'REACTION RATE(RESP):TRACK LENGTH'
946: C          -----
947: C
948:          do 420 N = 1, NRESP
949:              write(IORS) TAG, (SRETR(K,N,1),K=1,NTREG)
950: &              (SRETR(K,N,2),K=1,NTREG)
951: 420      continue
952: C
953: C          -----0-----1-----2-----3--
954: C          TAG = 'REACTION RATE(RESP):COLLISION '
955: C          -----
956: C
957:          do 430 N = 1, NRESP
958:              write(IORS) TAG, (SRECL(K,N,1),K=1,NTREG),
959: &              (SRECL(K,N,2),K=1,NTREG)
960: 430      continue
961: C
962: C          -----0-----1-----2-----3--
963: C          TAG = 'REACTION RATE(STAL):TRACK LENGTH'
964: C          -----
965: C
966:          do 440 N = 1, NSTAL
967:              write(IORS) TAG, (SRSTR(K,N,1),K=1,NTREG),
968: &              (SRSTR(K,N,2),K=1,NTREG)
969: 440      continue
970: C
971: C          -----0-----1-----2-----3--
972: C          TAG = 'REACTION RATE(STAL):COLLISION '
973: C          -----
974: C
975:          do 450 N = 1, NSTAL

```

```

976:          write(IORS) TAG, (SRSC(L,K,N,1),K=1,NTREG),
977: &          (SRSC(L,K,N,2),K=1,NTREG)
978: 450      continue
979:      end if
980: C
981: C-----
982: C OUTPUT TO BINARY FILE (III.2)
983: C          MICRO & MACRO REACTION RATE & CROSS SECTION
984: C-----
985: C
986: C          -----0-----1-----2-----3--
987: C          TAG = 'MICRO & MACRO TALLY DATA '
988: C          -----
989: C
990:          write(IORS) TAG, NUC, NEMIC, NEMAC, JRTTR, JRTCL,
991: &          (JMICE(L),L=1,8), (JMACE(L),L=1,8)
992: &          , KY, KYM
993: &          , JSCTM, JSCMU, NPKIND
994: C
995: C          -----0-----1-----2-----3--
996: C          TAG = 'MICRO & MACRO TALLY DATA 2 '
997: C          -----
998: C
999:          write(IORS) TAG, (LEMIC(L),L=1,16), (LEMAC(L),L=1,16),
1000: &          (NUCID(N)(:16),N=1,NUC)
1001: C
1002: C ... added from file version 3.05
1003: C          -----0-----1-----2-----3--
1004: C          TAG = 'NUCLIDE ID'
1005: C          -----
1006:          write(IORS) TAG, len(NUCID(1)), (NUCID(N),N=1,NUC)
1007: C
1008: C ... added from file version 3.05
1009: C          -----0-----1-----2-----3--
1010: C          TAG = 'NUCLIDE TEMPERATURE'
1011: C          -----
1012:          write(IORS) TAG, (TEMPN(N),N=1,NUC)
1013: C
1014: C          -----0-----1-----2-----3--
1015: C          TAG = 'MICROSCOPIC REACTION RATE '
1016: C          -----
1017: C
1018:          write(IORS) TAG,
1019: &          (((SRMIC(IG,K,N,M,1),IG=1,NGROUP),K=1,NTREG),N=1,NUC),
1020: &          M=1,NEMIC),
1021: &          (((SRMIC(IG,K,N,M,2),IG=1,NGROUP),K=1,NTREG),N=1,NUC),
1022: &          M=1,NEMIC)
1023: C
1024: C          -----0-----1-----2-----3--
1025: C          TAG = 'MICROSCOPIC REACTION RATE 2 '
1026: C          -----
1027: C
1028:          write(IORS) TAG, (
1029: &          (((RMICR(J,K,N,M,1),K=1,NTREG),N=1,NUC),M=1,NEMIC),
1030: &          (((RMICR(J,K,N,M,2),K=1,NTREG),N=1,NUC),M=1,NEMIC),
1031: &          J=1,NPKIND)
1032: C
1033: C          -----0-----1-----2-----3--
1034: C          TAG = 'MICROSCOPIC CROSS SECTION '
1035: C          -----
1036: C
1037: C
1038:          write(IORS) TAG,
1039: &          (((XMIC(IG,K,N,M,1),IG=1,NG1),K=1,NTREG),N=1,NUC),
1040: &          M=1,NEMIC),

```


src/mvp/tallyo.f

```

1041: &      (((XMIC(IG,K,N,M,2),IG=1,NG1),K=1,NTREG),N=1,NUC),
1042: &      M=1,NEMIC)
1043: C
1044: C -----0-----1-----2-----3---
1045: TAG   = 'MACROSCOPIC REACTION RATE'
1046: C -----
1047: C
1048: write(IORS) TAG,
1049: &      (((RMAC(IG,K,M,1),IG=1,NGROUP),K=1,NTREG),M=1,NEMAC),
1050: &      (((RMAC(IG,K,M,2),IG=1,NGROUP),K=1,NTREG),M=1,NEMAC)
1051: C
1052: C -----0-----1-----2-----3---
1053: TAG   = 'MACROSCOPIC REACTION RATE 2'
1054: C -----
1055: C
1056: write(IORS) TAG, (((RMACR(J,K,M,1),K=1,NTREG),M=1,NEMAC),
1057: &      (((RMACR(J,K,M,2),K=1,NTREG),M=1,NEMAC),J=1,NPKIND)
1058: C
1059: C -----0-----1-----2-----3---
1060: TAG   = 'MACROSCOPIC CROSS SECTION'
1061: C -----
1062: C
1063: write(IORS) TAG,
1064: &      (((XMAC(IG,K,M,1),IG=1,NG1),K=1,NTREG),M=1,NEMAC),
1065: &      (((XMAC(IG,K,M,2),IG=1,NG1),K=1,NTREG),M=1,NEMAC)
1066: C
1067: if ( JSCTM.ne.0.and.KY.gt.0 ) then
1068:   KY0 = 0
1069:   if ( JMICE(4).ne.0 ) then
1070:     KY0 = KY0 + 1
1071: C
1072: C -----0-----1-----2-----3---
1073: TAG   = 'MICRO ELAS REAC SCAT MATRIX'
1074: C -----
1075: write(IORS) TAG,
1076: &      (((((SRMICSM(IG1,IG2,K,N,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1077: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM),
1078: &      (((((SRMICSM(IG1,IG2,K,N,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1079: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM)
1080: C
1081: C -----0-----1-----2-----3---
1082: c%c TAG   = 'MICRO ELAS PROB SCAT MATRIX'
1083: C -----
1084: c%c write(IORS) TAG,
1085: c%c &      (((XMICSM(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1),
1086: c%c &      K=1,NTREG),N=1,NUC),
1087: c%c &      (((XMICSM(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1),
1088: c%c &      K=1,NTREG),N=1,NUC)
1089: C
1090: end if
1091: if ( JMICE(6).ne.0 ) then
1092:   KY0 = KY0 + 1
1093: C
1094: C -----0-----1-----2-----3---
1095: TAG   = 'MICRO INEL REAC SCAT MATRIX'
1096: C -----
1097: write(IORS) TAG,
1098: &      (((((SRMICSM(IG1,IG2,K,N,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1099: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM),
1100: &      (((((SRMICSM(IG1,IG2,K,N,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1101: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM)
1102: C
1103: C -----0-----1-----2-----3---
1104: c%c TAG   = 'MICRO INEL PROB SCAT MATRIX'
1105: C -----

```

```

1106: c%c write(IORS) TAG,
1107: c%c &      (((XMICSM(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1),
1108: c%c &      K=1,NTREG),N=1,NUC),
1109: c%c &      (((XMICSM(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1),
1110: c%c &      K=1,NTREG),N=1,NUC)
1111: C
1112: end if
1113: if ( JMICE(7).ne.0 ) then
1114:   KY0 = KY0 + 1
1115: C
1116: C -----0-----1-----2-----3---
1117: TAG   = 'MICRO N2N REAC SCAT MATRIX'
1118: C -----
1119: write(IORS) TAG,
1120: &      (((((SRMICSM(IG1,IG2,K,N,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1121: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM),
1122: &      (((((SRMICSM(IG1,IG2,K,N,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1123: &      K=1,NTREG),N=1,NUC),IM=1,JSCTM)
1124: C
1125: C -----0-----1-----2-----3---
1126: c%c TAG   = 'MICRO N2N PROB SCAT MATRIX'
1127: C -----
1128: c%c write(IORS) TAG,
1129: c%c &      (((XMICSM(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1),
1130: c%c &      K=1,NTREG),N=1,NUC),
1131: c%c &      (((XMICSM(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1),
1132: c%c &      K=1,NTREG),N=1,NUC)
1133: C
1134: end if
1135: end if
1136: if ( JSCTM.ne.0.and.KYM.gt.0 ) then
1137:   KY0 = 0
1138:   if ( JMACE(4).ne.0 ) then
1139:     KY0 = KY0 + 1
1140: C
1141: C -----0-----1-----2-----3---
1142: TAG   = 'MACRO ELAS REAC SCAT MATRIX'
1143: C -----
1144: write(IORS) TAG,
1145: &      (((RMACSM(IG1,IG2,K,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1146: &      K=1,NTREG),IM=1,JSCTM),
1147: &      (((RMACSM(IG1,IG2,K,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1148: &      K=1,NTREG),IM=1,JSCTM)
1149: C
1150: C -----0-----1-----2-----3---
1151: c%c TAG   = 'MACRO ELAS PROB SCAT MATRIX'
1152: C -----
1153: c%c write(IORS) TAG,
1154: c%c &      (((XMACSM(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG),
1155: c%c &      (((XMACSM(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG)
1156: C
1157: end if
1158: if ( JMACE(6).ne.0 ) then
1159:   KY0 = KY0 + 1
1160: C
1161: C -----0-----1-----2-----3---
1162: TAG   = 'MACRO INEL REAC SCAT MATRIX'
1163: C -----
1164: write(IORS) TAG,
1165: &      (((((RMACSM(IG1,IG2,K,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1166: &      K=1,NTREG),IM=1,JSCTM),
1167: &      (((((RMACSM(IG1,IG2,K,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1168: &      K=1,NTREG),IM=1,JSCTM)
1169: C
1170: C -----0-----1-----2-----3---

```

src/mvp/tallyo.f

```

1171: c%c TAG = 'MACRO INEL PROB SCAT MATRIX '
1172: C -----
1173: c%c write(IORS) TAG,
1174: c%c & (((XMACSM(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG),
1175: c%c & (((XMACSM(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG)
1176: C
1177: end if
1178: if ( JMACE(7).ne.0 ) then
1179: KY0 = KY0 + 1
1180: C
1181: C -----0-----1-----2-----3--
1182: TAG = 'MACRO N2N REAC SCAT MATRIX '
1183: C -----
1184: write(IORS) TAG,
1185: & (((RMACSM(IG1,IG2,K,1,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1186: & K=1,NTREG),IM=1,JSCM),
1187: & (((RMACSM(IG1,IG2,K,2,IM,KY0),IG1=1,NGP1),IG2=1,NGP1),
1188: & K=1,NTREG),IM=1,JSCM)
1189: C
1190: C -----0-----1-----2-----3--
1191: c%c TAG = 'MACRO N2N PROB SCAT MATRIX '
1192: C -----
1193: c%c write(IORS) TAG,
1194: c%c & (((XMACSM(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG),
1195: c%c & (((XMACSM(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1),K=1,NTREG)
1196: C
1197: end if
1198: end if
1199: if ( JSCMU.ne.0.and.KY.gt.0 ) then
1200: KY0 = 0
1201: if ( JMICE(4).ne.0 ) then
1202: KY0 = KY0 + 1
1203: C
1204: C -----0-----1-----2-----3--
1205: TAG = 'MICRO ELAS MUBA SCAT '
1206: C -----
1207: write(IORS) TAG,
1208: & (((SMIMU(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1209: & K=1,NTREG),N=1,NUC),
1210: & (((SMIMU(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1211: & K=1,NTREG),N=1,NUC)
1212: C
1213: end if
1214: if ( JMICE(6).ne.0 ) then
1215: KY0 = KY0 + 1
1216: C
1217: C -----0-----1-----2-----3--
1218: TAG = 'MICRO INEL MUBA SCAT '
1219: C -----
1220: write(IORS) TAG,
1221: & (((SMIMU(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1222: & K=1,NTREG),N=1,NUC),
1223: & (((SMIMU(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1224: & K=1,NTREG),N=1,NUC)
1225: C
1226: end if
1227: if ( JMICE(7).ne.0 ) then
1228: KY0 = KY0 + 1
1229: C
1230: C -----0-----1-----2-----3--
1231: TAG = 'MICRO N2N MUBA SCAT '
1232: C -----
1233: write(IORS) TAG,
1234: & (((SMIMU(IG1,IG2,K,N,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1235: & K=1,NTREG),N=1,NUC),

```

```

1236: & (((SMIMU(IG1,IG2,K,N,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),
1237: & K=1,NTREG),N=1,NUC)
1238: C
1239: end if
1240: end if
1241: if ( JSCMU.ne.0.and.KYM.gt.0 ) then
1242: KY0 = 0
1243: if ( JMACE(4).ne.0 ) then
1244: KY0 = KY0 + 1
1245: C
1246: C -----0-----1-----2-----3--
1247: TAG = 'MACRO ELAS MUBA SCAT '
1248: C -----
1249: write(IORS) TAG,
1250: & (((RMAMU(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG),
1251: & (((RMAMU(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG)
1252: C
1253: end if
1254: if ( JMACE(6).ne.0 ) then
1255: KY0 = KY0 + 1
1256: C
1257: C -----0-----1-----2-----3--
1258: TAG = 'MACRO INEL MUBA SCAT '
1259: C -----
1260: write(IORS) TAG,
1261: & (((RMAMU(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG),
1262: & (((RMAMU(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG)
1263: C
1264: end if
1265: if ( JMACE(7).ne.0 ) then
1266: KY0 = KY0 + 1
1267: C
1268: C -----0-----1-----2-----3--
1269: TAG = 'MACRO N2N MUBA SCAT '
1270: C -----
1271: write(IORS) TAG,
1272: & (((RMAMU(IG1,IG2,K,1,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG),
1273: & (((RMAMU(IG1,IG2,K,2,KY0),IG1=1,NGP1),IG2=1,NGP1+1),K=1,NTREG)
1274: C
1275: end if
1276: end if
1277: return
1278: end

```

src/mvp/tallyp.f

```

1:      subroutine TALLYP( IOW, JREAC, JREGN, JZONE, JRESP, FLX, NN,
2:      & IEG, IRG, IBP, DNF, NGROUP,NGP2, NREG,
3:      & NRESP, NUC, NPATOM,NBANK, NSMIC, NSTAL, IZN,
4:      & DNZON, NZONE, LEMIC, NEMICP,RESP, SMIC,
5:      & SGTAL, FLXX, REXX, RMIC, DNFLX, RSTAL )
6: C=<MVP>=====
7: C PURPOSE: TAKE TALLIES (PHOTON)
8: C CALLED IN: FLIONE, FLIGHT, PHOTR
9: C=====
10: C
11: C JREAC : 1 = TAKE TALLIES OF REACTION RATES OTHER THAN FLUX,
12: C          TOTAL & PRODUCTION
13: C          0 = TAKE TALLIES OF ONLY FLUX, TOTAL & PRODUCTION
14: C* JZONE : 1 = ALL ZONES (FROM FLIGHT)
15: C*          -1 = ALL ZONES (FROM PHOTR)
16: C*          0 = SINGLE ZONE (FROM FLIONE)
17: C JREGN : 1 = MULTI REGION (FROM FLIGHT & FLIONE )
18: C          -1 = MULTI REGION (FROM PHOTR)
19: C          0 = SINGLE REGION (FROM FLIONE)
20: C JZONE : 1 = MULTI ZONE
21: C          0 = SINGLE ZONE
22: C FLX : FLUX      NN : NUMBER OF PARTICLES
23: C IEG : GROUP #
24: C IRG : REGION #
25: C IBP : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
26: C       JREAC=1 OR JEIGN=1)
27: C IZN : ZONE #
28: C DNF : WORK AREA (ADDED FEB.19,1993)
29: C=====
30:      implicit real*8(A-H,O-Z)
31: C
32:      real*8 FLX(NN)
33:      integer IEG(NN), IBP(NN), IRG(NN), IZN(NN)
34: C
35: C ... FLXX : FLUX TALLY BY ARABITRARY ESTIMATOR (FLTR /FLCL) .....
36: C ... REXX : FLUX TALLY BY ARABITRARY ESTIMATOR (RETR /RECL) .....
37: C
38:      real*8 FLXX(NGROUP,NREG), REXX(NREG,NRESP)
39:      real*8 RMIC(NGROUP,NREG,NUC,*), RSTAL(NREG,NSTAL),
40:      & DNFLX(NGROUP,NREG,NUC)
41: C
42:      real RESP(NGROUP,NRESP)
43:      real SMIC(NBANK,NUC,NSMIC), SGTAL(NBANK,NSTAL)
44:      real DNZON(NUC,NZONE,2)
45:      integer LEMIC(8,2,2)
46: C
47: C ... working array ...
48: C
49:      real*8 DNF(NN)
50: C
51: C
52: C***** TALLY FOR ONE REGION *****
53: C
54: C
55:      if ( JREGN.eq.0 ) then
56:      if ( NGP2.gt.1 ) then
57: *VOCL LOOP,SCALAR
58:      do 100 I = 1, NN
59:      FLXX(IEG(I),IRG(1)) = FLXX(IEG(I),IRG(1)) + FLX(I)
60: 100 continue
61:      else
62:      IE = IEG(1)
63:      do 110 I = 1, NN
64:      FLXX(IE,IRG(1)) = FLXX(IE,IRG(1)) + FLX(I)
65: 110 continue

```

```

66:      end if
67: C
68: CM      DO 200 N=1,NRESP
69: CM      DO 200 I=1,NN
70: CM      REXX(IRG(1),N)=REXX(IRG(1),N) + FLX(I)*RESP(IEG(I),N)
71: CM200 CONTINUE
72: C
73:      if( JRESP.ne.0 ) then
74:      do 130 N = 1, NSTAL
75:      do 120 I = 1, NN
76:      RSTAL(IRG(1),N) = RSTAL(IRG(1),N) + SGTAL(IBP(I),N)*
77:      & FLX(I)
78: 120 continue
79: 130 continue
80:      endif
81: C
82:      if ( JREAC.ne.0.and.NEMICP.ne.0 ) then
83:      do 210 N = 1, NPATOM
84:      DN = DNZON(N,IZN(1),2)
85:      if ( DN.ne.0.0 ) then
86:      if ( NGP2.gt.1 ) then
87:      do 140 I = 1, NN
88:      DNF(I) = DN*FLX(I)
89:      continue
90: *VOCL LOOP,SCALAR
91:      do 150 I = 1, NN
92:      IE = IEG(I)
93:      DNFLX(IE,IRG(1),N) = DNFLX(IE,IRG(1),N) +
94:      & DNF(I)
95: 150 continue
96: C
97:      do 170 J = 1, NEMICP
98: *VOCL LOOP,SCALAR
99:      do 160 I = 1, NN
100:      IE = IEG(I)
101:      RMIC(IE,IRG(1),N,J) = RMIC(IE,IRG(1),N,J)
102:      & + SMIC(IBP(I),N,LEMIC(J,2,2))*DNF(I)
103: 160 continue
104: 170 continue
105: C
106: CC      .... NGP2 = 1 (vectorizable)
107: C
108:      else
109:      IE1 = IEG(1)
110:      do 180 I = 1, NN
111:      DNF(I) = DN*FLX(I)
112:      DNFLX(IE1,IRG(1),N) = DNFLX(IE1,IRG(1),N)
113:      & + DNF(I)
114: 180 continue
115: C
116:      do 200 J = 1, NEMICP
117:      do 190 I = 1, NN
118:      RMIC(IE1,IRG(1),N,J) =
119:      & RMIC(IE1,IRG(1),N,J)
120:      & + SMIC(IBP(I),N,LEMIC(J,2,2))*DNF(I)
121: 190 continue
122: 200 continue
123:      end if
124:      end if
125: 210 continue
126: C
127:      end if
128: C
129: C**** TALLY INCLUDING MULTIPLE REGIONS ****
130: C

```

src/mvp/tallyp.f

```

131:      else
132:        do 220 I = 1, NN
133:          FLXX(IEG(I),IRG(I)) = FLXX(IEG(I),IRG(I)) + FLX(I)
134:        220 continue
135:      C
136:      CM      DO 1200 N=1,NRESP
137:      CM      DO 1200 I=1,NN
138:      CM      REXX(IRG(I),N) = REXX(IRG(I),N) + FLX(I)*RESP(IEG(I),N)
139:      C1200 CONTINUE
140:      C
141:      if ( JRESP.ne.0 ) then
142:        do 240 N = 1, NSTAL
143:          do 230 I = 1, NN
144:            RSTAL(IRG(I),N) = RSTAL(IRG(I),N) + SGTAL(IEG(I),N)*
145:              & FLX(I)
146:          230 continue
147:        240 continue
148:      endif
149:      C
150:      if ( JREFAC.ne.0.and.NEMICP.ne.0 ) then
151:        if ( JZONE.eq.1 ) then
152:          do 290 N = 1, NPATOM
153:            do 250 I = 1, NN
154:              DNF(I) = DNZON(N,IZN(I),2)*FLX(I)
155:            250 continue
156:          C
157:          *VOCL LOOP,SCALAR
158:          do 260 I = 1, NN
159:            if ( DNF(I).ne.0.0 ) then
160:              IE = IEG(I)
161:              IR = IRG(I)
162:              DNFLX(IE,IR,N) = DNFLX(IE,IR,N) + DNF(I)
163:            end if
164:          260 continue
165:          do 280 J = 1, NEMICP
166:            do 270 I = 1, NN
167:              if ( DNF(I).ne.0.0 ) then
168:                IE = IEG(I)
169:                IR = IRG(I)
170:                RMIC(IE,IR,N,J) = RMIC(IE,IR,N,J)
171:                & + SMIC(IEG(I),N,LEMIC(J,2,2))*DNF(I)
172:              end if
173:            270 continue
174:          280 continue
175:          290 continue
176:        else
177:          do 330 N = 1, NPATOM
178:            DN = DNZON(N,IZN(1),2)
179:            if ( DN.ne.0.0 ) then
180:              do 300 I = 1, NN
181:                DNF(I) = DN*FLX(I)
182:                IE = IEG(I)
183:                IR = IRG(I)
184:                DNFLX(IE,IR,N) = DNFLX(IE,IR,N) + DNF(I)
185:              300 continue
186:              do 320 J = 1, NEMICP
187:                do 310 I = 1, NN
188:                  IE = IEG(I)
189:                  IR = IRG(I)
190:                  RMIC(IE,IR,N,J) = RMIC(IE,IR,N,J)
191:                  & + SMIC(IEG(I),N,LEMIC(J,2,2))*DNF(I)
192:                310 continue
193:              320 continue
194:            end if
195:          330 continue

```

```

196:      end if
197:    end if
198:  end if
199: C
200: return
201: end

```

src/mvp/talprt.f

```

1:      subroutine TALPRT( TITLE, A,      CHA,      NTHIST,NGROUP,NPKIND,NREG,
2:      &                  NRESP, NBATCH,NUC,      NPATOM,NEMIC, NEMAC,
3:      &                  NTREG, NSTAL, NSKIP, NHIST, NTASK, NEVENT,WSUM,
4:      &                  WLEK,  SFLTR, SFLCL, SRETR, SRECL, ENGYB, RVOL,
5:      &                  TRVOL, WCNTR, SRMIC, RMICR, XMIC,  RMAC,
6:      &                  RMACR, XMAC,  LEMIC, LEMAC, NUCID, SRSTR,
7:      &                  SRSCl, TFLHS, ITRNM, TNAMS, NNAMES,IWK2,
8:      &                  NGP1,SRMICSM,XMICSM,RMACSM,XMACSM,  ! scat-mtx
9:      &                  SMIMU, RMAMU,                      ! scat-mtx
10:     &                  UU )                                ! photo-nuc
11: C
12: C=<MVP=====
13: C PURPOSE: PRINTOUT FLUX, REACTION RATE AND THEIR ERRORS
14: C CALLED IN: CADENZ
15: C=====
16:      implicit real*8(A-H,O-Z)
17: C
18:      real A(*)
19:      character*4 CHA(*)
20: C
21:      include 'INC/_KPID5'
22:      include 'INC/_NGPS'
23:      include 'INC/_KPSYMS'
24: C
25:      include 'INC/_FLAGS'
26:      include '../shared/INC/_IOUNIT'
27:      include 'INC/_IOUNIT2'
28: Ccccc include '../shared/INC/_TASKDT'
29: C
30:      parameter( NL      = 8 )
31:      character*72 TITLE(2)
32: C
33: C/#IF INTEGER8
34: *      integer*8 NTHIST
35: C/#ELSE
36:      integer      NTHIST
37: C/#ENDIF
38: C
39:      real*8 WCNTR(NEVENT,KPLIM)
40:      real*8 WSUM, WLEK
41:      real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
42:      &      SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
43: C
44:      real*8 SRMIC(NGROUP,NTREG,NUC,NEMIC,2),
45:      &      RMICR(NPKIND,NTREG,NUC,NEMIC,2),
46:      &      XMIC(NGROUP+NPKIND,NTREG,NUC,NEMIC,2),
47:      &      RMAC(NGROUP,NTREG,NEMAC,2), RMACR(NPKIND,NTREG,NEMAC,2),
48:      &      XMAC(NGROUP+NPKIND,NTREG,NEMAC,2), SRSTR(NTREG,NSTAL,2),
49:      &      SRSCl(NTREG,NSTAL,2)
50:
51:      real*8 SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,JSCTM,*), ! scat-mtx
52:      &      RMACSM(NGP1+1,NGP1+1,NTREG,2,JSCTM,*)      ! scat-mtx
53:      real*8 SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,*),          ! scat-mtx
54:      &      RMAMU(NGP1+1,NGP1+2,NTREG,3,*)              ! scat-mtx
55:
56:      real*8 TFLHS(KPLIM,2)
57: C
58:      character*16 NUCID(NUC)
59:      integer LEMIC(16), LEMAC(16)
60:      integer IWK2(8)
61: C
62:      real RVOL(NREG)
63:      real ENGYB(*)
64:      real TRVOL(NTREG)
65:

```

```

66:      real UU(NGROUP) ! photo-nuc
67: C
68: C
69:      integer ITRNM(NTREG)
70:      include '../shared/INC/_LNAME'
71:      character*(LNAME) TNAMS(NNAMES)
72: C
73: C***** LOCAL VARIABLES *****
74: C
75:      real*8 TOTVAL(NL), TOTERR(NL) ! photo-nuc
76:      character*32 ESTMTR
77:      integer MREAC(20)
78:      character*100 NAME
79:      character*100 HEADR
80:
81:      real ZERO ! scat-mtx, photo-nuc
82: C
83:      include 'INC/_REACC'
84: C
85: C
86: C-----
87: C
88:      NG1      = NGROUP + 1
89:      NSNEU     = NGROUP
90:      ZERO = 0 ! scat-mtx, photo-nuc
91: C
92: C .... ACCESS INDICES FOR PARTICLE SUM ...
93: C
94: C      NSNEU & NSPHT FOR (..., NGROUP+NPKIND) ARRAYS
95: C      ISNEU & ISPHT FOR (..., NPKIND) ARRAYS
96: C
97:      if ( JNEUT.ne.0 ) NSNEU = NGROUP + 1
98:      if ( JPHOT.ne.0 ) NSPHT = NSNEU + 1
99:      ISNEU = NSNEU - NGROUP
100:      ISPHT = NSPHT - NGROUP
101: C
102: C-----
103: C ..... PRINT TALLY-REGION # & NAMES HERE .....
104: C-----
105: C
106: C
107: C
108: C =====
109: call HEADER( IOT, 'TALLY REGION NO. AND INPUT-NAME' )
110: C =====
111: C
112:      write(IOT,7000)
113: 7000 format(' T-REG. # VOLUME(CC) NAME' /
114:      &      ' ----- ',80('-') /)
115:      do 100 KR = 1, NTREG
116:      if ( ITRNM(KR).lt.0 ) then
117:      write(IOT,'(4X,I8,3X,1PE12.5,2X,A)') KR, TRVOL(KR),
118:      &      TNAMS(ABS(ITRNM(KR)))
119:      else
120:      call REGNM0( ITRNM(KR), '>', NAME, LNM, A, CHA )
121:      write(IOT,'(4X,I8,3X,1PE12.5,2X,A)') KR, TRVOL(KR),
122:      &      NAME(:LNM)
123:      end if
124: 100 continue
125: C
126: C-----
127: C ..... PRINT weighted time of flight .....
128: C-----
129: C
130:      if ( JTIME.ne.0 ) then

```

src/mvp/talprt.f

```

131:
132:     call LABEL( IOT, 'TIME OF FLIGHT PER HISTORY (SEC)' )
133:
134:     do 110 IK = 1, KPLIM
135:         if ( JKPARG(IK).ne.0 ) then
136:             write(IOT,7020) KPSYM(1,IK) (:ICLEN2(KPSYM(1,IK))),
137:             &             TFLHS(IK,1), TFLHS(IK,2)
138:         end if
139:     continue
140: 7020 format(/lx,' * * ',A,' * * '//5X,'AVERAGE: ',1P,E13.5,
141: &         ' (STANDARD DEV. ',1P,E10.3,' % )')
142: end if
143: C
144: C-----
145: C .... PRINT OUT OF FLUX TALLIES (ON I/O UNIT IOF) .....
146: C-----
147: C
148:     if ( JFPRT.ne.0 ) then
149: C
150: C
151: C =====
152:     call HEADER( IOF,
153: &     'FLUX BY TRACK LENGTH ESTIMATOR (/SOURCE/LETHARGY/VOLUME)'
154: &     )
155: C =====
156: C
157:     do 140 IK = 1, KPLIM
158:         if ( JKPARG(IK).ne.0 ) then
159:             if ( NPKIND.gt.1 ) then
160:                 call LABEL( IOF, KPSYM(1,IK) (:ICLEN2(KPSYM(1,IK))) )
161:             end if
162: C
163:             do 130 K = 1, NTREG, NL
164:                 K1 = MIN(NTREG,K+NL-1)
165:                 write(IOF,7040) (' T-REG.',L,L=K,K1)
166:                 format(/lx,' GRP ENERGY ',8(A8,I4,1X)/)
167: c##<2007/03/14:PN3:
168: c7060 format(1X,1X,I3,1X,1P,E10.4,1P,8E13.5)
169: 7060 format(1X,I4,1P,E11.4,8E13.5)
170: c##>
171: 7080 format(1X,16X,8(' (' ,F7.3,' % ) ':))
172: c##<2007/03/14:PN3:
173:         K3 = K1 - K + 1
174:         do K2 = 1, K3
175:             TOTVAL(K2) = 0
176:             TOTERR(K2) = 0
177:         end do
178: c##>
179: C
180: *VOCL LOOP, SCALAR
181:         do 120 IG0 = 1, NGP(IK)
182:             IG = KNGP(IK) + IG0 - 1
183:             IGB = KNGP(IK) + IG0 - 1
184:             write(IOF,7060) IG0, ENGYB(IGB),
185:             &             (SFLTR(IG,L,1),L=K,K1)
186:             write(IOF,7080) (SFLTR(IG,L,2),L=K,K1)
187: c##<2007/03/14:PN3:
188:             do K2 = 1, K3
189:                 S = SFLTR(IG,K+K2-1,1) * UU(IG)
190:                 TOTVAL(K2) = TOTVAL(K2) + S
191:                 TOTERR(K2) = TOTERR(K2) +
192:                 &                 (S * SFLTR(IG,K+K2-1,2)) ** 2
193:             end do
194: c##>
195: 120 continue

```

```

196:             write(IOF,7100) ENGYB(KENGP(IK)+NGP(IK))
197: c##<2007/03/14:PN3:
198: c7100 format(1X,1X,3X,1X,1PE10.4/)
199: 7100 format(5X,1P,E11.4/)
200:         do K2 = 1, K3
201:             if ( TOTVAL(K2).le.ZERO ) then
202:                 TOTERR(K2) = 0
203:             else
204:                 TOTERR(K2) = sqrt(TOTERR(K2)) / TOTVAL(K2)
205:             end if
206:         end do
207:         write(IOF,7320) (TOTVAL(K2),K2=1,K3)
208:         write(IOF,7380) (TOTERR(K2),K2=1,K3)
209:         format(1X,' [ /SRC/VOL ] ',8(' (' ,F7.3,' % ) ':))
210: c##>
211: 130 continue
212:     end if
213: 140 continue
214: C
215: C =====
216:     call HEADER( IOF,
217: &     'FLUX BY COLLISION ESTIMATOR (/SOURCE/LETHARGY/VOLUME)'
218: &     )
219: C =====
220: C
221:     do 170 IK = 1, KPLIM
222:         if ( JKPARG(IK).ne.0 ) then
223:             if ( NPKIND.gt.1 ) then
224:                 call LABEL( IOF, KPSYM(1,IK) (:ICLEN2(KPSYM(1,IK))) )
225:             end if
226: C
227:             do 160 N = 1, NTREG, NL
228:                 N1 = MIN(NTREG,N+NL-1)
229:                 write(IOF,7040) (' T-REG.',L,L=N,N1)
230: c##<2007/03/14:PN3:
231:                 K3 = N1 - N + 1
232:                 do K2 = 1, K3
233:                     TOTVAL(K2) = 0
234:                     TOTERR(K2) = 0
235:                 end do
236: c##>
237: *VOCL LOOP, SCALAR
238:                 do 150 IG0 = 1, NGP(IK)
239:                     IG = KNGP(IK) + IG0 - 1
240:                     IGB = KNGP(IK) + IG0 - 1
241:                     write(IOF,7060) IG0, ENGYB(IGB),
242:                     &                     (SFLCL(IG,L,1),L=N,N1)
243:                     write(IOF,7080) (SFLCL(IG,L,2),L=N,N1)
244: c##<2007/03/14:PN3:
245:                     do K2 = 1, K3
246:                         S = SFLCL(IG,N+K2-1,1) * UU(IG)
247:                         TOTVAL(K2) = TOTVAL(K2) + S
248:                         TOTERR(K2) = TOTERR(K2) +
249:                         &                         (S * SFLCL(IG,N+K2-1,2)) ** 2
250:                     end do
251: c##>
252: 150 continue
253:                 write(IOF,7100) ENGYB(KENGP(IK)+NGP(IK))
254: c##<2007/03/14:PN3:
255:                 do K2 = 1, K3
256:                     if ( TOTVAL(K2).le.0.d0 ) then
257:                         TOTERR(K2) = 0
258:                     else
259:                         TOTERR(K2) = sqrt(TOTERR(K2)) / TOTVAL(K2)
260:                     end if

```

src/mvp/talprt.f

```

261:          end do
262:          write(IOF,7320) (TOTVAL(K2),K2=1,K3)
263:          write(IOF,7380) (TOTERR(K2),K2=1,K3)
264: c##>
265:   160      continue
266:          end if
267:   170      continue
268: C
269:      end if
270: C
271: C-----
272: C ..... PRINT OUT OF REACTION RATES .....
273: C-----
274: C
275:      if ( JRESP.ne.0.and.NRESP.gt.0 ) then
276: C
277: C
278: C =====
279: call HEADER( IOT, 'REACTION RATE BY TRACK LENGTH ESTIMATOR' )
280: C =====
281: C
282:      do 190 N = 1, NRESP, NL
283:          NL = MIN(NRESP,N+NL-1)
284:          write(IOT,7120) ' T-REG. ', ( ' REACT.',L=L,N,N1)
285:   7120      format(/1X,A10,8(A8,I4,1X))
286:          write(IOT,7140) ' ----- ', ( ' ----- ',L=L,N,N1)
287:   7140      format(/1X,A10,8A13/)
288: *VOCL LOOP,SCALAR
289:          do 180 KR = 1, NTREG
290:              write(IOT,7160) KR, (SRSTR(KR,L,1),L=N,N1)
291:              write(IOT,7180) (SRSTR(KR,L,2),L=N,N1)
292:   7160          format(1X,I7,3X,1P,8E13.5)
293:   7180          format(1X,10X,8(' (',F7.3,'%') ':))
294:   180          continue
295:   190          continue
296: C
297: C
298: C =====
299: call HEADER( IOT, 'REACTION RATE BY COLLISION ESTIMATOR' )
300: C =====
301: C
302:      do 210 N = 1, NRESP, NL
303:          NL = MIN(NRESP,N+NL-1)
304:          write(IOT,7120) ' T-REG. ', ( ' REACT.',L=L,N,N1)
305:          write(IOT,7140) ' ----- ', ( ' ----- ',L=L,N,N1)
306: *VOCL LOOP,SCALAR
307:          do 200 KR = 1, NTREG
308:              write(IOT,7160) KR, (SRECL(KR,L,1),L=N,N1)
309:              write(IOT,7180) (SRECL(KR,L,2),L=N,N1)
310:   200          continue
311:   210          continue
312:      end if
313: C
314:      if ( JRESP.ne.0.and.NSTAL.gt.0 ) then
315: C
316: C
317: C =====
318: call HEADER( IOT,
319: & 'REACTION RATE WITH POINT-WISE DATA BY TRACK LENGTH ESTIMATOR'
320: & )
321: C =====
322: C
323:      do 230 N = 1, NSTAL, NL
324:          NL = MIN(NSTAL,N+NL-1)
325:          write(IOT,7120) ' T-REG. ', ( ' REACT.',L=L,N,N1)

```

```

326:          write(IOT,7140) ' ----- ', ( ' ----- ',L=N,N1)
327: *VOCL LOOP,SCALAR
328:          do 220 KR = 1, NTREG
329:              write(IOT,7160) KR, (SRSTR(KR,L,1),L=N,N1)
330:              write(IOT,7180) (SRSTR(KR,L,2),L=N,N1)
331:   220          continue
332:   230          continue
333: C
334: C
335: C =====
336: call HEADER( IOT,
337: & 'REACTION RATE WITH POINT-WISE DATA BY COLLISION ESTIMATOR'
338: & )
339: C =====
340: C
341:          do 250 N = 1, NSTAL, NL
342:              NL = MIN(NSTAL,N+NL-1)
343:              write(IOT,7120) ' T-REG. ', ( ' REACT.',L=L,N,N1)
344:              write(IOT,7140) ' ----- ', ( ' ----- ',L=N,N1)
345: *VOCL LOOP,SCALAR
346:          do 240 KR = 1, NTREG
347:              write(IOT,7160) KR, (SRSTR(KR,L,1),L=N,N1)
348:              write(IOT,7180) (SRSTR(KR,L,2),L=N,N1)
349:   240          continue
350:   250          continue
351:          end if
352: C
353: C
354: C-----
355: C ..... PRINT OUT OF MACROSCOPIC REACTION RATES
356: C-----
357: C
358: C
359: C
360: C ..... ESTMTR : NAME OF ESTIMATOR FOR REACTION RATES.
361: C          LEST : EFFECTIVE LENGTH OF ESTMTR
362: C
363:      if ( JRTTR.ne.0 ) then
364:          ESTMTR = 'TRACK LENGTH ESTIMATOR'
365:      else if ( JRTCL.ne.0 ) then
366:          ESTMTR = 'COLLISION ESTIMATOR'
367:      end if
368:      LEST = ICLEN2(ESTMTR)
369: C
370:      NS = 0
371:      do 260 M = 1, 8
372:          if ( JMACE(M).ge.3 ) then
373:              NS = NS + 1
374:              MREAC(NS) = M
375:              IWK2(NS) = LEMAC(M)
376:          end if
377:   260          continue
378: C
379:      if ( NS.gt.0 ) then
380: C
381: C =====
382:      HEADR = 'MACROSCOPIC REACTION RATES ( '//ESTMTR(:LEST) //' )'
383:      call HEADER( IOT, HEADR )
384: C =====
385: C
386:      write(IOT,
387: & ' ( ' *** TALLY-REGION SUM (VOLUME INTEGRATED) *** ' )'
388: & )
389:      K1 = 0
390:      do 280 K = 1, NS, NL

```

src/mvp/talprt.f

```

391:      K1      = MIN(NS,K+NL-1)
392:      write(IOT,7200) (REAC(MREAC(L)),L=K,K1)
393: 7200      format(/1X,' TREG#  VOLUME(CC)  ',8(A12,1X))
394:      write(IOT,7220) ('-----',L=K,K1)
395: 7220      format(1X,' ----- ',8(A12,1X))
396:      do 270 KR = 1, NTREG
397:          write(IOT,7240) KR, TRVOL(KR),
398:      &          (RMACR(ISNEU,KR,IWK2(L),1),L=K,K1)
399:          write(IOT,7260) (RMACR(ISNEU,KR,IWK2(L),2),L=K,K1)
400: 270      continue
401: 7240      format(1X,I6,1P,E13.5,8E13.5)
402: 7260      format(1X,19X,8(' ',F7.3,'%'))
403: 280      continue
404: C
405: C
406:      write(IOT,7220) ('-----',L=K,K1)
407:      write(IOT,'(''*** TALLI-REGION SUM (VOLUME AVERAGED) ***''))
408:      &
409:      K1      = 0
410:      do 300 K = 1, NS, NL
411:          K1      = MIN(NS,K+NL-1)
412:          write(IOT,7200) (REAC(MREAC(L)),L=K,K1)
413:          write(IOT,7220) ('-----',L=K,K1)
414:          do 290 KR = 1, NTREG
415:              if ( TRVOL(KR).ne.0 ) then
416:                  write(IOT,7240) KR, TRVOL(KR),
417:      &                  (RMACR(ISNEU,KR,IWK2(L),1)/TRVOL(KR),L=K,K1)
418:                  write(IOT,7260) (RMACR(ISNEU,KR,IWK2(L),2),L=K,K1)
419:              end if
420:          continue
421: 300      continue
422:      write(IOT,7220) ('-----',L=K,K1)
423: C
424: C
425: C
426:      do 350 KR = 1, NTREG
427:          write(IOT,'(/1X,'' === T-REG.  '' ,I4,A)'') KR,
428:      &          '=== REACTION RATE ==='
429:          do 340 K = 1, NS, NL
430:              K1      = MIN(NS,K+NL-1)
431:              write(IOT,7280) (REAC(MREAC(L)),L=K,K1)
432: 7280      format(/1X,'          ENERGY  ',8(A12,1X))
433:              write(IOT,7300) ('-----',L=K,K1)
434: 7300      format(1X,' --- ----- ',8(A12,1X))
435: C
436:          ITF      = 0
437:          do 310 L = K, K1
438:              if ( RMACR(ISNEU,KR,IWK2(L),1).ne.0.0 ) then
439:                  ITF      = 1
440:                  go to 320
441:              end if
442: 310      continue
443: 320      continue
444: C
445:          if ( ITF.ne.0 ) then
446: *VOCL LOOP,SCALAR
447:              do 330 IG = 1, NGP(KPNEUT)
448:                  write(IOT,7060) IG, ENGYB(IG),
449:      &                  (XMAC(IG,KR,IWK2(L),1),L=K,K1)
450:                  write(IOT,7080) (XMAC(IG,KR,IWK2(L),2),L=K,K1)
451: 330      continue
452:          end if
453: C
454:          write(IOT,7300) ('-----',L=K,K1)
455:          write(IOT,7320) (RMACR(ISNEU,KR,IWK2(L),1),L=K,K1)

```

```

456:          write(IOT,7080) (RMACR(ISNEU,KR,IWK2(L),2),L=K,K1)
457: 7320      format(1X,'          TOTAL  ',1P,8E13.5)
458: 340      continue
459: 350      continue
460:      end if
461: C
462: C-----
463: C      PRINT OUT OF MACROSCOPIC CROSS SECTIONS
464: C-----
465: C
466:      NS      = 0
467:      do 360 M = 1, 8
468:          if ( JMACE(M).eq.2 .or. JMACE(M).eq.4 ) then
469:              NS      = NS + 1
470:              MREAC(NS) = M
471:              IWK2(NS)  = LEMAC(M)
472:          end if
473: 360      continue
474: C
475:      if ( NS.gt.0 ) then
476: C
477: C
478: C      =====
479:      HEADR  = 'MACROSCOPIC CROSS SECTIONS ('//ESTMTR(:LEST) //')'
480:      call HEADER( IOT, HEADR )
481: C
482: C      =====
483: C
484:      do 410 KR = 1, NTREG
485:          write(IOT,'(/1X,'' === T-REG.  '' ,I4,A24)'') KR,
486:      &          '=== CROSS SECTION === '
487:          do 400 K = 1, NS, NL
488:              K1      = MIN(NS,K+NL-1)
489:              write(IOT,7280) (REAC(MREAC(L)),L=K,K1)
490:              write(IOT,7300) ('-----',L=K,K1)
491: C
492:              ITF      = 0
493:              do 370 L = K, K1
494:                  if ( RMACR(ISNEU,KR,IWK2(L),1).ne.0.0 ) then
495:                      ITF      = 1
496:                      go to 380
497:                  end if
498: 370      continue
499: 380      continue
500: C
501:          if ( ITF.ne.0 ) then
502: *VOCL LOOP,SCALAR
503:              do 390 IG = 1, NGROUP
504:                  write(IOT,7060) IG, ENGYB(IG),
505:      &                  (XMAC(IG,KR,IWK2(L),1),L=K,K1)
506:                  write(IOT,7080) (XMAC(IG,KR,IWK2(L),2),L=K,K1)
507: 390      continue
508:                  write(IOT,7300) ('-----',L=K,K1)
509:                  write(IOT,7340) (XMAC(NG1,KR,IWK2(L),1),L=K,K1)
510:                  write(IOT,7080) (XMAC(NG1,KR,IWK2(L),2),L=K,K1)
511: 7340      format(1X,'          AVERAGE  ',1P,8E13.5)
512:              else
513:                  write(IOT,7360)
514: 7360      format(1X,10X,' === ALL CROSS SECTIONS ARE ZERO === '//
515:      &                  )
516:              end if
517:          write(IOT,7300) ('-----',L=K,K1)
518:      400      continue
519:      410      continue
520:      end if

```


src/mvp/talprt.f

```

521: C
522: C-----
523: C ..... PRINT OUT OF MICROSCOPIC REACTION RATES
524: C-----
525: C
526:       NS       = 0
527:       do 420 M = 1, 8
528:         if ( JMICE(M).ge.3 ) then
529:           NS     = NS + 1
530:           MREAC(NS) = M
531:           IWK2(NS) = LEMIC(M)
532:         end if
533: 420 continue
534: C
535:       if ( NS.gt.0 ) then
536: C
537: C=====
538:       HEADR = 'MICROSCOPIC REACTION RATES ( '//ESTMTR(:LEST) //' )
539:       call HEADER( IOT, HEADR )
540: C=====
541: C
542: C
543:       do 480 NN = 1, NUC
544:         write(IOT,'(/1X, ''===== NUCLIDE <''',A, ''> ====='')' )
545:         &      NUCID(NN) (:ICLEN(NUCID(NN)))
546:         do 470 KR = 1, NTREG
547:           ITF0 = 0
548:           do 460 K = 1, NS, NL
549:             K1 = MIN(NS,K+NL-1)
550: C
551:             ITF = 0
552:             do 430 L = K, K1
553:               if ( RMICR(ISNEU,KR,NN,IWK2(L),1).ne.0.0 ) then
554:                 ITF = 1
555:                 go to 440
556:               end if
557: 430 continue
558: 440 continue
559: C
560:             if ( ITF.ne.0 ) then
561:               if ( ITF0.eq.0 ) then
562:                 write(IOT,'(/1X, '' <''',A, ''> === T-REG. ''',
563: &      I4,A24)' ) NUCID(NN) (:ICLEN(NUCID(NN))), KR,
564: &      '=== REACTION RATE === '
565:                 ITF0 = 1
566:               end if
567:               write(IOT,7280) (REAC(MREAC(L)),L=K,K1)
568:               write(IOT,7300) ('-----',L=K,K1)
569: *VOCL LOOP, SCALAR
570:               do 450 IG = 1, NGROUP
571:                 write(IOT,7060) IG, ENGYB(IG),
572: &      (SRMIC(IG,KR,NN,IWK2(L),1),L=K,K1)
573:                 write(IOT,7080)
574: &      (SRMIC(IG,KR,NN,IWK2(L),2),L=K,K1)
575: 450 continue
576:               write(IOT,7300) ('-----',L=K,K1)
577:               write(IOT,7320)
578: &      (RMICR(ISNEU,KR,NN,IWK2(L),1),L=K,K1)
579:               write(IOT,7080)
580: &      (RMICR(ISNEU,KR,NN,IWK2(L),2),L=K,K1)
581:             end if
582: 460 continue
583: 470 continue
584: 480 continue
585:         end if

```

```

586: C
587: C-----
588: C ..... PRINT OUT OF MICROSCOPIC CROSS SECTIONS
589: C-----
590: C
591:       NS       = 0
592:       do 490 M = 1, 8
593:         if ( JMICE(M).eq.2 .or. JMICE(M).eq.4 ) then
594:           NS     = NS + 1
595:           MREAC(NS) = M
596:           IWK2(NS) = LEMIC(M)
597:         end if
598: 490 continue
599: C
600:       if ( NS.gt.0 ) then
601: C
602: C=====
603:       if ( JNWMX.eq.0 ) then
604:         HEADR = 'MICROSCOPIC CROSS SECTIONS ( '//
605: &      ESTMTR(:LEST) //' )
606:       else
607:         HEADR = 'NUCLIDE-WISE MACROSCOPIC CROSS SECTIONS ( '//
608: &      ESTMTR(:LEST) //' )
609:       end if
610:       call HEADER( IOT, HEADR )
611: C=====
612: C
613: C
614:       do 550 NN = 1, NUC
615:         write(IOT,'(/1X, ''===== NUCLIDE <''',A, ''> ====='')' )
616:         &      NUCID(NN) (:ICLEN(NUCID(NN)))
617:         do 540 KR = 1, NTREG
618:           write(IOT,'(/1X, '' <''',A, ''> === T-REG. ''',I4,A24)' )
619: &      NUCID(NN) (:ICLEN(NUCID(NN))), KR,
620: &      '=== CROSS SECTION === '
621:         do 530 K = 1, NS, NL
622:           K1 = MIN(NS,K+NL-1)
623:           write(IOT,7280) (REAC(MREAC(L)),L=K,K1)
624:           write(IOT,7300) ('-----',L=K,K1)
625: C
626:           ITF = 0
627:           do 500 L = K, K1
628:             if ( RMICR(ISNEU,KR,NN,IWK2(L),1).ne.0.0 ) then
629:               ITF = 1
630:               go to 510
631:             end if
632: 500 continue
633: 510 continue
634: C
635:           if ( ITF.ne.0 ) then
636: *VOCL LOOP, SCALAR
637:             do 520 IG = 1, NGROUP
638:               write(IOT,7060) IG, ENGYB(IG),
639: &      (XMIC(IG,KR,NN,IWK2(L),1),L=K,K1)
640:               write(IOT,7080)
641: &      (XMIC(IG,KR,NN,IWK2(L),2),L=K,K1)
642: 520 continue
643:             write(IOT,7300) ('-----',L=K,K1)
644:             write(IOT,7340) (XMIC(NG1,KR,NN,IWK2(L),1),L=K,K1)
645:             write(IOT,7080) (XMIC(NG1,KR,NN,IWK2(L),2),L=K,K1)
646:           else
647:             write(IOT,7360)
648:           end if
649:           write(IOT,7300) ('-----',L=K,K1)
650: 530 continue

```

src/mvp/talprt.f

```

651: 540 continue
652: 550 continue
653: end if
654: C
655: C-----
656: C PRINT OUT OF SCATTERING MATRIX TALLY
657: C-----
658: C
659: if ( JSCTM.ne.0 ) then
660: C
661: IG0 = NL
662: IGB = KENGP(KPNEUT) - 1
663: NGP1 = NGP(KPNEUT)
664: C
665: C***** macroscopic reaction rates of scattering matrix
666: C
667: NS = 0
668: if ( JMACE(4).ge.3 ) then
669: NS = NS + 1
670: MREAC(NS) = 4
671: end if
672: if ( JMACE(6).ge.3 ) then
673: NS = NS + 1
674: MREAC(NS) = 6
675: end if
676: if ( JMACE(7).ge.3 ) then
677: NS = NS + 1
678: MREAC(NS) = 7
679: end if
680: if ( NS.le.0 ) go to 702
681: C
682: =====
683: HEADR = 'MACROSCOPIC GROUP-TO-GROUP SCATTERING RATES ('
684: & //ESTMTR(:LEST) //'')
685: call HEADER( IOT, HEADR )
686: C
687: =====
688: do KR = 1, NTREG
689: write(IOT,6101) KR, '===== REACTION RATE ====='
690: do K = 1, NS
691: write(IOT,6102) KR, REAC(MREAC(K))
692: continue
693: do IM = 1, JSCTM
694: write(IOT,6114) KR, REAC(MREAC(K)), IM-1
695: do IG2 = 1, NGP1
696: do IG1 = 1, NGP1
697: if ( RMACSM(IG1,IG2,KR,1,IM,K).ne.ZERO )
698: go to 700
699: end do
700: end do
701: write(IOT,6113)
702: go to 701
703: continue
704: do IG2 = 1, NGP1, IG0
705: IG3 = min(IG2+IG0-1,NGP1)
706: write(IOT,6103) (IG,IG=IG2,IG3)
707: write(IOT,6104) (ENGYB(IGB+IG),IG=IG2,IG3)
708: do IG1 = 1, NGP1
709: write(IOT,6105) IG1, ENGYB(IGB+IG1),
710: & (RMACSM(IG1,IG,KR,1,IM,K),IG=IG2,IG3)
711: & write(IOT,6106)
712: & (RMACSM(IG1,IG,KR,2,IM,K),IG=IG2,IG3)
713: end do
714: write(IOT,6107)
715: end do
716: continue

```

```

716: end do
717: end do
718: end do
719: 702 continue
720: C
721: C***** macroscopic probability of scattering matrix
722: C
723: NS = 0
724: c% if ( JMACE(4).eq.2.or.JMACE(4).eq.4 ) then
725: c% NS = NS + 1
726: c% MREAC(NS) = 4
727: c% end if
728: c% if ( JMACE(6).eq.2.or.JMACE(6).eq.4 ) then
729: c% NS = NS + 1
730: c% MREAC(NS) = 6
731: c% end if
732: c% if ( JMACE(7).eq.2.or.JMACE(7).eq.4 ) then
733: c% NS = NS + 1
734: c% MREAC(NS) = 7
735: c% end if
736: c% if ( NS.le.0 ) go to 705
737: C
738: c% =====
739: c% & HEADR = 'MACROSCOPIC GROUP-TO-GROUP SCATTERING PROBABILITIES ('
740: c% //ESTMTR(:LEST) //'')
741: c% call HEADER( IOT, HEADR )
742: C
743: c% =====
744: c% do KR = 1, NTREG
745: c% write(IOT,6101) KR, '===== PROBABILITY ====='
746: c% do K = 1, NS
747: c% write(IOT,6102) KR, REAC(MREAC(K))
748: c% do IG2 = 1, NGP1
749: c% do IG1 = 1, NGP1
750: c% if ( RMACSM(IG1,IG2,KR,1,K).gt.ZERO ) go to 703
751: c% end do
752: c% end do
753: c% write(IOT,6113)
754: c% go to 704
755: c% 703 continue
756: c% do IG2 = 1, NGP1, IG0
757: c% IG3 = min(IG2+IG0-1,NGP1)
758: c% write(IOT,6103) (IG,IG=IG2,IG3)
759: c% write(IOT,6104) (ENGYB(IGB+IG),IG=IG2,IG3)
760: c% do IG1 = 1, NGP1
761: c% RT = 0.d0
762: c% do IG4 = 1, NGP1
763: c% RT = RT + RMACSM(IG1,IG4,KR,1,K)
764: c% end do
765: c% if ( RT.gt.0.d0 ) then
766: c% write(IOT,6105) IG1, ENGYB(IGB+IG1),
767: c% & (RMACSM(IG1,IG,KR,1,K)/RT,IG=IG2,IG3)
768: c% & write(IOT,6106)
769: c% & (RMACSM(IG1,IG,KR,2,K),IG=IG2,IG3)
770: c% else
771: c% write(IOT,6105) IG1, ENGYB(IGB+IG1),
772: c% & (0.d0,IG=IG2,IG3)
773: c% & write(IOT,6106) (0.d0,IG=IG2,IG3)
774: c% end if
775: c% end do
776: c% write(IOT,6107)
777: c% end do
778: c% 704 continue
779: c% end do
780: c% 705 continue

```

src/mvp/talprt.f

```

781: C
782: C***** microscopic reaction rates of scattering matrix
783: C
784:       NS = 0
785:       if ( JMICE(4).ge.3 ) then
786:         NS = NS + 1
787:         MREAC(NS) = 4
788:       end if
789:       if ( JMICE(6).ge.3 ) then
790:         NS = NS + 1
791:         MREAC(NS) = 6
792:       end if
793:       if ( JMICE(7).ge.3 ) then
794:         NS = NS + 1
795:         MREAC(NS) = 7
796:       end if
797:       if ( NS.le.0 ) go to 708
798: C
799:       HEADR = 'MICROSCOPIC GROUP-TO-GROUP SCATTERING RATES ('
800:       & //ESTMTR(:LEST) //'')
801:       call HEADER( IOT, HEADR )
802: C
803:       do N = 1, NUC
804:         write(IOT,6109) N, NUCID(N)(:ICLEN(NUCID(N)))
805:         do KR = 1, NTREG
806:           do K = 1, NS
807:             do IM = 1, JSCTM
808:               do IG2 = 1, NGP1
809:                 do IG1 = 1, NGP1
810:                   if ( SRMCSM(IG1,IG2,KR,N,1,IM,K).ne.ZERO )
811:                     & go to 716
812:                   end do
813:                 end do
814:               end do
815:             end do
816:           go to 717
817:         716 continue
818:         write(IOT,6110) NUCID(N)(:ICLEN(NUCID(N))), KR,
819:         & '===== REACTION RATE ====='
820:         do K = 1, NS
821:           write(IOT,6111) NUCID(N)(:ICLEN(NUCID(N))), KR,
822:           & REAC(MREAC(K))
823:         c%706 continue
824:         c%707 continue
825:         do IM = 1, JSCTM
826:           write(IOT,6115) NUCID(N)(:ICLEN(NUCID(N))), KR,
827:           & REAC(MREAC(K)), IM-1
828:         do IG2 = 1, NGP1
829:           do IG1 = 1, NGP1
830:             if ( SRMCSM(IG1,IG2,KR,N,1,IM,K).ne.ZERO )
831:               & go to 706
832:             end do
833:           end do
834:         write(IOT,6113)
835:         go to 707
836:       706 continue
837:       do IG2 = 1, NGP1, IG0
838:         IG3 = min(IG2+IG0-1,NGP1)
839:         write(IOT,6103) (IG,IG=IG2,IG3)
840:         write(IOT,6104) (ENGYB(IGB+IG),IG=IG2,IG3)
841:         do IG1 = 1, NGP1
842:           write(IOT,6105) IG1, ENGYB(IGB+IG1),
843:           & (SRMCSM(IG1,IG,KR,N,1,IM,K),IG=IG2,IG3)
844:         write(IOT,6106)
845:         & (SRMCSM(IG1,IG,KR,N,2,IM,K),IG=IG2,IG3)

```

```

846:       end do
847:       write(IOT,6107)
848:     end do
849:   707 continue
850: end do
851: end do
852: 717 continue
853: end do
854: end do
855: 708 continue
856: C
857: C***** microscopic probability of scattering matrix
858: C
859: c% NS = 0
860: c% if ( JMICE(4).eq.2.or.JMICE(4).eq.4 ) then
861: c% NS = NS + 1
862: c% MREAC(NS) = 4
863: c% end if
864: c% if ( JMICE(6).eq.2.or.JMICE(6).eq.4 ) then
865: c% NS = NS + 1
866: c% MREAC(NS) = 6
867: c% end if
868: c% if ( JMICE(7).eq.2.or.JMICE(7).eq.4 ) then
869: c% NS = NS + 1
870: c% MREAC(NS) = 7
871: c% end if
872: c% if ( NS.le.0 ) go to 721
873: C
874: c% HEADR = 'MICROSCOPIC GROUP-TO-GROUP SCATTERING PROBABILITIES ('
875: c% & //ESTMTR(:LEST) //'')
876: c% call HEADER( IOT, HEADR )
877: C
878: c% =====
879: c% do N = 1, NUC
880: c% write(IOT,6109) N, NUCID(N)(:ICLEN(NUCID(N)))
881: c% do KR = 1, NTREG
882: c% write(IOT,6110) NUCID(N)(:ICLEN(NUCID(N))), KR,
883: c% & '===== PROBABILITY ====='
884: c% do K = 1, NS
885: c% write(IOT,6112) NUCID(N)(:ICLEN(NUCID(N))), KR,
886: c% & REAC(MREAC(K))
887: c% do IG2 = 1, NGP1
888: c% do IG1 = 1, NGP1
889: c% if ( SRMCSM(IG1,IG2,KR,N,1,K).gt.ZERO )
890: c% & go to 709
891: c% end do
892: c% end do
893: c% write(IOT,6113)
894: c% go to 720
895: c% continue
896: c% do IG2 = 1, NGP1, IG0
897: c% IG3 = min(IG2+IG0-1,NGP1)
898: c% write(IOT,6103) (IG,IG=IG2,IG3)
899: c% write(IOT,6104) (ENGYB(IGB+IG),IG=IG2,IG3)
900: c% do IG1 = 1, NGP1
901: c% RT = 0.d0
902: c% do IG4 = 1, NGP1
903: c% RT = RT + SRMCSM(IG1,IG4,KR,N,1,K)
904: c% end do
905: c% if ( RT.gt.0.d0 ) then
906: c% write(IOT,6105) IG1, ENGYB(IGB+IG1),
907: c% & (SRMCSM(IG1,IG,KR,N,1,K)/RT,IG=IG2,IG3)
908: c% write(IOT,6106)
909: c% & (SRMCSM(IG1,IG,KR,N,2,K),IG=IG2,IG3)
910: c% else

```

src/mvp/talprt.f

```

911: c%                write(IOT,6105) IG1, ENGYB(IGB+IG1),
912: c%      &          (0.d0,IG=IG2,IG3)
913: c%                write(IOT,6106) (0.d0,IG=IG2,IG3)
914: c%                end if
915: c%                end do
916: c%                write(IOT,6107)
917: c%                end do
918: c%720            continue
919: c%                end do
920: c%                end do
921: c%                end do
922: c%721            continue
923:
924:      end if
925:
926: 6101 format(''      ===== T-REG.  'I4,2X,A)
927: 6102 format(''      T-REG.  'I4,'  === REACTION RATE:  'A12,'  ===')
928: 6103 format(''      GRP      ENERGY,8('  OUT-GRP',I4:))
929: 6104 format(''      ---      ---,1P,8E13.5)
930: 6105 format(''      1X,I4,1P,E11,4,8E13.5)
931: 6106 format(''      16X,8('      ('F7.3,'%')':))
932: 6107 format(''      1H )
933: 6108 format(''      T-REG.  'I4,'  === PROBABILITY:  'A12,'  ===')
934: 6109 format(''      ===== NUCLIDE 'I2,'  <'A,'>  =====')
935: 6110 format(''      <'A,'>  ===== T-REG.  'I4,2X,A)
936: 6111 format(''      <'A,'>  === T-REG.  'I4,'  === REACTION RATE:  'A12,'
937:      & '  ===')
938: 6112 format(''      <'A,'>  === T-REG.  'I4,'  === PROBABILITY:  'A12,'
939:      & '  ===')
940: 6113 format(''      16X,'..... ALL DATA ARE ZERO .....')
941: 6114 format(''      T-REG.  'I4,'  === REACTION RATE:  'A12,'  ===')
942:      & ' PL-MOMENT FL:  'I2,'  ===')
943: 6115 format(''      <'A,'>  === T-REG.  'I4,'  === REACTION RATE:  'A12,'
944:      & '  === PL-MOMENT FL:  'I2,'  ===')
945: C
946: C-----
947: C      PRINT OUT OF SCATTERING MUBAR TALLY
948: C-----
949: C
950:      if ( JSCMU.ne.0 ) then
951: C
952:          IG0 = NL
953:          IGB = KENGP(KPNEUT) - 1
954:          NGP1 = NGP(KPNEUT)
955: C
956: C***** macroscopic scattering mubar
957: C
958:          NS = 0
959:          if ( JMACE(4).ge.3 ) then
960:              NS = NS + 1
961:              MREAC(NS) = 4
962:          end if
963:          if ( JMACE(6).ge.3 ) then
964:              NS = NS + 1
965:              MREAC(NS) = 6
966:          end if
967:          if ( JMACE(7).ge.3 ) then
968:              NS = NS + 1
969:              MREAC(NS) = 7
970:          end if
971:          if ( NS.le.0 ) go to 712
972: C      =====
973:          HEADR = 'MACRO SCAT. MUBAR (COLLISION ESTIMATOR)'
974:          call HEADER( IOT, HEADR )
975: C      =====

```

```

976:          IG2 = NGP1 + 1
977:          do K = 1, NS
978:              write(IOT,6121) REAC(MREAC(K))
979:              do KR = 1, NTREG
980:                  do IG1 = 1, NGP1
981:                      if ( RMAMU(IG1,IG2,KR,1,K).gt.ZERO ) go to 710
982:                  end do
983:              end do
984:              write(IOT,6113)
985:              go to 711
986:          continue
987:          do KR1 = 1, NTREG, IG0
988:              KR2 = min(KR1+IG0-1,NTREG)
989:              write(IOT,6122) (KR,KR=KR1,KR2)
990:              write(IOT,6123) ('-----',KR=KR1,KR2)
991:              do IG1 = 1, NGP1
992:                  write(IOT,6105) IG1, ENGYB(IGB+IG1),
993:                      &          (RMAMU(IG1,IG2,KR,1,K),KR=KR1,KR2)
994:                  &          write(IOT,6106) (RMAMU(IG1,IG2,KR,2,K),KR=KR1,KR2)
995:              end do
996:              write(IOT,6107)
997:          end do
998:          continue
999:      end do
1000: 712      continue
1001: C
1002: C***** microscopic scattering mubar
1003: C
1004:          NS = 0
1005:          if ( JMICE(4).ge.3 ) then
1006:              NS = NS + 1
1007:              MREAC(NS) = 4
1008:          end if
1009:          if ( JMICE(6).ge.3 ) then
1010:              NS = NS + 1
1011:              MREAC(NS) = 6
1012:          end if
1013:          if ( JMICE(7).ge.3 ) then
1014:              NS = NS + 1
1015:              MREAC(NS) = 7
1016:          end if
1017:          if ( NS.le.0 ) go to 715
1018: C      =====
1019:          HEADR = 'MICRO SCAT. MUBAR (COLLISION ESTIMATOR)'
1020:          call HEADER( IOT, HEADR )
1021: C      =====
1022:          do N = 1, NUC
1023:              write(IOT,6109) N, NUCID(N)(:ICLEN(NUCID(N)))
1024:              do K = 1, NS
1025:                  write(IOT,6124) NUCID(N)(:ICLEN(NUCID(N))),REAC(MREAC(K))
1026:                  do KR = 1, NTREG
1027:                      do IG1 = 1, NGP1
1028:                          if ( SMIMU(IG1,IG2,KR,N,1,K).gt.ZERO ) go to 713
1029:                      end do
1030:                  end do
1031:                  write(IOT,6113)
1032:                  go to 714
1033:              continue
1034:          do KR1 = 1, NTREG, IG0
1035:              KR2 = min(KR1+IG0-1,NTREG)
1036:              write(IOT,6122) (KR,KR=KR1,KR2)
1037:              write(IOT,6123) ('-----',KR=KR1,KR2)
1038:              do IG1 = 1, NGP1
1039:                  write(IOT,6105) IG1, ENGYB(IGB+IG1),
1040:                      &          (SMIMU(IG1,IG2,KR,N,1,K),KR=KR1,KR2)

```

src/mvp/talprt.f

```
1041:             write(IOT,6106)(SMIMU(IG1,IG2,KR,N,2,K),KR=KR1,KR2)
1042:         end do
1043:         write(IOT,6107)
1044:     end do
1045: 714         continue
1046:     end do
1047: end do
1048: 715         continue
1049: end if
1050: 6121 format('//      ===== REACTION TYPE:  ',A,
1051: &'      ===== MUBAR (-1 < mu < 1) =====')
1052: 6122 format('//      GRP      ENERGY',8('  T-REG.',I5:))
1053: 6123 format('  ---      -----',8('  ',A11:))
1054: 6124 format('//      <',A,'> ===== REACTION TYPE:  ',A,
1055: &'      ===== MUBAR (-1 < mu < 1) =====')
1056: C
1057:     return
1058: end
```

src/mvp/talsme.f

```

1:      subroutine TALSME( IOW, JSCTM, FLX, NN, IEG, IRG, IBP, NUC, NMT,
2:      &                    NGROUP, NZONE, NREG, NBANK, ETOP, EBOT, ETHMAX,
3:      &                    NGP1, ENGYB,
4:      &                    JMICE, JMACE, DNZON, MZONE, KSPI, EEE, AAA,
5:      &                    BBB, CCC, CX, ICX, MCX, SMIC, LMIC, NSMIC,
6:      &                    KLIB1, KLIB2, XLIB1, XLIB2,
7:      &                    DNFLXSM, RMICSM,
8:      &                    IWK1, IWK2, IWK3, IWK4, IWK5, IWK6, IWK7, IWK8,
9:      &                    IWK9, IWK10, IWK11, IWK12, IWK13,
10:     &                    R, WK1, WK2, WK3, WK4, WK5, WK6, WK7, WK8, WK9,
11:     &                    WK10, DWK1, DWK2, WKF1 )
12: C=<MVP>=====
13: C purpose: take tallies for elastic scattering matrix (neutron).
14: C called in: FLIONE
15: C=====
16:      implicit real*8 (A-H,O-Z)
17:      parameter( PI = 3.141592653589793200D+0 )
18:      parameter( PI2 = 2*PI, PIH = PI/2 )
19: C
20: C ..... flag options
21:      integer JMICE(*), JMACE(*)
22: C
23: C ..... cross section data
24:      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4), EBOT, ETOP, ETHMAX
25:      real DNZON(NUC,NZONE,2)
26:      real SMIC(NBANK,NUC,NSMIC)
27:      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21), LMIC(8),
28:      &                    KSPI(NBANK,NUC)
29: C
30: C ..... particle bank
31:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
32:      real EEE(NBANK)
33: C
34:      real*8 FLX(NN)
35:      integer IBP(NN), IEG(NN), IRG(NN)
36: C
37: C ..... tally array
38:      real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*),
39:      &                    RMICSM(NGP1,NGP1,NREG,NUC,JSCTM,*),
40:      real ENGYB(*)
41: C
42: C ..... work area
43:      real*8 DWK1(NBANK), DWK2(NBANK)
44:      real WK1(NBANK), WK2(NBANK), WK3(NBANK), WK4(NBANK), WK5(NBANK),
45:      &                    WK6(NBANK), WK7(NBANK), WK8(NBANK), WK9(NBANK), WK10(NBANK)
46:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
47:      &                    IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
48:      &                    IWK9(NBANK), IWK10(NBANK), IWK11(NBANK), IWK12(NBANK),
49:      &                    IWK13(NBANK)
50:      real R(3*NBANK), ZERO
51:      real WKF1(NBANK) ! as CTH: scattering angle cosine in Lab system
52: C
53: C      zone number treats as single zone by MZONE.
54: C
55: C-----
56: C
57:      ZERO = 0
58:      DZERO = 0
59: C
60:      if ( JSCTM.ne.0 ) then
61: C
62:          do N = 1, NUC
63:              DN = DNZON(N,MZONE,1)
64:              if ( DN.le.DZERO ) go to 400
65: C
66:          KY = 0 ! location of reaction types for matrix tally (1:3
67: C
68: C=====
69: C===== elastic and thermal scattering
70: C=====
71: C
72:          if ( JMICE(4).ne.0.or.JMACE(4).ne.0.or.
73:              &                    JMICE(6).ne.0.or.JMACE(6).ne.0 ) then
74:              if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) KY = KY + 1
75:              NCOLN = 0 ! number of non-thermal reaction
76:              NCOLF = 0 ! number of free gas
77:              NCOLT = 0 ! number of thermal reaction, S(alpha
, beta)
78:              *VOCL LOOP,NOVREC
79:              do I = 1, NN
80:                  IP = IBP(I)
81:                  if ( EEE(IP).gt.ETHMAX ) then
82:                      NCOLN = NCOLN + 1
83:                      WK1(NCOLN) = DN*FLX(I) ! neutron flux weighted by num
ber density
84:                      IWK1(NCOLN) = IP
85:                      IWK11(NCOLN) = I
86:                      else if ( KLIB1(N,15).eq.0.or.
87:                          &                    ( EEE(IP).gt.XLIB1(N,3).and.
88:                          &                    EEE(IP).gt.XLIB1(N,4) ) ) then
89:                          NCOLF = NCOLF + 1
90:                          WK2(NCOLF) = DN*FLX(I) ! neutron flux weighted by num
ber density
91:                          IWK2(NCOLF) = IP
92:                          IWK12(NCOLF) = I
93:                          else
94:                              NCOLT = NCOLT + 1
95:                              WK3(NCOLT) = DN*FLX(I) ! neutron flux weighted by num
ber density
96:                              IWK3(NCOLT) = IP
97:                              IWK13(NCOLT) = I
98:                              end if
99:                              end do
100:                              if ( KY.eq.0.and.NCOLT.le.0 ) go to 190 ! no contributi
on from incoherent inelastic
101:                              if ( NCOLN+NCOLF+NCOLT.le.0 ) go to 190 ! no contributi
on
102:                              else
103:                                  go to 190
104:                              end if
105: C
106: C-----
107: C      mutigroup tally
108: C-----
109: C
110:          if ( NGP1.gt.1 ) then
111:              ATW = XLIB1(N,1)
112: C
113: C***** elastic scattering
114: C
115:          if ( NCOLN.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0))then
116:              MTI = 2
117:              LANG = KLIB2(N,MTI,12)
118:              NEANG = KLIB2(N,MTI,7)
119:              do I = 1, NCOLN
120:                  WK4(I) = EEE(IWK1(I))
121:                  IWK5(I) = NEANG
122:                  IWK6(I) = LANG
123:              end do

```

src/mvp/talsme.f

```

124:      call BSDEC3( CX, IWK5, IWK7, IWK6, WK4, IWK4, NCOLN )
125:      call RANU2( IRAND, R, NCOLN*3, ICON )
126:      N1 = NCOLN
127:      N2 = NCOLN*2
128:      NBINA = KLIB1(N,7)
129:      NBINA1= KLIB1(N,20)
130: *VOCL LOOP,NOVREC
131:      do J = 1, NCOLN
132:         IP = IWK1(J)
133:         if ( IWK4(J).eq.0 ) IWK4(J) = 1
134:         L2 = LANG + IWK4(J)
135:         L1 = L2 - 1
136:         INTE = mod(ICX(L2+NEANG),10)
137:         if ( INTE.eq.5.or.INTE.eq.3 ) then
138:            X1 =log(CX(L1)/EEE(IP))/log(CX(L1)/CX(L2))
139:         else
140:            X1 = (CX(L1)-EEE(IP))/(CX(L1)-CX(L2))
141:         end if
142:         if ( X1.lt.DZERO ) then
143:            X1 = 0
144:         else if ( X1.gt.1.0D+0 ) then
145:            X1 = 1
146:         end if
147:         IT = min(1,int(R(J)+X1)) + IWK4(J)
148:         NA = int(NBINA*R(N1+J)) + 1
149:         L3 = min(NA,NBINA) + LANG + NEANG*2 + NBINA1*(IT-1)
150:         XMU = (CX(L3-1)-CX(L3))*R(N2+J) + CX(L3) ! scattering
cosine (mu)
151:         GE = 1 - XLIB2(N,MTI,2) / EEE(IP)
152:         if ( GE.lt.DZERO ) GE = 0
153:         GE = ATW*sqrt(GE)
154:         GM = GE*XMU
155:         X2 = 1 + 2*GM + GE*GE
156:         WKF1(IWK11(J)) = (1+GM)/sqrt(X2) ! mu in Lab s
system
157:         WK4(J) = EEE(IP)*XLIB1(N,7)*X2 ! outgoing en
ergy
158:         WK4(J) = min(WK4(J),ETOP)
159:      end do
160:      call BSVDEC( ENGYB, NGP1+1, WK4, IWK5, NCOLN ) ! outgoing en
ergy group; IWK5
161: *VOCL LOOP,SCALAR
162:      do J = 1, NCOLN
163:         I = IWK11(J)
164:         IE = IEG(I)
165:         JE = IWK5(J)
166:         IR = IRG(I)
167:         DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
168:            & WK1(J)
169:         RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1) +
170:            & SMIC(IWK1(J),N,LMIC(4))*WK1(J)
171:         if ( JSCTM.gt.1 ) then
172:            XMU = WKF1(I)
173:            DNF = SMIC(IWK1(J),N,LMIC(4))*WK1(J)
174:            do IM = 2, JSCTM
175:               RMICSM(IE,JE,IR,N,IM,1) =
176:            & RMICSM(IE,JE,IR,N,IM,1) + XMU*(IM-1)*DNF
177:            end do
178:         end if
179:      end do
180:      end if
181: C
182: C***** thermal scattering by free gas model
183: C
184:      if ( NCOLF.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0))then
185:         time approximation
186:         JFREE = 0 ! first trial for short collision
187:         do J = 1, NCOLF
188:            IP = IWK2(J)
189:            WK10(J) = XLIB1(N,9)*EEE(IP)
190:            if ( EEE(IP).le.5.0 ) then
191:               WK4(J) = XLIB1(N,11)*EEE(IP) ! ATW/KTeff*EEE
192:            else
193:               WK4(J) = XLIB1(N,9)*EEE(IP) ! ATW/KT*EEE
194:            end if
195:         end do
196:         if ( NCOLF*7.gt.NBANK*3 ) go to 950
197:         call RANU2( IRAND, R, NCOLF*7, ICON )
198:         NF1 = NCOLF
199:         NF2 = NCOLF*2
200:         NF3 = NCOLF*3
201:         NF4 = NCOLF*4
202:         NF5 = NCOLF*5
203:         NF6 = NCOLF*6
204:         NF0 = 0
205:         do J = 1, NCOLF
206:            IP = IWK2(J)
207:            if ( 4*R(J)*R(J).gt.PI*WK4(J)*(1-R(J))*(1-R(J)) )
208:            & then
209:               X2 = - log(R(NF1+J)*R(NF2+J)) / WK4(J)
210:            else
211:               COS2 = cos(PIH*R(NF1+J))
212:               COS2 = COS2*COS2
213:               X2 = - (log(R(NF2+J))+log(R(NF3+J))*COS2)/WK4(J)
214:            end if
215:            WK5(J) = sqrt(X2) ! V (target nuclide) / V (ne
utron) : KT
216:            WK6(J) = 2*R(NF4+J) - 1 ! target direction sampled i
sotropically
217:            SINT = sqrt(1-WK6(J)*WK6(J))
218:            FAI = PI2*R(NF5+J)
219:            WK7(J) = SINT*cos(FAI)
220:            WK8(J) = SINT*sin(FAI)
221:            XMU = WK6(J)*AAA(IP)+WK7(J)*BBB(IP)+WK8(J)*CCC(IP)
222:            WK9(J) = sqrt(1+X2-2*XMU*WK5(J))
223:            if ( WK9(J).lt.R(NF6+J)*(1+WK5(J)) ) then
224:               NF0 = NF0 + 1
225:               IWK4(NF0) = J
226:            end if
227:         end do
228:         NREJ = 0
229:         continue
230:         if ( NF0.gt.0 ) then ! resampling for rejected particle
231:            NF = NF0
232:            NF0 = 0
233:            if ( NF*7.gt.NBANK*3 ) go to 950
234:            call RANU2( IRAND, R, NF*7, ICON )
235:            NF1 = NF
236:            NF2 = NF*2
237:            NF3 = NF*3
238:            NF4 = NF*4
239:            NF5 = NF*5
240:            NF6 = NF*6
241:         *VOCL LOOP,NOVREC
242:         do K = 1, NF
243:            J = IWK4(K)
244:            IP = IWK2(J)
245:            if ( 4*R(K)*R(K).gt.PI*WK4(J)*(1-R(K))*(1-R(K))

```

src/mvp/talsme.f

```

246:      &                                     ) then
247:      X2 = - log(R(NF1+K)*R(NF2+K)) / WK4(J)
248:      else
249:      COS2 = cos(PIH*R(NF1+K))
250:      COS2 = COS2*COS2
251:      X2 = - (log(R(NF2+K))+log(R(NF3+K))*COS2) /
252:      WK4(J)
253:      end if
254:      WK5(J) = sqrt(X2)
255:      WK6(J) = 2*R(NF4+K) - 1      ! target direction sample
d isotropically
256:      SINT = sqrt(1-WK6(J))*WK6(J)
257:      FAI = PI2*R(NF5+K)
258:      WK7(J) = SINT*cos(FAI)
259:      WK8(J) = SINT*sin(FAI)
260:      XMU=WK6(J)*AAA(IP)+WK7(J)*BBB(IP)+WK8(J)*CCC(IP)
261:      WK9(J) = sqrt(1+X2-2*XMU*WK5(J))
262:      if ( WK9(J).lt.R(NF6+K)*(1+WK5(J)) ) then
263:      NF0 = NF0 + 1
264:      IWK4(NF0) = J
265:      end if
266:      end do
267:      go to 130
268:      end if
269:      if ( JFREE.eq.0 ) then      ! first trial in case of short co
llision time approximation
270:      JFREE = 1
271:      call RANU2( IRAND, R, NCOLF*3, ICON )
272:      NF1 = NCOLF
273:      NF2 = NCOLF*2
274:      *VOCL LOOP,NOVREC
275:      do J = 1, NCOLF
276:      IP = IWK2(J)
277:      U1 = 2*R(J) - 1
278:      SINT = sqrt(1-U1*U1)
279:      FAI = PI2*R(NF1+J)
280:      U2 = SINT*cos(FAI)
281:      U3 = SINT*sin(FAI)
282:      O1 = (WK9(J)*U1+WK5(J)*WK6(J))*ATW
283:      O2 = (WK9(J)*U2+WK5(J)*WK7(J))*ATW
284:      O3 = (WK9(J)*U3+WK5(J)*WK8(J))*ATW
285:      X1 = O1 + AAA(IP)
286:      X2 = O2 + BBB(IP)
287:      X3 = O3 + CCC(IP)
288:      XX2 = X1*X1 + X2*X2 + X3*X3
289:      RATIO = XX2*XLIB1(N,7)
290:      C1 = (WK4(J)-WK10(J))*(RATIO-1)
291:      if ( C1.lt.log(R(NF2+J))*ATW ) then
292:      NF0 = NF0 + 1
293:      IWK4(NF0) = J
294:      IWK5(NF0) = J
295:      else
296:      ENEW = EEE(IP)*RATIO
297:      WK10(J) = max(dble(EBOT),ENEW)
298:      WKF1(IWK12(J)) =
299:      &      (X1*AAA(IP)+X2*BBB(IP)+X3*CCC(IP))
300:      &      /sqrt(XX2)
301:      end if
302:      end do
303:      if ( NF0.gt.0 ) then
304:      NREJ = NF0
305:      go to 130
306:      end if
307:      else      ! subsequent trial in case of short
collision time approximation
246:      call RANU2( IRAND, R, NREJ*3, ICON )
247:      NF1 = NREJ
248:      NF2 = NREJ*2
do K = 1, NREJ
249:      J = IWK5(K)
250:      IP = IWK2(J)
251:      U1 = 2*R(K) - 1
252:      SINT = sqrt(1-U1*U1)
253:      FAI = PI2*R(NF1+K)
254:      U2 = SINT*cos(FAI)
255:      U3 = SINT*sin(FAI)
256:      O1 = (WK9(J)*U1+WK5(J)*WK6(J))*ATW
257:      O2 = (WK9(J)*U2+WK5(J)*WK7(J))*ATW
258:      O3 = (WK9(J)*U3+WK5(J)*WK8(J))*ATW
259:      X1 = O1 + AAA(IP)
260:      X2 = O2 + BBB(IP)
261:      X3 = O3 + CCC(IP)
262:      XX2 = X1*X1 + X2*X2 + X3*X3
263:      RATIO = XX2*XLIB1(N,7)
264:      C1 = (WK4(J)-WK10(J))*(RATIO-1)
265:      if ( C1.lt.log(R(NF2+K))*ATW ) then
266:      NF0 = NF0 + 1
267:      IWK4(NF0) = J
268:      IWK5(NF0) = J
269:      else
270:      ENEW = EEE(IP)*RATIO
271:      WK10(J) = max(dble(EBOT),ENEW)
272:      WKF1(IWK12(J)) =
273:      &      (X1*AAA(IP)+X2*BBB(IP)+X3*CCC(IP))
274:      &      /sqrt(XX2)
275:      end if
276:      end do
277:      if ( NF0.gt.0 ) then
278:      NREJ = NF0
279:      go to 130
280:      end if
281:      call BSVDEC( ENGVB, NGP1+1, WK10, IWK5, NCOLF ) ! outgoing e
nergy group; IWK5
282:      *VOCL LOOP,SCALAR
283:      do J = 1, NCOLF
284:      I = IWK12(J)
285:      IE = IEG(I)
286:      JE = IWK5(J)
287:      IR = IRG(I)
288:      DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
289:      WK2(J)
290:      RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1) +
291:      SMIC(IWK2(J),N,LMIC(4))*WK2(J)
292:      if ( JSCTM.gt.1 ) then
293:      XMU = WKF1(I)
294:      DNF = SMIC(IWK2(J),N,LMIC(4))*WK2(J)
295:      do IM = 2, JSCTM
296:      RMICSM(IE,JE,IR,N,IM,1) =
297:      RMICSM(IE,JE,IR,N,IM,1) + XMU**(IM-1)*DNF
298:      end do
299:      end if
300:      end do
301:      end if
302:      C
303:      C***** thermal scattering (S(a,b) and thermal elastic)
304:      C
305:      if ( NCOLT.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0.or.
306:      JMICE(6).ne.0.or.JMACE(6).ne.0))then

```



```

372:      NTII = 0      ! number of thermal incoherent inel
astic
373:      NTIE = 0      ! number of thermal incoherent elas
tic
374:      NTCE = 0      ! number of thermal coherent elasti
c
375:      NT = 0
376:      *VOCL LOOP,NOVREC
377:      do I = 1, NCOLT
378:      IP = IWK3(I)
379:      WK4(I) = EEE(IP)
380:      WK6(I) = 0
381:      if ( JMICE(4).ne.0.or.JMACE(4).ne.0 ) then
382:      WK5(I) = SMIC(IP,N,LMIC(4))      ! microscopic elastic s
cattering cross section
383:      else
384:      WK5(I) = 0
385:      end if
386:      if ( WK5(I).gt.ZERO ) then
387:      NT = NT + 1
388:      IWK4(NT) = I
389:      else
390:      NTII = NTII + 1
391:      IWK5(NTII) = I
392:      IWK6(NTII) = KLIB2(N,92,13) + 3      ! pointer of incident
t energy for energy distribution
393:      IWK7(NTII) = 1CX(IWK6(NTII)-1)      ! number of incident
energy points
394:      WK7(NTII) = EEE(IP)      ! incident energy
395:      end if
396:      end do
397:      if ( JMICE(6).eq.0.and.JMACE(6).eq.0 ) NTII = 0      ! no requi
ire thermal inelastic
398:      if ( NTII.gt.0.and.(JMICE(6).ne.0.or.JMACE(6).ne.0) )
399:      &      then
400:      if ( NTII.gt.0.and.LMIC(6).eq.0 ) then
401:      do J = 1, NTII
402:      I = IWK5(J)
403:      IP = IWK3(I)
404:      if ( KSPI(IP,N).ge.KLIB2(N,4,3) ) then
405:      L2 = KLIB1(N,1) + KSPI(IP,N)
406:      R1 = (WK4(I)-CX(L2)) / (CX(L2-1)-CX(L2))
407:      L1 = KLIB2(N,4,11) - KLIB2(N,4,3) +
408:      &      KSPI(IP,N)
409:      XIN = CX(L1)*R1 + CX(L1+1)*(1-R1)
410:      else
411:      XIN = 0
412:      end if
413:      WK6(I) = XIN      ! thermal inelastic
scattering cross section
414:      end do
415:      else
416:      do J = 1, NTII
417:      I = IWK5(J)
418:      IP = IWK3(I)
419:      WK6(I) = SMIC(IP,N,LMIC(6))
420:      end do
421:      end if
422:      end if
423:      if ( NT.gt.0 ) then
424:      call RANU2( IRAND, R, NT, ICON )
425:      if ( LMIC(6).eq.0 ) then      ! thermal inelastic cross
section are not given in smic array.
426:      *VOCL LOOP,NOVREC
427:      do J = 1, NT

```

```

428:      I = IWK4(J)
429:      IP = IWK3(I)
430:      if ( JMICE(6).eq.0.and.JMACE(6).eq.0 ) then
431:          XIN = 0
432:      else if ( KSPI(IP,N).ge.KLIB2(N,4,3) ) then
433:          L2 = KLIB1(N,1) + KSPI(IP,N)
434:          R1 = (WK4(I)-CX(L2)) / (CX(L2-1)-CX(L2))
435:          L1 = KLIB2(N,4,11) - KLIB2(N,4,3) +
436:              & KSPI(IP,N)
437:          XIN = CX(L1)*R1 + CX(L1+1)*(1-R1)
438:      else
439:          XIN = 0
440:      end if
441:      WK6(I) = XIN
442:      X2 = (XIN+WK5(I))*R(J)
443:      if ( X2.lt.XIN ) then
444:          NTII = NTII + 1
445:          IWK5(NTII) = I
446:          IWK6(NTII) = KLIB2(N,92,13) + 3
447:          IWK7(NTII) = ICX(IWK6(NTII)-1)
448:          WK7(NTII) = EEE(IP)
449:      else if ( KLIB2(N,93,7).eq.0 ) then
450:          NTCE = NTCE + 1
451:          IWK8(NTCE) = I
452:      else
453:          NTIE = NTIE + 1
454:          IWK9(NTIE) = I
455:      end if
456:      end do
457:      else
458:          ! thermal inelastic cross
section are given in smic array.
459:      *VOCL LOOP,NOVREC
460:      do J = 1, NT
461:          I = IWK4(J)
462:          IP = IWK3(I)
463:          if ( JMICE(6).eq.0.and.JMACE(6).eq.0 ) then
464:              XIN = 0
465:          else
466:              XIN = SMIC(IP,N,LMIC(6))
467:          end if
468:          WK6(I) = XIN
469:          X2 = (XIN+WK5(I))*R(J)
470:          if ( X2.lt.XIN ) then
471:              NTII = NTII + 1
472:              IWK5(NTII) = I
473:              IWK6(NTII) = KLIB2(N,92,13) + 3
474:              IWK7(NTII) = ICX(IWK6(NTII)-1)
475:              WK7(NTII) = EEE(IP)
476:          else if ( KLIB2(N,93,7).eq.0 ) then
477:              NTCE = NTCE + 1
478:              IWK8(NTCE) = I
479:          else
480:              NTIE = NTIE + 1
481:              IWK9(NTIE) = I
482:          end if
483:      end do
484:      end if
485:      if ( NTII.le.0 ) go to 140
486:      call BSINC3( CX, IWK7, WK8, IWK6, WK7, IWK10, NTII )
487:      call RANU2( IRAND, R, NTII, ICON )
488:      *VOCL LOOP,NOVREC
489:      do J = 1, NTII
490:          L2 = IWK6(J) + IWK10(J)

```

src/mvp/talsme.f

```

491:      L1 = L2 - 1
492:      INTE = ICX(L2+IWK7(J))
493:      if ( INTE.eq.5.or.INTE.eq.3 ) then
494:        X1 = log(CX(L1)/WK7(J))/log(CX(L1)/CX(L2))
495:      else
496:        X1 = (CX(L1)-WK7(J))/(CX(L1)-CX(L2))
497:      end if
498:      if ( X1.lt.DZERO ) then
499:        X1 = 0
500:      else if ( X1.gt.1.0D+0 ) then
501:        X1 = 1
502:      end if
503:      IWK10(J) = min(1,int(R(J)+X1)) + IWK10(J)
504:      if ( IWK10(J).le.6 ) IWK10(J) = 1
505:    end do
506: C ..... sampling of thermal inelastic scattering
507:    if ( JSCTM.gt.1 ) then
508:      call RANU2( IRAND, R, NTII*5, ICON )
509:      N3 = NTII*3
510:      N4 = NTII*4
511:    else
512:      call RANU2( IRAND, R, NTII*3, ICON )
513:    end if
514:    N1 = NTII
515:    N2 = NTII*2
516: *VOCL LOOP,NOVREC
517:    do J = 1, NTII
518:      I = IWK5(J)
519:      NELF = IWK7(J)
520:      IIN = IWK10(J)
521:      L1 = IWK6(J) + 3*NELF*IIN - NELF - 1
522:      L2 = min(int(NELF*R(J))+1,NELF)
523:      L3 = L1 + L2
524:      L4 = 2*L2
525:      L4 = min(L4,int(L4+R(N1+J)-CX(L3))) + L1 + NELF
526:      IOUT = ICX(L4)
527:      if ( IOUT.eq.1 ) then
528:        WK8(J) = CX(IWK6(J))
529:      else
530:        L6 = IWK6(J) + IOUT - 1
531:        L5 = L6 - 1
532:        WK8(J) = (CX(L5)-CX(L6))*R(N2+J) + CX(L6)
533:        IOUTT = R(N2+J) + 0.5d0
534:        IOUT = IOUT - IOUTT
535:      end if
536:      WK8(J) = max(WK8(J),EBOT) ! outgoing energy
537:      if ( JSCTM.gt.1 ) then
538:        if ( KLIB1(N,31).ne.1 ) then
539:          I1 = min(IIN,IOUT)
540:          I2 = max(IIN,IOUT)
541:          I22 = I2*(I2-1)/2
542:          L7 = KLIB2(N,92,12) + NELF*2 + (I22+I1-1)*5
543:        else
544:          L7 = KLIB2(N,92,12) + NELF*2 + NELF*5*(IIN-1)
545:          + 5*(IOUT-1)
546:        end if
547:        RANSU = R(N3+J)
548:        if ( RANSU.le.CX(L7+1) ) then
549:          I3 = CX(L7) - RANSU + 1
550:          I3 = 2*I3 - 1
551:          XMU = R(N4+J)*I3
552:        else
553:          I4 = 1 + CX(L7+3) - RANSU
554:          I5 = 1 + CX(L7+2) - RANSU
555:          I5 = 2*I5 - 1
556:
557:          XMU = I4*I5
558:        end if
559:        WKF1(IWK13(I)) = XMU
560:      end if
561:      call BSVDEC( ENGYB, NGP1+1, WK8, IWK4, NTII ) ! outgoing en
562:    end do
563: ergy group: IWK4
564: *VOCL LOOP,SCALAR
565:    do J = 1, NTII
566:      I = IWK5(J)
567:      K = IWK13(I)
568:      IE = IEG(K)
569:      JE = IWK4(J)
570:      IR = IRG(K)
571:      DNFLXSM(IE,JE,IR,N,KY+1) = DNFLXSM(IE,JE,IR,N,KY+1)
572:      + WK3(I)
573:      RMICSM(IE,JE,IR,N,1,KY+1) =
574:      RMICSM(IE,JE,IR,N,1,KY+1)+(WK6(I)+WK5(I))*WK3(I)
575:      if ( JSCTM.gt.1 ) then
576:        XMU = WKF1(K)
577:        DNF = (WK6(I)+WK5(I))*WK3(J)
578:        do IM = 2, JSCTM
579:          RMICSM(IE,JE,IR,N,IM,KY+1) =
580:          RMICSM(IE,JE,IR,N,IM,KY+1)+XMU*(IM-1)*DNF
581:        end do
582:      end if
583:    end do
584:    continue
585:    if ( NTCE.le.0.and.NTIE.le.0 ) go to 150
586: C ..... sampling of thermal elastic scattering (no change outgoing energy)
587: C ..... thermal coherent elastic scattering
588:    if ( NTCE.gt.0 ) then
589:      if ( JSCTM.gt.1 ) call RANU2(IRAND,R,NTCE*2,ICON)
590:      do J = 1, NTCE
591:        I = IWK8(J)
592:        K = IWK13(I)
593:        IE = IEG(K)
594:        JE = IE
595:        IR = IRG(K)
596:        DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
597:        WK3(I)
598:        RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1)+
599:        (WK6(I)+WK5(I))*WK3(I)
600:      if ( JSCTM.gt.1 ) then
601:        IBRAG = IWK10(I)
602:        L5 = ICX(KLIB2(N,93,13)) + 2
603:        L6 = ICX(L5-1)
604:        if ( WK4(I).gt.CX(L5+L6-1) ) IBRAG = L6
605:        if ( IBRAG.eq.0 ) IBRAG = 1
606:        L5BRAG = ICX(L5+2*L6+IBRAG-1)
607:        NBIN = ICX(L5BRAG)
608:        LL2 = min(int(NBIN*R(J))+1,NBIN)
609:        LL3 = L5BRAG + LL2
610:        LL4 = 2*LL2
611:        LL4 = min(LL4,int(LL4+R(NTCE+J)-CX(LL3))) +
612:        L5BRAG + NBIN
613:        L3 = ICX(LL4)
614:        WKF1(K) = 1 - 2*CX(L5+L3-1)/WK4(I)
615:        if ( WKF1(K).gt.1.0 ) then
616:          WKF1(K) = 1
617:        else if ( WKF1(K).lt.-1.0 ) then
618:          WKF1(K) = -1
619:        end if
620:        DNF = (WK6(I)+WK5(I))*WK3(I)
621:        XMU = WKF1(K)

```

src/mvp/talsme.f

```

620:         do IM = 2, JSCTM
621:             RMICSM(IE,JE,IR,N,IM,1) =
622:             &             RMICSM(IE,JE,IR,N,IM,1)+XMU**(IM-1)*DNF
623:         end do
624:         end if
625:     end do
626: end if
627: C ..... thermal incoherent elastic scattering
628:     if ( NTIE.gt.0 ) then
629:         if ( JSCTM.gt.1 ) call RANU2(IRAND,R,NTIE*2,ICON)
630:         do J = 1, NTIE
631:             I = IWK9(I)
632:             K = IWK13(I)
633:             IE = IEG(K)
634:             JE = IE
635:             IR = IRG(K)
636:             DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
637:             &             WK3(I)
638:             RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1)+
639:             &             (WK6(I)+WK5(I))*WK3(I)
640:         if ( JSCTM.gt.1 ) then
641:             LKL1 = KLIB1(N,7)
642:             L3 = min(int(LKL1*R(J))+1,LKL1) +
643:             &             KLIB2(N,93,12) + KLIB2(N,93,7)*2 +
644:             &             KLIB1(N,20)*(IWK10(I)-1)
645:             WKF1(K) = (CX(L3-1)-CX(L3))*R(NTIE+J)+CX(L3)
646:             DNF = (WK6(I)+WK5(I))*WK3(I)
647:             XMU = WKF1(K)
648:             do IM = 2, JSCTM
649:                 RMICSM(IE,JE,IR,N,IM,1) =
650:                 &                 RMICSM(IE,JE,IR,N,IM,1)+XMU**(IM-1)*DNF
651:             end do
652:         end if
653:     end do
654:     end if
655:     continue
656: 150 end if
657: C
658:     else
659: C -----
660: C -----
661: C one group (vectrizable): meaningless ?
662: C -----
663: C
664:         IE = IEG(1)
665:         JE = IE
666:         if ( NCOLN.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0))then
667:             do I = 1, NCOLN
668:                 IR = IRG(IWK11(I))
669:                 DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
670:                 &                 WK1(I)
671:                 RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1) +
672:                 &                 SMIC(IWK1(I),N,LMIC(4))*WK1(I)
673:             end do
674:         end if
675:         if ( NCOLF.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0))then
676:             do I = 1, NCOLF
677:                 IR = IRG(IWK12(I))
678:                 DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
679:                 &                 WK2(I)
680:                 RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1) +
681:                 &                 SMIC(IWK2(I),N,LMIC(4))*WK2(I)
682:             end do
683:         end if
684:         if ( NCOLT.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0))then

```

```

685:         do I = 1, NCOLT
686:             IR = IRG(IWK13(I))
687:             DNFLXSM(IE,JE,IR,N,1) = DNFLXSM(IE,JE,IR,N,1) +
688:             &             WK3(I)
689:             RMICSM(IE,JE,IR,N,1,1) = RMICSM(IE,JE,IR,N,1,1) +
690:             &             SMIC(IWK3(I),N,LMIC(4))*WK3(I)
691:         end do
692:     end if
693:     if ( NCOLT.gt.0.and.(JMICE(6).ne.0.or.JMACE(6).ne.0))then
694:         do I = 1, NCOLT
695:             IR = IRG(IWK13(I))
696:             DNFLXSM(IE,JE,IR,N,KY+1) = DNFLXSM(IE,JE,IR,N,KY+1)
697:             &             + WK3(I)
698:             if ( LMIC(6).eq.0 ) then
699:                 if ( KSPI(IWK3(I),N).ge.KLIB2(N,4,3) ) then
700:                     L2 = KLIB1(N,1) + KSPI(IWK3(I),N)
701:                     R1 = (EEE(IWK3(I))-CX(L2))/ (CX(L2-1)-CX(L2))
702:                     L1 = KLIB2(N,4,11) - KLIB2(N,4,3) +
703:                     &                     KSPI(IWK3(I),N)
704:                     XIN = CX(L1)*R1 + CX(L1+1)*(1-R1)
705:                     RMICSM(IE,JE,IR,N,1,KY+1) =
706:                     &                     RMICSM(IE,JE,IR,N,1,KY+1) + XIN*WK3(I)
707:                 else
708:                     XIN = 0
709:                 end if
710:             else
711:                 RMICSM(IE,JE,IR,N,1,KY+1) =
712:                 &                 RMICSM(IE,JE,IR,N,1,KY+1)
713:                 &                 + SMIC(IWK3(I),N,LMIC(6))*WK3(I)
714:             end if
715:         end do
716:     end if
717:     end if
718: 190 continue
719: C
720: C=====
721: C===== inelastic scattering and (n,2n) reaction
722: C=====
723: C
724:         if ( JMICE(6).eq.0.and.JMACE(6).eq.0.and.
725:         &         JMICE(7).eq.0.and.JMACE(7).eq.0 ) then
726:             go to 390
727:         end if
728: C
729:         if ( JMICE(6).ne.0.or.JMACE(6).ne.0 ) KY = KY + 1
730:         if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) KY2 = KY + 1
731: C
732:         NLEV = KLIB1(N,6)
733:         ATW = XLIB1(N,1)
734:         do I = 1, NN
735:             IP = IBP(I)
736:             WK1(I) = DN*FLX(I)
737:             IWK1(I) = I
738:             L1 = KLIB1(N,1) + KSPI(IP,N)
739:             DWK1(I) = (EEE(IP)-CX(L1)) / (CX(L1-1)-CX(L1))
740:             DWK2(I) = 1 - DWK1(I)
741:         end do
742:
743:         NTH = NN
744:         do ITH = 1, NLEV
745:             M = KLIB2(N,ITH,10)
746:             LCTMT = KLIB2(N,M,6)
747:             NEANG = KLIB2(N,M,7)
748:             NKPF5 = KLIB2(N,M,8)
749:             LSTF4 = KLIB2(N,M,12)

```

src/mvp/talsme.f

```

750:      EIDI = CX(KLIB1(N,1)+KLIB2(N,M,4)-1)
751:      M1 = 0
752:      if ( ITH.lt.NLEV ) then
753:        M1 = KLIB2(N,ITH+1,10)
754:        EIDI2 = CX(KLIB1(N,1)+KLIB2(N,M1,4)-1)
755:      end if
756:
757:      NCOL = NTH
758:      NTH = 0
759: *VOCL LOOP, NOVREC
760:      do J = 1, NCOL
761:        I = IWK1(J)
762:        IP = IBP(I)
763:        if ( EEE(IP).ge.EIDI ) then
764:          NTH = NTH + 1
765:          IWK1(NTH) = I
766:        end if
767:      end do
768:      if ( NTH.le.0 ) go to 390
769:
770:      do J = 1, NTH
771:        I = IWK1(J)
772:        IP = IBP(I)
773:        WK3(J) = EEE(IP)
774:        L2 = KLIB2(N,M,11) - KLIB2(N,M,3) + KSPI(IP,N)
775:        X1 = CX(L2)*DWK1(I) + CX(L2+1)*DWK2(I)
776:        if ( M1.ne.0.and.WK3(J).ge.EIDI2 ) then
777:          L3 = KLIB2(N,M1,11) - KLIB2(N,M1,3) +
778:            & KSPI(IP,N)
779:          X2 = CX(L3)*DWK1(I) + CX(L3+1)*DWK2(I)
780:          X1 = X1 - X2
781:        end if
782:        WK9(J) = X1
783:      end do
784:
785:      if( KLIB2(N,M,5).eq.1 ) then
786:        if( LCTMT.eq.2 ) then
787:          if ( NEANG.le.0.or.NKF5.ne.0 ) then
788:            write(IOW,901) N,M,LCTMT,NEANG,NKF5
789:            stop 666
790:          end if
791:          do J = 1, NTH
792:            IWK4(J) = NEANG
793:            IWK5(J) = LSTF4
794:          end do
795:          call BSDEC3( CX, IWK4, IWK7, IWK5, WK3, IWK3,
796:            & NTH )
797:          call RANU2( IRAND, R, NTH*3, ICON )
798:          N1 = NTH
799:          N2 = NTH*2
800:          do J = 1, NTH
801:            if ( IWK3(J).eq.0 ) IWK3(J) = 1
802:            L2 = IWK5(J) + IWK3(J)
803:            L1 = L2 - 1
804:            INTE = mod(ICX(L2+IWK4(J)),10)
805:            if ( INTE.eq.5.or.INTE.eq.3 ) then
806:              X1 = log(CX(L1)/WK3(J))/log(CX(L1)/CX(L2))
807:            else
808:              X1 = (CX(L1)-WK3(J))/(CX(L1)-CX(L2))
809:            end if
810:            if ( X1.lt.DZERO ) then
811:              X1 = 0
812:            else if ( X1.gt.1.0D+0 ) then
813:              X1 = 1
814:            end if
815:
816:            IT = min(1,int(R(J)+X1)) + IWK3(J)
817:            L3 = min(int(KLIB1(N,7)*R(N1+J))+1,KLIB1(N,7)) +
818:              & IWK5(J) + IWK4(J)*2 + KLIB1(N,20)*(IT-1)
819:            XMU = (CX(L3-1)-CX(L3))*R(N2+J) + CX(L3)
820:            GE = 1 - XLIB2(N,M,2)/WK3(J)
821:            if ( GE.le.DZERO ) then
822:              WK8(J) = WK3(J)*XLIB1(N,7)
823:              WKF1(IWK1(J)) = 1
824:            else
825:              GE = ATW*sqrt(GE)
826:              GM = GE*XMU
827:              X2 = 1 + 2*GM + GE*GE
828:              WK8(J) = WK3(J)*XLIB1(N,7)*X2
829:              WKF1(IWK1(J)) = (1+GM)/sqrt(X2)
830:            end if
831:            WK8(J) = min(WK8(J),ETOP)
832:          end do
833:          call BSVDEC( ENGYB, NGP1+1, WK8, IWK7, NTH ) ! outgoing e
834:
835:          energy group; IWK7
836:          833: *VOCL LOOP,SCALAR
837:          do J = 1, NTH
838:            I = IWK1(J)
839:            IE = IEG(I)
840:            JE = IWK7(J)
841:            IR = IRG(I)
842:            DNFLXSM(IE,JE,IR,N,KY) = DNFLXSM(IE,JE,IR,N,KY)+
843:              & WK1(I)
844:            RMICSM(IE,JE,IR,N,1,KY) =
845:              & RMICSM(IE,JE,IR,N,1,KY) + WK9(J)*WK1(I)
846:            if ( JSCTM.gt.1 ) then
847:              DNF = WK9(J)*WK1(I)
848:              XMU = WKF1(I)
849:              do IM = 2, JSCTM
850:                RMICSM(IE,JE,IR,N,IM,KY) =
851:                  & RMICSM(IE,JE,IR,N,IM,KY)+XMU**((IM-1)*DNF)
852:              end do
853:            end if
854:          end do
855:
856:          C***** continuum inelastic scattering
857:          C
858:          else if ( LCTMT.eq.1 ) then
859:            MTI = M
860:            call TALSMF5( IOW, N, MTI, NBANK, NTH, WK3, NUC,
861:              & NMT, CX, ICX, MCX, KLIB1, KLIB2, IRAND,
862:              & ETOP,
863:              & IWK3, IWK5, IWK6, IWK7, R, WK7, WK10 )
864:            call BSVDEC( ENGYB, NGP1+1, WK7, IWK7, NTH ) ! outgoing
865:
866:            do J = 1, NTH
867:              I = IWK1(J)
868:              IE = IEG(I)
869:              JE = IWK7(J)
870:              IR = IRG(I)
871:              DNFLXSM(IE,JE,IR,N,KY) = DNFLXSM(IE,JE,IR,N,KY)+
872:                & WK1(I)
873:              RMICSM(IE,JE,IR,N,1,KY) =
874:                & RMICSM(IE,JE,IR,N,1,KY) + WK9(J)*WK1(I)
875:              if ( JSCTM.gt.1 ) then
876:                DNF = WK9(J)*WK1(I)
877:                XMU = WK10(J)
878:                WKF1(I) = XMU
879:                do IM = 2, JSCTM
880:                  RMICSM(IE,JE,IR,N,IM,KY) =

```

src/mvp/talsme.f

```

878:      &                      RMICSM(IE,JE,IR,N,IM,KY)+XMU**(IM-1)*DNF
879:      end do
880:      end if
881:      end do
882:      else
883:      MTI = M
884:      call TALSMF6( IOW, N, MTI, NBANK, NTH, WK3, NUC,
885:      &              NMT, CX, ICX, MCX, KLIB1, KLIB2, XLIB1,
886:      &              XLIB2, IRAND, ETOP,
887:      &              IWK3, IWK5, IWK6, IWK7, IWK8, IWK9, R, WK7,
888:      &              WK8, WK6, WK10 )
889:      call BSVDEC( ENGYB, NGP1+1, WK7, IWK7, NTH ) ! outgoing
energy group; IWK7
890:      *VOCL LOOP, SCALAR
891:      do J = 1, NTH
892:      I = IWK1(J)
893:      IE = IEG(I)
894:      JE = IWK7(J)
895:      IR = IRG(I)
896:      DNFLXSM(IE,JE,IR,N,KY) = DNFLXSM(IE,JE,IR,N,KY) +
897:      &              WK1(I)
898:      RMICSM(IE,JE,IR,N,1,KY) =
899:      &              RMICSM(IE,JE,IR,N,1,KY) + WK9(J)*WK1(I)
900:      if ( JSCTM.gt.1 ) then
901:      DNF = WK9(J)*WK1(I)
902:      XMU = WK10(J)
903:      WK1(I) = XMU
904:      do IM = 2, JSCTM
905:      RMICSM(IE,JE,IR,N,IM,KY) =
906:      &              RMICSM(IE,JE,IR,N,IM,KY)+XMU**(IM-1)*DNF
907:      end do
908:      end if
909:      end do
910:      end if
911:      901 format(' XXX(talsme) Discrete inelastic scattering has not ',
912:      & 'kinematic reaction.')
913:      &/' nuclide=',i4,' mt=',i4,' lctmt=',i3,' neang=',
914:      & i4,' nkf5=',i4)
915:      902 format(' XXX(talsme) Continuum inelastic scattering is not ',
916:      & 'allowed the center-of-mass system for MF=5.')
917:      &/' nuclide=',i4,' lctmt=',i3,' nkf5=',i4)
918: C
919: C=====
920: C===== (n,2n) reaction
921: C=====
922: C
923: ccc      if ( JMICE(7).ne.0.or.JMACE(7).ne.0 ) then
924:      else if( KLIB2(N,M,5).eq.2 ) then
925:      if( M.ge.6.and.M.le.9 ) then
926:      go to 390
927:      else
928:      MTI = M
929:      if ( LCTMT.eq.2 ) then
930:      write(IOW,903) N,LCTMT,NKF5
931:      stop 666
932:      else if ( LCTMT.eq.1 ) then
933:      call TALSMF5( IOW, N, MTI, NBANK, NTH, WK3, NUC,
934:      &              NMT, CX, ICX, MCX, KLIB1, KLIB2, IRAND, ETOP,
935:      &              IWK3, IWK5, IWK6, IWK7, R, WK7, WK10 )
936:      else
937:      call TALSMF6( IOW, N, MTI, NBANK, NTH, WK3, NUC,
938:      &              NMT, CX, ICX, MCX, KLIB1, KLIB2, XLIB1, XLIB2,
939:      &              IRAND, ETOP,
940:      &              IWK3, IWK5, IWK6, IWK7, IWK8, IWK9, R, WK7,
941:      &              WK8, WK6, WK10 )
942:      end if
943:      end if
944:      call BSVDEC( ENGYB, NGP1+1, WK7, IWK7, NTH ) ! outgoing en
ergy group; IWK7
945:      *VOCL LOOP, SCALAR
946:      do J = 1, NTH
947:      I = IWK1(J)
948:      IE = IEG(I)
949:      JE = IWK7(J)
950:      IR = IRG(I)
951:      DNFLXSM(IE,JE,IR,N,KY2) = DNFLXSM(IE,JE,IR,N,KY2) +
952:      &              WK1(I)*2.0
953:      RMICSM(IE,JE,IR,N,1,KY2)=RMICSM(IE,JE,IR,N,1,KY2)+
954:      &              WK9(J)*WK1(I)*2
955:      if ( JSCTM.gt.1 ) then
956:      DNF = WK9(J)*WK1(I)*2
957:      XMU = WK10(J)
958:      WK1(I) = XMU
959:      do IM = 2, JSCTM
960:      RMICSM(IE,JE,IR,N,IM,KY2) =
961:      &              RMICSM(IE,JE,IR,N,IM,KY2)+XMU**(IM-1)*DNF
962:      end do
963:      end if
964:      end do
965:      end if
966:      end do
967:      390 continue
968:      903 format(' XXX(talsme) (n,2n) reaction is not allowed the center-',
969:      & '-of-mass system for MF=5.')
970:      &/' nuclide=',i4,' lctmt=',i3,' nkf5=',i4)
971: C
972:      400 continue
973:      end do
974:      end if
975:      go to 410
976: C
977:      950 continue
978:      write(IOW,904)
979:      904 format('/' XXX(talsme) Array size for random number (R) is lack.'
980:      1/' Please increase the size in wkaray.f.')
981:      stop 666
982: C
983:      410 continue
984:      return
985:      end

```

src/mvp/talsmf5.f

```

1:      subroutine TALSMF5( IOW, N, MTF5, NBANK, NCOL5, EIN, NUC, NMT,
2:      &                  CX, ICX, MCX, KLIB1, KLIB2, IRAND, ETOP,
3:      &                  IWK3, IWK5, IWK6, IWK7, R, WK7, CTH )
4: C=<MVP>=====
5: C purpose:  sampling of outgoing energy from MF=5 data (neutron).
6: C called in:  TALSMF
7: C primary arguments:
8: C      N ..... nuclide number for sampling
9: C      MTF5 .... reaction type (MT) for sampling
10: C      NCOL5 ... number of incident neutron for sampling with this
11: C      reaction type
12: C      EIN ..... array of incident neutron energy
13: C      WK7 ..... array of outgoing energy sampled
14: C=====
15:      implicit real*8 (A-H,O-Z)
16:      parameter( PI = 3.141592653589793200D+0 )
17:      parameter( PI2 = 2*PI,  PIH = PI/2 )
18:      parameter( SMALL = 1.0D-35 )
19: C
20: C ..... cross section data
21:      real CX(MCX)
22:      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
23:      real ETOP
24: C
25: C ..... temporary array
26:      real EIN(*)
27:      real CTH(*)
28: C
29: C ..... work array
30:      real R(3*NBANK), WK7(NBANK)
31:      integer IWK3(NBANK), IWK5(NBANK), IWK6(NBANK), IWK7(NBANK)
32: C
33:      real ZERO
34: C
35: C-----
36: C
37:      ZERO = 0
38:      DZERO = 0
39: C
40: C ..... determine energy table used
41:      NKF5 = KLIB2(N,MTF5,8)
42:      LSTF5 = KLIB2(N,MTF5,13)
43:      L2 = ICX(LSTF5+NKF5-1)
44:      LF = ICX(L2)
45: C
46:      do J = 1, NCOL5
47:          IWK5(J) = L2 + 2
48:          IWK6(J) = ICX(L2+1)
49:      end do
50:      call BSDEC3( CX, IWK6, IWK7, IWK5, EIN, IWK3, NCOL5 )
51:      call RANU2( IRAND, R, NCOL5*5, ICON )
52:      if ( NCOL5*5.gt.NBANK*3 ) go to 950
53:      N1 = NCOL5
54:      N2 = NCOL5*2
55:      N3 = NCOL5*3
56:      N4 = NCOL5*4
57:      NBINE1 = KLIB1(N,21)
58:      NBINE = KLIB1(N,8)
59: C
60: C***** sampling of outgoing energy
61: C
62: *VOCL LOOP,NOVREC
63:      do J = 1, NCOL5
64:          if ( IWK3(J).eq.0 ) IWK3(J) = 1
65:          L2 = IWK5(J) + IWK3(J)

```

```

66:      L1 = L2 - 1
67:      INTE0 = ICX(L2+IWK6(J))
68:      INTE1 = INTE0 / 10
69:      INTE = INTE0 - 10*INTE1
70:      if ( INTE.eq.5.or.INTE.eq.3 ) then
71:          X1 = log(CX(L1)/EIN(J))/log(CX(L1)/CX(L2))
72:      else
73:          X1 = (CX(L1)-EIN(J))/(CX(L1)-CX(L2))
74:      end if
75:      if ( X1.lt.DZERO ) then
76:          X1 = 0
77:      else if ( X1.gt.1.0D+0 ) then
78:          X1 = 1
79:      end if
80:      L1 = IWK5(J) + IWK6(J)*2
81: C
82: C ..... LF=1 : tabulated probability
83:      if ( LF.eq.1 ) then
84:          IT = min(1,int(R(J)+X1)) + IWK3(J)
85:          if ( NBINE1.gt.1 ) then
86:              LLE = NBINE1*2*(IT-1) + L1
87:              LLE2 = NBINE1*2*IWK3(J) + L1
88:              LLE1 = LLE2 - NBINE1*2
89:              NEP = NBINE1 - 1
90:              NEP1 = NEP
91:              NEP2 = NEP
92:              L3 = min(NBINE-1,int(NBINE*R(N1+J))) + LLE
93:              L4 = L3 + NBINE1
94:          else
95:              LSTF5E = ICX(L1+IT-1)
96:              NEP = ICX(LSTF5E)
97:              LL2 = min(int(NEP*R(N1+J))+1,NEP)
98:              LL3 = LSTF5E + LL2
99:              LL4 = 2*LL2
100:              LL4 = min(LL4,int(LL4+R(N3+J)-CX(LL3))) + LSTF5E + NEP
101:              LLL = LSTF5E + 3*NEP
102:              L3 = LLL + ICX(LL4)
103:              L4 = L3 + NEP + 1
104:              LLE = LLL + 1
105:              if ( IT.eq.IWK3(J) ) then
106:                  NEP1 = NEP
107:                  LLE1 = LLE
108:                  LSTF5T = ICX(L1+IT)
109:                  NEP2 = ICX(LSTF5T)
110:                  LLE2 = LSTF5T + 3*NEP2 + 1
111:              else
112:                  NEP2 = NEP
113:                  LLE2 = LLE
114:                  LSTF5T = ICX(L1+IT-2)
115:                  NEP1 = ICX(LSTF5T)
116:                  LLE1 = LSTF5T + 3*NEP1 + 1
117:              end if
118:          end if
119:          E0 = CX(L3)
120:          E1 = CX(L3+1)
121:          P0 = CX(L4)
122:          P1 = CX(L4+1)
123:          RRR = (P0+P1)*R(N2+J)
124:          EE = E1 + (E0-E1)*RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
125:          if ( INTE1.eq.2 ) then
126:              EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
127:              EE = EE*EMAX/CX(LLE)
128:          else if ( INTE1.eq.1 ) then
129:              EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
130:              EMIN = CX(LLE1+NEP1) + (CX(LLE2+NEP2)-CX(LLE1+NEP1))*X1

```

src/mvp/talsmf5.f

```

131:      EE = EMIN +
132:      &      (EMAX-EMIN)*(EE-CX(LLE+NEP))/(CX(LLE)-CX(LLE+NEP))
133:      end if
134: C
135: C ..... LF=5 : general evapolution spectrum
136:      else if ( LF.eq.5 ) then
137:          L5 = L1 + IWK3(J)
138:          THETA5 = CX(L5)*X1 + CX(L5-1)*(1-X1)
139:          LL = L1 + IWK6(J)
140:          if ( NBINE.gt.0 ) then
141:              U5 = CX(LL+NBINE1*2)
142:              L6 = min(NBINE-1,int(NBINE*R(J))) + LL
143:              L7 = L6 + NBINE1
144:              NBE = NBINE
145:              NBE1 = NBINE1
146:          else
147:              NBE = ICX(LL)
148:              NBE1 = 1
149:              LL5 = min(int(NBE*R(J))+1,NBE)
150:              LL6 = LL + LL5
151:              LL7 = 2*LL5
152:              LL7 = min(LL7,int(LL7+R(N1+J)-CX(LL6))) + LL + NBINE
153:              L6 = LL + 3*NBE + ICX(LL7)
154:              L7 = L6 + NBE + 1
155:              U5 = CX(LL+5*NBE+3)
156:          end if
157:          EHI5 = EIN(J) - U5
158:          E0 = CX(L6)
159:          E1 = CX(L6+1)
160:          P0 = CX(L7)
161:          P1 = CX(L7+1)
162:          RRR = (P0+P1)*R(N2+J)
163:          X2 = E1 + (E0-E1)*RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
164:          EE = X2*THETA5
165:          if ( EE.gt.EHI5 ) then
166:              LR = 1
167: 110      continue
168:              if ( LR+2.gt.NCOL5*2 ) then
169:                  N0 = max(3,NCOL5*2)
170:                  call RANU2( IRAND, R(N3+1), N0, ICON )
171:                  LR = 1
172:              end if
173:              if ( NBE1.gt.1 ) then
174:                  L6 = min(NBINE-1,int(NBINE*R(N3+LR))) + LL
175:                  L7 = L6 + NBINE1
176:                  LR = LR + 1
177:              else
178:                  LL5 = min(int(NBE*R(N3+LR))+1,NBE)
179:                  LL6 = LL + LL5
180:                  LL7 = 2*LL5
181:                  LL7 = min(LL7,int(LL7+R(N3+LR+1)-CX(LL6))) + LL+NBINE
182:                  L6 = LL + 3*NBE + ICX(LL7)
183:                  L7 = L6 + NBE + 1
184:                  LR = LR + 2
185:              end if
186:              E0 = CX(L6)
187:              E1 = CX(L6+1)
188:              P0 = CX(L7)
189:              P1 = CX(L7+1)
190:              RRR = (P0+P1)*R(N3+LR)
191:              X2 = E1 + (E0-E1)*RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
192:              EE = X2*THETA5
193:              LR = LR + 1
194:              if ( EE.gt.EHI5 ) go to 110
195:          end if

196: C
197: C ..... LF=7 : simple fission spectrum (Maxwellian)
198:      else if ( LF.eq.7 ) then
199:          L3 = L1 + IWK3(J)
200:          THETA7 = CX(L3)*X1 + CX(L3-1)*(1-X1)
201:          U7 = CX(L1+IWK6(J))
202:          EHI7 = EIN(J) - U7
203:          COS1 = cos(PIH*R(J))
204:          COS2 = COS1*COS1
205:          EE = - THETA7*(log(R(N1+J))+log(R(N2+J))*COS2)
206:          if ( EE.gt.EHI7 ) then
207:              LR = 1
208: 120      continue
209:              if ( LR+2.gt.NCOL5*2 ) then
210:                  N0 = max(3,NCOL5*2)
211:                  call RANU2( IRAND, R(N3+1), N0, ICON )
212:                  LR = 1
213:              end if
214:              COS1 = cos(PIH*R(N3+LR))
215:              COS2 = COS1*COS1
216:              EE = - THETA7*(log(R(N3+LR+1))+log(R(N3+LR+2))*COS2)
217:              LR = LR + 3
218:              if ( EE.gt.EHI7 ) go to 120
219:          end if
220: C
221: C ..... LF=9 : evapolution spectrum
222:      else if ( LF.eq.9 ) then
223:          L3 = L1 + IWK3(J)
224:          THETA9 = CX(L3)*X1 + CX(L3-1)*(1-X1)
225:          U9 = CX(L1+IWK6(J))
226:          EHI9 = EIN(J) - U9
227:          EE = - THETA9*log(R(N1+J)*R(J))
228:          if ( EE.gt.EHI9 ) then
229:              LR = 1
230: 130      continue
231:              if ( LR+1.gt.NCOL5*3 ) then
232:                  N0 = max(2,NCOL5*3)
233:                  call RANU2( IRAND, R(N2+1), N0, ICON )
234:                  LR = 1
235:              end if
236:              EE = - THETA9*log(R(N2+LR)*R(N2+LR+1))
237:              LR = LR + 1
238:              if ( EE.gt.EHI9 ) go to 130
239:          end if
240: C
241: C ..... LF=11 : Watt fission spectrum
242:      else if ( LF.eq.11 ) then
243:          L3 = L1 + IWK3(J)
244:          L4 = L3 + IWK6(J)
245:          A = CX(L3)*X1 + CX(L3-1)*(1-X1)
246:          B = CX(L4)*X1 + CX(L4-1)*(1-X1)
247:          U11 = CX(L1+IWK6(J)*2)
248:          EHI11 = EIN(J) - U11
249:          COS1 = cos(PIH*R(J))
250:          COS2 = COS1*COS1
251:          V2 = - A*(log(R(N1+J))+log(R(N2+J))*COS2)
252:          XMU = 2*R(N3+J) - 1
253:          EE = V2 + A*XMU*sqrt(V2*B) + A*A*B/4
254:          if ( EE.gt.EHI11 ) then
255:              LR = 1
256: 140      continue
257:              if ( LR+3.gt.NCOL5 ) then
258:                  N0 = max(4,NCOL5)
259:                  call RANU2( IRAND, R(N4+1), N0, ICON )
260:                  LR = 1

```

src/mvp/talsmf5.f

```

261:          end if
262:          COS1 = cos(PHI*R(N4+LR))
263:          COS2 = COS1*COS1
264:          V2 = - A*(log(R(N4+LR+1))+log(R(N4+LR+2)))*COS2)
265:          XMU = 2*R(N4+LR+3) - 1
266:          EE = V2 + A*XMU*sqrt(V2*B) + A*A*B/4
267:          LR = LR + 4
268:          if ( EE.gt.EHI11 ) go to 140
269:        end if
270:      end if
271: C
272:      EE = min(EE,ETOP)
273:      WK7(J) = EE          ! final outgoing energy
274:    end do
275: C
276: C***** sampling of outgoing scattering cosine
277: C
278:      NEANG = KLIB2(N,MTF5,7)
279:      L2 = KLIB2(N,MTF5,12)
280:      do J = 1, NCOL5
281:        IWK5(J) = L2
282:        IWK6(J) = NEANG
283:      end do
284:      call BSDEC3( CX, IWK6, IWK7, IWK5, EIN, IWK3, NCOL5 )
285:      if ( NCOL5*8.gt.NBANK*3 ) go to 950
286:      call RANU2( IRAND, R, NCOL5*8, ICON )
287:      N5 = NCOL5*5
288:      N6 = NCOL5*6
289:      N7 = NCOL5*7
290:      NBINA = KLIB1(N,7)
291:      NBINA1 = KLIB1(N,20)
292: C
293: *VOCL LOOP,NOVREC
294:      do J = 1, NCOL5
295:        if ( IWK3(J).eq.0 ) IWK3(J) = 1
296:        L2 = IWK5(J) + IWK3(J)
297:        L1 = L2 - 1
298:        INTE0 = ICX(L2+IWK6(J))
299:        INTE1 = INTE0 / 10
300:        INTE = INTE0 - 10*INTE1
301:        if ( INTE.eq.5.or.INTE.eq.3 ) then
302:          X1 = log(CX(L1)/EIN(J))/log(CX(L1)/CX(L2))
303:        else
304:          X1 = (CX(L1)-EIN(J))/(CX(L1)-CX(L2))
305:        end if
306:        if ( X1.lt.DZERO ) then
307:          X1 = 0
308:        else if ( X1.gt.1.0D+0 ) then
309:          X1 = 1
310:        end if
311:        IT = min(1,int(R(N5+J)+X1)) + IWK3(J)
312:        L3 = min(int(NBINA*R(N6+J))+1,NBINA) + IWK5(J) + IWK6(J)*2 +
313:      &      NBINA1*(IT-1)
314:        XMU = (CX(L3-1)-CX(L3))*R(N7+J) + CX(L3)
315:        CTH(J) = XMU
316:      end do
317:      go to 410
318: C
319: 950 continue
320:      write(IOW,901)
321: 901 format(/' XXX(talsmf5) Array size for random number (R) is lack.'
322: 1/'      Please increase the size in wkaray.f.')
323:      stop 666
324: C
325: 410 continue
326:          return
327:          end

```


src/mvp/talsmf6.f

```

1:      subroutine TALSMF6( IOW, N, MTF6, NBANK, NCOL6, EIN, NUC, NMT,
2:      &                    CX, ICX, MCX, KLIB1, KLIB2, XLIB1, XLIB2,
3:      &                    IRAND, ETOP,
4:      &                    IWK3, IWK5, IWK6, IWK7, IWK8, IWK9,
5:      &                    R, WK7, WK8, WK9, CTH )
6: C=<MVP>=====
7: C purpose:  sampling of outgoing energy from MF=6 data (neutron).
8: C called in:  TALSM6
9: C primary arguments:
10: C      N ..... nuclide number for sampling
11: C      MTF6 .... reaction type (MT) for sampling
12: C      NCOL6 ... number of incident neutron for sampling with this
13: C      reaction type
14: C      EIN ..... array of incident neutron energy
15: C      WK7 ..... array of outgoing energy sampled
16: C=====
17:      implicit real*8 (A-H,O-Z)
18:      parameter( PI = 3.141592653589793200D+0 )
19:      parameter( PI2 = 2*PI, PIH = PI/2 )
20:      parameter( SMALL = 1.0D-35 )
21: C
22: C ..... cross section data
23:      real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4)
24:      integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21)
25:      real ETOP
26: C
27: C ..... temporary array
28:      real EIN(*)
29:      real CTH(*)
30: C
31: C ..... work array
32:      real R(3*NBANK), WK7(NBANK), WK8(NBANK), WK9(NBANK)
33:      integer IWK3(NBANK), IWK5(NBANK), IWK6(NBANK), IWK7(NBANK),
34:      &          IWK8(NBANK), IWK9(NBANK)
35: C
36:      real ZERO
37: C
38: C-----
39: C
40:      ZERO = 0
41:      DZERO = 0
42: C
43: C ..... determine energy table used
44:      NK5 = abs(KLIB2(N,MTF6,8))
45:      LSTF5 = KLIB2(N,MTF6,13)
46:      L2 = ICX(LSTF5+NKF5-1)
47:      LF = ICX(L2)
48: C
49:      do J = 1, NCOL6
50:          IWK5(J) = L2 + 2
51:          IWK6(J) = ICX(L2+1)
52:      end do
53:      call BSDEC3( CX, IWK6, IWK7, IWK5, EIN, IWK3, NCOL6 )
54:      N1 = NCOL6
55:      N2 = NCOL6*2
56:      N3 = NCOL6*3
57:      N4 = NCOL6*4
58:      N5 = NCOL6*5
59: C
60: C***** sampling of outgoing energy
61: C
62: C ..... LF=61 : Kalbach-87 density function
63:      if ( LF.eq.61 ) then
64:          ATW = XLIB1(N,1)
65:          AT1 = ATW + 1

```

```

66:      if ( NCOL6*6.gt.NBANK*3 ) go to 950
67:      call RANU2( IRAND, R, NCOL6*6, ICON )
68:      *VOCL LOOP,NOVREC
69:      do J = 1, NCOL6
70:          if ( IWK3(J).eq.0 ) IWK3(J) = 1
71:          L2 = IWK5(J) + IWK3(J)
72:          L1 = L2 - 1
73:          INTE0 = ICX(L2+IWK6(J))
74:          INTE1 = INTE0 / 10
75:          INTE = INTE0 - 10*INTE1
76:          if ( INTE.eq.5.or.INTE.eq.3 ) then
77:              X1 = log(CX(L1)/EIN(J))/log(CX(L1)/CX(L2))
78:          else
79:              X1 = (CX(L1)-EIN(J))/(CX(L1)-CX(L2))
80:          end if
81:          if ( X1.lt.DZERO ) then
82:              X1 = 0
83:          else if ( X1.gt.1.0D+0 ) then
84:              X1 = 1
85:          end if
86:          L1 = IWK5(J) + IWK6(J)*2
87: C
88:          IT = min(1,int(R(J)+X1)) + IWK3(J)
89:          LSTF5E = ICX(L1+IT-1)
90:          NEP = ICX(LSTF5E)
91:          LSTF5E = LSTF5E + 1
92:          ND = ICX(LSTF5E)
93:          LL2 = min(int(NEP*R(N1+J))+1,NEP)
94:          LL3 = LSTF5E + LL2
95:          LL4 = 2*LL2
96:          LL4 = min(LL4,int(LL4+R(N3+J)-CX(LL3))) + LSTF5E + NEP
97:          LL5 = ICX(LL4)
98:          LLL = LSTF5E + 3*NEP
99:          L3 = LLL + LL5
100:          L4 = L3 + NEP + 1
101:          L5 = L4 + NEP + 1
102:          L6 = L5 + NEP + 1
103:          E0 = CX(L3)
104:          E1 = CX(L3+1)
105:          P0 = CX(L4)
106:          P1 = CX(L4+1)
107:          R0 = CX(L5)
108:          R1 = CX(L5+1)
109:          A0 = CX(L6)
110:          A1 = CX(L6+1)
111:          if ( LL5.le.ND ) then          ! continuous spectra
112:              RRR = (P0+P1)*R(N2+J)
113:              RRR = RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
114:              PRECF = R1 + (R0-R1)*RRR
115:              ANGDS = A1 + (A0-A1)*RRR
116:          else          ! discrete spectra
117:              RRR = 0
118:              PRECF = R1
119:              ANGDS = A1
120:          end if
121:          EE = E1 + (E0-E1)*RRR
122:          if ( R(N4+J).gt.PRECF ) then
123:              T = (2*R(N5+J)-1)*sinh(ANGDS)
124:              XCM = log(T+sqrt(T*T+1))/ANGDS
125:          else
126:              XCM = log(R(N5+J)*exp(ANGDS) + (1-R(N5+J))*exp(-ANGDS)) /
127:              &          ANGDS
128:          end if
129:          if ( INTE1.gt.0.and.LL5.le.ND ) then
130:              LLE = LLL + 1

```

```

      LLE2 = LLE
      LSTF5T = ICX(LLE+IT-2)
      NEP1 = ICX(LSTF5T)
      ND1 = ICX(LSTF5T+1)
      LLE1 = LSTF5T + 3*NEP1 + 2
    end if
    if ( INTE1.eq.2 ) then
      EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
      EE = EE*EMAX/CX(LLE)
    else if ( INTE1.eq.1 ) then
      EMAX = CX(LLE1) + (CX(LLE2)-CX(LLE1))*X1
      EMIN = CX(LLE1+ND1) + (CX(LLE2+ND2)-CX(LLE1+ND1))*X1
      EE = EMIN +
        (EMAX-EMIN)*(EE-CX(LLE+ND1))/(CX(LLE)-CX(LLE+ND1))
    end if
    end if
    ECM = EE
    EE = EE + (EIN(J)+2*AT1*XCM*sqrt(EIN(J)*EE))/AT1/AT1 ! CM --> La
    CTH(J) = XCM*sqrt(ECM/EE) + sqrt(EIN(J)/EE)/AT1 ! cosine: C
  end do
  EE = min(EE,ETOP)
  WK7(J) = EE
end do

..... LF=66 : n-body phase space distribution
else if ( LF.eq.66 ) then
  ATW = XLIB1(N,1)
  QVL = XLIB2(N,MTF6,1)
  AT1 = ATW + 1
  if ( NCOL6*5.gt.NBANK*3 ) go to 950
  call RANU2( IRAND, R, NCOL6*5, ICON )
LOOP,NOVREC
  do J = 1, NCOL6
    LL = IWK5(J) + IWK6(J)*2
    APSX = CX(LL)
    IWK7(J) = ICX(LL+IWK6(J))
    EIMAX = ATW*EIN(J)/(ATW+1) - QVL
    WK7(J) = EIMAX*(APSX-1)/APSX
    R1 = sqrt(R(J))
    T1 = PIH*R(N1+J)
    X1 = (R1*cos(T1))**2
    S1 = (R1*sin(T1))**2 + X1
    WK8(J) = - X1*log(S1)/S1 - log(R(N2+J))
  end do

```

b. system

b. system

M --> Lab. system

src/mvp/talsmf6.f

```

257:         end if
258:         if ( X1.lt.DZERO ) then
259:             X1 = 0
260:         else if ( X1.gt.1.0D+0 ) then
261:             X1 = 1
262:         end if
263:         L1 = IWK5(J) + IWK6(J)*2
264:         IT = min(1,int(R(J)+X1)) + IWK3(J)
265:         LSTF5E = ICX(L1+IT-1)
266:         IWK8(J) = LSTF5E + NBINA1 + 2
267:         IWK9(J) = ICX(IWK8(J)-1)
268:         if ( INTE1.gt.0 ) then
269:             if ( IT.eq.IWK3(J) ) then
270:                 LSTF5T = ICX(L1+IT)
271:                 IWK5(J) = LSTF5T + NBINA1 + 2
272:                 IWK6(J) = ICX(IWK5(J)-1)
273:                 WK8(J) = X1
274:             else
275:                 LSTF5T = ICX(L1+IT-2)
276:                 IWK5(J) = LSTF5T + NBINA1 + 2
277:                 IWK6(J) = ICX(IWK5(J)-1)
278:                 WK8(J) = 1 - X1
279:             end if
280:         end if
281:     end do
282:     call BSINC3( CX, IWK9, R, IWK8, WK9, IWK3, NCOL6 )
283:     call RANU2( IRAND, R, NCOL6*4, ICON )
284:     if ( NCOL6*4.gt.NBANK*3 ) go to 950
285:     do J = 1, NCOL6
286:         if ( IWK3(J).eq.0 ) IWK3(J) = 1
287:         L2 = IWK8(J) + IWK3(J)
288:         L1 = L2 - 1
289:         L3 = IWK8(J) + IWK9(J) - 1
290:         INTA = ICX(IWK8(J)-2)
291:         if ( INTA.eq.1 ) then
292:             LST2A = ICX(L3+IWK3(J))
293:         else
294:             X1 = (CX(L1)-WK9(J))/(CX(L1)-CX(L2))
295:             if ( X1.lt.DZERO ) then
296:                 X1 = 0
297:             else if ( X1.gt.1.0D+0 ) then
298:                 X1 = 1
299:             end if
300:             IT = min(1,int(R(J)+X1)) + IWK3(J)
301:             LST2A = ICX(IT+L3)
302:         end if
303:         NEP = ICX(LST2A)
304:         LL2 = min(int(NEP*R(N1+J))+1,NEP)
305:         LL3 = LST2A + LL2
306:         LL4 = 2*LL2
307:         LL4 = min(LL4,int(LL4+R(N2+J)-CX(LL3))) + LST2A + NEP
308:         LLL = LST2A + 3*NEP
309:         L3 = LLL + ICX(LL4)
310:         L4 = L3 + NEP + 1
311:         E0 = CX(L3)
312:         E1 = CX(L3+1)
313:         P0 = CX(L4)
314:         P1 = CX(L4+1)
315:         RRR = (P0+P1)*R(N3+J)
316:         EE = E1 + (E0-E1)*RRR/(sqrt(P1*P1+(P0-P1)*RRR)+P1+SMALL)
317:         if ( INTE1.gt.0 ) then
318:             R(J) = CX(LL4+1)
319:             R(NBANK+J) = CX(LL4+NEP+1)
320:             WK7(J) = EE
321:         else

```

```

322:             EE = min(EE,ETOP)
323:             WK7(J) = EE
324:         end if
325:     end do
326:     if ( INTE1.gt.0 ) then
327:         NB2 = NBANK*2
328:         call BSINC3( CX, IWK6, R(NB2+1), IWK5, WK9, IWK3, NCOL6 )
329:         call RANU2( IRAND, R(NB2+1), NCOL6, ICON )
330:         do J = 1, NCOL6
331:             if ( IWK3(J).eq.0 ) IWK3(J) = 1
332:             L2 = IWK5(J) + IWK3(J)
333:             L1 = L2 - 1
334:             L3 = IWK5(J) + IWK6(J) - 1
335:             INTA = ICX(IWK5(J)-2)
336:             if ( INTA.eq.1 ) then
337:                 LST2A = ICX(L3+IWK3(J))
338:             else
339:                 X1 = (CX(L1)-WK9(J))/(CX(L1)-CX(L2))
340:                 if ( X1.lt.DZERO ) then
341:                     X1 = 0
342:                 else if ( X1.gt.1.0D+0 ) then
343:                     X1 = 1
344:                 end if
345:                 IT = min(1,int(R(NBANK+J)+X1)) + IWK3(J)
346:                 LST2A = ICX(IT+L3)
347:             end if
348:             NEP = ICX(LST2A)
349:             LLE = LST2A + 3*NEP + 1
350:             if ( INTE1.eq.2 ) then
351:                 EMAX = R(J) + (CX(LLE)-R(J))*WK8(J)
352:                 EE = WK7(J)*EMAX/R(J)
353:             else
354:                 EMAX = R(J) + (CX(LLE)-R(J))*WK8(J)
355:                 EMIN = R(NBANK+J) + (CX(LLE+NEP)-R(NBANK+J))*WK8(J)
356:                 EE = EMIN + (EMAX-EMIN)*(WK7(J)-R(NBANK+J))/
357:                     (R(J)-R(NBANK+J))
358:             &
359:             end if
360:             EE = min(EE,ETOP)
361:             WK7(J) = EE
362:         end do
363:     end if
364:     go to 410
365: C
366:     950 continue
367:     write(IOW,901)
368:     901 format('/' XXX(talsmf6) Array size for random number (R) is lack.'
369:         1/' Please increase the size in wkaray.f.')
370:     stop 666
371: C
372:     410 continue
373:     return
374: end

```

src/mvp/talspt.f

```

1:      subroutine TALSPT( IOW,  A, H, CHA,
2:      &                  NBATCH,NSKIP, NBPINT,JBPRNT,NGROUP,NGP1,
3:      &                  NGP2,  NPKIND,NREG,  NRESP, NGENEO,NUC,
4:      &                  NEMIC, NEMAC, NSTAL, NFISB, NEVENT,NTREG,
5:      &                  XSOC,  XKEFP, WCTRP, NUCPT, NTPT )
6: C=<MVP>=====
7: C PURPOSE: Sum perturbation tally arrays after each batch.
8: C CALLED IN: ACTION
9: C Written by Y.Nagaya 22 Nov 1999
10: C UPDATE:
11: C=====
12:      implicit real*8(A-H,O-Z)
13: c
14:      include 'INC/_FLAGS'
15:      include '../shared/INC/_TASKDT'
16: c
17: c ... dynamic size memory array ...
18: c
19:      real A(*)
20:      real H(*)
21:      character*4 CHA(*)
22: c
23:      real*8 XSOC(*)
24:      real*8 XKEFP(NUCPT,NTPT,7,*)
25:      real*8 WCTRP(NUCPT,7,NTPT)
26: c
27: c XSOC   : Cumulative total source weight
28: c XKEFP  : Cumulative dk/da weight ( not normalized by source weight )
29: c
30: c
31: C-----
32: C TALLY FOR DELTA-K
33: C-----
34: c
35:      if ( JEIGN.ne.0 ) then
36:
37:          if ( NBATCH.ge.2 ) then
38:
39:              do 109 INC = 1, NUCPT
40:                  do 110 IPT = 1, NTPT
41:
42: c                  XKEF(1,NBATCH)  = WCNTR(15,1) + XKEF(1,NBATCH-1)
43:
44:                      XKEFP(INC,IPT,2,NBATCH) = WCTRP(INC,2,IPT)
45:                  &
46:                      + XKEFP(INC,IPT,2,NBATCH-1)
47: c
48: c                  XKEF(3,NBATCH)  = WCNTR(21,1) + XKEF(3,NBATCH-1)
49: c                  XKEF(4,NBATCH)  = WCNTR(20,1) + XKEF(4,NBATCH-1)
50: c                  XKEF(5,NBATCH)  = WCNTR(19,1) + XKEF(5,NBATCH-1)
51: c                  XKEF(6,NBATCH)  = WCNTR(22,1) + XKEF(6,NBATCH-1)
52: c                  XKEF(7,NBATCH)  = WCNTR(7,1) + XKEF(7,NBATCH-1)
53: c                  XSOC(NBATCH)    = XSOC(NBATCH) + XSOC(NBATCH-1)
54:                  110      continue
55:                  109      continue
56:
57:              else
58:
59:                  do 119 INC = 1, NUCPT
60:                      do 120 IPT = 1, NTPT
61:
62: c                      XKEF(1,1)    = WCNTR(15,1)
63:
64:                          XKEFP(INC,IPT,2,1) = WCTRP(INC,2,IPT)
65:
66: c                      XKEF(5,1)    = WCNTR(19,1)
67: c                      XKEF(6,1)    = WCNTR(22,1)
68: c                      XKEF(7,1)    = WCNTR(7,1)
69:                  120      continue
70:                  119      continue
71:
72:              end if
73:
74: c                  WLEK    = WLEK + WCNTR(7,1)
75: c                  NCNTR(2,1) = 0D0
76: c                  NCNTR(14,1) = 0D0
77: c                  NCNTR(15,1) = 0D0
78: c                  NCNTR(19,1) = 0D0
79: c                  NCNTR(20,1) = 0D0
80: c                  NCNTR(21,1) = 0D0
81: c                  NCNTR(22,1) = 0D0
82: c                  WCNTR(2,1)  = 0.D0
83: c                  WCNTR(14,1) = 0.D0
84: c                  WCNTR(15,1) = 0.D0
85: c                  WCNTR(19,1) = 0.D0
86: c                  WCNTR(20,1) = 0.D0
87: c                  WCNTR(21,1) = 0.D0
88: c                  WCNTR(22,1) = 0.D0
89: c                  WCNTR(7,1)  = 0.D0
90:
91:                  do 129 INC = 1, NUCPT
92:                      do 130 IPT = 1, NTPT
93:                          WCTRP(INC,2,IPT) = 0.0d0
94:                  130      continue
95:                  129      continue
96: c
97: c ... fixed source ...
98: c
99:      else
100: c ... The perturbation option has not been supported for fixed source
101: c problems yet. ...
102: c                  WLEK    = WCNTR(7,1)
103: c
104:      end if
105:
106:      return
107:      end

```

src/mvp/talsum.f

```

1:      subroutine TALSUM( IOW,  A, H, CHA,
2:      &                  NBATCH,NSKIP, NTMINT,NBPINT,JBPRNT,NGROUP,
3:      &                  NPKIND,NREG,  NRESP, NGENEO,WSUMB,  NUC,
4:      &                  NEMIC, NEMAC, NSTAL, NFISB, NEVENT,NTREG,
5:      &                  IPTRG, LPTRG, WSUM,  WLEK,  FLTR,  FLCL,
6:      &                  SFLTR, SFLCL, RESP,  SRETR, SRECL, XSOC,  XKEF,
7:      &                  NCNTR, WCNTR, WCXTY, RMIC,  SRMIC, RMICR, XMIC,
8:      &                  DNFLX, RMAC,  RMACR, XMAC,  DMAC,  RSTR,  RSCL,
9:      &                  SRSTR, SRSCl, LEMIC, LEMAC, TFLH,  TFLHS,
10:     &                  IETAL, NETALY, IDTAL, NDTALY, ETALY, NLETAL,
11:     &                  DTALY, NLDTAL,
12:     &                  NETRV, METRV, IETRV, ETRV,
13:     &                  DNFLXSM, RMICSM, SRMICSM, XMICSM, RMACSM,
14:     &                  XMACSM, KY, KYPOS, NZONE, KZREG, DNZON, NGP1,
15:     &                  RMIMU, SMIMU, WMIMU, RMAMU,
16:     &                  NSUZON, NSPACE, KCELL, ISUSP, NINPZ, KMAT,
17:     &                  KREG,
18:     &                  FTR,  FCL,  FLR,  DNF,  RRR,
19:     &                  RTR,  RCL,  STR,  SCL,  DWKR,
20:     &                  XKEF1, XSOEB, XKB, DNFSM, RRRSM, KJSCTM,
21: c+beff1
22:     &                  NEBEF, WCBEF, XBEF )
23: c-beff1
24: C=<MVP>=====
25: C PURPOSE: TAKE SUMMATION OF TALLY ARRAYS AFTER EACH BATCH.
26: C CALLED IN: ACTION
27: C=====
28:      implicit real*8(A-H,O-Z)
29: c##<2007/03/14:PN4:
30:      parameter ( MAXXK = 10 )
31: c##>
32: C
33:      include 'INC/_KPIDS'
34:      include 'INC/_NGPS'
35: C
36:      include 'INC/_FLAGS'
37:      include '../shared/INC/_TASKDT'
38: C
39: C ... dynamic size memory array ...
40: C
41:      real A(*)
42:      real H(*)
43:      character*4 CHA(*)
44: C
45:      real*8 WSUM, WLEK, WSUMB
46:      real RESP(NGROUP,NRESP)
47:      real*8 FLTR(NGROUP,NREG), SFLTR(NGROUP,NTREG,2),
48:      &      FLCL(NGROUP,NREG), SFLCL(NGROUP,NTREG,2)
49:      real*8 SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
50: c##<2007/03/14:PN4:
51: c##      real*8 XSOC(*), XKEF(7,*)
52:      real*8 XSOC(*), XKEF(MAXXK,*)
53: c##>
54: CCC      real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
55:      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
56: C
57: C ..... MICROSCOPIC REACTION RATE .....
58: C
59:      real*8 RMIC(NGROUP,NREG,NUC,NEMIC),
60:      &      SRMIC(NGROUP,NTREG,NUC,NEMIC,2),
61:      &      RMICR(NPKIND,NTREG,NUC,NEMIC,2),
62:      &      XMIC(NGROUP+NPKIND,NTREG,NUC,NEMIC,2),
63:      &      DNFLX(NGROUP,NREG,NUC)
64:      integer LEMIC(8,2)
65:      real*8 WCXTY(NGROUP+NPKIND,NTREG)

```

```

66:      real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*),
67: c%c &      XMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,*),
68:      &      RMICSM(NGP1,NGP1,NREG,NUC,KJSCTM,*),
69:      &      SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,KJSCTM,*)
70:      integer KYPOS(3)
71:      real*8 RMIMU(NGP1,NGP1+1,NREG,NUC,*),WMIMU(NGP1,NGP1+1,NREG,NUC,*)
72:      real*8 SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,*)
73: C
74: C ..... MACROSCOPIC REACTION RATE .....
75: C
76:      real*8 RMAC(NGROUP,NTREG,NEMAC,2), RMACR(NPKIND,NTREG,NEMAC,2),
77:      &      DMAC(NGROUP,NTREG), XMAC(NGROUP+NPKIND,NTREG,NEMAC,2)
78:      integer LEMAC(8,2)
79:      real*8 RMACSM(NGP1+1,NGP1+1,NTREG,2,KJSCTM,*)
80: c%c      real*8 XMACSM(NGP1+1,NGP1+1,NTREG,2,*)
81: C
82:      real DNZON(NUC,NZONE,2)
83:      integer KZREG(NZONE), KCELL(NZONE), ISUSP(NSUZON,NSPACE)
84:      integer KMAT(NINPZ), KREG(NINPZ)
85:      real*8 RMAMU(NGP1+1,NGP1+2,NTREG,3,*)
86: C
87: C .... weighted flight time
88: C
89: CC      real*8 TFLH(NPKIND)
90: CC      real*8 TFLHS(NPKIND,2)
91:      real*8 TFLH(KPLIM)
92:      real*8 TFLHS(KPLIM,2)
93: c+beff1
94: C
95: C ..... beta-effective
96: C
97:      real*8 WCBEF(NEBEF)
98:      real*8 XBEF(NEBEF,*)
99: c-beff1
100: C
101: C ..... SPECIAL TALLY .....
102: C
103:      integer IETAL(*)
104:      integer IDTAL(*)
105:      real*8 ETALY(NLETAL,2)
106:      real*8 DTALY(NLDTAL)
107: c
108:      real*8 RSTR(NREG,NSTAL), RSCL(NREG,NSTAL)
109:      real*8 SRSTR(NTREG,NSTAL,2), SRSCl(NTREG,NSTAL,2)
110: C
111: C
112: C ..... REGION / TALLY-REGION TABLE .....
113: C
114:      integer IPTRG(NTREG+1), LPTRG(*)
115: C
116: C .... special tallies requiring "real variance" calculation.
117: C
118:      integer IETRV(2,NETRV+1)
119:      real*8 ETRV(METRV,*)
120: C
121: C ..... WORKING ARRAY .....
122: C
123:      real*8 FLR(NTREG,NPKIND), DNF(NGROUP,NTREG)
124:      real*8 FTR(NGROUP,NTREG)
125:      real*8 FCL(NGROUP,NTREG)
126:      real*8 RRR(NGROUP,NTREG)
127:      real*8 RTR(NTREG,NRESP), RCL(NTREG,NRESP)
128:      real*8 STR(NTREG,NSTAL), SCL(NTREG,NSTAL)
129:      real*8 DWKR(*)
130: c##<2007/03/14:PN4:

```

src/mvp/talsum.f

```

131: c##    real*8 XKEF1(7,*)
132:    real*8 XKEF1(MAXXX,*)
133: c##>
134:    real*8 XSOCB(*)
135: c##<2007/03/14:PN4:
136: c##    real*8 XKB(7,*)
137:    real*8 XKB(MAXXX,*)
138: c##>
139:    real*8 DNFSM(NGP1,NGP1+1), RRRSM(NGP1,NGP1+1,KJSCTM)
140: C
141: C ... local array ....
142: C
143: c##<2007/03/14:PN4:
144: c##    real*8 XKAV1(6), XKER1(6), XKAV2(6), XKER2(6)
145: c##    real*8 XKAV3(6), XKER3(6)
146:    real*8 XKAV1(MAXXX), XKER1(MAXXX), XKAV2(MAXXX), XKER2(MAXXX)
147:    real*8 XKAV3(MAXXX), XKER3(MAXXX)
148: c##>
149:    real*8 FMOM(10)
150: C
151: C ... statement functions ...
152: C
153:    include '../shared/INC/_PMLATT'
154:    include '../shared/INC/_SFLATT'
155: C
156: C-----
157:    NBATCH = NBATCH + 1
158:
159:    DZERO = 0
160: C
161: C-----
162: C NSNEU,NSPHT:  POSITION OF ENERGY GROUP SUM FOR
163: C      SUCH AS (... ,NGROUP+NPKIND, ...) ARRAY
164: C ISNEU,ISPHT:  PARTICLE INDEX FOR SUCH AS (... ,NPKIND, ...) ARRAY
165: C-----
166: C    NSNEU = NGROUP
167: C    if ( JNEUT.ne.0 ) NSNEU = NGROUP + 1
168: C    ISNEU = NSNEU - NGROUP
169: C
170: C    NSPHT = 1
171: C    ISPHT = 0
172: C    if ( JPHOT.ne.0 ) then
173: C      NSPHT = NSNEU + 1
174: C      ISPHT = NSPHT - NGROUP
175: C    end if
176: C
177: C-----
178: C  Do something in customized version
179: C-----
180:    call ZTTALS( NGENE0, WSUMB, A, A, H, H, CHA )
181: C
182: C-----
183: C  CLEAR TALLY ARRAYS FOR REJECTED BATCHES
184: C-----
185:    if ( NBATCH.eq.NSKIP ) then
186:      do 110 IR = 1, NREG
187:        do 100 IG = 1, NGROUP
188:          FLTR(IG,IR) = 0.0D0
189:          FLCL(IG,IR) = 0.0D0
190:        100 continue
191:      110 continue
192: CC
193:    if ( JRESP.ne.0.and.NSTAL.gt.0 ) then
194:      do 130 N = 1, NSTAL
195:        do 120 IR = 1, NREG

```

```

196:          RSTR(IR,N) = 0.0D0
197:          RSCL(IR,N) = 0.0D0
198:        120 continue
199:      130 continue
200:    end if
201: C
202:    do 170 L = 1, NEMIC
203:      do 160 IN = 1, NUC
204:        do 150 IR = 1, NREG
205:          do 140 IG = 1, NGROUP
206:            RMIC(IG,IR,IN,L) = 0.0D0
207:          140 continue
208:        150 continue
209:      160 continue
210:    170 continue
211: C
212:    if ( NEMIC.gt.0 ) then
213:      do 190 IR = 1, NTREG
214:        do 180 IG = 1, NGROUP + NPKIND
215:          WCXTY(IG,IR) = 0.0D0
216:        180 continue
217:      190 continue
218:      do 220 IN = 1, NUC
219:        do 210 IR = 1, NREG
220:          do 200 IG = 1, NGROUP
221:            DNFLX(IG,IR,IN) = 0.0D0
222:          200 continue
223:        210 continue
224:      220 continue
225:    end if
226: C
227:    if ( JSCTM.ne.0 ) then
228:      do K = 1, KY
229:        do IN = 1, NUC
230:          do IR = 1, NREG
231:            do IG2 = 1, NGP1
232:              do IG1 = 1, NGP1
233:                DNFLXSM(IG1,IG2,IR,IN,K) = 0
234:                do IM = 1, JSCTM
235:                  RMICSM(IG1,IG2,IR,IN,IM,K) = 0
236:                end do
237:              end do
238:            end do
239:          end do
240:        end do
241:      end do
242:    end if
243: C
244:    if ( JSCMU.ne.0 ) then
245:      do K = 1, KY
246:        do IN = 1, NUC
247:          do IR = 1, NREG
248:            do IG2 = 1, NGP1+1
249:              do IG1 = 1, NGP1
250:                WMIMU(IG1,IG2,IR,IN,K) = 0
251:                RMIMU(IG1,IG2,IR,IN,K) = 0
252:              end do
253:            end do
254:          end do
255:        end do
256:      end do
257:    end if
258: C
259:    WSUM = 0.0D0
260:    if ( JTIME.ne.0 ) then

```

src/mvp/talsum.f

```

261: CCCCCC      do 230 K = 1, NPKIND
262:              do 230 K = 1, KPLIM
263:                TFLH(K) = 0.0D0
264:                TFLHS(K,1) = 0.0D0
265:                TFLHS(K,2) = 0.0D0
266:      230      continue
267:      end if
268: C
269:      if ( NDTALY.gt.0 ) then
270:        do 240 I = 1, NLDTAL
271:          DTALY(I) = 0.0D0
272:      240      continue
273:      end if
274:      if ( NETALY.gt.0 ) then
275:        do 260 J = 1, 2
276:          do 250 I = 1, NLETAL
277:            ETALY(I,J) = 0.0D0
278:          250      continue
279:        260      continue
280:      end if
281: C
282: C-----
283: C TAKE TALLIES FOR FLUX,REACTION RATE & CROSS SECTION
284: C-----
285: C
286:      else if ( NBTACH.gt.NSKIP ) then
287:        DNG = DBLE(NGENEN)
288:        DNG = WSUMB
289: C-----
290: C FLUX TALLY
291: C-----
292:      do 280 KR = 1, NTREG
293:        do 270 IG = 1, NGROUP
294:          FTR(IG,KR) = 0.0D0
295:          FCL(IG,KR) = 0.0D0
296:        270      continue
297:      280      continue
298:      do 310 KR = 1, NTREG
299:        do 300 MR = IPTRG(KR), IPTRG(KR+1) - 1
300:          IR = LPTRG(MR)
301:          do 290 IG = 1, NGROUP
302:            FTR(IG,KR) = FTR(IG,KR) + FLTR(IG,IR)
303:            FCL(IG,KR) = FCL(IG,KR) + FLCL(IG,IR)
304:          290      continue
305:        300      continue
306:      310      continue
307: C
308:      do 330 KR = 1, NTREG
309:        do 320 IG = 1, NGROUP
310:          SFLTR(IG,KR,1) = SFLTR(IG,KR,1) + FTR(IG,KR)
311:          SFLCL(IG,KR,1) = SFLCL(IG,KR,1) + FCL(IG,KR)
312:          SFLTR(IG,KR,2) = SFLTR(IG,KR,2) + (FTR(IG,KR)**2) /DNG
313:          SFLCL(IG,KR,2) = SFLCL(IG,KR,2) + (FCL(IG,KR)**2) /DNG
314:        320      continue
315:      330      continue
316: C
317: check %%%%%%%%%%
318: C write(*,'(lx,a)') '%%% sfltr neutron ngpl-1'
319: C write(*,'(lx,10e11.4)') (fltr(ngp(kpneut)-1,ir),ir=1,ntreg)
320: C write(*,'(lx,a)') '%%% sfltr neutron ngpl'
321: C write(*,'(lx,10e11.4)') (fltr(ngp(kpneut),ir),ir=1,ntreg)
322: C %%%%%%%%%%
323: C
324: C
325: C

```

```

326: C -----
327: C MICROSCOPIC REACTION RATE & CROSS SECTION
328: C -----
329: C
330:      if ( NEMIC.ne.0 ) then
331:        if ( JRTTR.ne.0 ) then
332:          do 350 KR = 1, NTREG
333:            do 340 IG = 1, NGROUP
334:              if ( FTR(IG,KR).ne.0.0D0 ) WCXTY(IG,KR) =
335:                & WCXTY(IG,KR) + DNG
336:            340      continue
337:          350      continue
338:        else
339:          do 370 KR = 1, NTREG
340:            do 360 IG = 1, NGROUP
341:              if ( FCL(IG,KR).ne.0.0D0 ) WCXTY(IG,KR) =
342:                & WCXTY(IG,KR) + DNG
343:            360      continue
344:          370      continue
345:        end if
346: C
347:      do 520 IN = 1, NUC
348: C
349:        do 390 KR = 1, NTREG
350:          do 380 IG = 1, NGROUP
351:            DNF(IG,KR) = 0.0D0
352:          380      continue
353:        390      continue
354:        do 420 KR = 1, NTREG
355:          do 410 MR = IPTRG(KR), IPTRG(KR+1) - 1
356:            IR = LPTRG(MR)
357:            do 400 IG = 1, NGROUP
358:              DNF(IG,KR) = DNF(IG,KR) + DNFLX(IG,IR,IN)
359:            400      continue
360:          410      continue
361:        420      continue
362: C
363:      do 510 L = 1, NEMIC
364:        do 500 KR = 1, NTREG
365: C
366:          do 430 IG = 1, NGROUP
367:            RRR(IG,KR) = 0.0D0
368:          430      continue
369:          do 450 MR = IPTRG(KR), IPTRG(KR+1) - 1
370:            IR = LPTRG(MR)
371:            do 440 IG = 1, NGROUP
372:              RRR(IG,KR) = RRR(IG,KR) + RMIC(IG,IR,IN,L)
373:            440      continue
374:          450      continue
375: C
376: C
377: C DTOT = 0.0D0
378: C if ( JNEUT.ne.0 ) then
379: C   do 460 IG = 1, NGP1
380: C     DTOT = DTOT + RRR(IG,KR)
381: C   460      continue
382: C   end if
383: C   DTOTP = 0.0D0
384: C   if ( JPHOT.ne.0 ) then
385: C     do 470 IG = NGP1 + 1, NGP1 + NGP2
386: C       DTOTP = DTOTP + RRR(IG,KR)
387: C     470      continue
388: C   end if
389: C
390:      do 460 IG = 1, NGROUP

```

src/mvp/talsum.f

```

391:      SRMIC(IG,KR,IN,L,1) = SRMIC(IG,KR,IN,L,1)
392:      &      + RRR(IG,KR)
393:      SRMIC(IG,KR,IN,L,2) = SRMIC(IG,KR,IN,L,2)
394:      &      + (RRR(IG,KR)**2) /DNG
395:      if ( RRR(IG,KR).ne.0.0D0 ) then
396:      if ( JNWMX.eq.0 ) then
397:      DX      = RRR(IG,KR) /DNF(IG,KR)
398:      else
399:      if ( JRTTR.ne.0 ) then
400:      DX      = RRR(IG,KR)/FTR(IG,KR)
401:      else
402:      DX      = RRR(IG,KR)/FCL(IG,KR)
403:      end if
404:      end if
405:      XMIC(IG,KR,IN,L,1) = XMIC(IG,KR,IN,L,1)
406:      &      + DX*DNG
407:      XMIC(IG,KR,IN,L,2) = XMIC(IG,KR,IN,L,2)
408:      &      + DX*DX*DNG
409:      end if
410: 460      continue
411: C
412:      NK      = 0
413:      do 490 IK = 1, KPLIM
414:      if ( JKPAP(IK).ne.0 ) then
415:      DTOT      = 0.0D0
416:      do 470 IG = KNGP(IK), KNGP(IK+1) - 1
417:      DTOT      = DTOT + RRR(IG,KR)
418: 470      continue
419: C
420:      NK      = NK + 1
421:      IKSUM      = NGROUP + NK
422: C
423:      if ( DTOT.ne.0.0 ) then
424:      RMICR(NK,KR,IN,L,1) = RMICR(NK,KR,IN,L,1)
425:      &      + DTOT
426:      RMICR(NK,KR,IN,L,2) = RMICR(NK,KR,IN,L,2)
427:      &      + (DTOT*DTOT) /DNG
428: C
429:      DNN      = 0.0D0
430:      do 480 IG = KNGP(IK), KNGP(IK+1) - 1
431:      if ( JNWMX.eq.0 ) then
432:      DNN      = DNN + DNF(IG,KR)
433:      else
434:      if ( JRTTR.ne.0 ) then
435:      DNN      = DNN + FTR(IG,KR)
436:      else
437:      DNN      = DNN + FCL(IG,KR)
438:      end if
439:      end if
440: 480      continue
441: C
442:      if ( DNN.gt.0.0D0 ) then
443:      DX      = DTOT/DNN
444:      XMIC(IKSUM,KR,IN,L,1) =
445:      &      XMIC(IKSUM,KR,IN,L,1) + DX*DNG
446:      XMIC(IKSUM,KR,IN,L,2) =
447:      &      XMIC(IKSUM,KR,IN,L,2)
448:      &      + DX*DX*DNG
449:      end if
450:      end if
451:      end if
452: 490      continue
453: 500      continue
454: 510      continue
455: 520      continue

```

```

456: C
457:      NK      = 0
458:      do 570 IK = 1, KPLIM
459:      if ( JKPAP(IK).ne.0 ) then
460:      NK      = NK + 1
461:      IKSUM      = NGROUP + NK
462:      if ( JRTTR.ne.0 ) then
463:      do 540 KR = 1, NTREG
464:      FLR(KR,NK) = 0.0D0
465:      do 530 IG = KNGP(IK), KNGP(IK+1) - 1
466:      FLR(KR,NK) = FLR(KR,NK) + FTR(IG,KR)
467: 530      continue
468:      if ( FLR(KR,NK).ne.0.0 ) WCXTY(IKSUM,KR) =
469:      &      WCXTY(IKSUM,KR) + DNG
470: 540      continue
471: C
472:      else
473:      do 560 KR = 1, NTREG
474:      FLR(KR,NK) = 0.0D0
475:      do 550 IG = KNGP(IK), KNGP(IK+1) - 1
476:      FLR(KR,NK) = FLR(KR,NK) + FCL(IG,KR)
477: 550      continue
478:      if ( FLR(KR,NK).ne.0.0 ) WCXTY(IKSUM,KR) =
479:      &      WCXTY(IKSUM,KR) + DNG
480: 560      continue
481:      end if
482:      end if
483: 570      continue
484: C
485:      end if
486: C
487:      if ( JSCTM.ne.0 ) then
488:      do K = 1, KY
489:      do IN = 1, NUC
490:      do KR = 1, NTREG
491:      do IG2 = 1, NGP1
492:      do IG1 = 1, NGP1
493:      DNFSM(IG1,IG2) = 0
494:      do IM = 1, KJSCTM
495:      RRRSM(IG1,IG2,IM) = 0
496:      end do
497:      end do
498:      end do
499:      do MR = IPTRG(KR), IPTRG(KR+1)-1
500:      IR = LPTRG(MR)
501:      do IG2 = 1, NGP1
502:      do IG1 = 1, NGP1
503:      DNFSM(IG1,IG2) = DNFSM(IG1,IG2) +
504:      &      DNFLXSM(IG1,IG2,IR,IN,K)
505:      do IM = 1, KJSCTM
506:      RRRSM(IG1,IG2,IM) = RRRSM(IG1,IG2,IM) +
507:      &      RMICSM(IG1,IG2,IR,IN,IM,K)
508:      end do
509:      end do
510:      end do
511:      do IG2 = 1, NGP1
512:      do IG1 = 1, NGP1
513:      DX = DNFSM(IG1,IG2)
514: C%
515: C%
516: C%
517: C%
518: C%
519:      if ( RRRSM(IG1,IG2,1).gt.DZERO ) then
520:      SRMICSM(IG1,IG2,KR,IN,1,1,K) =

```


src/mvp/talsum.f

```

521:      &          SRMICSM(IG1,IG2,KR,IN,1,1,K) +
522:      &          RRRSM(IG1,IG2,1)
523:      SRMICSM(IG1,IG2,KR,IN,2,1,K) =
524:      &          SRMICSM(IG1,IG2,KR,IN,2,1,K) +
525:      &          (RRRSM(IG1,IG2,1)**2)/DNG
526:      if ( JSCTM.gt.1 ) then
527:      do IM = 2, KJSCTM
528:          RRRSM(IG1,IG2,IM)=RRRSM(IG1,IG2,IM)/
529:      &          RRRSM(IG1,IG2,1)
530:      end do
531:      do IM = 2, KJSCTM
532:      if ( IM.eq.2 ) then
533:          FMOM(1)=RRRSM(IG1,IG2,IM)
534:      else if ( IM.eq.3 ) then
535:          FMOM(2)=(3*RRRSM(IG1,IG2,IM)-1)/2
536:      else if ( IM.eq.4 ) then
537:          FMOM(3)=(5*RRRSM(IG1,IG2,IM)-
538:      &          3*RRRSM(IG1,IG2,IM-2))/2
539:      else if ( IM.eq.5 ) then
540:          FMOM(4)=(35*RRRSM(IG1,IG2,IM)-
541:      &          51*RRRSM(IG1,IG2,IM-2)+3)/8
542:      else if ( IM.eq.6 ) then
543:          FMOM(5)=(315*RRRSM(IG1,IG2,IM)-
544:      &          539*RRRSM(IG1,IG2,IM-2)+
545:      &          75*RRRSM(IG1,IG2,IM-4))/40
546:      else if ( IM.eq.7 ) then
547:          FMOM(6)=(3465*RRRSM(IG1,IG2,IM)-
548:      &          6804*RRRSM(IG1,IG2,IM-2)+
549:      &          2100*RRRSM(IG1,IG2,IM-4)-
550:      &          75)/240
551:      else if ( IM.eq.8 ) then
552:          FMOM(7)=(45045*RRRSM(IG1,IG2,IM)-
553:      &          99792*RRRSM(IG1,IG2,IM-2)+
554:      &          46704*RRRSM(IG1,IG2,IM-4)-
555:      &          3675*RRRSM(IG1,IG2,IM-6))/
556:      &          1680
557:      else
558:          write(IMG,997) KJSCTM
559:          stop 666
560:      end if
561:      end do
562:      do IM = 2, KJSCTM
563:          SRMICSM(IG1,IG2,KR,IN,1,IM,K) =
564:      &          SRMICSM(IG1,IG2,KR,IN,1,IM,K) +
565:      &          FMOM(IM-1)*DNG
566:          SRMICSM(IG1,IG2,KR,IN,2,IM,K) =
567:      &          SRMICSM(IG1,IG2,KR,IN,2,IM,K) +
568:      &          (FMOM(IM-1)**2)*DNG
569:      end do
570:      end if
571:      end if
572:      end do
573:      end do
574:      end do
575:      end do
576:      end do
577:      end if
578:      997 format(' XXX(talsum) Number of high-order moments is exceeded ',
579:      &          'the available limit for microscopic. JSCTM=',i5)
580: C
581:      if ( JSCMU.ne.0 ) then
582:          do K = 1, KY
583:              do IN = 1, NUC
584:                  do KR = 1, NTREG
585:                      do IG2 = 1, NGP1+1

```

```

586:                      do IG1 = 1, NGP1
587:                          DNFSM(IG1,IG2) = 0
588:                          RRRSM(IG1,IG2,1) = 0
589:                      end do
590:                  end do
591:                  do MR = IPTRG(KR), IPTRG(KR+1)-1
592:                      IR = LPTRG(MR)
593:                      do IG2 = 1, NGP1+1
594:                          do IG1 = 1, NGP1
595:                              DNFSM(IG1,IG2) = DNFSM(IG1,IG2) +
596:      &                              WMIMU(IG1,IG2,IR,IN,K)
597:                              RRRSM(IG1,IG2,1) = RRRSM(IG1,IG2,1) +
598:      &                              RMIMU(IG1,IG2,IR,IN,K)
599:                          end do
600:                      end do
601:                  end do
602:                  do IG2 = 1, NGP1+1
603:                      do IG1 = 1, NGP1
604:                          if ( DNFSM(IG1,IG2).gt.DZERO ) then
605:                              WMU = RRRSM(IG1,IG2,1)
606:                              SMIMU(IG1,IG2,KR,IN,1,K) =
607:      &                              SMIMU(IG1,IG2,KR,IN,1,K) + WMU
608:                              SMIMU(IG1,IG2,KR,IN,2,K) =
609:      &                              SMIMU(IG1,IG2,KR,IN,2,K) + (WMU**2)/
610:      1                              DNFSM(IG1,IG2)
611:                              SMIMU(IG1,IG2,KR,IN,3,K) =
612:      &                              SMIMU(IG1,IG2,KR,IN,3,K)+DNFSM(IG1,IG2)
613:                          end if
614:                      end do
615:                  end do
616:                  end do
617:                  end do
618:                  end do
619:                  end if
620: C
621: C -----
622: C MACROSCOPIC REACTION RATE & CROSS SECTION
623: C -----
624: C
625:                  if ( NEMAC.ne.0 ) then
626:                      do 680 M = 1, NEMAC
627:                          JR = LEMAC(M,2)
628: C                          JCX = JMACE(JR)
629:                          JMIC = LEMIC(JR,1)
630:                          do 590 KR = 1, NTREG
631:                              do 580 IG = 1, NGROUP
632:                                  DMAC(IG,KR) = 0.0D0
633:                              continue
634:                          continue
635: C
636:                      do 630 IN = 1, NUC
637:                          do 620 KR = 1, NTREG
638:                              do 610 MR = IPTRG(KR), IPTRG(KR+1) - 1
639:                                  IR = LPTRG(MR)
640:                                  do 600 IG = 1, NGROUP
641:                                      DMAC(IG,KR) = DMAC(IG,KR)
642:      &                                      + RMIC(IG,IR,IN,JMIC)
643:                                  continue
644:                              continue
645:                          continue
646:                      continue
647: C
648:                      do 670 KR = 1, NTREG
649: C
650: CC                      DTOT = 0.0D0

```

src/mvp/talsum.f

```

651: CC      if ( JNEUT.ne.0 ) then
652: CC          do 680 IG = 1, NGP1
653: CC              DTOT = DTOT + DMAC(IG,KR)
654: CC680      continue
655: CC      end if
656: CC          DTOTP = 0.0D0
657: CC      if ( JPHOT.ne.0 ) then
658: CC          do 690 IG = NGP1 + 1, NGP1 + NGP2
659: CC              DTOTP = DTOTP + DMAC(IG,KR)
660: CC690      continue
661: CC      end if
662: C
663:      do 640 IG = 1, NGROUP
664:          RMAC(IG,KR,M,1) = RMAC(IG,KR,M,1) + DMAC(IG,KR)
665:          RMAC(IG,KR,M,2) = RMAC(IG,KR,M,2) +
666:      &          (DMAC(IG,KR)**2) /DNG
667:      if ( JRTIR.ne.0 ) then
668:          if ( FTR(IG,KR).ne.0.0D0 ) then
669:              DX = DMAC(IG,KR) /FTR(IG,KR)
670:              XMAC(IG,KR,M,1) = XMAC(IG,KR,M,1) + DX*DNG
671:              XMAC(IG,KR,M,2) = XMAC(IG,KR,M,2) + DX*DX*DNG
672:          end if
673:      end if
674: C
675:      if ( JRTCL.ne.0 ) then
676:          if ( FCL(IG,KR).ne.0.0D0 ) then
677:              DX = DMAC(IG,KR) /FCL(IG,KR)
678:              XMAC(IG,KR,M,1) = XMAC(IG,KR,M,1) + DX*DNG
679:              XMAC(IG,KR,M,2) = XMAC(IG,KR,M,2) + DX*DX*DNG
680:          end if
681:      end if
682:      640      continue
683: C
684:      NK = 0
685:      do 660 IK = 1, KPLIM
686:          if ( JKPAP(IK).ne.0 ) then
687:              NK = NK + 1
688:              IKSUM = NGROUP + NK
689:              DTOT = 0.0D0
690:              do 650 IG = KNGP(IK), KNGP(IK+1) - 1
691:                  DTOT = DTOT + DMAC(IG,KR)
692:      650      continue
693:              if ( DTOT.ne.0.0 ) then
694:                  RMACR(NK,KR,M,1) = RMACR(NK,KR,M,1) + DTOT
695:                  RMACR(NK,KR,M,2) = RMACR(NK,KR,M,2)
696:      &                  + (DTOT*DTOT) /DNG
697: C
698:                  if ( FLR(KR,NK).ne.0.0D0 ) then
699:                      DX = DTOT/FLR(KR,NK)
700:                      XMAC(IKSUM,KR,M,1) = XMAC(IKSUM,KR,M,1)
701:      &                      + DX*DNG
702:                      XMAC(IKSUM,KR,M,2) = XMAC(IKSUM,KR,M,2)
703:      &                      + DX*DX*DNG
704:                  end if
705:              end if
706:          end if
707:      660      continue
708:      670      continue
709:      680      continue
710: C
711:      end if
712: C
713:      if ( JSCTM.ne.0 ) then
714:          KYM = 0
715:          if ( JMACE(4).gt.0 ) KYM = KYM + 1

```

```

716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765: 685
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:

```

```

if ( JMACE(6).gt.0 ) KYM = KYM + 1
if ( JMACE(7).gt.0 ) KYM = KYM + 1
do K = 1, KYM
    do KR = 1, NTREG
        do IG2 = 1, NGP1
            do IG1 = 1, NGP1
                DNFSM(IG1,IG2) = 0
                do IM = 1, KJSCTM
                    RRRSM(IG1,IG2,IM) = 0
                end do
            end do
        end do
    end do
do MR = IPTRG(KR), IPTRG(KR+1)-1
    IR = LPTRG(MR)
    if ( JTLLT.eq.0 ) then
        do I = 1, NZONE
            if ( IR.eq.KZREG(I) ) go to 685
        end do
        write(IOW,991) IR, NZONE, (KZREG(I),I=1,NZONE)
        stop 666
    else
        do M = 1, NSPACE
            do I = 1, NINPZ
                if ( .not.ISLATT(KMAT(I)) ) then
                    ISU = KREG(I)
                    if ( ISU.gt.0.and.ISUSP(ISU,M).ne.0 )
                        then
                            if ( IR.eq.ISUSP(ISU,M) ) go to 685
                        end if
                    end if
                end if
            end do
        end do
        M = min(NZONE,NINPZ)
        do I = 1, M
            if ( KCELL(I).eq.0 ) then
                if ( KMAT(I).ge.0 ) then
                    if ( IR.eq.KREG(I) ) go to 685
                end if
            end if
        end do
        write(IOW,992) IR, NZONE, (KZREG(I),I=1,NZONE)
        write(IOW,993) (KCELL(I),I=1,NZONE)
        write(IOW,995) NINPZ, (KMAT(I),I=1,NINPZ)
        write(IOW,996) NINPZ, (KREG(I),I=1,NINPZ)
        do M = 1, NSPACE
            write(IOW,994) M, (ISUSP(I,M),I=1,NSUZON)
        enddo
        stop 666
    end if
continue
do IN = 1, NUC
    DN = DNZON(IN,I,1)
    if ( DN.gt.DZERO ) then
        do IG2 = 1, NGP1
            do IG1 = 1, NGP1
                DNFSM(IG1,IG2) = DNFSM(IG1,IG2) +
                &                DNFLXSM(IG1,IG2,IR,IN,K)/DN
            do IM = 1, KJSCTM
                RRRSM(IG1,IG2,IM) =
                &                RRRSM(IG1,IG2,IM) +
                &                RMICSM(IG1,IG2,IR,IN,IM,K)
            end do
        end do
    end do
end do
end if

```

src/mvp/talsum.f

```

781:         end do
782:     end do
783:     do IG2 = 1, NGP1
784:         do IG1 = 1, NGP1
785:             c%c          DX = DNFSM(IG1,IG2)
786:             c%c          XMACSM(IG1,IG2,KR,1,K) =
787:             c%c          XMACSM(IG1,IG2,KR,1,K) + DX
788:             c%c          XMACSM(IG1,IG2,KR,2,K) =
789:             c%c          XMACSM(IG1,IG2,KR,2,K) + DX*DX/DNG
790:             if ( RRRSM(IG1,IG2,1).gt.DZERO ) then
791:                 RMACSM(IG1,IG2,KR,1,1,K) =
792:                 RMACSM(IG1,IG2,KR,1,1,K)+RRRSM(IG1,IG2,1)
793:                 RMACSM(IG1,IG2,KR,2,1,K) =
794:                 RMACSM(IG1,IG2,KR,2,1,K) +
795:                 (RRRSM(IG1,IG2,1)**2)/DNG
796:                 if ( JSCM.gt.1 ) then
797:                     do IM = 2, KJSCTM
798:                         RRRSM(IG1,IG2,IM) = RRRSM(IG1,IG2,IM) /
799:                         RRRSM(IG1,IG2,1)
800:                     end do
801:                     do IM = 2, KJSCTM
802:                         if ( IM.eq.2 ) then
803:                             FMOM(1)=RRRSM(IG1,IG2,IM)
804:                         else if ( IM.eq.3 ) then
805:                             FMOM(2)=(3*RRRSM(IG1,IG2,IM)-1)/2
806:                         else if ( IM.eq.4 ) then
807:                             FMOM(3)=(5*RRRSM(IG1,IG2,IM)-
808:                             3*RRRSM(IG1,IG2,IM-2))/2
809:                         else if ( IM.eq.5 ) then
810:                             FMOM(4)=(35*RRRSM(IG1,IG2,IM)-
811:                             51*RRRSM(IG1,IG2,IM-2)+3)/8
812:                         else if ( IM.eq.6 ) then
813:                             FMOM(5)=(315*RRRSM(IG1,IG2,IM)-
814:                             539*RRRSM(IG1,IG2,IM-2)+
815:                             75*RRRSM(IG1,IG2,IM-4))/40
816:                         else if ( IM.eq.7 ) then
817:                             FMOM(6)=(3465*RRRSM(IG1,IG2,IM)-
818:                             6804*RRRSM(IG1,IG2,IM-2)+
819:                             2100*RRRSM(IG1,IG2,IM-4)-
820:                             75)/240
821:                         else if ( IM.eq.8 ) then
822:                             FMOM(7)=(45045*RRRSM(IG1,IG2,IM)-
823:                             99792*RRRSM(IG1,IG2,IM-2)+
824:                             46704*RRRSM(IG1,IG2,IM-4)-
825:                             3675*RRRSM(IG1,IG2,IM-6))/
826:                             1680
827:                         else
828:                             write(IMG,998) KJSCTM
829:                             stop 666
830:                         end if
831:                     end do
832:                     do IM = 2, KJSCTM
833:                         RMACSM(IG1,IG2,KR,1,IM,K) =
834:                         RMACSM(IG1,IG2,KR,1,IM,K)+
835:                         FMOM(IM-1)*DNG
836:                         RMACSM(IG1,IG2,KR,2,IM,K) =
837:                         RMACSM(IG1,IG2,KR,2,IM,K)+
838:                         (FMOM(IM-1)**2)*DNG
839:                     end do
840:                 end if
841:             end if
842:         end do
843:     end do
844: end do
845:
846:         end if
847:     991 format('// XXX(talsum) Region number is not found from zone',
848:         & ' number.'
849:         &/'          region='i5,' nzone='i4
850:         &/'          list of region number in KZREG'
851:         &/(13x,10i8))
852:     992 format('// XXX(talsum) Region number is not found from zone',
853:         & ' number for tally-lattice.'
854:         &/'          region='i5,' nzone='i4
855:         &/'          list of region number in KZREG'
856:         &/(12x,15i7))
857:     993 format('          list of cell number in KCELL'
858:         &/(12x,15i7))
859:     994 format('          list of lattice-region number in ISUSP for ',
860:         & 'cell-space ',i2
861:         &/(12x,20i6))
862:     995 format('          list of material in KMAT ... ninpz='i9
863:         &/(12x,15i7))
864:     996 format('          list of region number in KREG ... ninpz='i9
865:         &/(12x,15i7))
866:     998 format('// XXX(talsum) Number of high-order moments is exceeded ',
867:         & 'the available limit for macroscopic. JSCTM='i5)
868: C
869:         if ( JSCMU.ne.0 ) then
870:             KYM = 0
871:             if ( JMACE(4).gt.0 ) KYM = KYM + 1
872:             if ( JMACE(6).gt.0 ) KYM = KYM + 1
873:             if ( JMACE(7).gt.0 ) KYM = KYM + 1
874:             do K = 1, KYM
875:                 do KR = 1, NTREG
876:                     do IG2 = 1, NGP1+1
877:                         do IG1 = 1, NGP1
878:                             DNFSM(IG1,IG2) = 0
879:                             RRRSM(IG1,IG2,1) = 0
880:                         end do
881:                     end do
882:                     do MR = IPTRG(KR), IPTRG(KR+1)-1
883:                         IR = LPTRG(MR)
884:                         if ( JTLLT.eq.0 ) then
885:                             do I = 1, NZONE
886:                                 if ( IR.eq.KZREG(I) ) go to 686
887:                             end do
888:                             write(IOW,991) IR, NZONE, (KZREG(I),I=1,NZONE)
889:                             stop 666
890:                         else
891:                             do M = 1, NSPACE
892:                                 do I = 1, NINPZ
893:                                     if ( .not.ISLATT(KMAT(I)) ) then
894:                                         ISU = KREG(I)
895:                                         if ( ISU.gt.0.and.ISUSP(ISU,M).ne.0 )
896:                                             &
897:                                             then
898:                                                 if ( IR.eq.ISUSP(ISU,M) ) go to 686
899:                                             end if
900:                                         end if
901:                                     end do
902:                                     M = min(NZONE,NINPZ)
903:                                     do I = 1, M
904:                                         if ( KCELL(I).eq.0 ) then
905:                                             if ( KMAT(I).ge.0 ) then
906:                                                 if ( IR.eq.KREG(I) ) go to 686
907:                                             end if
908:                                         end if
909:                                     end do
910:                                     write(IOW,992) IR, NZONE, (KZREG(I),I=1,NZONE)

```

src/mvp/talsum.f

```

911:      write(IOW,993) (KCELL(I),I=1,NZONE)
912:      write(IOW,995) NINPZ, (KMAT(I),I=1,NINPZ)
913:      write(IOW,996) NINPZ, (KREG(I),I=1,NINPZ)
914:      do M = 1, NSPACE
915:        write(IOW,994) M, (ISUSP(I,M),I=1,NSUZON)
916:      enddo
917:      stop 666
918:    end if
919:  686  continue
920:    do IN = 1, NUC
921:      DN = DNZON(IN,I,1)
922:      if ( DN.gt.DZERO ) then
923:        do IG2 = 1, NGP1+1
924:          do IG1 = 1, NGP1
925:            DNFSM(IG1,IG2) = DNFSM(IG1,IG2) +
926:              & WMIMU(IG1,IG2,IR,IN,K)
927:            RRRSM(IG1,IG2,1) = RRRSM(IG1,IG2,1) +
928:              & RMIMU(IG1,IG2,IR,IN,K)
929:          end do
930:        end do
931:      end if
932:    end do
933:  end do
934:    do IG2 = 1, NGP1+1
935:      do IG1 = 1, NGP1
936:        if ( DNFSM(IG1,IG2).gt.DZERO ) then
937:          WMU = RRRSM(IG1,IG2,1)
938:          RMAMU(IG1,IG2,KR,1,K) =
939:            & RMAMU(IG1,IG2,KR,1,K) + WMU
940:          RMAMU(IG1,IG2,KR,2,K) =
941:            & RMAMU(IG1,IG2,KR,2,K) + (WMU**2)/
942:            & DNFSM(IG1,IG2)
943:          RMAMU(IG1,IG2,KR,3,K) =
944:            & RMAMU(IG1,IG2,KR,3,K) + DNFSM(IG1,IG2)
945:        end if
946:      end do
947:    end do
948:  end do
949: end do
950: end if
951: C
952: C
953: C -----
954: C REACTION RATE USING RESPONSE FUNCTION
955: C -----
956: if ( JRESP.ne.0.and.NRESP.gt.0 ) then
957:   do 710 N = 1, NRESP
958:     do 700 KR = 1, NTREG
959:       RTR(KR,N) = 0.0D0
960:       RCL(KR,N) = 0.0D0
961:       do 690 IG = 1, NGROUP
962:         RTR(KR,N) = RTR(KR,N) + RESP(IG,N)*FTR(IG,KR)
963:         RCL(KR,N) = RCL(KR,N) + RESP(IG,N)*FCL(IG,KR)
964:       690 continue
965:     700 continue
966:   710 continue
967: C
968:   do 730 N = 1, NRESP
969:     do 720 KR = 1, NTREG
970:       SRETR(KR,N,1) = SRETR(KR,N,1) + RTR(KR,N)
971:       SRECL(KR,N,1) = SRECL(KR,N,1) + RCL(KR,N)
972:       SRETR(KR,N,2) = SRETR(KR,N,2) + (RTR(KR,N)**2) /DNG
973:       SRECL(KR,N,2) = SRECL(KR,N,2) + (RCL(KR,N)**2) /DNG
974:     720 continue
975:   730 continue

```

```

976: C
977: CM      DO 420 N=1,NRESP
978: CM      DO 420 IR=1,NREG
979: CM          RETR(IR,N) = 0.0D0
980: CM          RECL(IR,N) = 0.0D0
981: CM420    CONTINUE
982: C      end if
983: C
984: C -----
985: C REACTION RATE USING CROSS SECTION IN SGTAL
986: C -----
987: C
988:   if ( JRESP.ne.0.and.NSTAL.gt.0 ) then
989:     do 760 N = 1, NSTAL
990:       do 750 KR = 1, NTREG
991:         STR(KR,N) = 0.0D0
992:         SCL(KR,N) = 0.0D0
993:         do 740 MR = IPTRG(KR), IPTRG(KR+1) - 1
994:           IR = LPTRG(MR)
995:           STR(KR,N) = STR(KR,N) + RSTR(IR,N)
996:           SCL(KR,N) = SCL(KR,N) + RSCL(IR,N)
997:         740 continue
998:       750 continue
999:     760 continue
1000: C
1001:   do 780 N = 1, NSTAL
1002:     do 770 KR = 1, NTREG
1003:       SRSTR(KR,N,1) = SRSTR(KR,N,1) + STR(KR,N)
1004:       SRSCL(KR,N,1) = SRSCL(KR,N,1) + SCL(KR,N)
1005:       SRSTR(KR,N,2) = SRSTR(KR,N,2) + (STR(KR,N)**2) /DNG
1006:       SRSCL(KR,N,2) = SRSCL(KR,N,2) + (SCL(KR,N)**2) /DNG
1007:     770 continue
1008:   780 continue
1009: C
1010:   do 800 N = 1, NSTAL
1011:     do 790 IR = 1, NREG
1012:       RSTR(IR,N) = 0.0D0
1013:       RSCL(IR,N) = 0.0D0
1014:     790 continue
1015:   800 continue
1016:   end if
1017: C
1018: C -----
1019: C TAKE SPECIAL TALLIES
1020: C -----
1021: C
1022:   if ( NETALY.gt.0 ) then
1023:     IBSUM = NBATCH - NSKIP
1024:     call STLSUM( IOW, IETAL, NETALY, IDTAL, ETALY, NLETAL,
1025:       & DTALY, NGENEO, WSUMB, IBSUM, NETRV, METRV, IETRV,
1026:       & ETRV, DWRK )
1027:   end if
1028: C
1029: C -----
1030: C weighted flight time
1031: C -----
1032: C
1033:   if ( JTIME.ne.0 ) then
1034: CCCCCC   do 850 KKK = 1, NPKIND
1035: C          do 810 KKK = 1, KPLIM
1036: C            TFLHS(KKK,1) = TFLHS(KKK,1) + TFLH(KKK)
1037: C            TFLHS(KKK,2) = TFLHS(KKK,2) + TFLH(KKK)**2/DNG
1038: C            TFLH(KKK) = 0.0D0
1039: C          810 continue
1040: C        end if

```

src/mvp/talsum.f

```

1041:
1042: C
1043: C -----
1044: C CLEAR ARRAYS FOR NEXT BATCH
1045: C -----
1046: C
1047:       do 850 L = 1, NEMIC
1048:         do 840 IN = 1, NUC
1049:           do 830 IR = 1, NREG
1050:             do 820 IG = 1, NGROUP
1051:               RMIC(IG,IR,IN,L) = 0.0D0
1052:             820 continue
1053:           830 continue
1054:         840 continue
1055:       850 continue
1056:       do 870 IR = 1, NREG
1057:         do 860 IG = 1, NGROUP
1058:           FLTR(IG,IR) = 0.0D0
1059:           FLCL(IG,IR) = 0.0D0
1060:         860 continue
1061:       870 continue
1062:       if ( NEMIC.gt.0 ) then
1063:         do 900 IN = 1, NUC
1064:           do 890 IR = 1, NREG
1065:             do 880 IG = 1, NGROUP
1066:               DNFLX(IG,IR,IN) = 0.0D0
1067:             880 continue
1068:           890 continue
1069:         900 continue
1070:       end if
1071: C
1072:       if ( JSCTM.ne.0 ) then
1073:         do K = 1, KY
1074:           do IN = 1, NUC
1075:             do IR = 1, NREG
1076:               do IG2 = 1, NGP1
1077:                 do IG1 = 1, NGP1
1078:                   DNFLXSM(IG1,IG2,IR,IN,K) = 0
1079:                   do IM = 1, KJSCTM
1080:                     RMICSM(IG1,IG2,IR,IN,IM,K) = 0
1081:                   end do
1082:                 end do
1083:               end do
1084:             end do
1085:           end do
1086:         end do
1087:       end if
1088:       if ( JSCMU.ne.0 ) then
1089:         do K = 1, KY
1090:           do IN = 1, NUC
1091:             do IR = 1, NREG
1092:               do IG2 = 1, NGP1+1
1093:                 do IG1 = 1, NGP1
1094:                   WMIMU(IG1,IG2,IR,IN,K) = 0
1095:                   RMIMU(IG1,IG2,IR,IN,K) = 0
1096:                 end do
1097:               end do
1098:             end do
1099:           end do
1100:         end do
1101:       end if
1102: C
1103:       if ( NDTALY.gt.0 ) then
1104:         do 910 I = 1, NLD TAL
1105:           DTALY(I) = 0.0D0

```

```

1106:       910 continue
1107:       end if
1108:     end if
1109: C
1110: C -----
1111: C TAKE TALLIES FOR EIGENVALUE
1112: C -----
1113: C
1114:       if ( JEIGN.ne.0 ) then
1115:         if ( JPRTS(7).ne.1 ) then
1116: C/#IF PARA(SX* CRAY)
1117: * call MVPSYNC_LOCK(2)
1118: C/#ENDIF
1119:         if ( JBPRNT.ne.0 ) then
1120:           write(IOW,7000) NCNTR(21,KPNEUT), NCNTR(2,KPNEUT), NFISB
1121: C
1122: C .... EIGENVALUE ESTIMATOR BY BALANCE
1123: C
1124: C PRODUCTION / ( LEAK + LOSS )
1125: C
1126: C LOSS : CAPTURE + FISSION - (N,2N) - 2*(N,3N) - 3*(N,4N)
1127: C
1128:           XKEF4 = WCNTR(15,KPNEUT) /
1129:             & (WCNTR(7,KPNEUT)+WCNTR(20,KPNEUT))
1130:           XKEF5 = WCNTR(14,KPNEUT) /
1131:             & (WCNTR(7,KPNEUT)+WCNTR(19,KPNEUT))
1132:           XKEF6 = WCNTR(21,KPNEUT) /
1133:             & (WCNTR(7,KPNEUT)+WCNTR(22,KPNEUT))
1134: C
1135:           write(IOW,7020) WCNTR(15,KPNEUT) /XSOC(NBATCH),
1136:             & WCNTR(14,KPNEUT) /XSOC(NBATCH), WCNTR(21,KPNEUT)
1137:             & /XSOC(NBATCH), XKEF4, XKEF5, XKEF6
1138: C##<2007/03/14:PN4:
1139:           if ( JPHNU.ne.0 ) then
1140:             write(IOW,7030) WCNTR(31,KPNEUT)/XSOC(NBATCH),
1141:               & WCNTR(32,KPNEUT)/XSOC(NBATCH),
1142:               & WCNTR(33,KPNEUT)/XSOC(NBATCH)
1143:           end if
1144: C##>
1145: C+beff1
1146: C
1147: C if( JBEFF.ne.0 ) then
1148: C   XBEF1 = WCBEF(4) / WCNTR(15,KPNEUT)
1149: C   XBEF2 = WCBEF(5) / WCNTR(14,KPNEUT)
1150: C   write(IOW,'(1x,' BEFF = ',1p,e10.2,'(TRK)',
1151: C     & e10.2,0p,'(COL)')') XBEF1, XBEF2
1152: C   end if
1153: C   write(6,*) ' nagaya-talsum: WCBEF(4)=',WCBEF(4)
1154: C   write(6,*) ' nagaya-talsum: WCBEF(7)=',WCBEF(7)
1155: C   write(6,*) ' nagaya-talsum: WCNTR(15)=',WCNTR(15,KPNEUT)
1156: C   write(6,*) ' nagaya-talsum: WCBEF(10)=',WCBEF(10)
1157: C   write(6,*) ' nagaya-talsum: WCBEF(5)=',WCBEF(5)
1158: C   write(6,*) ' nagaya-talsum: WCBEF(8)=',WCBEF(8)
1159: C   write(6,*) ' nagaya-talsum: WCBEF(6)=',WCBEF(6)
1160: C   write(6,*) ' nagaya-talsum: WCBEF(9)=',WCBEF(9)
1161: C   c-beff1
1162: C   end if
1163: C/#IF PARA(SX* CRAY)
1164: * call MVPSYNC_UNLOCK(2)
1165: C/#ENDIF
1166: C
1167:       7000 format(1X,' FISSION POINT =',F11.0,' FISSION NEUTRONS =',F11.0,
1168:         & ' SELECTED FOR NEXT BATCH =',I7)
1169:       7020 format(1X,' KEFF( PRODUCTION ) =',F6.3,'(TRK) ',F6.3,'(COL) ',
1170:         & F6.3,'(ANL) ', ' KEFF(BALANCE) =',F6.3,'(TRK) ',F6.3,

```

src/mvp/talsum.f

```

1171:      &      '(COL) ',F6.3,'(ANL)')
1172: c##<2007/03/14:PN4:
1173: 7030 format(1X,'      KEFF(PHOTONEUTRON) =',1P,E10.3,'(TRK) ',E10.3,
1174:      & '(COL) ',E10.3,'(ANL)')
1175: c##>
1176: C
1177:      if ( NBATCH.ge.2 ) then
1178:          XKEF(1,NBATCH) = WCNTR(15,KPNEUT) + XKEF(1,NBATCH-1)
1179:          XKEF(2,NBATCH) = WCNTR(14,KPNEUT) + XKEF(2,NBATCH-1)
1180:          XKEF(3,NBATCH) = WCNTR(21,KPNEUT) + XKEF(3,NBATCH-1)
1181:          XKEF(4,NBATCH) = WCNTR(20,KPNEUT) + XKEF(4,NBATCH-1)
1182:          XKEF(5,NBATCH) = WCNTR(19,KPNEUT) + XKEF(5,NBATCH-1)
1183:          XKEF(6,NBATCH) = WCNTR(22,KPNEUT) + XKEF(6,NBATCH-1)
1184:          XKEF(7,NBATCH) = WCNTR(7,KPNEUT) + XKEF(7,NBATCH-1)
1185:          XSOC(NBATCH) = XSOC(NBATCH) + XSOC(NBATCH-1)
1186: c##<2007/03/14:PN4:
1187:      if ( JPHNU.ne.0 ) then
1188:          XKEF(8,NBATCH) = WCNTR(31,KPNEUT) + XKEF(8,NBATCH-1)
1189:          XKEF(9,NBATCH) = WCNTR(32,KPNEUT) + XKEF(9,NBATCH-1)
1190:          XKEF(10,NBATCH) = WCNTR(33,KPNEUT) + XKEF(10,NBATCH-1)
1191:      end if
1192: c##>
1193:      else
1194:          XKEF(1,1) = WCNTR(15,KPNEUT)
1195:          XKEF(2,1) = WCNTR(14,KPNEUT)
1196:          XKEF(3,1) = WCNTR(21,KPNEUT)
1197:          XKEF(4,1) = WCNTR(20,KPNEUT)
1198:          XKEF(5,1) = WCNTR(19,KPNEUT)
1199:          XKEF(6,1) = WCNTR(22,KPNEUT)
1200:          XKEF(7,1) = WCNTR(7,KPNEUT)
1201: c##<2007/03/14:PN4:
1202:      if ( JPHNU.ne.0 ) then
1203:          XKEF(8,1) = WCNTR(31,KPNEUT)
1204:          XKEF(9,1) = WCNTR(32,KPNEUT)
1205:          XKEF(10,1) = WCNTR(33,KPNEUT)
1206:      end if
1207: c##>
1208:      end if
1209: c+beff1
1210:      if( JBEFF.ne.0 ) then
1211:          if( NBATCH.ge.2 ) then
1212:              do IEB = 1, NEBEF
1213:                  XBEF(IEB,NBATCH) = WCBEF(IEB) + XBEF(IEB,NBATCH-1)
1214:              end do
1215:          else
1216:              do IEB = 1, NEBEF
1217:                  XBEF(IEB,1) = WCBEF(IEB)
1218:              end do
1219:          end if
1220:      end if
1221: c-beff1
1222: C
1223:      WLEK = WLEK + WCNTR(7,KPNEUT)
1224:      NCNTR(2,KPNEUT) = 0D0
1225:      NCNTR(14,KPNEUT) = 0D0
1226:      NCNTR(15,KPNEUT) = 0D0
1227:      NCNTR(19,KPNEUT) = 0D0
1228:      NCNTR(20,KPNEUT) = 0D0
1229:      NCNTR(21,KPNEUT) = 0D0
1230:      NCNTR(22,KPNEUT) = 0D0
1231:      WCNTR(2,KPNEUT) = 0.D0
1232:      WCNTR(14,KPNEUT) = 0.D0
1233:      WCNTR(15,KPNEUT) = 0.D0
1234:      WCNTR(19,KPNEUT) = 0.D0
1235:      WCNTR(20,KPNEUT) = 0.D0

```

```

1236:      WCNTR(21,KPNEUT) = 0.D0
1237:      WCNTR(22,KPNEUT) = 0.D0
1238:      WCNTR(7,KPNEUT) = 0.D0
1239: c##<2007/03/14:PN4:
1240:      if ( JPHNU.ne.0 ) then
1241:          WCNTR(31,KPNEUT) = 0.d0
1242:          WCNTR(32,KPNEUT) = 0.d0
1243:          WCNTR(33,KPNEUT) = 0.d0
1244:      end if
1245: c##>
1246: c+beff1
1247:      if ( JBEFF.ne.0 ) then
1248:          do 920 IEB = 1, NEBEF
1249:              WCBEF(IEB) = 0.d0
1250:          920 continue
1251:      end if
1252: c-beff1
1253: C
1254: C ... fixed source ...
1255: C
1256:      else
1257:          WLEK = WCNTR(7,KPNEUT)
1258:      end if
1259: C
1260: C ... tally monitoring output (k-eff)
1261: C
1262:      if ( NTASK.eq.1.and.JEIGN.ne.0.and.MOD(NBATCH,NTMINT).eq.0 ) then
1263:          IPBT = NBATCH
1264:          JPR = 1
1265:          call KEFF0( NBATCH, IPBT, NSKIP, JPR, XSOC, XKEF, XKAV1(1),
1266:              &          XKER1(1), XKAV2(1), XKER2(1), XKAV3(1), XKER3(1),
1267:              c##<2007/03/14:PN4:
1268:              &          XKEF1(1,1), XSOCB(1), XKB(1,1) )
1269:              &          XKEF1(1,1), XSOCB(1), XKB(1,1), JPHNU )
1270:          c##>
1271:      end if
1272: C
1273: C-----
1274: C JBPRNT is set to zero if next batch # (-NSKIP) is not a multiple of
1275: C NBPINT.
1276: C If JBPRNT is equal to zero, batch monitor of the next batch is not
1277: C printed.
1278: C-----
1279:      if ( NBATCH.ge.NSKIP.and.NBPINT.gt.1 ) then
1280:          JBPRNT = 1
1281:          if ( MOD(NBATCH+1-NSKIP,NBPINT).ne.1 ) JBPRNT = 0
1282:      end if
1283: C
1284:      return
1285:      end

```

src/mvp/thsc.f

```

1:      subroutine THSCT( IOW,  IBP,  ICLN,  CTH,  JMNTR, JDEBG, JVMNT,
2:      &                  NBANK, NUC,  NNK,   NMT,  NSMIC, NCOLT, ETOP,
3:      &                  EBOT,  ETHMAX, ICX,  CX,   MCX,  KLIB1, XLIB1,
4:      &                  KLIB2, KLB2S, XLIB2, SMIC,  LMIC, KSPI,  LSDED,
5:      &                  NDEAD, WWW,  EEE,   NCNTR, WCNTR, IRAND,
6:      &                  JSCTM, JRTCL, JTLLT, JMICE, JMACE, ENGYB, NGP1,
7:      &                  NGROUP, NZONE, NREG, IGG,  IZZ,  DNZON, KZREG,
8:      &                  IBREG, SMAC, MB,  NSMAC, LMAC, MMAC, KYPOS,
9:      &                  DNFLXSM, RMICSM, JSCMU, RMIMU, WMIMU,
10:     W                  IWK6,  IWK2,  IWK8,  IWK9,  IWK10, IWK11, IWK12,
11:     W                  WK2,   WK3,   WK4,   WK5,   R )
12: C=====
13: C PURPOSE: ANALYSIS OF THERMAL SCATTERING (SAB AND THERMAL ELASTIC)
14: C IBP(I) : STACK, ICLN(I) : COLLIDING NUCLIDE,
15: C NCOLT : NUMBER OF PARTICLES
16: C -----> CTH(I) : SCATTERING ANGLE COSINE IN LAB SYSTEM,
17: C=====
18: C CALLED IN: NEUTR
19: C CALL : RANU2,BSINC3
20: C
21: C=====
22: C
23: C === BANK DATA TO BE UPDATED ===
24: C
25: C EEE : ENERGY
26: C
27: C=====
28: C implicit real*8(A-H,O-Z)
29: C
30: C integer JDEBG(*)
31: C
32: C ..... flag options
33: C integer JMICE(*), JMACE(*)
34: C
35: C ..... geometry data
36: C integer KZREG(NZONE)
37: C
38: C**** INPUT
39: C integer IBP(NCOLT), ICLN(NCOLT)
40: C
41: C**** OUTPUT
42: C real CTH(NCOLT)
43: C
44: C**** CROSS SECTION DATA IN CX ARRAY
45: C real ETOP, EBOT, ETHMAX
46: c##<2007/03/14:PN3:
47: c## real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,2),
48: c## real CX(MCX), XLIB1(NUC,11), XLIB2(NUC,NMT,4),
49: c##>
50: & SMIC(NBANK,NUC,NSMIC)
51: integer LMIC(8)
52: c##<2007/03/14:PN3:
53: c## integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,13),
54: c## integer ICX(MCX), KLIB1(NUC,31), KLIB2(NUC,NMT,21),
55: c##>
56: & KSPI(NBANK,NUC)
57: real SMAC(NBANK,MB,NSMAC)
58: integer MMAC(NBANK,2), LMAC(8)
59: C
60: C**** STACK
61: C integer LSDED(NBANK)
62: C
63: C**** PARTICLE BANK
64: C real EEE(NBANK), WWW(NBANK)
65: C integer IZZ(NBANK), IBREG(NBANK), IGG(NBANK)

```

```

66: C
67: C**** TALLY ARRAY
68: C real*8 WCNTR(*), NCNTR(*)
69: C real DNZON(NUC,NZONE,2), ENGYB(*)
70: C real*8 DNFLXSM(NGP1,NGP1,NREG,NUC,*),
71: C & RMICSM(NGP1,NGP1,NREG,NUC,JSCTM,*)
72: C integer KYPOS(3)
73: C real*8 RMIMU(NGP1,NGP1+1,NREG,NUC,*), WMIMU(NGP1,NGP1+1,NREG,NUC,*)
74: C
75: C**** WORK AREA
76: C integer IWK6(NBANK), IWK2(NBANK), IWK8(NBANK), IWK9(NBANK),
77: C & IWK10(NBANK), IWK11(NBANK), IWK12(NBANK)
78: C real WK2(NBANK), WK3(NBANK), WK4(NBANK), WK5(NBANK), R(5*NBANK)
79: C 0...NBANK..2NBANK..3NBANK..4NBANK..5NBANK..6NBANK..7NBANK..8NBANK
80: C IWK6 IWK2 IWK3 IWK4 IWK5 RP R,IWORK
81: C
82: Cccc include 'INC/_LINEAR'
83: C logical LGJSCTM,LGJSMU
84: C
85: C**** MAKE TEMPORARY STACKS IWK11 FOR THERMAL INELASTIC (SAB), N1
86: C IWK12 FOR THERMAL ELASTIC , N3
87: C USUAL COLLISION STACK IS OBTAINED BY IBP( IWK1X(J) )
88: C
89: C GATHER
90: C WK2(I) : INCIDENT ENERGY OF PARTICLE OF IBP(I)
91: C IWK8(I): POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
92: C IWK9(I): NUMBER OF INCIDENT ENERGY POINTS
93: C NOTE. ENERG DISTRIBUTION (INELASTIC)
94: C ANGULAR DISTRIBUTION (ELASTIC)
95: C
96: N1 = 0
97: N2 = 0
98: N3 = 0
99: N4 = 0
100: c##<2007/03/14:PN3:
101: KIBNK19 = IWK6(1)
102: c##>
103: LGJSMU = .false.
104: LGJSCTM = .false.
105: DZERO = 0
106: C
107: do 100 I = 1, NCOLT
108: WK2(I) = EEE(IBP(I))
109: WK3(I) = SMIC(IBP(I),ICLN(I),LMIC(4))
110: IWK6(I) = 0
111: *VOCL STMT,IF(0)
112: if ( WK3(I).gt.0. ) then
113: N2 = N2 + 1
114: IWK12(N2) = I
115: else
116: N1 = N1 + 1
117: IWK11(N1) = I
118: IWK8(I) = KLIB2(ICLN(I),92,13) + 3
119: IWK9(I) = ICX(IWK8(I)-1)
120: end if
121: 100 continue
122: C
123: if ( JSCMU.ne.0.and.N1.gt.0.and.(JMICE(4).ne.0.or.JMACE(4).ne.0) )
124: & LGJSMU = .true.
125: if ( JSCTM.ne.0.and.JRTCL.ne.0.and.N1.gt.0.and.
126: & (JMICE(4).ne.0.or.JMACE(4).ne.0) ) then
127: LGJSCTM = .true.
128: if ( LMIC(6).eq.0 ) then
129: *VOCL LOOP,NOVREC
130: do J = 1, N1

```

src/mvp/thscf.f

```

131:      I = IWK11(J)
132:      if ( KSPI(IBP(I),ICLN(I)).ge.KLIB2(ICLN(I),4,3) ) then
133:        L2 = KLIB1(ICLN(I),1) + KSPI(IBP(I),ICLN(I))
134:        R1 = (WK2(I)-CX(L2))/(CX(L2-1)-CX(L2))
135:        L1 = KLIB2(ICLN(I),4,11) - KLIB2(ICLN(I),4,3) +
136:      &      KSPI(IBP(I),ICLN(I))
137:        XIN = CX(L1)*R1 + CX(L1+1)*(1-R1)
138:      else
139:        XIN = 0
140:      end if
141:      WK4(J) = XIN
142:    end do
143:  else
144:    do J = 1, N1
145:      I = IWK11(J)
146:      XIN = SMIC(IBP(I),ICLN(I),LMIC(6))
147:      WK4(J) = XIN
148:    end do
149:  end if
150: end if
151: if ( N2.gt.0 ) then
152:   call RANU2( IRAND, R, N2, ICON )
153: C
154: C..... THERMAL INELASTIC CROSS SECTION ARE NOT GIVEN IN SMIC ARRAY
155:   if ( LMIC(6).eq.0 ) then
156:     *VOCL LOOP,NOVREC
157:     do 110 J = 1, N2
158:       I = IWK12(J)
159:       if ( KSPI(IBP(I),ICLN(I)).ge.KLIB2(ICLN(I),4,3) ) then
160:         L2 = KLIB1(ICLN(I),1) + KSPI(IBP(I),ICLN(I))
161:         R1 = (WK2(I)-CX(L2))/(CX(L2-1)-CX(L2))
162:         CM      L1 = KLIB2(ICLN(I),92,11) - KLIB2(ICLN(I),92,3)
163:         CM      + KSPI(IBP(I),ICLN(I))
164:         L1 = KLIB2(ICLN(I),4,11) - KLIB2(ICLN(I),4,3)
165:         &      + KSPI(IBP(I),ICLN(I))
166:         XIN = CX(L1)*R1 + CX(L1+1)*(1.-R1)
167:       else
168:         XIN = 0.0
169:       end if
170:       X2 = (XIN+WK3(I))*R(J)
171:       if ( X2.lt.XIN ) then
172:         N1 = N1 + 1
173:         IWK11(N1) = I
174:         IWK8(I) = KLIB2(ICLN(I),92,13) + 3
175:         IWK9(I) = ICX(IWK8(I)-1)
176:         if ( LGJSCTM ) WK4(N1) = XIN
177:       else if ( KLIB2(ICLN(I),93,7).eq.0 ) then
178:         N4 = N4 + 1
179:         IWK2(N4) = I
180:         IWK8(I) = ICX(KLIB2(ICLN(I),93,13)) + 2
181:         IWK9(I) = ICX(IWK8(I)-1)
182:         IWK6(I) = 1
183:       else
184:         N3 = N3 + 1
185:         IWK12(N3) = I
186:         IWK8(I) = KLIB2(ICLN(I),93,12)
187:         IWK9(I) = KLIB2(ICLN(I),93,7)
188:       end if
189:     110 continue
190: C..... THERMAL INELASTIC CROSS SECTION ARE GIVEN IN SMIC ARRAY
191:   else
192:     *VOCL LOOP,NOVREC
193:     do 120 J = 1, N2
194:       I = IWK12(J)
195:       XIN = SMIC(IBP(I),ICLN(I),LMIC(6))

```

```

196:       X2 = (XIN+WK3(I))*R(J)
197:       if ( X2.lt.XIN ) then
198:         N1 = N1 + 1
199:         IWK11(N1) = I
200:         IWK8(I) = KLIB2(ICLN(I),92,13) + 3
201:         IWK9(I) = ICX(IWK8(I)-1)
202:         if ( LGJSCTM ) WK4(N1) = XIN
203:       else if ( KLIB2(ICLN(I),93,7).eq.0 ) then
204:         N4 = N4 + 1
205:         IWK2(N4) = I
206:         IWK8(I) = ICX(KLIB2(ICLN(I),93,13)) + 2
207:         IWK9(I) = ICX(IWK8(I)-1)
208:         IWK6(I) = 1
209:       else
210:         N3 = N3 + 1
211:         IWK12(N3) = I
212:         IWK8(I) = KLIB2(ICLN(I),93,12)
213:         IWK9(I) = KLIB2(ICLN(I),93,7)
214:       end if
215:     120 continue
216:   end if
217: C
218:   end if
219: C
220: C**** SELECTION OF USED TABLE ----> IWK10
221: C   WK2(I) : INCIDENT ENERGY OF PARTICLE OF IBP(I)
222: C   IWK8(I): POINTER OF INCIDENT ENERGY FOR WHICH D. ARE GIVEN
223: C   IWK9(I): NUMBER OF INCIDENT ENERGY POINTS
224: C   NOTE. ENERG DISTRIBUTION (INELASTIC)
225: C   ANGULAR DISTRIBUTION (ELASTIC)
226: C
227:   call BSINC3( CX, IWK9, WK5, IWK8, WK2, IWK10, NCOLT )
228: C
229:   call RANU2( IRAND, R, NCOLT, ICON )
230: C
231:   *VOCL LOOP,NOVREC
232:   do 140 I = 1, NCOLT
233:     if ( IWK6(I).eq.0 ) then
234:       L2 = IWK8(I) + IWK10(I)
235:       L1 = L2 - 1
236:       INTE = ICX(L2+IWK9(I))
237:       if ( INTE.eq.5 .or. INTE.eq.3 ) then
238:         C..... LOG INTERPOLATION
239:         X1 = LOG(CX(L1)/WK2(I)) /LOG(CX(L1)/CX(L2))
240:       else
241:         C..... LINEAR INTERPOLATION
242:         X1 = (CX(L1)-WK2(I)) / (CX(L1)-CX(L2))
243:       end if
244:     end if
245: C/#IF ROUNDOFF(NEAREST)
246:     IWK10(I) = min(1,INT(R(I)+X1)) + IWK10(I)
247: C/#ELSE
248:     *      R1 = R(I) + X1
249:     *      IWK10(I) = IWK10(I) + R1
250: C/#ENDIF
251: C added 1999/06/17
252: C   This "if" statement seems to be redundant. But there are some cases
253: C   where IWK10 is 0 or negative. This is caused by a non-zero cross
254: C   section at the lowest incident energy, especially for thermal
255: C   scattering cross sections. This problem should be fixed in the
256: C   future.
257:
258:     if ( IWK10(I).le.0 ) IWK10(I) = 1
259:   end if
260:

```


src/mvp/thset.f

```

261: 140 continue
262: C
263: C**** THERMAL INCOHERENT ELASTIC SCATTERING
264: C
265:      call RANU2( IRAND, R, N3*2, ICON )
266: C
267: *VOCL LOOP,NOVREC
268:      do 150 J = 1, N3
269:          I          = IWK12(J)
270:
271: C/#IF ROUNDOFF(NEAREST)
272:      LK11 = KL1B1(ICLN(I),7)
273:      L3   = MIN(INT(LK11*R(J))+1,LK11)
274:      &    + IWK8(I) + IWK9(I)*2 + KL1B1(ICLN(I),20)*(IWK10(I)-1)
275:      CTH(I) = (CX(L3-1)-CX(L3))*R(N3+J) + CX(L3)
276: C/#ELSE
277:      *    L2 = R(J)*KL1B1(ICLN(I),7)
278:      *    L3 = L2 + IWK8(I)+IWK9(I)*2
279:      *    &    + KL1B1(ICLN(I),20)*(IWK10(I)-1)
280:      *    CTH(I) = (CX(L3-1)-CX(L3))*R(N3+J) + CX(L3)
281: C/#ENDIF
282:      if ( LGJSCTM ) then
283:          N = ICLN(I)
284:          IP = IBP(I)
285:          DN = DNZON(N,IZZ(IP),1)
286:          if ( DN.gt.DZERO ) then
287:              IE = IGG(IP)
288:              IR = KZREG(IZZ(IP))
289:              if ( JTLLT.ne.0 ) IR = IBREG(IP)
290:              DNF = DN*WWW(IP)/SMAC(IP,MMAC(IP,1),LMAC(1))
291:              DNFLXSM(IE,IE,IR,N,KYPOS(1)) =
292:              &      DNFLXSM(IE,IE,IR,N,KYPOS(1)) + DNF
293:              RMICSM(IE,IE,IR,N,1,KYPOS(1)) =
294:              &      RMICSM(IE,IE,IR,N,1,KYPOS(1)) +
295:              &      SMIC(IP,N,LMIC(4))*DNF
296:              if ( JSCTM.gt.1 ) then
297:                  SDN = SMIC(IP,N,LMIC(4))*DNF
298:                  XMU = CTH(I)
299:                  do K = 2, JSCTM
300:                      XMUN = XMU**(K-1)*SDN
301:                      RMICSM(IE,IE,IR,N,K,KYPOS(1)) =
302:              &      RMICSM(IE,IE,IR,N,K,KYPOS(1)) + XMUN
303:                  end do
304:              end if
305:          end if
306:      end if
307:      if ( LGJSMU ) then
308:          N = ICLN(I)
309:          IP = IBP(I)
310:          IE = IGG(IP)
311:          IR = KZREG(IZZ(IP))
312:          if ( JTLLT.ne.0 ) IR = IBREG(IP)
313:          XMU = CTH(I) + 1
314:          WMIMU(IE,IE,IR,N,KYPOS(1)) =
315:          &      WMIMU(IE,IE,IR,N,KYPOS(1)) + WWW(IP)
316:          RMIMU(IE,IE,IR,N,KYPOS(1)) =
317:          &      RMIMU(IE,IE,IR,N,KYPOS(1)) + WWW(IP)*XMU
318:          WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
319:          &      WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)
320:          RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
321:          &      RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)*XMU
322:      end if
323:
324: 150 continue
325: C

```

```

326: C**** THERMAL COHERENT ELASTIC SCATTERING
327: C
328:      call RANU2( IRAND, R, N4*2, ICON )
329: C
330: *VOCL LOOP,NOVREC
331:      do 160 J = 1, N4
332:          I          = IWK2(J)
333:          IBRAG      = IWK10(I)
334:          if ( WK2(I).gt.CX(IWK8(I)+IWK9(I)-1) ) IBRAG = IWK9(I)
335:          if ( IBRAG.eq.0 ) IBRAG = 1
336:          LSBRAG     = ICX(IWK8(I)+2*IWK9(I)+IBRAG-1)
337:          NBIN       = ICX(LSBRAG)
338:
339: C/#IF ROUNDOFF(NEAREST)
340:          LL2       = MIN(INT(NBIN*R(J))+1,NBIN)
341: C/#ELSE
342:          *    LL2       = NBIN*R(J) + 1
343: C/#ENDIF
344:
345:          LL3       = LSBRAG + LL2
346:
347: C/#IF ROUNDOFF(NEAREST)
348:          LL4       = 2*LL2
349:          LL4       = MIN(LL4,INT(LL4+R(N4+J)-CX(LL3))) + LSBRAG +
350:          &          NBIN
351: C/#ELSE
352:          *    R2       = R(N4+J) - CX(LL3)
353:          *    LL4       = LSBRAG + NBIN + 2*LL2 + R2
354: C/#ENDIF
355:          L3        = ICX(LL4)
356: C
357:          CTH(I) = 1.0 - 2.0 * CX(IWK8(I)+L3-1) / WK2(I)
358:          if ( CTH(I).gt. 1.0 ) CTH(I) = 1.0
359:          if ( CTH(I).lt.-1.0 ) CTH(I) = -1.0
360: C added 2002/08/13
361: C This "if" statement seems to be redundant. But there are some cases
362: C where IWK10 is 0 or negative. This is caused by a non-zero cross
363: C section at the lowest incident energy, especially for thermal
364: C scattering cross sections. This problem should be fixed in the
365: C future.
366: C
367:          if ( LGJSCTM ) then
368:              N = ICLN(I)
369:              IP = IBP(I)
370:              DN = DNZON(N,IZZ(IP),1)
371:              if ( DN.gt.DZERO ) then
372:                  IE = IGG(IP)
373:                  IR = KZREG(IZZ(IP))
374:                  if ( JTLLT.ne.0 ) IR = IBREG(IP)
375:                  DNF = DN*WWW(IP)/SMAC(IP,MMAC(IP,1),LMAC(1))
376:                  DNFLXSM(IE,IE,IR,N,KYPOS(1)) =
377:                  &      DNFLXSM(IE,IE,IR,N,KYPOS(1)) + DNF
378:                  RMICSM(IE,IE,IR,N,1,KYPOS(1)) =
379:                  &      RMICSM(IE,IE,IR,N,1,KYPOS(1)) +
380:                  &      SMIC(IP,N,LMIC(4))*DNF
381:                  if ( JSCTM.gt.1 ) then
382:                      SDN = SMIC(IP,N,LMIC(4))*DNF
383:                      XMU = CTH(I)
384:                      do K = 2, JSCTM
385:                          XMUN = XMU**(K-1)*SDN
386:                          RMICSM(IE,IE,IR,N,K,KYPOS(1)) =
387:                  &      RMICSM(IE,IE,IR,N,K,KYPOS(1)) + XMUN
388:                      end do
389:                  end if
390:              end if

```

src/mvp/thsc.f

```

391:         end if
392:         if ( LGJSMU ) then
393:             N = ICLN(I)
394:             IP = IBP(I)
395:             IE = IGG(IP)
396:             IR = KZREG(IZZ(IP))
397:             if ( JTLT.ne.0 ) IR = IBREG(IP)
398:             XMU = CTH(I) + 1
399:             WMIMU(IE,IE,IR,N,KYPOS(1)) =
400:             & WMIMU(IE,IE,IR,N,KYPOS(1)) + WWW(IP)
401:             RMIMU(IE,IE,IR,N,KYPOS(1)) =
402:             & RMIMU(IE,IE,IR,N,KYPOS(1)) + WWW(IP)*XMU
403:             WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
404:             & WMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)
405:             RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) =
406:             & RMIMU(IE,NGP1+1,IR,N,KYPOS(1)) + WWW(IP)*XMU
407:         end if
408:         160 continue
409: C
410: C**** THERMAL INELASTIC SCATTERING
411: C
412:         call RANU2( IRAND, R, N1*5, ICON )
413:         N12 = N1*2
414:         N13 = N1*3
415:         N14 = N1*4
416: C
417: *VOCL LOOP,NOVREC
418:         do 170 J = 1, N1
419:             I = IWK11(J)
420: C..... DETERMINE THE OUTGOING ENERGY --- MODIFY EEE( IBP(I) )
421:             NELF = IWK9(I)
422:             IIN = IWK10(I)
423:             L1 = IWK8(I) + 3*NELF*IIN - NELF - 1
424:
425: C/#IF ROUNDOFF(NEAREST)
426:             L2 = MIN(INT(NELF*R(J))+1,NELF)
427: C/#ELSE
428:             * L2 = NELF*R(J) + 1
429: C/#ENDIF
430:
431:             L3 = L1 + L2
432:
433: C/#IF ROUNDOFF(NEAREST)
434: C97/11/07 L4 = INT(2*L2+R(N1+J)-CX(L3)) + L1 + NELF
435:             L4 = 2*L2
436:             L4 = min(L4,INT(L4+R(N1+J)-CX(L3))) + L1 + NELF
437: C/#ELSE
438: CM1993-9 L4 = L1 + NELF + 2*L2 + R(N1+J) - CX(L3)
439:             * R1 = R(N1+J) - CX(L3)
440:             * L4 = L1 + NELF + 2*L2 + R1
441: C/#ENDIF
442:
443:             IOUT = ICX(L4)
444:             IP = IBP(I)
445: *VOCL STMT,IF(0)
446:             if ( IOUT.eq.1 ) then
447:                 EEE(IP) = CX(IWK8(I))
448:             else
449:                 L6 = IWK8(I) + IOUT - 1
450:                 L5 = L6 - 1
451: CM1991-12-18
452: CM EE = ( CX(L5)-CX(L6) ) * R(N12+J) + CX(L6)
453: CM EEE(IP) = MIN( EE, ETHMAX )
454: EEE(IP) = (CX(L5)-CX(L6))*R(N12+J) + CX(L6)
455: IOUTT = R(N12+J) + 0.5

```

```

456:             IOUT = IOUT - IOUTT
457: C..... SET EEE(IP) =< ETHMAX
458:             end if
459: C
460: C..... DETERMINE SCATTERING ANGLE COSINE ----> XMU --- CTH
461: C
462:             if ( KL1B1(ICLN(I),31).ne.1 ) then
463:                 I1 = MIN(IIN,IOUT)
464:                 I2 = MAX(IIN,IOUT)
465:                 I22 = I2*(I2-1) /2
466:                 L7 = KL1B2(ICLN(I),92,12) + NELF*2 + (I22+I1-1)*5
467:             else
468:                 L7 = KL1B2(ICLN(I),92,12) + NELF*2 +
469:                 & NELF*5*(IIN-1) + 5*(IOUT-1)
470:             end if
471: C
472:             RANSU = R(N13+J)
473:             if ( RANSU.le.CX(L7+1) ) then
474: C
475:                 XMU = R(N14+J) * SIGN( 1.0, CX(L7)-RANSU )
476:                 I3 = CX(L7) - RANSU + 1.
477:                 I3 = 2*I3 - 1
478:                 XMU = R(N14+J)*I3
479:             else
480: C
481:                 I4 = 1. + CX(L7+3) - RANSU
482:                 XMU = SIGN( 1.0, CX(L7+2)-RANSU ) * I4
483:                 I5 = CX(L7+2) - RANSU + 1.
484:                 I5 = 2*I5 - 1
485:                 XMU = I4*I5
486:             end if
487:             CTH(I) = XMU
488: Cml994-8
489:             EEE(IP) = MAX(EEE(IP),EBOT)
490:             170 continue
491: C##<2007/03/14:PN3:
492: C
493:             if ( KIBNK19.gt.0 ) then
494:                 IWK6(1) = N3
495:                 IWK6(2) = N1
496:                 if ( N3.gt.0 ) then
497:                     do I = 1, N3
498:                         IWK6(I+2) = IBP(IWK12(I))
499:                     end do
500:                 end if
501:                 if ( N1.gt.0 ) then
502:                     do I = 1, N1
503:                         IWK6(I+2+N3) = IBP(IWK11(I))
504:                     end do
505:                 end if
506:             end if
507: C##>
508: C
509: C ..... scattering matrix tally by thermal inelastic
510:             if ( LGJSCTM.and.N1.gt.0.and.KYPOS(2).gt.0 ) then
511:                 do J = 1, N1
512:                     IP = IBP(IWK11(J))
513:                     WK5(J) = EEE(IP)
514:                 end do
515:                 call BSVDEC( ENGYB, NGP1+1, WK5, IWK12, N1 )
516:                 do J = 1, N1
517:                     I = IWK11(J)
518:                     N = ICLN(I)
519:                     IP = IBP(I)
520:                     DN = DNZON(N,IZZ(IP),1)
521:                     if ( DN.gt.DZERO ) then
522:                         IE = IGG(IP)

```

src/mvp/thst.f

```
521:         JE = IWK12(J)
522:         IR = KZREG(IZZ(IP))
523:         if ( JTLLT.ne.0 ) IR = IBREG(IP)
524:         DNF = DN*WWW(IP)/SMAC(IP,MMAC(IP,1),LMAC(1))
525:         DNFLXSM(IE,JE,IR,N,KYPOS(2)) =
526:         &         DNFLXSM(IE,JE,IR,N,KYPOS(2)) + DNF
527: c         RMICSM(IE,JE,IR,N,KYPOS(2)) =
528: c         &         RMICSM(IE,JE,IR,N,KYPOS(2)) + WK4(J)*DNF
529:         RMICSM(IE,JE,IR,N,1,KYPOS(2)) =
530:         &         RMICSM(IE,JE,IR,N,1,KYPOS(2)) + WWW(IP)
531:         if ( JSCTM.gt.1 ) then
532:             XMU = CTH(I)
533:             do K = 2, JSCTM
534:                 XMUN = XMU**(K-1)*WWW(IP)
535:                 RMICSM(IE,JE,IR,N,K,KYPOS(2)) =
536:                 &                 RMICSM(IE,JE,IR,N,K,KYPOS(2)) + XMUN
537:             end do
538:         end if
539:     end if
540: end do
541: end if
542: C
543: C ..... scattering mubar tally by thermal inelastic
544: if ( LGJSMU.and.N1.gt.0.and.KYPOS(2).gt.0 ) then
545:     if ( .not.LGJSCTM ) then
546:         do J = 1, N1
547:             IP = IBP(IWK11(J))
548:             WK5(J) = EEE(IP)
549:         end do
550:         call BSVDEC( ENGYB, NGP1+1, WK5, IWK12, N1 )
551:     end if
552:     do J = 1, N1
553:         I = IWK11(J)
554:         N = ICLN(I)
555:         IP = IBP(I)
556:         IE = IGG(IP)
557:         JE = IWK12(J)
558:         IR = KZREG(IZZ(IP))
559:         if ( JTLLT.ne.0 ) IR = IBREG(IP)
560:         XMU = CTH(I) + 1
561:         WMIMU(IE,IE,IR,N,KYPOS(2)) =
562:         &         WMIMU(IE,IE,IR,N,KYPOS(2)) + WWW(IP)
563:         RMIMU(IE,IE,IR,N,KYPOS(2)) =
564:         &         RMIMU(IE,IE,IR,N,KYPOS(2)) + WWW(IP)*XMU
565:         WMIMU(IE,NGP1+1,IR,N,KYPOS(2)) =
566:         &         WMIMU(IE,NGP1+1,IR,N,KYPOS(2)) + WWW(IP)
567:         RMIMU(IE,NGP1+1,IR,N,KYPOS(2)) =
568:         &         RMIMU(IE,NGP1+1,IR,N,KYPOS(2)) + WWW(IP)*XMU
569:     end do
570: end if
571: C
572: return
573: end
```

src/mvp/ttbr.f

```

1:      subroutine TTBR( IOW , JTIME, JIMPT, JLATT, JHLAT, JTLT,
2:      F      IRAND, NZONE, NEST , NMAT , NBANK,
3:      N      NEVENT,NCNTR, WCNTR,
4:      T      KZMAT,
5:      X      EBOTP, NEE , MTOP, EEL , RKT , PBT ,
6:      X      EBT , EBTP , IEBTP,
7:      %      LSCLP, NCOLP, NELC , LSDED, NDEAD, ND0 ,
8:      B      XXX , YYY , ZZZ , AAA , BBB , CCC , WWW , EEE ,
9:      B      TTT,ITT,XIM,KLSF,KKP,IZZ,IGG,LEVL,LZZ,LPOS,LCRS,IBREG,IBSPC,
10:     I      NKILP,
11:     W      R , IMTTE0, UOUT,
12:     W      WK30 , WK31 , WK32 , IWK31,
13:     W      WK33 , WK34 , WK35 , WK36 ,
14:     W      IWK32, IWK33, IWK34 , IWK35, IWK36, IWK37, IWK38,
15:     W      WK41 , WK42 , WK43 )
16: C=====
17: C  PURPOSE : GENERATE BREMSSTRAHLUNG PHOTONS WITH THE THICK-TARGET
18: C            BREMSSTRAHLUNG APPROXIMATION.
19: C  CALLED IN:  PHOTR
20: C=====
21: C
22: C  NBANK          I4  PARTICLE BANK LENGTH.
23: C  LSCLP(NBANK)   I4  BANK INDEX OF PHOTONS WAITING FOR COLLISION.
24: C  NCOLP          I4  LENGTH OF PHOTON COLLISION STACK.
25: C  NKILP          I4  NUMBER OF KILLED PARENT PHOTON FOR BANK OVERFLOW.
26: C*
27: C  NELC           I4  NUMBER OF GENERATED ELECTRONS BY PHOTON COLLISION
28: C                  BY EMAKES.
29: C  WK30(2*NBANK)  R4  EMITTED ELECTRON WEIGHTS WITH PHOTON COLLISIONS.
30: C                  (WEIGHTS MAY BE CHANGE IN TTBR.)
31: C  WK31(2*NBANK)  R4  EMITTED ELECTRON ENERGIES WITH PHOTON COLLISIONS.
32: C  WK32(2*NBANK)  R4  DIRECTION COSINE OF EMITTED ELECTRONS.
33: C  IWK31(2*NBANK) I4  BANK INDEX OF PARENT PHOTON EMITTING ELECTRON
34: C                  (IF NEGATIVE, PARENT PHOTON DIED).
35: C  UOUT(NBANK)    R4  DIRECTION COSINE OF PHOTON AFTER COLLISION.
36: C  WK41(2*NBANK)  R8  DIRECTION OF X-AXIS FOR PAIR PRODUCTION AND
37: C                  COEFFICIENT OF ANGLE BEFOR COLISSION FOR COMPTON
38: C                  SCATTERING.
39: C  WK42(2*NBANK)  R8  DIRECTION OF Y-AXIS FOR PAIR PRODUCTION AND
40: C                  COEFFICIENT OF ANGLE AFTER COLLISION FOR COMPTON
41: C                  SCATTERING.
42: C  WK43(2*NBANK)  R8  DIRECTION OF Z-AXIS FOR PAIR PRODUCTION
43: C  IMTTE0(4)      I4  NUMBER OF GENERATED ELECTRONS.
44: C                  1  NUMBER BY PHOTOELECTRIC
45: C                  2  NUMBER BY ELECTRON OF PAIR PRODUCTIONS
46: C                  3  NUMBER BY POSITRON OF PAIR PRODUCTIONS
47: C                  4  NUMBER BY INCOHERENT SCATTERING
48: C*
49: C  NEE            I4  NUMBER OF ELECTRON ENERGY GRIDS.
50: C  MTOP           I4  NUMBER OF DATA POINTS FOR BREMSSTRAHLUNG
51: C                  PHOTON/ELECTRON ENERGY (K/T) RATIOS.
52: C  EEL(NEE)       R4  ELECTRON GRIDS OF ELECTRON CROSS SECTIONS, IN EV.
53: C  RKT(MTOP)      R4  BREMSSTRAHLUNG PHOTON/ELECTRON ENERGY RATIOS.
54: C  PBR(NEE,NMAT)  R4  BREMSSTRAHLUNG CROSS SECTIONS BY MATERIALS.
55: C  PBT(NEE,NMAT)  R4  BREMSSTRAHLUNG YIELDS FOR THICK-TARGET BREMS-
56: C                  TRAHUNG (TTB) APPROXIMATION BY MATERIALS.
57: C  EBA(MTOP,NEE,NMAT)
58: C                  R4  BREMSSTRAHLUNG CUMULATIVE PROBABILITIES BY
59: C                  MATERIALS.
60: C  EBT(MTOP,NEE,NMAT)
61: C                  R4  BREMSSTRAHLUNG CUMULATIVE PROBABILITIES FOR TTB
62: C                  BY MATERIALS.
63: C  EBTP(MTOP-1,NEE,NMAT)
64: C                  R4  BREMSSTRAHLUNG PARTIAL PROBABILITIES FOR TTB BY
65: C                  MATERIALS.

```

```

66: C  IEBTP(2*(MTOP-1),NEE,NMAT)
67: C                  I4  CONDITIONAL DISCRETE SAMPLING TABLE FOR BREMS-
68: C                  TRAHUNG PROBABILITIES FOR TTB BY MATERIALS.
69: C*
70: C  NMAT           I4  NUMBER OF MATERIALS.
71: C  NZONE          I4  NUMBER OF ZONES.
72: C  KZMAT(NZONE)   I4  MATERIAL NUMBER OF EACH ZONE.
73: C  IZZ(NBANK)     I4  ZONE NUMBER OF PARTICLES.
74: C*
75: C  R(NBANK*4)     R4  TEMPORARY AREA FOR RANDUM NUMBER.
76: C*
77: C  WCNTR(NEVENT,2) R8  SUMS OF WEIGHT FOR VARIOUS EVENTS.
78: C
79: C=====
80: C
81: C      implicit real*8(A-H,O-Z)
82: C
83: C      include 'INC/_KPIDS'
84: C
85: C**** GEOMETRY ARRAY DATA
86: C
87: C      integer KZMAT(NZONE)
88: C
89: C**** CROSS SECTION DATA
90: C
91: C      integer IEBTP((MTOP-1)*2,NEE,NMAT)
92: C      real EBOTP
93: C      real EEL(NEE), RKT(MTOP), PBT(NEE,NMAT), EBT(MTOP,NEE,NMAT),
94: C      &      EBTP(MTOP-1,NEE,NMAT)
95: C
96: C**** STACK
97: C
98: C      integer NCOLP, NELC, NDEAD
99: C      integer LSCLP(NBANK), LSDED(NBANK)
100: C
101: C**** PARTICLE BANK
102: C
103: C      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
104: C      &      CCC(NBANK)
105: C      integer KKP(NBANK)
106: C      integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
107: C      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
108: C      real EEE(NBANK), WWW(NBANK), XIM(NBANK)
109: C      real*8 TTT(NBANK)
110: C      integer ITT(NBANK)
111: C      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
112: C
113: C**** OUTPUT
114: C
115: C      integer NKILP
116: C
117: C**** TALLY ARRAY
118: C
119: C      CCC      real*8 WCNTR(NEVENT,2), NCNTR(NEVENT,2)
120: C      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
121: C
122: C**** WORKING AREA
123: C
124: C      .... PROVIDED DATA ....
125: C
126: C      integer IWK31(2*NBANK), IMTTE0(4)
127: C      real UOUT(NBANK), WK30(2*NBANK), WK31(2*NBANK), WK32(2*NBANK)
128: C      real*8 WK41(2*NBANK), WK42(2*NBANK), WK43(2*NBANK)
129: C
130: C      .... TEMPORARY DATA ....

```

src/mvp/ttbr.f

```

131: C
132: integer IWK32(2*NBANK), IWK33(2*NBANK), IWK34(2*NBANK),
133: & IWK35(NBANK), IWK36(NBANK), IWK37(NBANK), IWK38(NBANK)
134: real WK33(2*NBANK), WK34(2*NBANK), WK35(NBANK), WK36(NBANK),
135: & R(NBANK*4)
136: C
137: integer IMTTE1(4)
138: C
139: C-----
140: C
141: C BANKE : GENERATION PROBABILITY OF BREMSSTRAHLUNG
142: C PHOTON, IF PARTICLE BANK FILLS.
143: C ( R < BANKE, BREMSSTRAHLUNG PHOTON )
144: C ( R > BANKE, PARENT PHOTON )
145: C
146: data BANKE /0.5D0/
147: C
148: I1BASE = IMTTE0(1)
149: C
150: NKILP = 0
151: C
152: C
153: C-----
154: C INTERPOLATION OF YIELD DATA, PBT
155: C-----
156: C
157: C IWK32 : INDEX VECTOR ON EEL TABLE FOR EMITTED
158: C ELECTRON ENERGY.
159: C WK33 : FRACTION RATIO OF ELECTRON ENERGY IN A BIN.
160: C IWK33 : MATERIAL NUMBER OF ZONE EXISTING ELECTRON.
161: C WK34 : REAL NUMBER (YIELDS) OF GENERATING PHOTONS
162: C BY BREMSSTRAHLUNG.
163: C
164: call BSVDEC( EEL, NEE, WK31, IWK32, NELC )
165: C
166: call RANU2( IRAND, R, NELC, ICON )
167: C
168: C-----
169: C DETERMINE THE NUMBER OF GENERATING PHOTONS BY BREMSSTRAHLUNG.
170: C-----
171: C
172: C IWK34 : INTEGER NUMBER OF GENERATING PHOTONS BY
173: C BREMSSTRAHLUNG (REAL NUMBER PRESERVE
174: C STATISTICALLY).
175: C NT : TOTAL NUMBER OF GENERATING PHOTONS.
176: C
177: C
178: NT = 0
179: C
180: do 100 I = 1, NELC
181: WK33(I) = (EEL(IWK32(I))-WK31(I)) /
182: & (EEL(IWK32(I))-EEL(IWK32(I)+1))
183: IWK33(I) = KZMAT(IZZ(ABS(IWK31(I))))
184: WK34(I) = (1.0-WK33(I))*PBT(IWK32(I),IWK33(I)) + WK33(I)*
185: & PBT(IWK32(I)+1,IWK33(I))
186: C
187: IWK34(I) = WK34(I) + R(I)
188: C .... WK30(I)LE.0 ==> THE ELECTRON OR POSITRON FROM PAIR PRODUCTION
189: C HAS ENERGY <= EBOTE
190: if ( WK30(I).le.0.0 ) IWK34(I) = 0
191: NT = NT + IWK34(I)
192: 100 continue
193: C
194: C ..... NT = 0, THEN RETURN
195: if ( NT.le.0 ) then

```

```

196: go to 480
197: end if
198: C
199: C-----
200: C CHECK BANK-OVERFLOW : ALL PHOTONS CAN BE GENERATED.
201: C-----
202: C
203: C NKILP : NUMBER OF KILLED PARENT PHOTONS.
204: C
205: if ( NT.gt.NDEAD ) then
206: C
207: C
208: C-----
209: C CHECK BANK-OVERFLOW : BANK HAS INSUFFICIENT SPACE TO ACCEPT ALL
210: C GENERATED PHOTONS.
211: C-----
212: C
213: C NB : TOTAL NUMBER OF ELECTRON GENERATING PHOTON,
214: C IN PRACTICE.
215: C NY : TOTAL NUMBER OF ELECTRON GENERATING MULTIPLE
216: C PHOTONS.
217: C NMI : MAXIMUM NUMBER OF PHOTONS GENERATED BY AN
218: C ELECTRON.
219: C
220: NB = 0
221: NY = 0
222: do 110 I = 1, NELC
223: if ( IWK34(I).gt.0 ) NB = NB + 1
224: if ( IWK34(I).gt.1 ) NY = NY + 1
225: 110 continue
226: C
227: C ..... BANK SPACE IS GREATER THAN NUMBER OF ELECTRONS .....
228: C
229: if ( NB.le.NDEAD ) then
230: NV = NDEAD - NB
231: C
232: C ..... REDUCE THE MAXIMUM NUMBER OF PHOTONS TO 2 .....
233: C
234: if ( NV.ge.NY ) then
235: do 120 I = 1, NELC
236: if ( IWK34(I).gt.2 ) then
237: WK30(I) = WK30(I)*IWK34(I) /2.0
238: IWK34(I) = 2
239: end if
240: 120 continue
241: C
242: C
243: C ..... SELECTIVE REDUCE MULTIPLE PHOTONS TO 1 .....
244: C THIS PART DOESNOT WORK
245: C ELSEIF( NV.GT.0 ) THEN
246: C
247: C CALL RANU2 ( IRAND, R, NY, ICON )
248: C
249: C NS = NT - NB
250: C QY = NS
251: C SV = NV / QY * 0.9
252: C J = 0
253: C ND = 0
254: C DO 220 I=1,NELC
255: C IF( IWK34(I).GT.1 ) THEN
256: C J = J + 1
257: C IF( R(J).GT.SV ) THEN
258: C ND = ND + IWK34(I) - 1
259: C WK30(I) = WK30(I) * IWK34(I)
260: C IWK34(I) = 1

```

src/mvp/ttbr.f

```

261: C                ENDIF
262: C                ENDIF
263: C 220            CONTINUE
264: C                IF( NS-ND.GT.NV ) THEN
265: C                    DO 230 I=1,NELC
266: C                        IF( IWK34(I).GT.1 ) THEN
267: C                            ND = ND + IWK34(I) - 1
268: C                            WK30(I) = WK30(I) * IWK34(I)
269: C                            IWK34(I) = 1
270: C                        ENDIF
271: C                    IF( NS-ND.LE.NV ) GO TO 240
272: C 230            CONTINUE
273: C                ENDIF
274: C 240            CONTINUE
275: C                else
276: C                    do 130 I = 1, NELC
277: C                        if ( IWK34(I).gt.0 ) then
278: C                            WK30(I) = WK30(I)*IWK34(I)
279: C                            IWK34(I) = 1
280: C                        end if
281: C 130            continue
282: C                end if
283: C
284: C                .... BANK SPACE IS LESS THAN NUMBER OF ELECTRONS .....
285: C
286: C                else
287: C
288: C                .... REDUCE MULTIPLE PHOTONS TO 1 .....
289: C
290: C                NPPR = IMTTE0(2) - IMTTE0(1)
291: C                call RANU2( IRAND, R, NPPR, ICON )
292: C
293: C 294: *VOCL LOOP,NOVREC
295: C                do 140 I = IMTTE0(1) + 1, IMTTE0(2)
296: C                    if ( IWK34(I).gt.0.and.IWK34(I+NPPR).gt.0 ) then
297: C                        J = R(I) + 0.5
298: C                        J1 = NPPR*J + I
299: C                        J2 = NPPR*(1-J) + I
300: C                        IWK34(J1) = 0
301: C                        WK30(J2) = WK30(J2)*2.0
302: C                    end if
303: C 140            continue
304: C
305: C                NV = 0
306: C                NB = 0
307: C                do 150 I = 1, NELC
308: C                    if ( IWK34(I).gt.0 ) then
309: C                        NB = NB + 1
310: C                        WK30(I) = WK30(I)*IWK34(I)
311: C                        IWK34(I) = 1
312: C                        if ( IWK31(I).gt.0 ) NV = NV + 1
313: C                    end if
314: C 150            continue
315: C                if ( NB.gt.NDEAD ) then
316: C
317: C                NOTE: PHOTONS FROM DIED PARENT PHOTON HAVE NOT ROULETTED
318: C                TO REDUCE FOR BANK SPACE, NOW.
319: C
320: C                do 160 I = 1, NCOLP
321: C                    IWK36(LSCLP(I)) = I
322: C 160            continue
323: C
324: C                ND = NB - NDEAD
325: C                QV = NV

```

```

326: C                SV = (NV-ND) /QV
327: C                ..... SV : ESCAPE PROBABILITY FROM RUSSIAN-ROULETTE
328: C
329: C 170            SV = SV*0.95
330: C
331: C                call RANU2( IRAND, R, NELC, ICON )
332: C
333: C                JJ = 0
334: C                do 180 I = 1, NELC
335: C                    if ( IWK34(I).gt.0.and.IWK31(I).gt.0.and.R(I).gt.SV )
336: C                        & then
337: C                            JJ = JJ + 1
338: C                            IWK38(JJ) = I
339: C                        end if
340: C 180            continue
341: C                if ( JJ.lt.ND ) go to 170
342: C
343: C                call RANU2( IRAND, R, JJ, ICON )
344: C
345: C 345: *VOCL LOOP,NOVREC
346: C                do 190 J1 = 1, JJ
347: C                    I = IWK38(J1)
348: C                    IB = IWK31(I)
349: C                    BANKE = WK30(I) /(WK30(I)+WWW(IB))
350: C
351: C                ..... PARTICLES ARE ALIVE AS BREMSSTRAHLUNG PHOTON .....
352: C
353: C                if ( R(J1).lt.BANKE ) then
354: C                    J = IWK36(IB)
355: C                    LSCLP(J) = -IB
356: C                    WK30(I) = WK30(I) /BANKE
357: C                    IWK31(I) = -IB
358: C                    NKILP = NKILP + 1
359: C                    LSDDED(NDEAD+NKILP) = IB
360: C                    WCNTR(24,KPPHOT) = WCNTR(24,KPPHOT) + WWW(IB)
361: C
362: C                ..... PARTICLES ARE ALIVE AS PARENT PHOTON .....
363: C
364: C                else
365: C                    IWK34(I) = 0
366: C                    WWW(IB) = WWW(IB) /(1.0-BANKE)
367: C                end if
368: C 190            continue
369: C
370: C                NDEAD = NDEAD + NKILP
371: C                NCNTR(24,KPPHOT) = NCNTR(24,KPPHOT) + NKILP
372: C
373: C                if ( NKILP.gt.0 ) then
374: C                    J = 0
375: C 375: *VOCL LOOP,NOVREC
376: C                    do 200 I = 1, NCOLP
377: C                        if ( LSCLP(I).gt.0 ) then
378: C                            J = J + 1
379: C                            LSCLP(J) = LSCLP(I)
380: C                            UOUT(J) = UOUT(I)
381: C                        end if
382: C 200            continue
383: C                    NCOLP = J
384: C                end if
385: C            end if
386: C        end if
387: C    end if
388: C
389: C    .... FIND MAXIMUM PHOTONS GENERATED BY EACH REACTION
390: C

```

src/mvp/ttbr.f

```

391:      NM1      = 0
392:      NM2      = 0
393:      NM3      = 0
394:      NM4      = 0
395:      do 210 I = 1, IMTTE0(1)
396:         NM1      = MAX(IWK34(I),NM1)
397: 210 continue
398:      do 220 I = IMTTE0(1) + 1, IMTTE0(2)
399:         NM2      = MAX(IWK34(I),NM2)
400: 220 continue
401:      do 230 I = IMTTE0(2) + 1, IMTTE0(3)
402:         NM3      = MAX(IWK34(I),NM3)
403: 230 continue
404:      do 240 I = IMTTE0(3) + 1, IMTTE0(4)
405:         NM4      = MAX(IWK34(I),NM4)
406: 240 continue
407: C
408: C      .... NO PHOTON IS GENERATED, THEN RETURN
409: C
410:      if ( NM1+NM2+NM3+NM4.le.0 ) then
411:         go to 480
412:      end if
413: C
414: C-----
415: C DETERMINE THE GENERATED PHOTON ENERGY.
416: C-----
417: C
418: C      IWK35 : INDEX OF PARENT ELECTRON GENERATING PHOTONS.
419: C
420:      NN      = 0
421: C
422:      do 250 I = ND0 + 1, NDEAD
423:         IWK36(LSDED(I)) = I
424: 250 continue
425: C
426:      if ( NM1.ge.1 ) then
427: *VOCL LOOP,NOVREC
428:         do 260 I = 1, IMTTE0(1)
429:            if ( IWK34(I).ge.1 ) then
430:               NN      = NN + 1
431:               IWK35(NN) = I
432:               LSCLP(NCOLP+NN) = 0
433:               if ( IWK31(I).lt.0 ) then
434:                  IP      = ABS(IWK31(I))
435:                  LSCLP(NCOLP+NN) = IP
436:                  LSDED(IWK36(IP)) = 0
437:               end if
438:            end if
439: 260 continue
440:         do 280 J = 2, NM1
441:            do 270 I = 1, IMTTE0(1)
442:               if ( IWK34(I).ge.J ) then
443:                  NN      = NN + 1
444:                  IWK35(NN) = I
445:                  LSCLP(NCOLP+NN) = 0
446:               end if
447: 270 continue
448: 280 continue
449:            end if
450:            IMTTE1(1) = NN
451: C
452:            if ( NM2.ge.1 ) then
453: *VOCL LOOP,NOVREC
454:               do 290 I = IMTTE0(1) + 1, IMTTE0(2)
455:                  if ( IWK34(I).ge.1 ) then

```

```

456:                  NN      = NN + 1
457:                  IWK35(NN) = I
458:                  LSCLP(NCOLP+NN) = 0
459:                  if ( IWK31(I).lt.0 ) then
460:                     IP      = ABS(IWK31(I))
461:                     LSCLP(NCOLP+NN) = IP
462:                     LSDED(IWK36(IP)) = 0
463:                  end if
464:            end if
465: 290 continue
466:            do 310 J = 2, NM2
467:               do 300 I = IMTTE0(1) + 1, IMTTE0(2)
468:                  if ( IWK34(I).ge.J ) then
469:                     NN      = NN + 1
470:                     IWK35(NN) = I
471:                     LSCLP(NCOLP+NN) = 0
472:                  end if
473: 300 continue
474: 310 continue
475:            end if
476:            IMTTE1(2) = NN
477: C
478:            if ( NM3.ge.1 ) then
479: *VOCL LOOP,NOVREC
480:               do 320 I = IMTTE0(2) + 1, IMTTE0(3)
481:                  if ( IWK34(I).ge.1 ) then
482:                     NN      = NN + 1
483:                     IWK35(NN) = I
484:                     LSCLP(NCOLP+NN) = 0
485:                     if ( IWK31(I).lt.0 ) then
486:                        IP      = ABS(IWK31(I))
487:                        LSCLP(NCOLP+NN) = LSDED(IWK36(IP))
488:                        LSDED(IWK36(IP)) = 0
489:                     end if
490:                  end if
491: 320 continue
492:               do 340 J = 2, NM3
493:                  do 330 I = IMTTE0(2) + 1, IMTTE0(3)
494:                     if ( IWK34(I).ge.J ) then
495:                        NN      = NN + 1
496:                        IWK35(NN) = I
497:                        LSCLP(NCOLP+NN) = 0
498:                     end if
499: 330 continue
500: 340 continue
501:                  end if
502:                  IMTTE1(3) = NN
503: C
504:                  if ( NM4.ge.1 ) then
505: *VOCL LOOP,NOVREC
506:                     do 350 I = IMTTE0(3) + 1, IMTTE0(4)
507:                        if ( IWK34(I).ge.1 ) then
508:                           NN      = NN + 1
509:                           IWK35(NN) = I
510:                           LSCLP(NCOLP+NN) = 0
511:                           if ( IWK31(I).lt.0 ) then
512:                              IP      = ABS(IWK31(I))
513:                              LSCLP(NCOLP+NN) = IP
514:                              LSDED(IWK36(IP)) = 0
515:                           end if
516:                        end if
517: 350 continue
518:                     do 370 J = 2, NM4
519:                        do 360 I = IMTTE0(3) + 1, IMTTE0(4)
520:                           if ( IWK34(I).ge.J ) then

```

src/mvp/ttbr.f

```

521:          NN      = NN + 1
522:          IWK35(NN) = I
523:          LSCLP(NCOLP+NN) = 0
524:        end if
525: 360      continue
526: 370      continue
527:    end if
528:    IMTTE1(4) = NN
529: C
530:    if ( NN.gt.NDEAD ) then
531:      write(IOW,*) ' '
532:      write(IOW,*) ' **** ERROR MESSAGE FROM TTBR ****'
533:      write(IOW,*) ' NUMBER OF GENERATED BREMSSTRAHLUNG PHOTONS ',
534: & 'IS GREATER THAN BANK SPACE.'
535:      write(IOW,*) ' GENERATION = ', NN, ' : SPACE = ', NDEAD
536:      write(IOW,*) ' NO. OF ELECTRON GENERATING PHOTON = ', NB
537:      write(IOW,*) ' NO. OF ELECTRON WITH MULTI. PHOTONS= ', NY
538:      write(IOW,*) ' NO. OF KILLED PARENT PHOTONS = ', NKILP
539:      write(IOW,*) ' '
540:      stop 888
541:    end if
542: C
543:    J      = ND0
544: *VOCL LOOP,NOVREC
545:    do 380 I = ND0 + 1, NDEAD
546:      if ( LSDED(I).gt.0 ) then
547:        J      = J + 1
548:        LSDED(J) = LSDED(I)
549:      end if
550:    380 continue
551:    NDEAD = J
552: C
553: C
554: C
555: CM      IE      = 1
556: C
557: CM      IF( IE.EQ.2 ) GO TO 1000
558: C
559: C      .... CONDITIONAL DISCRETE SAMPLING (CDS), IE=1 .....
560: C
561:    MTOP1 = MTOP - 1
562:    call RANU2( IRAND, R, NN*4, ICON )
563:    N2      = NN*2
564:    N3      = NN*3
565: C
566:    do 390 I = 1, NN
567:      RR      = WK33(IWK35(I)) + R(I)
568:      M      = IWK32(IWK35(I)) + RR
569:      IMED    = IWK33(IWK35(I))
570:      LS      = MTOP1*R(NN+I) + 1
571: C/#IF ROUND OFF(NEAREST)
572:      M      = MIN(M,IWK32(IWK35(I))+1)
573:      LS      = MIN(LS,MTOP1)
574: C/#ENDIF
575: c97/11/07 LS2 = LS*2 + R(N2+I) - EBTP(LS,M,IMED)
576:      LS2    = LS*2
577:      LS2    = min(LS2,int(LS2 + R(N2+I) - EBTP(LS,M,IMED)))
578:      LS3    = IEFTP(LS2,M,IMED)
579: C
580: C      EBTX : PROBABILITY FOR K/T RATIO SEARCH.
581: C      RKT1 : K/T RATIO.
582: C      WK33 : ENERGY OF BREMSSTRAHLUNG PHOTONS.
583: C
584:      EBT1 = EBT(LS3,M,IMED)
585:      EBT2 = EBT(LS3+1,M,IMED)

```

```

586: ccccccc EBTX      = EBT1 * R(N3+I) + EBT2 * (1.0 - R(N3+I))
587:      EBTX      = EBT2 + (EBT1-EBT2)*R(N3+I)
588:      EBTA      = (EBT1+EBT2) /2.0
589:      if ( LS3.eq.1 .or. EBTX.gt.EBTA ) then
590:        L      = LS3
591:        EBT3    = EBT(L+2,M,IMED)
592:      else
593:        L      = LS3 - 1
594:        EBT3    = EBT2
595:        EBT2    = EBT1
596:        EBT1    = EBT(L,M,IMED)
597:      end if
598:      RKT1      = (EBTX-EBT2)*(EBTX-EBT3)*RKT(L) /
599: & ((EBT1-EBT2)*(EBT1
600: & -EBT3)) + (EBTX-EBT1)*(EBTX-EBT3)*RKT(L+1) /
601: & ((EBT1-EBT2)*(EBT3
602: & -EBT2)) + (EBTX-EBT1)*(EBTX-EBT2)*RKT(L+2) /
603: & ((EBT3-EBT1)*(EBT3-EBT2))
604:      WK34(I) = WK31(IWK35(I))*RKT1
605: 390 continue
606: C
607: CM      GO TO 2000
608: C
609: C      .... MCNP METHOD, IE=2 ....
610: C
611: C1000 CONTINUE
612: C
613: C      RKT1 : K/T RATIO AT ENERGY BIN N.
614: C      RKT2 : K/T RATIO AT ENERGY BIN N+1.
615: C      RKT3 : K/T RATIO AT ELECTRON ENERGY.
616: C      WK34 : ENERGY OF BREMSSTRAHLUNG PHOTONS.
617: C
618: CM      CALL RANU2 ( IRAND, R, NN, ICON )
619: C
620: C      DO 620 I=1,NN
621: C      RKT1 = QPOL( R(I),
622: C & EBT(1,IWK32(IWK35(I)),IWK33(IWK35(I))),
623: C & RKT, MTOP )
624: C      RKT2 = QPOL( R(I),
625: C & EBT(1,IWK32(IWK35(I))+1,IWK33(IWK35(I))),
626: C & RKT, MTOP )
627: C      RKT3 = (1.0 - WK33(IWK35(I))) * RKT1 +
628: C & WK33(IWK35(I)) * RKT2
629: C      WK34(I) = WK31(IWK35(I)) * RKT3
630: C 620 CONTINUE
631: C
632: C2000 CONTINUE
633: C
634: C
635: C      .... CHECK PHOTON ENERGY ....
636: C
637: C      IWK38 : INDEX OF NEW PARENT ELECTRON.
638: C
639:      do 400 I = 1, 4
640:        IMTTE0(I) = 0
641: 400 continue
642: C
643: C      NP      = 0
644: C      NP1     = 0
645: C      NP3     = 0
646: C      ND0     = 0
647: C      ND      = NDEAD + 1
648: C
649: *VOCL LOOP,NOVREC
650:      do 410 I = 1, NN

```


src/mvp/ttbr.f

```

651:      IP      = LSCLP(NCOLP+I)
652:      if ( WK34(I).gt.EBOTP ) then
653:        if ( IP.le.0 ) then
654:          ND      = ND - 1
655:          IP      = LSDED(ND)
656:        end if
657:        NP      = NP + 1
658:        EEE(IP) = WK34(I)
659:        WWW(IP) = WK30(IWK35(I))
660:        IWK34(NP) = IWK31(IWK35(I))
661:        IWK38(NP) = IWK35(I)
662:        JOUT(NCOLP+NP) = WK32(IWK35(I))
663:        LSCLP(NCOLP+NP) = IP
664: C
665:        WCNTR(2,KPPHOT) = WCNTR(2,KPPHOT) + WWW(IP)
666: C
667:        if ( I.le.IMTE1(1) ) NP1 = NP1 + 1
668:        if ( I.le.IMTE1(3) ) NP3 = NP3 + 1
669:      else
670: C*
671:        WCNTR(8,2) = WCNTR(8,2) + WK30(IWK35(I))
672:        if ( IP.gt.0 ) then
673:          ND0      = ND0 + 1
674:          IWK36(ND0) = IP
675:        end if
676:      end if
677: C
678:      NCNTR(2,KPPHOT) = NCNTR(2,KPPHOT) + NP
679: C
680:      NDEAD = ND - 1
681:      do 420 I = 1, ND0
682:        NDEAD = NDEAD + 1
683:        LSDED(NDEAD) = IWK36(I)
684:      420 continue
685: C
686: C=====
687: C
688: C STORE TO THE PARTICLE BANK EXCEPT EEE, WWW, IGG ETC.
689: C      EEE(IP) ENERGY
690: C      WWW(IP) WEIGHT
691: C      (NOTE: Energy group (IGG) is set in subroutine PHOTR.)
692: C      ( Direction (AAA,BBB,CCC) is that of collided photon )
693: C
694: C=====
695: C
696: C
697: *VOCL LOOP,NOVREC
698:   do 430 I = 1, NP
699:     IP      = LSCLP(NCOLP+I)
700:     IB      = ABS(IWK34(I))
701:     XXX(IP) = XXX(IB)
702:     YYY(IP) = YYY(IB)
703:     ZZZ(IP) = ZZZ(IB)
704:     AAA(IP) = AAA(IB)
705:     BBB(IP) = BBB(IB)
706:     CCC(IP) = CCC(IB)
707:     TTT(IP) = TTT(IB)
708:     IZZ(IP) = IZZ(IB)
709:     KKP(IP) = KPPHOT
710:     KLSF(IP) = 0
711:     if ( JLATT.ne.0 ) LEVL(IP) = LEVL(IB)
712:     if ( JIMPT.ne.0 ) XIM(IP) = XIM(IB)
713:     CCCC   if ( JTLLT.ne.0 ) IBREG(IP) = IBREG(IB)
714:     IBREG(IP) = IBREG(IB)
715:     if ( JTIME.ne.0 ) ITT(IP) = ITT(IB)

```

```

716: C
717:   430 continue
718: C
719:   if ( JLATT.ne.0 ) then
720:     do 450 K = 1, NEST
721:   *VOCL LOOP,NOVREC
722:     do 440 I = 1, NP
723:       IP      = LSCLP(NCOLP+I)
724:       IB      = ABS(IWK34(I))
725:       LZZ(IP,K) = LZZ(IB,K)
726:       LPOS(IP,K) = LPOS(IB,K)
727:       if ( JHLAT.ne.0 ) LCRS(IP,K) = LCRS(IB,K)
728:       if ( JTLLT.ne.0 ) IBSPC(IP,K) = IBSPC(IB,K)
729:     440 continue
730:   450 continue
731:   end if
732: C
733: C ..... COPY PARTICLE BANK DATA OF DIRECTION FOR PAIR PRODUCTION ...
734: C
735: C
736:   if ( NP3-NP1.gt.0 ) then
737:     do 460 I = NP1 + 1, NP3
738:       IP      = LSCLP(NCOLP+I)
739:       IB      = IWK38(I) - IIBASE
740:       AAA(IP) = WK41(IB)
741:       BBB(IP) = WK42(IB)
742:       CCC(IP) = WK43(IB)
743:   460 continue
744:       IMTTE0(1) = NCOLP + NP1 + 1
745:       IMTTE0(2) = NCOLP + NP3
746:     end if
747: C
748: C ..... TRANSFER COEFFICIENCIES FOR CALCULATING DIRECTION OF
749: C COMPTON SCATTERING .....
750: C
751: C      IWK38 : STORAGE OF PARENT PHOTON BANK INDEX FOR
752: C DIRECTION OF COMPTON SCATTERING
753: C
754:   if ( NP-NP3.gt.0 ) then
755:     NS      = 0
756:   *VOCL LOOP,NOVREC
757:     do 470 I = NP3 + 1, NP
758:       NS      = NS + 1
759:       IB      = IWK38(I) - IIBASE
760:       WK41(NS) = WK41(IB)
761:       WK42(NS) = WK42(IB)
762:       WK43(NS) = WK43(IB)
763:       IWK38(NS) = IWK34(I)
764:   470 continue
765:       IMTTE0(3) = NCOLP + NP3 + 1
766:       IMTTE0(4) = NCOLP + NP
767:     end if
768: C
769:     NELC      = NP
770:     NCOLP     = NCOLP + NP
771: C
772:     return
773: C
774: C .... NO BREMSSTRAHLUNG PHOTON IS GENERATED
775: C
776:   480 continue
777:   NELC      = 0
778:   do 490 I = 1, 4
779:     IMTTE0(I) = 0
780:   490 continue

```

src/mvp/ttbr.f

781: return
782: C
783: end



src/mvp/verstr.f

```
1:      subroutine VERSTR
2: C=====
3: C Purpose: set version string of printout headers
4: C
5: C=====
6: Ccc  CALL HVERSN( 'MVP 94.1 ' )
7: Ccc  CALL HVERSN( 'MVP 94.1a ' )
8: C
9: C ... 94.2betan : 94.2 beta n
10: C    call HVERSN( 'MVP 94.2beta1' )
11: C    call HVERSN( 'MVP 96.1beta1' )
12: C ... 96.1beta3 from 18 May 1997
13: C    call HVERSN( 'MVP 96.1beta3' )
14: Cccc 96.1beta4 from 22 Oct 1997
15: C    call HVERSN( 'MVP 96.1beta4.pre' )
16: Cccc 96.1beta4pre2 from 16 Nov 1997
17: C    call HVERSN( 'MVP 96.1beta4.pre2' )
18: C
19: C .... beta 4 patch 3 (Mar 3 1998)
20: C .... beta 4 patch 4 (Mar 23 1998)
21: C    call HVERSN( 'MVP 96.1beta4p4' )
22: C .... version 2.0 preliminary (Apr 1998)
23: C    call HVERSN( 'MVP v2.0pre' )
24: C ... version 2.0 preliminary1 (14 May 1998)
25: C    call HVERSN( 'MVP v2.0pre1' )
26: C .... version 2.0 preliminary2 ( 8 June 1998)
27: C    call HVERSN( 'MVP v2.0pre2' )
28: C
29: C .... version 2.0 beta (22 June 1998)
30: C
31: CC    call HVERSN( 'MVP v2.0beta' )
32: C
33: C .... version 2.0 beta2 ( ? 1998)
34: C
35: C    call HVERSN( 'MVP v2.0beta2-pre' )
36: C
37: C .... version 2.0 beta2 (Jul 1998)
38: CCC   call HVERSN( 'MVP v2.0beta2' )
39: C
40: C .... version 2.0 beta2.5 (25 Aug 1998)
41: C    call HVERSN( 'MVP v2.0beta2.5' )
42: C
43: C    call HVERSN( 'MVP v2.0beta3-pre' )
44: C
45: C ... 9 Sep 1998 ...
46: C    call HVERSN( 'MVP v2.0beta3-pre1' )
47: C
48: C ... 5 Oct 1998 : version regression to beta 3.0
49: C
50: C    call HVERSN( 'MVP v2.0beta3.0' )
51: C
52: C ... 14 Oct 1998 : version regression to beta 3.01
53: C
54: C    call HVERSN( 'MVP v2.0beta3.01' )
55: C
56: C ... 6 Nov 1998 : version beta 3.05
57: C
58: C    call HVERSN( 'MVP v2.0beta3.05' )
59: C
60: C ... 17 Nov 1998 : version beta 3.06
61: C ... 18 Nov 1998 : version beta 3.07
62: C
63: C    call HVERSN( 'MVP v2.0beta3.07' )
64: C
65: C ... 11 Dec 1998 : version beta 3.08
66: C ... 16 Dec 1998 : version beta 3.09
67: C
68: C    call HVERSN( 'MVP v2.0beta4.0pre' )
69: C
70: C ... 11 Jan 1999 : v2.0beta4.0
71: C    call HVERSN( 'MVP v2.0beta4.0' )
72: C ... 20 Jan 1999 : v2.0beta4.01
73: C    call HVERSN( 'MVP v2.0beta4.01' )
74: C ... 1 feb 1999 : v2.0beta4.02
75: C    call HVERSN( 'MVP v2.0beta4.02', ' 1 February 1999' )
76: C ... 10 feb 1999 : v2.0beta4.03
77: C    call HVERSN( 'MVP v2.0beta4.03', '10 February 1999' )
78: C
79: C    call HVERSN( 'MVP v2.0beta4.04', '17 April 1999' )
80: C
81: C    call HVERSN( 'MVP v2.0beta4.05', '25 April 1999' )
82: C    call HVERSN( 'MVP v2.0beta4.06', '10 May 1999' )
83: C    call HVERSN( 'MVP v2.0beta4.07', '17 May 1999' )
84: C    call HVERSN( 'MVP v2.0beta4.08', '25 May 1999' )
85: C    call HVERSN( 'MVP v2.0beta4.09', '21 June 1999' )
86: C    call HVERSN( 'MVP v2.0beta4.10', ' 7 July 1999' )
87: C    call HVERSN( 'MVP v2.0beta4.11', ' 9 August 1999' )
88: C    call HVERSN( 'MVP v2.0beta4.12', ' 7 October 1999' )
89: C    call HVERSN( 'MVP v2.0beta4.13', '22 October 1999' )
90: C    call HVERSN( 'MVP v2.0beta4.14pre', '22 October 1999' )
91: C    call HVERSN( 'MVP v2.0beta4.14', ' 9 November 1999' )
92: C    call HVERSN( 'MVP v2.0beta4.15pre', ' 9 November 1999' )
93: C    call HVERSN( 'MVP v2.0beta4.15', '22 November 1999' )
94: C    call HVERSN( 'MVP v2.0beta4.16', '24 December 1999' )
95: C    call HVERSN( 'MVP v2.0beta4.17pre', '24 December 1999' )
96: C    call HVERSN( 'MVP v2.0beta4.17pre1', '20 February 2000' )
97: C    call HVERSN( 'MVP v2.0beta4.17', ' 3 April 2000' )
98: C    call HVERSN( 'MVP v2.0beta4.18pre', ' 4 April 2000' )
99: C    call HVERSN( 'MVP v2.0beta4.18', '30 April 2000' )
100: C    call HVERSN( 'MVP v2.0beta5', '13 July 2000' )
101: C    call HVERSN( 'MVP v2.0beta6.00', '18 September 2000' )
102: C    call HVERSN( 'MVP v2.0beta6.01', '13 October 2000' )
103: C    call HVERSN( 'MVP v2.0beta6.02', '24 Dec 2000' )
104: C    call HVERSN( 'MVP v2.0beta6.03', '28 Dec 2000' )
105: C    call HVERSN( 'MVP v2.0beta6.04', '30 Jan 2001' )
106: C    call HVERSN( 'MVP v2.0beta6.05', ' 04 Jun 2001' )
107: C    call HVERSN( 'MVP v2.0beta6.06', ' 05 Jul 2001' )
108: C    call HVERSN( 'MVP v2.0beta6.07', ' 07 Sep 2001' )
109: C    call HVERSN( 'MVP v2.0beta6.08', '11 Sep 2001' )
110: C    call HVERSN( 'MVP v2.0beta6.09', ' 09 Dec 2001' )
111: C    call HVERSN( 'MVP v2.0beta6.10', ' 07 Jan 2002' )
112: C    call HVERSN( 'MVP v2.0beta6.11', '22 Jan 2002' )
113: C    call HVERSN( 'MVP v2.0beta6.12', ' 05 Feb 2002' )
114: C    call HVERSN( 'MVP v2.0beta6.13', '12 Feb 2002' )
115: C    call HVERSN( 'MVP v2.0beta6.14', '28 Feb 2002' )
116: C    call HVERSN( 'MVP v2.0beta6.15', ' 08 Mar 2002' )
117: C    call HVERSN( 'MVP v2.0beta6.16', '10 Jun 2002' )
118: C    call HVERSN( 'MVP v2.0beta6.17', '13 Aug 2002' )
119: C    call HVERSN( 'MVP v2.0beta6.18', '21 Nov 2002' )
120: C    call HVERSN( 'MVP v2.0beta6.19', '25 Jul 2003' )
121: C    call HVERSN( 'MVP v2.0beta6.20', '19 Aug 2003' )
122: C    call HVERSN( 'MVP v2.0beta6.21', '28 Sep 2003' )
123: C    call HVERSN( 'MVP v2.0beta6.22', '26 Nov 2003' )
124: C    call HVERSN( 'MVP v2.0beta6.23', '10 Feb 2004' )
125: C    call HVERSN( 'MVP v2.0beta6.24', ' 07 Apr 2004' )
126: C    call HVERSN( 'MVP v2.0beta6.25', '15 Nov 2004' )
127: C    call HVERSN( 'MVP v2.0beta6.26', ' 09 Jan 2005' )
128: C    call HVERSN( 'MVP v2.0beta6.27', ' 02 Mar 2005' )
129: C    call HVERSN( 'MVP v2.0beta6.28', '11 Jul 2005' )
130: C    call HVERSN( 'MVP v2.0beta6.29', ' 07 Sep 2005' )
```

src/mvp/verstr.f

```
131: C      call HVERSN( 'MVP 2.0', '11 Nov 2005' )
132: C      call HVERSN( 'MVP 2.0.1', '06 Feb 2006' )
133: C      call HVERSN( 'MVP 2.0.2', '27 Feb 2006' )
134: C      call HVERSN( 'MVP 2.0.3', '06 Mar 2006' )
135: C      call HVERSN( 'MVP 2.0.4', '07 Apr 2006' )
136: C      call HVERSN( 'MVP 2.0.5', '13 Apr 2006' )
137: C      call HVERSN( 'MVP 2.0.6', '28 Apr 2006' )
138: C      call HVERSN( 'MVP 2.0.7', '02 May 2006' )
139: C      call HVERSN( 'MVP 2.0.8', '15 May 2006' )
140: C      call HVERSN( 'MVP 2.0.9', '09 Jun 2006' )
141: C      call HVERSN( 'MVP 2.0.10', '19 Jun 2006' )
142: C      call HVERSN( 'MVP 2.0.11', '15 Aug 2006' )
143: C      call HVERSN( 'MVP 2.0.12', '05 Sep 2006' )
144: C      call HVERSN( 'MVP 2.0.13', '05 Oct 2006' )
145: C      call HVERSN( 'MVP 2.0.14', '06 Nov 2006' )
146: C      call HVERSN( 'MVP 2.0.15', '02 Feb 2007' )
147: C      call HVERSN( 'MVP 2.0.16', '05 Feb 2007' )
148: C      call HVERSN( 'MVP 2.0.17', '09 Oct 2007' )
149: C      call HVERSN( 'MVP 2.0.18', '11 Oct 2007' )
150: C      call HVERSN( 'MVP 2.0.19', '31 Oct 2007' )
151: C      call HVERSN( 'MVP 2.0.20', '10 Apr 2009' )
152: C      call HVERSN( 'MVP 2.0.21', '18 Jul 2009' )
153: C      call HVERSN( 'MVP 2.0.22', '06 Oct 2009' )
154: C      call HVERSN( 'MVP 2.0.23', '18 Nov 2009' )
155: C      call HVERSN( 'MVP 2.0.24', '03 Mar 2010' )
156: C      call HVERSN( 'MVP 2.0.25', '09 Jul 2010' )
157: C      call HVERSN( 'MVP 2.0.26', '25 Aug 2011' )
158: C      call HVERSN( 'MVP 2.0.27', '15 Dec 2011' )
159: C      call HVERSN( 'MVP 2.0.28', '23 Jan 2013' )
160: C      call HVERSN( 'MVP 2.0.29', '12 Mar 2013' )
161: C      call HVERSN( 'MVP 2.0.30', '20 Dec 2013' )
162: C      call HVERSN( 'MVP 2.0.31', '11 Mar 2015' )
163: C      call HVERSN( 'MVP 2.0.32', '24 Mar 2015' )
164: C      call HVERSN( 'MVP 2.0.33', '07 Oct 2015' )
165: C      call HVERSN( 'MVP 2.0.34', '29 Sep 2016' )
166: C      call HVERSN( 'MVP 2.0.35', '04 Oct 2016' )
167: C      call HVERSN( 'MVP 2.0.36', '14 Oct 2016' )
168: C      call HVERSN( 'MVP 2.0.37', '24 Oct 2016' )
169: C      call HVERSN( 'MVP 2.0.38', '09 Nov 2016' )
170: C      call HVERSN( 'MVP 2.0.39', '20 Jan 2017' )
171: C      call HVERSN( 'MVP 3.0', '31 Jan 2017' )
172: C
173:      return
174:      end
```

src/mvp/vmntr.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine VMNTR( NS,      IOW )
3: C=====
4: C  PURPOSE:   MONITORING OF CPU & VECTOR CALCULATION.
5: C  CALLED IN: SOURCE,SELECT,FLIGHT,SEARCH,COLISN,MIRROR
6: C  CALLS:     CLOCK
7: C
8: C=====
9:      include 'INC/_NMON'
10:     parameter( NMONSQ = NMON*NMON )
11:     character*6 SUBS(NMON)
12:     real CPU(NMON,NMON)
13:     integer NEX(NMON,NMON), NV(NMON)
14:     real VL(NMON)
15:     real VL2(NMON)
16:     integer MINVL(NMON), MAXVL(NMON)
17:     real SCPU(NMON,NMON), SCPU2(NMON)
18: C/#IF SYSTEM(ACOS SX*)
19: *     real*8 T0(NMON), T1(NMON)
20: C/#ELSE
21:     real T0(NMON), T1(NMON)
22: C/#ENDIF
23: C/#IF SYSTEM(DECOSF* PARAGON*)
24: *     external ETIME
25: *     real TARRAY(2)
26: C/#ENDIF
27: C
28: C  CPU(I,J)  R4  CPU TIME OF SUBROUTINE-I CALLED FROM SUBROUTINE-J.
29: C  NEX(I,J)  I4  NUMBER OF EXECUTION OF SUBROUTINE
30: C  NV (I)    I4  TOTAL NUMBER OF SUBROUTINE CALL
31: C  VL (I)    R4  SUM OR AVERAGE OF PARTICLES PROCESSD PER CALL
32: C  VL2(I)    R4  SQUARE SUM OR STANDARD DEVIATION OF VL
33: C  MIMVL(I)  I4  MINIMUM VECTOR LENGTH
34: C  MAXVL(I)  I4  MAXIMUM VECTOR LENGTH
35: C
36: C  SCPU(I,J) R4  CPU TIME FOR CPU-MONITOR CALLS      (SEC/CALL)
37: C              ( VMNTR0, VMNTR2 , VMNT00, VMNT22 )
38: C  SCPU2(I)  R4  CPU TIME FOR VECTER LENGTH MONITOR (SEC/CALL)
39: C              ( VMNTR1 )
40: C
41:     data SCPU /NMONSQ*0.0/
42:     data SCPU2 /NMON*0.0/
43:     data CPU /NMONSQ*0.0/
44:     data NEX /NMONSQ*0/
45:     data NV /NMON*0/
46:     data VL /NMON*0.0/
47:     data VL2 /NMON*0.0/
48:     data MINVL /NMON*9999999/
49:     data MAXVL /NMON*0/
50:     data T0 /NMON*0.0/
51:     data T1 /NMON*0.0/
52: C
53:     data SUBS /'SOURCE', 'SELECT', 'FLIGHT', 'SEARCH', 'NEUTR ',
54: &             'MIRROR', 'LATICE', 'GETMIC', 'GETMAC', 'TALLY ',
55: &             'LFRAME', 'PHOTR '/
56: C
57:     return
58: C
59: C ..... CPU TIME MONITORING .....
60: C
61:     entry VMNTR0(NS,IOW)
62: C/#IF SYSTEM(FACOM*)
63: *     call CLOCK(T0(NS),0,1)
64: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
65: C     call CPUTIME(T0(NS))

```

```

66: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
67: *     T0(NS) = ETIME( TARRAY )
68: C/#ELSEIF SYSTEM(ACOS SX*)
69: *     call CLOCK(T0(NS))
70: C/#ELSEIF SYSTEM(CRAY*)
71: *     call SECOND(T0(NS))
72: C/#ELSEIF SYSTEM(IBMRS* AIX*)
73: *     T0(NS) = MCLOCK()/100.0
74: C/#ELSE
75: C     T0(NS) = 0.0
76: *     call CPUTM(T0(NS))
77: C/#ENDIF
78:     NEX(NS,NS) = NEX(NS,NS) + 1
79:     return
80: C
81:     entry VMNTR2(NS)
82: C
83: C/#IF SYSTEM(FACOM*)
84: *     call CLOCK(T1(NS),0,1)
85: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
86: C     CALL cputime(T1(NS))
87: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
88: *     T1(NS) = ETIME( TARRAY )
89: C/#ELSEIF SYSTEM(ACOS SX*)
90: *     call CLOCK(T1(NS))
91: C/#ELSEIF SYSTEM(CRAY*)
92: *     call SECOND(T1(NS))
93: C/#ELSEIF SYSTEM(IBMRS* AIX*)
94: *     T1(NS) = MCLOCK()/100.0
95: C/#ELSE
96: *     call CPUTM(T1(NS))
97: C/#ENDIF
98:     CPU(NS,NS) = CPU(NS,NS) + T1(NS) - T0(NS)
99: C
100:     return
101: C
102: C ..... CPU MONITORING OF ROUTINE-NS CALLED FROM ROUTINE-NU ...
103: C
104: C
105:     entry VMNT00(NS,NU)
106: C
107: C/#IF SYSTEM(FACOM*)
108: *     call CLOCK(T1(NU),0,1)
109: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
110: C     CALL cputime(T1(NU))
111: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
112: *     T1(NU) = ETIME( TARRAY )
113: C/#ELSEIF SYSTEM(ACOS SX*)
114: *     call CLOCK(T1(NU))
115: C/#ELSEIF SYSTEM(CRAY*)
116: *     call SECOND(T1(NU))
117: C/#ELSEIF SYSTEM(IBMRS* AIX*)
118: *     T1(NU) = MCLOCK()/100.0
119: C/#ELSE
120: *     call CPUTM(T1(NU))
121: C/#ENDIF
122: C
123:     CPU(NU,NU) = CPU(NU,NU) + T1(NU) - T0(NU)
124: C
125: C/#IF SYSTEM(FACOM*)
126: *     call CLOCK(T0(NS),0,1)
127: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
128: C     CALL cputime(T0(NS))
129: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
130: *     T0(NS) = ETIME( TARRAY )

```

src/mvp/vmnr.f

```

131: C/#ELSEIF SYSTEM(ACOS SX*)
132: *      call CLOCK(T0(NS))
133: C/#ELSEIF SYSTEM(CRAY*)
134: *      call SECOND(T0(NS))
135: C/#ELSEIF SYSTEM(IBMRS* AIX*)
136: *      T0(NS) = MCLOCK()/100.0
137: C/#ELSE
138: *      call CPUTM(T0(NS))
139: C/#ENDIF
140:      NEX(NS,NU) = NEX(NS,NU) + 1
141:      return
142: C
143:      entry VMNT22(NS,NU)
144: C
145: C/#IF SYSTEM(FACOM*)
146: *      call CLOCK(T1(NS),0,1)
147: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
148: C      call CPUTIME(T1(NS))
149: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
150: *      T1(NS) = ETIME( TARRAY )
151: C/#ELSEIF SYSTEM(ACOS SX*)
152: *      call CLOCK(T1(NS))
153: C/#ELSEIF SYSTEM(CRAY*)
154: *      call SECOND(T1(NS))
155: C/#ELSEIF SYSTEM(IBMRS* AIX*)
156: *      T1(NS) = MCLOCK()*100.0
157: C/#ELSE
158: *      call CPUTM(T1(NS))
159: C/#ENDIF
160:
161:      CPU(NS,NU) = CPU(NS,NU) + T1(NS) - T0(NS)
162:
163: C/#IF SYSTEM(FACOM*)
164: *      call CLOCK(T0(NU),0,1)
165: CC/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
166: C      call CPUTIME(T0(NU))
167: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
168: *      T0(NU) = ETIME( TARRAY )
169: C/#ELSEIF SYSTEM(ACOS SX*)
170: *      call CLOCK(T0(NU))
171: C/#ELSEIF SYSTEM(CRAY*)
172: *      call SECOND(T0(NU))
173: C/#ELSEIF SYSTEM(IBMRS* AIX*)
174: *      T0(NU) = MCLOCK()/100.0
175: C/#ELSE
176: *      call CPUTM(T0(NU))
177: C/#ENDIF
178:
179:      return
180: C
181: C*****... VECTOR LENGTH MONITORING .....
182: C ..... PARTICLES-PER-CALL MONITORING .....
183: C
184:      entry VMNTR1(NS,NN)
185:      NV(NS) = NV(NS) + 1
186:      VL(NS) = VL(NS) + NN
187:      VL2(NS) = VL2(NS) + NN*NN
188:      MINVL(NS) = MIN(MINVL(NS),NN)
189:      MAXVL(NS) = MAX(MAXVL(NS),NN)
190:      return
191: C
192: C
193: C ***** CLEAR ALL MONITOR ARRAYS *****
194: C
195:      entry VMNTRC

```

```

196: C
197:      do 110 I = 1, NMON
198:      do 100 J = 1, NMON
199:          CPU(I,J) = 0.0
200:          NEX(I,J) = 0
201:      100 continue
202:          NV(I) = 0
203:          VL(I) = 0.0
204:          VL2(I) = 0.0
205:          MINVL(I) = 99999999
206:          MAXVL(I) = 0
207:      110 continue
208:      return
209: C
210: C ***** GIVE SCPU A VALUE *****
211: C
212:      entry VMNTRG(TT,NS,NU)
213:      SCPU(NS,NU) = TT
214:      return
215: C
216:      entry VMNTRH(TT2,NS)
217:      SCPU2(NS) = TT2
218:      return
219: C
220: C ***** TAKE A VALUE FROM CPU *****
221: C
222:      entry VMNTRT(TT,NS,NU)
223:      TT = CPU(NS,NU)
224:      return
225: C
226: C
227: C
228: C ..... PRINT OUT MONITORING INFORMATION .....
229: C
230: C
231:      entry VMNTRF(IOW)
232: C
233: C ..... CPU SUM FOR EACH ROTINE ....
234: C
235:      call LABEL( IOW, 'CPU TIME FOR EVENTS' )
236:      write(IOW,7000)
237: 7000 format(5X,'(CPU TIME FOR MONITOR ROUTINE CALLS IS REMOVED)'/1X,
238: &          'EVENT CPU(SEC) CALL CPU/CALL CPU / ',
239: &          ' AV.PARTCL. STANDARD MAX.PARTICLE MIN.PARTICLE'/1X,
240: &          ' (SEC) PARTICLE ',
241: &          ' NUMBER DEVIATION NUMBER NUMBER'/)
242: C
243: 7020 format(1X,A6,1PE12.4,I10,E12.4,E12.4,2X,E11.4,E11.4,3X,I6,7X,I6)
244: C
245:      TCPU = 0.0
246: *VOCL LOOP,SCALAR
247:      do 140 N = 1, NMON
248:          CPUN = 0.0
249:          NEXN = 0
250: C
251:      do 120 K = 1, NMON
252:          CPUN = CPUN + CPU(N,K)
253:          NEXN = NEXN + NEX(N,K)
254:      120 continue
255: C
256:      if ( CPUN.ne.0.0 ) then
257:          CPUN = CPUN - SCPU2(N)*NV(N)
258:          do 130 K = 1, NMON
259:              CPUN = CPUN - NEX(N,K)*SCPU(N,K)
260:          130 continue

```

src/mvp/vmntr.f

```
261: CCCCCCCCCCCC IF( CPUN.LT. 0.0 ) CPUN = 0.0
262:         TCPU      = TCPU + CPUN
263:         end if
264: C
265:         if ( VL(N).ne.0.0 ) then
266:             CPUP      = CPUN/VL(N)
267:         else
268:             CPUP      = 0.0
269:         end if
270: C
271:         if ( NV(N).gt.0 ) VL(N) = VL(N) /NV(N)
272:         if ( NV(N).gt.1 ) VL2(N) =
273: &      SQRT(ABS((VL2(N)-NV(N)*VL(N)**2))/(NV(N)-1))
274: C
275:         if ( NV(N).eq.1 ) VL2(N) = 0.0
276: C
277:         CN      = 0.0
278:         if ( NV(N).eq.0 ) then
279:             MAXVL(N) = 0
280:             MINVL(N) = 0
281:         end if
282:         if ( NEXN.ne.0 ) CN = CPUN/NEXN
283: C
284:         write(IOW,7020) SUBS(N), CPUN, NEXN, CN, CPUP, VL(N), VL2(N),
285: &      MAXVL(N), MINVL(N)
286: 140 continue
287: C
288:         write(IOW,7040) NMON, TCPU
289: 7040 format(/1X,' == CPU TOTAL FOR THESE ',I2,' EVENTS :',1PE11.4,
290: &      ' SEC')
291: C
292: C
293: C
294:         call LABEL( IOW, 'CPU TIME PERCENTAGE ( WITH CALLING-ROUTINES )' )
295: C
296:         write(IOW,7060) 'ACTION', (SUBS(J),J=1,NMON)
297: 7060 format(1X,' ROUTINE      CPU%           CALLING-ROUTINE '/1X,
298: &      ' ',14(1X,A6:))
299: C
300:         TTCPU      = TCPU
301:         if ( TCPU.eq.0.0 ) TTCPU      = 1.0
302: C
303:         do 160 I = 1, NMON
304:             CPUN      = 0.0
305:             CPUI      = 0.0
306:             if ( TCPU.ne.0 ) then
307:                 do 150 J = 1, NMON
308:                     CPUN      = CPUN + CPU(I,J)
309: 150             continue
310:                     CPUN      = CPUN/TTCPU*100.0
311:                     CPUI      = CPU(I,I) /TTCPU*100.0
312:                 end if
313:                 CPU(I,I)      = 0.0
314: C
315:                 write(IOW,7080) SUBS(I), CPUN, CPUI,
316: &      (CPU(I,J)/TTCPU*100.0,J=1,NMON)
317: 7080 format(1X,1X,A6,2X,F6.2,1X,14(1X,F6.2:)/1X,1X,6X,2X,6X,1X,14
318: &      (1X,F6.2:))
319:             160 continue
320: C
321:         return
322:         end
```

src/mvp/wgtfdinp.f

```

1:      subroutine WGTFDINP(IMG, A, IA, LIMIT, LAST, JDEBG, NREG, LWGTF,
2:      &                      JTLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM,
3:      &                      ISUSP, KSUZ, IRGSP, NEST, NSPACE, NSUZON,
4:      &                      NZONE, TNAMS, NNAMES, ITRNM, NTREG, IPTRG,
5:      &                      LPTRG, KR, NRRRO )
6: C=<MVP>=====
7: C purpose:  input array data for secondary generation weight of each
8: C           region for prompt and delayed neutron produced by neutron-
9: C           induced fission.
10: C called in:  INTRO
11: C calls:
12: C=====
13:      include '../shared/INC/_LNAM'
14: C
15:      real A(LIMIT)
16:      integer IA(LIMIT), JDEBG(*)
17: C
18: C ..... data for region name matching .....
19:      integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ),
20:      &          ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
21:      &          KSUZ(NINPZ,2), IRGSP(2,NREG)
22:      integer ITRNM(NTREG), IPTRG(NTREG+1), LPTRG(*), KR(NRRRO)
23:      character TNAMS(NNAMES)*(LNAM)
24: C
25: C ..... local variable .....
26:      character LINE*72, CHAR*4
27: C
28: C-----
29: C
30: C ..... check and initialize .....
31:      if ( NREG.le.0 ) go to 901
32:      JSKIP = 0
33:      NN = NREG * 2
34:      if ( LWGTF.eq.0 )
35:      & call KEVP( A(1), 'WGTF', LWGTF, NN, 'R4', LAST, 1.0, JDEBG ) ! defa
ult = 1.0
36: CM2016
37:      call GTLAST( LASTTM )
38:      call KEEP( 'WGTFD', LTEMP, NN, 'R4', LAST, JDEBG )
39: C
40: C =====
41: C region-wise form: WGTFD( !region-name( prompt_weight delayed_weight ) ...
)
42: C           WGTFD( !region-name( prompt_weight )
43: C           !region-name( prompt_weight delayed_weight ) ...
)
44: C sequential-region form:
45: C           WGTFD( prompt_weight_reg1 delayed_weight_reg1
46: C           prompt_weight_reg2 delayed_weight_reg2 ... )
47: C =====
48: C
49:      100 call FPROBE( IPP, ' (', LINE )
50:      if ( IPP.eq.0 ) then
51:          call GTLINE( IEND )
52:          if ( IEND.ne.0 ) go to 903
53:          go to 100
54:      end if
55: CM2016      if ( LINE(1:1).ne.'!' ) go to 200
56:      if ( INDEX('<+-.0123456789',LINE(1:1)).ne.0.or.LINE(1:2).eq.'R('
57:      &          ) go to 200
58: C
59:      CHAR(1:2) = '()'
60: C
61: C >>> !region-name( ... )
62: C

```

```

63:      110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
64: C
65:      if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.' ' ) go to 300      ! end
of input
66:      if ( NLEN.eq.0 ) then
67:          if ( IEND.eq.0 ) go to 110
68:          go to 905
69:      end if
70:      NR = 0
71:      IK = INDEX(LINE(:NLEN),'@')
72:      if ( IK.gt.0 ) then
73: C
74: C ..... @NAME without wildcard character .....
75:      if ( INDEX(LINE(:NLEN),'?').eq.0 .and.
76:      &          INDEX(LINE(:NLEN),'*').eq.0 ) then
77:          call CBSINC( TNAMS, NNAMES, LINE(IK:NLEN), IPOS )
78:          if ( IPOS.eq.0 ) then
79:              write(IMG,910) LINE(IK:NLEN)
80:              call CNTERR( 'FATAL' )
81:              call DMREAD( ' ', IDML, N1, IIER )
82:              if ( IIER.ne.0 ) go to 911
83:              go to 110
84:          end if
85:          do I = 1, NTREG
86:              if ( ITRNM(I).lt.0.and.IPOS.eq.abs(ITRNM(I)) ) go to 120
87:          end do
88:          go to 913
89:      120      IRD = I
90:          if ( JSKIP.ne.0 ) then
91:              call DMREAD( ' ', IDML, N1, IIER )
92:              if ( IIER.ne.0 ) go to 911
93:              go to 110
94:          end if
95:          do K = IPTRG(IRD), IPTRG(IRD+1)-1
96:              NR = NR + 1
97:              KR(NR) = LPTRG(K)
98:          end do
99: C
100: C ..... @NAME with wildcard character .....
101:      else
102:          do J = 1, NTREG
103:              if ( ITRNM(J).lt.0 ) then
104:                  if ( IMATCH(LINE(IK:NLEN),TNAMS(abs(ITRNM(J))))).ne.0 )
105:                  &                      then
106:                      do 130 I = IPTRG(J), IPTRG(J+1)-1
107:                          do K = 1, NR
108:                              if ( KR(K).eq.LPTRG(I) ) go to 130
109:                          end do
110:                          NR = NR + 1
111:                          KR(NR) = LPTRG(I)
112:                      130      continue
113:                      end if
114:                  end if
115:              end do
116:          end if
117: C
118: C ..... regular region name .....
119:      else if ( IK.eq.0 ) then
120:          IBEGIN = 0
121:          JEND = 0
122:          150      continue
123:          call RGFIND( IMG, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
124:      &                      JTLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
125:      &                      KSUZ, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
126:      &                      NNAMES )

```


src/mvp/wgtfdinp.f

```

127:         if ( IERR.ne.0 ) then
128:             write(IMG,916) LINE(:NLEN)
129:             call CNTERR( 'FATAL' )
130:             call DMREAD( ' ', IDM1, N1, IIER )
131:             if ( IIER.ne.0 ) go to 911
132:             go to 110
133:         end if
134:         if ( IREG.ne.0 ) then
135:             do K = 1, NR
136:                 if ( KR(K).eq.IREG ) go to 150
137:             end do
138:             NR = NR + 1
139:             KR(NR) = IREG
140:         end if
141:         if ( JEND.eq.0 ) go to 150
142:     end if
143: C
144: C ..... read the secondary generation weights .....
145: NB = 2
146: call R4READ( 'WGTFD', A(LTEMP), NA, NB, IERR )
147: if ( IERR.ne.0 ) go to 917
148: C
149: if ( NA.gt.0 ) then
150:     WGTFD0 = A(LTEMP)
151:     if ( WGTFD0.lt.0 ) WGTFD0 = 1.0
152:     do K = 1, NR
153:         A(LWGTF+KR(K)-1) = WGTFD0
154:     end do
155: C
156:     if ( NA.eq.2 ) then
157:         WGTFD0 = A(LTEMP+1)
158:         if ( WGTFD0.lt.0 ) WGTFD0 = 1.0
159:         do K = 1, NR
160:             A(LWGTF+NREG+KR(K)-1) = WGTFD0
161:         end do
162:     end if
163: end if
164: C
165: go to 110
166: C
167: C >>> sequential region : prompt_weight_reg1 delayed_weight_reg1...
168: C
169: 200 continue
170: NB = NN
171: call R4READ( 'WGTFD', A(LTEMP), NA, NB, IERR )
172: if ( IERR.ne.0 ) go to 919
173: C
174: do I = 1, NA
175:     if ( A(LTEMP+I-1) .lt. 0 ) A(LTEMP+I-1) = 1.0
176:     I0 = (I-1) / 2
177:     I1 = MOD( I , 2 )
178:     if ( I1.eq.1 ) then
179:         A(LWGTF+I0) = A(LTEMP+I-1)
180:     else
181:         A(LWGTF+I0+NREG) = A(LTEMP+I-1)
182:     end if
183: end do
184: C
185: 300 continue
186: call STLAST( 'WGTFD', LASTTM, LAST )
187: return
188: C
189: C ..... error process .....
190: 901 write(IMG,902)
191: call PRSTOP( 1, 'Problem in order of input data.' )

192: go to 999
193: 903 write(IMG,904)
194: call PRSTOP( 1, 'Unexpected end of input file.' )
195: go to 999
196: 905 write(IMG,906)
197: call PRSTOP( 1, 'Unexpected end of input file.' )
198: go to 999
199: 911 write(IMG,912)
200: call PRSTOP( 1, 'Error in region dependent data input.' )
201: go to 999
202: 913 write(IMG,914) (ITRNM(I),I=1,NTREG)
203: call PRSTOP( 1, 'Problem in region dependent data input.' )
204: go to 999
205: 917 write(IMG,918)
206: call PRSTOP( 1, 'Unexpected end of input file.' )
207: go to 999
208: 919 write(IMG,920)
209: call PRSTOP( 1, 'Unexpected end of input file.' )
210: 999 stop 888
211: 902 format(/' XXX(wgtfdinp) Number of "region" (NREG) is not determi',
212: & 'ned before REGION dependent data input.'
213: &/' Your input for <WGTFD> must be placed after',
214: & ' $GEOMETRY block.')
215: 904 format(/' XXX(wgtfdinp) End of file is encountered for <WGTFD>.')
216: 906 format(/' XXX(wgtfdinp) End of file is encountered during',
217: & ' "!region-name(...)" search for <WGTFD>.')
218: 910 format(/' XXX(wgtfdinp) Tally region <',a,'> does not exist.'
219: &/' Data of <WGTFD> for this region are meaningless.')
220: 912 format(/' XXX(wgtfdinp) Unexpected end of data or input error',
221: & ' during skipping unnecessary or invalid data.'
222: &/' (REGION dependent data input)')
223: 914 format(/' XXX(wgtfdinp) Fatal error for tally-region name',
224: & ' (ITRNM).')
225: &/(i5,i10))
226: 916 format(/' XXX(wgtfdinp) Region <',a,'> is not defined or invalid',
227: & ' name.'
228: &/' Data of <WGTFD> for this region are meaningless.')
229: 918 format(/' XXX(wgtfdinp) End of file is encountered during data',
230: & ' in "!region-name(weight_p,weight_d)..." for <WGTFD>.')
231: 920 format(/' XXX(wgtfdinp) End of file is encountered during',
232: & ' sequential region weights for <WGTFD>.')
233: end

```

src/mvp/wgtpninp.f

```

1:      subroutine WGTPNINP(IMSG, A, IA, LIMIT, LAST, JDEBG, NREG, NUCPN,
2:      &                    NMTPN, LWGTPN, NCIDPN, MNUCPN, LPIDPN, JTLLT,
3:      &                    KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
4:      &                    KSUZN, IRGSP, NEST, NSPACE, NSUZON, NZONE,
5:      &                    TNAMS, NNAMES, ITRNM, NTREG, IPTRG, LPTRG, KR,
6:      &                    NRRRO )
7: C=<MVP>=====
8: C  purpose:  input array data for secondary generation weight of each
9: C            region and nuclide by photonuclear reaction.
10: C  called in:  INTRO
11: C  calls:
12: C-----
13:      include '../shared/INC/_LNAM'
14: C
15:      real A(LIMIT)
16:      integer IA(LIMIT), JDEBG(*)
17: C
18: C      .... data for photonuclear ....
19:      integer MNUCPN(*), LPIDPN(*)
20:      character NCIDPN(*)*16
21: C
22: C      .... data for region name matching ....
23:      integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ),
24:      &        ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
25:      &        KSUZN(NINPZ,2), IRGSP(2,NREG)
26:      integer ITRNM(NTREG), IPTRG(NTREG+1), LPTRG(*), KR(NRRRO)
27:      character TNAMS(NNAMES)*LNAM
28: C
29: C      .... local variable ....
30:      character LINE*72, CHAR*4, NM*5
31: c##<2007/03/14:PN4:
32: C
33:      real*8 DENTAK
34:      external DENTAK
35: c##>
36: C
37: C-----
38: C
39: C      ..... check and initialize .....
40:      if ( NUCPN.le.0 ) go to 901
41:      if ( NREG .le.0 ) go to 903
42:      if ( NMTPN.le.0.or.NMTPN.gt.135 ) go to 927
43:      JSKIP = 0
44:      NN = NMTPN * NUCPN * NREG
45:      if ( LWGTPN.eq.0 )
46:      & call KEVP( A(1), 'WGTPN', LWGTPN, NN, 'R4', LAST, 0.001, JDEBG)
! default = 0.001
47: C
48: C =====
49: C < independent on reaction type >
50: C region-wise form: WGTPN( !region-name( weight( nuclide ... ) ... ) )
51: C                    WGTPN( !region-name( weight ) ... )
52: C nuclide-wise form: WGTPN( weight( nuclide ... ) ... )
53: C all-nuclide form: WGTPNR( weight ... ) in regdat.f & intro.f
54: C
55: C < dependent on reaction type >
56: C region-wise form: WGTPN( !region-name( weight( nuclide mt nuclide mt ...
) ... ) )
57: C                    WGTPN( !region-name( weight( nuclide nuclide mt
58: C                    nuclide mt nuclide nuclide
... ) ... ) )
59: C nuclide-wise form: WGTPN( weight( nuclide mt nuclide mt ... ) ... )
60: C                    WGTPN( weight( nuclide nuclide mt nuclide nuclide mt ..
) ... )
61: C      [note: If no reaction type (mt), it means all types.

62: C      mt=18 means prompt neutron of fission event.
63: C      mt=-18 or 98 means delayed neutron of fission event.]
64: C =====
65: C
66:      100 call FPROBE( IPP, '(', LINE )
67:      if ( IPP.eq.0 ) then
68:      call GTLINE( IEND )
69:      if ( IEND.ne.0 ) go to 905
70:      go to 100
71:      end if
72:      CHAR(1:2) = '()'
73:      if ( LINE(1:1).ne.'!' ) go to 200
74: C
75: C      >>> !region-name( ... )
76: C
77:      110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
78: C
79:      if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.'') go to 300      ! end
of input
80:      if ( NLEN.eq.0 ) then
81:      if ( IEND.eq.0 ) go to 110
82:      go to 907
83:      end if
84:      NR = 0
85:      IK = INDEX(LINE(:NLEN),'@')
86:      if ( IK.gt.0 ) then
87: C
88: C      ..... @NAME without wildcard character .....
89:      if ( INDEX(LINE(:NLEN),'?').eq.0 .and.
90:      & INDEX(LINE(:NLEN),'*').eq.0 ) then
91:      call CBSINC( TNAMS, NNAMES, LINE(IK:NLEN), IPOS )
92:      if ( IPOS.eq.0 ) then
93:      write(IMSG,910) LINE(IK:NLEN)
94:      call CNTERR( 'FATAL' )
95:      call DMREAD( ' ', ID1, N1, IIER )
96:      if ( IIER.ne.0 ) go to 911
97:      go to 110
98:      end if
99:      do I = 1, NTREG
100:      if ( ITRNM(I).lt.0.and.IPOS.eq.abs(ITRNM(I)) ) go to 120
101:      end do
102:      go to 913
103:      120      IRD = I
104:      if ( JSKIP.ne.0 ) then
105:      call DMREAD( ' ', ID1, N1, IIER )
106:      if ( IIER.ne.0 ) go to 911
107:      go to 110
108:      end if
109:      do K = IPTRG(IRD), IPTRG(IRD+1)-1
110:      NR = NR + 1
111:      KR(NR) = LPTRG(K)
112:      end do
113: C
114: C      ..... @NAME with wildcard character .....
115:      else
116:      do J = 1, NTREG
117:      if ( ITRNM(J).lt.0 ) then
118:      if ( IMATCH(LINE(IK:NLEN),TNAMS(abs(ITRNM(J)))) .ne.0 )
119:      & then
120:      do 130 I = IPTRG(J), IPTRG(J+1)-1
121:      do K = 1, NR
122:      if ( KR(K).eq.LPTRG(I) ) go to 130
123:      end do
124:      NR = NR + 1
125:      KR(NR) = LPTRG(I)

```

src/mvp/wgtpninp.f

```

126: 130          continue
127:          end if
128:      end if
129:  end do
130:  end if
131: C
132: C      ..... regular region name .....
133:  else if ( IK.eq.0 ) then
134:      IBEGIN = 0
135:      JEND = 0
136: 150  continue
137:      call RGFIND( MSG, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
138: &      JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPM, ISUSP,
139: &      KSUZ, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
140: &      NNAME )
141:      if ( IERR.ne.0 ) then
142:          write(IMG,916) LINE(:NLEN)
143:          call CNTERR( 'FATAL' )
144:          call DMREAD( ' ', IDMI, NI, IIER )
145:          if ( IIER.ne.0 ) go to 911
146:          go to 110
147:      end if
148:      if ( IREG.ne.0 ) then
149:          do K = 1, NR
150:              if ( KR(K).eq.IREG ) go to 150
151:          end do
152:          NR = NR + 1
153:          KR(NR) = IREG
154:      end if
155:      if ( JEND.eq.0 ) go to 150
156:  end if
157: C
158: C      ..... read the secondary generation weight .....
159: 160 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
160:  if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.' ' ) then ! end of a reg
ion
161:      if ( NLEN.gt.0 ) then ! all nuclides in a region are same we
ight.
162: c##<2007/03/14:PN4:
163: c##      read(LINE(:NLEN),*) WGTPN0
164:      IER = 0
165:      WGTPN0 = DENTAK( LINE(1:NLEN), MSG, IER )
166: c##>
167:      if ( WGTPN0.lt.0.0 ) WGTPN0 = 0
168:      do K = 1, NR
169:          I1 = NUCPN * (KR(K)-1) - 1
170:          do I = 1, NUCPN
171:              A(LWGTPN+I1+I) = WGTPN0
172:          end do
173:      end do
174:      end if
175:      go to 110
176:  end if
177:  if ( NLEN.eq.0 ) then
178:      if ( IEND.eq.0 ) go to 160
179:      go to 917
180:  end if
181: c##<2007/03/14:PN4:
182: c##      read(LINE(:NLEN),*) WGTPN0
183:      IER = 0
184:      WGTPN0 = DENTAK( LINE(1:NLEN), MSG, IER )
185: c##>
186:  if ( WGTPN0.lt.0.0 ) WGTPN0 = 0
187: C
188: C      ..... read the nuclide names .....
189:      ND = 0
190: 170 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
191:  if ( IEND.ne.0 ) go to 923
192: 175 continue
193:      ND = ND + 1
194:      NL0 = NLEN
195:      if ( NLEN.eq.5 ) then
196:          NM = LINE(:NLEN)
197:      else if ( NLEN.eq.3 ) then
198:          NM = LINE(:NLEN) // ' '
199:      else if ( NLEN.eq.2 ) then
200:          NM = LINE(:NLEN) // ' '
201:      else if ( NLEN.eq.1 ) then
202:          NM = LINE(:NLEN) // '0 '
203:      else
204:          go to 929
205:      end if
206:      IKW = 0
207:      if ( NM(3:5).eq.'000' ) then
208:          IKW = 1
209:      else if ( NM(3:3).eq.'*' ) then
210:          IKW = 1
211:      else if ( NM(2:2).eq.'*' ) then
212:          IKW = 1
213:          NM(2:2) = '0'
214:      else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
215:          IKW = 1
216:      end if
217:      if ( ITERM.gt.0 ) then
218:          MT0 = 0
219:          KNAME0 = 0
220:      else
221:          call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
222:          if ( NLEN.le.0 ) go to 929
223:          if ( IEND.ne.0 ) go to 923
224:          KNAME0 = INDEX('ABCDEFGHIJKLMNPOQRSTUVWXYZ',LINE(1:1))
225:          if ( KNAME0.eq.0 ) then
226:              read(LINE(:NLEN),*) MT0
227:              if ( MT0.eq.-18 ) MT0 = 98
228:              if ( MT0.le.0.or.MT0.gt.NMTPN ) go to 931
229:          else
230:              MT0 = 0
231:          end if
232:      end if
233:      if ( IKW.eq.0 ) then
234:          do I = 1, NUCPN
235:              if ( NM(1:NLEN).eq.NCIDPN(I)(1:NLEN) ) go to 180
236:          end do
237:          write(IMG,891) NM, (KR(K),K=1,NR)
238:          call CNTERR( 'WARNING' )
239:          ND = ND - 1
240:          go to 190
241: 180  do K = 1, NR
242:          I1 = NMTPN*NUCPN*(KR(K)-1) + NMTPN*(I-1) - 1
243:          if ( MT0.eq.0 ) then
244:              do I2 = 1, NMTPN
245:                  A(LWGTPN+I1+I2) = WGTPN0
246:              end do
247:          else
248:              A(LWGTPN+I1+MT0) = WGTPN0
249:          end if
250:      end do
251:      else if ( IKW.eq.1 ) then
252:          II = 0
253: 184  do I = II+1, NUCPN

```

src/mvp/wgtpninp.f

```

254:         if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 185
255:     end do
256:     if ( II.le.0 ) then
257:         write(IMG,891) NM(1:2), (KR(K),K=1,NR)
258:         call CNTERR( 'WARNING' )
259:         ND      = ND - 1
260:     end if
261:     go to 190
262: 185     do K = 1, NR
263:         I1      = NMTPN*NUCPN*(KR(K)-1) + NMTPN*(I-1) - 1
264:         if ( MT0.eq.0 ) then
265:             do I2 = 1, NMTPN
266:                 A(LWGTPN+I1+I2) = WGTPN0
267:             end do
268:         else
269:             A(LWGTPN+I1+MT0) = WGTPN0
270:         end if
271:     end do
272:     if ( I.lt.NUCPN ) then
273:         II      = I
274:         go to 184
275:     end if
276: end if
277: 190 continue
278: if ( KNAME0.gt.0 ) go to 175
279: if ( ITERM.eq.0 ) go to 170
280: go to 160
281: C
282: C >>> nuclide-wise for all regions : weight(...)
283: C
284: 200 continue
285: C ..... read the secondary generation weight .....
286: call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
287: if ( ITERM.ne.0 .and. CHAR(ITERM:ITERM).eq.' ' ) go to 300
288: if ( NLEN.eq.0 ) then
289:     if ( IEND.eq.0 ) go to 200
290:     go to 925
291: end if
292: c##<2007/03/14:PN4:
293: c## read(LINE(:NLEN),*) WGTPN0
294: IER = 0
295: WGTPN0 = DENTAK( LINE(1:NLEN), IMG, IER )
296: c##>
297: if ( WGTPN0.lt.0.0 ) WGTPN0 = 0
298: C
299: C ..... read the nuclide names .....
300: ND      = 0
301: 210 call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
302: if ( IEND.ne.0 ) go to 919
303: 215 continue
304: ND      = ND + 1
305: NL0     = NLEN
306: if ( NLEN.eq.5 ) then
307:     NM      = LINE(:NLEN)
308: else if ( NLEN.eq.3 ) then
309:     NM      = LINE(:NLEN) // ' '
310: else if ( NLEN.eq.2 ) then
311:     NM      = LINE(:NLEN) // ' ' '
312: else if ( NLEN.eq.1 ) then
313:     NM      = LINE(:NLEN) // '0 '
314: else
315:     go to 929
316: end if
317: IKW      = 0
318: if ( NM(3:5).eq.'000' ) then

```

```

319:         IKW = 1
320:     else if ( NM(3:3).eq.'*' ) then
321:         IKW = 1
322:     else if ( NM(2:2).eq.'*' ) then
323:         IKW = 1
324:         NM(2:2) = '0'
325:     else if ( NM(3:3).eq.'N' .or. NM(3:3).eq.' ' ) then
326:         IKW = 1
327:     end if
328: if ( ITERM.gt.0 ) then
329:     MT0      = 0
330:     KNAME0    = 0
331: else
332:     call CHREAD( ' ', LINE, ' ', NLEN, ITERM, IEND )
333:     if ( NLEN.le.0 ) go to 929
334:     if ( IEND.ne.0 ) go to 923
335:     KNAME0    = INDEX('ABCDEFGHIJKLMNQRSTUUVWXYZ',LINE(1:1))
336:     if ( KNAME0.eq.0 ) then
337:         read(LINE(:NLEN),*) MT0
338:         if ( MT0.eq.-18 ) MT0 = 98
339:         if ( MT0.le.0.or.MT0.gt.NMTPN ) go to 931
340:     else
341:         MT0 = 0
342:     end if
343: end if
344: if ( IKW.eq.0 ) then
345:     do I = 1, NUCPN
346:         if ( NM(1:NLO).eq.NCIDPN(I)(1:NLO) ) go to 220
347:     end do
348:     write(IMG,891) NM
349:     call CNTERR( 'WARNING' )
350:     ND      = ND - 1
351:     go to 250
352: 220     do K = 1, NREG
353:         I1      = NMTPN*NUCPN*(K-1) + NMTPN*(I-1) - 1
354:         if ( MT0.eq.0 ) then
355:             do I2 = 1, NMTPN
356:                 A(LWGTPN+I1+I2) = WGTPN0
357:             end do
358:         else
359:             A(LWGTPN+I1+MT0) = WGTPN0
360:         end if
361:     end do
362: else if ( IKW.eq.1 ) then
363:     II      = 0
364: 230     do I = II+1, NUCPN
365:         if ( NM(1:2).eq.NCIDPN(I)(1:2) ) go to 240
366:     end do
367:     if ( II.le.0 ) then
368:         write(IMG,891) NM(1:2)
369:         call CNTERR( 'WARNING' )
370:         ND      = ND - 1
371:     end if
372:     go to 250
373: 240     do K = 1, NREG
374:         I1      = NMTPN*NUCPN*(K-1) + NMTPN*(I-1) - 1
375:         if ( MT0.eq.0 ) then
376:             do I2 = 1, NMTPN
377:                 A(LWGTPN+I1+I2) = WGTPN0
378:             end do
379:         else
380:             A(LWGTPN+I1+MT0) = WGTPN0
381:         end if
382:     end do
383:     if ( I.lt.NUCPN ) then

```

src/mvp/wgtpninp.f

```

384:      II      = I
385:      go to 230
386:      end if
387:      end if
388: 250 continue
389:      if ( KNAME0.gt.0 ) go to 215
390:      if ( ITERM.eq.0 ) go to 210
391:      go to 200
392: C
393: C      >>> end of input
394: C
395: 300 continue
396:      return
397: C
398: C ..... warning message .....
399: 891 format(/' !!!(wgtpninp) Nuclide name <' ,a,'> is undefined as',
400:      & ' photonuclear ',
401:      & ' these region number:' ,i5i6
402:      & /(21x,i5i6))
403: C
404: C ..... error process .....
405: 901 write(IMG,902)
406:      call PRSTOP( 1, 'Problem in order of input data.' )
407:      go to 999
408: 903 write(IMG,904)
409:      call PRSTOP( 1, 'Problem in order of input data.' )
410:      go to 999
411: 905 write(IMG,906)
412:      call PRSTOP( 1, 'Unexpected end of input file.' )
413:      go to 999
414: 907 write(IMG,908)
415:      call PRSTOP( 1, 'Unexpected end of input file.' )
416:      go to 999
417: 911 write(IMG,912)
418:      call PRSTOP( 1, 'Error in region dependent data input.' )
419:      go to 999
420: 913 write(IMG,914) (ITRNM(I),I=1,NTREG)
421:      call PRSTOP( 1, 'Problem in region dependent data input.' )
422:      go to 999
423: 917 write(IMG,918)
424:      call PRSTOP( 1, 'Unexpected end of input file.' )
425:      go to 999
426: 919 write(IMG,920)
427:      call PRSTOP( 1, 'Unexpected end of input file.' )
428:      go to 999
429: 923 write(IMG,924)
430:      call PRSTOP( 1, 'Unexpected end of input file.' )
431:      go to 999
432: 925 write(IMG,926)
433:      call PRSTOP( 1, 'Unexpected end of input file.' )
434:      go to 999
435: 927 write(IMG,928)
436:      call PRSTOP( 1, 'Problem in order of input data.' )
437:      go to 999
438: 929 write(IMG,930) NLEN
439:      call PRSTOP( 1, 'Incomplete data structure of input file.' )
440:      go to 999
441: 931 write(IMG,932) MT0
442:      call PRSTOP( 1, 'Incomplete data structure of input file.' )
443: 999 stop 888
444: 902 format(/' XXX(wgtpninp) Number of "photonuclear nuclide" (NUCPN)',
445:      & ' is not determined before photonuclear dependent data input.'
446:      & '
447:      & ' $XSEC block.')
448: 904 format(/' XXX(wgtpninp) Number of "region" (NREG) is not determi',

```

```

449:      & 'ned before REGION dependent data input.'
450:      & '
451:      & ' $GEOMETRY block.')
452: 906 format(/' XXX(wgtpninp) End of file is encountered for <WGTPN>.')
453: 908 format(/' XXX(wgtpninp) End of file is encountered during',
454:      & ' " !region-name(...)" search for <WGTPN>.')
455: 910 format(/' XXX(wgtpninp) Tally region <' ,a,'> does not exist.'
456:      & '
457:      & ' Data of <WGTPN> for this region are meaningless.')
458: 912 format(/' XXX(wgtpninp) Unexpected end of data or input error',
459:      & ' during skipping unnecessary or invalid data.'
460:      & '
461:      & ' (REGION dependent data input)')
462: 914 format(/' XXX(wgtpninp) Fatal error for tally-region name',
463:      & ' (ITRNM).',
464:      & /(i5,i10))
465: 916 format(/' XXX(wgtpninp) Region <' ,a,'> is not defined or invalid',
466:      & ' name.'
467:      & '
468:      & ' Data of <WGTPN> for this region are meaningless.')
469: 918 format(/' XXX(wgtpninp) End of file is encountered during data',
470:      & ' in " !region-name(weight(...))" for <WGTPN>.')
471: 920 format(/' XXX(wgtpninp) End of file is encountered during data',
472:      & ' in "weight(nuclide ...)" of nuclide-wise for <WGTPN>.')
473: 924 format(/' XXX(wgtpninp) End of file is encountered during data',
474:      & ' in " !region-name(weight(nuclide ...))" for <WGTPN>.')
475: 926 format(/' XXX(wgtpninp) End of file is encountered during',
476:      & ' "weight(...)" search for <WGTPN>.')
477: 928 format(/' XXX(wgtpninp) Number of reactions on "photonuclear',
478:      & ' nuclide" (NMTPN) is not determined before photonuclear',
479:      & ' dependent data input.'
480:      & '
481:      & ' Your input for <WGTPN> must be placed after',
482:      & ' $XSEC block.')
483: 930 format(/' XXX(wgtpninp) Length of data in "(nuclide,reaction)",
484:      & ' is illegal for <WGTPN>.'
485:      & '
486:      & ' NLEN=',i5)
487: 932 format(/' XXX(wgtpninp) Reaction type for photonuclear is out',
488:      & ' of range (1 to 135) for <WGTPN>.'
489:      & '
490:      & ' MT0=',i5)
491:      end
492: C##
493: C      subroutine WGTPNWRT( WGTPN, NMTPN, NUCPN, NREG, IPR )
494: C=====
495: C<MVP>=====
496: C purpose: print array data for secondary generation weight of each
497: C region, nuclide and reaction type in photonuclear reaction.
498: C called in: INTRO2
499: C calls:
500: C=====
501: C
502: C ..... data for photonuclear .....
503: C      real WGTPN(NMTPN,NUCPN,NREG)
504: C
505: C ..... local variable .....
506: C      character HL1*120,HL2*120,HL0(10)*12
507: C      equivalence (HL0,HL2)
508: C-----
509: C
510: C ..... initialize .....
511: C      INEG      = 0
512: C      NR        = 1
513: C
514: C ..... print the reading data and check .....
515: C      write(IPR,600) NMTPN,NUCPN,NREG
516: C      IS        = 10
517: C 110 continue
518: C      write(IPR,605) NR
519: C      do I = 1, NUCPN, IS

```

src/mvp/wgtpninp.f

```

514:      I1      = min(I+IS-1, NUCPN)
515:      write(IPR,601) ('-----',I2=I,I1)
516:      write(IPR,602) (I2,I2=I,I1)
517:      write(IPR,601) ('-----',I2=I,I1)
518:      HL1      = ' '
519:      ISAME1    = 0
520:      do J = 1, NMTPN
521:         HL2    = ' '
522:         I3     = 0
523:         do I2 = I, I1
524:            I3   = I3 + 1
525:            write(HL0(I3),'(1p,e12.4)') WGTPN(J,I2,NR)
526:            if ( WGTPN(J,I2,NR).lt.0.0 ) INEG = INEG + 1
527:         end do
528:         I4     = I3 * 12
529:         if ( HL1(1:I4).eq.HL2(1:I4) ) then
530:            if ( ISAME1.eq.0 ) ISAME1 = J
531:            ISAME2 = J
532:         else
533:            if ( ISAME1.gt.0 ) then
534:               write(IPR,604) ISAME1,ISAME2
535:               ISAME1 = 0
536:            end if
537:            write(IPR,603) J,HL2(1:I4)
538:            HL1      = HL2
539:         end if
540:         end do
541:         if ( ISAME1.gt.0 ) write(IPR,604) ISAME1,ISAME2
542:         write(IPR,601) ('-----',I2=I,I1)
543:      end do
544: C
545: C ..... check the same data .....
546: if ( NR.ge.NREG ) go to 120
547: do K = NR+1, NREG
548:   do J = 1, NMTPN
549:     do I = 1, NUCPN
550:       if (WGTPN(J,I,NR).ne.WGTPN(J,I,K) ) then
551:         if ( K.gt.NR+1 ) write(IPR,606) NR+1,K-1
552:         NR      = K
553:         go to 110
554:       end if
555:     end do
556:   end do
557: end do
558: write(IPR,606) NR+1,NREG
559: C
560: 120 continue
561: if ( INEG.gt.0 ) then
562:   write(IPR,901) INEG
563:   call CNTERR( 'FATAL' )
564: end if
565: return
566: C
567: 600 format('// <<< ARRAY WGTPN(I,J,K) (I(mt)=1,',i4,', J(nuclide)=1,',
568:   & i4,', K(region)=1,',i4,', >>> ..... generation weight')
569: 601 format(' -----',10a12)
570: 602 format('          | NUCLIDE'
571:   &          '// MT      |',10(i9:3x))
572: 603 format(i8,' |',a)
573: 604 format(5x,'... | ( reaction (mt) ',i3,' to ',i3,
574:   & ' are same as above reaction )')
575: 605 format(' < REGION ',i4,' >')
576: 606 format(' < REGION ',i4,' to ',i4,' are same as above region >'
577:   &          '// < ----- >')
578: 901 format('// XXX(wgtpnwrt) Secondary generation weights of photo-',

```

```

579:      & 'nuclear reaction includes ',i6,' negative weights.')
580:      end

```

src/mvp/wrkary.f

```

1:      subroutine WRKARY( A,      H,      LIMIT, LIMITL,MAXSF, MWVEC,
2:      &                  MVSTK, LLXYZ, LLPWRK,LLVSTK,LXYZ,  LPWRK,
3:      &                  LVSTK, MAXWK )
4:      C
5:      C=<MVP>=====
6:      C PURPOSE : CALCULATE WORKIG-ARRAY-POINTERS AND SIZE OF MEMORY
7:      C           FOR EACH EVENT-PROCESSING ROUTINE.
8:      C CALLED IN : INTRO2
9:      C CALLS : KEEP
10:     C=====
11:     real A(*)
12:     real H(*)
13:     C
14:     integer LXZY(10), LPWRK(MWVEC), MVSTK(MVSTK)
15:     C
16:     include 'INC/_KPIDS'
17:     include 'INC/_NGPS'
18:     C
19:     include 'INC/_FLAGS'
20:     include 'INC/_shared/INC/_SIZES'
21:     include 'INC/_SIZES2'
22:     include 'INC/_shared/INC/_PTALY0'
23:     include 'INC/_PTALY1'
24:     include 'INC/_PTALY2'
25:     include 'INC/_shared/INC/_STALY'
26:     include 'INC/_shared/INC/_PTLSP'
27:     include 'INC/_CXSEC'
28:     include 'INC/_SBANK'
29:     C
30:     include 'INC/_WKSOU'
31:     include 'INC/_WKFSS'
32:     include 'INC/_WKFL1'
33:     include 'INC/_WKSEL1'
34:     include 'INC/_WKCOL'
35:     include 'INC/_WKMIR'
36:     include 'INC/_WKLAT'
37:     include 'INC/_WKFLA'
38:     include 'INC/_WKSEA'
39:     include 'INC/_WKTLS'
40:     include 'INC/_WKTLO'
41:     include 'INC/_WKPHT'
42:     C
43:     include 'INC/_WKC2A'
44:     C
45:     include 'INC/_WKNXT'
46:     C
47:     include 'INC/_shared/INC/_IOUNIT'
48:
49:     include 'INC/_PERT'
50:
51:     c##<2007/03/14:PN4:
52:     parameter ( MAXXX = 10 )
53:     c##>
54:     C
55:     C ... keep a dummy data for memory violation check if necessary,
56:     C and for memory boundary adjustment.
57:     C
58:     call LKEEPV(H(1), 'WDUMY', LWK, 1, 'R8', LAST, 1.234D-12, JDEBG )
59:     call LGTLST( LWORK )
60:     C
61:     write(IPR,7000) LWORK
62:     7000 format(/1X,' == Working area starts at ',I12,'th element',
63:     &          ' of task local data array. =='/)
64:     C
65:     C .... WORKING ARRAY OVERFLOW COUNTER ....

```

```

66:     C
67:     IOVFL = 0
68:     MAXWK = 0
69:     C
70:     C
71:     C ===== FOR SOURCE =====
72:     C
73:     C for 'SOURCE' & 'NUCFIS' routine
74:     C
75:     C
76:     call LSTLST( 'SOURCE', LWORK, LASTP )
77:     call LKEEP( 'R', P1(1), 7*NBANK, 'R4D', LASTP, JDEBG )
78:     call LKEEP( 'X', P1(2), NBANK, 'R8', LASTP, JDEBG )
79:     call LKEEP( 'Y', P1(3), NBANK, 'R8', LASTP, JDEBG )
80:     call LKEEP( 'Z', P1(4), NBANK, 'R8', LASTP, JDEBG )
81:     call LKEEP( 'IWK1', P1(5), NBANK, 'I4', LASTP, JDEBG )
82:     call LKEEP( 'IWK2', P1(6), NBANK, 'I4', LASTP, JDEBG )
83:     call LKEEP( 'IWK3', P1(7), NBANK, 'I4', LASTP, JDEBG )
84:     call LKEEP( 'EW1', P1(8), NBANK, 'R4D', LASTP, JDEBG )
85:     call LKEEP( 'RWK1', P1(9), NBANK, 'R4', LASTP, JDEBG )
86:     call LKEEP( 'IFL', P1(10), NBANK, 'I4D', LASTP, JDEBG )
87:     call LKEEP( 'IFI', P1(11), NBANK, 'I4', LASTP, JDEBG )
88:     call LKEEP( 'NNSRC', P1(12), NSOUR, 'I4', LASTP, JDEBG )
89:     call LKEEP( 'XSOUR', P1(13), NSOUR, 'R8', LASTP, JDEBG )
90:     C
91:     call LKEEP( 'SMCW', P1(14), NBANK*NSMIC, 'R4', LASTP, JDEBG )
92:     call LKEEP( 'RWF', P1(15), NHIST, 'R4', LASTP, JDEBG )
93:
94:     if ( JPRT.ne.0 ) then
95:         call LKEEP( 'SMCWP', P1(16), NBANK*NSMIC, 'R4', LASTP, JDEBG )
96:         call LKEEP( 'RP', P1(17), 7*NBANK, 'R4', LASTP, JDEBG )
97:         call LKEEP( 'E2P', P1(18), NBANK, 'R4', LASTP, JDEBG )
98:         call LKEEP( 'DEP', P1(19), NBANK, 'I4', LASTP, JDEBG )
99:         if ( NPTBE.gt.0 ) then
100:             call LKEEP( 'SWLD0', P1(20), NGSP, 'R8', LASTP, JDEBG )
101:             call LKEEP( 'SWBD0', P1(21), NGSP, 'R8', LASTP, JDEBG )
102:             call LKEEP( 'SWB0', P1(22), NGSP+1, 'R8', LASTP, JDEBG )
103:         end if
104:         call LKEEP( 'SWD0', P1(23), (NGSP+1)*NPTDS*NOPSIDS, 'R8',
105:         & LASTP,
106:         & JDEBG )
107:         call LKEEP( 'SWR0', P1(24), (NGSP+1)*NPTCS, 'R8', LASTP,
108:         & JDEBG )
109:     end if
110:
111:     call LKEEP( 'RWK2', P1(25), NBANK, 'R4', LASTP, JDEBG )
112:     call LKEEP( 'IWK4', P1(26), NBANK, 'I4', LASTP, JDEBG )
113:     C
114:     C ... allocation of pointer array itself might have failed ...
115:     C
116:     NXYZ = 10
117:     if ( LLXYZ.ne.0 ) then
118:         do 100 I = 1, NXYZ
119:             call LKEEP( 'XYZ', LXZY(I), NBANK, 'R8', LASTP, JDEBG )
120:         continue
121:     else
122:         do 110 I = 1, NXYZ
123:             call LKEEP( 'XYZ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
124:         continue
125:     end if
126:
127:     if ( LLPWRK.ne.0 ) then
128:         do 120 I = 1, MWVEC
129:             call LKEEP( 'PWRK', LPWRK(I), NBANK, 'R8', LASTP, JDEBG )
130:         continue
120

```

src/mvp/wrkary.f

```

131:      else
132:        do 130 I = 1, MWVEC
133:          call LKEEP( 'PWRK ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
134: 130      continue
135:        end if
136:
137:        if ( LLVSTK.ne.0 ) then
138:          do 140 I = 1, MVSTK
139:            call LKEEP( 'VSTK ', LVSTK(I), NBANK, 'R8', LASTP, JDEBG )
140: 140      continue
141:        else
142:          do 150 I = 1, MVSTK
143:            call LKEEP( 'VSTK ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
144: 150      continue
145:          end if
146:
147:          call LSTLST( 'SOURCE', LASTP, LASTPS )
148: C
149:          if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
150:          MAXWK = MAX(MAXWK,LASTP)
151:          write(IPR,7020) 'SOURCE', LASTP - LWORK, LASTP
152: C
153: C ===== For FISSET =====
154: C
155:          if ( JEIGN.ne.0 ) then
156:            call LSTLST( 'FISSET', LWORK, LASTP )
157:
158:            call LKEEP( 'RWF', P10(1), NHIST, 'R4', LASTP, JDEBG )
159:
160:            if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
161:            MAXWK = MAX(MAXWK,LASTP)
162:            write(IPR,7020) 'FISSET', LASTP - LWORK, LASTP
163:          end if
164: C
165: C ===== FOR FLIONE =====
166: C
167:          if ( JONEZ.eq.1 ) then
168: C
169:            call LSTLST( 'FLIONE', LWORK, LASTP )
170:            call LKEEP( 'X ', P0(1), NBANK, 'R8', LASTP, JDEBG )
171:            call LKEEP( 'Y ', P0(2), NBANK, 'R8', LASTP, JDEBG )
172:            call LKEEP( 'Z ', P0(3), NBANK, 'R8', LASTP, JDEBG )
173:            call LKEEP( 'AI ', P0(4), NBANK, 'R8', LASTP, JDEBG )
174:            call LKEEP( 'BI ', P0(5), NBANK, 'R8', LASTP, JDEBG )
175:            call LKEEP( 'CI ', P0(6), NBANK, 'R8', LASTP, JDEBG )
176:            call LKEEP( 'W ', P0(7), NBANK, 'R8', LASTP, JDEBG )
177:            call LKEEP( 'DI ', P0(8), NBANK, 'R8', LASTP, JDEBG )
178:            call LKEEP( 'IGI ', P0(9), NBANK, 'I4', LASTP, JDEBG )
179:            call LKEEP( 'IBP ', P0(10), NBANK, 'I4', LASTP, JDEBG )
180:            call LKEEP( 'IFC ', P0(11), NBANK, 'I4', LASTP, JDEBG )
181:            call LKEEP( 'R ', P0(12), 3*NBANK, 'R4D', LASTP, JDEBG )
182:            call LKEEP( 'DLOC1 ', P0(13), NBANK, 'R8', LASTP, JDEBG )
183:            call LKEEP( 'DLOC2 ', P0(14), NBANK, 'R8', LASTP, JDEBG )
184:            call LKEEP( 'IKL ', P0(15), NBANK, 'I4', LASTP, JDEBG )
185:            call LKEEP( 'IKL2 ', P0(16), NBANK, 'I4', LASTP, JDEBG )
186:            if ( JFISX.ne.0 .or. JHLAT.ne.0 .or. JTSRF.ne.0 ) then
187:              call LKEEP( 'XUP ', P0(17), NBANK, 'R8', LASTP, JDEBG )
188:              call LKEEP( 'YUP ', P0(18), NBANK, 'R8', LASTP, JDEBG )
189:              call LKEEP( 'ZUP ', P0(19), NBANK, 'R8', LASTP, JDEBG )
190:              call LKEEP( 'AUP ', P0(20), NBANK, 'R8', LASTP, JDEBG )
191:              call LKEEP( 'BUP ', P0(21), NBANK, 'R8', LASTP, JDEBG )
192:              call LKEEP( 'CUP ', P0(22), NBANK, 'R8', LASTP, JDEBG )
193:            end if
194:            call LKEEP( 'IRG', P0(23), NBANK, 'I4', LASTP, JDEBG )
195:            call LKEEP( 'TI', P0(24), NBANK, 'R8', LASTP, JDEBG )

```

```

196:          if ( JTIME.ne.0 ) then
197:            call LKEEP( 'VI ', P0(25), NBANK, 'R8', LASTP, JDEBG )
198:          end if
199:          call LKEEP( 'ISTG ', P0(26), NBANK, 'I4', LASTP, JDEBG )
200:          call LKEEP( 'ILUP ', P0(27), NBANK, 'I4', LASTP, JDEBG )
201:          call LKEEP( 'ILST ', P0(28), NBANK, 'I4', LASTP, JDEBG )
202:
203:          if ( JFISX.ne.0 .or. JTSRF.ne.0 ) then
204:            call LKEEP( 'XYZ1 ', P0(29), 3*NBANK, 'R8', LASTP, JDEBG )
205:            call LKEEP( 'ABC1 ', P0(30), 3*NBANK, 'R8', LASTP, JDEBG )
206:            call LKEEP( 'XYZ2 ', P0(31), 3*NBANK, 'R8', LASTP, JDEBG )
207:            call LKEEP( 'ABC2 ', P0(32), 3*NBANK, 'R8', LASTP, JDEBG )
208:            call LKEEP( 'DDI ', P0(33), NBANK, 'R8', LASTP, JDEBG )
209:            call LKEEP( 'IBP0 ', P0(34), NBANK, 'I4', LASTP, JDEBG )
210:            call LKEEP( 'IBP1 ', P0(35), NBANK, 'I4', LASTP, JDEBG )
211:            call LKEEP( 'IBP2 ', P0(36), NBANK, 'I4', LASTP, JDEBG )
212:            call LKEEP( 'KPLT ', P0(37), NLBZ+1, 'I4', LASTP, JDEBG )
213:            call LKEEP( 'JKSF ', P0(38), NLBZ, 'I4', LASTP, JDEBG )
214:          end if
215:
216: c##<2007/03/14:PN3:
217: c##          if ( JAMXC.ne.0 ) then
218:            if ( JAMXC.ne.0 .or. NTEVE.gt.0 ) then
219: c##>
220:              call LKEEP( 'SMCW ', P0(39), NBANK*NSMIC, 'R4', LASTP, JDEBG )
221:            end if
222:
223:            if ( JPERT.ne.0 ) then
224:              call LKEPV(H(1), 'STOTP ', P0(40), NBANK, 'R8', LASTP,0D0,
225:                & JDEBG )
226:              call LKEPV(H(1), 'SNUFP ', P0(41), NBANK, 'R8', LASTP,0D0,
227:                & JDEBG )
228:              call LKEPV(H(1), 'SLOSP ', P0(42), NBANK, 'R8', LASTP,0D0,
229:                & JDEBG )
230:            end if
231:
232:            if ( JSCTM.ne.0 ) then
233:              call LKEEP( 'IWKf1 ', P0(43), NBANK, 'I4', LASTP, JDEBG )
234:              if ( JSCTM.gt.1 ) then
235:                call LKEEP( 'WKf1 ', P0(44), NBANK, 'R4', LASTP, JDEBG )
236:              end if
237:            end if
238:            if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
239:            MAXWK = MAX(MAXWK,LASTP)
240:            write(IPR,7020) 'FLIONE', LASTP - LWORK, LASTP
241:          end if
242: C
243: C ===== FOR SEAONE =====
244: C
245:          if ( JONEZ.eq.1 ) then
246:            call LSTLST( 'SEAONE', LWORK, LASTP )
247:            call LKEEP( 'X ', P2(1), NBANK, 'R8', LASTP, JDEBG )
248:            call LKEEP( 'Y ', P2(2), NBANK, 'R8', LASTP, JDEBG )
249:            call LKEEP( 'Z ', P2(3), NBANK, 'R8', LASTP, JDEBG )
250:            call LKEEP( 'W ', P2(4), NBANK, 'R8', LASTP, JDEBG )
251:            call LKEEP( 'IBP ', P2(5), NBANK, 'I4', LASTP, JDEBG )
252:            call LKEEP( 'IFI ', P2(6), NBANK, 'I4', LASTP, JDEBG )
253:            call LKEEP( 'IZI ', P2(7), NBANK, 'I4', LASTP, JDEBG )
254:            call LKEEP( 'IWK ', P2(8), NBANK, 'I4', LASTP, JDEBG )
255:            call LKEEP( 'IFL ', P2(9), NBANK, 'I4', LASTP, JDEBG )
256:            call LKEEP( 'R ', P2(10), NBANK, 'R4', LASTP, JDEBG )
257:            call LKEEP( 'ISRF ', P2(11), NBANK, 'I4', LASTP, JDEBG )
258:            call LKEEP( 'FXYZ ', P2(12), NBANK, 'R4', LASTP, JDEBG )
259:            call LKEEP( 'DWK1 ', P2(13), NBANK, 'R8', LASTP, JDEBG )
260:            if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1

```


src/mvp/wrkary.f

```

261:      MAXWK = MAX(MAXWK, LASTP)
262:      write(IPR,7020) 'SEAONE', LASTP - LWORK, LASTP
263:      end if
264: C
265: C ===== FOR NEUTR =====
266: C
267:      if ( JNEUT.ne.0 ) then
268: C
269:          call LSTLST( 'NEUTR', LWORK, LASTP )
270:          call LKEEP( 'IWK1 ', P3(1), NBANK, 'I4', LASTP, JDEBG )
271:          call LKEEP( 'IWK2 ', P3(2), NBANK, 'I4', LASTP, JDEBG )
272:          call LKEEP( 'IWK3 ', P3(3), NBANK, 'I4', LASTP, JDEBG )
273:          call LKEEP( 'IWK4 ', P3(4), NBANK, 'I4', LASTP, JDEBG )
274:          call LKEEP( 'IWK5 ', P3(5), NBANK, 'I4', LASTP, JDEBG )
275:          call LKEEP( 'IWK6 ', P3(6), NBANK, 'I4', LASTP, JDEBG )
276:          call LKEEP( 'IWK7 ', P3(7), NBANK, 'I4', LASTP, JDEBG )
277:          call LKEEP( 'IWK8 ', P3(8), NBANK, 'I4', LASTP, JDEBG )
278:          call LKEEP( 'IWK9 ', P3(9), NBANK, 'I4', LASTP, JDEBG )
279:          call LKEEP( 'IWK10 ', P3(10), NBANK, 'I4', LASTP, JDEBG )
280:          call LKEEP( 'IWK11 ', P3(11), NBANK, 'I4', LASTP, JDEBG )
281:          call LKEEP( 'IWK12 ', P3(12), NBANK, 'I4', LASTP, JDEBG )
282:          call LKEEP( 'IWK13 ', P3(13), NBANK, 'I4', LASTP, JDEBG )
283:          call LKEEP( 'IWK14 ', P3(14), NBANK, 'I4', LASTP, JDEBG )
284:          call LKEEP( 'IWK15 ', P3(15), NBANK, 'I4', LASTP, JDEBG )
285:          call LKEEP( 'IWK16 ', P3(16), NBANK, 'I4', LASTP, JDEBG )
286:          call LKEEP( 'IWK17 ', P3(17), NBANK, 'I4', LASTP, JDEBG )
287:          call LKEEP( 'IWK18 ', P3(18), NBANK, 'I4', LASTP, JDEBG )
288:          call LKEEP( 'IWK19 ', P3(19), NBANK, 'I4', LASTP, JDEBG )
289:          call LKEEP( 'DWK1 ', P3(20), NBANK, 'R8', LASTP, JDEBG )
290:          call LKEEP( 'DWK2 ', P3(21), NBANK, 'R8', LASTP, JDEBG )
291:          call LKEEP( 'WK3 ', P3(22), NBANK, 'R4', LASTP, JDEBG )
292:          call LKEEP( 'WK4 ', P3(23), NBANK, 'R4', LASTP, JDEBG )
293:          call LKEEP( 'WK5 ', P3(24), NBANK, 'R4', LASTP, JDEBG )
294: CM2014 call LKEEP( 'R ', P3(25), 8*NBANK, 'R4', LASTP, JDEBG )
295:          call LKEEP( 'R ', P3(25), 9*NBANK, 'R4', LASTP, JDEBG )
296:          call LKEEP( 'SMCW ', P3(26), NBANK*NSMIC, 'R4', LASTP, JDEBG )
297:
298:          if ( JPERT.ne.0 ) then
299:              call LKEEP( 'SMCWP ', P3(27), NBANK*NSMIC, 'R4', LASTP,
300:                  & JDEBG )
301:              call LKEEP( 'E2P ', P3(28), NBANK, 'R8', LASTP, JDEBG )
302:              call LKEEP( 'DEP ', P3(29), NBANK, 'I4', LASTP, JDEBG )
303:              call LKEEP( 'RP ', P3(30), 8*NBANK, 'R4', LASTP, JDEBG )
304:              call LKEEP( 'WKPT ', P3(31), NBANK*NPTDS, 'R8', LASTP,
305:                  & JDEBG )
306:              call LKEEP( 'WKPD ', P3(32), NBANK*NPTDS, 'R8', LASTP,
307:                  & JDEBG )
308:              call LKEEP( 'WKPN ', P3(33), NBANK*NPTDS*NUCPT, 'R8', LASTP,
309:                  & JDEBG )
310:          end if
311:
312: C##<2007/03/14:PN3:
313:          call LKEEP( 'DWK3 ', P3(34), NBANK, 'R8', LASTP, JDEBG )
314: C##>
315:
316:          if ( JBEFF.ne.0 ) then
317:              call LKEEP( 'WK6 ', P3(35), NBANK, 'R4', LASTP, JDEBG )
318:              call LKEEP( 'WK7 ', P3(36), NBANK, 'R4', LASTP, JDEBG )
319:              call LKEEP( 'WK8 ', P3(37), NBANK, 'R4', LASTP, JDEBG )
320:          end if
321: C
322:          if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
323:          MAXWK = MAX(MAXWK, LASTP)
324:          write(IPR,7020) 'NEUTR', LASTP - LWORK, LASTP
325:      end if

```

```

326: C
327: C ===== FOR PHOTR =====
328: C
329:          call LSTLST( 'PHOTR', LWORK, LASTP )
330:          call LKEEP( 'R ', PP(1), 4*NBANK, 'R4', LASTP, JDEBG )
331:          call LKEEP( 'DWK1 ', PP(2), NBANK, 'R8', LASTP, JDEBG )
332:          call LKEEP( 'DWK2 ', PP(3), NBANK, 'R8', LASTP, JDEBG )
333:          call LKEEP( 'WK3 ', PP(4), 2*NBANK, 'R4', LASTP, JDEBG )
334:          call LKEEP( 'WK4 ', PP(5), 2*NBANK, 'R4', LASTP, JDEBG )
335:          call LKEEP( 'ICLA ', PP(6), NBANK, 'I4', LASTP, JDEBG )
336:          call LKEEP( 'EIN ', PP(7), NBANK, 'R4', LASTP, JDEBG )
337:          call LKEEP( 'UOUT ', PP(8), NBANK, 'R4', LASTP, JDEBG )
338:          call LKEEP( 'IP2 ', PP(9), NBANK, 'I4', LASTP, JDEBG )
339:          call LKEEP( 'IP3 ', PP(10), NBANK, 'I4', LASTP, JDEBG )
340:          call LKEEP( 'IP4 ', PP(11), NBANK, 'I4', LASTP, JDEBG )
341:          call LKEEP( 'IP5 ', PP(12), NBANK, 'I4', LASTP, JDEBG )
342:          call LKEEP( 'IWK1 ', PP(13), NBANK, 'I4', LASTP, JDEBG )
343:          call LKEEP( 'IWK2 ', PP(14), NBANK, 'I4', LASTP, JDEBG )
344:          call LKEEP( 'IWK3 ', PP(15), NBANK, 'I4', LASTP, JDEBG )
345:          call LKEEP( 'IWK4 ', PP(16), NBANK, 'I4', LASTP, JDEBG )
346:          call LKEEP( 'IWK5 ', PP(17), NBANK, 'I4', LASTP, JDEBG )
347:          call LKEEP( 'IWK6 ', PP(18), 2*NBANK, 'R4', LASTP, JDEBG )
348:          call LKEEP( 'IWK7 ', PP(19), 2*NBANK, 'R4', LASTP, JDEBG )
349:          call LKEEP( 'IWK10 ', PP(20), 2*NBANK, 'R4', LASTP, JDEBG )
350:          call LKEEP( 'IWK11 ', PP(21), NBANK, 'R4', LASTP, JDEBG )
351:          if ( JBREM.ne.0 ) then
352:              call LKEEP( 'IWK20 ', PP(22), NBANK, 'I4', LASTP, JDEBG )
353:              call LKEEP( 'WK20 ', PP(23), NBANK, 'R4', LASTP, JDEBG )
354:              call LKEEP( 'WK21 ', PP(24), NBANK, 'R4', LASTP, JDEBG )
355:              call LKEEP( 'WK22 ', PP(25), NBANK, 'R4', LASTP, JDEBG )
356:              call LKEEP( 'WK23 ', PP(26), NBANK, 'R4', LASTP, JDEBG )
357:              call LKEEP( 'WK41 ', PP(27), 2*NBANK, 'R8', LASTP, JDEBG )
358:              call LKEEP( 'WK42 ', PP(28), 2*NBANK, 'R8', LASTP, JDEBG )
359:              call LKEEP( 'WK43 ', PP(29), 2*NBANK, 'R8', LASTP, JDEBG )
360:          end if
361: C##<2007/03/14:PN3:
362:          call LKEEP( 'WK5 ', PP(30), NBANK, 'R4', LASTP, JDEBG )
363:          call LKEEP( 'SMCW ', PP(31), NBANK*NSMIC, 'R4', LASTP, JDEBG )
364: C##>
365: C##<2007/03/14:PN4:
366:          if ( JPHNU.ne.0.and.JPTDT.ne.0 ) then
367:              call LKEEP( 'DWK3 ', PP(32), NBANK, 'R8', LASTP, JDEBG )
368:              call LKEEP( 'DWK4 ', PP(33), NBANK, 'R8', LASTP, JDEBG )
369:              call LKEEP( 'DWK5 ', PP(34), NBANK, 'R8', LASTP, JDEBG )
370:              call LKEEP( 'DWK6 ', PP(35), NBANK, 'R8', LASTP, JDEBG )
371:              call LKEEP( 'DWK7 ', PP(36), NBANK, 'R8', LASTP, JDEBG )
372:              call LKEEP( 'DWK8 ', PP(37), NBANK, 'R8', LASTP, JDEBG )
373:              call LKEEP( 'DWK9 ', PP(38), NBANK, 'R8', LASTP, JDEBG )
374:              call LKEEP( 'DWK10 ', PP(39), NBANK, 'R8', LASTP, JDEBG )
375:              call LKEEP( 'DWK11 ', PP(40), NBANK, 'R8', LASTP, JDEBG )
376:              call LKEEP( 'DWK12 ', PP(41), NBANK, 'R8', LASTP, JDEBG )
377:              call LKEEP( 'DWK13 ', PP(42), NBANK, 'R8', LASTP, JDEBG )
378:              call LKEEP( 'DWK14 ', PP(43), NBANK, 'R8', LASTP, JDEBG )
379:              call LKEEP( 'DWK15 ', PP(44), NBANK, 'R8', LASTP, JDEBG )
380:              call LKEEP( 'DWK16 ', PP(45), NBANK, 'R8', LASTP, JDEBG )
381:          end if
382: C##>
383: C##<2007/03/14:PN3:
384:          call LKEEP( 'DWK17 ', PP(46), NBANK, 'R8', LASTP, JDEBG )
385: C##>
386: C
387:          if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
388:          MAXWK = MAX(MAXWK, LASTP)
389:          write(IPR,7020) 'PHOTR', LASTP - LWORK, LASTP
390: C

```

src/mvp/wrkary.f

```

391: C ===== FOR MIRROR =====
392: C
393:   if ( JREFL.ne.0 ) then
394:     call LSTLST( 'MIRROR', LWORK, LASTP )
395:     call LKEEP( 'KKSF1', P4(1), NREFS+1, 'I4', LASTP, JDEBG )
396:     call LKEEP( 'KKSF2', P4(2), NREFS+1, 'I4', LASTP, JDEBG )
397:     call LKEEP( 'JKSF', P4(3), NREFS+1, 'I4', LASTP, JDEBG )
398:     call LKEEP( 'IPSURF', P4(4), NREFS+1, 'I4', LASTP, JDEBG )
399:     call LKEEP( 'X', P4(5), NBANK, 'R8', LASTP, JDEBG )
400:     call LKEEP( 'Y', P4(6), NBANK, 'R8', LASTP, JDEBG )
401:     call LKEEP( 'Z', P4(7), NBANK, 'R8', LASTP, JDEBG )
402:     call LKEEP( 'AI', P4(8), NBANK, 'R8', LASTP, JDEBG )
403:     call LKEEP( 'BI', P4(9), NBANK, 'R8', LASTP, JDEBG )
404:     call LKEEP( 'CI', P4(10), NBANK, 'R8', LASTP, JDEBG )
405:     call LKEEP( 'R', P4(11), 2*NBANK, 'R4', LASTP, JDEBG )
406:     P4(12) = P4(11)
407:     call LKEEP( 'IBP', P4(13), NBANK, 'I4', LASTP, JDEBG )
408:     call LKEEP( 'IZT', P4(14), NBANK, 'I4', LASTP, JDEBG )
409:     if ( MAXSF.ge.1 )
410:       & call LKEEP( 'DSDA0', P4(15), NBANK, 'R8', LASTP, JDEBG )
411:     if ( MAXSF.ge.2 )
412:       & call LKEEP( 'DSDA1', P4(16), NBANK, 'R8', LASTP, JDEBG )
413:     if ( MAXSF.ge.3 )
414:       & call LKEEP( 'DSDA2', P4(17), NBANK, 'R8', LASTP, JDEBG )
415:     if ( MAXSF.ge.4 )
416:       & call LKEEP( 'DSDA3', P4(18), NBANK, 'R8', LASTP, JDEBG )
417:     if ( MAXSF.ge.5 )
418:       & call LKEEP( 'DSDA4', P4(19), NBANK, 'R8', LASTP, JDEBG )
419:     if ( MAXSF.ge.6 )
420:       & call LKEEP( 'DSDA5', P4(20), NBANK, 'R8', LASTP, JDEBG )
421:     if ( MAXSF.ge.7 )
422:       & call LKEEP( 'DSDA6', P4(21), NBANK, 'R8', LASTP, JDEBG )
423:     if ( MAXSF.ge.8 )
424:       & call LKEEP( 'DSDA7', P4(22), NBANK, 'R8', LASTP, JDEBG )
425:     if ( MAXSF.ge.9 )
426:       & call LKEEP( 'DSDA8', P4(23), NBANK, 'R8', LASTP, JDEBG )
427:     if ( LSIZE(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
428:     MAXWK = MAX(MAXWK,LASTP)
429:     write(IPR,7020) 'MIRROR', LASTP - LWORK, LASTP
430:   end if
431: C
432: C ===== FOR LATTICE =====
433: C
434:   if ( JLATT.ne.0 ) then
435:     call LSTLST( 'LATTICE', LWORK, LASTP )
436:     call LKEEP( 'X', P5(1), NBANK, 'R8', LASTP, JDEBG )
437:     call LKEEP( 'Y', P5(2), NBANK, 'R8', LASTP, JDEBG )
438:     call LKEEP( 'Z', P5(3), NBANK, 'R8', LASTP, JDEBG )
439:     call LKEEP( 'AI', P5(4), NBANK, 'R8', LASTP, JDEBG )
440:     call LKEEP( 'BI', P5(5), NBANK, 'R8', LASTP, JDEBG )
441:     call LKEEP( 'CI', P5(6), NBANK, 'R8', LASTP, JDEBG )
442:     call LKEEP( 'IWK', P5(7), NBANK, 'I4', LASTP, JDEBG )
443:     call LKEEP( 'JKSF', P5(8), NZONE+1, 'I4', LASTP, JDEBG )
444:     call LKEEP( 'IPZONE', P5(9), NZONE+1, 'I4', LASTP, JDEBG )
445:     call LKEEP( 'MEM', P5(10), NZONE+1, 'I4', LASTP, JDEBG )
446:     if ( JPISX.ne.0 .or. JSTGR.ne.0 ) then
447:       call LKEEP( 'DI', P5(11), NBANK, 'R8', LASTP, JDEBG )
448:       call LKEEP( 'DLOC1', P5(12), NBANK, 'R8', LASTP, JDEBG )
449:       call LKEEP( 'DLOC2', P5(13), NBANK, 'R8', LASTP, JDEBG )
450:       call LKEEP( 'IKL', P5(14), NBANK, 'I4', LASTP, JDEBG )
451:       call LKEEP( 'IKL2', P5(15), NBANK, 'I4', LASTP, JDEBG )
452:       call LKEEP( 'R', P5(16), 3*NBANK, 'I4', LASTP, JDEBG )
453:     end if
454:     if ( LSIZE(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
455:     MAXWK = MAX(MAXWK,LASTP)

```

```

456:       write(IPR,7020) 'LATTICE', LASTP - LWORK, LASTP
457:   end if
458: C
459: C
460: C ===== FOR TALSUM ===== NOV/14/91
461: C
462:   call LSTLST( 'TALSUM', LWORK, LASTP )
463:   call LKEEP( 'FTR', P8(1), NGROUP*NTREG, 'R8', LASTP, JDEBG )
464:   call LKEEP( 'FCL', P8(2), NGROUP*NTREG, 'R8', LASTP, JDEBG )
465:   call LKEEP( 'FLR', P8(3), NTREG*NPKIND, 'R8', LASTP, JDEBG )
466:   call LKEEP( 'DNF', P8(4), NGROUP*NTREG, 'R8', LASTP, JDEBG )
467:   call LKEEP( 'RRR', P8(5), NGROUP*NTREG, 'R8', LASTP, JDEBG )
468:   call LKEEP( 'RTR', P8(6), NTREG*NRESP, 'R8', LASTP, JDEBG )
469:   call LKEEP( 'RCL', P8(7), NTREG*NRESP, 'R8', LASTP, JDEBG )
470:   call LKEEP( 'STR', P8(8), NTREG*NSTAL, 'R8', LASTP, JDEBG )
471:   call LKEEP( 'SCL', P8(9), NTREG*NSTAL, 'R8', LASTP, JDEBG )
472: CCC call LKEEP( 'DWRK', P8(10), NEDTMX, 'R8', LASTP, JDEBG )
473:   call LKEEP( 'DWRK', P8(10), NEDTMX+NDTWMX, 'R8', LASTP, JDEBG )
474:   if ( JEIGN.ne.0 ) then
475:     c##<2007/03/14:PN4:
476:     c## call LKEEP( 'XKEF1', P8(11), 7*NLENG, 'R8', LASTP, JDEBG )
477:     call LKEEP( 'XKEF1', P8(11), MAXXK*NLENG, 'R8', LASTP, JDEBG )
478:     c##>
479:     call LKEEP( 'XSOCB', P8(12), NLENG, 'R8', LASTP, JDEBG )
480:     c##<2007/03/14:PN4:
481:     c## call LKEEP( 'XKB', P8(13), 7*NLENG, 'R8', LASTP, JDEBG )
482:     call LKEEP( 'XKB', P8(13), MAXXK*NLENG, 'R8', LASTP, JDEBG )
483:     c##>
484:   end if
485:   if ( JSCTM.ne.0.or.JSCMU.ne.0 ) then
486:     NN = NGP(KPNEUT)*(NGP(KPNEUT)+1)
487:     call LKEEP( 'DNFSM', P8(14), NN, 'R8', LASTP, JDEBG )
488:     if ( JSCTM.gt.1 ) NN = NN*JSCTM
489:     call LKEEP( 'RRRSM', P8(15), NN, 'R8', LASTP, JDEBG )
490:   end if
491:   if ( LSIZE(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
492:   MAXWK = MAX(MAXWK,LASTP)
493:   write(IPR,7020) 'TALSUM', LASTP - LWORK, LASTP
494: C
495: C
496:   if ( JALLZ.eq.1 ) then
497: C
498: C ===== FOR FLIGHT =====
499: C
500:     call LSTLST( 'FLIGHT', LWORK, LASTP )
501:     call LKEEP( 'KKSF', P6(1), NZONE+1, 'I4', LASTP, JDEBG )
502:     call LKEEP( 'JKSF', P6(2), NZONE+1, 'I4', LASTP, JDEBG )
503:     call LKEEP( 'IPZONE', P6(3), NZONE+1, 'I4', LASTP, JDEBG )
504:     call LKEEP( 'X', P6(4), NBANK, 'R8', LASTP, JDEBG )
505:     call LKEEP( 'Y', P6(5), NBANK, 'R8', LASTP, JDEBG )
506:     call LKEEP( 'Z', P6(6), NBANK, 'R8', LASTP, JDEBG )
507:     call LKEEP( 'AI', P6(7), NBANK, 'R8', LASTP, JDEBG )
508:     call LKEEP( 'BI', P6(8), NBANK, 'R8', LASTP, JDEBG )
509:     call LKEEP( 'CI', P6(9), NBANK, 'R8', LASTP, JDEBG )
510:     call LKEEP( 'W', P6(10), NBANK, 'R8', LASTP, JDEBG )
511:     call LKEEP( 'DI', P6(11), NBANK, 'R8', LASTP, JDEBG )
512:     call LKEEP( 'IGI', P6(12), NBANK, 'I4', LASTP, JDEBG )
513:     call LKEEP( 'IBP', P6(13), NBANK, 'I4', LASTP, JDEBG )
514:     call LKEEP( 'R', P6(14), NBANK, 'R4', LASTP, JDEBG )
515:     call LKEEP( 'D11', P6(15), NBANK, 'R8', LASTP, JDEBG )
516:     call LKEEP( 'D22', P6(16), NBANK, 'R8', LASTP, JDEBG )
517:     call LKEEP( 'DLOC1', P6(17), NBANK, 'R8', LASTP, JDEBG )
518:     call LKEEP( 'DLOC2', P6(18), NBANK, 'R8', LASTP, JDEBG )
519:     call LKEEP( 'LLS', P6(19), NBANK, 'I4', LASTP, JDEBG )
520:     call LKEEP( 'IZT', P6(20), NBANK, 'I4', LASTP, JDEBG )

```

src/mvp/wrkary.f

```

521:      call LKEEP( 'DSDA0 ', P6(21), NBANK, 'R8', LASTP, JDEBG )
522:      call LKEEP( 'DSDA1 ', P6(22), NBANK, 'R8', LASTP, JDEBG )
523:      call LKEEP( 'DSDA2 ', P6(23), NBANK, 'R8', LASTP, JDEBG )
524:      call LKEEP( 'DSDA3 ', P6(24), NBANK, 'R8', LASTP, JDEBG )
525:      call LKEEP( 'DSDA4 ', P6(25), NBANK, 'R8', LASTP, JDEBG )
526:      call LKEEP( 'DSDA5 ', P6(26), NBANK, 'R8', LASTP, JDEBG )
527:      if ( MAXSF.ge.7 )
528:      &      call LKEEP( 'DSDA6 ', P6(27), NBANK, 'R8', LASTP, JDEBG )
529:      if ( MAXSF.ge.8 )
530:      &      call LKEEP( 'DSDA7 ', P6(28), NBANK, 'R8', LASTP, JDEBG )
531:      if ( MAXSF.ge.9 )
532:      &      call LKEEP( 'DSDA8 ', P6(29), NBANK, 'R8', LASTP, JDEBG )
533:      if ( MAXSF.ge.10 )
534:      &      call LKEEP( 'DSDA9 ', P6(30), NBANK, 'R8', LASTP, JDEBG )
535:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
536:      MAXWK = MAX(MAXWK,LASTP)
537:      write(IPR,7020) 'FLIGHT', LASTP - LWORK, LASTP
538: C
539: C ===== FOR SEARCH =====
540: C
541:      call LSTLST( 'SEARCH', LWORK, LASTP )
542:      call LKEEP( 'IZT ', P7(1), NZONE+1, 'I4', LASTP, JDEBG )
543:      call LKEEP( 'IPZONE', P7(2), NZONE+1, 'I4', LASTP, JDEBG )
544:      call LKEEP( 'MEM ', P7(3), NZONE+1, 'I4', LASTP, JDEBG )
545:      call LKEEP( 'JKSF ', P7(4), NZONE+1, 'I4', LASTP, JDEBG )
546:      call LKEEP( 'IWK2 ', P7(5), NZONE+1, 'I4', LASTP, JDEBG )
547:      call LKEEP( 'IWK3 ', P7(6), NZONE+1, 'I4', LASTP, JDEBG )
548:      call LKEEP( 'IWK4 ', P7(7), NZONE+1, 'I4', LASTP, JDEBG )
549:      call LKEEP( 'X ', P7(8), NBANK, 'R8', LASTP, JDEBG )
550:      call LKEEP( 'Y ', P7(9), NBANK, 'R8', LASTP, JDEBG )
551:      call LKEEP( 'Z ', P7(10), NBANK, 'R8', LASTP, JDEBG )
552:      call LKEEP( 'W ', P7(11), NBANK, 'R8', LASTP, JDEBG )
553:      call LKEEP( 'IWK ', P7(12), NBANK, 'I4', LASTP, JDEBG )
554:      call LKEEP( 'IFG ', P7(13), NBANK, 'I4', LASTP, JDEBG )
555:      call LKEEP( 'LLS ', P7(14), NBANK, 'I4', LASTP, JDEBG )
556:      call LKEEP( 'IWK0 ', P7(15), NBANK, 'I4', LASTP, JDEBG )
557:      call LKEEP( 'IFI ', P7(16), NBANK, 'I4', LASTP, JDEBG )
558:      call LKEEP( 'IFL ', P7(17), NBANK, 'I4', LASTP, JDEBG )
559:      call LKEEP( 'ISRF ', P7(18), NBANK, 'I4', LASTP, JDEBG )
560:      call LKEEP( 'FXYZ ', P7(19), NBANK, 'R4', LASTP, JDEBG )
561:      call LKEEP( 'R ', P7(20), NBANK, 'R4', LASTP, JDEBG )
562:      call LKEEP( 'DSDA0 ', P7(21), NBANK, 'R8', LASTP, JDEBG )
563:      call LKEEP( 'DSDA1 ', P7(22), NBANK, 'R8', LASTP, JDEBG )
564:      call LKEEP( 'DSDA2 ', P7(23), NBANK, 'R8', LASTP, JDEBG )
565:      call LKEEP( 'DSDA3 ', P7(24), NBANK, 'R8', LASTP, JDEBG )
566:      call LKEEP( 'DSDA4 ', P7(25), NBANK, 'R8', LASTP, JDEBG )
567:      call LKEEP( 'DSDA5 ', P7(26), NBANK, 'R8', LASTP, JDEBG )
568:      if ( MAXSF.ge.7 )
569:      &      call LKEEP( 'DSDA6 ', P7(27), NBANK, 'R8', LASTP, JDEBG )
570:      if ( MAXSF.ge.8 )
571:      &      call LKEEP( 'DSDA7 ', P7(28), NBANK, 'R8', LASTP, JDEBG )
572:      if ( MAXSF.ge.9 )
573:      &      call LKEEP( 'DSDA8 ', P7(29), NBANK, 'R8', LASTP, JDEBG )
574:      if ( MAXSF.ge.10 )
575:      &      call LKEEP( 'DSDA9 ', P7(30), NBANK, 'R8', LASTP, JDEBG )
576:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
577:      MAXWK = MAX(MAXWK,LASTP)
578:      write(IPR,7020) 'SEARCH', LASTP - LWORK, LASTP
579: C
580:      end if
581: C
582: C ===== FOR CEL2AB used for FISSION source file output =====
583: C
584:      if ( JEIGN.ne.0.and.JLATT.ne.0.and.JOSRC.ne.0 ) then
585:      call LSTLST( 'CEL2AB', LWORK, LASTP )

```

```

586:
587:      call LKEEP( 'X', PCA(1), NFBNK0, 'R8', LASTP, JDEBG )
588:      call LKEEP( 'Y', PCA(2), NFBNK0, 'R8', LASTP, JDEBG )
589:      call LKEEP( 'Z', PCA(3), NFBNK0, 'R8', LASTP, JDEBG )
590:
591:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
592:      MAXWK = MAX(MAXWK,LASTP)
593:      write(IPR,7020) 'CEL2AB', LASTP - LWORK, LASTP
594:      end if
595: C
596: C ===== FOR TALLYO =====
597: C
598:      call LSTLST( 'TALLYO', LWORK, LASTP )
599: C
600:      if ( JEIGN.ne.0 ) then
601: C
602:      call LKEEP( 'XSOCB ', P9(1), NLENG, 'R8', LASTP, JDEBG )
603: c##<2007/03/14:PN4:
604: c##      call LKEEP( 'XKB ', P9(2), 7*NLENG, 'R8', LASTP, JDEBG )
605: c##      call LKEEP( 'COVXK ', P9(3), 6*6*NLENG, 'R8', LASTP, JDEBG )
606: c##      call LKEEP( 'XLMLE ', P9(4), 6*NLENG, 'R8', LASTP, JDEBG )
607: c##      call LKEEP( 'XKERR ', P9(5), 6*NLENG, 'R8', LASTP, JDEBG )
608: c##      call LKEEP( 'CORR ', P9(6), 6*6*3, 'R8', LASTP, JDEBG )
609:      NN = MAXXK*NLENG
610:      call LKEEP( 'XKB ', P9(2), NN, 'R8', LASTP, JDEBG )
611:      NN = MAXXK*MAXXK*NLENG
612:      call LKEEP( 'COVXK ', P9(3), NN, 'R8', LASTP, JDEBG )
613:      NN = MAXXK*NLENG
614:      call LKEEP( 'XLMLE ', P9(4), NN, 'R8', LASTP, JDEBG )
615:      call LKEEP( 'XKERR ', P9(5), NN, 'R8', LASTP, JDEBG )
616:      NN = MAXXK*MAXXK*3
617:      call LKEEP( 'CORR ', P9(6), NN, 'R8', LASTP, JDEBG )
618: c##>
619:      end if
620: C
621: c##<2007/03/14:PN4:
622: c##      call LKEEP( 'IWK1 ', P9(7), 8, 'I4', LASTP, JDEBG )
623: c##      call LKEEP( 'IWK2 ', P9(8), 8, 'I4', LASTP, JDEBG )
624:      NN = max(MAXXK,8)
625:      call LKEEP( 'IWK1 ', P9(7), NN, 'I4', LASTP, JDEBG )
626:      call LKEEP( 'IWK2 ', P9(8), NN, 'I4', LASTP, JDEBG )
627: c##>
628:      call LKEEP( 'UU ', P9(9), NGROUP, 'R4', LASTP, JDEBG )
629:      NN = (NGROUP+NPKIND)*NTREG
630:      call LKEEP( 'WWK ', P9(10), NN, 'R8', LASTP, JDEBG )
631: c##<2007/04/10: bug fix
632: c##      call LKEEP( 'BIN ', P9(11), 2*8*NBINMX, 'R4', LASTP, JDEBG )
633:      call LKEEP( 'BIN ', P9(11), 2*8*NBINMX, 'R8', LASTP, JDEBG )
634: c##>
635: C ... RNM is a character array and should not be kept here
636: CC      call LKEEP( 'RNM ', P9(12), 128*NBINMX, 'R4', LASTP, JDEBG )
637: C
638:      if ( JEIGN.ne.0 ) then
639: c##<2007/03/14:PN4:
640: c##      call LKEEP( 'XKSD', P9(13), 6*NLENG, 'R8', LASTP, JDEBG )
641:      NN = MAXXK*NLENG
642:      call LKEEP( 'XKSD', P9(13), NN, 'R8', LASTP, JDEBG )
643: c##>
644:      call LKEEP( 'XKBB', P9(14), NLENG, 'R8', LASTP, JDEBG )
645:      end if
646:      if ( JEIGN.ne.0 .or. NETRV.ne.0 ) then
647:      call LKEEP( 'CA', P9(15), NMXLG, 'R8', LASTP, JDEBG )
648:      call LKEEP( 'CR', P9(16), NMXLG, 'R8', LASTP, JDEBG )
649:      end if
650:

```

src/mvp/wrkary.f

```

651:      if ( JEIGN.ne.0 .and. JPERT.ne.0 ) then
652:        call LKEEP( 'XKPB ', P9(21), NTPT*7*NLENG*NUCPT, 'R8',
653:          &          LASTP, JDEBG )
654:        call LKEEP( 'XKPSD ', P9(22), NTPT*6*NLENG*NUCPT, 'R8',
655:          &          LASTP, JDEBG )
656:        call LKEEP( 'XKPML ', P9(23), NTPT*6*NLENG*NUCPT, 'R8',
657:          &          LASTP, JDEBG )
658:      end if
659:
660:      if( JEIGN.ne.0 .and. JBEFF.ne.0 ) then
661:        call LKEEP( 'XBB ', P9(24), 9*NLENG, 'R8', LASTP, JDEBG )
662:        call LKEEP( 'XBSD ', P9(25), 6*NLENG, 'R8', LASTP, JDEBG )
663:        call LKEEP( 'COVXB ', P9(26), 3*3*NLENG, 'R8', LASTP, JDEBG )
664:        call LKEEP( 'XBWKC ', P9(27), 3*NLENG, 'R8', LASTP, JDEBG )
665:        call LKEEP( 'XBWKB ', P9(28), 3*NLENG, 'R8', LASTP, JDEBG )
666:        call LKEEP( 'CORRE ', P9(29), 3*3*3, 'R8', LASTP, JDEBG )
667:        call LKEEP( 'XMLE ', P9(30), 3*NLENG, 'R8', LASTP, JDEBG )
668:        call LKEEP( 'XBERR ', P9(31), 3*NLENG, 'R8', LASTP, JDEBG )
669:      end if
670:
671:      if ( LSIZE(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
672:      MAXWK = MAX(MAXWK,LASTP)
673:      write(IPR,7020) 'K-EFF ', LASTP - LWORK, LASTP
674: C
675: C ===== FOR NXTNR ===== Nov 25 1999
676: C
677: C=====
678:      if ( JPTDT.ne.0 ) then
679: C
680:          LASTPC = 0
681:          LASTPP = 0
682:          if ( JNEUT.ne.0 ) then
683:            call LSTLST( 'NXTNR', LWORK, LASTP )
684:            call LKEEP( 'X ', PC(1), NBANK, 'R8', LASTP, JDEBG )
685:            call LKEEP( 'Y ', PC(2), NBANK, 'R8', LASTP, JDEBG )
686:            call LKEEP( 'Z ', PC(3), NBANK, 'R8', LASTP, JDEBG )
687:            call LKEEP( 'A ', PC(4), NBANK, 'R8', LASTP, JDEBG )
688:            call LKEEP( 'B ', PC(5), NBANK, 'R8', LASTP, JDEBG )
689:            call LKEEP( 'C ', PC(6), NBANK, 'R8', LASTP, JDEBG )
690:            call LKEEP( 'AA ', PC(7), NBANK, 'R8', LASTP, JDEBG )
691:            call LKEEP( 'BB ', PC(8), NBANK, 'R8', LASTP, JDEBG )
692:            call LKEEP( 'CC ', PC(9), NBANK, 'R8', LASTP, JDEBG )
693:            call LKEEP( 'DI ', PC(10), NBANK, 'R8', LASTP, JDEBG )
694:            call LKEEP( 'TI ', PC(11), NBANK, 'R8', LASTP, JDEBG )
695:            call LKEEP( 'WW ', PC(12), NBANK, 'R4', LASTP, JDEBG )
696:            call LKEEP( 'IBP ', PC(13), NBANK, 'I4', LASTP, JDEBG )
697:            call LKEEP( 'ICLN ', PC(14), NBANK, 'I4', LASTP, JDEBG )
698:            call LKEEP( 'IMT ', PC(15), NBANK, 'I4', LASTP, JDEBG )
699:            call LKEEP( 'ISF4 ', PC(16), NBANK, 'I4', LASTP, JDEBG )
700:            call LKEEP( 'ISF5 ', PC(17), NBANK, 'I4', LASTP, JDEBG )
701:            call LKEEP( 'ILF ', PC(18), NBANK, 'I4', LASTP, JDEBG )
702:            call LKEEP( 'EIN ', PC(19), NBANK, 'R4', LASTP, JDEBG )
703:            call LKEEP( 'W ', PC(20), NBANK, 'R4', LASTP, JDEBG )
704:            call LKEEP( 'THEA ', PC(21), NBANK, 'R4', LASTP, JDEBG )
705:            call LKEEP( 'THEB ', PC(22), NBANK, 'R4', LASTP, JDEBG )
706:            call LKEEP( 'EMIU ', PC(23), NBANK, 'R4', LASTP, JDEBG )
707:            call LKEEP( 'NRTYP ', PC(24), 10, 'I4', LASTP, JDEBG )
708:            call LKEEP( 'EEEJ ', PC(25), NBANK, 'R4', LASTP, JDEBG )
709:            call LKEEP( 'ITO ', PC(26), NBANK, 'I4', LASTP, JDEBG )
710:            call LKEEP( 'IGO ', PC(27), NBANK, 'I4', LASTP, JDEBG )
711:            call LKEEP( 'ISEL ', PC(28), NBANK, 'I4', LASTP, JDEBG )
712: C
713:          call LSTLST( 'NXTNR', LASTP, LASTPC )
714: C
715:          call LKEEP( 'IWK1 ', PC(29), NBANK, 'I4D', LASTP, JDEBG )

```

```

716:          call LKEEP( 'IWK2 ', PC(30), NBANK, 'I4D', LASTP, JDEBG )
717:          call LKEEP( 'IWK3 ', PC(31), NBANK, 'I4D', LASTP, JDEBG )
718:          call LKEEP( 'IWK3 ', PC(32), NBANK, 'I4D', LASTP, JDEBG )
719:          call LKEEP( 'IWK5 ', PC(33), NBANK, 'I4D', LASTP, JDEBG )
720:          call LKEEP( 'IWK6 ', PC(34), NBANK, 'I4D', LASTP, JDEBG )
721:          call LKEEP( 'IWK7 ', PC(35), NBANK, 'I4D', LASTP, JDEBG )
722:          call LKEEP( 'IWK8 ', PC(36), NBANK, 'I4D', LASTP, JDEBG )
723:          call LKEEP( 'IWK9 ', PC(37), NBANK, 'I4D', LASTP, JDEBG )
724:          call LKEEP( 'IWK10 ', PC(38), NBANK, 'I4D', LASTP, JDEBG )
725:          call LKEEP( 'WK1 ', PC(39), NBANK, 'R4D', LASTP, JDEBG )
726:          call LKEEP( 'WK2 ', PC(40), NBANK, 'R4D', LASTP, JDEBG )
727:          call LKEEP( 'WK3 ', PC(41), NBANK, 'R4D', LASTP, JDEBG )
728:          call LKEEP( 'WK4 ', PC(42), NBANK, 'R4D', LASTP, JDEBG )
729:          call LKEEP( 'WK5 ', PC(43), NBANK, 'R4D', LASTP, JDEBG )
730:          call LKEEP( 'WK6 ', PC(44), NBANK, 'R4D', LASTP, JDEBG )
731:          call LKEEP( 'COSL ', PC(45), NBANK, 'R4D', LASTP, JDEBG )
732:          call LKEEP( 'R ', PC(46), 8*NBANK, 'R4D', LASTP, JDEBG )
733:          call LKEEP( 'SMCW ', PC(47), NSMICI*NBANK, 'R4', LASTP,
734:            &          JDEBG )
735: c##<2007/03/14:PN4:
736:          call LKEEP( 'IWK11 ', PC(48), NBANK, 'I4D', LASTP, JDEBG )
737: c##>
738: C
739:          if ( LSIZE(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
740:          MAXWK = MAX(MAXWK,LASTP)
741:          write(IPR,7020) 'NXTNR ', LASTP - LWORK, LASTP
742:        end if
743: C
744: C ===== FOR NXPHR ===== Feb 23 2000
745: C
746:          if ( JPHOT.ne.0 ) then
747: C
748:            call LSTLST( 'NXPHR', LWORK, LASTP )
749:            call LKEEP( 'X ', PG(1), NBANK, 'R8', LASTP, JDEBG )
750:            call LKEEP( 'Y ', PG(2), NBANK, 'R8', LASTP, JDEBG )
751:            call LKEEP( 'Z ', PG(3), NBANK, 'R8', LASTP, JDEBG )
752:            call LKEEP( 'A ', PG(4), NBANK, 'R8', LASTP, JDEBG )
753:            call LKEEP( 'B ', PG(5), NBANK, 'R8', LASTP, JDEBG )
754:            call LKEEP( 'C ', PG(6), NBANK, 'R8', LASTP, JDEBG )
755:            call LKEEP( 'AA ', PG(7), NBANK, 'R8', LASTP, JDEBG )
756:            call LKEEP( 'BB ', PG(8), NBANK, 'R8', LASTP, JDEBG )
757:            call LKEEP( 'CC ', PG(9), NBANK, 'R8', LASTP, JDEBG )
758:            call LKEEP( 'DI ', PG(10), NBANK, 'R8', LASTP, JDEBG )
759:            call LKEEP( 'TI ', PG(11), NBANK, 'R8', LASTP, JDEBG )
760:            call LKEEP( 'WW ', PG(12), NBANK, 'R4', LASTP, JDEBG )
761:            call LKEEP( 'IBP ', PG(13), NBANK, 'I4', LASTP, JDEBG )
762:            call LKEEP( 'ICLA ', PG(14), NBANK, 'I4', LASTP, JDEBG )
763:            call LKEEP( 'ISTB ', PG(15), NBANK, 'I4', LASTP, JDEBG )
764:            call LKEEP( 'NTBL ', PG(16), NBANK, 'I4', LASTP, JDEBG )
765:            call LKEEP( 'NRTYP ', PG(17), 10, 'I4', LASTP, JDEBG )
766:            call LKEEP( 'EIN ', PG(18), NBANK, 'R4', LASTP, JDEBG )
767:            call LKEEP( 'W ', PG(19), NBANK, 'R4', LASTP, JDEBG )
768:            call LKEEP( 'AAE ', PG(20), NBANK, 'R8', LASTP, JDEBG )
769:            call LKEEP( 'BBE ', PG(21), NBANK, 'R8', LASTP, JDEBG )
770:            call LKEEP( 'CCE ', PG(22), NBANK, 'R8', LASTP, JDEBG )
771:            call LKEEP( 'WE ', PG(23), NBANK, 'R4', LASTP, JDEBG )
772:            call LKEEP( 'EELE ', PG(24), NBANK, 'R4', LASTP, JDEBG )
773:            call LKEEP( 'EBRM ', PG(25), NBANK, 'R4', LASTP, JDEBG )
774:            call LKEEP( 'EEEJ ', PG(26), NBANK, 'R4', LASTP, JDEBG )
775:            call LKEEP( 'WWE ', PG(27), NBANK, 'R4', LASTP, JDEBG )
776:            call LKEEP( 'ITO ', PG(28), NBANK, 'I4', LASTP, JDEBG )
777:            call LKEEP( 'IGO ', PG(29), NBANK, 'I4', LASTP, JDEBG )
778:            call LKEEP( 'ISEL ', PG(30), NBANK, 'I4', LASTP, JDEBG )
779: C
780:          call LSTLST( 'NXPHR', LASTP, LASTPP )

```

src/mvp/wrkary.f

```

781: C
782:      call LKEEP( 'IWK1 ', PG(31), NBANK, 'I4D', LASTP, JDEBG )
783:      call LKEEP( 'IWK2 ', PG(32), NBANK, 'I4D', LASTP, JDEBG )
784:      call LKEEP( 'IWK3 ', PG(33), NBANK, 'I4D', LASTP, JDEBG )
785:      call LKEEP( 'IWK4 ', PG(34), NBANK, 'I4D', LASTP, JDEBG )
786:      call LKEEP( 'IWK5 ', PG(35), NBANK, 'I4D', LASTP, JDEBG )
787:      call LKEEP( 'IWK6 ', PG(36), NBANK, 'I4D', LASTP, JDEBG )
788:      call LKEEP( 'IWK7 ', PG(37), NBANK, 'I4D', LASTP, JDEBG )
789:      call LKEEP( 'IWK8 ', PG(38), NBANK, 'I4D', LASTP, JDEBG )
790:      call LKEEP( 'IWK9 ', PG(39), NBANK, 'I4D', LASTP, JDEBG )
791:      call LKEEP( 'IWK10 ', PG(40), NBANK, 'I4D', LASTP, JDEBG )
792:      call LKEEP( 'WK1 ', PG(41), NBANK, 'R4D', LASTP, JDEBG )
793:      call LKEEP( 'WK2 ', PG(42), NBANK, 'R4D', LASTP, JDEBG )
794:      call LKEEP( 'WK3 ', PG(43), NBANK, 'R4D', LASTP, JDEBG )
795:      call LKEEP( 'WK4 ', PG(44), NBANK, 'R4D', LASTP, JDEBG )
796:      call LKEEP( 'WK5 ', PG(45), NBANK, 'R4D', LASTP, JDEBG )
797:      call LKEEP( 'COSL ', PG(46), NBANK, 'R4D', LASTP, JDEBG )
798:      call LKEEP( 'R ', PG(47), 4*NBANK, 'R4D', LASTP, JDEBG )
799: C      call LKEEP( 'SMCW ', PG(48), NSMICI*NBANK, 'R4', LASTP,
800: C      & JDEBG )
801: C
802:      if ( LSIZL(LASTP).gt.LIMITL ) IOVPL = IOVPL + 1
803:      MAXWK = MAX(MAXWK,LASTP)
804:      write(IPR,7020) 'NXPHR ', LASTP - LWORK, LASTP
805:      end if
806: C
807: C      .... FOR NXTEE ....
808: C
809:      LASTPC = MAX(LASTPC,LASTPP)
810:      LASTPN = MAX(LASTPS,LASTPC)
811: C
812: C      .... FOR NXTFLI CALLED IN NXTEE
813: C
814:      call LSTLST( 'NXTFLI', LASTPN, LASTPF )
815:      call LKEEP( 'X ', PF(1), IMPMAX, 'R8', LASTPF, JDEBG )
816:      call LKEEP( 'Y ', PF(2), IMPMAX, 'R8', LASTPF, JDEBG )
817:      call LKEEP( 'Z ', PF(3), IMPMAX, 'R8', LASTPF, JDEBG )
818:      call LKEEP( 'AI ', PF(4), IMPMAX, 'R8', LASTPF, JDEBG )
819:      call LKEEP( 'BI ', PF(5), IMPMAX, 'R8', LASTPF, JDEBG )
820:      call LKEEP( 'CI ', PF(6), IMPMAX, 'R8', LASTPF, JDEBG )
821:      call LKEEP( 'OPT ', PF(7), IMPMAX, 'R8', LASTPF, JDEBG )
822:      call LKEEP( 'DI ', PF(8), IMPMAX, 'R8', LASTPF, JDEBG )
823:      call LKEEP( 'IGI ', PF(9), IMPMAX, 'I4', LASTPF, JDEBG )
824:      call LKEEP( 'IBP ', PF(10), IMPMAX, 'I4', LASTPF, JDEBG )
825:      call LKEEP( 'IFC ', PF(11), IMPMAX, 'I4', LASTPF, JDEBG )
826:      call LKEEP( 'R ', PF(12), 3*IMPMAX, 'R4', LASTPF, JDEBG )
827:      call LKEEP( 'DLOC1 ', PF(13), IMPMAX, 'R8', LASTPF, JDEBG )
828:      call LKEEP( 'DLOC2 ', PF(14), IMPMAX, 'R8', LASTPF, JDEBG )
829:      call LKEEP( 'IKL ', PF(15), IMPMAX, 'I4', LASTPF, JDEBG )
830:      call LKEEP( 'IKL2 ', PF(16), IMPMAX, 'I4', LASTPF, JDEBG )
831:      if ( JHLAT.ne.0.or.JFISX.ne.0 ) then
832:      call LKEEP( 'XUP ', PF(17), IMPMAX, 'R8', LASTPF, JDEBG )
833:      call LKEEP( 'YUP ', PF(18), IMPMAX, 'R8', LASTPF, JDEBG )
834:      call LKEEP( 'ZUP ', PF(19), IMPMAX, 'R8', LASTPF, JDEBG )
835:      call LKEEP( 'AUP ', PF(20), IMPMAX, 'R8', LASTPF, JDEBG )
836:      call LKEEP( 'BUP ', PF(21), IMPMAX, 'R8', LASTPF, JDEBG )
837:      call LKEEP( 'CUP ', PF(22), IMPMAX, 'R8', LASTPF, JDEBG )
838:      end if
839:      call LKEEP( 'SIGT ', PF(23), IMPMAX, 'R4', LASTPF, JDEBG )
840:      call LKEEP( 'ISTG ', PF(24), IMPMAX, 'I4', LASTPF, JDEBG )
841:      call LKEEP( 'ILUP ', PF(25), IMPMAX, 'I4', LASTPF, JDEBG )
842:      call LKEEP( 'ILST ', PF(26), IMPMAX, 'I4', LASTPF, JDEBG )
843:      if ( JAMXC.ne.0 ) then
844:      call LKEEP( 'SMCW ', P0(27), IMPMAX*NSMICI, 'R4', LASTP,
845:      & JDEBG )

```

```

846:
847: C      end if
848: C      if ( JFISX.ne.0 ) then
849: C      call LKEEP( 'XYZ1 ', PF(27), 3*IMPMAX, 'R8', LASTPF, JDEBG )
850: C      call LKEEP( 'ABC1 ', PF(28), 3*IMPMAX, 'R8', LASTPF, JDEBG )
851: C      call LKEEP( 'XYZ2 ', PF(29), 3*IMPMAX, 'R8', LASTPF, JDEBG )
852: C      call LKEEP( 'ABC2 ', PF(30), 3*IMPMAX, 'R8', LASTPF, JDEBG )
853: C      call LKEEP( 'DDI ', PF(31), IMPMAX, 'R8', LASTPF, JDEBG )
854: C      call LKEEP( 'IBP0 ', PF(32), IMPMAX, 'I4', LASTPF, JDEBG )
855: C      call LKEEP( 'IBP1 ', PF(33), IMPMAX, 'I4', LASTPF, JDEBG )
856: C      call LKEEP( 'IBP2 ', PF(34), IMPMAX, 'I4', LASTPF, JDEBG )
857: C      call LKEEP( 'KPLT ', PF(35), NLBZ+1, 'I4', LASTPF, JDEBG )
858: C      call LKEEP( 'JKSF ', PF(36), NLBZ, 'I4', LASTPF, JDEBG )
859: C      end if
860: C
861: C      .... FOR NEESEA CALLED IN NXTEE
862: C
863:      call LSTLST( 'NEESEA', LASTPN, LASTPS )
864:      call LKEEP( 'X ', PS(1), IMPMAX, 'R8', LASTPS, JDEBG )
865:      call LKEEP( 'Y ', PS(2), IMPMAX, 'R8', LASTPS, JDEBG )
866:      call LKEEP( 'Z ', PS(3), IMPMAX, 'R8', LASTPS, JDEBG )
867:      call LKEEP( 'W ', PS(4), IMPMAX, 'R8', LASTPS, JDEBG )
868:      call LKEEP( 'IBP ', PS(5), IMPMAX, 'I4', LASTPS, JDEBG )
869:      call LKEEP( 'IFI ', PS(6), IMPMAX, 'I4', LASTPS, JDEBG )
870:      call LKEEP( 'FXYZ ', PS(7), IMPMAX, 'R4', LASTPS, JDEBG )
871:      call LKEEP( 'ISRF ', PS(8), IMPMAX, 'I4', LASTPS, JDEBG )
872:      call LKEEP( 'IZI ', PS(9), IMPMAX, 'I4', LASTPS, JDEBG )
873: C      call LKEEP( 'IFL ', PS(10), IMPMAX, 'I4', LASTPS, JDEBG )
874: C
875: C      .... FOR LATICE CALLED IN NXTEE
876: C
877:      call LSTLST( 'LATICE', LASTPN, LASTPL )
878:      if ( JLATT.ne.0 ) then
879:      &      call LKEEP( 'X ', PL(1), IMPMAX, 'R8',
880:      &      LASTPL, JDEBG )
881:      &      call LKEEP( 'Y ', PL(2), IMPMAX, 'R8',
882:      &      LASTPL, JDEBG )
883:      &      call LKEEP( 'Z ', PL(3), IMPMAX, 'R8',
884:      &      LASTPL, JDEBG )
885:      &      call LKEEP( 'AI ', PL(4), IMPMAX, 'R8',
886:      &      LASTPL, JDEBG )
887:      &      call LKEEP( 'BI ', PL(5), IMPMAX, 'R8',
888:      &      LASTPL, JDEBG )
889:      &      call LKEEP( 'CI ', PL(6), IMPMAX, 'R8',
890:      &      LASTPL, JDEBG )
891:      &      call LKEEP( 'IWK ', PL(7), IMPMAX, 'I4',
892:      &      LASTPL, JDEBG )
893:      &      call LKEEP( 'JKSF ', PL(8), NZONE+1, 'I4',
894:      &      LASTPL, JDEBG )
895:      &      call LKEEP( 'IPZONE', PL(9), NZONE+1, 'I4',
896:      &      LASTPL, JDEBG )
897:      &      call LKEEP( 'MEM ', PL(10), NZONE+1, 'I4',
898:      &      LASTPL, JDEBG )
899:      if ( JFISX.ne.0 .or. JSTGR.ne.0 ) then
900:      &      call LKEEP( 'DI ', PL(11), IMPMAX, 'R8',
901:      &      LASTPL, JDEBG )
902:      &      call LKEEP( 'DLOC1 ', PL(12), IMPMAX, 'R8',
903:      &      LASTPL, JDEBG )
904:      &      call LKEEP( 'DLOC2 ', PL(13), IMPMAX, 'R8',
905:      &      LASTPL, JDEBG )
906:      &      call LKEEP( 'IKL ', PL(14), IMPMAX, 'I4',
907:      &      LASTPL, JDEBG )
908:      &      call LKEEP( 'IKL2 ', PL(15), IMPMAX, 'I4',
909:      &      LASTPL, JDEBG )
910:      &      call LKEEP( 'R ', PL(16), 3*IMPMAX, 'I4',
911:      &      LASTPL, JDEBG )

```

src/mvp/wrkary.f

```
911:          end if
912:        end if
913: C
914:          LASTP  = MAX(LASTPF,LASTPS,LASTPL)
915:          if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
916: C
917:          MAXWK  = MAX(MAXWK,LASTP)
918:          write(IPR,7020) 'NXTEE ', LASTP - LWORK, LASTP
919: C
920:        end if
921: C
922: C ===== FOR "underground" modules =====
923: C Working area for "underground" modules must be put after any
924: C ordinary working area (
925: C
926:          if ( JUNDG.ne.0 ) then
927:            LWORK2 = MAXWK
928:            call WKAA2Z( 'MVP', A, H, LIMIT, LIMITL, JLATT, JHEAT, JLLT,
929:              &          JDEBG, IOVFL, LWORK2, MAXWK )
930:          end if
931: C
932: C=====
933: C
934: 7020 format(/1X,' == SUBROUTINE <A6,> NEEDS WORKING AREA OF ',I12,
935:    &        ' WORDS. ==== ( LAST POSITION ',I12,')')
936: C
937: C
938: Cccc call lgtlst( last )
939:          if ( LSIZL(MAXWK).gt.LIMITL ) then
940:            write(IPR,7040) LSIZL(MAXWK), LIMITL
941: 7040    format(1X,'XXX INSUFFICIENT TASK LOCAL MEMORY TO KEEP',
942:    &        ' WORKING AREA.'/1X,' ( NECESSARY ',I12,
943:    &        ' WORDS. LIMIT ',I12,'WORDS)')
944:            call CNTERR( 'FATAL' )
945:          end if
946:          return
947:        end
```

src/mvp/wwdprs.f

```

1:      subroutine WWDPRS( ICSP,      NRS,      WBC, IRAND,
2:      S  LSCOL,  NCOLS,  LSDED,  NDEAD,
3:      &  NBANK,  NZONE,  NGROUP,  NREG,  NTIME,  NEST,
4:      V  WLLIM,  KZREG,  NCNTR,  WCNTR,  NEVENT,
5:      C  NKILD,  WKILD,  NSURV,  WSURV,  NSPLT,  WSPLT,
6:      B  XXX,    YYY,    ZZZ,    AAA,    BBB,    CCC,
7:      B  TTT,    WWW,    EEE,    XIM,
8:      B  KKP,    IZZ,    IGG,    ITT,    LEVL,  LZZ,
9:      B  LPOS,   LCRS,   KLSF,   IBREG,   IBSPC,
10: c  B  NSMAC,   NSMIC,   MB,     NUC,     NSTAL,
11: c  B  SMAC,    KMAC,    MMAC,    SMIC,    KSPI,   SGTAL,
12: c  B  DBNK,    KDBNK,   MDBNK,   IBNK,    KIBNK,  MIBNK,
13: c  W  R,       W1,      W2,      P1,      ISP,    IPT,
14: c  &  SMICP,   SMACP,
15: c  &  NPTDS,   NPTCS,   NGSP,    WWD,     WWD2,   WWR,    WSD,    WSC,
16: c  &  NORDDS,  NOPSDS,
17: c##<2007/03/14:PN3:
18: c  &  NUCPN,  NSMICPN, SMICPN
19: c##>
20: c+beff2
21: c  &  WWB,   WSB,  NPTBE,
22: c  &  WWBD,  WSD,  WWLD,  WSLD
23: c-beff2
24: c  & )
25: C=<MVP>=====
26: C PURPOSE: Weight Window for Doppler Resonance Elastic Scattering (DPRS)
27: C          in order to set
28: C          WBC*RKILL < WWW < WBC*RSPLIT
29: C          RKILL, RSPLIT: Parameters defined in this routine
30: C          WWW(IP): current weight, which would be modified
31: C          WBC(IP): weight before correction for DPRS
32: C          ICSP(J): collision stack pointer, IP=LSCOL(ICSP(J))
33: C          ICSP(J) < ICSP(J+1)
34: C          NRS      : number of neutrons processed
35: C          ==> NCOLS, LSCOL, NDEAD, LSDED and related BANK data would be
36: C             modified.
37: C
38: C CALLED IN: NEUTR
39: C
40: C=====
41: C-----
42: C  CONTENTS OF COLLISION STACK & NCOLS WILL BE CHANGED.
43: C-----
44: C
45: c      include 'INC/_KPIDS'
46: c      include 'INC/_NGPS'
47: C
48: c      include 'INC/_FLAGS'
49: C
50: C ... VARIANCE REDUCTION PARAMETERS ...
51: C
52: c      real WLLIM
53: C
54: C ... Collision Stack Pointers and Bank to be processed
55: C
56: c      integer ICSP(NBANK)
57: c      real WBC(NBANK)
58: C
59: C ... COLLISION STACK .....
60: C
61: c      integer LSCOL(NBANK)
62: C
63: C ... DEAD PARTICLE STACK.....
64: C
65: c      integer LSDED(NBANK), NDEAD

```

```

66: C
67: C ... COUNTER .....
68: C
69: c      real*8 WKILD(NGROUP,NREG), WSURV(NGROUP,NREG), WSPLT(NGROUP,NREG)
70: c      real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG), NSPLT(NGROUP,NREG)
71: c      real*8 WCNTR(NEVENT,KPLIM), NCNTR(NEVENT,KPLIM)
72: C
73: C ... BANK .....
74: C
75: c      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
76: c      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
77: c      real WWW(NBANK), XIM(NBANK), EEE(NBANK)
78: c      real*8 TTT(NBANK)
79: c      integer ITT(NBANK)
80: c      integer KKP(NBANK)
81: c      integer IGG(NBANK), IZZ(NBANK), LEVL(NBANK), LPOS(NBANK,NEST),
82: c      &      LZZ(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
83: c      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
84: C
85: C ... SIGMA BANK .....
86: C
87: c      real SMAC(NBANK,MB,NSMAC), SMIC(NBANK,NUC,NSMIC),
88: c      &      SGTAL(NBANK,NSTAL)
89: c##<2007/03/14:PN3:
90: c      real SMICPN(NBANK,NUCPN,NSMICPN)
91: c##>
92: c      integer KMAC(NBANK,MB,2), MMAC(NBANK,2), KSPI(NBANK,NUC)
93: C
94: C ... variables for perturbation calculation ...
95: C
96: c      real*8 WWD(NBANK,NPTDS,NORDDS), WWD2(NBANK,NPTDS)
97: c      real*8 WSD(NBANK,0:NGSP,NPTDS,NOPSDS)
98: c      real*8 WWR(NBANK,NPTCS)
99: c      real*8 WSC(NBANK,0:NGSP,NPTCS)
100: c      real SMACP(NBANK,MB,NSMAC), SMICP(NBANK,NUC,NSMIC)
101: c+beff2
102: c      real*8 WWB(NBANK)
103: c      real*8 WSB(NBANK,0:NGSP)
104: c      real*8 WWBD(NBANK)
105: c      real*8 WSD(NBANK,NGSP)
106: c      real*8 WWLD(NBANK)
107: c      real*8 WSLD(NBANK,NGSP)
108: c-beff2
109: C
110: C ... optional bank parameters
111: C
112: c      real*8 DBNK(NBANK,*)
113: c      integer KDBNK(0:MDBNK)
114: c      integer IBNK(NBANK,*)
115: c      integer KIBNK(0:MIBNK)
116: C
117: C ... REGION NUMBERS .....
118: C
119: c      integer KZREG(NZONE)
120: C
121: C ... WORKING ARRAYS .....
122: C
123: c      integer ISP(NBANK), IPT(NBANK)
124: c      real R(NBANK), W1(NBANK), W2(NBANK), P1(NBANK)
125: C
126: C ... Parameters for weight window .....
127: C
128: c      parameter( RKILL = 0.5 )
129: c      parameter( RSPLIT = 2.0 )
130: C

```

src/mvp/wwdprs.f

```

131: C-----
132: C
133:       WMIN      = WLLIM
134: C
135:       IPAR       = 1
136: C
137: C==== WEIGHT WINDOW =====
138: C
139: C ISP(J) : A PARTICLE SPLITS INTO ISP(J) PARTICLES (ISP(J) > 1)
140: C          OR DIES (ISP(J)=0) OR NEITHER SPLITS NOR DIES (ISP(J)=1).
141: C W1(J)   : WEIGHT BEFORE SPLITTING.
142: C W2(J)   : WEIGHT THAT IS GIVEN TO PARTICLES AFTER WEIGHT-WINDOWE
143: C          IF SPLITTING IS ALLOWED.
144: C
145: C check
146: C write(6,*) ' ##### NRS, NCOLS, NDEAD, NBANK =',
147: C &          NRS, NCOLS, NDEAD, NBANK
148: C write(6,*) '      ICSP(J) = ', (ICSP(J),j=1,NRS)
149: C write(6,*) '      LSCOL(I) = ', (LSCOL(ICSP(J)),j=1,NRS)
150: C write(6,*) '      WWW/WBC = ',
151: C &          (WWW(LSCOL(ICSP(J)))/WBC(LSCOL(ICSP(J))),j=1,NRS)
152: C
153:       NSPL       = 0
154:       NWW        = 0
155: C
156:       call RANU2( IRAND, R, NRS, ICON )
157: C
158: *VOCL LOOP,NOVREC
159: do J = 1, NRS
160:   I = ICSP(J)
161:   W1J = WWW(LSCOL(I))
162: C
163:   WK = WBC(LSCOL(I)) * RKILL
164:   WS = WBC(LSCOL(I))
165:   WK = MAX(WMIN,WK)
166:   WS = MAX(WMIN,WS)
167:   WMAX = RSPLIT * WS
168:   WMAX = MIN(WMAX,2.0)
169: C
170: C ... NWW : number of particles with weights outside of wright window
171:   if ( W1J.gt.WMAX .or. W1J.lt.WK ) then
172:     NWW = NWW + 1
173:     ICSP(NWW) = ICSP(J)
174:     ISP(NWW) = W1J / WS + R(J)
175:     W1(NWW) = W1J
176:     W2(NWW) = WS
177:   end if
178: end do
179: C check
180: C write(6,*) '      ISP(J) = ', (ISP(J),j=1,NRS)
181: C
182: C-----
183: C REMOVE KILLED PARTICLES FROM COLLISION STACK ====
184: C-----
185: C
186:       ND = NDEAD
187: *VOCL LOOP,NOVREC
188: CM do J = 1, NRS
189:   do J = 1, NWW
190:     if ( ISP(J).eq.0 ) then
191:       I = ICSP(J)
192:       ND = ND + 1
193:       LSDDED(ND) = LSCOL(I)
194:     end if
195:   end do

```

```

196: C
197: C .... monitor
198: C
199: *VOCL LOOP,NOVREC
200: CM do J = 1, NRS
201:   do J = 1, NWW
202:     if ( ISP(J).eq.0 ) then
203:       WCNTR(9,IPAR) = WCNTR(9,IPAR) + W1(J)
204:     else if ( W1(J).lt.W2(J) ) then
205:       WCNTR(10,IPAR) = WCNTR(10,IPAR) + W2(J)
206:       NCNTR(10,IPAR) = NCNTR(10,IPAR) + 1.0D0
207:     end if
208:   end do
209:   NCNTR(9,IPAR) = NCNTR(9,IPAR) + ND - NDEAD
210: C
211:   if ( JMNTN.ne.0 ) then
212: *VOCL LOOP,SCALAR
213: CM do J = 1, NRS
214:   do J = 1, NWW
215:     I = ICSP(J)
216:     IGT = IGG(LSCOL(I))
217:     if ( JTLLT.eq.0 ) then
218:       IRT = KZREG(IZZ(LSCOL(I)))
219:     else
220:       IRT = IBREG(LSCOL(I))
221:     end if
222:     if ( ISP(J).eq.0 ) then
223:       NKILD(IGT,IRT) = NKILD(IGT,IRT) + 1.0D0
224:       WKILD(IGT,IRT) = WKILD(IGT,IRT) + W1(J)
225:     else if ( W1(J).lt.W2(J) ) then
226:       NSURV(IGT,IRT) = NSURV(IGT,IRT) + 1.0D0
227:       WSURV(IGT,IRT) = WSURV(IGT,IRT) + W2(J)
228:     end if
229:   end do
230:   end if
231: C
232: C-----
233: C COMPRESS COLLISION STACK IF THERE ARE ANY KILLED PARTICLES.
234: C-----
235: C
236:   if ( ND.gt.NDEAD ) then
237:     do I = 1, NCOLS
238:       IPT(I) = 0
239:     end do
240: CM do J = 1, NRS
241:   do J = 1, NWW
242:     I = ICSP(J)
243:     IPT(I) = J
244:     if ( ISP(J).eq.0 ) LSCOL(I) = 0
245:   end do
246: C
247:   II = 0
248:   NSV = 0
249:   NDEAD = ND
250: C
251: *VOCL LOOP,NOVREC
252:   do I = 1, NCOLS
253:     if ( LSCOL(I).gt.0 ) then
254:       II = II + 1
255:       LSCOL(II) = LSCOL(I)
256: C
257:     if ( IPT(I).gt.0 ) then
258:       NSV = NSV + 1
259:       ICSP(NSV) = II
260:       W1(NSV) = W1(IPT(I))

```


src/mvp/wwdprs.f

```

261:          W2(NSV)      = W2(IPT(I))
262:          ISP(NSV)      = ISP(IPT(I))
263:          end if
264: C
265:          end if
266:          end do
267: C
268:          NCOLS      = II
269: C
270:          if ( NSV.eq.0 ) go to 9000
271: C
272:          else
273: CM          NSV      = NRS
274:          NSV      = NWW
275:          end if
276: C
277: C  ISP(J) : NUMBER OF PARTICLES GENERATED BY SPLITTING
278: C
279:          do J = 1, NSV
280:              ISP(J) = ISP(J) + 1
281:              NSPL   = NSPL + ISP(J)
282:          end do
283: check
284: c  write(6,*) ' NRS, NSV=',NRS,NSV
285: c  write(6,*) ' ISP(J) =', (ISP(J),j=1,NSV)
286: c  write(6,*) ' ICSP(J) =', (ICSP(J),j=1,NSV)
287: C-----
288: C      CHECK WHETHER ALL SPLITTINGS ARE ALLOWED OR NOT.
289: C-----
290: C
291: C-----
292: C      NSPL = 0 : NO NEW PERTICLE IS BORN ! CHANGE WEIGHTS ONLY
293: C      AND RETURN.
294: C-----
295: C
296:          if ( NSPL.eq.0 ) then
297: *VOCL LOOP,NOVREC
298:              do J = 1, NSV
299:                  WWW(LSCOL(ICSP(J))) = W2(J)
300:              end do
301:              go to 9000
302:          end if
303: C-----
304: C      NSPL > NDEAD : TOO MANY PARTICLES ARE BORN! SOME SPLITTINGS
305: C      ARE PREVENTED.
306: C-----
307:          if ( NSPL.gt.NDEAD ) then
308: C-----
309: C      ALL SPLITTINGS ARE PREVENTED ! MONITOR ONLY ! AND RETURN
310: C-----
311:          if ( NDEAD.eq.0 ) then
312: *VOCL LOOP,NOVREC
313:              do J = 1, NSV
314:                  if ( ISP(J).le.0.and.W1(J).lt.W2(J) ) then
315:                      I = ICSP(J)
316:                      WWW(LSCOL(I)) = W2(J)
317:                  end if
318:              end do
319: C
320: *VOCL LOOP,NOVREC
321:              do J = 1, NSV
322:                  if ( ISP(J).gt.0 ) then
323:                      WCNTR(6,IPAR) = WCNTR(6,IPAR) + W1(J)
324:                      NCNTR(6,IPAR) = NCNTR(6,IPAR) + 1.0D0
325:                  end if

```

```

326:          end do
327: C
328:          go to 9000
329:          end if
330: C
331: C-----
332: C      SELECT PARTICLES TO SPLIT.
333: C-----
334: C
335:          FS      = NDEAD/FLOAT(NSPL)
336:          NS      = 0
337:          do J = 1, NSV
338:              P1(J) = FS**ISP(J)
339:              ISP(J) = -ISP(J)
340:          end do
341: C
342: C      ..... REPEAT RANDOM SAMPLING UPTO 10 TIMES .....
343: C
344:          do K = 1, 10
345:              call RANU2( IRAND, R, NSV, ICON )
346:              do J = 1, NSV
347:                  if ( R(J).lt.-ISP(J)*P1(J) ) then
348:                      NS      = NS - ISP(J)
349: C
350: C      ..... RECOVER PLUS SIGN OF ISP FOR SELECTED PARTICLES....
351: C
352:                      if ( NS.le.NDEAD ) ISP(J) = -ISP(J)
353:                  end if
354:              end do
355:              if ( NS.ge.NDEAD ) go to 100
356:          end do
357:          100 continue
358: C
359: C-----
360: C      PREVENT SPLITTING FOR PARTICLES WHOSE ISP < 0
361: C-----
362: C
363:          NSPL = 0
364: *VOCL LOOP,NOVREC
365:          do J = 1, NSV
366:              if ( ISP(J).lt.0 ) then
367: C
368: C      ... restore weight
369:              I = ICSP(J)
370:              WWW(LSCOL(I)) = W2(J)*(1-ISP(J))
371:          else
372:              NSPL = NSPL + ISP(J)
373:          end if
374:          end do
375: C
376: *VOCL LOOP,NOVREC
377:          do J = 1, NSV
378:              if ( ISP(J).lt.0 ) then
379:                  WCNTR(6,IPAR) = WCNTR(6,IPAR) + W1(J)
380:                  NCNTR(6,IPAR) = NCNTR(6,IPAR) + 1.0D0
381:              end if
382:          end do
383:          end if
384: C
385: C-----
386: C      RENEW WEIGHTS IN THE BANK & MONITOR FOR CREATED PARTICLES
387: C-----
388: C
389: *VOCL LOOP,NOVREC
390:          do J = 1, NSV

```

src/mvp/wwdprs.f

```

391:      if ( ISP(J).ge.0 ) then
392:        I = ICSP(J)
393:        WWW(LSCOL(I)) = W2(J)
394:      end if
395:    end do
396:  C
397:    if ( NSPL.eq.0 ) go to 9000
398:  C
399:  *VOCL LOOP,NOVREC
400:    do J = 1, NSV
401:      if ( ISP(J).ge.0 ) then
402:        WCNTR(5,IPAR) = WCNTR(5,IPAR) + ISP(J)*W2(J)
403:      end if
404:    end do
405:    WCNTR(5,IPAR) = WCNTR(5,IPAR) + NSPL
406:  C
407:  C-----
408:  C MX: THE MAXIMUM NUMBER OF NEWLY GENERATED PARTICLES FROM ONE PARTICLE
409:  C IF MX = 0, THERE ARE NO NEED TO SPLIT !!
410:  C-----
411:    MX = 0
412:    do J = 1, NSV
413:      MX = MAX(MX,ISP(J))
414:    end do
415:  C
416:  C-----
417:  C DETERMINE BANK POINTERS FOR EACH GENERATED PARTICLE,
418:  C REGISTER THEM TO COLLISION STACK
419:  C AND MEMORIZE BANK-POINTERS OF THEIR MOTHER-PARTICLES (IPT)
420:  C-----
421:    NSP = NCOLS
422:    NDEAD = NDEAD - NSPL
423:    NDD = NDEAD - NCOLS
424:  C
425:    do M = 1, MX
426:  *VOCL LOOP,NOVREC
427:      do J = 1, NSV
428:        if ( ISP(J).ge.M ) then
429:          NSP = NSP + 1
430:          I = ICSP(J)
431:          IPT(NSP) = LSCOL(I)
432:          LSCOL(NSP) = LSDED(NDD+NSP)
433:        end if
434:      end do
435:    end do
436:  C
437:  C=====
438:  C=== SPLITTING STARTS HERE !! =====
439:  C=====
440:  C
441:    IS = NCOLS + 1
442:    IE = NSP
443:  C-----
444:  C POSITION, DIRECTION, ENERGY, ZONE & TIME
445:  C-----
446:  *VOCL LOOP,NOVREC
447:    do I = IS, IE
448:      XXX(LSCOL(I)) = XXX(IPT(I))
449:      YYY(LSCOL(I)) = YYY(IPT(I))
450:      ZZZ(LSCOL(I)) = ZZZ(IPT(I))
451:      AAA(LSCOL(I)) = AAA(IPT(I))
452:      BBB(LSCOL(I)) = BBB(IPT(I))
453:      CCC(LSCOL(I)) = CCC(IPT(I))
454:      WWW(LSCOL(I)) = WWW(IPT(I))
455:      KKP(LSCOL(I)) = KKP(IPT(I))

```

```

456:      IZZ(LSCOL(I)) = IZZ(IPT(I))
457:      IGG(LSCOL(I)) = IGG(IPT(I))
458:      TTT(LSCOL(I)) = TTT(IPT(I))
459:      KLSF(LSCOL(I)) = KLSF(IPT(I))
460:      EEE(LSCOL(I)) = EEE(IPT(I))
461:      if ( JIMPT.ne.0 ) XIM(LSCOL(I)) = XIM(IPT(I))
462:      IBREG(LSCOL(I)) = IBREG(IPT(I))
463:      if ( JTIME.ne.0 ) ITT(LSCOL(I)) = ITT(IPT(I))
464:    end do
465:
466:    if ( JPERT.ne.0 ) then
467:  *VOCL LOOP,NOVREC
468:      do I = IS, IE
469:        do IPTX = 1, NPTDS
470:          do IOD = 1, NORDDS
471:            WWD(LSCOL(I),IPTX,IOD) = WWD(IPT(I),IPTX,IOD)
472:          end do
473:          WWD2(LSCOL(I),IPTX) = WWD2(IPT(I),IPTX)
474:          do ISPX = 0, NGSP
475:            do IOP = 1, NOPSDS
476:              WSD(LSCOL(I),ISPX,IPTX,IOP) =
477:                & WSD(IPT(I),ISPX,IPTX,IOP)
478:            end do
479:            WWT(LSFFL(I),IPTX) = WWT(IPT(I),IPTX)
480:          end do
481:        end do
482:      do IPTX = 1, NPTCS
483:        WWR(LSCOL(I),IPTX) = WWR(IPT(I),IPTX)
484:        do ISPX = 0, NGSP
485:          WSC(LSCOL(I),ISPX,IPTX) = WSC(IPT(I),ISPX,IPTX)
486:        end do
487:      end do
488:    end do
489:  c+beff2
490:    if ( NPTBE.gt.0 ) then
491:      WWB(LSCOL(I)) = WWB(IPT(I))
492:      do ISPX = 0, NGSP
493:        WSB(LSCOL(I),ISPX) = WSB(IPT(I),ISPX)
494:      end do
495:      WWBD(LSCOL(I)) = WWBD(IPT(I))
496:      WWLD(LSCOL(I)) = WWLD(IPT(I))
497:      do ISPX = 1, NGSP
498:        WSD(LSCOL(I),ISPX) = WSD(IPT(I),ISPX)
499:        WSLD(LSCOL(I),ISPX) = WSLD(IPT(I),ISPX)
500:      end do
501:    end if
502:  c-beff2
503:    end do
504:  end if
505:
506:  C-----
507:  C LATTICE GEOMETRY PARAMETER
508:  C-----
509:    if ( JLATT.ne.0 ) then
510:  *VOCL LOOP,NOVREC
511:      do I = IS, IE
512:        LEVL(LSCOL(I)) = LEVL(IPT(I))
513:      end do
514:      do K = 1, NEST
515:  *VOCL LOOP,NOVREC
516:        do I = IS, IE
517:          LZZ(LSCOL(I),K) = LZZ(IPT(I),K)
518:          LPOS(LSCOL(I),K) = LPOS(IPT(I),K)
519:          if ( JHLAT.ne.0 ) LCRS(LSCOL(I),K) = LCRS(IPT(I),K)
520:          if ( JTLT.ne.0 ) IBSPC(LSCOL(I),K) = IBSPC(IPT(I),K)

```

src/mvp/wwdprs.f

```

521:          end do
522:        end do
523:      end if
524: C-----
525: C   SIGMA BANK for microscopic cross sections
526: C   ( Not performed when called before calculating them)
527: C-----
528: C
529: C   do K = 1, NUC
530: C     do L = 1, NSMIC
531: C       *VOCL LOOP,NOVREC
532: C         do I = IS, IE
533: C           SMIC(LSCOL(I),K,L) = SMIC(IPT(I),K,L)
534: C           if ( JPERT.ne.0 )
535: C             & SMICP(LSCOL(I),K,L) = SMICP(IPT(I),K,L)
536: C           end do
537: C         end do
538: C       end do
539: C###<2007/03/14:PN3:
540: C
541: C   if ( JPHNU.ne.0 ) then
542: C     do K = 1, NUCPN
543: C       do L = 1, NSMICPN
544: C         *VOCL LOOP,NOVREC
545: C           do I = IS, IE
546: C             SMICPN(LSCOL(I),K,L) = SMICPN(IPT(I),K,L)
547: C           end do
548: C         end do
549: C       end do
550: C     end if
551: C###>
552: C
553: C   do K = 1, NUC
554: C     *VOCL LOOP,NOVREC
555: C       do I = IS, IE
556: C         KSPI(LSCOL(I),K) = KSPI(IPT(I),K)
557: C       end do
558: C     end do
559: C   do K = 1, NSTAL
560: C     *VOCL LOOP,NOVREC
561: C       do I = IS, IE
562: C         SGTAL(LSCOL(I),K) = SGTAL(IPT(I),K)
563: C       end do
564: C     end do
565: C
566: C-----
567: C   optional bank parameters
568: C-----
569: C   do K = 1, KDBNK(0)
570: C     *VOCL LOOP,NOVREC
571: C       do I = IS, IE
572: C         DBNK(LSCOL(I),K) = DBNK(IPT(I),K)
573: C       end do
574: C     end do
575: C   do K = 1, KIBNK(0)
576: C     *VOCL LOOP,NOVREC
577: C       do I = IS, IE
578: C         IBNK(LSCOL(I),K) = IBNK(IPT(I),K)
579: C       end do
580: C     end do
581: C
582: C..... MONITOR ON
583: C
584: C   if ( JMNTR.ne.0 ) then
585: C     *VOCL LOOP,SCALAR

```

```

586:       do I = IS, IE
587:         if ( JTLT.eq.0 ) then
588:           IRT = KZREG(IZZ(LSCOL(I)))
589:         else
590:           IRT = IBREG(LSCOL(I))
591:         end if
592:         IGT = IGG(LSCOL(I))
593:         NSPLT(IGT,IRT) = NSPLT(IGT,IRT) + 1.0D0
594:         WSPLT(IGT,IRT) = WSPLT(IGT,IRT) + WWW(LSCOL(I))
595:       end do
596:     end if
597: C
598: C
599: C==== Modify NCOLS =====
600: C
601: C   NCOLS = IE
602: C
603: C
604: C   9000 continue
605: C
606: C   check
607: C     write(6,*) ' # END NSPL, NCOLS, NDEAD, NBANK =',
608: C     & NSPL,NCOLS,NDEAD, NBANK
609: C     return
610: C   end

```

src/mvp/xsecd.f

```
1:      subroutine XSECD( ICRES, KCRES, CRES, LCRES, LAST, LIMIT,  
2:      & NLTEMP,NSTAL, NICRES,MCRES )  
3: C=<MVP>=====
```

4: C PURPOSE : INPUT & PROCESSING OF RPOINT-WISE RESPONSE FUNCTIONS
5: C CALLED IN: INTRO2
6: C CALLS :
7: C=====

```
8:      implicit real(A-H,O-Z)  
9:      integer ICRES(NICRES), KCRES(NSTAL,4)  
10:     real CRES(*)  
11:     include '../shared/INC/_WORDL'  
12:     include '../shared/INC/_IOUNIT'  
13:     include 'INC/_IOUNIT2'  
14:  
15: C/IF PARA(PVM)  
16: *   include '../shared/INC/_PVMPARA'  
17: C/ELSEIF PARA(MPI)  
18: *   include 'mpif.h'  
19: C/ENDIF  
20:     include '../shared/INC/_TASKDT'  
21: C  
22: C ... local variables  
23: C  
24:     character ID*72  
25:     character MT*72  
26:     character*6 TAG  
27:     character*12 THEAD  
28:     logical OPEND, NMD  
29:     character PATH*128  
30:     character FILE*256  
31:     character LINE*80  
32:     character CWRK*256  
33:     integer NXS(16), JXS(32), IDUM(1000)  
34: C  
35:     data PATH /' '/  
36:     data LPATH /0/  
37: C  
38: C-----  
39: C  
40: C     call HEADER( IPR, 'POINT-WISE RESPONSE FUNCTIONS' )  
41: C     call LABEL( IPR, 'POINT-WISE RESPONSE FUNCTIONS' )  
42: C-----  
43: C  INITIALIZATION  
44: C-----  
45: C  
46:     MCRES = 0  
47: C  
48:     LINE = ''  
49:     read(IDXD,fmt ='(A)') LINE  
50: C  
51: C .... GET ID CHARACTER OR 'PATH' STRING ....  
52: C  
53:     IS = 1  
54:     IE = 8  
55: C  
56: C  
57: C .... PATH NAME ....  
58: C  
59:     if ( LINE(1:8).eq.'DATAPATH' .or. LINE(1:8).eq.'datapath' ) then  
60: C  
61: C         .... GET PATH NAME ...  
62: C  
63:         IEQ = INDEX(LINE,'=')  
64:         if ( IEQ.ne.0 ) IE = IEQ  
65:         IS = IE + 1
```

```
66:         call NOBLNK( LINE, IS, IE, LEN(LINE) )  
67:         IE = IS + MIN(LEN(PATH)-1,IE-IS)  
68:         if ( IE.ge.IS ) then  
69:             PATH = LINE(IS:IE)  
70:             LPATH = IE - IS + 1  
71:         end if  
72: C  
73: C/IF .NOT.NOGETENV  
74: C  
75: C ... When PATH includes "$xxxx", expand it with environment variable  
76: C $xxxx . Here "xxxx" contains alphanumeric characters and "_".  
77: C  
78:     KK = INDEX(PATH,'$')  
79:     if ( KK.ne.0 ) then  
80:         call ENVEXP( PATH, CWRK, IOPR, IERR )  
81:         if ( IERR.ne.0 ) then  
82:             write(IMSG,7000) PATH(:ICLEN2(PATH))  
83:             7000 format(1x,'!!!(XSECD) Failed to expand $xxx string ',  
84:             & ' in PATH line of library index file.'/1x,  
85:             & ' <PATH ',A,'>')  
86:             call CNTERR( 'WARNING' )  
87:         else  
88:             PATH = CWRK  
89:             LPATH = ICLEN2(PATH)  
90:         end if  
91:     end if  
92: C/ENDIF  
93: C  
94:     end if  
95: C  
96: C/IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)  
97: C  
98:     if ( LPATH.gt.0 ) then  
99: C  
100: C ... when PATH is given by relative path ...  
101: C  
102: C UNIX : not start with '/'  
103: C MS-DOS(Windows) : not have disk drive letter ( "[a-z]:" )  
104: C MacOS : beginning with ":"  
105: C  
106: C/IF UNIX  
107:     if ( PATH(LPATH:LPATH).ne.'/' ) then  
108:         LPATH = LPATH + 1  
109:         PATH(LPATH:LPATH) = '/'  
110:     end if  
111:     if ( PATH(1:1).ne.'/' ) then  
112: C/ELSEIF FILENAME(DOS)  
113: *     if ( PATH(LPATH:LPATH).ne.CHAR(92) ) then  
114: *         LPATH = LPATH + 1  
115: *         PATH(LPATH:LPATH) = CHAR(92)  
116: *     end if  
117: *     if ( PATH(2:2).ne.':' ) then  
118: C/ELSEIF FILENAME(MACOS)  
119: *     if ( PATH(LPATH:LPATH).ne.':' ) then  
120: *         LPATH = LPATH + 1  
121: *         PATH(LPATH:LPATH) = ':'  
122: *     end if  
123: *     if ( PATH(1:1).eq.':' ) then  
124: C/ENDIF  
125: C  
126: C ... get absolute path of index file again ...  
127: C     call GTPATH( IDXD, FILE, LPP )  
128: C  
129:     PATH = FILE(:LPP) //PATH(:LPATH)  
130:     LPATH = LPP + LPATH
```

src/mvp/xsecd.f

```

131: C
132:       end if
133:       end if
134: C
135: C ... get path name of index file ...
136: C
137:       if ( PATH.eq.' ' ) then
138:         call GTPATH( INDX, PATH, LPATH )
139:       end if
140: C
141: C/#ENDIF
142: C
143: C-----
144:       do 190 N = 1, NSTAL
145:         TAG = ' '
146:         write(TAG,'(I5,'''')' ) N
147:         call CCOMP( TAG, LEN(TAG), TAG, IIF )
148:         ITAG = ICLEN(TAG)
149:         THEAD = TAG(:ITAG) //'&ICRES'
150:         call FCTLB( ICRES, '&ICRES', THEAD(:ICLEN2(THEAD)), IRET )
151:         call UNPKCS( ICRES, '&ID', ID, NLEN, IRET )
152:         call UNPKCS( ICRES, '&MT', MT, NL, IRET )
153: C
154:         write(IPR,7020) N, ID(:NLEN), MT(:NL)
155:       7020 format(2X,' === RESPONSE #',I3,' : ',A,2X,A)
156:         if ( MT(:2).eq.'MT' ) then
157:           Cccc read(MT(3:NL),* ) / MTT
158:           call READI4( MT, 3, NL, MTT )
159:         else
160:           MTT = 0
161:         end if
162: C
163: C/#IF PARA( PVM MPI)
164: Cc       if ( ITASK0.lt.0 ) then
165: C*       if ( IDTASK.eq.1 ) then
166: C/#ENDIF
167:         IRWND = 0
168:       100 LINE = ' '
169:         read(IDXD,fmt ='(A)',end =180) LINE
170:         IS = 1
171:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
172:         if ( LINE(IS:IE).ne.ID(:NLEN) ) go to 100
173: C
174: C .... skip atomic weight ratio
175: C
176:         IS = IE + 1
177:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
178: C
179: C .... get FILE name
180: C
181:         IS = IE + 1
182:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
183: C
184:         if ( LPATH.gt.0 ) then
185: C/#IF UNIX
186: C
187: C ... filename with relative directory path ..
188: C
189:         if ( LINE(IS:IS).ne.'/' ) then
190:           FILE = PATH(1:LPATH) //LINE(IS:IE)
191:         else
192:           FILE = LINE(IS:IE)
193:         end if
194: C
195: C/#ELSE

```

```

196: C*       FILE = PATH(1:LPATH) //LINE(IS:IE)
197: C/#ENDIF
198:       else
199:         FILE = LINE(IS:IE)
200:       end if
201: C
202: C .... get other parameters
203: C
204:         IS = IE + 1
205:       Ccccc read( LINE(IS:80), * ) I1,i2,i3,i4
206:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
207:       c##<2007/03/14:PN3:
208:       c## call READI4( LINE, IS, IE, I1 )
209:         call READI4( LINE, IS, IE, I1 ) ! route
210:       c##>
211: C
212:         IS = IE + 1
213:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
214:       c##<2007/03/14:PN3:
215:       c## call READI4( LINE, IS, IE, I2 )
216:         call READI4( LINE, IS, IE, I2 ) ! file type
217:       c##>
218: C
219:         IS = IE + 1
220:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
221:       c##<2007/03/14:PN3:
222:       c## call READI4( LINE, IS, IE, I3 )
223:         call READI4( LINE, IS, IE, I3 ) ! address
224:       c##>
225: C
226:         IS = IE + 1
227:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
228:       c##<2007/03/14:PN3:
229:       c## call READI4( LINE, IS, IE, I4 )
230:         call READI4( LINE, IS, IE, I4 ) ! table length
231:       c##>
232: C
233: C .... read response
234: C
235:         inquire(IXD,opened =OPEND)
236:         if ( OPEND ) close( IXD )
237: C
238:         IL = ICLEN(FILE)
239:         call OPENF( IXD, FILE(:IL), 'FORMATTED', 'OLD', 'READ', JERR )
240: C
241:         if ( JERR.ne.0 ) then
242:           write(IMSG,*) 'XXX(XSECD) Failed to open file : ', FILE(:IL)
243:           call CNTERR( 'FATAL' )
244:           go to 190
245:         end if
246: C
247:         do 110 I = 1, I3 - 1
248:           read(IXD,'(A80)') LINE
249:           110 continue
250: C
251:         read(IXD,'(A80)') LINE
252:         NL = ICLEN2(LINE)
253:         write(IPR,'(5X,A)') LINE(:NL)
254:         read(IXD,'(A80)') LINE
255:         NL = ICLEN2(LINE)
256:         write(IPR,'(5X,A)') LINE(:NL)
257:         do 120 I = 3, 6
258:           read(IXD,'(A80)') LINE
259:           120 continue
260:         read(IXD,'(8I9)') (NXS(I),I=1,16)

```

src/mvp/xsecd.f

```

261:      read(IXD,'(8I9)') (JXS(I),I=1,32)
262: C
263:      ISS      = 0
264:      IE       = 0
265:      NNN      = 4
266:      NW       = 20
267: C
268:      NLIM     = 1000
269:      NR       = 0
270:      NR0      = 0
271:      LMTR     = JXS(3)
272:      NMT      = NXS(4)
273: C
274: C 130
275:      call RDDOS( LINE, LMTR, NMT, NR, ISS, IS, IE, 'I4', IDUM, CRES,
276: &      NNN, NW, NLIM )
277:      do 140 I = 1, NR
278:          if ( IDUM(I).eq.MTT ) go to 150
279: C 140
280:          continue
281:          if ( NR.lt.NMT ) then
282:              NMT = NMT - NR
283:              LMTR = LMTR + NR
284:              NR = 0
285:              NR0 = NR0 + NR
286:              go to 130
287:          end if
288:          write(IMG,*) 'XXX(XSECD) Reaction ', MTT, ' cannot be found.'
289:          call CNTERR( 'FATAL' )
290:          go to 190
291:          continue
292: C 150
293:          IMT = NR0 + I
294: C
295:          write(6,*) ' MT=',(IDUM(I),I=1,NR0+NR), ' IMT=',IMT
296: C
297:          LSIG = JXS(6) + IMT - 1
298:          NR = 0
299:          N1 = 1
300:          call RDDOS( LINE, LSIG, N1, NR, ISS, IS, IE, 'I4', LOC1, CRES,
301: &      NNN, NW, N1 )
302: C
303:          write(6,*) ' LOC1=',LOC1
304: C
305:          LSIGD = JXS(7) + LOC1 - 1
306:          NR = 0
307:          N1 = 1
308:          call RDDOS( LINE, LSIGD, N1, NR, ISS, IS, IE, 'I4', NRRR, CRES,
309: &      NNN, NW, N1 )
310: C
311:          write(6,*) ' NRRR=',NRRR
312:          LSIGD = LSIGD + 1
313:          if ( NRRR.gt.0 ) then
314:              NR = 0
315:              N1 = NRRR*2
316:              call RDDOS( LINE, LSIGD, N1, NR, ISS, IS, IE, 'I4', IDUM,
317: &      CRES, NNN, NW, NLIM )
318:              LSIGD = LSIGD + 2*NRRR
319:              N1 = MIN(NRRR,NR-NRRR)
320:              do 160 I = 1, N1
321:                  if ( IDUM(NRRR+I).ne.2 ) then
322:                      write(IMG,*)
323: &      'XXX(XSECD) Unexpected interpolation code ( ',
324: &      IDUM(NRRR+I), ' ) is encountered. STOP'
325:                      stop 888
326:                  end if
327:              continue
328: C 160
329:          end if
330:          continue
331: C
332:          end if
333: C
334:          NR = 0
335:          N1 = 1
336:          call RDDOS( LINE, LSIGD, N1, NR, ISS, IS, IE, 'I4', NE, CRES,
337: &      NNN, NW, N1 )
338: C
339:          write(6,*) ' NE=',NE
340: C
341:          LSIGD = LSIGD + 1
342:          KCRES(N,1) = MCRES + 1
343: C
344:          write(6,*) ' KCRES(N,1)=',KCRES(N,1)
345:          if ( KCRES(N,1)+2*NE-1.gt.NLTEMP ) then
346:              KCRES(N,2) = 0
347:              KCRES(N,3) = 0
348:              go to 200
349:          else
350:              N1 = 2*NE
351:              NR = 0
352:              call RDDOS( LINE, LSIGD, N1, NR, ISS, IS, IE, 'R4', IDUM,
353: &      CRES(KCRES(N,1)), NNN, NW, N1 )
354:              MCRES = MCRES + 2*NE
355:              KCRES(N,2) = NE
356:              KCRES(N,3) = 1
357: C##<2007/03/14:PN3:
358: C##
359:              KCRES(N,3) = NXS(6)
360:              KCRES(N,3) = NXS(16) ! special flag in dosimetry library
361:          end if
362:          for mvp
363:              C##>
364:              if ( KCRES(N,3).eq.0 ) KCRES(N,3) = 1
365: C
366:          write(6,*) ' N,NR,NE,N1=',N,NR,NE,N1
367:          do 170 I = 1, NE
368:              CRES(KCRES(N,1)+I-1) = CRES(KCRES(N,1)+I-1)*1.E6
369:              continue
370:          end if
371:          write(6,*) ' EEEE '
372:          write(6,'(1p5E10.3)') (CRES(KCRES(N,1)+I-1),I=1,NE)
373:          write(6,*) ' XXXX '
374:          write(6,'(1p5E10.3)') (CRES(KCRES(N,1)+I-1),I=NE+1,2*NE)
375:          end if
376:          go to 190
377: C
378:          if ( IRWND.eq.0 ) then
379:              IRWND = 1
380:              rewind IDX
381:              go to 100
382:          else
383:              write(IMG,('(1x,a)'))
384:              &      'XXX(XSECD) No point-wise response function with this ID.'
385:              call CNTERR( 'FATAL' )
386:              end if
387: C
388:          C/#IF PARA( PVM MPI )
389:          *      end if
390:          C/#ENDIF
391:          C
392:          190 continue
393:          C
394:          inquire(IXD,opened =OPEND)
395:          if ( OPEND ) close( IXD )
396:          C
397:          C/#IF PARA( PVM MPI )

```

src/mvp/xsecd.f

```

390:
391:   if ( NTASK.gt.1 ) then
392:     call TOKEI( TX0, 0 )
393:     IT0      = 1
394:     call MVPCOM_BCAST_I4( IT0, 55, KCRES, NSTAL*3, INFO )
395:     call MVPCOM_BCAST_I4( IT0, 55, MCRES, 1, INFO )
396:     call MVPCOM_BCAST_R4( IT0, 55, CRES, MCRES, INFO )
397:     call TOKEI( TX1, 0 )
398:     write(IPR, '(lx,a,a,i9,a,lp,e12.5)')
399:     &      '=== Elapsed time to send point-wise response',
400:     &      ' to subtasks :', MCRES, ' words) : ', TX1 - TX0
401:   end if
402: C
403: C/#ENDIF
404:
405:   call RESIZE( 'CRES' , LCRES, MCRES, 'R4', LAST )
406: C
407: C
408:   return
409: C
410: C
411: C
412:   200 write(IMG, '(lx,a,i10,a)')
413:   &      'XXX MEMORY OVER IN XSECD !!! (LIMIT = ', LIMIT, ' STOP '
414:   call PRSTOP( 1, 'MEMORY OVER.' )
415:   stop 999
416:
417: end
418: C
419: C=====
420: C
421:   subroutine RDDOS( LINE, LMTR, NMT, NR, ISS, IS, IE,
422:   &      TYP, IDUM, R, NNN, NW, NLIM )
423:   include 'INC/_IOUNIT2'
424:   include '../shared/INC/_IOUNIT'
425:   character*(*) LINE, TYP
426:   integer IDUM(*)
427:   real R(*)
428: C
429:   100 if ( ISS.eq.0 ) then
430:     read(IXD, '(A80)', end =120) LINE
431:     IS      = IE + 1
432:     IE      = IE + NNN
433:     ISS     = 1
434:   end if
435:   if ( NR.eq.0.and.LMTR.lt.IS ) then
436:     write(IMG,*) ' XXX(RDDOS) SOMETHING WRONG IS DETECTED, STOP.'
437:     stop 888
438:   end if
439:   if ( LMTR.gt.IE ) then
440:     ISS      = 0
441:     go to 100
442:   end if
443:   if ( NR.eq.0 ) ISS = MOD(LMTR-1,NNN) + 1
444:   110 ISSS    = (ISS-1)*NW + 1
445:   NR         = NR + 1
446:   if ( TYP.eq.'I4' ) then
447: Cc      read( LINE(ISSS:ISSS+NW-1), * ) IDUM(NR)
448:       call READI4( LINE, ISSS, ISSS+NW-1, IDUM(NR) )
449:   else
450: Cc      read( LINE(ISSS:ISSS+NW-1), * ) R(NR)
451:       call READR4( LINE, ISSS, ISSS+NW-1, R(NR) )
452:   end if
453:   ISS        = ISS + 1
454:   if ( ISS.gt.NNN ) then
455:
456:     ISS      = 0
457:     if ( NR.lt.NMT.and.NR.lt.NLIM ) go to 100
458:   else
459:     if ( NR.lt.NMT.and.NR.lt.NLIM ) go to 110
460:   end if
461:   return
462: C
463:   120 write(IMG,*)
464:   &      'XXX(RDDOS) UNEXPECTED END OF FILE IS ENCOUNTERED, STOP.'
465:   stop 888
466: end

```

src/mvp/xsece.f

```

1:      subroutine XSECE( CXE,   MMCXE, LCXE,  MCXE,  LAST,  LIMIT, NEE,
2:      &                  MTOP,  KLBE1, XLBE1, EEL,   RKT,  PBR,  EBA,
3:      &                  PBT,   EBT,  RNG,   EBTP,  ETOPL,
4:      &                  EBOTLE, MLEN,  NPATOM, NMAT,  NATMT, MPATM, LPDNP,
5:      &                  IATMT, DNSTP, FME,   WWAG,  EEDG,  EEEK,  JDEBG
6:      &                  )
7: C
8: C=====
9: C  PURPOSE :  GROUPWISE ENERGY CROSS SECTION INPUT & PROCESSING
10: C            - ELECTRON LIBRARY -
11: C  CALLED IN: CTXSIN
12: C=====
13: C
14: C  CXE(*)          R4  ELECTRON CROSS SECTION DATA
15: C  MCXE           I4  LENGTH OF CXE ARRAY
16: C
17: C  < CROSS SECTION DATA FOR EACH ATOMIC ELEMENT >
18: C
19: C  KLBE1(NPATOM,20)  I4  RECORD #1 (SPECIFICATION RECORD)
20: C
21: C      1:  NTDATA  LENGTH OF RECORD #2
22: C      2:  NRAD   LENGTH OF TR AND RR TABLES (USUALLY 49)
23: C      3:  NMOT   LENGTH OF ER AND RS TABLES (USUALLY 18)
24: C      4:  NCOR   LENGTH OF TC AND CP TABLES (USUALLY 32)
25: C      5:  NHFL   LENGTH OF TH AND FL TABLES (USUALLY 39)
26: C      6:  NEE0   NUMBER OF ENERGY GRIDS FOR ELECTRON CROSS SECTION
27: C                DATA (USUALLY 134)
28: C      7:  MTOPO  NUMBER OF BREMSSTRAHLUNG PHOTON/ELECTRON (K/T)
29: C                RATIOS (USUALLY 49)
30: C      8:  NEK    NUMBER OF ENERGY GRIDS FOR EK TABLE (USUALLY 9)
31: C      9:  NPL    NUMBER OF THE LEGENDRE SERIES FOR GK TABLE
32: C                (USUALLY 240)
33: C     10:  LSTC   STARTING POSITION OF DATA FOR AN ATOMIC ELEMENT
34: C     11:  LSTTR  STARTING POSITION OF TR TABLE (LSTC+4)
35: C     12:  LSTER  STARTING POSITION OF ER TABLE (LSTTR+NRAD*2)
36: C     13:  LSTEK  STARTING POSITION OF EK TABLE (LSTER+NMOT*6)
37: C     14:  LSTTC  STARTING POSITION OF TC TABLE (LSTER+NEK*14)
38: C     15:  LSTTH  STARTING POSITION OF TH TABLE (LSTTC+NCOR*2)
39: C     16:  LSTEEL STARTING POSITION OF ENERGY GRIDS FOR ELECTRON
40: C                CROSS SECTION DATA, IN EV (LSTTH+NHFL*2)
41: C     17:  LSTPBR STARTING POSITION OF BREMSSTRAHLUNG CROSS SECTIONS
42: C                (LSTEEL+NEE)
43: C     18:  LSTRKT STARTING POSITION OF K/T RATIOS (LSTPBR+NEE)
44: C     19:  LSTEB  STARTING POSITION OF BREMSSTRAHLUNG CUMULATIVE
45: C                PROBABILITIES, EBA (LSTRKT+MTOP)
46: C     20:  LSTGK  STARTING POSITION OF GK TABLE (LSTEB+NEE*MTOP)
47: C
48: C  XLBE1(NPATOM,6)  R4  ATOMIC PARAMETERS
49: C
50: C      1:  AWR    ATOMIC WEIGHT
51: C      2:  Z      ATOMIC NUMBER
52: C      3:  Z13    Z**(1/3)
53: C      4:  ZLN    LOG(Z)
54: C      5:  CCL    BREMSSTRAHLUNG COULOMB CORRECTION FACTOR
55: C      6:  TZ     PROCESSING TEMPERATURE OF CROSS SECTIONS, IN EV
56: C
57: C  ETOPL           R4  UPPER ENERGY LIMIT IN ELECTRON LIBRARY
58: C  EBOTLE          R4  LOWER ENERGY LIMIT IN ELECTRON LIBRARY
59: C
60: C  NPATOM          I4  NUMBER OF ATOMIC ELEMENTS USED IN PROBLEM
61: C  NMAT            I4  NUMBER OF MATERIALS USED IN PROBLEM
62: C  MLEN            I4  LENGTH OF ATOMIC NUMBER TABLE FOR ALL MATERIALS
63: C
64: C  NEE            I4  NUMBER OF ENERGY GRIDS FOR ELECTRON CROSS SECTION
65: C                DATA (USUALLY 134)

```

```

66: C  MTOP           I4  NUMBER OF BREMSSTRAHLUNG PHOTON/ELECTRON (K/T)
67: C                RATIOS FOR RKT TABLE (USUALLY 49)
68: C
69: C  MPATM(NMAT)     I4  NUMBER OF ATOM INCLUDED IN EACH MATERIAL
70: C  LPDNP(NMAT+1)   I4  POSITION POINTER FOR MATERIAL TABLES
71: C  NATMT(NPATOM)   I4  ATOMIC NUMBER TABLE USED IN PROBLEM
72: C  IATMT(MLEN)     I4  ATOMIC NUMBER TABLE BY MATERIALS
73: C  DNSTP(MLEN)     R4  ATOM NUMBER DENSITY TABLE BY MATERIALS
74: C  FME(MLEN)       R4  ATOM FRACTION TABLE BY NUMBER DENSITY BY MATERIAL
75: C  WWAG(NMAT)      R4  GENERATING PROBABILITIES OF AUGER ELECTRON BY
76: C                HIGHEST Z IN MATERIALS
77: C  EEDG(NAMT)      R4  K-EDGE ENERGIES OF HIGHEST Z IN MATERIALS
78: C  EEEK(NAMT)      R4  X-RAY ENERGIES OF HIGHEST Z IN MATERIALS
79: C
80: C=====
81: C
82: C      implicit real(A-H,O-Z)
83: C
84: C      integer NATMT(NPATOM), MPATM(NMAT), IATMT(MLEN), LPDNP(NMAT+1)
85: C      real      DNSTP(MLEN), FME(MLEN), WWAG(NMAT), EEDG(NMAT), EEEK(NMAT)
86: C
87: C      integer JDEBG(*)
88: C
89: C      integer KLBE1(NPATOM,20)
90: C      real      XLBE1(NPATOM,6)
91: C
92: C      real      CXE(MMCXE)
93: C      integer  IDUM(7)
94: C
95: C      integer  IEPTP(2*(MTOP-1),NEE,NMAT)
96: C      real      EEL(NEE), RKT(MTOP), PBR(NEE,NMAT), PBT(NEE,NMAT),
97: C      &          EBA(MTOP,NEE,NMAT), EBT(MTOP,NEE,NMAT), RNG(NEE,NMAT),
98: C      &          EBTP(MTOP-1,NEE,NMAT)
99: C
100: C      real TEMP(48)
101: C
102: C      character MATT*8, PDATE*8, PCOMMT*120
103: C
104: C      character CHSYMB*2
105: C      external CHSYMB
106: C
107: C      character FILE*256
108: C
109: C      include '../shared/INC/_IOUNIT'
110: C      include 'INC/_IOUNIT2'
111: C
112: C      .... I/O UNIT NUMBER OF ELECTRON LIBRARY INDEX FILE ....
113: C      --> SEE SUBROUTINE FALOC.
114: C
115: C-----
116: C  SET INITIAL DATA
117: C-----
118: C
119: C      ETOPL = 1.0E+30
120: C      EBOTLE = -1.0E+30
121: C      LST = 0
122: C      LST0 = 0
123: C      NERR = 0
124: C      J999 = 0
125: C
126: C      call LABEL( IPR, 'ELECTRON LIBRARY INPUT' )
127: C
128: C      write(IPR, '()')
129: C
130: C

```


src/mvp/xsece.f

```

131:      do 110 N = 1, NPATOM
132:        NZ      = NATMT(N)
133: C
134:        write(IPR, '(1X, '' == ATOM '', I3, '' : <'', A,
135: &                ''>      ATOMIC NUMBER = '', I3 ) '
136: &                ) N, CHSYMB(NZ), NZ
137: C
138: C-----
139: C  OPEN LIBRARY FILE FOR ELECTRON
140: C-----
141: C
142: 100 call FALOCE( IXE, CHSYMB(NZ), FILE, IERR )
143: C
144:      if ( IERR.ne.0 ) then
145:        NERR = NERR + 1
146:        write(IMG,*) 'XXX CANNOT FIND DATA FOR <', CHSYMB(NZ),
147: &                '>' IN SUBROUTINE XSECE.'
148:        call CNTERR( 'FATAL' )
149:        go to 110
150:      end if
151: C
152:      LFN      = ICLEN(FILE)
153:      write(IPR,*) '      <', CHSYMB(NZ), '>      FILE = ', FILE(1:LFN)
154: C
155: C-----
156: C  READ SPECIFICATION RECORD #1
157: C-----
158: C
159:      read(IXE) MATT, PDATE, PCOMMT, NTDATA, NRAD, NMOT, NCOR, NHFL,
160: &      NEE0, MTOP0, NEK, NPL, (IDUM(I), I=1, 2), LSTC, LSTTR,
161: &      LSTER, LSTEK, LSTTC, LSTTH, LSTEEL, LSTPBR, LSTRKT,
162: &      LSTEB, LSTGK, (IDUM(I), I=1, 5), AWR, Z, Z13, ZLN, CCL,
163: &      TZ, ELOWE, EHIGHE
164: C
165: C      .... LIBRARY SPECIFICATION ERROR ? ATOMIC NUMBER MISMATCH ....
166: C
167:      if ( NZ.ne.INT(Z) ) then
168:        NERR = NERR + 1
169:        call CNTERR( 'FATAL' )
170:        write(IMG,*) ' XXX ATOMIC NUMBER MISMATCH <', CHSYMB(NZ),
171: &                '>'
172:        write(IMG,*) ' XXX FILE = ', FILE
173:        write(IMG,*) ' XXX MATT = ', MATT
174:        write(IMG,*) ' XXX DATE = ', PDATE
175:        write(IMG,*) ' XXX COMMENT: ', PCOMMT
176:        write(IMG,*) ' XXX NZ = ', NZ, '      Z IN FILE = ', Z
177:        write(IMG,*) ' '
178:        go to 110
179:      end if
180: C
181:      if ( NEE.ne.NEE0 .or. MTOP.ne.MTOP0 ) then
182:        NERR = NERR + 1
183:        call CNTERR( 'FATAL' )
184:        write(IMG,*) ' XXX ENERGY GRIDS OR K/T RATIOS MISMATCH ',
185: &                '<', CHSYMB(NZ), '>'
186:        write(IMG,*) ' XXX FILE = ', FILE
187:        write(IMG,*) ' XXX MATT = ', MATT
188:        write(IMG,*) ' XXX GRIDS = ', NEE, ' VS. ', NEE0
189:        write(IMG,*) ' XXX RATIOS = ', MTOP, ' VS. ', MTOP0
190:        write(IMG,*) ' '
191:        go to 110
192:      end if
193: C
194: C      .... STORE NECESSARY DATA TO KLBE1 AND XLBE1 ARRAY ....
195: C

```

```

196:      KLBE1(N,1) = NTDATA
197:      KLBE1(N,2) = NRAD
198:      KLBE1(N,3) = NMOT
199:      KLBE1(N,4) = NCOR
200:      KLBE1(N,5) = NHFL
201:      KLBE1(N,6) = NEE
202:      KLBE1(N,7) = MTOP
203:      KLBE1(N,8) = NEK
204:      KLBE1(N,9) = NPL
205:      KLBE1(N,10) = LST + LSTC
206:      KLBE1(N,11) = LST + LSTTR
207:      KLBE1(N,12) = LST + LSTER
208:      KLBE1(N,13) = LST + LSTEK
209:      KLBE1(N,14) = LST + LSTTC
210:      KLBE1(N,15) = LST + LSTTH
211: C      KLBE1(N,16) = LST + LSTEEL
212:      KLBE1(N,16) = 0
213: C      KLBE1(N,17) = LST + LSTPBR
214:      KLBE1(N,17) = LST + LSTPBR - NEE
215: C      KLBE1(N,18) = LST + LSTRKT
216:      KLBE1(N,18) = 0
217: C      KLBE1(N,19) = LST + LSTEB
218:      KLBE1(N,19) = LST + LSTEB - NEE - MTOP
219:      KLBE1(N,20) = LST + LSTGK - NEE - MTOP
220: C
221:      XLBE1(N,1) = AWR
222:      XLBE1(N,2) = Z
223:      XLBE1(N,3) = Z13
224:      XLBE1(N,4) = ZLN
225:      XLBE1(N,5) = CCL
226:      XLBE1(N,6) = TZ
227: C
228:      ETOPLE = EHIGHE
229:      EBOTLE = ELOWE
230: C
231: C-----
232: C  READ RECORD #2 (CROSS SECTION DATA)
233: C-----
234: C
235:      NTDATA = NTDATA - NEE - MTOP
236:      LST0 = LST + NTDATA
237: C
238:      if ( LST0.gt.MMCXE ) then
239:        call MEMERR( 'XSECE '
240: &                'IN KEEPING MEMORY FOR RECORD #2 DATA FOR ELECTRON',
241: &                LSIZ(LCXE+LST0), LIMIT )
242:        J999 = 1
243:      end if
244: C
245: C      .... IN READING, EEL AND RKT TABLES IN ELECTRON CROSS SECTION
246: C      DATA WAS ELIMINATED ....
247: C
248: C
249: C      if ( N.eq.1 ) then
250: C
251: C        read(IXE) (CXE(I), I=LST+1, KLBE1(N,17)-1), (EEL(I), I=1, NEE),
252: C &                (CXE(I), I=KLBE1(N,17), KLBE1(N,19)-1),
253: C &                (RKT(I), I=1, MTOP), (CXE(I), I=KLBE1(N,19), LST+NTDATA)
254: C      else
255: C
256: C        read(IXE) (CXE(I), I=LST+1, KLBE1(N,17)-1), (RDUM, I=1, NEE),
257: C &                (CXE(I), I=KLBE1(N,17), KLBE1(N,19)-1),
258: C &                (RDUM, I=1, MTOP), (CXE(I), I=KLBE1(N,19), LST+NTDATA)
259: C
260: C      end if

```

src/mvp/xsece.f

```

261: C
262:       write(IPR,7010) NZ, LST + 1, LST0
263: 7010 format('          DATA FOR ATOM (Z=',I2,',) : FROM CXE('',I12,
264:       &      ' ) TO CXE('',I12,',) '//)
265:       KLBE1(N,1) = NTDATA
266:       LST       = LST0
267: C
268:       close( IXE )
269: C
270: 110 continue
271: C-----
272: C Memory Over
273: C-----
274:       if ( 9999.ne.0 ) then
275:         call PRSTOP( 0, 'MEMORY OVER IN ELECTRON CROSS SECTION INPUT.'
276:         &      )
277:         stop 999
278:       end if
279: C
280: C-----
281: C SET LENGTH OF CXE ARRAY (MCXE) AND LAST POSITION POINTER (LAST)
282: C-----
283: C
284:       MCXE = LST0
285:       call RESIZE( 'CXE', LCXE, MCXE, 'R4', LAST )
286: C
287: C
288: C .... PROVIDE ATOM FRACTION TABLE, FME ....
289: C
290:       do 140 M = 1, NMAT
291:         T = 0.0
292:         LP = LPDNP(M) - 1
293: C
294:         do 120 N = 1, MPATM(M)
295:           T = T + DNSTP(LP+N)
296: 120       continue
297:         do 130 N = 1, MPATM(M)
298:           FME(LP+N) = DNSTP(LP+N) /T
299: 130       continue
300: C
301: 140 continue
302: C
303: C .... PROVIDE AUGER PROBABILITY TABLE, WWAG,
304: C          K-EDGE ENERGY TABLE, EEDG, AND
305: C          X-RAY ENERGY TABLE, EEEK, FOR HIGHEST Z ....
306: C
307:       do 160 M = 1, NMAT
308:         IZ = 0
309:         IA = 1
310:         LP = LPDNP(M) - 1
311:         do 150 N = 1, MPATM(M)
312:           IAT = IATMT(LP+N)
313:           if ( IZ.lt.NATMT(IAT) ) then
314:             IZ = NATMT(IAT)
315:             IA = IAT
316:           end if
317: 150       continue
318: C
319:       WWAG(M) = 1.0/(1.0+1.0/(-0.064+0.034*IZ-1.03D-6*IZ**3)**4)
320:       EEDG(M) = CXE(KLBE1(IA,10))
321:       EEEK(M) = CXE(KLBE1(IA,10)+1)
322: C
323: 160 continue
324: C
325: C-----

```

```

326: C CALCULATE MATERIAL-WISE BREMSSTRAHLUNG CROSS SECTION, PBR
327: C-----
328: C
329:       do 200 M = 1, NMAT
330:         LP = LPDNP(M) - 1
331:         do 170 I = 1, NEE
332:           PBR(I,M) = 0.0
333: 170       continue
334: C
335:         do 190 N = 1, MPATM(M)
336:           IAT = IATMT(LP+N)
337:           DEN = FME(LP+N)
338: C
339:           do 180 I = 1, NEE
340:             PBR(I,M) = PBR(I,M) + CXE(KLBE1(IAT,17)+I-1)*DEN
341: 180           continue
342: C
343: 190       continue
344: C
345: 200 continue
346: C
347: C-----
348: C CALCULATE MATERIAL-WISE NORMALIZED CUMULATIVE PROBABILITIES, EBA
349: C-----
350: C
351:       do 280 M = 1, NMAT
352:         LP = LPDNP(M) - 1
353:         do 220 I = 1, NEE
354:           do 210 K = 1, MTOP
355:             EBA(K,I,M) = 0.0
356: 210           continue
357: 220         continue
358: C
359:         do 250 N = 1, MPATM(M)
360:           IA = IATMT(LP+N)
361:           DEN = FME(LP+N)
362: C
363:           do 240 I = 1, NEE
364:             NN = MTOP*(I-1)
365:             DPB = CXE(KLBE1(IA,17)+I-1)*DEN
366:             do 230 K = 2, MTOP
367:               EBA(K,I,M) = EBA(K,I,M) + CXE(KLBE1(IA,19)+NN+K-1)*
368:               &      DPB
369: 230             continue
370: 240           continue
371: C
372: 250         continue
373: C
374: C .... NORMALIZE THE CUMULATIVE PROBABILITY, EBA ....
375: C
376:           do 270 I = 1, NEE
377:             do 260 K = 2, MTOP
378:               EBA(K,I,M) = EBA(K,I,M) /EBA(MTOP,I,M)
379: 260             continue
380: 270           continue
381: C
382: 280 continue
383: C
384: C-----
385: C CALCULATE BREMSSTRAHLUNG YIELDS, PBT, AND ENERGY DISTRIBUTION, EBT,
386: C FOR THE THICK-TARGET BREMSSTRAHLUNG APPROXIMATION
387: C-----
388: C
389: C .... RANGE OF ELECTRON IN MATERIAL ....
390: C

```

src/mvp/xsece.f

```

391:      call RANGE( CXE, MMCXE, KLBE1, XLBE1, NEE, EEL, RNG, NPATOM,
392:      &          NMAT, MLEN, MPATM, LPDNP, IATMT, NATMT, DNSTP, FME )
393: C
394: C      .... CALCULATE BREMSSTRAHLUNG YIELDS ....
395: C
396:      do 340 M = 1, NMAT
397:          PBT(NEE,M) = 0.0
398: C
399:          do 320 NZ = 1, NEE - 1
400:              N = NEE - NZ
401:              PBT(N,M) = PBT(N+1,M) + 0.5*(PBR(N,M)+PBR(N+1,M))*
402:              &          (RNG(N,M)-RNG(N+1,M))
403: C
404:          do 300 K = 2, MTOP
405:              E = RKT(K)*EEL(N)
406:              do 290 J = N, NEE - 1
407:                  if ( E.lt.EEL(J) ) then
408:                      Q = QPOL(E/EEL(J),RKT,EBA(1,J,M),MTOP)
409:                  else
410:                      Q = 1.0
411:                  end if
412: C
413:                  EBT(K,N,M) = EBT(K,N,M) + Q*0.5*(PBR(J,M)+PBR(J+1,M))
414:              &          *(RNG(J,M)-RNG(J+1,M))
415:          290      continue
416:          300      continue
417: C
418: C      .... NORMALIZE THE TTB CUMULATIVE PROBABILITY, EBT ....
419: C
420:          do 310 K = 2, MTOP
421:              EBT(K,N,M) = EBT(K,N,M) /EBT(MTOP,N,M)
422:          310      continue
423: C
424:          320      continue
425: C
426:          do 330 K = 1, MTOP
427:              EBT(K,NEE,M) = EBT(K,NEE-1,M)
428:          330      continue
429: C
430:          340      continue
431: C
432: C-----
433: C      PROVIDE THE SAMPLING TABLE FOR THE CONDITINAL DISCRETE SAMPLING
434: C-----
435: C
436: C      .... PARTIAL PROBABILITIES OF TTB SPECTRA ....
437: C
438:          do 370 M = 1, NMAT
439:              do 360 N = 1, NEE
440:                  do 350 K = 1, MTOP - 1
441: C
442:                      EBTP(K,N,M) = EBT(K+1,N,M) - EBT(K,N,M)
443: C
444:          350      continue
445:          360      continue
446:          370      continue
447: C
448: C      .... PROVIDE THE SAMPLING TABLE ....
449: C
450:          do 390 M = 1, NMAT
451:              do 380 N = 1, NEE
452: C
453:                  call BMCMK1( EBTP(1,N,M), TEMP, IEBTP(1,N,M), MTOP-1 )
454:                  call ATRANS( TEMP, EBTP(1,N,M), MTOP-1 )
455: C

```

```

456:      380      continue
457:      390      continue
458: C
459: C-----
460: C      PRINT OUT OF INFORMATION FOR ELECTRON CROSS SECTION
461: C-----
462: C
463:      call LABEL( IPR, 'ATOM SPECIFICATION DATA FOR ELECTRON' )
464: C
465:      write(IPR,7020)
466:      7020      format(
467:      &          ' NO.   NAME      Z      LENGTH      ENERGY   (K/T)      LSTC      LSTPBR',
468:      &          '          LSTGBA      LSTGK      COULOMB      TEMP.'//
469:      &          '          OF DATA      GRIDS      RATIOS      ',
470:      &          '          CORREC.      (EV)'//
471:      &          ' --   ----   --   ----   ----   ----   ----   ----',
472:      &          '          ----   ----   ----   ----   ----   ----')
473:      do 400 J = 1, NPATOM
474:          write(IPR,7030) J, CHSYMB(NATMT(J)), INT(XLBE1(J,2)),
475:          &          KLBE1(J,1), KLBE1(J,6), KLBE1(J,7), KLBE1(J,10),
476:          &          KLBE1(J,17), KLBE1(J,19), KLBE1(J,20), XLBE1(J,5),
477:          &          XLBE1(J,6)
478:      7030      format(' ',I2,4X,A2,I6,I10,I9,I8,4I9,1PE12.5,E10.3)
479:      400      continue
480: C
481: C
482:      if ( JDEBG(1).ne.0 ) then
483: C
484:          write(IPR,7040) NMAT
485:          7040      format(/
486:          &          ' ELECTRON TTB AND RANGE TABLE FOR MATERIAL (NUMBER =',I3,')'//
487:          &          ' -----')
488:          do 410 M = 1, NMAT
489:              write(IPR,7050) M, MPATM(M)
490:          7050      format(/' NO.',I2,' MATERIAL (NUMBER OF ELEMENTS =',I3,')'//
491:          &          '          NO.      Z      NUMBER DENSITY      FRACTION')
492:          LP = LPDNP(M) - 1
493:          write(IPR,7060)
494:          &          (J,NATMT(IATMT(LP
495:          &          +J)),DNSTP(LP+J),FME(LP+J),J=1,MPATM(M))
496:          7060      format(7X,I2,I7,1PE17.5,0PF14.5)
497:          write(IPR,7070)
498:          7070      format(/
499:          &          '          NO.      ENERGY(EV)      RANGE(G/CM**2)      PBR',
500:          &          '          PBT')
501:          write(IPR,7080)
502:          &          (N,EEL(N),RNG(N,M),PBR(N,M),PBT(N,M),N=1,NEE)
503:          7080      format(' ',I5,1P4E16.7)
504:          410      continue
505: C
506:      end if
507: C
508: C      .... ELECTRON LIBRARY INPUT ERROR ....
509: C
510:      if ( NERR.gt.0 ) then
511:          write(IMG,'(/lx,a/)')
512:          &          'XXX(XSECE) ELECTRON LIBRARY INPUT ERROR !!'
513:          stop 888
514:      end if
515: C
516:      if ( JDEBG(1).ne.0 ) then
517:          write(IPR,*) ' '
518:          write(IPR,*) ' !!! DEBUG PRINT FOR ELECTRON LIBRARY'
519:          write(IPR,*) ' '
520:          write(IPR,*) ' '

```

src/mvp/xsece.f

```
521:      end if
522: C
523:      return
524:      end
```

SAFE

src/mvp/xsec.f

```

1:      subroutine XSEC( CX,      ICX,      MMCX,      LCX,      MCX,      LAST,      LIMIT,
2:      &                NMT,      MATT,      KLIB1,      XLIB1,      KLIB2,      XLIB2,      ETOPL,
3:      &                EBOTL,      ESABL,      NUC,      NUCID,      NCIDI,      TEMPN,      JVPOOL,
4:      &                JXPOOL,      ATPOOL,
5:      &                JDEBG )
6: C=<MVP>=====
7: C  PURPOSE : Continuous energy cross section input & processing
8: C            (neutron library)
9: C  CALLED IN: CTXSIN
10: C  CALLS :   faloc cnterr label
11: C=====
12:      implicit real(A-H,O-Z)
13:      real CX(MMCX)
14:      integer ICX(MMCX), IDUM(100)
15: C
16:      character MATT(NUC)*16, NUCID(NUC)*16, NCIDI(NUC)*16
17:      real*8      TEMPN(NUC)
18:      character*(*) ATPOOL
19: C
20:      integer JDEBG(*)
21: C
22:      integer KLIB1(NUC,31), KLIB2(NUC,NMT,21)
23:      real XLIB1(NUC,11), XLIB2(NUC,NMT,4)
24: C
25:      character FILE*256
26: CC      logical OPEND
27: C
28:      include '../shared/INC/_IOUNIT'
29:      include 'INC/_IOUNIT2'
30: C
31: C
32:      ETOPL = 1.0E30
33:      EBOTL = -1.0E30
34:      ESABL = -1.0E30
35:      LST = 1
36:      LST0 = 0
37:      NERR = 0
38:      J999 = 0
39: C      WRITE(IPR,'(//)')
40: C
41:      do 170 N = 1, NUC
42:          MATT(N) = ' '
43: C##<2007/03/14:PN3:
44: C      ... check the duplicated neutron nuclide ID for photonuclear
45:      if ( N.gt.1 ) then
46:          do I = 1, N-1
47:              if ( NUCID(N).eq.NUCID(I) ) then
48:                  do K = 1, 31
49:                      KLIB1(N,K) = KLIB1(I,K)
50:                  end do
51:                  do K = 1, 11
52:                      XLIB1(N,K) = XLIB1(I,K)
53:                  end do
54:                  do K = 1, 21
55:                      do J = 1, NMT
56:                          KLIB2(N,J,K) = KLIB2(I,J,K)
57:                      end do
58:                  end do
59:                  do K = 1, 4
60:                      do J = 1, NMT
61:                          XLIB2(N,J,K) = XLIB2(I,J,K)
62:                      end do
63:                  end do
64:                  MATT(N) = MATT(I)
65:                  go to 170

```

```

66:      end if
67:      end do
68:      end if
69: C
70: C##>
71: C-----
72: C      OPEN LIBRARY FILE
73: C-----
74: C
75: C      ... free temperature library (needs doppler broadning)
76: C
77:      JPOOL = 0
78:      IERR = 0
79:      if ( TEMPN(N).gt.0.0D0 ) then
80: C check %%%
81: C      write(ipr,*) '%%% ixsf ncidi ',ixsf, ncidi(n),' nucid ',nucid(n)
82: C %%%%%%%%%
83: C
84: C      ... check existence of processed xsec by ART in directory ATPOOL.
85: C      If processed one is found under directory ATPOOL,
86: C      and the file is opened on I/O unit IXS and returns JPOOL = 1
87: C
88:      if( JXPOOL.ne.0 .and. ATPOOL.ne.' ' ) then
89:          call CKPOOL( ATPOOL, NUCID(N), FILE, IXS, IPR, JPOOL )
90:      end if
91: C
92:      if ( JPOOL.eq.0 ) then
93:          call FALOCF( IXSF, NCIDI(N), FILE, IERR )
94:      end if
95: C
96: C      ... fixed temperature library
97: C      else
98: C      call FALOC( IXS, NUCID(N), FILE, IERR )
99: C      end if
100: C
101: C      if ( IERR.ne.0 ) then
102: C          NERR = NERR + 1
103: C          call CNTERR( 'FATAL' )
104: C          go to 170
105: C      end if
106: C
107:      LFN = ICLEN(FILE)
108:      write(IPR,'(1X,A,A)') ' === NUCLIDE : ',
109:      &                NUCID(N):(ICLEN(NUCID(N)))
110:      write(IPR,'(1X,A,A,A)') '      FILE = <', FILE(:LFN), '>'
111: C
112: C-----
113: C      Doppler broadning (ART) module.
114: C
115: C      Read data from I/O unit IXSF and make doppler broadning.
116: C      Record 1 & 2 are output on unit IXS, and
117: C      record 3 are stored on memory staring at CX(LST).
118: C-----
119: C      if ( TEMPN(N).gt.0.0d0 .and. JPOOL.eq.0 ) then
120: C
121: C      ... unit IXS is opened in ARTLIB and it may be a scratch file
122: C      or named file depending on status of pooled file ...
123: C
124: C      inquire(IXS,opened =OPEND)
125: C      if ( OPEND ) close( IXS )
126: C      call OPENF( IXS, ' ', 'UNFORMATTED', 'SCRATCH', 'WRITE', IERR )
127: C
128:      KLIMCX = MMCX - LST + 1
129:      TTEMP = real(TEMPN(N))
130: C

```

src/mvp/xsec.f

```

131:      call ARTLIB( TTEMP, NCIDI(N), NUCID(N),
132:      &           IXSF, IXS, NTDATA, CX, LST,
133:      &           KLIMCX, JVPOP, IPR, IXLOCK, JXPOOL, ATPool,
134:      &           JDEBG, IERR )
135:      end if
136: C-----
137: C READ SPECIFICATION RECORD
138: C-----
139: C
140: C
141:      read(IXS) MATT(N)(1:25)
142:      call RWIND( IXS )
143: C
144:      if ( MATT(N)(17:25).eq.'VERSION 3' ) then
145:      read(IXS) MATT(N) (1:136),
146:      &      NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA, NBINE, NNU,
147:      &      NMT0, NNK0, IGFLG, LFI, LNU, ITHERM, ISTU, IENDU,
148:      &      LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF, LSNUD,
149:      &      NCAP, NCAPG, LF6, LSTF6, LLSSF, LSTDOP, LASYM,
150:      &      NOTDOP, (IDUM(K),K=33,100),
151:      &      (XLIB1(N,J),J=1,6),XLIB1(N,10)
152: C
153:      write(IPR,7000)
154:      &      MATT(N) (1:16), MATT(N) (129:136), MATT(N) (17:76),
155:      &      MATT(N) (77:128)
156:      7000 format(/6X,'ID: <' ,A,'>' ,8X,'DATE: ' ,A
157:      &      /6X,'COMMENT: <' ,A,'>' )
158: c##<2007/03/14:PN3:
159: c## &      /6X,'<' ,A,'>' )
160:      &      /6X,'<' ,A,'>' )
161: c##>
162: C7000 format(/6X,'ID: <' ,A8,'>' ,8X,'DATE: ' ,A8
163: C &      /6X,'COMMENT: <' ,A60,'>' )
164: C &      /6X,'<' ,A60,'>' )
165:      else
166:      read(IXS) MATT(N) (1:8), MATT(N) (9:16), MATT(N) (17:136),
167:      &      NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA, NBINE, NNU,
168:      &      NMT0, NNK0, IGFLG, LFI, LNU, ITHERM, ISTU, IENDU,
169:      &      LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF, LSNUD,
170:      &      NCAP, NCAPG, LF6, LSTF6, LLSSF, LSTDOP, LASYM,
171:      &      (IDUM(K),K=8,10),
172:      &      (XLIB1(N,J),J=1,6),XLIB1(N,10)
173: C
174:      write(IPR,7000)
175:      &      MATT(N) (1:8), MATT(N) (9:16), MATT(N) (17:76),
176:      &      MATT(N) (77:136)
177:      end if
178: C
179:      KLIB1(N,1) = LST
180:      KLIB1(N,2) = NPTS
181:      KLIB1(N,3) = NTDATA
182:      KLIB1(N,4) = NUNR
183:      KLIB1(N,5) = NUNR2
184:      KLIB1(N,6) = NLEV
185:      KLIB1(N,7) = NBINA
186:      KLIB1(N,8) = NBINE
187:      KLIB1(N,9) = NNU
188:      KLIB1(N,10) = NMT0
189:      KLIB1(N,11) = NNK0
190:      KLIB1(N,12) = IGFLG
191:      KLIB1(N,13) = LFI
192:      KLIB1(N,14) = LNU
193:      KLIB1(N,15) = ITHERM
194:      KLIB1(N,16) = ISTU
195:      KLIB1(N,17) = IENDU

```

```

196:      KLIB1(N,18) = LSUNR
197:      KLIB1(N,19) = LSNU
198:      KLIB1(N,20) = NBINA1
199:      KLIB1(N,21) = NBINE1
200:      KLIB1(N,22) = NNUD
201:      KLIB1(N,23) = LNUD
202:      KLIB1(N,24) = NNF
203:      KLIB1(N,25) = LSNUD
204:      KLIB1(N,26) = NCAP
205:      KLIB1(N,27) = NCAPG
206: C
207:      KLIB1(N,28) = LF6
208:      KLIB1(N,29) = LSTF6
209:      KLIB1(N,30) = LLSSF
210: C
211:      KLIB1(N,31) = LASYM
212: C
213:      ATW = XLIB1(N,1)
214:      TLAB = XLIB1(N,2)
215:      ESAB = XLIB1(N,4)
216:      ELOW = XLIB1(N,5)
217:      EHI = XLIB1(N,6)
218: C
219:      TEFF = XLIB1(N,10)
220:      if ( TEFF.le.TLAB*1.001 ) then
221:      TEFF = TLAB
222:      end if
223:      ETOPL = MIN(ETOPL,EHI)
224:      EBOTL = MAX(EBOTL,ELOW)
225:      ESABL = MAX(ESABL,ESAB)
226:      XLIB1(N,7) = 1.0/(ATW+1.0)**2
227:      XLIB1(N,8) = 1.0/(ATW+1.0)
228: c <note> Y.Nagaya, 2017/01/16
229: c TLAB becomes zero when libraries generated from pseudo nuclides are
230: c used. So may does when "TZ" libraries are used for Doppler scattering
231: c option. A flag should be defined in the future.
232: c
233:      XLIB1(N,9) = ATW/(1.380662/1.6021892*1.0E-4*TLAB)
234:      if (TLAB.gt.0.) then
235:      XLIB1(N,9) = ATW/(1.380662/1.6021892*1.0E-4*TLAB)
236:      end if
237: c <note> Y.Nagaya, 2017/01/16
238: c TLAB becomes zero when libraries generated from pseudo nuclides are
239: c used. A flag should be defined in the future.
240: c
241:      XLIB1(N,11) = ATW/(1.380662/1.6021892*1.0E-4*TEFF)
242:      if (TEFF.gt.0.) then
243:      XLIB1(N,11) = ATW/(1.380662/1.6021892*1.0E-4*TEFF)
244:      end if
245: C
246: C-----
247: C READ #2 RECORD
248: C-----
249: C
250:      if ( INDEX(MATT(N)(17:136),'VERSION 2').ne.0.or.
251:      &      INDEX(MATT(N)(17:136),'VERSION 3').ne.0 ) then
252:      IVERS = 2 ! photonuc
253:      read(IXS) ((KLIB2(N,K,L),K=1,NMT),L=1,13),
254:      &      ((XLIB2(N,K,L),K=1,NMT),L=1,2),
255:      &      ((KLIB2(N,K,L),K=1,NMT),L=14,21),
256:      &      ((XLIB2(N,K,L),K=1,NMT),L=3,4)
257:      else if ( INDEX(MATT(N)(17:136),'VERSION 1').ne.0 ) then
258:      IVERS = 1 ! photonuc
259:      read(IXS) ((KLIB2(N,K,L),K=1,NMT),L=1,13),
260:      &      ((XLIB2(N,K,L),K=1,NMT),L=1,2),

```

src/mvp/xsec.f

```

261:      &                ((KLIB2(N,K,L),K=1,NMT),L=14,19)
262:      else
263:        write(IMG,'(1X,A,A,A)')
264:      &                'XXX(XSEC) Unsupported version of MVP lib!!  <',
265:      &                MATT(N) (17:25), ' >'
266:      &                call CNTERR('FATAL')
267:      end if
268: C
269: C -----
270: C  POINTER CACLUATION
271: C -----
272: C
273: C >>>>>> POINTERS >>>>>>  LSTF3, LSTF4, LSTF5, LSTGAM >>>>>>>>
274: C
275:      do 100 K = 1, NMT
276:        KLIB2(N,K,11) = KLIB2(N,K,11) + LST0
277:        KLIB2(N,K,12) = KLIB2(N,K,12) + LST0
278: C.....KLIB2(N,K,13) = KLIB2(N,K,13) + LST0
279: C
280: C .... POSITION OF LST5FS (NOT NEF5(MT))
281: C        KLIB2(N,K,13) = KLIB2(N,K,13) + LST0 + 2
282: C
283: C .... POSITION OF NEG(MT)
284: C        KLIB2(N,K,14) = KLIB2(N,K,14) + LST0
285: 100    continue
286: C
287: C >>>>>> POINTERS >>>>>>  LSUNR, LSNU, LSNUD
288: C
289:      KLIB1(N,18) = KLIB1(N,18) + LST0
290:      KLIB1(N,19) = KLIB1(N,19) + LST0
291:      KLIB1(N,25) = KLIB1(N,25) + LST0
292: C
293:      KLIB1(N,29) = KLIB1(N,29) + LST0
294: C
295:      LST0 = LST0 + NTDATA
296: C
297:      write(IPR,7050) LST, LST0
298: 7050    format(/1X,8X,'DATA FOR THIS NUCLIDE : FROM CX(' ,I12,') TO ',
299:      &                ' CX(' ,I12,')  '//)
300: C
301:      if ( LST0.gt.MMCX ) then
302:
303:        write(IMG,7100) LSIZ(LCX+LST0), LIMIT
304: C
305: 7100    format(/1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'//
306:      &                1X,' XXX  MEMORY OVER IN STORING X-SEC DATA !'//
307:      &                1X,' XXX  ',I10,' WORDS REQUIRED (LIMIT=' ,I10,')'//
308:      &                1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'//)
309: C
310:      call CNTERR( 'FATAL' )
311:      J999 = 1
312:      end if
313: C
314: C -----
315: C  READ #3 DATA
316: C -----
317:      if ( J999.eq.0 ) then
318:
319: C
320: C ... record 3 of free temperature nuclide is stored on memory in
321: C ARTLIB routine and no need to input here.
322: C
323:      if ( TEMPN(N).le.0.0d0 .or. JPOOL.ne.0 ) then
324:        read(IXS) (CX(I),I=LST,LST0)
325:      end if

```

```

326: C
327: C >>>>>> POINTERS >>>>>>  LSTF5S : SUBSECTION FOR SECONDARY NEUTRON
328: C
329:      do 130 K = 1, NMT
330: C
331: C//////// KLIB2(N,K,1) : MTINFO ,  KLIB2(N,K,2) : MTPAR
332: C
333:      if ( KLIB2(N,K,1).gt.0 .or. KLIB2(N,K,2).gt.0 ) then
334: C
335: C//////// KLIB2(N,K,8) : NKF5,  KLIB2(N,K,13) : LSTF5
336: C
337:      do 120 I = 1, IABS(KLIB2(N,K,8))
338:        II = KLIB2(N,K,13) + I - 1
339:        ICX(II) = ICX(II) + LST - 1
340:        LF = ICX(ICX(II))
341:        if ( NBINE.eq.0.and.
342:      &                (LF.eq.1.or.LF.eq.61.or.LF.eq.67 ) ) then
343:          NEF5S = ICX(ICX(II)+1)
344:          III = ICX(II) + 1 + 2*NEF5S
345:          do 110 L = 1, NEF5S
346:            ICX(III+L) = ICX(III+L) + LST - 1
347: C
348:          if ( LF.eq.67 ) then
349:            IIII = ICX(III+L) + NBINAL + 1
350:            NMU = ICX(IIII)
351:            IIII = IIII + NMU
352:            do 115 LL = 1, NMU
353:              ICX(IIII+LL) = ICX(IIII+LL) + LST - 1
354: 115          continue
355:          endif
356: C
357: 110          continue
358:          end if
359: C
360:          if ( LF.eq.71 ) then
361:            NEBRAG = ICX(ICX(II)+1)
362:            III = ICX(II) + 1 + 2*NEBRAG
363:            do 116 L = 1, NEBRAG
364:              ICX(III+L) = ICX(III+L) + LST - 1
365: 116          continue
366:          end if
367: C
368: 120          continue
369:          end if
370: 130          continue
371: C
372: C >>>>>> POINTERS >>>>>>  LSTF5G : SUBSECTION FOR SECONDARY PHOTON
373: C
374:      do 160 K = 1, NMT
375: C
376: C        KLIB2(N,K,14) : LSTGAM
377: C        KLIB2(N,K,16) : NEG
378: C        KLIB2(N,K,17) : NEGP
379: C        KLIB2(N,K,18) : NKG
380: C
381:      if ( KLIB2(N,K,17).gt.0 .or. KLIB2(N,K,18).gt.0 ) then
382:        II = KLIB2(N,K,14) + KLIB2(N,K,16)*2 + 2
383:        do 150 I = 1, KLIB2(N,K,18)
384:          ICX(II+I) = ICX(II+I) + LST - 1
385:          LF = ICX(ICX(II+I))
386:          NEF5S = ICX(ICX(II+I)+1)
387:          if ( NBINE.eq.0.and.(LF.eq.6.or.LF.eq.61) ) then
388:            III = ICX(II+I) + 1 + 2*NEF5S
389: C >>>>>> POINTERS >>>>>>  LSTF5E : ENERGY DISTRIBUTION FOR EACH
390:          do 140 L = 1, NEF5S

```

```
src/mvp/xsec.f
```

```

391:          ICX(III+L) = ICX(III+L) + LST - 1
392: 140          continue
393:          III = ICX(III+NEF5S)
394:          NEP = ICX(III)
395:          III = 1 + NEP*3 + (NEP+1)*2 + III
396:          if ( LF.eq.61 ) III = III + (NEP+1)*2 + 1
397:          else if ( LF.eq.6 ) then
398:            III = ICX(II+I) + 2 + 2*NEF5S*(1+NBINE1)
399:          else
400:            III = ICX(II+I) + 4
401:          end if
402: C >>>>> POINTERS >>>>> LSTGAN : ANGULAR DISTRIBUTION
403:          ICX(III) = ICX(III) + LST - 1
404: 150          continue
405:          end if
406: 160          continue
407:          else
408:            read(IXS)
409:          end if
410:          LST = LST0 + 1
411: C
412:          close( IXS )
413: C
414: 170 continue
415: C
416: C-----
417: C Memory over
418: C-----
419:          if ( J999.ne.0 ) then
420:            call PRSTOP( 0, 'MEMORY OVER IN NEUTRON CROSS SECTION INPUT.' )
421:            stop 999
422:          end if
423: C
424: C-----
425: C SET LENGTH OF CX ARRAY (MCX) & LAST POSITION POINTER (LAST)
426: C-----
427: C
428:          MCX = LST0
429:          call RESIZE( 'CX', LCX, MCX, 'R4', LAST )
430: C
431: C-----
432: C PRINT OUT OF CROSS SECTION DATA
433: C-----
434: C
435:          call LABEL( IPR, 'NUCLIDE SPECIFICATION DATA' )
436:          write(IPR,7300)
437:          write(IPR,7310) (NUCID(N), (KLIB1(N,K), K=1,13), N=1, NUC)
438: C
439: 7300 format(3x, ' NAME ', 8x,
440: & ' START NPTS NDATA UNNR UNNR2 NLEV ',
441: & ' NBINA NBINE NNU NMT NNK IGFLG LFI' /
442: & ' ----- ', 12('----- '))
443: 7310 format(1X, 2X, A, I12, I2I8)
444: C
445:          write(IPR,7320)
446:          write(IPR,7330) (NUCID(N), (KLIB1(N,K), K=14, 25), N=1, NUC)
447: C
448: 7320 format(/3x, ' NAME ', 8x,
449: & ' LNU ITERM ISTU IENDU', 2x, ' LUNR', 7x, ' LSNU', 7x,
450: & ' NBINA1 NBINE1 NNUD LNUD NNF', 4x, ' LSNUD' /
451: & ' ----- ', 4('----- '), 2('----- '),
452: & 5('----- '), ('----- '))
453: 7330 format(1X, 2X, A, 4I8, 2I12, 5I8, I12)
454: C
455:          write(IPR,7340)

```

```

456: write(IPR,7350) (NUCID(N),(XLIB1(N,L),L=1,8),N=1,NUC)
457: C
458: 7340 format(/3x,' NAME ',8x,
459: & ' ATW TLAB ETHEM ESAB ',
460: & ' ELOW EHI 1/(A+1)**2 1/(A+1)/'
461: & ' ----- ',8('----- '))
462: 7350 format(1X,2X,A,1P,8E12.5)
463: C
464: write(IPR,7360)
465: write(IPR,7370) (NUCID(N),(XLIB1(N,L),L=9,11),N=1,NUC)
466: C
467: 7360 format(/3x,' NAME ',8x,
468: & ' A/(K*TLAB) TEFF A/(K*TEFF)/'
469: & ' ----- ',3('----- '))
470: 7370 format(1X,2X,A,1P,3E12.5)
471: C
472: C
473: C-----
474: C STOP IF ANY X-SEC INPUT ERROR
475: C-----
476: if ( NERR.gt.0 ) then
477: write(IMGMSG,*) 'XXX CROSS SECTION INPUT ERROR !! STOP ! XXX'
478: call PRSTOP( 0, 'ERROR IN CROSS SECTION INPUT.' )
479: stop 888
480: end if
481: C
482: C
483: C
484: if ( JDEBG(1).ne.0 ) then
485:
486: write(IPR,*) ' === KLIB2(NUC,NMT,1-13) ====='
487:
488: do 180 N = 1, NUC
489: write(IPR,('(1X,a,a,a)'))
490: & ' ==< ', NUCID(N)(:iclen(NUCID(N))), ' >== '
491: write(IPR,7400)
492: write(IPR,7410) (K,(KLIB2(N,K,L),L=1,13),K=1,NMT)
493: 180 continue
494:
495: 7400 format(/1X,' MT MTINFO MTPAR ISTMT IENDMT NEUMT ',
496: & 'LCTMT NEANG NKF5 NEF5 MTTHR LSTF3 ',
497: & 'LSTF4 LSTF5 '/' --- ',13('----- '))
498: 7410 format(1X,I4,I3I8)
499: C
500: write(IPR,*) ' === KLIB2(NUC,NMT,14-19) ====='
501:
502: do 190 N = 1, NUC
503: write(IPR,('(1X,a,a,a)'))
504: & ' ==< ', NUCID(N)(:iclen(NUCID(N))), ' >== '
505: write(IPR,7420)
506: write(IPR,7430) (K,(KLIB2(N,K,L),L=14,19),K=1,NMT)
507: 190 continue
508:
509: 7420 format(/1X,' MT LSTGAM MTTOG NEG NEGP NNK ',
510: & 'LCTMTG '/' --- ',6('----- '))
511: 7430 format(1X,I4,I6I8)
512: C
513: write(IPR,*) ' === SUBSECTION INFORMATION ====='
514: C
515: do 230 N = 1, NUC
516:
517: write(IPR,('(1X,a,a,a)'))
518: & ' ==< ', NUCID(N)(:iclen(NUCID(N))), ' >== '
519: write(IPR,*) ' -- NEF5S -----'
520: write(IPR,*) ' MT SUBSECTION -->'

```



```

      NKF5      = IABS(KLIB2(N,K,8))
      if ( NKF5.gt.0 ) write(IPR,7500) K,
        (ICX(ICX(KLIB2(N,K,13)+J)),J=0,NKF5-1)
      continue

      write(IPR,*) ' -- LSTF5S -----'
      write(IPR,*) ' MT      SUBSECTION -->'

      do 220 K = 1, NMT
        NKF5      = IABS(KLIB2(N,K,8))
        if ( NKF5.gt.0 ) write(IPR,7500) K,
          (ICX(KLIB2(N,K,13)+J),J=0,NKF5-1)
        continue
      continue

      format(1X,I4,10I8)

end if

CHECK OF LCTMT = KLIB2(NUC,MT,6)
THIS MODIFICATION IS REQUIRED WHEN MAT=1192(IRON IN ENDF/B4)

do 250 N = 1, NUC
  do 240 K = 1, KLIB1(N,6)
    MT      = KLIB2(N,K,10)
    if ( (MT.ge.10.and.MT.le.49) .or. MT.eq.91 ) then
      LCT      = KLIB2(N,MT,6)
      if ( LCT.eq.2 ) then
        write(IPR,'(/1x,a,i10,a,a,i4,a,i4,a)')
          '<MESSAGE>> LCTMT IS CHANGED FROM ', LCT,
            ' TO 1', ' FOR REACTION MT=', MT,
            ' OF NUCLIDE(', N, ') '
        KLIB2(N,MT,6) = 1
        call CNTERR( 'MESSAGE' )
      end if

      NKF5      = KLIB2(N,MT,8)
      if ( NKF5.lt.-1 ) then
        write(IMG,'(1x,a,i10,a,a,i4,a,i4,a)')
          'XXX NO. OF FILE 6 SUBSECTIONS ( ', IABS(NKF5),
            ' ) > 1', ' FOR REACTION MT=', MT,
            ' OF NUCLIDE(', N, ') '
        call CNTERR( 'FATAL' )
      endif
    endif
  enddo
enddo

```

```

575:         endif
576:     else if ( MT.eq.2 .or. (MT.ge.6.and.MT.le.9)
577:         &      .or. (MT.ge.51.and.MT.le.90) ) then
578:         LCT      = KLIB2(N,MT,6)
579:         if ( LCT.ne.2 .and. LCT.gt.0 ) then
580:             write(IPR,'(/lx,a,i10,a,a,i4,a,i4,a)')
581:             &      '<<MESSAGE>> LCTMT IS CHANGED FROM ', LCT,
582:             &      ' TO 2', ' FOR REACTION MT=', MT,
583:             &      ' OF NUCLIDE(', N, ') '
584:             KLIB2(N,MT,6) = 2
585:             call CNTERR( 'MESSAGE' )

```

src/mvp/xsecp.f

```

1:      subroutine XSECP( CXP,   ICXP, MMCXP, LCXP, MCXP,  LAST,  LIMIT,
2:      &                 NMTP,  MATTP, KLBPl, XLBP1, KLBp2, XLBP2,
3:      &                 ETOPLP,EBOTLP,NPATOM,NATMT, JDEBG )
4: C=====
5: C  PURPOSE :  CONTINUOUS ENERGY CROSS SECTION INPUT & PROCESSING
6: C             -  PHOTON LIBRARY -
7: C  CALLED IN: CTXSIN
8: C=====
9: C
10: C  CXP(*)          R4  PHOTON CROSS SECTION DATA
11: C  MCXP           I4  LENGTH OF CXP ARRAY
12: C
13: C  *** CROSS SECTION DATA FOR EACH ATOMIC ELEMENTS *****
14: C
15: C  NMTP           I4  NUMBER OF PHOTON REACTIONS
16: C
17: C  MATTP(NPATOM)  CH136  MAT-ID / DATE / COMMENT
18: C
19: C  KLBPl(NPATOM,20) I4  RECORD #1 (SPECIFICATION RECORD)
20: C
21: C      1:  STARTING POSITION OF DATA FOR A ATOMIC ELEMENT
22: C      2:  NPTS      3:  NDATA      4:  NMTP      5:  NSUBS      6:  NSF
23: C      7:  NFF       8:  NFEF
24: C      9:  LSTSF    10:  LSTFF    11:  NPEF    12:  LSPEF
25: C      (13-20 DUMMY FOR FUTURE USE)
26: C      ( NMTP = 60 :  FIXED )
27: C
28: C  XLBP1(NPATOM,10) R4  ATOMIC PARAMETERS
29: C
30: C      1:  AWR      2:  Z          3:  M0*C**2  4:  AK          5:  R0
31: C      6:  PAI*R0*2  7:  ELOW      8:  EHI
32: C
33: C  KLBp2(NPATOM,NMTP,6) I4  POINTERS TO #2 RECORD DATA (1)
34: C
35: C      1:  IDINFO  2:  MT          3:  ISTID  4:  IENDID  5:  LSTF3
36: C      6:  INT
37: C
38: C  XLBP2(NPATOM,NMTP,2) I4  POINTERS TO #2 RECORD DATA (2)
39: C
40: C      1:  EPE      2:  EFL
41: C
42: C  ETOPP          R4  UPPER ENERGY LIMIT SPECIFIED BY USER
43: C  EBOTP          R4  LOWER ENERGY LIMIT SPECIFIED BY USER
44: C
45: C  ETOPLP         R4  UPPER ENERGY LIMIT IN PHOTON LIBRARY
46: C  EBOTLP         R4  LOWER ENERGY LIMIT IN PHOTON LIBRARY
47: C
48: C
49: C
50: C
51: C      implicit real(A-H,O-Z)
52: C
53: C      integer NATMT(NPATOM)
54: C
55: C      integer JDEBG(*)
56: C
57: C      character MATTP(NPATOM)*136
58: C      integer KLBPl(NPATOM,20), KLBp2(NPATOM,NMTP,10)
59: C      real XLBP1(NPATOM,10), XLBP2(NPATOM,NMTP,2)
60: C
61: C      real CXP(MMCXP)
62: C      integer ICXP(MMCXP), IDUM(20)
63: C
64: C
65: C      character*2 CHSYMB

```

```

66:      external CHSYMB
67: C
68: C
69: C      character*128 FILE
70: C
71: C      include '../shared/INC/_IOUNIT'
72: C      include 'INC/_IOUNIT2'
73: C
74: C      .... I/O UNIT NUMBER OF PHOTON LIBRARY INDEX FILE .....
75: C      --> SEE FALOCF ROUTINE
76: C
77: C
78: C      ETOPLP = 1.0E30
79: C      EBOTLP = -1.0E30
80: C      LST    = 1
81: C      LST0   = 0
82: C      NERR   = 0
83: C      J999   = 0
84: C
85: C
86: C      *****
87: C      call LABEL( IPR, 'PHOTON LIBRARY INPUT' )
88: C      *****
89: C
90: C      write(IPR, '(/)')
91: C      NERR = 0
92: C
93: C
94: C      do 120 N = 1, NPATOM
95: C          NZ = NATMT(N)
96: C
97: C          write(IPR, '(1X, '' == ATOM '', I3, '' : <'', A,
98: C      &          ''> ATOMIC NUMBER = '', I3)') N, CHSYMB(NZ), NZ
99: C
100: C -----
101: C      OPEN LIBRARY FILE
102: C -----
103: C      100 call FALOCF( IXP, CHSYMB(NZ), FILE, IERR )
104: C      if ( IERR.ne.0 ) then
105: C          NERR = NERR + 1
106: C          write(IMG,*)
107: C      &          'XXX(XSECP) CANNOT FIND DATA FOR <', CHSYMB(NZ), '>'
108: C          call CNTERR( 'FATAL' )
109: C          go to 120
110: C      end if
111: C
112: C      LFN = ICLEN(FILE)
113: C      write(IPR,*) '      <', CHSYMB(NZ), '> FILE = ', FILE(:LFN)
114: C
115: C -----
116: C      READ SPECIFICATION RECORD
117: C -----
118: C
119: C      read(IXP) MATTP(N) (1:8), MATTP(N) (9:16), MATTP(N) (17:136),
120: C      &      NPTS, NTDATA, NMT, NSUBS, NSF, NFF, NFEF, LSTSF, LSTFF,
121: C      &      NPEF, LSPEF, (IDUM(K),K=1,20), (XLBP1(N,L),L=1,8)
122: C
123: C
124: C      .... LIBRARY SPECIFICATION ERROR ? ATOMIC NUMBER MISMATCH ...
125: C
126: C
127: C      ccccc IF( NZ .NE. INT( XLBP1(N,2) ) ) THEN
128: C      if ( NZ.ne.NINT(XLBP1(N,2)) ) then
129: C          NERR = NERR + 1
130: C          call CNTERR( 'FATAL' )

```

src/mvp/xsecp.f

```

131:      write(IMG,'(1x,a,a,a)') ' XXX ATOMIC NUMBER MISMATCH <',
132:      &      CHSYMB(NZ), ' >'
133:      write(IMG,'(1x,a,a)') ' : FILE = ', FILE(:LFN)
134:      write(IMG,'(1x,a,a)') ' : MATTP = ', MATTP(N) (1:8)
135:      write(IMG,'(1x,a,a)') ' : DATE = ', MATTP(N) (9:16)
136:      write(IMG,'(1x,a,a)') ' : COMMENT: ', MATTP(N) (17:136)
137: c##<2007/03/14:PN3:
138: c##      write(IMG,'(1x,a,i4,a)') ' : NZ = ', NZ, ' Z IN FILE ',
139: c## &      XLBP1(N,2)
140:      write(IMG,'(1x,a,i4,a,1p,e12.5)') ' : NZ = ', NZ,
141:      &      ' Z IN FILE ', XLBP1(N,2)
142: c##>
143:      go to 120
144: ccccccc GOTO 111
145:      end if
146: C
147:      KLBPl(N,1) = LST
148:      KLBPl(N,2) = NPTS
149:      KLBPl(N,3) = NTDATA
150:      KLBPl(N,4) = NMT
151:      KLBPl(N,5) = NSUBS
152:      KLBPl(N,6) = NSF
153:      KLBPl(N,7) = NFF
154:      KLBPl(N,8) = NFEE
155:      KLBPl(N,9) = LSTSF
156:      KLBPl(N,10) = LSTFF
157:      KLBPl(N,11) = NPEF
158:      KLBPl(N,12) = LSPEF
159: C
160:      ELOW = XLBP1(N,7)
161:      EHI = XLBP1(N,8)
162:      ETOPLP = MIN(ETOPLP,EHI)
163:      EBOTLP = MAX(EBOTLP,ELOW)
164: C
165: C-----
166: C READ #2 RECORD
167: C-----
168: C
169:      read(IXP) ((KLBPl(N,K,L),K=1,NMTP),L=1,6),
170:      &      ((XLBP2(N,K,L),K=1,NMTP),L=1,2)
171: C
172: C-----
173: C POINTER CALCULATION
174: C-----
175: C
176: C >>>>>> POINTERS >>>>>> LSTF3 >>>>>>>>>
177: C
178:      do 110 K = 1, NMTP
179:      KLBPl(N,K,5) = KLBPl(N,K,5) + LST0
180: 110 continue
181: C
182: C >>>>>> POINTERS >>>>>> LSTSF, LSTFF, LSPEF >>>>>>>>>
183: C
184: C
185:      KLBPl(N,9) = KLBPl(N,9) + LST0
186:      KLBPl(N,10) = KLBPl(N,10) + LST0
187:      KLBPl(N,12) = KLBPl(N,12) + LST0
188: C
189:      LST0 = LST0 + NTDATA
190: C      ENDIF
191:      write(IPR,7000) LST, LST0
192: 7000 format(/1X,8X,'DATA FOR THIS ATOM : FROM CXP(' ,I12,') TO ',
193:      &      ' CXP(' ,I12,')')/
194: C
195:      if ( LST0.gt.MMCXP ) then

```

```

196:      call MEMERR( 'XSECP ',
197:      &      'IN KEEPING MEMORY FOR RECORD #3 DATA FOR PHOTON',
198:      &      LSIZ(LCXP+LST0), LIMIT )
199:      J999 = 1
200:      end if
201: C
202: C-----
203: C READ #3 DATA
204: C-----
205: C
206:      if ( J999.eq.0 ) then
207:      read(IXP) (CXP(I),I=LST,LST0)
208:      else
209:      read(IXP)
210:      end if
211:      LST = LST0 + 1
212: C
213:      close( IXP )
214: C
215: 120 continue
216: C
217: C-----
218: C Memory over
219: C-----
220:      if ( J999.ne.0 ) then
221: c##<2007/03/14:PN3:
222: c##      call PRSTOP( 0, 'MEMORY OVER IN NEUTRON CROSS SECTION INPUT.' )
223: c##      call PRSTOP( 0,
224:      &      'MEMORY OVER IN PHOTO-ATOMIC CROSS SECTION INPUT.' )
225: c##>
226:      stop 999
227:      end if
228: C
229: C-----
230: C SET LENGTH OF CXP ARRAY (MCXP) & LAST POSITION POINTER (LAST)
231: C-----
232: C
233:      MCXP = LST0
234:      call RESIZE( 'CXP', LCXP, MCXP, 'R4', LAST )
235: C
236: C-----
237: C PRINT OUT OF CROSS SECTION DATA
238: C-----
239: C
240:      call LABEL( IPR, 'ATOM SPECIFICATION DATA' )
241:      write(IPR,7020)
242:      write(IPR,7040) (CHSYMB(NATMT(N)), (KLBPl(N,K),K=1,12),N=1,NPATOM)
243: 7020 format(
244:      &      ' NAME START NPTS NDATA NMT NSUBS NSF ',
245:      &      ' NFF NFEE LSTSF LSTFF NPEF LSPEF'/
246:      &      ' ----- ,12('----- ')')
247: c##<2007/03/14:PN3:
248: c7040 format(1X,6X,A2,2X,12I8)
249: c7040 format(1X,5X,A2,3X,12I8)
250: c##>
251: C
252:      write(IPR,7060)
253:      write(IPR,7080) (CHSYMB(NATMT(N)), (KLBPl(N,L),L=1,8),N=1,NPATOM)
254: 7060 format(1X/' =====//
255: c##<2007/03/14:PN3:
256: c## & ' NAME AWR Z M0*C**2 AK ',
257: c## & ' R0 PAI*R0**2 ELOW EHI'/
258: c## & ' ----- ,8('----- ')')
259: c7080 format(1X,6X,A2,2X,1P,8E12.5)
260:      &      ' NAME AWR Z M0*C**2 AK ',

```

src/mvp/xsecp.f

```
261:      &          ' R0          PAI*R0**2  ELOW      EHI'
262:      &          '/'  ----- ',8('----- '))
263:      7080 format(1X,5X,A2,3X,1P,8E12.5)
264: c##>
265: C
266: C      ..... LIBRARY INPUT ERROR .....
267: C
268:      if ( NERR.ne.0 ) then
269:          write(IMG,'(/1x,a/)') 'XXX PHOTON LIBRARY INPUT ERROR !!'
270:          call PRSTOP( 0, 'PHOTON LIBRARY INPUT ERROR.' )
271:          stop 888
272:      end if
273: C
274: C
275:      if ( JDEBG(1).ne.0 ) then
276: c##<2007/03/14*PN3:
277: c##      write(IPR,*) ' === KLBP2(NPATOM,NMT,1-8 ) ====='
278: c##      write(IPR,*) ' === KLBP2(NPATOM,NMTP,1-8 ) ====='
279: c##>
280:      do 130 N = 1, NPATOM
281:          write(IPR,*) ' '
282:          write(IPR,*) ' ==< ', CHSYMB(NATMT(N)), ' >== '
283:          write(IPR,7100)
284:          write(IPR,7120) (K,(KLBP2(N,K,L),L=1,8),K=1,NMTP)
285:      130 continue
286:      7100 format(/1X,'      IDINFO MT      ISTID  IENDID  LSTF3  ',
287:      &          'INT'/'  --- ',8('----- '))
288:      7120 format(1X,I4,8I8)
289: C
290: C
291:      end if
292:      return
293:      end
```

src/mvp/xsecpn.f

```

1:      subroutine XSECPN(CXPN, ICXPN, MMCXPN,LCXPN, MCXPN, LAST, LIMIT,
2:      &                  NMTPN, MATTPN,KLBPN1,XLBPN1,KLBPN2,XLBPN2,
3:      &                  ETOPLPN, EBOTLPN, NUC, NUCPNI,NCIDP,
4:      &                  NCIDPN,NMAT, MLEN, MNUC, LPDEN, INUCT,
5:      &                  NUCPNA,NUCPNB,NUCPN, JDEBG, JPNLCP)
6: C=<MVP>=====
7: C PURPOSE : Continuous energy cross section input & processing
8: C          (photonuclear data library)
9: C CALLED IN: CTXSIN
10: C CALLS : FALOCPN CNTERR LABEL
11: C=====
12:      implicit real(A-H,O-Z)
13:      include '../shared/INC/_IOUNIT'
14:      include 'INC/_IOUNIT2'
15:      include 'INC/_ISOTOP'
16: C
17:      real CXPN(MMCXPN)
18:      integer ICXPN(MMCXPN), MNUC(NMAT), LPDEN(NMAT+1), INUCT(MLEN),
19:      &      JDEBG(*)
20:      character MATTPN(NUCPNI)*136, NCIDP(NUC)*16, NCIDPN(NUCPNI)*16
21: C
22:      integer KLBPN1(NUCPNI,16), KLBPN2(NUCPNI,NMTPN,13)
23:      real XLBPN1(NUCPNI,6), XLBPN2(NUCPNI,NMTPN,2)
24: C
25: C .... data related to photo-nuclear in MTDATPN ....
26:      integer MNUCPN(NMAT+1), NNUCPN(NUCPNB,NMAT), LPDEN(NUCPNB,NMAT),
27:      &      LPIDPN(NUCPNA), IZTBN(NUCPNA), MNEUTPN(NUCPNA)
28:      real DENST(MLEN), DNSTPN(NUCPNA)
29: C
30: C .... local variables ....
31:      parameter (MAXLCL=120)
32:      integer INMNPN(MAXLCL), LNMNPN(MAXLCL), IZZZPN(5*MAXLCL), IDPM(20)
33:      character FILE*256, PNID*16, CHSYMB*2, HH*2, HABC0*10,HABC(8:9)*1,
34:      &      NCIDPN0*16, NCIDPN9(5*MAXLCL)*16
35:      equivalence (HABC0,HABC)
36:      external CHSYMB
37:      save NCIDPN9, INMNPN, LNMNPN, IZZZPN
38:      data HABC0 / 'JABCDEFGHI' /
39: C
40: C -----
41: C
42:      ETOPLPN = -1.0E30
43:      EBOTLPN = 1.0E30
44:      LST = 1
45:      LST0 = 0
46:      J999 = 0
47: C
48:      *****
49:      call LABEL( IPR, 'PHOTO-NUCLEAR DATA LIBRARY INPUT' )
50:      *****
51:      if ( NUC.gt.MAXLCL ) then
52:      &      write(IMG,*) 'XXX(XSECPN) Local maximum number of photo-',
53:      &      'nuclear nuclides is exceeded.'
54:      &      call CNTERR( 'FATAL' )
55:      &      stop 999
56:      end if
57:      N = 0
58:      N9 = 0
59: C
60:      do 170 NN = 1, NUC
61:      PNID = NCIDP(NN)
62:      LPN = ICLEN(PNID)
63:      IZBB = 0
64:      NAA0 = 0
65:      INMNPN(NN) = 0
66:      LNMNPN(NN) = 0

```

```

66:      HH = PNID(1:2)
67:      if ( HH(2:2).eq.'0' ) HH(2:2) = ' '
68:      do IZB = 1, MATOM
69:      if ( HH.eq.CHSYMB(IZB) ) go to 80
70:      end do
71:      write(IMG,7510) PNID(:LPN),HH
72: 7510 format(/' XXX(XSECPN) atomic symbol for <',a,'> was',
73:      &      ' unknown !!! ... ',a)
74:      call CNTERR('FATAL')
75:      go to 170
76: 80 continue
77:      if ( LPN.eq.8 ) then
78:      IVERF = 30
79:      else if ( LPN.ge.9 ) then
80:      IVERF = 31
81:      else
82:      IVERF = 30
83:      end if
84: C ..... case given nuclide ID for photo-nuclear data
85:      if ( PNID(LPNI-1:LPNI).eq.'PN' ) then
86: C ..... check the natural element
87:      if ( PNID(3:3).eq.'N' ) then
88:      PNID(3:3) = ' '
89:      IKK = ISOTOP(IZB)
90:      NAA0 = nint(TISOTP(1,1,IZB))
91:      IZBB = 1
92:      else if ( IVERF.eq.31.and.PNID(3:5).eq.'000' ) then
93:      PNID(3:5) = ' '
94:      IKK = ISOTOP(IZB)
95:      NAA0 = nint(TISOTP(1,1,IZB))
96:      IZBB = 1
97:      else
98:      IKK = 1
99:      end if
100: C ..... case given nuclide ID for neutron data
101:      else
102: C ..... check the natural element
103:      if ( PNID(3:3).eq.'N' ) then
104:      PNID(3:16) = ' '
105:      IKK = ISOTOP(IZB)
106:      NAA0 = nint(TISOTP(1,1,IZB))
107:      IZBB = 1
108:      else if ( IVERF.eq.31.and.PNID(3:5).eq.'000' ) then
109:      PNID(3:16) = ' '
110:      IKK = ISOTOP(IZB)
111:      NAA0 = nint(TISOTP(1,1,IZB))
112:      IZBB = 1
113:      else
114:      if ( IVERF.eq.30 ) then
115:      PNID(4:16) = ' '
116:      else if ( IVERF.eq.31 ) then
117:      PNID(6:16) = ' '
118:      end if
119:      IKK = 1
120:      end if
121:      end if
122:      IK = 0
123:      INMNPN(NN) = IKK
124: C
125: 90 IK = IK + 1
126:      if ( IK.gt.IKK ) go to 170
127:      N = N + 1
128:      N9 = N9 + 1
129:      MATTPN(N) = ' '
130:      NCIDPN9(N9) = ' '

```

src/mvp/xsecpn.f

```

131:      IZZZPN(N9) = 0
132:      if ( IZBB.eq.1 ) then
133:        NAA = nint(TISOTP(1,IK,IZB))
134:        NCIDPN0 = PNID
135:        if ( IVERF.eq.30 ) then
136:          NAAL = NAA - (NAA/10)*10
137:          if ( NAA.lt.NAA0+10 ) then
138:            write(NCIDPN0(3:3),'(I1)') NAAL
139:          else
140:            write(NCIDPN0(3:3),'(A1)') HABC(NAAL)
141:          end if
142:        else if ( IVERF.eq.31 ) then
143:          write(NCIDPN0(3:5),'(I3.3)') NAA
144:        end if
145:      else
146:        NCIDPN0 = PNID
147:      end if
148: C
149: C-----
150: C OPEN PHOTO-NUCLEAR LIBRARY FILE
151: C-----
152: C
153:      call FALOCNP( IXP, NCIDPN0, FILE, IERR )
154: C
155:      if ( IERR.eq.1 ) then ! without photo-nuclear data
156:        N = N - 1
157:        N9 = N9 - 1
158:        INMNP(N) = INMNP(N) - 1
159:        go to 90
160:      end if
161:      if ( LNMNP(N).le.0 ) LNMNP(N) = N9
162:      NCIDPN9(N9) = NCIDPN0
163:      IZZZPN(N9) = IZB * 1000 + IK
164: C
165:      if ( N.gt.1 ) then
166:        do I = 1, N-1
167:          if ( NCIDPN0.eq.NCIDPN(I) ) then
168:            N = N - 1
169:            go to 90
170:          end if
171:        end do
172:      end if
173:      NCIDPN(N) = NCIDPN0
174:      LFN = ICLEN(FILE)
175:      if ( N.gt.1 ) write(IPR,'(1h)')
176:      write(IPR,'(1X,A,A)') ' === NUCLIDE : ',
177:      & NCIDPN(N)(:ICLEN(NCIDPN(N)))
178:      write(IPR,'(1X,A,A,A)') ' FILE = <', FILE(:LFN), '>'
179: C
180: C-----
181: C READ #1 SPECIFICATION RECORD
182: C-----
183: C
184:      read(IXP) MATTPN(N),
185:      & NPTS, NTDATA, NMT2G, NBINA, NNU, NMT0, LFI, LNU, LSNU,
186:      & NNNU, LNU, NNF, LSNU, LSTPEA, IDUM,
187:      & (XLBP1(N,J),J=1,6)
188: C
189:      IVERSN = 31
190:      if ( MATTPN(N)(17:28).eq.'VERSION 3.1 ' ) then
191:        IVERSN = 31
192:        write(IPR,7001) MATTPN(N)(1:16), MATTPN(N)(17:76),
193:        & MATTPN(N)(77:128), MATTPN(N)(129:136)
194:      else if ( MATTPN(N)(17:28).eq.'VERSION 3 ' ) then
195:        IVERSN = 30

```

```

196:      write(IPR,7000) MATTPN(N)(1:8), MATTPN(N)(9:16),
197:      & MATTPN(N)(17:76), MATTPN(N)(77:136)
198:    end if
199: 7000 format(/6X,'ID: <',A8,'>',8X,'DATE: ',A8
200:    & /6X,'COMMENT: <',A60,'>'
201:    & /6X,' <',A60,'>' )
202: 7001 format(/6X,'ID: <',A16,'> : library format ... version 3.1'
203:    & /6X,'COMMENT: <',A60,'>'
204:    & /6X,' <',A52,'>'
205:    & /6X,'DATE: <',A8,'>' )
206: C
207:      KLBP1(N,1) = LST
208:      KLBP1(N,2) = NPTS
209:      KLBP1(N,3) = NTDATA
210:      KLBP1(N,4) = NMT2G
211:      KLBP1(N,5) = NBINA
212:      KLBP1(N,6) = NNU
213:      KLBP1(N,7) = NMT0
214:      KLBP1(N,8) = LFI
215:      KLBP1(N,9) = LNU
216:      KLBP1(N,10) = LSNU
217:      KLBP1(N,11) = NNNU
218:      KLBP1(N,12) = LNU
219:      KLBP1(N,13) = NNF
220:      KLBP1(N,14) = LSNU
221:      KLBP1(N,15) = LSTPEA
222:      KLBP1(N,16) = 0
223: C
224:      ATW = XLBP1(N,1)
225:      TLAB = XLBP1(N,2)
226:      ELW = XLBP1(N,3)
227:      EHI = XLBP1(N,4)
228:      ZA = XLBP1(N,5)
229:      RMTNO = XLBP1(N,6)
230:      ETOPLN = MAX(ETOPLN,EHI)
231:      EBOTLN = MIN(EBOTLN,ELW)
232: C
233: C-----
234: C READ #2 MT SECTION RECORD
235: C-----
236: C
237:      if ( IVERSN.eq.30.or.IVERSN.eq.31 ) then
238:        read(IXP) ((KLBP2(N,K,L),K=1,NMTPN),L=1,12),
239:        & ((KLBP2(N,K,L),K=1,NMTPN),L=1,2)
240: C
241:      else
242:        write(IMG,*) 'XXX(XSECPN) Unsupported version of MVP',
243:        & ' photo-nuclear library <', MATTPN(N) (17:28), '>'
244:        call CNTERR('FATAL')
245:        go to 90
246:      end if
247: C
248: C ..... POINTER CACULATION
249: C ..... POINTERS >>> LSTF3, LSTF4, LST2G
250:      do K = 1, NMTPN
251:        KLBP2(N,K,10) = KLBP2(N,K,10) + LST0
252:        KLBP2(N,K,11) = KLBP2(N,K,11) + LST0
253:        KLBP2(N,K,12) = KLBP2(N,K,12) + LST0
254:      end do
255: C
256: C ..... POINTERS >>> LSNU, LSNU, LSTPEA
257:      KLBP1(N,10) = KLBP1(N,10) + LST0
258:      KLBP1(N,14) = KLBP1(N,14) + LST0
259:      KLBP1(N,15) = KLBP1(N,15) + LST0
260: C

```

src/mvp/xsecpn.f

```

261:      LST0      = LST0 + NTDATA
262: C
263:      write(IPR,7050) LST, LST0
264: 7050      format(/1X,8X,'DATA FOR THIS NUCLIDE : FROM CXPN(' ,I12,' ) TO ' ,
265:      &      ' CXPN(' ,I12,' )')
266:      if ( LST0.gt.MMCXPN ) then
267:      write(IMG,7100) LST2(LCXPN+LST0), LIMIT
268: 7100      format(/1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /
269:      &      1X,' XXX MEMORY OVER IN STORING X-SEC DATA !!!' /
270:      &      1X,' XXX ',I10,' WORDS REQUIRED (LIMIT=' ,I10,' )' /
271:      &      1X,' XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' )
272:      call CNTERR( 'FATAL' )
273:      J999      = 1
274:      end if
275: C
276: C-----
277: C READ  #3  DATA RECORD
278: C-----
279: C
280:      if ( J999.eq.0 ) then
281: C
282:      read(IXPN) (CXPN(I),I=LST,LST0)
283: C
284: C ..... POINTERS >>> LST2G
285:      do K = 1, NMTPN
286:      II      = KLBNP1(N,15) + K - 1
287:      ICXPN(II) = ICXPN(II) + LST - 1
288:      end do
289: C
290:      do 130 K = 1, NMTPN
291:      if ( (KLBNP2(N,K,1).gt.0.or.K.eq.98).and.
292:      &      KLBNP2(N,K,7).ne.0 ) then
293:      KK      = KLBNP2(N,K,12) + 61
294: C
295: C ..... POINTERS >>> LST2GS, LSTF5E, (LST2A)
296:      do 120 I = 1, iabs(KLBNP2(N,K,7))
297:      ICXPN(KK+I) = ICXPN(KK+I) + LST - 1
298:      II      = ICXPN(KK+I)
299:      LF      = ICXPN(II)
300:      if ( LF.eq.1 .or. LF.eq.61 .or. LF.eq.62 ) then
301:      NEF5S    = ICXPN(II+1)
302:      III      = II + 1 + 2*NEF5S
303:      do L = 1, NEF5S
304:      ICXPN(III+L) = ICXPN(III+L) + LST - 1
305:      end do
306:      else if ( LF.eq.67 ) then
307:      NEF5S    = ICXPN(II+1)
308:      III      = II + 1 + 2*NEF5S
309:      do L = 1, NEF5S
310:      ICXPN(III+L) = ICXPN(III+L) + LST - 1
311:      IIJ      = ICXPN(III+L) + NBINA + 2
312:      NMU      = ICXPN(IIJ)
313:      IIJ      = IIJ + NMU
314:      do LL = 1, NMU
315:      ICXPN(IIJ+LL) = ICXPN(IIJ+LL) + LST - 1
316:      end do
317:      end do
318:      end if
319: 120      continue
320:      end if
321: 130      continue
322: C
323: C-----
324: C Eliminate charged particle data from stored data array
325: C-----

```

```

326: C
327:      if ( JPNLCP.ne.0 ) then
328:      LSTELIT = 0
329:      LST00   = LST0
330:      do K = 1, NMTPN
331:      if ( KLBNP2(N,K,1).gt.0.and.KLBNP2(N,K,7).lt.0 ) then
332:      LST2G99 = KLBNP2(N,K,12)
333:      NE2G99  = ICXPN(LST2G99)
334:      NK2G99  = ICXPN(LST2G99+1)
335:      IPNK    = 0
336:      if ( ICXPN(LST2G99+2).gt.0 )
337:      &      IPNK = ICXPN(LST2G99+2) + ICXPN(LST2G99+32) - 1
338:      if ( ICXPN(LST2G99+8).gt.0 )
339:      &      IPNK = ICXPN(LST2G99+8) + ICXPN(LST2G99+38) - 1
340: C
341: C ..... only neutron and/or gamma-ray: no elimination
342:      if ( IPNK.eq.NK2G99 ) go to 150
343: C
344:      NEF5S    = 0
345:      K99      = 0
346: C ..... no neutron and gamma-ray: full elimination
347:      if ( IPNK.eq.0 ) then
348:      do I = K+1, NMTPN
349:      if ( KLBNP2(N,I,12).gt.LST2G99 ) go to 140
350:      enddo
351:      LSTELI   = LST00 - LST2G99 + 1
352:      LSTNXT   = 0
353:      go to 141
354: 140      LSTELI = KLBNP2(N,I,12) - LST2G99
355:      LSTNXT   = KLBNP2(N,I,12)
356:      K99      = I
357: 141      continue
358:      LSTELIT = LSTELIT + LSTELI
359:      KLBNP2(N,K,7) = 0
360:      do I = 1, NMT2G
361:      if ( ICXPN(KLBNP1(N,15)+NMTPN+I-1).eq.K )
362:      &      go to 142
363:      end do
364:      write(IMG,7101) K
365:      call CNTERR('FATAL')
366:      go to 150
367: 142      if ( I.lt.NMT2G ) then
368:      do II = I+1, NMT2G
369:      ICXPN(KLBNP1(N,15)+NMTPN+II-2) =
370:      &      ICXPN(KLBNP1(N,15)+NMTPN+II-1)
371:      end do
372:      ICXPN(KLBNP1(N,15)+NMTPN+NMT2G-1) = 0
373:      end if
374:      NMT2G    = NMT2G - 1
375:      ICXPN(KLBNP1(N,15)+NMTPN*2) =
376:      &      ICXPN(KLBNP1(N,15)+NMTPN*2) - 1
377:      KLBNP2(N,K,8) = 0
378:      KLBNP2(N,K,12) = 0
379:      ICXPN(KLBNP1(N,15)+K-1) = 0
380: C
381: C ..... elimination of a part of charged particle data
382:      else
383:      LST2GS9 = ICXPN(LST2G99+62+IPNK)
384:      do I = K+1, NMTPN
385:      if ( KLBNP2(N,I,12).gt.LST2GS9 ) go to 145
386:      enddo
387:      LSTELI   = LST00 - LST2GS9 + 1
388:      LSTNXT   = 0
389:      go to 146
390: 145      LSTELI = KLBNP2(N,I,12) - LST2GS9

```

src/mvp/xsecpn.f

```

391:      LSTNXT = KLBPN2(N,I,12)
392:      K99 = I
393:      146 continue
394:      LSTELIT = LSTELIT + LSTELI
395:      KLBPN2(N,K,7) = - IPNK
396:      ICXPN(LST2G99+1) = IPNK
397:      do I = 1, 30
398:        if ( I.ne.1.and.I.ne.7 ) then
399:          ICXPN(LST2G99+I+1) = 0
400:          ICXPN(LST2G99+I+31) = 0
401:        end if
402:      end do
403:      do I = 1, IPNK+1
404:        ICXPN(LST0+I) = ICXPN(LST2G99+61+I)
405:      end do
406:      L = LST2G99 + 61 + IPNK
407:      L0 = LST2G99 + 61 + NK2G99
408:      do I = 1, IPNK
409:        CXPN(L+I) = CXPN(L0+I)
410:      end do
411:      L = L + IPNK
412:      L0 = L0 + NK2G99
413:      do I = 1, IPNK
414:        CXPN(L+I) = CXPN(L0+I)
415:      end do
416:      do I = 1, 2
417:        L = L + IPNK
418:        L0 = L0 + NK2G99
419:        do I = 1, IPNK
420:          ICXPN(L+I) = ICXPN(L0+I)
421:        end do
422:      end do
423:      L = L + IPNK
424:      L0 = L0 + NK2G99
425:      do I = 1, NE2G99*2
426:        CXPN(L+I) = CXPN(L0+I)
427:      end do
428:      L = L + NE2G99 * 2
429:      L0 = L0 + NE2G99 * 3
430:      do I = 1, NE2G99
431:        CXPN(L+I) = 0
432:      end do
433:      do J = 1, IPNK
434:        L1 = L0 + (J - 1) * NE2G99
435:        do I = 1, NE2G99
436:          CXPN(L+I) = CXPN(L+I) + CXPN(L1+I)
437:        end do
438:      end do
439:      L = L + NE2G99
440:      do J = 1, IPNK
441:        L2 = L + (J - 1) * NE2G99
442:        L1 = L0 + (J - 1) * NE2G99
443:        do I = 1, NE2G99
444:          CXPN(L2+I) = CXPN(L1+I)
445:        end do
446:      end do
447:      LSTELIK = (NK2G99 - IPNK) * (5 + NE2G99)
448:      L2 = L2 + NE2G99 + 1
449:      if ( ICXPN(LST2G99+62)-L2.ne.LSTELIK ) then
450:        write(IMG,7102) ICXPN(LST2G99+62),L2,LSTELIK
451:        call CNTERR('FATAL')
452:        go to 150
453:      end if
454:      LSTELIT = LSTELIT + LSTELIK
455:      LSTELI = LSTELI + LSTELIK

```

```

456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:

```

&

```

L = LST2G99 + 61
do J = 1, IPNK
  L1 = ICXPN(LST0+J)
  L11 = ICXPN(LST0+J+1) - 1
  ICXPN(L+J) = L2
  LF = ICXPN(L1)
  if ( LF.eq.61.or.LF.eq.62.or.LF.eq.67 ) then
    NEF5S = ICXPN(L1+1)
    do I = 1, NEF5S*2+2
      CXPN(L2+I-1) = CXPN(L1+I-1)
    end do
    L0 = L2 + NEF5S * 2 + 2
    L1 = L1 + NEF5S * 2 + 2
    L2 = L0 + NEF5S
    L00 = ICXPN(L1) - L2
    if ( L00.ne.LSTELIK ) then
      write(IMG,7103) L00,ICXPN(L1),L2,L0,
        LSTELIK
      call CNTERR('FATAL')
      go to 150
    end if
    do I = 1, NEF5S
      ICXPN(L0+I-1) = ICXPN(L1+I-1) - L00
    end do
    L1 = L1 + NEF5S
  end if
  if ( LF.eq.60.or.LF.eq.64 ) then
    ICXPN(L2) = ICXPN(L1)
    L2 = L2 + 1
  else if ( LF.eq.61.or.LF.eq.62 ) then
    II = 0
    do I = L1, L11
      II = II + 1
      CXPN(L2+II-1) = CXPN(I)
    end do
    L2 = L2 + II
  else if ( LF.eq.66 ) then
    do I = 1, 9
      CXPN(L2+I-1) = CXPN(L1+I-1)
    end do
    L2 = L2 + 9
  else if ( LF.eq.67 ) then
    do J0 = 1, NEF5S
      NMU = ICXPN(L1+NBINA+2)
      do I = 1, NBINA+NMU+3
        CXPN(L2+I-1) = CXPN(L1+I-1)
      end do
      L2 = L2 + NBINA + NMU + 3
      L1 = L1 + NBINA + NMU + 3
      do I = 1, NMU
        ICXPN(L2+I-1) = ICXPN(L1+I-1) - L00
      end do
      L2 = L2 + NMU
      L1 = L1 + NMU
      do I = 1, NMU
        NEP = ICXPN(L1)
        II = NEP * 5 + 3
        do I0 = 1, II
          CXPN(L2+I0-1) = CXPN(L1+I0-1)
        end do
        L2 = L2 + II
        L1 = L1 + II
      end do
    end do
  end if
end if

```


src/mvp/xsecpn.f

```

521:         end do
522:     end if
523: C
524:     if ( LSTNXT.gt.0 ) then
525:         do K0 = K99, NMTPN
526:             if ( (KLBPN2(N,K0,1).gt.0.or.K0.eq.98).and.
527: &             KLBPN2(N,K0,7).ne.0 ) then
528:                 KK = KLBPN2(N,K0,12) + 61
529:                 do I = 1, abs(KLBPN2(N,K0,7))
530:                     II = ICXPN(KK+I)
531:                     ICXPN(KK+I) = ICXPN(KK+I) - LSTELI
532:                     LF = ICXPN(II)
533:                     if ( LF.eq.61.or.LF.eq.62.or.LF.eq.1 )
534: &                     then
535:                         NEF5S = ICXPN(II+1)
536:                         III = II + 1 + 2*NEF5S
537:                         do L = 1, NEF5S
538:                             ICXPN(III+L) = ICXPN(III+L) -
539: &                             LSTELI
540:                         enddo
541:                         else if ( LF.eq.67 ) then
542:                             NEF5S = ICXPN(II+1)
543:                             III = II + 1 + 2*NEF5S
544:                             do L = 1, NEF5S
545:                                 ICXPN(III+L) = ICXPN(III+L) -
546: &                                 LSTELI
547:                                 IIJ = ICXPN(III+L) + NBINA + 2
548:                                 NMU = ICXPN(IIJ)
549:                                 IIJ = IIJ + NMU
550:                                 do LL = 1, NMU
551:                                     ICXPN(IIJ+LL) =
552: &                                     ICXPN(IIJ+LL) - LSTELI
553:                                 end do
554:                             end do
555:                         end if
556:                     end do
557:                 end if
558:             end do
559:             I = 0
560:             do L = LSTNXT, LST00
561:                 I = I + 1
562:                 CXPN(L2+I-1) = CXPN(L)
563:             end do
564:             LST00 = L2 + I - 1
565:             do K0 = K99, NMTPN
566:                 if ( KLBPN2(N,K0,12).gt.0 ) then
567:                     KLBPN2(N,K0,12) = KLBPN2(N,K0,12) - LSTELI
568:                     ICXPN(KLBPN1(N,15)+K0-1) =
569: &                     ICXPN(KLBPN1(N,15)+K0-1) - LSTELI
570:                 end if
571:             end do
572:         end if
573:     end if
574: 150 continue
575: end do
576: if ( LSTELIT.gt.0 ) then
577:     NTDATA = NTDATA - LSTELIT
578:     KLBPN1(N,3) = NTDATA
579:     KLBPN1(N,4) = NMT2G
580:     LST0 = LST0 - LSTELIT
581:     write(IPR,7051) LST, LST0
582: end if
583: end if
584: C
585: else

```

```

586:         read(IXPN)
587:     end if
588:     LST = LST0 + 1
589:     close( IXPN )
590:     go to 90
591: C
592: 170 continue
593: C
594: 7051 format(33X,'FROM CXPN('',I12,'') TO CXPN('',I12,'') : eliminated',
595: & ' charged particle data')
596: 7101 format(' XXX(XSECPN) In elimination of charged particle,',
597: & ' requested mt (='',i3,'') was not found in MT2G array.')
598: 7102 format(' XXX(XSECPN) In elimination of charged particle,',
599: & ' the eliminated number of header part was inconsistency.')
600: &/' LST2GS='',i8,' L2='',i8,' LSTELIK='',i7)
601: 7103 format(' XXX(XSECPN) In elimination of charged particle,',
602: & ' the eliminated number for starting position of incident',
603: & ' energy was not same.')
604: &/' L00='',i6,' LSTF5E='',i8,' L2='',i8,' L0='',i8,
605: & ' LSTELIK='',i8)
606: C
607: C-----
608: C Memory over and other error
609: C-----
610: C
611:     if ( J999.ne.0 ) then
612:         call PRSTOP( 0,
613: &         'Memory over in photo-nuclear cross section input.' )
614:         stop 999
615:     end if
616:     if ( N.gt.NUCPNI ) then
617:         write(IMG*,*) 'XXX(XSECPN) The number of nuclides for photo-',
618: &         'nuclear data was exceeded the presetted value.',
619: &         ' n='',n,' nucpni='',nucpni
620:         call CNTERR( 'FATAL' )
621:     else if ( N.le.0 ) then
622:         write(IMG*,*) 'XXX(XSECPN) Nuclides with photo-nuclear data',
623: &         ' are nothing. Please photo-nuclear option is',
624: &         ' off.'
625:         call CNTERR( 'FATAL' )
626:     end if
627:     call CHKERR( 'FATAL', J999 )
628:     if ( J999.gt.0 ) then
629:         call PRSTOP( 0, 'Terminated by above fatal error.' )
630:         stop 999
631:     end if
632:     NUCPN = N
633: C
634: C-----
635: C SET LENGTH OF CXPN ARRAY (MCXPN) & LAST POSITION POINTER (LAST)
636: C-----
637: C
638:     MCXPN = LST0
639:     call RESIZE( 'CXPN', LCXPN, MCXPN, 'R4', LAST )
640: C
641: C-----
642: C PRINT OUT OF CROSS SECTION DATA
643: C-----
644: C
645:     call LABEL( IPR, 'photo-nuclear specification data' )
646:     write(IPR,7300)
647:     write(IPR,7310) (NCIDPN(N), (KLBPN1(N,K), K=1,13), N=1, NUCPN)
648: 7300 format(3x, ' NAME ', 8x,
649: & ' START NPTS NTDATA NMT2G NBINA NNU ',
650: & ' NMT LFI LNU LSNU NNUD LNUD NNF'

```

src/mvp/xsecpn.f

```

651:      &/' ----- ' ,9('----- '),'----- ' ,3('----- ')
652:      7310 format(3X,A,9I8,I9,3I7)
653: C
654:      write(IPR,7320)
655:      write(IPR,7330) (NCIDPN(N), (KLBNP1(N,K), K=14,15), N=1, NUCPN)
656:      7320 format(/3x,' NAME ',8x,
657:      & ' LSNUD LSTPEA'
658:      & /' ----- ' ,2('----- '))
659:      7330 format(3X,A,2I9)
660: C
661:      write(IPR,7340)
662:      write(IPR,7350) (NCIDPN(N), (KLBNP1(N,L), L=1,6), N=1, NUCPN)
663:      7340 format(/3x,' NAME ',8x,
664:      & ' ATW TLAB ELOW EHI ',
665:      & ' ZA MATNO'
666:      & /' ----- ' ,6('----- '))
667:      7350 format(3X,A,1P,5E12.5,0P,F12.0)
668: C
669: C ..... debug process
670:      if ( JDEBG(1).ne.0 ) then
671:        write(IPR,*) ' === KLBNP2(NUCPN,NMTPN,1-12) ====='
672:        do N = 1, NUCPN
673:          write(IPR,' (/1x,a,a,a)')
674:          & ' ==< ', NCIDPN(N) / (ICLEN(NCIDPN(N))), ' >== '
675:          write(IPR,7400)
676:          write(IPR,7410) (K, (KLBNP2(N,K,L), L=1,12), K=1, NMTPN)
677:        end do
678:      7400 format(/1X,' MT MTINIG MTPAR ISTMT IENDMT LCTMT ',
679:      & ' NEANG NK2G NE2G MT2G LSTP4 LSTP4 ',
680:      & ' LST2G'
681:      & /' --- ' ,12('----- '))
682:      7410 format(1X,I4,12I8)
683:      end if
684: C
685: C -----
686: C set the table length of densities for photo-nuclear nuclides
687: C -----
688: C
689:      NUCPNA = 0
690:      NUCPNB = 0
691:      do M = 1, NMAT
692:        do N = 1, MNUC(M)
693:          L = LPDEN(M) + N - 1
694:          NUCPNA = NUCPNA + INMNP(NINUCT(L))
695:        end do
696:        NUCPNB = max( NUCPNB, MNUC(M) )
697:      end do
698: C
699:      return
700: C
701: C *****
702: C
703:      entry MTDATPN(NUCPNA,NUCPNB,NUCPN, NMAT, MLEN, MNUC, LPDEN,
704:      & INUCT, DENST, NUC, NUCPNI,NCIDP, NCIDPN,MNUCPN,
705:      & NNUCPN,LPDNP,NMSTPN,LPIDPN,IZTBP,NMNEUTPN)
706: C=MVP>=====
707: C PURPOSE : get number density table for photo-nuclear data.
708: C CALLED IN: CTXSIN
709: C CALLS : CNTERR
710: C=====
711: C
712: C MNUCPN(NMAT+1) : starting position of photo-nuclear nuclides
713: C included in each material
714: C NNUCPN(NUCPNB,NMAT): number of photo-nuclear nuclides for neutron
715: C ID in each material

```

```

716: C LPDNP(NUCPNB,NMAT): starting position of photo-nuclear nuclides
717: C for neutron ID in each material
718: C DNSTPN(NUCPNA) : number density table for photo-nuclear nuclides
719: C LPIDPN(NUCPNA) : reference pointer of photo-nuclear nuclide ID in
720: C NCIDPN
721: C IZTBP(NUCPNA) : atomic number of photo-nuclear nuclides
722: C MNEUTPN(NUCPNA): reference pointer of neutron ID corresponding to
723: C photo-nuclear nuclides
724: C
725: C -----
726: C
727:      N8 = 1
728:      N9 = 0
729:      MNUCPN(1) = 1
730: C
731: C ..... loop over materials
732:      do M = 1, NMAT
733:        MM = 0
734: C
735: C ..... loop over nuclides in a material
736:        do N = 1, MNUC(M)
737:          L = LPDEN(M) + N - 1
738:          NCIDPN0 = NCIDP(INUCT(L))
739:          NAA0 = INMNP(NINUCT(L))
740:          NAA1 = 0
741:          MM = MM + NAA0
742:          LPN = ICLEN(NCIDPN0)
743:          if ( LPN.ge.9 ) then
744:            IVERF = 31
745:          else
746:            IVERF = 30
747:          end if
748:          if ( NAA0.gt.0 ) then
749:            N99 = LNMNP(NINUCT(L)) - 1
750:            do NN = 1, NAA0
751:              do IK = 1, NUCPN
752:                if ( NCIDPN9(N99+NN).eq.NCIDPN(IK) ) go to 200
753:              end do
754:              LPIDPN(N9+NN) = IK
755:              IZTBP(N9+NN) = IZZZPN(N99+NN) / 1000
756:              MNEUTPN(N9+NN) = INUCT(L)
757:            end do
758: C ..... natural element
759:            if ( NCIDPN0(3:3).eq.'N'.or.
760:            & ( IVERF.eq.31.and.NCIDPN0(3:5).eq.'000' ) ) then
761:              N90 = N9
762:              N91 = N9 + NAA0
763:              do NN = 1, NAA0
764:                N9 = N9 + 1
765:                IZ0 = IZZZPN(N99+NN) / 1000
766:                IK = IZZZPN(N99+NN) - IZ0 * 1000
767:                DNSTPN(N9) = DENST(L) * TISOTP(3,IK,IZ0)
768:                if ( N.gt.1 ) then
769:                  ! combine with duplicated nucl
770:                  do NN1 = MNUCPN(M), N90
771:                    if ( LPIDPN(N9).eq.LPIDPN(NN1) ) then
772:                      DNSTPN(NN1) = DNSTPN(NN1) + DNSTPN(N9)
773:                      if ( NN.lt.NAA0 ) then
774:                        do I = N9+1, N91
775:                          LPIDPN(I-1) = LPIDPN(I)
776:                          IZTBP(I-1) = IZTBP(I)
777:                          MNEUTPN(I-1) = MNEUTPN(I)
778:                        end do
779:                        N91 = N91 - 1
780:                      end if

```

src/mvp/xsecpn.f

```
780:                N9      = N9 - 1
781:                NAA1     = NAA1 + 1
782:                go to 210
783:            end if
784:        end do
785:    end if
786:    210      continue
787:  end do
788:  C ..... isotope
789:      else
790:          N9      = N9 + 1
791:          DNSTPN(N9) = DENST(L)
792:          if ( N.gt.1 ) then      ! combine with duplicated nucl
ide
793:              do NN1 = MNUCPN(M), N9-1
794:                  if ( LPIDPN(N9).eq.LPIDPN(NN1) ) then
795:                      DNSTPN(NN1) = DNSTPN(NN1) + DNSTPN(N9)
796:                      N9      = N9 - 1
797:                      NAA1     = NAA1 + 1
798:                      go to 220
799:                  end if
800:              end do
801:              220      continue
802:          end if
803:      end if
804:  end if
805:      NNUCPN(N,M) = NAA0 - NAA1
806:      LPDNPN(N,M) = N8
807:      N8      = N9 + 1
808:      MM      = MM - NAA1
809:  end do
810:      MNUCPN(M+1) = MNUCPN(M) + MM
811:  end do
812:  C
813:  return
814:  end
```

```

real CXP1(MCX)
real CXP2(MCX)

character MATT(NUC)*136, NUCID(NUC)*16

integer JDERG(*)

integer KLIB1(NUC,30)

character FILE*256

include '../shared/INC/_IOUNIT'
include 'INC/_IOUNIT2'

-----
AD #4 AND #5 RECORD
-----

1999.2.19
write(*,*) '===== xsecpp.f 1999.2.19 ====='
call LABEL( IPR, 'CROSS SECTION LIBRARY FOR PERTURBATION-CALC.' )
LST = 1
LST0 = 0
Do 100 N = 1, NUC
  LST0 = LST0 + KLIB1(N,3)
  LNV = INDEX(NUCID(N),' ') - 1
  write(*,*) '== NUCLEIDE : ',NUCID(N)
  write(*,*) 'LST:',LST,' LST0:',LST0
  call faloc( IXS, NUCID(N), FILE, IERR )
  if( IERR.ne.0 ) then
    call CNTERR( 'FATAL' )
    goto 100
  endif
  read(IXS)                                ! skip RECORD #1
  read(IXS)                                ! skip RECORD #2
  read(IXS)                                ! skip RECORD #3
  read(IXS,iostat=ioerr) (CXP1(I),I=LST,LST0) ! RECORD #4
  if( ioerr.ne.0 ) then
    write(IPR,1000) NUCID(N)
    call CNTERR( 'FATAL' )
  endif
  read(IXS,iostat=ioerr) (CXP2(I),I=LST,LST0) ! RECORD #5
  if( ioerr.ne.0 ) then
    write(IPR,1000) NUCID(N)(:LNV)
    call CNTERR( 'FATAL' )
  endif
100 continue

```

```
65:      LST  = LST0 + 1
```

src/mvp/xsfile.f

```
1:      SUBROUTINE XSFILE
2: C=====
3: C  PURPOSE :  INPUT OF ALREADY PROCESSED CROSS SECTION DATA
4: C  CALLED IN: CTXSIN
5: C=====
6:      RETURN
7:      END
```

SAFE

src/mvp/zzbank.f

```
1:      subroutine ZZBANK( KDBNK, MDBNK, KIBNK, MIBNK,  
2:      &                  KDFBK, MDFBK, KIFBK, MIFBK )  
3: C=====
```

4: C PURPOSE: Optional particle bank reservation for custom use.
5: C CALLED IN: INTRO2
6: C CALLS:

7: C-----

8: C About optional data bank arrays:
9: C

10: C Particle attributes are stored in arrays XXX(1:NBANK) (X-coordinates)
11: C EEE(1:NBANK) (energy) etc. in MVP/GMVP and they are used for fixed
12: C purpose.
13: C MVP/GMVP version2.x and newer has arrays called "optinal bank data"
14: C used for any purpose and their data size may vary depending on
15: C problems to be solved or used in custom version by user.
16: C

17: C DBNK(1:NBANK,*) : double word bank array.
18: C IBNK(1:NBANK,*) : integer bank array.
19: C DFBK(1:NFBNK0,*) : double word bank array for fission neutron in
20: C eigenvalue calculation mode.
21: C IFBK(1:NFBNK0,*) : integer bank array for fission neutron in
22: C eigenvalue calculation mode.
23: C

24: C Array KDBNK(*) is used to point specified optinal attributes in
25: C array DBNK like DBNK(*,KDBNK(1)). Some elements of KDBNK(*) have
26: C meanings defined by developer as:
27: C

28: C DBNK(*,KDBNK(1)) is particle weight on birth,
29: C DBNK(*,KDBNK(2)) is time of birth,
30: C ...
31: C

32: C and elements KDBNK(11) to KDBNK(16) are for custom use and users can
33: C give any attributes in their custom version.
34: C

35: C KIBNK, KDFBK and KIFBK works in similar way for arrays IBNK, DFBK and
36: C IFBK respectively.
37: C

38: C-----

39: C

40: C In this routine, users must set size of blocks of NBANK/NFBNK0 words
41: C used from optional bank arrays. Conversion to second array index for
42: C DBNK etc. is performed automatically in subroutine INTRO2.
43: C

44: C For example, if a user want to use attribute 11 of DBNK with size of
45: C 3*NBANK words (double) and attribute 13 of IFBK with size of
46: C NBANK* (NEST + 1) words (single), necessary coding in this routine is
47: C as follows:
48: C

49: C KDBNK(11) = 3
50: C KIFBK(13) = NEST + 1
51: C

52: C Index 11 to 16 of KDBNK, KIBNK, KDFBK and KIFBK are provided for
53: C custom use.
54: C

55: C=====

56: C include './shared/INC/_IUNIT'
57: C include 'INC/_SIZES'
58: C include 'INC/_FLAGS'
59: C

60: C integer KDBNK(0:MDBNK)
61: C integer KIBNK(0:MIBNK)
62: C

63: C integer KDFBK(0:MDFBK)
64: C integer KIFBK(0:MIFBK)
65: C

```
66: C-----  
67: C KDBNK(11) = 3  
68: C KIFBK(13) = NEST + 1  
69: C  
70: C return  
71: C end
```

src/mvp/zzcustom.f

```
1:      subroutine CUSTOM( IOIN1, IPRI,  A, DA, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: input data for a custom version.
4: C       Users should add procedure to input/process data
5: C       related to customization.
6: C
7: C   Data block for custom version is assumed as follows.
8: C
9: C       $CUSTOM
10: C
11: C       custom data
12: C
13: C       $END CUSTOM
14: C
15: C   ( The string "$CUSTOM" is already input in "INTRO" routine.
16: C     Data are ignored in non-custom version )
17: C
18: C CALLED IN: INTRO
19: C----< comments for custom version >-----
20: C
21: C HISTORY:
22: C UPDATE:
23: C
24: C-----
25: C
26: C   arguments ( i = input, o = output, c = constant, w = work )
27: C
28: C c   IOIN1 : current input I/O-unit
29: C c   IPRI  : printout I/O-unit
30: C io  A(*)  : dynamic memory array area (real)
31: C io  DA(*) : dynamic memory array area (double precision)
32: C io  CHA(*) : dynamic memory array area (4 byte character strings)
33: C
34: C*io last : last position of used area in array A(*) of common /ARRAY/
35: C
36: C-----
37: C
38: C   include '../shared/INC/_ARRAY'
39: C
40: C   include 'INC/_KPIDS'
41: C   include 'INC/_NGPS'
42: C
43: C   include 'INC/_FLAGS'
44: C   include '../shared/INC/_SIZES'
45: C   include 'INC/_SIZES2'
46: C   include 'INC/_CXSEC'
47: C   include 'INC/_PXSEC'
48: C   include 'INC/_XBANK'
49: C   include 'INC/_SBANK'
50: C   include 'INC/_STACK'
51: C   include 'INC/_XWORK'
52: C   include '../shared/INC/_PGEOM'
53: C   include '../shared/INC/_PVRED'
54: C   include 'INC/_PSOUR'
55: C   include '../shared/INC/_PTALY0'
56: C   include 'INC/_PTALY'
57: C   include 'INC/_PTALY2'
58: C   include '../shared/INC/_PTLSP'
59: C   include '../shared/INC/_LISTON'
60: C   include '../shared/INC/_WORDL'
61: C
62: C-----
63: C
64: C   By default skip input data until "$END CUSTOM"
65: C
```

```
66: C   To customize , remove the following lines and add your own procedure.
67: C
68: C   Hints for customization:
69: C
70: C   You can use some utility routines for MVP/GMVP type free form inputs.
71: C   * FREADS routine to get data-name.
72: C   * I4READ, R4READ, R8READ & CHREAD to read data elements of
73: C     integer, real, double precision and string respectively.
74: C
75: C-----
76: C   Description of utility routines:
77: C   ( arguments : i = input, o = output, c =constant, w = work )
78: C
79: C-----
80: C   SUBROUTINE FREADS( VNAME, NLEN, IEND )
81: C-----
82: C
83: C       Get next data-name from current input file.
84: C
85: C       o VNAME (character*(*)): data-name found.
86: C       o NLEN (integer): length of data-name found.
87: C       o IEND (integer): end of file encountered if not zero.
88: C
89: C-----
90: C   subroutine I4READ( VNAME,IA, NA,NB,IEND )
91: C   subroutine R4READ( VNAME,RA, NA,NB,IEND )
92: C   subroutine R8READ( VNAME,RA8,NA,NB,IEND )
93: C-----
94: C
95: C       Get numerical data of integer, real and double real type.
96: C       This routine must be called after "FREADS" succeeded in getting
97: C       a data-name.
98: C
99: C       i VNAME (character*(*)): data-name.
100: C       o IA(*),RA(*),RA8(*) : arrays to put data
101: C                             (integer, real , duple real)
102: C       o NA (integer)       : number of data read in.
103: C       i NB (integer)       : if NB > 0, NB is the number of data
104: C                             expected to be input.
105: C                             if NB < 0, no expected number of data
106: C                             but data over abs(NB)'th data
107: C                             are skipped.
108: C                             NB = 0 also means no expected number of
109: C                             data but not recommended to use.
110: C       o IEND (integer)     : end of file encountered if not zero.
111: C
112: C-----
113: C   subroutine CHREAD( VNAME, CGET, CHAR, NLEN, ITERM, IEND )
114: C-----
115: C
116: C       Get a string data from current input file. This routine gets
117: C       only one non-blank string per call, so you may need to call
118: C       this routine repeatedly until a termination point is reached.
119: C
120: C       i VNAME (character*(*) ) : data-name.
121: C       o CGET (character*(*) ) : a string input.
122: C       i CHAR (character*(*) ) : list of termination characters
123: C                             other than blank. The most popular one is
124: C                             ')' which terminates data sequence such as;
125: C                             data-name( str1 str2 ... strn )
126: C       o NLEN (integer)         : length of string read in.
127: C       o ITERM (integer)        : If non-zero, encountered a termination
128: C                             charcater CHAR(ITERM:ITERM).
129: C       o IEND (integer)         : end of file encountered if not zero.
130: C
```

src/mvp/zzcustom.f

```
131: C=====
132: C comment:  subroutine GETLIN needs print out I/O units as argument
133: C          after MVP/GMVP version 2.0.
134: C=====
135: C
136:       real A(*)
137:       real*8 DA(*)
138:       character*4 CHA(*)
139: C
140: C ... local data ...
141: C
142:       character VNAME*32
143:       character LINE*72
144: C
145:       call HEADER( IPRI, 'INPUT FOR CUSTOMIZED VERSION' )
146:       write(IPRI, '/lx, ' ' !!! This version ignores "$CUSTOM" block.' )
147:       write(IPRI, '/lx, ' ' ( parameter definition is effective )' )
148:       call CNTERR( 'WARNING' )
149: C
150: C ... "GETLIN" gets a non-comment line from input data.
151: C (parameter line ( % name=value, name=value, ...) is processed.
152: C   If you want to ignore parameter definition also, use "CPLIN"
153: C   routine which has the same arguments as "GETLIN" )
154: C
155: 100 call GETLIN( IOIN1, IPRI, LINE, NCHAR, IEND )
156: C
157:       if ( IEND.ne.0 ) then
158:         write(IPRI,*) 'XXX Unexpected end of input-data ',
159:         &             'in custom data block.'
160:         write(IPRI,*) ' Probably missing "$END CUSTOM" line.'
161:         call CNTERR( 'FATAL' )
162:         call PRSTOP( 0, 'custom data input error.' )
163:         stop 888
164:       end if
165: C
166:       if ( LINE(1:11).ne.'$END CUSTOM' ) go to 100
167: C
168: C
169: C===== END OF DEFAULT PROCEDURE =====
170: C
171: C
172:       return
173:       end
```


src/mvp/zzinit.f

```
1:      subroutine ZZINIT
2: C=====
3: C  PURPOSE: initialization for custom version.
4: C  CALLED IN: MAIN
5: C  CALLS:
6: C-----
7: C This routine is called in main routine before any data input or
8: C command argument processing. It is provided for initialization for
9: C custom version by user.
10: C
11: C Users can set default value of user-added global data or
12: C printout message etc..
13: C
14: C-----
15: C      include '../shared/INC/_IUNIT'
16: C
17:      return
18:      end
```

src/mvp/zztalo.f

```
1:      subroutine ZZTALO( WSUM1, IPRI, A, DA, H, DH, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: tally output etc. after random walk in custom version
4: C CALLED IN: CADENZ
5: C-----< comments for custom version >-----
6: C
7: C HISTORY:
8: C UPDATE:
9: C
10: C-----
11: C
12: C   arguments ( i = input, o = output, c = constant, w = work )
13: C
14: C i WSUM1 (REAL*8) : sum of particle weights. May have the same address
15: C                   with WSUM in COMMON area.
16: C io A(*) : dynamic memory array (task shared,real)
17: C io DA(*) : dynamic memory array (task shared,double precision)
18: C io H(*) : dynamic memory array (task local,real)
19: C io DH(*) : dynamic memory array (task local,double precision)
20: C io CHA(*) : dynamic memory array area (4 byte character strings)
21: C
22: C-----
23: C
24: C   include '../shared/INC/_ARRAY'
25: C
26: C   include 'INC/_KPIDS'
27: C   include 'INC/_NGPS'
28: C
29: C   include 'INC/_FLAGS'
30: C   include '../shared/INC/_SIZES'
31: C   include 'INC/_SIZES2'
32: C   include 'INC/_CXSEC'
33: C   include 'INC/_PXSEC'
34: C   include 'INC/_XBANK'
35: C   include 'INC/_SBANK'
36: C   include 'INC/_STACK'
37: C   include 'INC/_XWORK'
38: C   include '../shared/INC/_PGEOM'
39: C   include '../shared/INC/_PVRED'
40: C   include 'INC/_PSOUR'
41: C   include '../shared/INC/_PTALY0'
42: C   include 'INC/_PTALY'
43: C   include 'INC/_PTALY2'
44: C   include '../shared/INC/_PTLSP'
45: C   include '../shared/INC/_WORDL'
46: C-----
47:      real*8 WSUM1
48:      real A(*)
49:      real*8 DA(*)
50:      real H(*)
51:      real*8 DH(*)
52:      character*4 CHA(*)
53: C
54:      return
55: end
```

src/mvp/zztals.f

```
1:      subroutine ZZTALS(NGENE0, WSUMB,  A, DA, H, DH, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: take tally sum etc. after each batch in custom version
4: C CALLED IN: TALSUM
5: C----< comments for custom version >-----
6: C
7: C HISTORY:
8: C UPDATE:
9: C
10: C-----
11: C
12: C  arguments ( i = input, o = output, c = constant, w = work )
13: C
14: C i  NGENE0 : number of particles generated in current batch
15: C i  WSUMB  : total weight of particles generated in current batch
16: C io A(*)   : dynamic memory array (task shared,real)
17: C io DA(*)  : dynamic memory array (task shared,double precision)
18: C io H(*)   : dynamic memory array (task local,real)
19: C io DH(*)  : dynamic memory array (task local,double precision)
20: C io CHA(*) : dynamic memory array area (4 byte character strings)
21: C
22: C-----
23: C
24: C      include 'INC/_KPIDS'
25: C      include 'INC/_NGPS'
26: C
27: C      include 'INC/_FLAGS'
28: C      include '../shared/INC/_SIZES'
29: C      include 'INC/_SIZES2'
30: C      include 'INC/_CXSEC'
31: C      include 'INC/_PXSEC'
32: C      include 'INC/_XBANK'
33: C      include 'INC/_SBANK'
34: C      include 'INC/_STACK'
35: C      include 'INC/_XWORK'
36: C      include '../shared/INC/_PGEOM'
37: C      include '../shared/INC/_PVRED'
38: C      include 'INC/_PSOUR'
39: C      include '../shared/INC/_PTALY0'
40: C      include 'INC/_PTALY'
41: C      include 'INC/_PTALY2'
42: C      include '../shared/INC/_PTLSP'
43: C      include '../shared/INC/_WORDL'
44: C-----
45:      real A(*)
46:      real*8 DA(*)
47:      real H(*)
48:      real*8 DH(*)
49:      real*8 WSUMB
50:      character*4 CHA(*)
51: C
52:      return
53:      end
```

src/mvp/INC/_ATPOOL

```
1: C
2:     character*128 ATPPOOL
3:     common /XSCSTR/ ATPPOOL
4: C
5: C ATPPOOL C128 File path name of a directory in which processed cross
6: C     section data by ART module is stored. (If non-blank)
7: C
```

SAFE

src/mvp/INC/_CXSEC

```

1: C*
2: C*.... POINTERS TO MATERIAL & NULCILDE DATA (CROSS SECTION).
3: C*.... THRESHOLD PARAMETERS .....
4: C*
5: C*   NOV 20 1991 : MODIFICATION FOR PHOTON PRODUCTION & PHOTON
6: C*
7: C*   16 FEB 1994 : MODIFICATION FOR ELECTRON
8: C*   28 Jun 1995 : add array DSMAT
9: C*   12 May 1998 : add array TEMPN, NCIDI
10: C*   17 Apr 2000 : add array ETOPX and EBOTX
11: C*   21 Jun 2000 : add array KCREST ( from LKCRSI )
12: C*   14 Mar 2007 : modification for photonuclear
13: C*   20 Dec 2013 : add array INTDS.
14: C*
15: Common /CXSEC/  ETOPX(KPLIM), EBOTX(KPLIM),
16: N              ETOP, EBOT, ETHMAX, AMLIM, EWCUT, ETOPL,
17: N              EBOTL, ESABL, NMT, NNK,
18: N              LIDMAT, LMNUC, LINUCT, LDENST, LLPDEN, LDSMAT
19: N              NUC,
20: N              LNUCID, LNCIDI, LTEMPN, LMATT, LKLIB1, LXLIB1,
21: N              LKLIB2, LKLB2S, LXLIB2, LCX, MCX,
22: c+pert          LCXP1, LCXP2,
23: &
24: c-pert
25: P              NMTP, NPATOM, LMPATM, LLPDNP, LNATMT, LIATMT,
26: P              LDNST, LCXP, MCRP, LMATP, LKLB1, LXLBP1,
27: P              LKLB2, LXLBP2, ETOPP, EBOTP, ETOPLP, EBOTLP,
28: E              LKLB1, LKLB2, LEEL, LRKT, LPBR, LKBA,
29: E              LPBT, LEBT, LRNG, LEBTP, LEBTP, LFME,
30: E              LWWAG, LEEDG, LEEK, LCXE, MCXE, NEE,
31: E              MTOP, ETOPE, EBOTE, ETOPL, EBOTL,
32: c##<2007/03/14:PN3:
33: c## S          LICRES, NICRES, LCRES, MCRES, LKCRS, LKCRSI
34: S          LICRES, NICRES, LCRES, MCRES, LKCRS, LKCRSI,
35: &             LKLBPN1, LKLBPN1, LKLBPN2, LKLBPN2, LCXPN, MCXPN,
36: &             LMATPN, LNCIDP, LNCIDPN, NUCPN1, NUCPN, NUCPNA,
37: &             NUCPNB, NMTPN, LMNUCPN, LNUCPN, LLPDNP, LDNSTPN,
38: &             LLPIDPN, LIZTBNP, LMNEUTPN, ETOPLPN, EBOTLPN,
39: c##>
40: &             LINTDS
41: C*
42:   real*8  ETOPX, EBOTX
43:   real    ETOP, EBOT, ETHMAX, AMLIM, EWCUT, ETOPL, EBOTL,
44: N        ESABL,
45: P        ETOPP, EBOTP, ETOPLP, EBOTLP,
46: c##<2007/03/14:PN3:
47: c## E      ETOPE, EBOTE, ETOPL, EBOTL
48: E      ETOPE, EBOTE, ETOPL, EBOTL,
49: &        ETOPLPN, EBOTLPN
50: c##>
51: C*
52: C*
53: C*
54: C ETOP      R4   Upper energy limit (eV) for neutron.
55: C EBOT      R4   Lower energy limit (eV) for neutron.
56: C ETHMAX    R4   Upper energy limit for neutron thermal scattering (eV).
57: C AMLIM     R4   Upper mass limit of free-gas treatment for neutron (eV).
58: C EWCUT     R4   Threshold of analog absorption for neutron (eV).
59: C           $   E < EWCUT : analog absorption
60: C           $   E >= EWCUT : weight reduction
61: C ETOPL     R4   Upper energy limit of neutron x-sec (minimum of EHI).
62: C EBOTL     R4   Lower energy limit of neutron x-sec (maximum of ELOW).
63: C ESABL     R4   Upper energy limit for S(A,B) (maximum of EAB).
64: C*
65: C ETOPX(KPLIM) R8 Upper energy limit (eV) for each particle species.

```

```

66: C EBOTX(KPLIM) R8 Lower energy limit (eV) for each particle species.
67: C*
68: C**** MATERIAL SPECIFICATION DATA *****
69: C*
70: C IDMAT(NMAT) I4 Material ID numbers.
71: C MNUC(NMAT) I4 Number of nuclides composing each material.
72: C INUCT(*) I4 Nuclide number table.
73: C DENST(*) R4 Number density table.
74: C LPDEN(NMAT+1) I4 Pointers to INUCT and DENST (data of I'th material$
75: C $ starts from LPDEN(I) and end at LPDEN(I+1)-1).
76: C DSMAT(NMAT) R4 Sum of nuclide number densities in each material.
77: C Used to judge whether this material can be replaced
78: C with void material or not.
79: C*
80: C**** NUCLIDE DATA *****
81: C*
82: C NUC I4 Number of nuclides used in problem.
83: C NUCID(NUC) CH16 nuclide ID's (16 characters each).
84: C NCIDI(NUC) CH16 nuclide ID's as input. Used to search library in
85: C index file.
86: C TEMPN(NUC) R8 Nuclide temperature in Kelvin.
87: C Used for run time doppler broadning of neutron library
88: C (ART module).
89: C Negative value means no doppler broadning.
90: C*
91: C**** CROSS SECTION DATA FOR EACH NUCLIDES *****
92: C*
93: c##<2007/03/14:PN3:
94: c##C MATT(NUC) CH136 Mat-ID / date / comment (neutron x-sec)
95: C MATT(NUC) CH136 (neutron x-sec)
96: C [until version 2] Mat-ID*8 / date*8 / comment*120
97: C [from version 3] Mat-ID*16 / comment*112 / date*8
98: c##>
99: C*
100: c##<2007/03/14:PN3:
101: c##C KLIB1(NUC,27) I4 Record #1 of neutron x-sec (specification record).
102: C KLIB1(NUC,31) I4 Record #1 of neutron x-sec (specification record).
103: c##>
104: C
105: C $ 1: Starting position of data for a nuclide
106: c##<2007/03/14:PN3:
107: c##C $ 2: NPTS 3: NDATA 4: NUNR 5: NUNR2
108: c##C $ 6: NLEV 7: NBINA 8: NBINE 9: NNU 10: NMT
109: c##C $ 11: NNK 12: IGFLG 13: LFI 14: LNU 15: IOTHERM
110: c##C $ 16: ISTU 17: IENDU 18: LUNR
111: c##C $ 19: LSNU (start position of nu-value)
112: c##C $ 20: NBINA1 = NBINA + 1 21: NBINE1 = NBINE + 1
113: c##C $ 22: NNUD 23: LNUD 24: NNF 25: LSNUD
114: c##C $ 26: NCAP 27: NCAPG
115: C $ 2: NPTS 3: NDATA 4: NUNR 5: NUNR2
116: C $ 6: NLEV 7: NBINA 8: NBINE 9: NNU 10: NMT
117: C $ 11: NNK 12: IGFLG 13: LFI 14: LNU 15: IOTHERM
118: C $ 16: ISTU 17: IENDU 18: LUNR 19: LSNU 20: NBINA1
119: C $ 21: NBINE1 22: NNUD 23: LNUD 24: NNF 25: LSNUD
120: C $ 26: NCAP 27: NCAPG 28: LF6 29: LSTF6 30: LLSSF
121: C $ 31: LASYM
122: c##>
123: C
124: C ( NMT = 120, NNK = 10 : FIXED )
125: C*
126: c##<2007/03/14:PN3:
127: c##C XLIB1(NUC,9) R4 Nuclide parameters (neuron x-sec)
128: c##C
129: c##C $ 1: ATW 2: TLAB 3: ETHEM 4: ESAB 5: ELOW
130: c##C $ 6: EHI 7: 1/(ATW+1)**2 8: 1/(ATW+1) 9: ATW/(K*TLAB)

```

src/mvp/INC/_CXSEC

```

131: C XLIB1(NUC,11) R4 Nuclide parameters (neuron x-sec)
132: C
133: C $ 1: ATW 2: TLAB 3: ETHEM
134: C $ 4: ESAB 5: ELOW 6: EHI
135: C $ 7: 1/(ATW+1)**2 8: 1/(ATW+1) 9: ATW/(K*TLAB)
136: C $ 10: TEFF 11: ATW/(K*TEFF)
137: c##>
138: C*
139: c##<2007/03/14:PN3:
140: c##C KLIB2(NUC,NMT,21) I4 Pointers to #2 record data (1)$
141: c##C $ (neuron x-sec)
142: C KLIB2(NUC,NMT,21) I4 Pointers to #2 record data (1) (neuron x-sec)
143: c##>
144: C
145: C $ 1: MTINFO 2: MTPAR 3: ISTMT 4: IENDMT 5: NEUMT
146: C $ 6: LCTMT 7: NEANG 8: NKF5 9: NEF5 10: MTHR
147: c##<2007/03/14:PN3:
148: c##C $ 11: LPS 12: LANG 13: LED
149: c##C $ 14: LSTGAM 15: MTTOG 16: NEG 17: NEGP 18: NKG
150: c##C $ 19: LCTMTG 20: MTCAP 21: MTCAPG
151: C $ 11: LPS 12: LANG 13: LED 14: LSTGAM 15: MTTOG
152: C $ 16: NEG 17: NEGP 18: NKG 19: LCTMTG 20: MTCAP
153: C $ 21: MTCAPG
154: c##>
155: C*
156: C KLB2S(NUC,NNK,NMT,3) I4 Pointers to #2 record data (2) (neuron x-sec)
157: C
158: C $ 1: NEF5S 2: LFF5S 3: LSTF5S
159: C*
160: C XLIB2(NUC,NMT,4) R4 Q-value & photon generation threshold
161: C
162: C $ 1: QVAL 2: (ATW+1)/ATW*QVAL
163: C $ 3: EG1L 4: EG2L
164: C*
165: C*
166: C CX(MCX) R4 Neutron cross section data.
167: C MCX I4 Length of CX array.
168: C*
169: C*
170: C* ----- FOR PHOTON REACTION ----- NOV 20 1991
171: C*
172: C*
173: C* << MATERIAL SPECIFICATIONS >>
174: C*
175: C NPATOM I4 Number of atomic elements used in photon problem.
176: C MPATM(NMAT) I4 Number of atomic elements included in each material$
177: C $ in photon problem.
178: C LPDNP(NMAT+1) I4 Position pointer for IATMT & DNSTP tables.
179: C $ Data of I'th material starts from LPDNP(I) and ends$
180: C $ at LPDNP(I+1)-1.
181: C NATMT(NPATOM) I4 Atomic number list used in photon problem
182: C IATMT(MLEN) I4 Atomic number table (position in NATMT(NMAT))$
183: C $ for each material in photon problem.
184: C DNSTP(MLEN) R4 Number density table for each material in photon$
185: C $ problem.
186: C*
187: C CXP(MCXP) R4 Photon cross section data
188: C MCXP I4 Length of CXP array
189: C*
190: C* *** CROSS SECTION DATA FOR EACH ATOMIC ELEMENTS *****
191: C*
192: C NMTP I4 Number of photon reactions
193: C*
194: C MATTP(NPATOM) CH136 Mat-ID / DATE / COMMENT (photon x-sec)
195: C*

```

```

196: C KLBP1(NPATOM,20) I4 Record #1 (specification record)$
197: C $ of photon x-sec.
198: C
199: C $ 1: Starting position of data for an atomic element
200: C $ 2: NPTS 3: NDATA 4: NMTP 5: NSUBS 6: NSF
201: C $ 7: NFF 8: NFEE
202: C $ 9: LSTS 10: LSTFF 11: NPEF 12: LSPEF
203: C $ (13-20 dummy for future use)
204: C $ ( NMTP = 60 : FIXED )
205: C*
206: C XLBP1(NPATOM,10) R4 Atomic parameters (photon x-sec)
207: C
208: C $ 1: AWR 2: Z 3: M0*C**2 4: AK 5: R0
209: C $ 6: PAI*R0*2 7: ELOW 8: EHI
210: C*
211: C KLBP2(NPATOM,NMTP,6) I4 Pointers to #2 record data (1)$
212: C $ of photon x-sec.
213: C
214: C $ 1: IDINFO 2: MT 3: ISTID 4: IENDID 5: LSTF3
215: C $ 6: INT
216: C*
217: C XLBP2(NPATOM,NMTP,2) I4 Pointers to #2 record data (2)$
218: C $ of photon x-sec.
219: C
220: C $ 1: EPE 2: EFL
221: C*
222: C ETOPP R4 Upper energy limit specified by user (photon).
223: C EBOTP R4 Lower energy limit specified by user (photon).
224: C*
225: C ETOLPL R4 Upper energy limit in photon library (photon).
226: C EBOTLP R4 Lower energy limit in photon library (photon).
227: C*
228: C*
229: C* ----- FOR ELECTRON REACTION ----- 16 FEB 1994
230: C*
231: C*
232: C* << MATERIAL SPECIFICATIONS >>
233: C*
234: C FME(MLEN) R4 Atom fraction table for each material in electron
235: C problem
236: C*
237: C CXE(MCXE) R4 Electron cross section data
238: C MCXE I4 Length of CXE array
239: C*
240: C* << CROSS SECTION DATA FOR ELECTRON >>
241: C*
242: C KLBE1(NPATOM,20) I4 Record #1 (specification record) of electron
243: C cross sections
244: C
245: C 1: NTDATA Length of record #2
246: C 2: NRAD Length of TR and RR tables (usually 49)
247: C 3: NMOT Length of ER and RS tables (usually 18)
248: C 4: NCOR Length of TC and CP tables (usually 32)
249: C 5: NHFL Length of TH and FL tables (usually 39)
250: C 6: NEE0 Number of energy grids for electron cross section
251: C data (usually 134)
252: C 7: MTOPO Number of bremsstrahlung photon/electron (k/t)
253: C ratios (usually 49)
254: C 8: NEK Number of energy grids for EK table (usually 9)
255: C 9: NPL Number of the legendre series for GK table
256: C (USUALLY 240)
257: C 10: LSTC Starting position of data for an atomic element
258: C 11: LSTTR Starting position of TR table (LSTC+4)
259: C 12: LSTER Starting position of ER table (LSTTR+NRAD*2)
260: C 13: LSTER Starting position of EK table (LSTER+NMOT*6)

```

src/mvp/INC/_CXSEC

```

261: C      14: LSTTC Starting position of TC table (LSTER+NEK*14)
262: C      15: LSTTH Starting position of TH table (LSTTC+NCOR*2)
263: C      16: LSTEEL Starting position of energy grids for electron
264: C cross section data, in eV (LSTTH+NHFL*2)
265: C      17: LSTPBR Starting position of bremsstrahlung cross sections
266: C (LSTEEL+NEE)
267: C      18: LSTRKT Starting position of k/t ratios (LSTPBR+NEE)
268: C      19: LSTEB A Starting position of bremsstrahlung cumulative
269: C probabilities, EBA (LSTRKT+MTOP)
270: C      20: LSTGK Starting position of GK table (LSTEB A+NEE*MTOP)
271: C*
272: C XLBPI(NPATOM,6) R4 Atomic parameters
273: C*
274: C      1: AWR atomic weight
275: C      2: Z atomic number
276: C      3: Z13 Z**(1/3)
277: C      4: ZLN log(Z)
278: C      5: CCL bremsstrahlung coulomb correction factor
279: C      6: TZ processing temperature of cross sections, in eV
280: C*
281: C NEE I4 Number of energy grids for electron cross section
282: C data
283: C MTOP I4 Number of bremsstrahlung photon/electron ratios
284: C*
285: C EEL(NEE) R4 Electron energy grids, in eV
286: C RKT(MTOP) R4 Bremsstrahlung photon/electron (k/t) ratios
287: C PBR(NEE,NMAT) R4 Bremsstrahlung cross sections by materials
288: C PBT(NEE,NMAT) R4 Bremsstrahlung yields for thick-target bremsstrah-
289: C lung (TTB) approximation by materials
290: C EBA(MTOP,NEE,NMAT)
291: C R4 Bremsstrahlung cumulative probabilities by
292: C materials
293: C EBT(MTOP,NEE,NMAT)
294: C R4 Bremsstrahlung cumulative probabilities for TTB by
295: C materials
296: C EBTP(MTOP-1,NEE,NMAT)
297: C R4 Bremsstrahlung partial probabilities for TTB by
298: C materials
299: C IEFTP(2*(MTOP-1),NEE,NMAT)
300: C I4 Conditiinal discrete sampling table for bremsstrah-
301: C lung probabilities for TTB by materials
302: C RNG(NEE,NMAT) R4 Electron ranges by materials
303: C* (for photoelectric)
304: C WWAG(NMAT) R4 Generating probabilities of auger electron by
305: C highest Z in materials
306: C EEDG(NMAT) R4 K-edge energies of highest z in materials (eV)
307: C EEEK(NMAT) R4 X-ray energies of highest z in materials (eV)
308: C*
309: C ETOPE R4 Upper energy limit specified by user
310: C EBOTE R4 Lower energy limit specified by user
311: C*
312: C ETOPLE R4 Upper energy limit in electron library
313: C EBOTLE R4 Lower energy limit in electron library
314: C*
315: C ICRES(NICRES) I4 Data packet for point-wise response functions
316: C CRES(MCRES) R4 Point-wise response functions
317: C KCRES(NSTAL,4) I4 pointers to CRES etc.
318: C 1: pointer to CRES for each response
319: C 2: number of energy points
320: C 3: particle ( 1/2/3 = neutron/photon/electron )
321: C 4: 0/1 = not used/ used
322: C KCRESI(NSTAL,4) I4 same meaning for next event estimator as KCRES
323: C*
324: C*=====
325: c##<2007/03/14:PN3:

```

```

326: C*
327: C*
328: C* ----- for photo-nuclear data ----- 18 Oct 2001
329: C*
330: C*
331: C* << material specifications >>
332: C*
333: C MATTPN(NUCPNI) CH136 Mat-ID (ch8) / date (ch8) / comment (ch120)
334: C NCIDP(NUC) CH16 photo-nuclear nuclide ID's as input
335: C (if no given, same as NCIDI)
336: C NCIDPN(NUCPNI) CH16 photo-nuclear nuclide ID's given photo-
337: C nuclear data from index file
338: C NUCPNI I4 maximum number of photo-nuclear nuclides to use
339: C in problem
340: C NUCPN I4 number of photo-nuclear nuclides used
341: C NUCPNA I4 length of photo-nuclear nuclide table
342: C NUCPNB I4 maximum number of photo-nuclear nuclides in
343: C used each element
344: C*
345: C NMTPN I4 total number of photo-nuclear reactions
346: C considered (fix to 135)
347: C*
348: C MNUCPN(NMAT+1) I4 number of photo-nuclear nuclides in each material
349: C NNUCPN(NUCPNB,NMAT) I4 number of photo-nuclear nuclides for neutron
350: C ID in each material
351: C LPDPNP(NUCPNB,NMAT) I4 starting position of photo-nuclear nuclides
352: C for neutron ID in each material
353: C DNSTPN(NUCPNA) I4 number density table for photo-nuclear nuclides
354: C LPIDPN(NUCPNA) I4 reference pointer of photo-nuclear nuclide ID in
355: C NCIDPN
356: C IZTBPN(NUCPNA) I4 atomic number of photo-nuclear nuclides
357: C MNEUTPN(NUCPNA) I4 reference pointer of neutron ID corresponding to
358: C photo-nuclear nuclides
359: C*
360: C ETOPLPN R4 upper energy limit in photo-nuclear library
361: C EBOTLPN R4 lower energy limit in photo-nuclear library
362: C*
363: C* << cross section data for photo-nuclear >>
364: C*
365: C KLBPN1(NUCPNI,16) I4 Record #1 (specification record) of photo-
366: C nuclear cross sections
367: C*
368: C 1: LST starting position of data for a nuclide
369: C 2: NPTS number of smooth data grid points
370: C 3: NTDATA total words of #3 record
371: C 4: NMT2G number of reactions producing secondary products
372: C 5: NBINA number of equal probability bins in angular
373: C distribution
374: C 6: NNU number of coefficients for LNU=1, or number of
375: C energy in tabulation for LNU=2
376: C 7: NMT total number of reactions considered (=135)
377: C 8: LFI flag of fission cross section data (0/1=no/yes)
378: C 9: LNU flag of nu-value (0/1/2=no/polynomial/tabulated)
379: C 10: LSNU starting position of nu-value data
380: C 11: NNUD number of coefficients for LNUD=1, or number of
381: C energy in tabulation for LNUD=2
382: C 12: LNUD flag of nu-value for delayed neutron
383: C 13: NNF number of precursor families
384: C 14: LSNUD starting position of delayed nu-value data
385: C 15: LSTPEA starting position of product energy-angle
386: C distribution data
387: C 16: (spare)
388: C*
389: C XLBPN1(NUCPNI,6) R4 Record #1 (specification record) of photo-
390: C nuclear cross sections

```

src/mvp/INC/_CXSEC

```
391: C*
392: C      1:  ATW      atomic weight in n.m.u.
393: C      2:  TLAB     laboratory temperature in Kelvin
394: C      3:  ELOW     lowest energy (eV)
395: C      4:  EHI      highest energy (eV)
396: C      5:  ZA       the (Z,A) designation same as in ENDF
397: C      6:  MATNO    material number in real data
398: C*
399: C  KLBNP2(NUCPNI,NMTPN,13)  I4  Pointers to record #2 of photo-nuclear
400: C*
401: C      1:  MTINFO    data flag for each MT reaction
402: C      2:  MTPAR     flag for cross section type
403: C      3:  ISTMT     upper energy mesh position in energy mesh table
404: C      4:  IENDMT    lower energy mesh position in energy mesh table
405: C      5:  ICTMT     flag to specify frame of reference given in angular
406: C                    distribution
407: C      6:  NEANG      number of energy points in angular distribution
408: C      7:  NK2G       number of product energy or energy-angle
409: C                    distributions (subsections)
410: C      8:  NE2G       number of energy points in subsection probability
411: C      9:  MT2G       MT identification of secondary product generation
412: C                    reaction (NMT2G)
413: C     10:  LSTF3      starting position of smooth cross section for each
414: C                    MT reaction in #3 record
415: C     11:  LSTF4      starting position of angular distribution for each
416: C                    MT reaction in #3 record
417: C     12:  LST2G      starting position of energy-angle distribution for
418: C                    each MT reaction in #3 record
419: C     13:              (spare)
420: C*
421: C  XLBNP2(NUCPNI,NMTPN,2)   R4  Pointers to record #2 of photo-nuclear
422: C*
423: C      1:  QVAL      Q-value with its opposite sign (eV)
424: C      2:  QVAL2     QVAL*(ATW+1)/ATW
425: C*
426: C  CXPN(MCXPN)              R4  photo-nuclear cross section data.
427: C  MCXPN                    I4  length of CXPN array.
428: C*
429: C  .... for Doppler scattering ....
430: C  INTDS(NUC)              I4  Table of nuclides for which Doppler scattering
431: C                           is considered.
432: C*=====
433: c##>
```


src/mvp/INC/_FBANK2

```
1: C*
2: C*.... Secondary fission bank (copy of fission bank for calculation
3: C*   which needs more than one sub-batch for a batch)
4: C*
5: C*   See _XBANK for description of variables (XXXF2 corresponds to
6: C*   XXXF and array size is NHIST etc.)
7: C*
8:   parameter ( MDFBK2 =20 , MIFBK2 = 20 )
9:   common /XBNKF2/
10:  & LXXXF2,LYYF2, LZZZF2, LIZZF2, LEEEF2, LINUF2, LKLSFF2,
11:  & LLEVLF2, LLZZF2, LLPOSF2, LLCRS2, LLCRSF2,
12:  & LIBSFF2, LIBRGF2,
13:  & LDFBK2, KDFBK2(0:MDFBK2), LIFBK2, RIFBK2(0:MIFBK2)
```

src/mvp/INC/_FLAGS

```

1: C*
2: C* .... OPTIOTN FLAGS (MVP) .....
3: C*
4:     parameter (MAXJDB = 16, MXJVP = 8)
5:     parameter (MKPAR = 32)
6:     COMMON /FLAGS /
7:     & JREST, JARST,
8:     & JNEUT, JPHOT, JIMAG, JTIME, JDELT, JFISS, JEIGN, JFIXD,
9:     & JADJM, JDDX, JIMPT, JWWND, JPSTR, JWTIM, JFCOL, JBIAS, JRWVR,
10:    & JRESP, JMNTR, JVMNT, JSTPRN(3), JDEBG(MAXJDB),
11:    & JALLZ, JONEZ, JSIMP, JPICT, JREFL, JLATT, JHLAT, JFISX, JSTGR,
12:    & JFPRT, JRRLT, JSTOP, JRTTR, JRTCL, JPTS(20), JMICE(8), JMACE(8),
13:    & JGAMM, JTLLT, JMCHK, JREPR, JBREM,
14:    & JRUNM, JNCOL, JOSRC, JRSTF, JSCTI, JTSKR, JUNDG, JVPSP(MXJVP),
15:    & JAMXC, JANLF, JPTIM, JPTDT, JTSRF, JKPAP(MKPAR), JFMUL, JEXDP,
16:    & JPRT, JPTMP, JPDEN, ! pert
17:    & JBYLN, ! dn
18:    & JNRUN,
19:    & JSCTM, JSCMU, ! scat-mtx
20:    & JPHNU, ! photonuc
21:    & JNWMX,
22:    & JBEFF, ! beff
23:    & JTLST ! time-list
24: C*
25: C*.... OPTION FLAGS. DEFAULT VALUES ARE GIVEN IN 'FLAGIN'
26: C*
27: C**      INPUT-SYMBOL      DEFAULT      INCOMPATIBLE OPTION
28: C JREST I4      RESTART          : NO
29: C JARST I4      AUTO-RESTART     : NO
30: C JNEUT I4      NEUTRON          : YES
31: C JPHOT I4      PHOTON(PHOTON) OR PHOTON : NO
32: C JGAMM I4      PHOTON(GAMMA)    : NO
33: C JBREM I4      BREMSSTRAHLUNG   : NO
34: C JPHNU I4      PHOTO-NUCLEAR    : NO
35: C JIMAG I4      IMAGINARY-PARTICLE : NO      JNEUT JPHOT
36: C JTIME I4      TIME-DEPENDENT   : NO
37: C JDELT I4      DELTA-TRACKING    : NO
38: C JFISS I4      FISSION          : NO
39: C JEIGN I4      EIGEN-VALUE      : NO
40: C JFIXD I4      FIXED-SOURCE     : YES
41: C JDDX I4      DDX (NOT NECESSARY, NOW) : NO
42: C JADJM I4      ADJOINT          : NO
43: C JRRLT I4      RUSSIAN-ROULETTE : YES      JWWND
44: C JIMPT I4      IMPORTANCE       : NO      JWWND
45: C JWWND I4      WEIGHT-WINDOW    : NO      JIMPT JRRLT
46: C JPSTR I4      PATH-STRETCHING  : NO
47: C** JWTIM added July 1998
48: C JWTIM I4      apply time weight : NO      (to be set internally)
49: C JFCOL I4      FORCED-COLLISION  : NO
50: C JBIAS I4      SOURCE-BIAS      : NO
51: C** JRWVR added July 1998
52: C JRWVR I4      RELATIVE-WEIGHT   : NO
53: C      Apply weight reduction and secondary particle generation$
54: C      $ by weight relative to that of originating particle
55: C      $ at particle generation.$
56: C      $ (source or secondary/tertiary/... particle generation)
57: C JRESP I4      RESPONSE          : NO
58: C JMNTR I4      MONITOR          : NO
59: C JVMNT I4      VP-MONITOR       : YES
60: C JDEBG(1) I4   DEBUG-PRINT      : NO
61: C      Debug printout in data input phase.
62: C JDEBG(2) I4   DEBUG-PRINT      : NO
63: C      Unused.
64: C JDEBG(3) I4   DEBUG-PRINT      : NO
65: C      Printout particle source attributes.

```

```

66: C JDEBG(4) I4   DEBUG-PRINT      : NO
67: C      Check in event selection (SELONE)
68: C      .eq.2 : printout selected event # and stack sizes
69: C      after every call of SELONE.
70: C JDEBG(5) I4   DEBUG-PRINT      : NO
71: C      Check stack size consistency in event selection (SELONE)
72: C JDEBG(6) I4   DEBUG-PRINT      : NO
73: C      Save last event # for last JDEBG(6) events for each
74: C      particle.
75: C JALLZ I4      ALL-ZONE          : YES
76: C JONEZ I4      ONE-ZONE          : NO
77: C JPICT I4      PICTURE           : NO
78: C JREFL I4      (REFLECTIVE SURFACE) : NO      (TO BE SET INTERNALLY)
79: C JLATT I4      LATTICE GEOMETRY   : NO
80: C JFPRT I4      FLUX-RPINT        : NO
81: C JSTOP I4      JOB TERMINATION    (TO BE SET INTERNALLY)
82: C JNRUN I4      NO RANDOM WALK     (TO BE SET INTERNALLY)
83: C      If JNRUN != 0, no random walk is performed but a restart
84: C      file is output.
85: C
86: C JRTRT I4      EDIT-BY-TRACK-LENGTH : YES
87: C JRTCL I4      EDIT-BY-COLLISION   : NO
88: C JTLLT I4      TALLY-LATTICE      : NO
89: C      UNIVERSE-DEPENDENT-TALLY
90: C      FRAME-DEPENDENT-TALLY
91: C      $ Independent region assignments to each nested zones in$
92: C      $ lattice geometry
93: C JHLAT I4      Problem with hexagonal lattice.
94: C JSIMP I4      Geometry is simple (no (-) signed body having more than one$
95: C      $ surface). set by code in subroutine 'zonein'.
96: C
97: C JREPR I4      REPRODUCIBLE-RUN    : YES
98: C      $ Run multitasking mode so as to get reproducible $
99: C      $results. Non-reproducible run (if possible) may be $
100: C      $more effective in multitasking mode.
101: C*
102: C***** EDIT FLAGS *****
103: C*
104: C*      DEFAULT      DEFAULT
105: C*      <JEIGN=1>    <JEIGN=0>
106: C JMICE I4      EDIT-MICROSCOPIC-DATA(K) :
107: C      $ defaults of K;
108: C      $ <JEIGN=1> 00000000 <JEIGN=0> 00000000
109: C JMACE I4      EDIT-MACROSCOPIC-DATA(K) :
110: C      $ defaults of K;
111: C      $ <JEIGN=1> 00000000 <JEIGN=0> 00000000
112: C
113: C      $ K is an 8-digit integer whose digits means treatement for$
114: C      $ microscopic and macroscopic reactions rate and cross sections.
115: C      10000000 * < TOTAL >
116: C      + 1000000 * < NU*FISSION >
117: C      + 100000 * < FISSION >
118: C      + 10000 * < ELASTIC SC >
119: C      + 1000 * < CAPTURE >
120: C      + 100 * <INELASTIC SC>
121: C      + 10 * < (N,2N) >
122: C      + 1 * < LOSS >
123: C
124: C <MEANS OF EACH DIGIT>
125: C
126: C      0 : no calculation
127: C      >0: calculate reaction rate & cross section.
128: C      1 : saved in file (unit 30) but not printed out.
129: C      2 : saved in file (unit 30)
130: C      $      & only cross section printed out.

```

src/mvp/INC/_FLAGS

```
131: C      3 : saved in file (unit 30)
132: C      $ & only reaction rate printed out.
133: C      4 : saved in file (unit 30) & printed out.
134: C*
135: C*
136: C***** PRINT SUPPRESS FLAGS *****
137: C*
138: C JPRTS(20) I4 If JPRTS=1, print is suppressed.
139: C
140: C      1 : group-wise flux          2 : angular-dependent flux
141: C      3 : time-dependent flux      4 : response
142: C      5 : time-dependent response  6 : source information #1
143: C      7 : source information #2
144: C      8-16 : not used
145: C      17: microscopic reaction rate 18: microscopic cross section
146: C      19: macroscopic reaction rate 20: macroscopic cross section
147: C*
148: C JMCHK I4 'MEMORY-CHECK' option for debugging.
149: C      *Print memory assignment in 'keep' routine.
150: C      *Programmers can check
151: C      invalid value assignments for memories on /ARRAY/ common
152: C      by calling 'MMCHK' routine.
153: C*
154: C JRUNM I4 'RUN-MODE' option to stop processing before random-walk
155: C      for input data checking etc.
156: C
157: C      Run mode is controlled by each decimal digits as follows:
158: C
159: C      all digit is 0 : ORDINARY RUN.
160: C
161: C      n1*1 : n1>0 stop before randomwalk.
162: C      + n2*10 : n2=1 skip cross section input (and no randomwalk).
163: C      (n2: not implemented currently, Sep 1994)
164: C      + n3*100 : < no use >
165: C      + n4*1000 : < no use >
166: C      + n5*10000 : < no use >
167: C      + n6*100000 : < no use >
168: C      + n7*1000000 : < no use >
169: C      + n8*10000000 : < no use >
170: C
171: C*** JNCOL : Added Oct 1996
172: C JNCOL I4 COLLISION-LESS : particles have no collision.
173: C      Only collisionless contribution by source particles
174: C      is evaluated as tracklength estimators.
175: C      Particle weights are reduced with absorption probability.
176: C      Virtual materials such as perfect absorber(-1000),
177: C      perfect/white reflectors are treated as such.
178: C
179: C*** JOSRC : Added May 1997
180: C JOSRC I4 SOURCE-OUTPUT : output particle source data on a file.
181: C*** JRSFT : Added May 1998
182: C JRSTF I4 RESTART-FILE : output restart file.
183: C*** JOSRC : Added Jun 1997
184: C JSCRT I4 OPEN-SCRATCH : Open scratch I/O units
185: C      if they are not specified to open in command line.
186: C      (This is not to make ugly fort.** etc in sub-task of
187: C      PVM/MPI multi-task mode.)
188: C JTSKR I4 Spawn "tasker" process which assigns jobs to workhorse
189: C      tasks in multi-tasking mode if necessary.
190: C
191: C*** added 25 Jul 1997
192: C JUNDG I4 "Underground" level particle event processing is necessary.
193: C      Currently "underground" level processings are particle
194: C      tracking for next event estimator, and zone-based sampled
195: C      source rejection.
```

```
196: C      This flag is set internally and cannot be given as input.
197: C
198: C*** added 3 Aug 1997
199: C
200: C JVPPS(*) I4 option flags specific to VPP-Fortran multi-task mode.
201: C      JVPPS(1) = 1 : Printouts of "ACTION" phase of each task are
202: C      output on the same I/O unit.
203: C*** added Nov 1998
204: C JFISX I4 Lattice frames or cells may intersect each other.
205: C      (to be set internally)
206: C JSTGR I4 Problem with STG region (to be set internally)
207: C*** added Oct 1999
208: C JAMXC I4 ADAPTIVE-MICRO-CALCULATION :
209: C      Do not calculate micro cross section of all nuclides after
210: C      collision, but calculate micro cross section of nuclides
211: C      when they are necessary for macro cross section calculation
212: C      etc.
213: C*** added 9 Apr 1999
214: C JANLF I4 ANALOG-FISSION : NO
215: C      if JANLF=1, automatically set
216: C      JFIXD=1, JEIGN=0, JFISS=1,
217: C      no effective in GMVP now.
218: C*** added 7 Jul 1999
219: C JPTIM I4 PERIODIC-TIME : NO
220: C      effective only when if JTIME=1.
221: C*** added 25 Nov 1999
222: C JPSTD I4 POINT-DETECTOR : NO
223: C*** added 3 Feb 2000
224: C JSTPR I4 SUBTASK-PRINT
225: C      printout from subtask in multi-task mode.
226: C
227: C      JSTPRN(1) : for printout in input/preparation phase.
228: C
229: C      0: throw away all printout.
230: C      1: print warning/error message (I/O unit IMSEG) but throw away
231: C      other printout. (default in MPI/PVM)
232: C      2: print all printout. (default in multi-thread mode)
233: C      -1: same as 1 but append to printout of main task
234: C      (currently unsupported)
235: C      -2: same as 2 but append to printout of main task
236: C      (currently unsupported)
237: C
238: C      JSTPRN(2) : for printout in random walk phase.
239: C      (currently no control is effective)
240: C
241: C      Option value has the same meaning with SUBTASK-PRINT(1).
242: C      Default is SUBTASK-PRINT(2)=2 (print all)
243: C*** added Mar 2000
244: C JTSRF I4 Use surface crossing tally
245: C*** added Apr 2000
246: C JKPAR(MKPAR) I4 specified particles are treated in current
247: C      calculation if non-zero.
248: C*** added Oct 2000
249: C JFMUL I4 Option for fission multiplicity.
250: C      0 : total-nu
251: C      1 : prompt-nu only
252: C*
253: C* ... perturbation options ...
254: C JPRT I4 PERTURBATION-CALCULATION : NO
255: C JPTMP I4 TEMPERATURE PERTURBATION : NO
256: C JPDEN I4 DENSITY or NUMBER-DENSITY PERTURBATION : NO
257: C* ... end of perturbation options ...
258: C*
259: C JDLYN I4 DELAYED-NEUTRON : NO
260: C*** added Feb 2004
```

src/mvp/INC/_FLAGS

```
261: C JSCTM I4   SCATTERING-MATRIX      : NO
262: C           Use scattering matrix tally for elastic, thermal neutron or
263: C           inelastic.
264: C           0 : not used this tally
265: C           n : (n-1) is the order of high-moments for scattering matrix.
266: C           (default is 1)
267: C*** added Jul 2005
268: C JSCMU I4   SCATTERING-MUBAR         : NO
269: C           Use scattering mubar tally for elastic, inelastic or (n,2n)
270: C           reaction.
271: C JNWMX I4   NUCLIDE-WISE-MACRO-XSEC   : NO
272: C           Output nuclide-wise macroscopic cross sections instead of
273: C           microscopic cross sections.
274: C JBEPF I4   BETA-EFFECTIVE             NO
275: C           Beta-effective calculations with constant, Nauchi, Meulekamp
276: C           methods (next fission probability).
277: C JTLST I4   TIME-LIST                  : NO
278: C           Output time list data with noise analysis.
279: C*
```

src/mvp/INC/_IUNIT2

```
1: C*
2: C* I/O unit parameters specific to MVP:
3: C*
4: C* IXS : input unit of cross section library. (neutron)
5: C* IXP : input unit of cross section library. (photon)
6: c##<2007/03/14:PN3:
7: C* IXPN : input unit of cross section library. (photonuclear data)
8: c##>
9: C* IXE : input unit of cross section library. (electron)
10: C* IXD : input unit of point-wise response library. (dosimetry)
11: C*
12: C* IXSF : input unit of free temperature cross section library.
13: C* (neutron). Must be different from IXS.
14: C*
15: C* IXLOCK : I/O unit used to open "lock file" used when save processed
16: C* ART library.
17: C*
18: C* IDXNF : neutron cross section index file for free temperature
19: C* library (text). (added May 1998)
20: C* IDNX : neutron cross section index file for fixed temperature
21: C* library (text)
22: C* IDXP : photon cross section index file (text)
23: c##<2007/03/14:PN3:
24: C* IDXPN: photonuclear cross section index file (text)
25: c##>
26: C* IDXE : electron cross section index file (text)
27: C* IDXD : point-wise response (dosimetry cross section) index file
28: C* (text)
29: C*
30: parameter ( IXSF=49, IXS=50, IXP=50, IXE=50, IXLOCK = 51 )
31: parameter ( IDXNF=24, IDNX=25, IDXP=26, IDXE=27 )
32: parameter ( IXD=50, IDXD=29 )
33: c##<2007/03/14:PN3:
34: parameter ( IXPN=50, IDXPN=28 )
35: c##>
```

src/mvp/INC/_ISOTOP

```
1: C*
2: C*..... number of isotopes and their abundance and mass .....
3: C*
4:      parameter ( MATOM = 103 )
5:      common /ISTPTB/ TISOTP(3,10,MATOM),ISOTOP(MATOM)
```

SAFE

src/mvp/INC/_KPIDS

```
1: C
2: C-----
3: C Particle ID's and particle species number limit for MVP
4: C
5: C Please modify particle symbol information in file _KPSYM
6: C and symbol initialization routine (KPINIT) when particle species
7: C are added or removed from this file.
8: C
9: C
10: C ... neutron and photon
11: C     parameter ( KPNEUT=1 )
12: C     parameter ( KPPHOT=2 )
13: C ... electron and proton
14: C     parameter ( KPELEC=3 )
15: C     parameter ( KPPROT=4 )
16: C ... pi mesons (pi-0 pi+ pi- )
17: C     parameter ( KPPI0=5 )
18: C     parameter ( KPPIP=6 )
19: C     parameter ( KPPIM=7 )
20: C ... mu mesons (mu+ mu- )
21: C     parameter ( KPMUP =8 )
22: C     parameter ( KPMUM =9 )
23: C
24: C-----
25: C ... number of particles possible to treat in this code
26: C
27: C * this should be maximum of possible KP*
28: C * flag array JKPAR must have element size >= KPLIM.
29: C
30: C     parameter ( KPLIM = 3 )
31: C
```

src/mvp/INC/_KPMASS

```
1: C
2: C ... particle mass in neturon-mass-unit.
3: C   values should be given in KPINIT routine.
4: C
5:   common /KPMASCG/ PMASS
6:   real*8 PMASS(KPLIM)
7: C
```

SAE

src/mvp/INC/_KPSYMS

```
1: C
2: C Particle ID's and symbol matching table used in symbol/ID conversion
3: C routine and its initialization routine of MVP.
4: C
5: C -----
6: C
7: C << "Must-be" 's of this file !!! >>
8: C
9: C * Particle ID definition file _KPIDS *must be* included before
10: C this file
11: C
12: C * When contents of particle ID definition file _KPIDS are changed,
13: C this file *must be* modified to follow the change.
14: C
15: C -----
16: C
17: C ... number of particles possible to treat in this code
18: C (this should be maximum of possible KP* )
19: C
20: C parameter ( KPLIM = ? )
21: C
22: C
23: C ... particle symbol string and their alias (or short form)
24: C
25: C character*32 KPSYM
26: C common /CKPSYM/ KPSYM(2,KPLIM)
27: C
28: C === possible initialization
29: C
30: C ... neutron and photon
31: C KPSYM(1,KPNEUT) = 'NEUTRON'
32: C KPSYM(2,KPNEUT) = 'N'
33: C KPSYM(1,KPPHOT) = 'PHOTON'
34: C KPSYM(2,KPPHOT) = 'P'
35: C ... electron and proton
36: C KPSYM(1,KPELEC) = 'ELECTRON'
37: C KPSYM(2,KPELEC) = 'EL'
38: C KPSYM(1,KPPROT) = 'PROTON'
39: C KPSYM(2,KPPROT) = 'PR'
40: C ... pi mesons (pi-0 pi+ pi- )
41: C KPSYM(1,KPPIO) = 'PIO'
42: C KPSYM(2,KPPIO) = 'PI0'
43: C KPSYM(1,KPPIP) = 'PI-PLUS'
44: C KPSYM(2,KPPIP) = 'PIP'
45: C KPSYM(1,KPPIM) = 'PI-MINUS'
46: C KPSYM(2,KPPIM) = 'PIM'
47: C ... mu mesons (mu+ mu- )
48: C KPSYM(1,KPMUP) = 'MU-PLUS'
49: C KPSYM(2,KPMUP) = 'MUP'
50: C KPSYM(1,KPMUM) = 'MU-MINUS'
51: C KPSYM(2,KPMUM) = 'MUM'
52: C
```

src/mvp/INC/_NGPS

```
1: C
2: C   ... INC/_KPIDS must be included before this file to refer KPLIM
3: C
4: C   COMMON /NGPCOM/  NGP(KPLIM), KNGP(KPLIM+1), KENGP(KPLIM+1)
5: C
6: C NGP(KPLIM)   I4   number of (common) energy groups for each particle
7: C               species.
8: C KNGP(KPLIM+1) I4   KNGP(J) is < Sum of NGP(i),i=1,J-1> + 1.
9: C               It is also used to calcualte energy group # for each
10: C              particle species (from 1 to NGP(particleID)) from
11: C              "unified" energy group # (from 1 to NGROUP).
12: C KENGP(KPLIM+1) I4   KNGP(J) is < Sum of (NGP(i)+1),i=1,J-1> + 1.
13: C              It is used to point arrays on which data for each particle
14: C              species having length of <NGP(*)+1> (such as energy group
15: C              boundaries) are packed in the order of particle species ID.
16: C
```

src/mvp/INC/_NMON

```
1: C*  
2: C* >>> NUMBER OF CPU-MONITORED ROUTINES ( VP-MONITOR MODE )  
3: C*  
4:     PARAMETER ( NMON = 12 )
```

SAFE

src/mvp/INC/_PERT

```

1: C*
2: C*      MXPTOP:I4: Number of flags for perturbation calculation.
3: c      Added by JRI on Dec 24 in 2003.
4:      integer MXPTOP
5:      parameter ( MXPTOP=9 )
6: C*
7: C*..... POINTERS TO WORKING ARRAYS : PERTURBATION CALCULATION
8: C*
9:      COMMON /PERT/ PT(30)
10:     INTEGER PT
11: C*
12: C PT(6) I4 Pointers to working arrays for perturbation calculation.
13: c added by Y.Nagaya 1999/11/22
14: C*
15: C*..... Tally array parameters : PERTURBATION CALCULATION
16: C*
17:     common /PERT0/ NPTDS, LJPTTR, LJPTNU, NPTCS, LDELA,
18:     &      MAXDA, NUCPT, NGSP, NTPT, LJPTOP, NORDDS, NOCS,
19:     &      NOPSDS, NPTBE
20:     integer NPTDS, LJPTTR, LJPTNU, NPTCS, LDELA,
21:     &      MAXDA, NUCPT, NGSP, NTPT, LJPTOP, NORDDS, NOCS,
22:     &      NOPSDS, NPTBE
23: C*
24: c NPTDS : I4 : Number of perturbation tallies for differential
25: c      operator sampling method.
26: c JPTTR(NREG,NPTDS) : I4 : Flags to specify perturbed regions.
27: c JPTNU(NUC,NPTDS) : I4 : Flags to specify perturbed nuclides.
28: c NPTCS : I4 : Number of perturbation tallies for correlated
29: c      sampling method.
30: c NUCPT : I4 : Number of perturbed nuclides.
31: c NGSP : I4 : Number of generations for source perturbation.
32: c NTPT : I4 : (NGSP+3)*NPTDS+(NGSP+2)*NPTCS
33: c JPTOP(MXPTOP,NPTDS+NPTCS): I4 : Array of flags for perturbation calculation.
34: c      Added by JRI on Dec 24 in 2003.
35: c NORDDS : I4 : Order of differential sampling method. (2006/10/10)
36: c NOPSDS : I4 : Order of perturbed source for differential sampling method.
37: c      (2006/10/11)
38: c NOCS : I4 : (starting location - 1) of correlated sampling data
39: c      in WCTRP. (2006/10/10)
40: c NPTBE : I4 : Number of perturbation tallies for beta-effective
41: c      with correlated sampling. (2007/10/28)
42: C*.....
43: C* pointers to task local data
44: C*.....
45: C*
46:     common /PERTL/ LXKEFP,LWWD,LWD0,LWCTRP,LWWD2,LWWR,LWR0,
47:     &      LWWT,LWT0,LWTOA,LWT0B,LWWN,LWN0,LWNOA,LWNOB,
48:     &      LWSD,LWSC
49: c+beff2
50:     &      ,LWWB,LWSB,LWB0,LWWBD,LWSBD,LWBD0,LWWLD,LWSLD,LWLD0
51: c-beff2
52:     integer LXKEFP,LWWD,LWD0,LWCTRP,LWWD2,LWWR,LWR0,
53:     &      LWWT,LWT0,LWTOA,LWT0B,LWWN,LWN0,LWNOA,LWNOB,
54:     &      LWSD,LWSC
55: c+beff2
56:     &      ,LWWB,LWSB,LWB0,LWWBD,LWSBD,LWBD0,LWWLD,LWSLD,LWLD0
57: c-beff2
58: c
59: c XKEFP(NPTDS*2,7,NLENG) : R8 : (MVP) dk/da tallies of each batch.
60: c      <JEIGN>
61: c      (,1,) track length (production)
62: c      (,2,) collision (production)
63: c      (,3,) analog (production)
64: c      (,4,) track length (absorption)
65: c      (,5,) collision (absorption)
66: c      (,6,) analog (absorption)
67: c      (,7,) leakage
68: c
69: c WWD(NBANK,NPTDS,NORDDS) : R8 : Additional weights for 1st order
70: c      differential operator sampling method.
71: c WWD2(NBANK,NPTDS) : R8 : Additional weights for 2nd order
72: c      differential operator sampling method.
73: c WSD(NBANK,0:NGSP,NPTDS) : R8 :
74: c      Additional weights for source perturbation
75: c      with differential operator sampling method.
76: c
77: c WWT(NBANK,NPTDS) : R8 : Additional weights for TEMPERATURE
78: c WWN(NUCPT,NBANK,NPTDS)
79: c      : R8 : Additional weights for NUMBER DENSITY
80: c ... Order of NPTDS is decided temporarily.
81: c (,1) 1st order differential operator without fission source change
82: c (,2) 1st order fission source change only,
83: c (,3) 1st order differential operator with fission source change
84: c
85: c WDO(NFBANK0,0:NGSP,NPTDS) : R8 :
86: c      Initial additional weights for differential operator sampling method.
87: c      1 dS
88: c      --
89: c      S da
90: c
91: c WRO(NFBANK0,0:NGSP,NPTCS) : R8 :
92: c      Initial additional weights for correlated sampling method.
93: c      S'
94: c      --
95: c      S
96: c
97: c WCTRP(NUCPT,7,NTPT) : R8 : Weight counter for perturbation calculation.
98: c      (1,) track length (production)
99: c      (2,) collision (production)
100: c      (3,) analog (production)
101: c      (4,) track length (absorption)
102: c      (5,) collision (absorption)
103: c      (6,) analog (absorption)
104: c      (7,) leakage
105: c
106: c end of addition
107: c+beff2
108: c
109: c WWBD(NBANK,NPTDS) : R8 : Additional weights for
110: c      beta-effective calculation.
111: c
112: c WBD0(NFBANK0,0:NGSP,NPTCS) : R8 :
113: c      Initial additional weights for beta-effective calculation.
114: c      S'
115: c      --
116: c      S
117: c
118: c WSBD(NBANK,0:NGSP,NPTDS) : R8 :
119: c      Additional weights for source perturbation
120: c      with beta-effective calculation.
121: c
122: c WWLD(NBANK,NPTDS) : R8 : Additional weights for
123: c      generation time calculation.
124: c
125: c WLD0(NFBANK0,0:NGSP,NPTCS) : R8 :
126: c      Initial additional weights for generation time calculation.
127: c      S'
128: c      --
129: c      S
130: c

```

src/mvp/INC/_PERT

```
131: c WSLD(NBANK,0:NGSP,NPTDS) : R8 :  
132: c   Additional weights for source perturbation  
133: c   with generation time calculation.  
134: c  
135: c  
136: c-beff2
```

SAFE

src/mvp/INC/_PMNTR

```
1: C*.... CPU & VECTOR LENGTH MONITOR .....
2: C*      (LOCAL VARIABLES
3: C*
4: cc*      COMMON /PMNTR/ CPUSOR,CPUSEL,CPUFLY,CPUSCH,CPUCOL,
5: cc*      &   NEXSOR,NVSOR,VLSOR,VL2SOR,MIVSOR,MAVSOR,
6: cc*      &   NEXSEL,NVSEL,VLSEL,VL2SEL,MIVSEL,MAVSEL,
7: cc*      &   NEXFLY,NVFLY,VLFLY,VL2FLY,MIVFLY,MAVFLY,
8: cc*      &   NEXSCH,NVSCH,VLSCH,VL2SCH,MIVSCH,MAVSCH,
9: cc*      &   NEXCOL,NVCOL,VLCOL,VL2COL,MIVCOL,MAVCOL
10: C
11: cc*      REAL          CPUSOR,CPUSEL,CPUFLY,CPUSCH,CPUCOL,
12: cc*      &             VLSOR,VL2SOR,
13: cc*      &             VLSEL,VL2SEL,
14: cc*      &             VLFLY,VL2FLY,
15: cc*      &             VLSCH,VL2SCH,
16: cc*      &             VLCOL,VL2COL
17: C*
18: C*
19: C*CPU???  R4   CPU TIME OF EVENTS
20: C*NEX???  I4   NUMBER OF EXECUTION OF EVENTS
21: C*NV???   I4   TOTAL NUMBER OF VECTOR EVENT
22: C*VL???   R4   SUM OF VECTOR LENGTH
23: C*VL2???  R4   SQUARE SUM OF VECTOR LENGTH
24: C*MI???   I4   MINIMUM VECTOR LENGTH
25: C*MA???   I4   MAXIMUM VECTOR LENGTH
26: C*
27: C*  '???' IS THE SYMBOL FOR EACH SUBROUTINE.
28: C*  SOR = SOURCE, SEL = SELECT, FLY = FLIGHT, SCH = SEARCH, COL = COLISEN
29: C*
```

src/mvp/INC/_PROGV

```
1: C*
2: C*..... DESCRIPTION OF PROGRAM VERSION  ETC. (MVP).....
3: C*   Last updated : 5 Apr 1994 ...
4: C*
5:      COMMON /CPROGV/ PROGV
6:      CHARACTER*60 PROGV(40)
```

SAFE

src/mvp/INC/_PSOUR

```
1: C*
2: C*.... SOURCE DATA & ARRAY POINTERS
3: C*
4:     COMMON /PSOUR/
5:     & LKSOUR, LISZON, LSOUR , LIDSRC,
6:     & LPSPAC, LPENRG, LKENRG, LNSTIM, LSTIM ,
7:     & LPSTIM, LISTIM, LNSANG, LSANG , LSAXIS, LPSANG, LISANG,
8:     & LIFISM, LMEMZN, LINSRC, LXSOUR,
9:     & LSRCSF, NSRCSF, LLXYZ, LLPWRK, LLVSTK, MWVEC, MVSTK, MXREJ
10: C*
11: C Ksour(NSOUR)      I4  Type of each particle source (old type).
12: C ISZON(2,NSOUR)    I4  Input zone # & zone # of each source (old type).
13: C**** 5 Nov 1996: SOUR(NSOUR) changed to real*8
14: C SOUR(NSOUR)       R8  Relative source intensity of each source.
15: C IDSRC(NSOUR)      I4  source set ID of each source. (this may not be
16: C                    $ a uniq number.)
17: C PSPAC(10,NSOUR)   R4  Spatial distribution of particle source $
18: C                    $(old type). Meanings of each element$
19: C                    $ depend on the type of each source.
20: C PENRG(NGROUP,NSOUR) R4 Energy distributions for each source
21: C                    (old type).
22: C*KENRG(2,NGROUP,NSOUR) I4 ENERGY BIN # (FOR DISCRETE SAMPLING)
23: C*NSTIM(NSOUR)      I4  TIME BIN NUMBER OF EACH SOURCE. <JTIME>
24: C*STIM(NSTIM(NSOUR),NSOUR) R4 TIME BINS OF EACH SOURCE. <JTIME>
25: C*PSTIM(NSTIM(NSOUR),NSOUR) R4 TIME DISTRIBUTION. <JTIME>
26: C*ISTIM(NSOUR)      I4  POINTER TO STIM & PSTIM OF EACH SOURCE
27: C                    $. <JTIME>
28: C*NSANG(NSOUR)      I4  ANGLE NUMBERS OF EACH SOURCE.
29: C*SANG(NSANG(NSOUR),NSOUR) R4 ANGLES (COSINE) BINS OF EACH SOURCE.
30: C*SAXIS(3,NSOUR)    R4  DIRECTION COSINES TO SOURCE GENERATION AXES.
31: C*PSANG(NSANG(NSOUR),NSOUR) R4 ANGLE DISTRIBUTION.
32: C*ISANG(NSOUR)      I4  POINTER TO SANG & PSANG OF EACH SOURCE.
33: C*
34: C*IFISM(NSOUR)      I4  MATERIAL # WHOSE FISSION SPECTRUM IS USED.
35: C IFISM(NSOUR)      I4  Nuclide # whose fission spectrum is used $
36: C                    $as energy spectrum of initial source in $
37: C                    $eigenvalue problem.
38: C MEMZN(NMEMS,NSOUR) I4  Generated zone # memory for particle sources.
39: C*
40: C*INSRC(NSOUR) I4  NUMBERS OF GENERATED PARTICLES FROM EACH SOURCE.$
41: C*                    $ (WORKING ARRAY)
42: C Xsour(NSOUR) R4  Working area to sample decimal part of NGENE*SOUR(N).
43: C*
44: C*
45: C* ---- For new type source sampling routines -----
46: C*
47: C SRCSP(*) I4 Data container of source sampling information.
48: C NSRCSF I4 Length of the data container of source sampling information.
49: C MWVEC I4 Maximum number of working vector necessary in sampling
50: C MVSTK I4 Maximum vector stack depth for vector calculator used
51: C          in 'SYSSRC' routine.
52: C LXZY(10) I4 Pointers to save sampled data temporary in 'SOURCE'
53: C          routine.
54: C LPWRK(MWVEC) I4 Pointers to working vector necessary in
55: C          source sampling
56: C LVSTK(MVSTK) I4 Pointers to vector stack for vector calculator used
57: C          in 'SYSSRC' routine.
58: C
59: C MXREJ I4 Maximum of number of rejections in "ACCEPT" block of source
60: C          generation.
61: C
```


src/mvp/INC/_PTALY2

```

1: C*
2: C*..... TALLY ARRAY POINTERS (CONTINUOUS ENERGY) .....
3: C*      (MODIFICATION MAY 1992 : REGION & TALLY REGION )
4:      COMMON /PTALY2/
5:      &  LRMIC,  LRMICR,  LXMIC,  LRMAC,  LRMACR,  LXMAC,  LDMAC,
6:      &  LRSTR,  LSRSTR,  LRSCL,  LRSCL,
7:      &  LDNZON, LDNREG, LLEMIC, LLEMIC, LWCXTY, LDNFLX,
8: c%%<2004/02/20:
9: c%% &  NEMIC, NEMAC , LSRMIC
10: &  NEMIC, NEMAC , LSRMIC, LDNFLXSM, LRMICSM, LSRMICSM, LXMICSM,
11: c%%<2005/07/20:
12: c%% &  LRMACSM, LXMACSM
13: &  LRMACSM, LXMACSM, LRMIMU, LSMIMU, LWMIMU, LRMAMU
14: c%%>
15: c%%>
16: C*
17: C*==== MICROSCOPIC REACTION RATES (TRACK LENGTH/COLLISION ESTIMATOR)
18: C*
19: C*
20: C NEMIC I4 Number of microscopic reaction rates and/or cross sections
21: C      $ calculated.
22: C*
23: C LEMIC(8,2,2) I4 Position indices of microscopic neutron reactions $
24: C      $in arrays such as "RMIC", "SRMIC" etc. $
25: C      $(arrays having dimension of size "NEMIC").
26: C
27: C      < LEMIC(I,1,1) : array index in ARRAY(...,NEMIC,...) >
28: C
29: C      LEMIC(1,1,1) : total
30: C      LEMIC(2,1,1) : nu*fission
31: C      LEMIC(3,1,1) : fission
32: C      LEMIC(4,1,1) : elastic scattering
33: C      LEMIC(5,1,1) : capture
34: C      LEMIC(6,1,1) : inelastic scattering
35: C      LEMIC(7,1,1) : (n,2n)
36: C      LEMIC(8,1,1) : loss
37: C
38: C      < LEMIC(J,2,1) J=1 to NEMIC >
39: C
40: C      LEMIC(J,2,1): reaction ID (1-8 ABOVE) for J-th array index
41: C
42: C      LEMIC(*,*,2) is reserved for photon reaction.
43: C*
44: C RMIC(NGROUP,NREG,NUC,NEMIC) R8 Microscopic reaction rate tally $
45: C      $accumulated during each batch and cleared after each batch. $
46: C      $(neutron)
47: C
48: C SRMIC(NGROUP,NTREG,NUC,NEMIC,2) R8 Microscopic reaction rate tally$
49: C      $ summed over batches. (neutron)
50: C
51: C      (,,,1) sum of batch tallies
52: C      (,,,2) square sum of batch tallies
53: C
54: C RMICR(NTREG,NUC,NEMIC,2) R8 group sum of rmic. (neutorn)
55: C
56: C      (,,,1) sum of batch tallys
57: C      (,,,2) square sum of batch tallies
58: C*
59: C XMIC(NGROUP,NTREG,NUC,NEMIC,2) R8 Microscopic cross section tally$
60: C      $ (neutron)
61: C
62: C      (,,,1) sum of batch tallies
63: C      (,,,2) square sum of batch tallies
64: C
65: C DNFLX(NGROUP,NREG,NUC) R8 Tally of density*flux values for each$

```

```

66: C      $ nuclide.
67: c%%<2004/02/20:
68: C*
69: c%%<2007/01/10:
70: c%%C RMICSM(NGP1,NGP1,NREG,NUC,1:3) R8 Microscopic reaction rate tally $
71: c%%C      $ of scattering matrix accumulated during each batch and $
72: C RMICSM(NGP1,NGP1,NREG,NUC,NFL+1,1:3) R8 Microscopic reaction rate $
73: C      $tally of scattering matrix accumulated during each batch and $
74: c%%>
75: C      $cleared each batch. (neutron)
76: C      dimension:
77: c%%<2007/01/10:
78: c%%C      ( group number of incident neutron energy,
79: c%%C      group number of outgoing neutron energy,
80: c%%C      region number,
81: c%%C      nuclide number,
82: c%%C      elastic:inelastic:(n,2n) )
83: C      ( #1 ... group number of incident neutron energy,
84: C      #2 ... group number of outgoing neutron energy,
85: C      #3 ... region number,
86: C      #4 ... nuclide number,
87: C      #5 ... moment order + 1 (number of meoments is NFL),
88: C      #6 ... elastic:inelastic:(n,2n) )
89: c%%>
90: C
91: c%%<2007/01/10:
92: c%%C SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,1:3) R8 Microscopic reaction $
93: c%%C      $rate tally of scattering matrix summed over batches. (neutron)
94: C SRMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,NFL+1,1:3) R8 Microscopic $
95: C      $reaction rate tally of scattering matrix summed over batches.$
96: C      $(neutron)
97: c%%>
98: C      dimension: #3 tally region number
99: c%%<2007/01/10:
100: c%%C      (,,,1,) sum of batch tallies
101: c%%C      (,,,2,) square sum of batch tallies
102: C      (,,,1,,) sum of batch tallies
103: C      (,,,2,,) square sum of batch tallies
104: c%%>
105: C
106: C XMICSM(NGP1+1,NGP1+1,NTREG,NUC,2,1:3) R8 Microscopic probability $
107: C      $tally of scattering matrix. (neutron)
108: C      ----- currently, not allocated in intro2.f -----
109: C
110: C DNFLXSM(NGP1,NGP1,NREG,NUC,1:3) R8 Tally of density*flux values $
111: C      $of scattering matrix for each nuclide.
112: C
113: c%%>
114: c%%<2005/07/20:
115: C*
116: C RMIMU(NGP1,NGP1+1,NREG,NUC,1:3) R8 Microscopic mubar tally of $
117: C      $scattering reaction accumulated during each batch and $
118: C      $cleared each batch. (neutron)
119: C      dimension:
120: C      ( group number of incident neutron energy,
121: C      group number of outgoing neutron energy + 1 (total),
122: C      region number,
123: C      nuclide number,
124: C      elastic:inelastic:(n,2n) )
125: C
126: C SMIMU(NGP1+1,NGP1+2,NTREG,NUC,3,1:3) R8 Microscopic mubar tally $
127: C      $of scattering reaction summed over batches. (neutron)
128: C      dimension: #3 tally region number
129: C      (,,,1,) sum of batch tallies
130: C      (,,,2,) square sum of batch tallies

```

src/mvp/INC/_PTALY2

```
131: C          (,,,3,) sum of batch tallies for weight
132: C
133: C WMIMU(NGP1,NGP1+1,NREG,NUC,1:3) R8 Tally of weight values of $
134: C          $scattering mubar for each nuclide.
135: C
136: C%%>
137: C*
138: C*==== MACROSCOPIC REACTION RATES =====
139: C*
140: C*
141: C NEMAC I4 Number of macroscopic reaction rates and/or cross sections $
142: C          $calculated.
143: C*
144: C LEMAC(16) I4 POSITION INDEXES IN RMIC ETC. (HAVING 'NEMAC' DIMENSION)
145: C LEMAC(8,2) I4 Position indices of macroscopic neutron reactions $
146: C          $in arrays such as "RMAC", "SRMAC" etc. $
147: C          $(arrays having dimension of size "NEMAC").
148: C
149: C
150: C < LEMAC(I,1) : array index in ARRAY(...,NEMAC,...) >
151: C
152: C $ LEMAC(1,1) : total
153: C $ LEMAC(2,1) : NU*FISSION
154: C $ LEMAC(3,1) : fission
155: C $ LEMAC(4,1) : elastic scattering
156: C $ LEMAC(5,1) : capture
157: C $ LEMAC(6,1) : inelastic scattering
158: C $ LEMAC(7,1) : (n,2n)
159: C $ LEMAC(8,1) : loss
160: C
161: C < LEMAC(J,2) J=1 to NEMAC >
162: C
163: C $ LEMAC(J,2,1): reaction ID (1-8 ABOVE) for J-th array index
164: C*
165: C*
166: C RMAC(NGROUP,NTREG,NEMAC,2) R8 Macroscopic reaction rate tally
167: C          (,,,1) sum of batch tallies
168: C          (,,,2) square sum of batch tallies
169: C RMACR(NTREG,NEMAC,2) R8 Group sum of "RMAC".
170: C          (,,,1) sum of batch tallies
171: C          (,,,2) square sum of batch tallies
172: C XMAC(NGROUP+NPKIND,NTREG,NEMAC,2) R8 Macroscopic cross section$
173: C          $ tally.
174: C          (,,,1) sum of batch tallies
175: C          (,,,2) square sum of batch tallies
176: C DMAC(NGROUP,NTREG) R8 temporary sum of RMIC to calculate$
177: C          $ XMAC.
178: C%%<2004/02/20:
179: C%%<2007/01/10:
180: C%%C RMACSM(NGP1+1,NGP1+1,NTREG,2,1:3) R8 Macroscopic reaction rate $
181: C%%C          $tally of scattering matrix summed over batches. (neutron)
182: C%%C          (,,,1,) sum of batch tallies
183: C%%C          (,,,2,) square sum of batch tallies
184: C RMACSM(NGP1+1,NGP1+1,NTREG,2,NFL+1,1:3) R8 Macroscopic reaction $
185: C          $rate tally of scattering matrix summed over batches. (neutron)
186: C          (,,,1,,) sum of batch tallies
187: C          (,,,2,,) square sum of batch tallies
188: C%%>
189: C XMACSM(NGP1+1,NGP1+1,NTREG,2,1:3) R8 Macroscopic probability $
190: C          $tally of scattering matrix. (neutron)
191: C          ----- currently, not allocated in intro2.f -----
192: C%%>
193: C%%<2005/07/20:
194: C*
195: C RMAMU(NGP1+1,NGP1+1,NTREG,3,1:3) R8 Macroscopic mubar tally of $
```

```
196: C          $scattering reaction summed over batches. (neutron)
197: C          (,,,1,) sum of batch tallies
198: C          (,,,2,) square sum of batch tallies
199: C          (,,,3,) sum of batch tallies for weight
200: C%%>
201: C*
202: C*==== MONITOR FOR TALLY OF CROSS SECTION =====
203: C*
204: C*WCXTY(NGROUP+NPKIND,NREG,NUC) R8 COUNTER OF PARTICLE CONTRIBUTING TO$
205: C WCXTY(NGROUP+NPKIND,NTREG) R8 Counter of histories$
206: C          $ contributing to cross section tally.
207: C*=====
208: C RSTR(NREG,NSTAL) R8 Special tally. (track length)
209: C          $accumulated during each batch and cleared after each batch.
210: C SRSTR(NTREG,NSTAL,2) R8 Special tally. (track length)
211: C          (,,1) sum of batch tallies
212: C          (,,2) square sum of batch tallies
213: C RSCL(NREG,NSTAL) R8 Special tally. (collision)
214: C          $accumulated during each batch and cleared after each batch.
215: C SRSCL(NTREG,NSTAL,2) R8 Special tally. (collision)
216: C          (,,1) sum of batch tallies
217: C          (,,2) square sum of batch tallies
218: C*          ( NSTAL : SEE SBANK )
219: C*=====
220: C*
221: C DNZON(NUC,NZONE,2) R4 Number density of each nuclide in each zone.
222: C
223: C          (1:NUC,,1) : NEUTRON (NUCLIDE NUMBER DENSITIES)
224: C          (1:NPATOM,,2) : PHOTON (ATOM NUMBER DENSITIES)
225: C
226: C*DNREG(NUC,NREG) R4 NUMBER DENSITY OF EACH NUCLIDE IN EACH REGION.
227: C DNREG(NUC,NTREG,2) R4 Number density of each nuclide in each region.
228: C
229: C          (1:NUC,,1) : NEUTRON (NUCLIDE NUMBER DENSITIES)
230: C          (1:NPATOM,,2) : PHOTON (ATOM NUMBER DENSITIES)
231: C*
```

src/mvp/INC/_PTALY

```
1: C
2: C ... contents of common /_PTALY/ is moved to /_PTALY0/ and
3: C GMVP includes the same file. (MVP 94.2 : Aug 1995)
4: C
5: C*
6: C* ... POINT DETECTOR SPECIFIC DATA (25 Nov, 1999) ...
7: C*
8: C ... local data pointer in multi processing
9: COMMON /PTALYD/ LFLPD, LSFLPD, LVFLPD, LREPD, LSREPD, LVREPD
10: C*
11: C FLPD(NGROUP,NPDET) R8 Flux -point estimator (batch) <JPTDT>
12: C SFLEP(NGROUP,NPDET) R8 Average or sum of 'FLPD' <JPTDT>
13: C VFLEP(NGROUP,NPDET) R8 Variance or square sum of 'FLPD' <JPTDT>
14: C REPD(NPDET,NRESP) R8 Reaction rate -point estimator <JRESP&JPTDT>
15: C SREPD(NPDET,NRESP) R8 Average or sum of 'REPD' <JRESP&JPTDT>
16: C VREPD(NPDET,NRESP) R8 Variance of 'REPD' <JRESP&JPTDT>
17: C*
```

src/mvp/INC/_PXSEC

```
1: C*
2: c##<2007/03/14:PN3:
3: c##C*.... FISSION WEIGHT .....
4: C*.... FISSION and PHOTONUCLEAR WEIGHT .....
5: c##>
6: C*
7: c##<2007/03/14:PN3:
8: c## COMMON /PXSEC/ EINCD, LWGTF, LWGTFI, LFKAI, LWGTP
9: COMMON /PXSEC/ EINCD, LWGTF, LWGTFI, LFKAI, LWGTP,
10: & LWGTPN,LPFCNR, NPFCN, LPPNBR,NPPNBR,LPMT,
11: & NPMT, LMONPNW,NMONPNW
12: c##>
13: REAL EINCD
14: C*
15: C EINCD R4 Incident energy for fission sources of the first$
16: C $ batch of eigenvalue problem.(default = 0.0253 eV)
17: c##<2007/03/14:PN3:
18: c##C WGTF(NREG) R4 Fission particle weight of each region. <JFISS>
19: c##C WGTFI(NREG) R4 1.0/WGTF. <JFISS>
20: C WGTF(NREG,2) R4 Fission particle weight of each region for prompt$
21: C $ and delayed neutron. <JFISS>
22: C WGTFI(NREG) R4 1.0/WGTF, for prompt. <JFISS>
23: c##>
24: C*
25: C WGTN(NREG) R4 Photon generation weight of each region. <JPHOT>
26: c##<2007/03/14:PN3:
27: C*
28: C WGTN(NMTPN,NUCPN,NREG) R4
29: C Secondary particle generation weight of each region,$
30: C $ nuclide and reaction type by photo-nuclear reaction.
31: C $ <JPHNU>
32: C*
33: C PFCNR(NUCPN+1,NREG) R4
34: C Probability of forced collision nuclides on photon$
35: C $ incident by regions. <JPHNU>
36: C NPFCN I4 Number of regions with forced collision nuclides.$
37: C $ <JPHNU>
38: C PPNBR(NUCPN) R4 Probability which photo-nuclear reaction is$
39: C $ forcedly selected in photon collision. <JPHNU>
40: C NPPNBR I4 Number of photonuclear nuclides given PPNBR. <JPHNU>
41: C*
42: C PMT(2,NUCPN,NREG) R4
43: C Probability for a specified reation type (mt) by$
44: C $ region and nuclide. <JPHNU>
45: C NPMT I4 Flag for PMT (0=not given, 1=given). <JPHNU>
46: C*
47: C MONPNWGT(4,NMONPNW) I4
48: C Monitoring information of weight by region, nuclide$
49: C $ and reaction in photonuclear reaction. <JPHNU>
50: C NMONPNW I4 number of monitoring data sets. <JPHNU>
51: c##>
52: C*
53: C*=====
```

src/mvp/INC/_REACC

```
1: C*..... REACTION .....
2:   CHARACTER*12 REAC(8)
3:   DATA REAC /
4:   & '   TOTAL    ',' NU*FISSION ',' FISSIION  ',' ELASTIC SC ','
5:   & '   CAPTURE  ',' INELASTIC SC',' (N,2N)    ','   LOSS   ' /
```

SAFE

src/mvp/INC/_SBANK

```

1: C*
2: C*... BANK POINTERS FOR CROSS SECTION AND OTHER DATA RELATED TO
3: C* CONTINUOUS ENERGY MONTE CARLO CALCULATION.
4: C*
5: COMMON /SBANK/ MB,NSMAC,NSMIC,
6: & LSMAC,LLMAC,LKMAC,LMMAC,LSMIC,LLMIC,LKSPI,LSGTAL,
7: & NSMACI,NSMICI,
8: & LSMACI,LLMACI,LKMACI,LMMACI,LSMICI,LLMICI,LKSPII,LSGTALI,LRNU
9: c+pert
10: & ,LSMACP,LSMICP,LMMACP
11: c-pert
12: c##<2007/03/14:PN3:
13: & ,NSMICPN,LSMICPN
14: c##>
15: C*
16: C*
17: C***** MACRO CROSS SECTION AND RELATED DATA *****
18: C*
19: C MB I4 Maximum number of materials whose macro data are
20: C $ stored in a particle path.
21: C NSMAC I4 Number of macro reactions stored in macro $
22: C $cross section bank "SMAC" (neutron)
23: C*
24: C $ =1 : no eigenvalue problem (total only)
25: C $ =3 : eigenvalue problem (total,nu*fission,loss)
26: C*
27: C SMAC(NBANK,MB,NSMAC) R4 Macro cross section bank. (neutron)
28: C LMAC(8) I4 Position indices of reactions in SMAC
29: C
30: C $ LMAC(1) total.
31: C $ LMAC(2) nu*fission
32: C $ LMAC(3) fission
33: C $ LMAC(4) elastic scattering
34: C $ LMAC(5) capture
35: C $ LMAC(6) inelastic scattering
36: C $ LMAC(7) (n,2n)
37: C $ LMAC(8) loss
38: C
39: C KMAC(NBANK,MB,2) I4 Banked data for collision reaction with a$
40: C $material;
41: C $ (,,1) Nuclide # with which the particle
42: C $ may collide.
43: C $ (,,2) Material # whose macro data are stored.
44: C
45: C MMAC(NBANK,2) I4 Banked data to access data in macro cross$
46: C $ section bank.
47: C
48: C $ (,1) Particle is in a zone whose macro cross
49: C $ section is stored in smac(I,MMAC(I,1),K)
50: C $ (K=1,3). Set to zero if particle are in a void-zone.
51: C $ (,2) Particle can use MMAC(I,2) set of macro
52: C $ data of materials (whose cross section data
53: C $ are already calculated and stored in SMAC).
54: C*
55: C***** MICRO CROSS SETION AND RELATED DATA *****
56: C*
57: C NSMIC I4 Number of micro reactions stored in micro $
58: C $cross section bank "SMIC". (neutron)
59: C
60: C $ 5 : no eigenvalue problem
61: C $ 6 : eigenvalue problem (no tally for inelastic & (n,2n))
62: C $ 7 : no eigenvalue problem (tally for inelastic & (n,2n))
63: C $ 8 : eigenvalue problem (tally for inelastic & (n,2n))
64: C
65: C LMIC(8) I4 Position indexes of each reaction in SMIC.

```

```

66: C
67: C $ *Neutron
68: C $ LMIC(1) total.
69: C $ LMIC(2) nu*fission
70: C $ LMIC(3) fission
71: C $ LMIC(4) elastic scattering
72: C $ LMIC(5) capture
73: C $ **** OPTIONAL ****
74: C $ LMIC(6) inelastic scattering
75: C $ LMIC(7) (n,2n)
76: C $ LMIC(8) loss
77: C $
78: C $ *Photon
79: C $ LMIC(1) total.
80: C $ LMIC(2) coherent scattering
81: C $ LMIC(3) incoherent scattering
82: C $ LMIC(4) pair production
83: C $ LMIC(5) photo electric reaction
84: C $ **** unused ****
85: C $ LMIC(6:8)
86: C*
87: C SMIC(NBANK,NUC,NSMIC) R4 Micro cross section bank. (neutron)
88: C*
89: C KSPI(NBANK,NUC) I4 Banked energy grid position of current energy $
90: C $ for each particle and nuclide.
91: C*
92: C SGTAL(NBANK,NSTAL) R4 Banked cross setions for special tally $
93: C $ NSTAL is given in shared/INC/_SIZES.
94: c##<2007/03/14:PN3:
95: C*
96: C NSMICPN I4 Number of micro reactions stored in micro cross section $
97: C $ bank "SMICPN". (photonuclear)
98: C SMICPN(NBANK,NUCPN,NSMICPN) R4 Micro cross section bank. $
99: C $ (photonuclear)
100: C*
101: C RNU(NBANK,NUC_MAX,3) R4
102: c##>
103: C*
104: C*---- Bank for imaginary particles for next event estimator -----
105: C SMACI(IMPMAX,MB) R4 Macro total cross section bank. (neutron)
106: C KMACI(IMPMAX,MB) I4 Banked data for collision reaction with a$
107: C $material;
108: C $ Material # whose macro data are stored.
109: C
110: C MMACI(IMPMAX,2) I4 Banked data to access data in macro cross$
111: C $ section bank.
112: C
113: C $ (,1) Partcle is in a zone whose macro cross
114: C $ section is stored in smaci(I,MMAC(I,1))
115: C $ Set to zero if particle are in a void-zone.
116: C $ (,2) Partcle can use MMACI(I,2) set of macro
117: C $ data of materials (whose cross section data
118: C $ are already calculated and stored in SMACI).
119: C SMICI(IMPMAX,NUC) R4 Micro total cross section bank. (neutron)
120: C
121: C SGTALI(IMPMAX,NSTAL) R4 Banked cross setions for special tally $
122: C $ NSTAL is given in shared/INC/_SIZES.
123: C*=====

```

src/mvp/INC/_SIZES2

```
1: C*
2: C* ... MVP specific parameters ....
3: C*
4: C**** COMMON /SIZES2/ NPKIND, BANKP
5:     COMMON /SIZES2/ BANKP
6:     REAL     BANKP
7: C*
8: C* --- NPKIND is moved to common /SIZE/ (Aug 1995)
9: C* NPKIND I4  Number of particle species in calculation.
10: C*      (    --> JNEUT + JPHOT )
11: C*
12: C BANKP  R4  Generation probability of photon if particles bank has $
13: C      $insufficient space (  BANKP => PHOTON    1-BANKP => NEUTRON )
14: C
```

src/mvp/INC/_STACK

```
1: C*
2: C*... PARTICLE STACK POINTERS & LENGTHS
3: C*
4: COMMON /STACK/
5: & LLSFFL, LNFFL, LIZFFL, LLSCOL, LLSSRC, LNNXT, LIZNXT,
6: & LLSDED, LLSREF, LIZREF, LISREF, LNBREF,
7: & LLSLAT, LIZLAT, LNLXT,
8: & LLSCLP,
9: & LIMSFL, LIMNFL, LIMZFL, LIMSNX, LIMNNX, LIMZNX,
10: & LIMSLT, LIMNLT, LIMZLT, LIMDED
11: C
12: C ... task local scalar variables ...
13: C
14: C/IF PARA($X*)
15: * LOCAL COMMON /LSTACK/
16: C/ELSEIF PARA(CRAY)
17: * TASK COMMON /LSTACK/
18: C/ELSE
19: COMMON /LSTACK/
20: C/ENDIF
21: & NCOLS, NCOLP, IDEFER, NDEAD, NFISB,
22: & NIMPT, NDIMPT, IMDEFR
23: C*
24: C* **** FREE FLIGHT STACK ****
25: C*
26: C LSFFL(NBANK) I4 Free flight stack (bank index of free-flight $
27: C $ particles).
28: C NFFL(NZONE+1) I4 Number of particles waiting for free-flight for$
29: C $ each zone. (NFFL(NZONE+1) is the total number)
30: C IZFFL(NBANK) I4 Zone # of particles in free flight stack.
31: C*
32: C* **** COLLISION STACK ****
33: C*
34: C LSCOL(NBANK) I4 Collision stack (bank index of collision particles).
35: C NCOLS I4 Number of particles in collision stack.
36: C*
37: C* **** SEARCH (NEXT ZONE SEARCH) STACK ****
38: C*
39: C LSSRC(NBANK) I4 Nest zone search stack (bank index of particles $
40: C $searching next zone).
41: C NNXT(NZONE+1) I4 Number of particles waiting for next-zone search$
42: C $ for each zone. (NNXT(NZONE+1) is the total number)
43: C IZNXT(NBANK) I4 Zone # of particles in next-zone-search stack.
44: C IDEFER I4 Number of "deferred" particles for $
45: C $next-zone-search. A "deferred" particle is a$
46: C $ particle whose next-zone is not found in $
47: C $memorized zones (see "KMEMO") and all possible$
48: C $ zones should be checked for it.
49: C*
50: C* **** MORGUE (DEAD PARTICLE STACK) ****
51: C*
52: C LSDDED(NBANK) I4 Dead particle stack (bank index of dead particles).
53: C NDEAD I4 Number of particles in dead particle stack.
54: C*
55: C***** FISSION BANK ****
56: C*
57: C NFISB I4 Number of particles in fission particle bank.
58: C*
59: C***** REFLECTION STACK ****
60: C*
61: C LSREF(NBANK) I4 Reflection stack (bank index of particles to be $
62: C $reflected).
63: C IZREF(NBANK) I4 Zone # to return after reflection of particles$
64: C $ reflection stack.
65: C ISREF(NBANK) I4 Reflection surface # for each particle$
```

```
66: C $ reflection stack.
67: C NBREF(NREFS+1) I4 Number of particles waiting for reflection$
68: C $ for each reflection surface.
69: C*
70: C***** LATTICE STACK ****
71: C*
72: C LSLAT(NBANK) I4 Lattice-search stack (bank index of particles waiting$
73: C $ for lattice-search).
74: C IZLAT(NBANK) I4 Lattice-buffer zone # for particles in$
75: C $ lattice-search stack.
76: C NXLT(NLBZ+1) I4 Number of particles waiting for lattice-search$
77: C $ for each lattice-buffer-zone.
78: C*
79: C*- NOV 22 1991 -----
80: C*
81: C* **** PHOTON COLLISION STACK ****
82: C*
83: C LSLCP(NBANK) I4 Photon collision stack (bank index of photons$
84: C $ waiting for collision).
85: C NCOLP I4 Number of particles in photon-collision stack.
86: C*
87: C* **** Imaginary particle STACK (25 Nov,1999) ****
88: C*
89: C NIMPT I4 Number of imaginary particles
90: C NDIMPT I4 Number of non-active imaginary particles
91: C
92: C IMSFL(IMPMAX) I4 Flight stack for imaginary particles.
93: C IMNFL(NZONE+1) I4 Number of imaginary particles in flight stack.
94: C IMZFL(IMPMAX) I4 Zone #'s of imaginary particles in flight stack.
95: C
96: C IMSNX(IMPMAX) I4 Next-zone-search stack
97: C IMNNX(NZONE+1) I4 Number of imaginary particles in search stack.
98: C IMZNX(IMPMAX) I4 Zone #'s of imaginary particles in search stack.
99: C IMDEFR I4 Number of deferred particles in search stack$
100: C $ for imaginary particles.
101: C IMSLT(IMPMAX) I4 Lattice-search stack
102: C IMNLT(NLBZ+1)) I4 Number of imaginary particles in lattice search$
103: C $ stack.
104: C IMZLT(IMPMAX) I4 Zone #'s of imaginary particles in lattice-search$
105: C $ stack.
106: C*
107: C IMDED(IMPMAX) I4 Stack for non-active imaginary particles
108: C*
```


src/mvp/INC/_WKC2A

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR CEL2AB  (July 1997)
3: C*
4:      COMMON /WKC2A/  PCA(10)
5:      INTEGER      PCA
6: C*
7: C PCA(10) I4  Pointers to working arrays for 'CEL2AB' routine.
```

WKC2A

src/mvp/INC/_WKCOL

```
1: C*<MVP>
2: C*..... POINTERS TO WORKING ARRAYS : NEUTR (COLISN)  NOV/14/1991
3: C*
4:      COMMON /WKCOL/  P3(50)
5:      INTEGER      P3
6: C*
7: C P3(30) I4 Pointers to working arrays for 'NEUTR' routine.
```

SAFE

src/mvp/INC/_WKFL1

```
1: C*<MVP>
2: C* ..... POINTER TO WORKING ARRAYS : FLIONE (FLION0)    NOV/14/91
3: C*
4:      COMMON /WKFL1/ P0(50)
5:      INTEGER   P0
6: C* array size from 30 to 50 (Nov 1998)
7: C P0(50) I4  Pointers to working arrays for 'FLIONE' routine.
```

SAFE

src/mvp/INC/_WKFLA

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : FLIGHT
3: C*
4:      COMMON /WKFLA/ P6(30)
5:      INTEGER   P6
6: C*
7: C P6(30) I4 Pointers to working arrays for 'FLIGHT' routine.
```

WFLA

src/mvp/INC/_WKFSS

```
1: C*<MVP>
2: C*  .... Pointers to working arrays for FISSET (Feb 2000)
3: C*
4:      COMMON /WKSOU/  P10(10)
5:      INTEGER      P10
6: C*
7: C P10(10) I4 Pointers to working arrays for 'FISSET' routine.
```

SAFE

src/mvp/INC/_WKLAT

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : LATIC
3: C*
4:      COMMON /WKLAT/ P5(20)
5:      INTEGER    P5
6: C*
7: C P5(20) I4 Pointers to working arrays for 'LATIC' routine.
```

WKLAT

src/mvp/INC/_WKMIR

```
1: C*
2: C*.... POINTERS TO WORKING ARRAYS : MIRROR
3: C*
4:      COMMON /WKMIR/  P4(30)
5:      INTEGER      P4
6: C*
7: C P4(30) I4   Pointers to working arrays for 'MIRROR' routine.
```

SAFE

src/mvp/INC/_WKNXT

```
1: C*
2: C* .... POINTERS TO WORKING ARRAYS : NXTEE CALLED IN ACTION
3: C*
4:      COMMON /WKNXT/ PC(50), PF(50), PS(20), PL(20), PG(50)
5:      INTEGER      PC, PF, PS, PL, PG
6: C*
7: C PC(50) I4   Pointers to working arrays : NXTNR  called in ACTION
8: C PF(50) I4   Pointers to working arrays : NEEFLI called in NXTEE
9: C PS(20) I4   Pointers to working arrays : NEESEA called in NXTEE
10: C PL(20) I4   Pointers to working arrays : LATICE called in NXTEE
11: C PG(50) I4   Pointers to working arrays : NXPFR  called in ACTION
12: C*
```


src/mvp/INC/_WKPHT

```
1: C*<MVP>
2: C*..... POINTERS TO WORKING ARRAYS : PHOTR  NOV/14/1991
3: C*
4: c##<2007/03/14:PN3:PN4:
5: c##      COMMON /WKPHT/ PP(30)
6:      COMMON /WKPHT/ PP(50)
7: c##>
8:      INTEGER      PP
9: C*
10: c##<2007/03/14:PN3:PN4:
11: c##C PP(30) I4 Pointers to working arrays for 'PHOTR' routine.
12: C PP(50) I4 Pointers to working arrays for 'PHOTR' routine.
13: c##>
```

src/mvp/INC/_WKSE1

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEAOE .....
3: C*
4:      COMMON /WKSE1/ P2(30)
5:      INTEGER      P2
6: C*
7: C P2(30) I4  Pointers to working arrays for 'SEAOE' routine.
```

SEA

src/mvp/INC/_WKSEA

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEARCH
3: C*
4:      COMMON /WKSEA/ P7(30)
5:      INTEGER      P7
6: C*
7: C P7(30) I4  Pointers to working arrays for 'SEARCH' routine.
```

SEARCH

src/mvp/INC/_WKSOU

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR SOURCE   NOV/14/91
3: C*
4:      COMMON /WKSOU/  P1(30)
5:      INTEGER      P1
6: C*
7: C P1(30) I4 Pointers to working arrays for 'SOURCE' routine.
```

SAFE

src/mvp/INC/_WKTLO

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR TALLYO   NOV/14/91
3: C*
4:      COMMON /WKTLO/  P9(40)
5:      INTEGER      P9
6: C*
7: C P9(40) I4 Pointers to working arrays for 'TALLYO' routine.
```

SAFE

src/mvp/INC/_WKTLS

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR TALSUM   NOV/14/91
3: C*
4:      COMMON /WKTLS/  P8(30)
5:      INTEGER      P8
6: C*
7: C P8(30) I4  Pointers to working arrays for 'TALSUM' routine.
```

SAFE

src/mvp/INC/_XBANK

```

1: C*
2: C*... PARTICLE BANK POINTERS
3: C*
4:     integer MDBNK, MIBNK
5:     integer MDFBK, MIFBK
6: C     parameter ( MDBNK =6 , MIBNK = 8 )
7: C     parameter ( MDFBK =6 , MIFBK = 8 )
8: C ... May 1999 ... extended
9: C     index 11 to 16 is assinged to custom use ...
10:    parameter ( MDBNK =20 , MIBNK = 20 )
11:    parameter ( MDFBK =20 , MIFBK = 20 )
12:
13:    COMMON /XBANK/
14:    & LXXX,LYYY,LZZZ,LAAA,LBBB,LCCC,
15:    & LKKP, LWWW,LEEE,LIZZ,LIGG,LITT,LITT,
16:    & LKLSF, LLEVL, LLZZ, LLPOS, LIBSPC, LIBREG, LXIM,
17: C##<2007/03/14:PN4:
18: C## & LDBNK, KDBNK(0:MDBNK), LIBNK, KIBNK(0:MIBNK)
19: C## & LDBNK, KDBNK(0:MDBNK), LIBNK, KIBNK(0:MIBNK),
20: C## & LKPNFG
21: C##>
22: C*
23: C XXX(NBANK)      R8      X-position      (cm)
24: C YYY(NBANK)      R8      Y-position      (cm)
25: C ZZZ(NBANK)      R8      Z-position      (cm)
26: C AAA(NBANK)      R8      X-direction
27: C BBB(NBANK)      R8      Y-direction
28: C CCC(NBANK)      R8      Z-direction
29: C*** added KKP on Apr 2000
30: C KKP(NBANK)      I4      particle species ID
31: C                    (ID's are defined in INC/_KPIDS)
32: C WWW(NBANK)      R4      Particle weight
33: C EEE(NBANK)      R4      Energy (eV)
34: C IZZ(NBANK)      I4      Zone number
35: C IGG(NBANK)      I4      Energy group
36: C TTT(NBANK)      R8      Time (sec) <JTIME>
37: C ITT(NBANK)      R4      Time bin number <JTIME>
38: C KLSF(NBANK)      I4      Surface data pointer if particle is on surface
39: C***** OLD IMPORTANCE < JIMPT > *****
40: C XIM(NBANK)      R4      Importance of the zone in which each particle $
41: C                    $ was moving before entering current zone.
42: C                    $ For splitting/russian roulette with importance <JIMPT>.
43: C*
44: C***** LATTICE PARAMETER BANK < JLATT > *****
45: C*
46: C LEVL(NBANK)      I4      Current lattice level number.
47: C LZZ (NBANK,NEST) I4      Zone number to return.
48: C LPOS(NBANK,NEST) I4      Position number in lattice.
49: C LCRS(NBANK,NEST) I4      Flag to particles whose flight-track may cross$
50: C                    $ lattice-frame. (hexagonal lattice).
51: C***** frame-DEPENDENT-TALLY (TALLY-LATTICE) *****
52: C IBSPC(NBANK,0:NEST) I4 Subspace number for particles in bank.
53: C IBREG(NBANK)      I4      Region number for particles in bank.
54: C                    To be used in frame dependent tally.
55: C*
56: C***** Other optional bank parameters and indices *****
57: C* added July 1998
58: C*
59: C DBNK(NBANK,*) R8 optional bank parameters in floating point $
60: C                    $(double precision) form. See KDBNK,
61: C MDBNK I4 (parameter constant) number of optional bank parameters $
62: C                    $(DBNK) in floating point$
63: C                    $(double precision) form. Currently this is 5.
64: C KDBNK(0:MDBNK) I4 entry #'s of bank parameters in DBNK array.
65: C                    When KDBNK(i) is zero, no bank data are stored for i'th $

```

```

66: C                    $ property.
67: C*
68: C                    $ KDBNK(0) : total number of DBNK entries
69: C                    $ DBNK(*,KDBNK(1)) : particle weight on birth
70: C                    $ DBNK(*,KDBNK(2)) : time of birth
71: C                    $ DBNK(*,KDBNK(3)) : energy on birth
72: C                    $ DBNK(*,KDBNK(4):KDBNK(4)+NEST) : distance to lattice frames
73: C                    (required when lattice frames or cells may
74: C                    intersect each other. JFISX .ne. 0 )
75: C                    DBNK(*,KDBNK(4)+i-1) is distance to lattice
76: C                    frame of i'th level. i=NEST is reserved for
77: C                    some future extension.
78: C                    $ DBNK(*,KDBNK(5):KDBNK(5)+3*NEST-1) :
79: C                    Position of STG cell in coordinate system
80: C                    of STG region frame.
81: C                    $ DBNK(*,KDBNK(6):KDBNK(6)+6*NEST-1) :
82: C                    Position(1-3) on frame boundary to which particles
83: C                    should reach (not the current position) and
84: C                    flight direction(4-6) in frame.
85: C                    of each lattice level. Those of level 1 are
86: C                    identical to absolute corrodinates and direction.
87: C                    $ DBNK(*,KDBNK(7)+2*NTSRF) :
88: C                    distance and crossing angle cosine to tally-surface
89: C                    $ DBNK(*,KDBNK(7)+IS-1) : distance to tally-surface IS
90: C                    $ DBNK(*,KDBNK(7)+NTSRF+IS-1) : cosine of crossing angle.
91: C                    for normal vector to tally surface IS.
92: C                    Positive for inside to outside crossing.
93: C                    Negative for outside to inside crossing.
94: C
95: C                    $ Elements KDBNK(11) to KDBNK(16) are reserved for custom use by
96: C                    users. MVP/GMVP developpers doesn't assign any attributes
97: C                    to these indices.
98: C*
99: C IBNK(NBANK,*) I4 optional bank parameters in integer form.
100: C MIBNK I4 (parameter constant) number of optional bank parameters
101: C                    (IBNK) in integer form.
102: C KIBNK(0:MIBNK) I4 entry #'s of bank parameters in IBNK array.
103: C                    When KIBNK(i) is zero, no bank data are stored for i'th $
104: C                    $property.
105: C*
106: C                    $ KIBNK(0) : total number of IBNK entries
107: C                    $ IBNK(*,KIBNK(1)) : source set # on birth
108: C                    $ IBNK(*,KIBNK(2)) : region # on birth
109: C                    $ IBNK(*,KIBNK(3)) : zone # on birth
110: C                    $ IBNK(*,KIBNK(4)) : generation of particle
111: C                    $ (primary,secondary,tertiary,...)
112: C                    $ IBNK(*,KIBNK(5)) : flight count in a flight path.
113: C                    Required when lattice frames or cells may
114: C                    intersect each other. JFISX .ne.0. etc.
115: C                    Set -1 when particles are born from sources.
116: C                    $ IBNK(*,KIBNK(6):KIBNK(6)+NEST-1) : lattice level giving
117: C                    the smallest distance until the level
118: C                    $ IBNK(*,KIBNK(7)) : type of Nearest Neighbour Distribution (NND)
119: C                    if STG particle position sampling is
120: C                    required.
121: C                    $ 1 = on the surface of STG particle.
122: C                    $ 2 = from base material (matrix) region.
123: C                    $ 3 = on outer surface if STG region.
124: C                    $ 0 = no NND sampling is required.
125: C                    $ IBNK(*,KIBNK(8)) : save event # of last JDEBG(6) events for
126: C                    debug.
127: C***** defined in Sep 1999 *****
128: C                    $ IBNK(*,KIBNK(9)+NMKREG) : flag for "marker region" passing
129: C*
130: C                    $ IBNK(*,KIBNK(10)+2*NTSRF) : tally-surface side flag

```

src/mvp/INC/_XBANK

```

131: C                                and angle bin
132: C $ IBNK(*,KIBNK(10)+IS-1) : tally-surface (IS) side flag
133: C $ 0 : flight path do not cross the surface
134: C $ +1 : inside of surface
135: C $ and distance is calculated.
136: C $ -1 : outside of surface
137: C $ and distance is calculated.
138: C $ +2 : inside on surface but distance is
139: C $ not calculated yet.
140: C $ -2 : outside of surface but distance is
141: C $ not calculated yet.
142: C $ 3 : inside/outside is unknown
143: C $ and distance is not calculated.
144: C $ IBNK(*,KIBNK(10)+NTSRF+IS-1) : angle bin #
145: C $ for ANGLB(1:NANGLE+1)
146: C
147: C $ Elements KIBNK(11) to KIBNK(16) are reserved for custom use by
148: C users. MVP/GMVP developers doesn't assign any attributes
149: C to these indices.
150: C##<2007/03/14:PN3:
151: C
152: C $ IBNK(*,KIBNK(17)) : save region # on production by collision
153: C $ IBNK(*,KIBNK(18)) : save nuclide # on production by collision
154: C $ IBNK(*,KIBNK(19)) : save reaction # on production by collision
155: C##>
156: C c+beff1
157: C
158: C $ IBNK(*,KIBNK(20)) : flag for "beta effect"
159: C
160: C c-beff1
161: C##<2007/03/14:PN4:
162: C
163: C KPNFG(NBANK) I4 Flag of particle generated by photonuclear
164: C reaction (depend on "PHOTO-NUCLEAR" and "PNPRODUCE"
165: C option)
166: C##>
167: C*
168: C*=====
169: C*
170: COMMON /XBANKF/
171: & LXXXF,LYYF, LZZZF, LZZZF, LEEEF, LINUF, LKLSFF,
172: & LLEVLf, LLZZF, LLPOSF, LLCRS, LLCRSF,
173: & LIBSPF, LIBRGF,
174: & LDFBK, KDFBK(0:MDFBK), LIFBK, KIFBK(0:MIFBK),
175: c%%<Nagaya
176: c%% & LIFISB
177: & LIFISB, LIMTF
178: c%%>Nagaya
179: C*
180: C***** FISSON PARTICLE BANK *****
181: C XXXF(NFBANK) R8 X-position of banked fission particle (cm)
182: C YYYF(NFBANK) R8 Y-position of banked fission particle (cm)
183: C ZZZF(NFBANK) R8 Z-position of banked fission particle (cm)
184: C IZZF(NFBANK) I4 Zone number of banked fission particle
185: C EEEF(NFBANK) R4 Energy of fission reaction for banked fission$
186: C $ particle (eV)
187: C INUF(NFBANK) I4 Nuclide # which breeds banked fission particle.
188: c%%<Nagaya
189: C IMTF(NFBANK) I4 MT number for fission. 18=total,prompt 98=delayed
190: c%%>Nagaya
191: C KLSFF(NFBANK) I4 Surface data pointer if particle is on surface
192: C***** FISSON BANK ***** <JLATT> & <JFISS>
193: C LEVLf(NFBANK) I4 Current lattice level number$
194: C $ for banked fission particles.
195: C LZZF (NFBANK,NEST) I4 Zone number to return$

```

```

196: C $ for banked fission particles.
197: C LPOSF(NFBANK,NEST) I4 Position number in lattice$
198: C $ for banked fission particles.
199: C LCRSF(NFBANK,NEST) I4 Flag to particles whose flight-track may$
200: C $ cross lattice-frame. (hexagonal lattice)$
201: C $ for banked fission particles.
202: C***** frame-DEPENDENT-TALLY (TALLY-LATTICE) *****
203: C IBSPF(NFBANK,NEST) I4 Subspace number for banked fission particles.
204: C IBRGF(NFBANK) I4 region number for banked fission particles.
205: C To be used in universe dependent tally.
206: C*
207: C***** Nov 1998 : DFBK & IFBK is added ***
208: C*
209: C DFBK(NFBANK,*) R8 optional bank parameters in floating point $
210: C $ (double precision) form for banked fission particles.
211: C See KDFBK,
212: C MDFBK I4 (parameter constant) number of optional bank parameters $
213: C $ (DFBK) in floating point$
214: C $ (double precision) form for banked fission particles.
215: C Currently this is 5.
216: C KDFBK(0:MDFBK) I4 entry #'s of bank parameters in DFBK array.
217: C When KDFBK(i) is zero, no bank data are stored for i'th $
218: C $ property. See KDBNK(*) for description of parameters.
219: C IFBK(NFBANK,*) R8 optional bank parameters in integer $
220: C $ (double precision) form for banked fission particles.
221: C See KIFBK,
222: C MIFBK I4 (parameter constant) number of optional bank parameters $
223: C $ (IFBK) in nteger form for banked fission particles.
224: C Currently this is 5.
225: C KIFBK(0:MIFBK) I4 entry #'s of bank parameters in IFBK array.
226: C When KIFBK(i) is zero, no bank data are stored for i'th $
227: C $ property. See KIBNK(*) for description of parameters.
228: C* ... added 7 Feb 2000 ...
229: C IFISB(NHIST) I4 store index pointers to fission bank arrays for
230: C fission sources selected for a batch.
231: C*=====
232: C*
233: C* ... Copy of fission bank used for store fission sources when
234: C* NHSUB < NHIST
235: C*
236: C COMMON /XBANKF2/
237: C & LXXXF2,LYYF2, LZZZF2, LZZZF2, LEEEF2, LINUF2, LKLSFF2,
238: C & LLEVLf2, LLZZF2, LLPOSF2, LLCRS2, LLCRSF2,
239: C & LIBSPF2, LIBRGF2,
240: C & LDFBK2, KDFBK2(0:MDFBK), LIFBK2, KIFBK2(0:MIFBK)
241: C*=====
242: C*
243: C*..... IMAGINARY PARTICLES ...
244: C*
245: C COMMON /XIBANK/
246: C & LXXI, LYYI, LZZI, LAAAI, LBBBI, LCCCI,
247: C & LWWI, LEEI, LIZI, LIGGI, LTTI, LTTI,
248: C & LLEVLi, LLZZI, LLPOSi,
249: C & LLCRSI, LKLSFI, LKDETP, LOPTI, LPATH ,
250: C & LDBNKI, LIBNKI
251: C*
252: C*5/27/91 === BANK DATA FOR IMGINARY PATICLES FOR NEXT EVENT ESTIMATOR
253: C KDETP(IMPMAX) I4 Detector numbers for each imaginary particles.
254: C OPTI(IMPMAX) R8 Optical path for imaginary particles.
255: C PATH(IMPMAX) R8 Path length for imaginary particles.

```


src/mvp/INC/_XWORK

```
1: C*
2: C*..... VECTOR WORKING AREA POINTERS .....
3: C*
4:      COMMON /XWORK/ NWORK,LWORK,NWORKB,LWORKB
5: C*
6: C*NWORK      I4 NUMBER OF WORK AREA WHOSE LENGTH IS NBANK*4 BYTE.
7: C*WORK(NBANK,NWORK) R4/I4/R8  WORK AREA
8: C*NWORKB      I4 NUMBER OF WORK AREA WHOSE LENGTH IS (NZONE+1)*4
9: C*                                     BYTE.
10: C*WORKB(NZONE+1,NWORKB)  R4/I4/R8  WORK AREA
11: C*
12: C LWORK I4 Pointer of the beginning point of working area in variable$
13: C      $ length data area in common /ARRAY/.
```

src/gmvp/aaalloc.f

```
1:      subroutine AAALOC
2:      C
3:      C/#!/IF .NOT.DYNAMIC
4:      C
5:      C=<GMVP>=====
6:      C Purpose: memory size determination in non-dynamic (static) memory
7:      C           allocation mode.
8:      C
9:      C           In environment such as FACOM-M series or VP series,
10:     C users can set actual memory size as follows;
11:     C * copy this routine.
12:     C * modify size parameters (MAXMEM & LMAX)
13:     C * compile & link before execution.
14:     C
15:     C called in : MAIN
16:     C-----
17:     C
18:     C === size of shared memory (word) ==
19:     C
20:     C parameter (MAXMEM = 40000000)
21:     C
22:     C === size of local memory (word) for multitasking ==
23:     C
24:     C/#! IF PARA
25:     C * parameter (LMAX = 40000000)
26:     C/#!/ENDIF
27:     C
28:     C common /ARRAY/ LIMIT,ILDUM, A(MAXMEM)
29:     C common /ARRAYZ/ IAINFL(3)
30:     C
31:     C ... for single tasking mode, array A & H are equivalent.
32:     C
33:     C/#!/IF .NOT.PARA.OR.PARA(VPP)
34:     C real H(1)
35:     C integer IAINFL(3)
36:     C equivalence (LIMITL,LIMIT), (IAINF,IAINFL), (H,A)
37:     C/#!/ENDIF
38:     C
39:     C ... for multi tasking mode ....
40:     C
41:     C/#!/IF PARA( SX* )
42:     C * local common /LARRAY/ LIMITL, ILDUM, H(LMAX)
43:     C * local common /LARRAYZ/ IAINFL(3)
44:     C/#!/ELSEIF PARA( CRAY )
45:     C * task common /LARRAY/ LIMITL, ILDUM, H(LMAX)
46:     C * task common /LARRAYZ/ IAINFL(3)
47:     C/#!/ENDIF
48:     C
49:     C
50:     C -----
51:     C
52:     C LIMIT = MAXMEM
53:     C
54:     C/#! IF PARA( CRAY SX* )
55:     C * LIMITL = LMAX
56:     C/#! ENDF
57:     C
58:     C/#!/ENDIF
59:     C return
60:     C end
```

src/gmvp/acaloc.f

```
1:      subroutine ACALOC
2:      C
3:      C=<GMVP>=====
4:      C Purpose: memory size setting for character memory
5:      C called in : MAIN
6:      C-----
7:      C
8:      C Estimation of variable sized character area requirement for GMVP.
9:      C (November 1997)
10:     C
11:     C IZNAM(NINPZ)*12 : input zone name
12:     C KREGS(NREG)*(LNAM) : list of region/frame names
13:     C TNAMS(*)*(LNAM) : list of region/frame names
14:     C
15:     C LNAM=12 or 16 depending on system
16:     C
17:     C << required size (bytes) >>
18:     C
19:     C NINPZ*12 + NREG*LNAM + <number of names>*LNAM
20:     C
21:     C For a large problem (LNAM=16);
22:     C
23:     C #of input zone : 2000
24:     C #of region : 2000
25:     C #of names : 2000
26:     C
27:     C size is 88000 bytes = 22000 * 4 bytes
28:     C
29:     C So 40000*4 bytes may be sufficient for most problem, I hope.
30:     C
31:     C=====
32:     C
33:     C === size of character memory ( 4 bytes as a unit) ==
34:     C
35:     C PARAMETER (LCMAX = 400000)
36:     C
37:     C ... character data area (task shared) ....
38:     C
39:     C integer LIMITC,IDUMC
40:     C common /CARRAY/ CHA(LCMAX)
41:     C common /CARRAYZ/ LIMITC, IDUMC, IAINFC(3)
42:     C character*4 CHA
43:     C integer IAINFC
44:     C
45:     C ... character data area (task local) .... for future use
46:     C
47:     C integer LIMITC,IDUMC
48:     C character*4 CHH
49:     C common /CLARRAY/ CHH(1)
50:     C common /CLARRAYZ/ LIMITLC, ILDUMC, IAINFLC(3)
51:     C
52:     C
53:     C LIMITC = LCMAX
54:     C
55:     C IAINFC(1) = 1
56:     C IAINFC(2) = IAINFC(1)
57:     C IAINFC(3) = 0
58:     C
59:     C return
60:     C END
```

src/gmvp/action.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTION( TITLE, A, H, CHA )
3: C=<GMVP>=====
4: C  PURPOSE:  CONTROL OF MONTE CARLO RUN ( GMVP )
5: C  CALLED IN: CENTER
6: C  CALLS:  SOURCE,SELECT,TALSUM,FLIGHT,SEARCH,COLISN,MIRROR,
7: C          VMNTR0,VMNTR2,VMNTRF
8: C
9: C=====
10:   CHARACTER TITLE(2)*72
11:   real A(*)
12:   real H(*)
13:   character*4 CHA(*)
14: C
15:   include 'INC/_KPIDS'
16:   include 'INC/_NGPS'
17: C
18:   include 'INC/_FLAGS'
19:   include '../shared/INC/_SIZES'
20:   include 'INC/_SIZES2'
21:   include 'INC/_XBANK'
22:   include 'INC/_FBANK2'
23:   include 'INC/_STACK'
24:   include 'INC/_XWORK'
25:   include '../shared/INC/_PGEOM'
26:   include 'INC/_PXSEC'
27:   include '../shared/INC/_PVRED'
28:   include 'INC/_PSOUR'
29:   include '../shared/INC/_PTALY0'
30:   include 'INC/_PTALY'
31:   include '../shared/INC/_PTLSP'
32:   include '../shared/INC/_STALY'
33: C
34:   include '../shared/INC/_COUNTS'
35:   include '../shared/INC/_TIMEDT'
36: C
37: C/#IF PARA(PVM)
38:   include '../shared/INC/_PVMPARA'
39: C/#ELSEIF PARA(MPI)
40: *   include 'mpif.h'
41: C/#ELSEIF PARA(VPP)
42:   include '../shared/INC/_VPPPARA'
43: C/#ENDIF
44:   include '../shared/INC/_TASKDT'
45: C
46: C..... WORKING ARRAY POINTERS .....
47: C
48:   include 'INC/_WKSOU'
49:   include 'INC/_WKFSS'
50:   include 'INC/_WKFL1'
51:   include 'INC/_WKSE1'
52:   include 'INC/_WKCOL'
53:   include 'INC/_WKMIR'
54:   include 'INC/_WKLAT'
55:   include 'INC/_WKFLA'
56:   include 'INC/_WKSEA'
57:   include 'INC/_WKTLS'
58: C
59:   include 'INC/_WKNXT'
60:   include '../shared/INC/_IOUNIT'
61:   include 'INC/_IOUNIT2'
62: C
63: C ... local variables ...
64: C
65: C/#IF INTEGER8

```

```

66: *   integer*8 NTGEN
67: C/#ELSE
68:   integer   NTGEN
69: C/#ENDIF
70:   LOGICAL   JMEM
71: C
72:   real*8 DT, WSUMB
73: C
74: C-----
75: C
76: C   ... elapsed time & CPU time at startup
77: C
78:   call TOKEI(TE0,0)
79:   call CPUTM(T0)
80: C
81: C   =====
82:   call HEADER(IOW,'MONTE CARLO RUN')
83: C   =====
84: C
85: C   ... update special event counters ( COMMON /COUNTS/ )
86: C
87:   MTANG = 0
88:   MDEFR = 0
89: C   ... for endless loop detection ..
90:   DESUM = 0
91:   DESUM2 = 0
92:   JLOOP = 0
93:   NLOOP = 0
94: C   ... for FNSSC2 routine ..
95:   IFNSSC = 0
96: C
97: C   ... print batch monitor of sources , keff's if nonzero
98: C
99:   JBPRNT = 1
100: C
101: 7000 format(/'   PROBLEM TITLE   : ',A72/21X,A72)
102: C
103: C/#IF PARA(SX* CRAY*)
104: *   call MVPSYNC_LOCK(2)
105: C/#ENDIF
106: C
107:   write(IOW,/'/' TASK ' ',i5,' START ==== CPU TIME ' ',
108: &      E11.4,' (SEC) ELAPSED ==== ' ',e11.4,' (SEC)')')
109: &      IDTASK,T0,TE0
110: C
111: C/#IF PARA(SX* CRAY*)
112: *   call MVPSYNC_UNLOCK(2)
113: C/#ENDIF
114: C
115: C   ... synchronize all task for safety ....
116: C
117: C/#IF PARA(SX* CRAY*)
118: *   call MVPSYNC_BARRIER( 1 )
119: C/#ENDIF
120: C
121: C
122: C
123: C   .... INITIALIZATION .....
124: C
125: C
126: cccc  lh = iainfl(1)
127:      LH = 1
128: C
129: C/#IF ARGSAVE
130:

```

src/gmvp/action.f

```

131: C/# IF SOURCE(NEW)
132:
133:     call SOURC0(IOW,A,H,H,
134: N NTGEN, NDEAD, NFISB, JBPRNT, H(LNLOST), NMKREG,
135: 3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
136: 4 H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
137: & H(LYYYF), H(LZZZF),H(LIZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
138: & H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),H(LIBRGF),
139: & H(LIBSPF),H(LXIM),
140: & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
141: & H(LDBFK),KDBFK,MDBFK,H(LIFBK),KIFBK,MIFBK,
142: & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
143: & A(LKZDA),A(LKZAA),A(LIPCEL),A(LLTYP),A(LXIMP),H(LLSFFL),H(LNFFL),
144: & H(LIZFFL),H(LLSDED),H(LLSLAT),H(LIZLAT),H(LNXL),
145: A(A(LSOUR),H(LSRCSP),A(LFKAI),A(LKKAI),NTGX,
146: D H(LWSUM),H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGTF),A(LENGYB),
147: D A(LENGPE),A(LTIMEB),H(LXSXV),
148: E H(LXAVT),H(LAAVT),H(LAVT),
149: E A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK, MXREJ,
150: F H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
151: G H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)) )
152:
153:     call SOURCU( NPDET, NPLEN, NIMPT,NDIMPT, A(LJPUSD),
154: D A(LJSPDT), A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX),
155: S A(LVEL), H(LIMSFL), H(LIMNFL), H(LIMZFL),
156: S H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
157: B H(LLZZI), H(LLPOSI), H(LLCRSI),
158: B H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI) )
159:
160: C/# ELSE
161: *
162: *     call SOURC0(IOW,A,H,H,
163: * 1 IRAND,NTGEN,NBANK,NDEAD,NGROUP,NZONE,NFBANK,NFBANK0,NFISB,NEST,
164: * 2 JBPRNT,H(LNLOST),NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NMKREG,
165: * 3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
166: * 4 H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
167: * & H(LYYYF), H(LZZZF),H(LIZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
168: * & H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),
169: * & H(LIBRGF),H(LIBSPF),H(LXIM),
170: * & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
171: * & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
172: * 8 A(LKZDA),A(LKZAA),A(LIPCEL),A(LXIMP), H(LLSFFL),H(LNFFL),
173: * 9 H(LIZFFL),H(LLSDED),H(LLSLAT),H(LIZLAT),H(LNXL),
174: *
175: * * A(LKSOUR),A(LISZON),A(LSOUR),A(LPSAPC),A(LPENRG),A(LKENRG),
176: * * A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),A(LNSANG),A(LSANG),
177: * * A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),A(LFKAI),A(LKKAI),NTGX,
178: *
179: * D H(LWSUM),H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGTF),A(LENGYB),
180: * D NLENG,H(LXSXV),
181: * E H(LXAVT),H(LAAVT),H(LAVT),A(LLXYZ),
182: * F H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
183: * G H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)) )
184: *
185: *     call SOURCU( IMPMAX, NPDET,NPLEN, NTIME, NIMPT,NDIMPT,A(LJPUSD),
186: * D A(LJSPDT), A(LTIMEB),
187: * D A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX), A(LVEL),
188: * S H(LIMSFL), H(LIMNFL), H(LIMZFL),
189: * S H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
190: * B H(LLZZI), H(LLPOSI), H(LLCRSI),
191: * B H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI) )
192: *
193: C/# ENDIF
194:
195: C

```

```

196: C
197: C* if( JALLZ.eq.1 .or. JONEZ.eq.0 ) then
198: C* call FLIGH0( IOW,NBANK,NZONE,NSDA,NZDA, NGROUP, NTGX, NREG,
199: C* N NCOLS, NDEAD, H(LNLOST), IRAND, DINF, NEST, NLATT, NLBZ,
200: C* B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
201: C* B H(LWWW),H(LIGG),H(LTTT),H(LLEVL),H(LLZZ),H(LLPOS),
202: C* B H(LLCRS),H(LIBREG),
203: C* S H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSCOL),H(LLSSRC),H(LNNXT),
204: C* S H(LIZNXT),H(LLSDED),A(LSTOTX),A(LSNUFX),A(LSNAPX),
205: C* G A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LNKZAA),
206: C* G A(LDALT),A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),A(LKSLAT),
207: C* G A(LLTYP),A(LMLBZZ),A(LKLATT),A(LCELSZ),
208: C* T H(LFLTR),
209: C* T H(LNLEAK),H(LWLEAK),H(LNCNTR),H(LWCNTR),
210: C* W H(P6(1)),H(P6(2)),H(P6(3)),
211: C* W H(P6(4)),H(P6(5)),H(P6(6)),H(P6(7)),H(P6(8)),H(P6(9)),
212: C* W H(P6(10)),H(P6(11)),H(P6(12)),H(P6(13)),H(P6(14)),H(P6(15)),
213: C* W H(P6(16)),H(P6(17)),H(P6(18)),H(P6(19)),H(P6(20)),H(P6(21)),
214: C* W H(P6(22)),H(P6(23)),H(P6(24)),H(P6(25)),H(P6(26)),H(P6(27)),
215: C* W H(P6(28)),H(P6(29)),H(P6(30)) )
216: C
217: C* call SEARCO( IOW, NBANK,NZONE,NSDA,NZDA,NREFS, NGROUP,
218: C* N NREG, H(LNLOST), NDEAD,IRAND,NLBZ,NEST,NSPACE,NSUZON, DEPS,
219: C* B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
220: C* B H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LLEVL),H(LLZZ),
221: C* B H(LLPOS),H(LLCRS),H(LXIM),H(LIBREG),H(LIBSPC),
222: C* S H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSSRC),H(LNNXT),H(LIZNXT),
223: C* S H(LLSDED),H(LLSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LKSREF),
224: C* * H(LLSLAT),H(LIZLAT),H(LNXL),
225: C* G A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LNKZAA),
226: C* * A(LKCELL),A(LIPCEL),A(LMLBZZ),A(LISUSP),
227: C* T A(LXIMP),A(LWKIL),A(LWSRV),H(LNKILD),H(LWKILD),H(LNSURV),
228: C* T H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
229: C* W H(P7(1)),H(P7(2)),H(P7(3)),H(P7(4)),H(P7(5)),H(P7(6)),
230: C* W H(P7(7)),H(P7(8)),H(P7(9)),H(P7(10)),H(P7(11)),H(P7(12)),
231: C* W H(P7(13)),H(P7(14)),H(P7(15)),H(P7(16)),H(P7(17)),H(P7(18)),
232: C* W H(P7(19)),H(P7(20)),H(P7(21)),H(P7(22)),H(P7(23)),H(P7(24)),
233: C* W H(P7(25)),H(P7(26)),H(P7(27)),H(P7(28)),H(P7(29)),H(P7(30)) )
234: C
235: C* else if(JALLZ.eq.0.or.JONEZ.eq.1) then
236: C
237: *     call FLION0( IOW,IRAND,NGROUP,NTGX,NGP1,NGP2,
238: * N NBANK,NZONE,NSDA,NZDA,NCELL,NDEAD,NREG,NCOLS,H(LNLOST),
239: * N DINF,DEPS,NLATT,NUNV, NSPACE,NKTCSP,NEST,NLBZ,NTIME,TCUT,NRESP,
240: * B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIGG),
241: * B H(LTTT),H(LITT),H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LKLSF),
242: * B H(LIBSPC),H(LIBREG),H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
243: * S H(LLSFFL),H(LNFFL),H(LIZFFL),
244: * S H(LLSCOL),H(LLSSRC),H(LNNXT),H(LIZNXT),H(LLSDED),
245: * G A(LSDA),A(LKZMAT),A(LKZREG),A(LKSPSU),A(LKTCSP),A(LKZDA),
246: * G A(LKZAA),A(LDALT),A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),
247: * G A(LIPCEL),A(LKSLAT),A(LLTYP),A(LMLBZZ),A(LKZLBZ),
248: * G A(LKCELL),A(LICTYP),A(LKLATT),A(LCELSZ),A(LPNND),A(LKNND),NPNNND,
249: * G A(LPPP),A(LKPPF),NPPPF,
250: * T A(LJDTTR),A(LJDTAL),A(LJTEVE),A(LKTEVE),NTEVE,
251: * T A(LPSALP),A(LPSXYZ),H(LDTALY),A(LRESP),A(LSTOTX),A(LSNUFX),
252: * T A(LSNAPX),A(LTIMEB),A(LVEL),NMKREG,A(LMKREG),
253: * T NTSRF,A(LKTSRF),A(LITSRF),A(LIDSRF),A(LANGLB),NANGLE,
254: * T H(LFLTR),H(LTFLH),H(LNCNTR),H(LWCNTR) )
255: C
256: *     call SEAN0( IOW,A,CHA, NBANK,NZONE,NSDA,NZDA,IDEFER,H(LNLOST),
257: * N NDEAD,NREFS,NGROUP,NGP1,NREG,NTIME,IRAND,
258: * N NLBZ,NEST,NSUZON,NSPACE,DEPS,
259: * B NMKREG,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
260: * B H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LLEVL),H(LLZZ),

```

src/gmvp/action.f

```

261: *   B H(LLPOS ), H(LLCRS ), H(LXIM ), H(LKLSF ), H(LIBREG), H(LIBSPC),
262: *   B H(LDBNK), KDBNK, MDBNK, H(LIBNK), KIBNK, MIBNK,
263: *   S H(LLSFFL), H(LNFFL ), H(LIZFFL), H(LLSSRC), H(LNNXT ), H(LIZNXT),
264: *   S H(LLSDED), H(LLSREF), H(LISREF), H(LIZREF), H(LNBREF), A(LKSREF),
265: *   S H(LLSLAT), H(LIZLAT), H(LNXLT ),
266: *   G A(LSDA ), A(LKZMAT), A(LKZREG), A(LKZDA ), A(LKZAA ), A(LKSFB),
267: *   G A(LKCELL), A(LIPCEL), A(LMLBZZ), A(LISUSP), A(LMKREG),
268: *   T A(LXIMP), A(LWKIL), A(LWSRV), A(LWTIME), WLLIM,
269: *   T H(LNKILD), H(LWKILD), H(LNSURV),
270: *   T H(LWSURV), H(LNSPLT), H(LWSPLT), H(LNCNTR), H(LWCNTR),
271: *   T A(LJDTRG), A(LIDTAL), A(LJTEVE), A(LLTEVE), A(LKTEVE), NTEVE,
272: *   T H(LDTALY), A(LRESP),
273: *   W H(P2(1) ), H(P2(2) ), H(P2(3) ), H(P2(4) ), H(P2(5) ), H(P2(6) ),
274: *   W H(P2(7) ), H(P2(8) ), H(P2(9) ), H(P2(10)), H(P2(11)), H(P2(12))
275:
276: *   end if
277: C
278: C
279: *   call COLISO( IOW,
280: *   2 NGROUP, NGP1, NGP2, NZONE, NMAT, NREG, NTIME, NBANK, NFBANK, NFBANK0, NEST,
281: *   3 A(LKZMAT), A(LKZREG), A(LXIMP), A(LWKIL), A(LWSRV), A(LWTIME), WLLIM,
282: *   4 MXPGEN, A(LWGTF), A(LWGTFI), A(LFKAI), A(LKKAI), A(LWGTG), A(LWGTGI),
283: *   4 NGPX , NGGX , NTGX , NDSGX , NDSGX , NDSTX ,
284: *   5 NUSX , NSCTX , NCORFX , NMEDX ,
285: *   6 A(LANGX ), A(LNGSX ), A(LNNX ),
286: *   7 A(LSTOTX), A(LSSCX ), A(LSNAPX), A(LSGPX ), A(LSNEPX),
287: *   8 A(LSNUSX), A(LSGPBX), A(LSGPBX), A(LSSCBX), A(LSSCBX),
288: *   9 A(LSDBX), A(LSDBX), A(LSDPX), A(LS2PPX),
289: *   A A(LSANGX), A(LSANGX), A(LSGPPX), A(LSSCPX), A(LSPORT),
290: *   B IRAND ,
291: *   C H(LLEVL), H(LLZZ), H(LLPOS), H(LLCRS), H(LKLSF), H(LIBREG), H(LIBSPC),
292: *   D H(LLEVL), H(LLZZ), H(LLPOS), H(LLCRS), H(LKLSF), H(LIBREG), H(LIBSPC),
293: *   T NSTAL, NRESP, A(LRESP), NMKREG, A(LMKREG),
294: *   T A(LJDTRG), A(LIDTAL), A(LJTEVE), A(LLTEVE), A(LKTEVE), NTEVE,
295: *   T H(LDTALY) )
296: *
297: C/ENDIF
298:
299: C
300: C .... CHECK CPU TIMES FOR MONITOR ROUTINE CALL .....
301: C
302:   if( JVMNT.ne.0 ) then
303:     call MNTCPU(1)
304:     call MNTCPU(2)
305:     call MNTCPU(3)
306:     call MNTCPU(4)
307:     call MNTCPU(5)
308:     if( JREFL.ne.0 ) call MNTCPU(6)
309:     if( JLATT.ne.0 ) call MNTCPU(7)
310:   end if
311: C
312: C .... START MONTE CARLO RUN. GENERATE PARTICLES IN BANK .....
313: C   NTGEN: TOTAL NUMBER OF GENERATED PARTICLES IN THIS RUN
314: C   NGENE: GENERATED PARTICLES IN A BATCH
315: C   NTHIST: TOTAL NUMBER OF GENERATED PARTICLES IN SUCCESSIVE RUNS
316: C   (EFFECTIVE IN RESTRT RUN)
317: C
318: C
319:   JTERM   = 0
320: C
321: C-----
322: C IF NPART < OR = TO NTHIST, NO RANDOM WALK IN THIS RUN !
323: C-----
324: C
325:   if( NPART .le. NTHIST ) then

```

```

326:   write(IOW,7020) NPART,NTHIST
327: 7020   format(///1X ,10('%%%%%%%%')//
328:   &   1X , '   NUMBER OF HISTORIES TO BE RUN (NPART=',I10,') IS ',
329:   &   'LESS THAN THAT OF ALREADY RUN (NTHIST=',I10,') !!'//
330:   &   1X , '   NO RANDOM WALK IS PERFORMED!'
331:   &   /1X ,10('%%%%%%%%') )
332:   return
333: end if
334: C
335:   NTGEN   = NTHIST
336: C
337: C   ... NGENE : history to be generated in sub-batch
338: C   ... NGBAT : remaining history to be processed in a batch
339: C   (this should be task shared variable or synchronized on all
340: C   tasks in parallel mode)
341: C   ... NHIST1: histories of current batch (<= NHIST)
342: C   ... NDEAD : unused particle in bank
343: C
344:   NGENE   = 0
345:   NGBAT   = MIN( NPART - NTGEN, NHIST )
346:   NHIST1  = NGBAT
347:   NGENE0  = 0
348:   NDEAD   = NBANK
349:   WSUMB   = 0.0D0
350: C
351: C
352:   if(JALLZ.eq.0.or.JONEZ.eq.1) goto 2222
353: C
354: C *****
355: C ***** ALL-ZONE FLIGHT & SEARCH *****
356: C *****
357: C
358: C .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
359: C
360: C1000 continue
361: C   if(JVMNT.ne.0) call VMNTR0(2,IOW)
362: C   call SELALZ( MACT, NZONE, NBANK, NHIST, NTGEN, NPART ,NGENE,
363: C   &   H(LNFFL), H(LNNXT), NCOLS, NDEAD ,H(LNBREF),NREFS,JREFL,
364: C   &   JLATT , H(LNXLT), NLBZ , JTERM )
365: C   if(JVMNT.ne.0) call VMNTR2(2)
366: C
367: C .....
368: C   MACT = 0 : GENERATE NEXT BATCH OF PARTICLES.
369: C   = 1 : FREE FLIGHT
370: C   = 2 : NEXT ZONE SEARCH
371: C   = 3 : COLLISION
372: C   = 4 : REFLECTION
373: C   = 5 : LATTICE-SEARCH
374: C   = 7 : NEXT EVENT ESTIMATOR (JUNE 1991)
375: C
376: C   = 99 : END OF RUN
377: C .....
378: C
379: C   if(MACT.eq.0) then
380: C
381: C   .... TAKE SUMMATION OF TALLY ARRAYS EXCEPT FOR THE FIRST BATCH
382: C   (NBATCH IS INCREMENTED IN THIS ROUTINE.)
383: C   if(NGENE0.gt.0) then
384: C
385: C   call TALSUM(IOW, NPKIND, NBATCH, NSKIP, NGROUP, NBPINT, JBPRNT,
386: C   &   NREG, NRESP, NGENE0, NTRG, A(LIPTRG), A(LLPTRG), H(LWSUM), H(LWLEK),
387: C   &   H(LFLTR ), H(LSFLTR), H(LFLCL ), H(LSFLCL ),
388: C   &   A(LRESP ), H(LSRETR), H(LSRECL), H(LTFLH), H(LTFLHS),
389: C   &   H(LXSOC ), H(LXKEF ), H(LNCNTR), H(LWCNTR), NFISS ,
390: C   T H(LIETAL), NETALY, A(LIDTAL), NDTALY, H(LETALY), NLETAL,

```

src/gmvp/action.f

```

391: C      T H(LDTALY),NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
392: C      & H(P8(1)), H(P8(2)), H(P8(3)) )
393:
394: C      if( JPTDT.ne.0 .and. NPDET.gt. 0) then
395: C          call NEESUM(IOW, NBATCH, NSKIP, NGROUP ,
396: C      &      NPDET , NRESP , NGENE0 , A(LRESP) ,
397: C      &      H(LFLPD ), H(LSFLPD),
398: C      &      H(LREPD ), H(LSREPD) )
399: C      endif
400: C      endif
401: C
402: C      .... OPTIMIZATION OF KMEMO ARRAY .....
403: C
404: C      if(NBATCH.le.NMEMOP) then
405: C          call MEMMEM(
406: C      &      IOW, H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
407: C      end if
408: C
409: C      .... CHECK CPU TIME
410: C
411: C      CC/#IF PARA(SX*)
412: C          call PTCLOCK(DT)
413: C          T1 = DT
414: C      CC/#ELSEIF PARA(CRAY*)
415: C          call TSECND(T1)
416: C      CC/#ELSE
417: C          call CPUTM(T1)
418: C      CC/#ENDIF
419: C          NBT1 = NBATCH+1
420: C
421: C      CC/#IF PARA(SX* CRAY*)
422: C          call MVPSYNC_LOCK(2)
423: C      CC/#ENDIF
424: C
425: C      if ( JBPRNT.ne.0 ) then
426: C          write(IOW,7120) IDTASK, NBT1, T1
427: C          format('/ - TASK ',i5,' BATCH ',i4,
428: C      &      ' --- CPU ',E11.4,' (SEC) ---- ')
429: C      end if
430: C
431: C      CC/#IF PARA(SX* CRAY*)
432: C          call MVPSYNC_UNLOCK(2)
433: C      CC/#ENDIF
434: C      if(TCPU.ne.0.0.and.T1.gt.TCPU*60.0) then
435: C          write(IOW,7124) NPART,T1,TCPU
436: C          format(/2x,8('%%%%%%%%%'))
437: C      &      3x,'RUN TERMINATED BEFORE RUNNING ',i10,
438: C      &      ' HISTORIES, BECAUSE'/
439: C      &      2x,'CONSUMED CPU TIME (' ,E11.4,' SEC) ',
440: C      &      'EXCEEDED PROGRAM LIMIT (' ,E11.4,' MIN.).'/
441: C      &      2x,8('%%%%%%%%%')/
442: C      &
443: C          goto 9999
444: C      endif
445: C
446: C      .... SKIP RANDOM NUMBER IF NECESSARY .....
447: C
448: C      if( NRSKIP.lt.0 .or. (NBATCH.eq.0 .and. NRSKIP.gt.0) ) then
449: C          call RSKIP(NRSKIP, IRAND)
450: C      end if
451: C      Cccc &      CALL RSKIP(NRSKIP, 5*NBANK, A(LWORK), IRAND)
452: C
453: C      .... GENERATE PARTICLES .....
454: C
455: C          if(JVMNT.ne.0) call VMNTR0(1,IOW)

```

```

456: CC/#IF ARGSAVE
457: C*      call SOURCE( NGENE , NGENE0, NBATCH, JBPRNT )
458: CC/#ELSE
459:
460: CC/# IF SOURCE(NEW)
461:
462: C          call SOURCE(IOW,A,H,H,IRAND,NTGEN,NBANK,NDEAD,
463: C      1  NGROUP,NGP1,NGP2,NZONE,NFBANK,NFBNK0,NFISB,NEST,JBPRNT,H(LNLOST),
464: C      2  NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NTIME,TCUT,NMKREG,
465: C      3  H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
466: C      4  H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
467: C      &  H(LYYYF), H(LZZZF),H(LIZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
468: C      &  H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),
469: C      &  H(LIBRGF),H(LIBSPF),H(LXIM),
470: C      &  H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
471: C      &  A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
472: C      8  A(LKZDA),A(LKZAA),A(LIPCEL),A(LXIMP), H(LLSFFL),H(LNFFL),
473: C      9  H(LIZFFL),H(LLSD),H(LLSLAT),H(LIZLAT),H(LNXLT),
474: C      A  A(LSOUR),H(LSRCSP),A(LFKAI),A(LKKAI),NTGX,
475: C      D  H(LWSUM), H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGTF),A(LENGYB),
476: C      D  A(LENGPB),A(LTIMEB),NLENG,H(LXSXV),
477: C      E  H(LXAVT),H(LAAVT),H(LEAVT),
478: C      E  A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK, MXREJ,
479: C      F  H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
480: C      G  H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),
481: C      %  NPDET, NPLEN, NIMPT,NDIMPT,A(LJPUSD),A(LJSPDT),
482: C      D  A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX), A(LVEL),
483: C      S  H(LIMSFL), H(LIMNFL), H(LIMZFL),
484: C      S  H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
485: C      B  H(LLZZI), H(LLPOSI), H(LLCRSI),
486: C      B  H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI),
487: C      %  NGENE , NGENE0, NBATCH )
488:
489: CC/# ELSE
490:
491: C          call SOURCE(IOW,A,H,H,
492: C      1  IRAND,NTGEN,NBANK,NDEAD,NGROUP,NZONE,NFBANK,NFBNK0,NFISB,NEST,
493: C      2  JBPRNT,H(LNLOST),NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NMKREG,
494: C      3  H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
495: C      4  H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
496: C      &  H(LYYYF), H(LZZZF),H(LIZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
497: C      &  H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),
498: C      &  H(LIBRGF),H(LIBSPF),H(LXIM),
499: C      &  H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
500: C      &  A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
501: C      8  A(LKZDA),A(LKZAA),A(LIPCEL),A(LXIMP), H(LLSFFL),H(LNFFL),
502: C      9  H(LIZFFL),H(LLSD),H(LLSLAT),H(LIZLAT),H(LNXLT),
503: C      *  A(LKSOUR),A(LISZON),A(LSOUR),A(LPSPAC),A(LPENRG),A(LKENRG),
504: C      *  A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),A(LNSANG),A(LSANG),
505: C      *  A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),A(LFKAI),A(LKKAI),NTGX,
506: C      D  H(LWSUM),H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGTF),A(LENGYB),
507: C      D  NLENG,H(LXSXV),
508: C      E  H(LXAVT),H(LAAVT),H(LEAVT),A(LLXYZ),
509: C      F  H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
510: C      G  H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),
511: C      %  IMPMAX, NPDET, NPLEN, NTIME, NIMPT,NDIMPT,A(LJPUSD),A(LJSPDT),
512: C      D  A(LTIMEB), A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX), A(LVEL),
513: C      S  H(LIMSFL), H(LIMNFL), H(LIMZFL),
514: C      S  H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
515: C      B  H(LLZZI), H(LLPOSI), H(LLCRSI),
516: C      B  H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI),
517: C      %  NGENE , NGENE0, NBATCH )
518:
519: CC/# ENDIF
520:

```

src/gmvp/action.f

```

521: CC/#ENDIF
522: C
523: C          NTHIST = NTHIST + NGENE
524: C          JMEM   = NBATCH .lt. NMEMOP
525: C
526: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT .....
527: C
528: C      else if(MACT.eq.1) then
529: C
530: C          if(JVMNT.ne.0) call VMNTR2(1)
531: C      CC/#IF ARGSAVE
532: C      CC/#ELSE
533: C          call FLIGHT( IOW, NBANK, NZONE, NSDA, NZDA, NGROUP, NTGX, NREG,
534: C      N NCOLS, NDEAD, H(LNLOST), IRAND, DINF, NEST, NLATT, NLBZ,
535: C      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
536: C      B H(LWWW ), H(LIGG ), H(LTTT ), H(LLEVL ), H(LLZZ ), H(LLPOS ),
537: C      B H(LLCRS ), H(LIBREG),
538: C      S H(LLSFFL), H(LNFFL ), H(LIZFFL), H(LLSCOL), H(LLSRC ), H(LNNXT ),
539: C      S H(LIZNXT), H(LLSDED), A(LSTOTX), A(LSNUFUX), A(LSNAPX),
540: C      G A(LSDA ), A(LKZMAT), A(LKZREG), A(LKZDA ), A(LKZAA ), A(LNKZAA),
541: C      G A(LDALT ), A(LKDALT), A(LNVLAT), A(LSZLAT), A(LIPLAT), A(LKSLAT),
542: C      G A(LLTYP ), A(LMLBZZ), A(LKLATT), A(LCELSZ),
543: C      T H(LFLTR ),
544: C      T H(LNLEAK), H(LWLEAK), H(LNCNTR), H(LWCNTR),
545: C      W H(P6(1) ), H(P6(2) ), H(P6(3) ),
546: C      W H(P6(4) ), H(P6(5) ), H(P6(6) ), H(P6(7) ), H(P6(8) ), H(P6(9) ),
547: C      W H(P6(10)), H(P6(11)), H(P6(12)), H(P6(13)), H(P6(14)), H(P6(15)),
548: C      W H(P6(16)), H(P6(17)), H(P6(18)), H(P6(19)), H(P6(20)), H(P6(21)),
549: C      W H(P6(22)), H(P6(23)), H(P6(24)), H(P6(25)), H(P6(26)), H(P6(27)),
550: C      W H(P6(28)), H(P6(29)), H(P6(30)) )
551: C      CC/#ENDIF
552: C
553: C          if(JVMNT.ne.0) call VMNTR2(3)
554: C
555: C
556: C      else if(MACT.eq.2) then
557: C
558: C          if(JVMNT.ne.0) call VMNTR0(4,IOW)
559: C      CC/#IF ARGSAVE
560: C      CC/#ELSE
561: C          call SEARCH( IOW, NBANK, NZONE, NSDA, NZDA, NREFS, NGROUP,
562: C      N NREG, H(LNLOST), NDEAD, IRAND, NLBZ, NEST, NSPACE, NSUZON, DEFS,
563: C      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
564: C      B H(LWWW ), H(LIZZ ), H(LIGG ), H(LTTT ), H(LITT ), H(LLEVL ), H(LLZZ ),
565: C      B H(LLPOS ), H(LLCRS ), H(LXIM ), H(LIBREG), H(LIBSPC),
566: C      S H(LLSFFL), H(LNFFL ), H(LIZFFL), H(LLSRC ), H(LNNXT ), H(LIZNXT),
567: C      S H(LLSDED), H(LLSREF), H(LISREF), H(LIZREF), H(LNBREF), A(LKSREF),
568: C      * H(LLSLAT), H(LIZLAT), H(LNXLT ),
569: C      G A(LSDA ), A(LKZMAT), A(LKZREG), A(LKZDA ), A(LKZAA ), A(LNKZAA),
570: C      * A(LKCELL), A(LIPCEL), A(LMLBZZ), A(LISUSP),
571: C      T A(LXIMP ), A(LWKIL ), A(LWSRV ), H(LNKILD), H(LWSURV),
572: C      T H(LWSURV), H(LNSPLT), H(LWCNTR), H(LWCNTR),
573: C      W H(P7( 1)), H(P7( 2)), H(P7( 3)), H(P7( 4)), H(P7( 5)), H(P7( 6)),
574: C      W H(P7( 7)), H(P7( 8)), H(P7( 9)), H(P7(10)), H(P7(11)), H(P7(12)),
575: C      W H(P7(13)), H(P7(14)), H(P7(15)), H(P7(16)), H(P7(17)), H(P7(18)),
576: C      W H(P7(19)), H(P7(20)), H(P7(21)), H(P7(22)), H(P7(23)), H(P7(24)),
577: C      W H(P7(25)), H(P7(26)), H(P7(27)), H(P7(28)), H(P7(29)), H(P7(30)),
578: C      %
579: C          JMEM, NMEMO, H(LKMEMO), H(LMEMC), H(LMEMZ))
580: C      CC/#ENDIF
581: C
582: C          if(JVMNT.ne.0) call VMNTR2(4)
583: C
584: C      else if(MACT.eq.3) then
585: C
586: C          if(JVMNT.ne.0) call VMNTR0(5,IOW)

```

```

586: C
587: C      if( JPTDT.ne. 0) then
588: C
589: C          call NXTEC( IOW, A, H, H(LLSCOL),
590: C      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
591: C      B H(LWWW ), H(LIZZ ), H(LIGG ), H(LTTT ), H(LITT ),
592: C      B H(LLEVL ), H(LLZZ ), H(LLPOS ), H(LLCRS ),
593: C      B H(LDBNK ), KDBNK, MDBNK, H(LIBNK), KIBNK, MIBNK,
594: C      B KDBNK(0), KIBNK(0),
595: C      B H(LLZZI ), H(LLPOSI), H(LLCRSI), H(LNCNTR), H(LWCNTR),
596: C      B H(LOPTI ), H(LPATH ), H(LKDETP), H(LKLSFI),
597: C      X A(LANGX ), A(LNGSX ), A(LNNNX ), A(LSTOTX), A(LSGPBX), A(LSGPBX),
598: C      X A(LSSCBX), A(LSSCBX), A(LSDDBX), A(LSDDBX), A(LFKAI ), A(LKKAI ),
599: C      X A(LSPORT), A(LSPORT), A(LS2DPX), A(LS2PPX), A(LVEL ),
600: C      G A(LKZMAT), A(LKZREG), A(LTIMEB), A(LXPDET), A(LIPDET), A(LIPDT2),
601: C      D A(LJPUSD),
602: C      S H(LIMSFL), H(LIMNFL), H(LIMZFL),
603: C      S H(LIMDED), H(LIMSLT), H(LIMZLT), H(LIMNLT),
604: C      W H(PC(1) ), H(PC(2) ), H(PC(3) ), H(PC(4) ), H(PC(5) ),
605: C      W H(PC(6) ), H(PC(7) ), H(PC(8) ), H(PC(9) ), H(PC(10)),
606: C      W H(PC(11)), H(PC(12)), H(PC(13)), H(PC(14)), H(PC(15)),
607: C      W H(PC(16)), H(PC(17)), H(PC(18)), H(PC(19)), H(PC(20)),
608: C      W H(PC(21)), H(PC(22)), H(PC(23)), H(PC(24)), H(PC(25)),
609: C      W H(PC(26)), H(PC(27)) )
610: C
611: C      end if
612: C
613: C      CC/#IF ARGSAVE
614: C      CC/#ELSE
615: C          call COLISN(
616: C      C* 1 H(LLSFFL), H(LNFFL ), H(LIZFFL),
617: C      C* 2 H(LLSCOL), NCOLS, H(LLSDED), NDEAD,
618: C      C* 3 H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
619: C      C* 4 H(LWWW ), H(LIZZ ), H(LIGG ), H(LTTT ), H(LITT ), H(LXIM ),
620: C      C* B H(LDBNK ), KDBNK, MDBNK, H(LIBNK), KIBNK, MIBNK,
621: C      C* 4 H(LXXXF ), H(LYYYF ), H(LZZZF ), H(LIZZF ), NFISB,
622: C      C* 5 H(LFLCL ), H(LNCLSN), H(LWCLSN),
623: C      C* 6 H(LNABSB), H(LWABSB), H(LNECUT), H(LWECUT), H(LNKILD), H(LWKILD),
624: C      C* 7 H(LNSURV), H(LWSURV), H(LNSPLT), H(LWSPLT), H(LNCNTR), H(LWCNTR),
625: C      C* 8 H(P3(1) ), H(P3(2) ), H(P3(3) ), H(P3(4) ), H(P3(5) ), H(P3(6) ),
626: C      C* 9 H(P3(7) ), H(P3(8) ), H(P3(9) ), H(P3(10)), H(P3(11)), H(P3(12)))
627: C      CC/#ELSE
628: C          call COLISN( IOW,
629: C      2 NGROUP, NGP1, NGP2, NZONE, NMAT, NREG, NTIME, NBANK, NFBANK, NFBNK0, NEST,
630: C      3 A(LKZMAT), A(LKZREG), A(LXIMP), A(LWKIL), A(LWSRV), A(LWTIME), WLLIM,
631: C      4 A(LWGTG), A(LWGTGI), A(LFKAI), A(LKKAI), A(LWGTG), A(LWGTGI),
632: C      4 NGPX, NGGX, NTGX, NDSGX, NDSGX, NDSGX,
633: C      5 NUSX, NSCTX, NCOEFX, NMEDX,
634: C      6 A(LANGX ), A(LNGSX ), A(LNNNX ),
635: C      7 A(LSTOTX), A(LSSCX ), A(LSNAPX), A(LSGPX ), A(LSNFPX),
636: C      8 A(LSNUSX), A(LSGPBX), A(LSGPBX), A(LSSCBX), A(LSSCBX),
637: C      9 A(LSDDBX), A(LSDDBX), A(LS2DPX), A(LS2PPX),
638: C      A(LSANGX), A(LSANGX), A(LSGPPX), A(LSSCPX), A(LSPORT),
639: C      B IRAND,
640: C      C H(LLEVL ), H(LLZZ ), H(LLPOS ), H(LLCRS ), H(LKLSF), H(LIBREG), H(LIBSPC),
641: C      D H(LLEVL ), H(LLZZ ), H(LLPOS ), H(LLCRS ), H(LIBRGF), H(LIBSPF),
642: C      % H(LLSFFL), H(LNFFL ), H(LIZFFL),
643: C      2 H(LLSCOL), NCOLS, H(LLSDED), NDEAD,
644: C      3 H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
645: C      4 H(LWWW ), H(LIZZ ), H(LIGG ), H(LTTT ), H(LITT ), H(LXIM ),
646: C      B H(LDBNK ), KDBNK, MDBNK, H(LIBNK), KIBNK, MIBNK,
647: C      B H(LXXXF ), H(LYYYF ), H(LZZZF ), H(LIZZF ), NFISB,
648: C      5 H(LFLCL ), H(LNCLSN), H(LWCLSN),
649: C      6 H(LNABSB), H(LWABSB), H(LNECUT), H(LWECUT), H(LNKILD), H(LWKILD),
650: C      7 H(LNSURV), H(LWSURV), H(LNSPLT), H(LWSPLT), H(LNCNTR), H(LWCNTR),
651: C      8 H(P3(1) ), H(P3(2) ), H(P3(3) ), H(P3(4) ), H(P3(5) ), H(P3(6) ),

```


src/gmvp/action.f

```

651: C      9 H(P3(7) ), H(P3(8) ), H(P3(9) ), H(P3(10)), H(P3(11)), H(P3(12)))
652: C      CC/#ENDIF
653: C
654: C
655: C      .... REFLECTION .....
656: C
657: C      else if(MACT.eq.4) then
658: C
659: C          call MIRROR( JMNTR, JDEBG, JVMNT, JLATT,
660: C      N      NBANK, NREFS, NSDA, IRAND, NLBZ, NZONE,
661: C      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
662: C      B H(LWWW ), H(LIZZ ),
663: C      S H(LNQNTR), H(LWCNTR), H(LLSFLL), H(LNFFL ), H(LIZFLL),
664: C      S H(LLSRC), H(LNNXT ), H(LIZNXT),
665: C      S H(LLSREF), H(LISREF), H(LIZREF), H(LNBREF), A(LSDA ), A(LKKREF),
666: C      S H(LLSLAT), H(LIZLAT), H(LNXLT ), A(LMLBZZ),
667: C      W H(P4(1) ), H(P4(2) ), H(P4(3) ), H(P4(4) ),
668: C      W H(P4(5) ), H(P4(6) ), H(P4(7) ), H(P4(8) ), H(P4(9) ), H(P4(10)),
669: C      W H(P4(11)), H(P4(12)), H(P4(13)), H(P4(14)), H(P4(15)), H(P4(16)),
670: C      W H(P4(17)), H(P4(18)), H(P4(19)), H(P4(20)), H(P4(21)), H(P4(22)),
671: C      W H(P4(23)) )
672: C
673: C          if(JVMNT.ne.0) call VMNTR2(6)
674: C      .... LATTICE SEARCH .....
675: C
676: C      else if(MACT.eq.5) then
677: C
678: C          call LATTICE(IOW, JDEBG, JVMNT, JTLTT, JHLAT, H(LNLOST),
679: C      N      NBANK, NZONE, NLATT, NCELL, NLBZ, NEST, NSPACE, NUNV, DEPS,
680: C      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
681: C      B H(LIZZ ), H(LLEVL ), H(LLZZ ), H(LLPOS ), H(LLSDED), NDEAD,
682: C      B H(LLCRS ), H(LIBREG), H(LIBSPC),
683: C      S H(LLSRC), H(LIZNXT), H(LNNXT ), H(LLSLAT), H(LIZLAT), H(LNXLT ),
684: C      G A(LKZMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
685: C      G A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
686: C      G A(LLTYP ), A(LICTYP), A(LKDALT), A(LDALT ),
687: C      G A(LKZREG), A(LKSPSU), A(LKTCSP), NKTCSF, A(LKHLAT),
688: C      W H(P5( 1)), H(P5( 2)), H(P5( 3)), H(P5( 4)), H(P5( 5)), H(P5( 6)),
689: C      W H(P5( 7)), H(P5( 8)), H(P5( 9)), H(P5(10)) )
690: C          if(JVMNT.ne.0) call VMNTR2(7)
691: C
692: C      else if(MACT.eq.99) then
693: C
694: C      .... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR INITIAL BATCH
695: C          call TALSUM(IOW, NPKIND, NBATCH, NSKIP, NGROUP, NBPINT, JDEPRNT,
696: C      & NREG, NRESP, NGENE0, NTREG, A(LIPTRG), A(LLPTRG), H(LWSUM), H(LMLEK),
697: C      & H(LFLTR ), H(LSFLTR), H(LFLCL ), H(LSFLCL),
698: C      & A(LRESP ), H(LSRETR), H(LSRECL), H(LTFLH), H(LTFLHS),
699: C      & H(LXSOC ), H(LXKEF ), H(LNCNTR), H(LWCNTR), NFISB,
700: C      T H(LIETAL), NETALY, A(LIDTAL), NDTALY, H(LETALY), NLETAL,
701: C      T H(LDTALY), NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
702: C      & H(P8(1)), H(P8(2)), H(P8(3)) )
703: C
704: C          if( JPTDT.ne.0 .and. NPDET.gt. 0) then
705: C              call NEESUM(IOW, NBATCH, NSKIP, NGROUP,
706: C      & NPDET, NRESP, NGENE0, A(LRESP),
707: C      & H(LFLPD ), H(LSFLPD),
708: C      & H(LREPD ), H(LSREPD) )
709: C          end if
710: C
711: C      .... OPTIMIZATION OF KMEMO ARRAY .....
712: C          if(NBATCH.le.NMEMOP)
713: C              call MEMMEM(
714: C      & IOW, H(LKMEMO), H(LMEMC), H(LMEMZ), NZONE, NMEMO, NBATCH)
715: C          goto 9999

```

```

716: C      end if
717: C
718: C      goto 1000
719: C
720: C      *****
721: C      ***** ONE-ZONE FLIGHT & SEARCH *****
722: C      *****
723: C
724: C          call TOKEI(TE1,0)
725: C      C/#IF PARA(SX*)
726: C      *      call PTCLOCK(DT)
727: C      *      t1 = dt
728: C      C/#ELSEIF PARA(CRAY*)
729: C      *      call TSECND(T1)
730: C      C/#ELSE
731: C          call CPUTM(T1)
732: C      C/#ENDIF
733: C          NBT1 = NBATCH+1
734: C
735: C      C/#IF PARA(SX* CRAY*)
736: C      *      call MVPSYNC_LOCK(2)
737: C      C/#ENDIF
738: C
739: C          ... print message for starting batch ...
740: C
741: C          write(IOW,7060) IDTASK, NBT1, T1, TE1
742: C
743: C      C/#IF PARA(SX* CRAY*)
744: C      *      call MVPSYNC_UNLOCK(2)
745: C      C/#ENDIF
746: C
747: C      -----
748: C      .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
749: C      -----
750: C
751: C      2222 continue
752: C
753: C          if(JVMNT.ne.0) call VMNTR0(2,IOW)
754: C
755: C          call SELONE( MACT, JTERM, MZONE, NTGEN, NGENE, NGENE0, NGBAT,
756: C      & H(1), H(LNFFL), H(LNNXT), H(LNBREF), H(LNXLT),
757: C      & IMMODE, H(LIMNFL), H(LIMNNX), H(LIMNLT),
758: C      *      H(LNCNTR), H(LWCNTR), NEVENT )
759: C
760: C          if(JVMNT.ne.0) call VMNTR2(2)
761: C      ....
762: C      MACT = 0 : Generate next batch of particles.
763: C      = 1 : Free flight
764: C      = 2 : Next zone search
765: C      = 3 : Collision
766: C      = 5 : Reflection
767: C      = 6 : Lattice-search
768: C      = 7 : Next event estimator
769: C      = 88 : End of batch
770: C      = 99 : End of run
771: C      MZONE : Selected zone number (if zonewise logic)
772: C      -----
773: C
774: C          if( MACT.eq.0 ) then
775: C
776: C      -----
777: C      .... SKIP RANDOM NUMBER IF NECESSARY .....
778: C      -----
779: C          if ( NGENE0.eq.0 ) then
780: C              if( NRSKIP.lt.0 .or. (NBATCH.eq.0 .and. NRSKIP.gt.0) ) then

```

src/gmvp/action.f

```

781:          call RSKIP(NRSKIP, IRAND)
782:          end if
783:        end if
784: C-----
785: C ..... GENERATE PARTICLES .....
786: C-----
787:          if(JVMNT.ne.0) call VMNTR0(1,IOW)
788: C/IF ARGSAVE
789:
790: C/IF SOURCE(NEW)
791: *      call SOURCE( NGENE,NGENE0,NHIST1,NGBAT,H(LIFISB),
792: *      & WFFACT,WSUMB )
793: C/IF ELSE
794:
795: *      call SOURCE( NGENE,NGENE0,NBATCH,WSUMB )
796:
797: C/IF ENDF
798: C/ELSE
799:
800: C/IF SOURCE(NEW)
801:
802:          call SOURCE(IOW,A,H,H,
803: N NTGEN, NDEAD, NFISB, JBPRNT, H(LNLOST), NMKREG,
804: 3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
805: 4 H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
806: & H(LYYYF), H(LZZZF),H(LZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
807: & H(LLCRS),H(LLEVL),H(LZZZF),H(LLPOSF),H(LLCRSF),
808: & H(LIBRGF),H(LIBSPF),H(LXIM),
809: & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
810: & H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
811: & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
812: 8 A(LKZDA),A(LKZAA),A(LIPCEL),A(LLTYP),A(LXIMP),H(LLSF),H(LNFFL),
813: 9 H(LIZFFL),H(LLSDED),H(LLSLAT),H(LIZLAT),H(LNXLT),
814: A A(LSOUR),H(LSRCSP),A(LFKAI),A(LKKAI),NTGX,
815: D H(LWSUM), H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGT),A(LLENGYB),
816: D A(LLENGPB),A(LTIMEB),H(LXSXV),
817: E H(LXAVT),H(LAAVT),H(LEAVT),
818: E A(LLXYZ),A(LLPWRK),A(LLVSTK),MWVEC,MVSTK, MXREJ,
819: F H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
820: G H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),
821: % NPDET, NPLEN, NIMPT,NDIMPT,A(LJPUSD),A(LJSPDT),
822: D A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX), A(LVEL),
823: S H(LIMSFL), H(LIMNFL), H(LIMZFL),
824: S H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
825: B H(LLZZI), H(LLPOSI), H(LLCRSI),
826: B H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI),
827: % NGENE, NGENE0, NHIST1, NGBAT, H(LIFISB), WFFACT, WSUMB )
828:
829: C/IF ELSE
830:
831: *      call SOURCE(IOW,A,H,H,
832: *      1 IRAND,NTGEN,NBANK,NDEAD,NGROUP,NZONE,NFBANK,NFBNK0,NFISB,NEST,
833: *      2 JBPRNT,H(LNLOST),NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NMKREG,
834: *      3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LWWW),H(LIZZ),
835: *      4 H(LIGG),H(LTTT),H(LITT),H(LKLSF),H(LIBREG),H(LIBSPC),H(LXXXF),
836: *      & H(LYYYF), H(LZZZF),H(LZZF),H(LKLSFF),H(LLEVL),H(LLZZ),H(LLPOS),
837: *      & H(LLCRS),H(LLEVL),H(LLZZF),H(LLPOSF),H(LLCRSF),
838: *      & H(LIBRGF),H(LIBSPF),H(LXIM),
839: *      & H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
840: *      & A(LMLBZZ),A(LKZMAT),A(LKZREG),H(LMEMZN),A(LSDA),
841: *      8 A(LKZDA),A(LKZAA),A(LIPCEL),A(LLTYP), H(LLSF),H(LNFFL),
842: *      9 H(LIZFFL),H(LLSDED),H(LLSLAT),H(LIZLAT),H(LNXLT),
843:
844: *      A(LKSOUR),A(LISZON),A(LSOUR),A(LPSAPAC),A(LPENRG),A(LKENRG),
845: *      A(LNSTIM),A(LSTIM),A(LPSTIM),A(LISTIM),A(LNSANG),A(LSANG),

```

```

846: *      A(LSAXIS),A(LPSANG),A(LISANG),A(LIFISM),A(LFKAI),A(LKKAI),NTGX,
847:
848: *      D H(LWSUM),H(LXSOC),H(LNCNTR),H(LWCNTR),A(LWGT),A(LLENGYB),
849: *      D NLENG,H(LXSXV),
850: *      E H(LXAVT),H(LAAVT),H(LEAVT),A(LLXYZ),
851: *      F H(P1(1)),H(P1(2)),H(P1(3)),H(P1(4)),H(P1(5)),H(P1(6)),H(P1(7)),
852: *      G H(P1(8)),H(P1(9)),H(P1(10)),H(P1(11)),H(P1(12)),
853: *      % IMPMAX, NPDET, NPLEN, NTIME, NIMPT,NDIMPT,A(LJPUSD),A(LJSPDT),
854: *      D A(LTIMEB), A(LXPDET), A(LIPDET), A(LIPDT2), A(LSTOTX), A(LVEL),
855: *      S H(LIMSFL), H(LIMNFL), H(LIMZFL),
856: *      S H(LIMDED), H(LIMSLT), H(LIMNLT), H(LIMZLT),
857: *      B H(LLZZI), H(LLPOSI), H(LLCRSI),
858: *      B H(LOPTI), H(LPATH), H(LKDETP), H(LKLSFI),
859: *      % NGENE, NGENE0, NBATCH, WSUMB )
860:
861: C/IF ENDF
862:
863: C/ENDIF
864:
865:          if(JVMNT.ne.0) call VMNTR2(1)
866:          NTHIST = NTHIST + NGENE
867:          JMEM = NBATCH.lt.NMEMOP
868: C-----
869: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT .....
870: C-----
871:          else if( MACT.eq.1 ) then
872:          if(JVMNT.ne.0) call VMNTR0(3,IOW)
873: C/IF ARGSAVE
874: *      call FLIONE( MZONE,
875: *      W H(P0(1)), H(P0(2)), H(P0(3)), H(P0(4)), H(P0(5)), H(P0(6)),
876: *      W H(P0(7)), H(P0(8)), H(P0(9)), H(P0(10)), H(P0(11)), H(P0(12)),
877: *      W H(P0(13)), H(P0(14)), H(P0(15)), H(P0(16)), H(P0(17)), H(P0(18)),
878: *      W H(P0(19)), H(P0(20)), H(P0(21)), H(P0(22)), H(P0(23)), H(P0(24)),
879: *      W H(P0(25)), H(P0(26)), H(P0(27)), H(P0(28)), H(P0(29)), H(P0(30)),
880: *      W H(P0(31)), H(P0(32)), H(P0(33)), H(P0(34)), H(P0(35)), H(P0(36)),
881: *      W H(P0(37))
882: C/ELSE
883:          call FLIONE( IOW, IRAND,NGROUP,NTGX,NGP1,NGP2,
884: N NBANK,NZONE,NSDA,NZDA,NCELL,NDEAD,NREG,NCOLS,H(LNLOST),DINF,DEPS,
885: N NLATT,NUNV, NSPACE,NKTCSP,NEST,NLBZ,NTIME,TCUT,NRESP,
886: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
887: B H(LWWW),H(LIGG),H(LTTT),H(LITT),H(LLEVL),H(LLZZ),
888: B H(LLPOS),H(LLCRS),H(LKLSF),H(LIBSPC),H(LIBREG),
889: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
890: S H(LLSF),H(LNFFL),H(LIZFFL),
891: S H(LLSCOL),H(LLSSRC),H(LNNXT),H(LIZNXT),H(LLSDED),
892: G A(LSDA),A(LKZMAT),A(LKZREG),A(LKSPSU),A(LKTCSP),A(LKZDA),
893: G A(LKZAA),A(LDALT),A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),
894: G A(LIPCEL),A(LKSLAT),A(LLTYP),A(LMLBZZ),A(LKZLBZ),A(LKCELL),
895: G A(LICTYP),A(LKLATT), A(LCELSZ), A(LPNND),A(LKNND),NPNND,
896: G A(LPPPF),A(LKPPF), NPPPF,
897: T A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
898: T A(LPSALP),A(LPSXYZ),H(LDTALY),A(LRESP), A(LSTOTX),A(LSNUF),
899: T A(LSNAPX),A(LTIMEB),A(LVEL),NMKREG,A(LMKREG),
900: T NTSRF,A(LKTSRF),A(LITSRF),A(LIDSRF),A(LANGLB),NANGLE,
901: T H(LPLTR),H(LTFLH),H(LNCNTR),H(LWCNTR),
902: W H(P0(1)),H(P0(2)),H(P0(3)),H(P0(4)),H(P0(5)),H(P0(6)),
903: W H(P0(7)),H(P0(8)),H(P0(9)),H(P0(10)),H(P0(11)),H(P0(12)),
904: W H(P0(13)),H(P0(14)),H(P0(15)),H(P0(16)),H(P0(17)),H(P0(18)),
905: W H(P0(19)),H(P0(20)),H(P0(21)),H(P0(22)),H(P0(23)),H(P0(24)),
906: W H(P0(25)),H(P0(26)),H(P0(27)),H(P0(28)),H(P0(29)),H(P0(30)),
907: W H(P0(31)),H(P0(32)),H(P0(33)),H(P0(34)),H(P0(35)),H(P0(36)),
908: W H(P0(37)),
909: % MZONE )
910: C/ENDIF
911:
912:          if(JVMNT.ne.0) call VMNTR2(3)

```

src/gmvp/action.f

```

911: C-----
912: C .... FIND NEXT ZONE TO ENTER .....
913: C-----
914:       else if( MACT.eq.2 ) then
915:               if(JVMNT.ne.0) call VMNTR0(4,IOW)
916:
917: C/#IF ARGSAVE
918: *       call SEAONE( MZONE, JMEM, NMEMO,H(LKMEMO),H(LMEMC),H(LMEMZ))
919: C/#ELSE
920:       call SEAONE( IOW,A,CHA, NBANK,NZONE,NSDA,NZDA,IDEFER,H(LNLOST),
921: N,NDEAD,NREFS,NGROUP,NGP1,NREG,NTIME,IRAND,
922: N,NLBZ,NEST,NSUZON,NSPACE,DEFS,
923: B,NMKREG,H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
924: B,H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LLEVL),H(LLZZ),
925: B,H(LLPOS),H(LLCRS),H(LXIM),H(LKLSF),H(LIBREG),H(LIBSPC),
926: B,H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
927: S,H(LLSFFL),H(LNFFL),H(LIZFFL),H(LLSSRC),H(LNNXT),H(LIZNXT),
928: S,H(LLSDED),H(LLSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LKSREF),
929: S,H(LLSLAT),H(LIZLAT),H(LNXLT),
930: G,A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LKSFBD),
931: G,A(LKCELL),A(LIPCEL),A(LMLBZZ),A(LISUSE),A(LMKREG),
932: T,A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
933: T,H(LNKILD),H(LWKILD),H(LNSURV),
934: T,H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
935: T,A(LJDTRG),A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,
936: T,H(LDTALY),A(LRESP),
937: W,H(P2(1)),H(P2(2)),H(P2(3)),H(P2(4)),H(P2(5)),H(P2(6)),
938: W,H(P2(7)),H(P2(8)),H(P2(9)),H(P2(10)),H(P2(11)),H(P2(12)),
939: %       MZONE, JMEM, NMEMO,H(LKMEMO),H(LMEMC),H(LMEMZ))
940: C/#ENDIF
941:               if(JVMNT.ne.0) call VMNTR2(4)
942: C-----
943: C .... COLLISION .....
944: C-----
945:       else if( MACT.eq.3 ) then
946:               if(JVMNT.ne.0) call VMNTR0(5,IOW)
947: C
948:       if( JPTDT.ne.0 ) then
949:               call NXTEC( IOW, A, H, H(LLSCOL),
950: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
951: B H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),
952: B H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),
953: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
954: B KDBNK(0),KIBNK(0),
955: B H(LLZZI),H(LLPOSI),H(LLCRSI),H(LNCNTR),H(LWCNTR),
956: B H(LOPTI),H(LPATH),H(LKDETP),H(LKLSFI),
957: X A(LANGX),A(LNGSX),A(LNNNX),A(LSTOTX),A(LSGPBX),A(LSGPBX),
958: X A(LSSCBX),A(LSSCBX),A(LSDDBX),A(LSDDBX),A(LFKAI),A(LKKAI),
959: X A(LSPORT),A(LSPORT),A(LS2DPX),A(LS2PPX),A(LVEL),
960: G A(LKZMAT),A(LKZREG),A(LTIMEB),A(LXPDET),A(LIPDET),A(LIPDT2),
961: D A(LJPUSD),
962: S H(LIMSFL),H(LIMNFL),H(LIMZFL),
963: S H(LIMDED),H(LIMSLT),H(LIMZLT),H(LIMNLT),
964: W H(PC(1)),H(PC(2)),H(PC(3)),H(PC(4)),H(PC(5)),
965: W H(PC(6)),H(PC(7)),H(PC(8)),H(PC(9)),H(PC(10)),
966: W H(PC(11)),H(PC(12)),H(PC(13)),H(PC(14)),H(PC(15)),
967: W H(PC(16)),H(PC(17)),H(PC(18)),H(PC(19)),H(PC(20)),
968: W H(PC(21)),H(PC(22)),H(PC(23)),H(PC(24)),H(PC(25)),
969: W H(PC(26)),H(PC(27)) )
970: C
971:       end if
972: C
973: C/#IF ARGSAVE
974: *       call COLISN(
975: *       1 H(LLSFFL),H(LNFFL),H(LIZFFL),

```

```

976: *       2 H(LLSCOL), NCOLS, H(LLSDED), NDEAD,
977: *       3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
978: *       4 H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LXIM),
979: *       B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
980: *       & H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
981: *       4 H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),NFISB,
982: *       5 H(LFLCL),H(LNCLSN),H(LWCLSN),
983: *       6 H(LNABSB),H(LWABSB),H(LNECUT),H(LWECUT),H(LNKILD),H(LWKILD),
984: *       7 H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
985: *       8 H(P3(1)),H(P3(2)),H(P3(3)),H(P3(4)),H(P3(5)),H(P3(6)),
986: *       9 H(P3(7)),H(P3(8)),H(P3(9)),H(P3(10)),H(P3(11)),H(P3(12)),
987: *       9 H(P3(13)),H(P3(14))
988: C/#ELSE
989:       call COLISN( IOW,
990: 2 NGROUP,NGP1,NGP2,NZONE,NMAT,NREG,NTIME,NBANK,NFBANK,NFBNK0,NEST,
991: 3 A(LKZMAT),A(LKZREG),A(LXIMP),A(LWKIL),A(LWSRV),A(LWTIME),WLLIM,
992: 4 MXPGEN,A(LWGTG),A(LWGTFI),A(LFKAI),A(LKKAI),A(LWGTG),A(LWGTGI),
993: 4 NGPX, NGGX, NTGX, NDSPX, NDSGX, NDSFX,
994: 5 NUSX, NSCTX, NCOEFX, NMEDX,
995: 6 A(LANGX),A(LNGSX),A(LNNNX),
996: 7 A(LSTOTX),A(LSSCX),A(LSNAPX),A(LSGPX),A(LSNFPX),
997: 8 A(LSNUSX),A(LSGPBX),A(LSGPBX),A(LSSCBX),A(LSSCBX),
998: 9 A(LSDDBX),A(LSDDBX),A(LS2DPX),A(LS2PPX),
999: A A(LSANGX),A(LSANGX),A(LSGPPX),A(LSSCPX),A(LSPORT),
1000: B IRAND,
1001: C H(LLEVL),H(LLZZ),H(LLPOS),H(LLCRS),H(LKLSF),H(LIBREG),H(LIBSPC),
1002: D H(LLEVLF),H(LLZZF),H(LLPOSF),H(LLCRSF),H(LIBRGF),H(LIBSPF),
1003: T NSTAL,NRESP,A(LRESP),NMKREG,A(LMKREG),A(LJDTRG),
1004: T A(LIDTAL),A(LJTEVE),A(LLTEVE),A(LKTEVE),NTEVE,H(LDTALY),
1005: % H(LLSFFL),H(LNFFL),H(LIZFFL),
1006: 2 H(LLSCOL), NCOLS, H(LLSDED), NDEAD,
1007: 3 H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
1008: 4 H(LWWW),H(LIZZ),H(LIGG),H(LTTT),H(LITT),H(LXIM),
1009: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1010: & H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK,
1011: 4 H(LXXXF),H(LYYYF),H(LZZZF),H(LIZZF),NFISB,
1012: 5 H(LFLCL),H(LNCLSN),H(LWCLSN),
1013: 6 H(LNABSB),H(LWABSB),H(LNECUT),H(LWECUT),H(LNKILD),H(LWKILD),
1014: 7 H(LNSURV),H(LWSURV),H(LNSPLT),H(LWSPLT),H(LNCNTR),H(LWCNTR),
1015: 8 H(P3(1)),H(P3(2)),H(P3(3)),H(P3(4)),H(P3(5)),H(P3(6)),
1016: 9 H(P3(7)),H(P3(8)),H(P3(9)),H(P3(10)),H(P3(11)),H(P3(12)),
1017: 9 H(P3(13)),H(P3(14))
1018: C/#ENDIF
1019:               if(JVMNT.ne.0) call VMNTR2(5)
1020: C-----
1021: C .... REFLECTION .....
1022: C-----
1023:       else if( MACT.eq.5 ) then
1024:               if(JVMNT.ne.0) call VMNTR0(6,IOW)
1025:       call MIRROR( JMNTR, JDEBG, JVMNT, JLATT,
1026: N NBANK, NREFS, NSDA, IRAND, NLBZ, NZONE,
1027: B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),
1028: B H(LWWW),H(LIZZ),H(LIBNK),KIBNK,MIBNK,
1029: S H(LNCNTR),H(LWCNTR),H(LLSFFL),H(LNFFL),H(LIZFFL),
1030: S H(LLSSRC),H(LNNXT),H(LIZNXT),
1031: S H(LLSREF),H(LISREF),H(LIZREF),H(LNBREF),A(LSDA),A(LKKREF),
1032: S H(LLSLAT),H(LIZLAT),H(LNXLT),A(LMLBZZ),
1033: W H(P4(1)),H(P4(2)),H(P4(3)),H(P4(4)),
1034: W H(P4(5)),H(P4(6)),H(P4(7)),H(P4(8)),H(P4(9)),H(P4(10)),
1035: W H(P4(11)),H(P4(12)),H(P4(13)),H(P4(14)),H(P4(15)),H(P4(16)),
1036: W H(P4(17)),H(P4(18)),H(P4(19)),H(P4(20)),H(P4(21)),H(P4(22)),
1037: W H(P4(23)) )
1038:
1039:               if(JVMNT.ne.0) call VMNTR2(6)
1040: C-----
1041: C .... LATTICE SEARCH .....

```

src/gmvp/action.f

```

1041: C-----
1042:     else if( MACT.eq.6 ) then
1043:         if(JVMNT.ne.0) call VMNTR0(7,IOW)
1044:         call LATIC( JDEBG,JVMNT,JTLLT,JHLAT,JFISX,H(LNLOST),
1045: N NBANK,NZONE,NREG,NLATT,NCELL,NLBZ,NEST,NSUZON,NSPACE,NUNV,
1046: N DINF,DEPS,IRAND,NMKREG,
1047: B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
1048: B H(LIZZ ), H(LLEVL ), H(LLZZ ), H(LLPOS ), H(LLSDDED), NDEAD ,
1049: B H(LLCRS ), H(LKLSF), H(LIBREG),H(LIBSPC),
1050: B H(LDBNK),KDBNK,MDBNK,H(LIBNK),KIBNK,MIBNK,
1051: S H(LLSSRC), H(LIZNXT), H(LNNXT ), H(LLSLAT), H(LIZLAT), H(LNXLT ),
1052: S H(LLSFFL), H(LNFFL), H(LIZFFL),
1053: G A(LSHA), A(LKZDA), A(LKZAA),
1054: G A(LKMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
1055: G A(LIDLAT), A(LIPLAT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
1056: G A(LLTYP), A(LICTYP), A(LKDAL), A(LDAL), A(LKZREG),
1057: G A(LISUSP),A(LKSPSU),A(LKTOSP),NKTOSP,A(LKHLAT),A(LMKREG),
1058: G A(LDEPPF), A(LKPPF), NPPPF,
1059: W H(P5( 1)), H(P5( 2)), H(P5( 3)), H(P5( 4)), H(P5( 5)), H(P5( 6)),
1060: W H(P5( 7)), H(P5( 8)), H(P5( 9)), H(P5(10)),
1061: W H(P5(11)), H(P5(12)), H(P5(13)), H(P5(14)),H(P5(15)), H(P5(16))
1062:         if(JVMNT.ne.0) call VMNTR2(7)
1063: C-----
1064: C ... NEXT EVENT ESTIMATOR .....
1065: C-----
1066:     else if( MACT.eq.7 ) then
1067: C***** IF(JVMNT.NE.0) CALL VMNTR0(7,IOW)
1068:         call NXTEE( IMMODE, A, H, NDIPT, NIMPT )
1069:
1070: C-----
1071: C end of a batch (and have remaining batch yet)
1072: C-----
1073:     else if( MACT.eq.88 ) then
1074: C
1075: C ..... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR THE FIRST BATCH
1076: C
1077:         call TALSUM(IOW,A,H,CHA,
1078: & NPKIND, NBATCH,NSKIP,NTMINT,NGROUP,NBPINT,JBPRNT,
1079: & NREG, NRESP,NGENE0,WSUMB, NTREG, NEVENT,
1080: & A(LIPTRG),A(LLPTRG), H(LWSUM),H(LWLEK),
1081: & H(LFLTR ), H(LSFLTR), H(LFLCL ), H(LSFLCL),
1082: & A(LRESP ), H(LSRETR), H(LSRECL), H(LTFLH), H(LTFLHS),
1083: & H(LXSOC ), H(LXKEF ), H(LNCNTR), H(LWCNTR), NFISB ,
1084: T H(LIETAL),NETALY,A(LIDTAL),NDTALY,H(LETALY),NLETAL,
1085: T H(LDTALY),NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
1086: & H(P8(1)), H(P8(2)), H(P8(3)), H(P8(4)), H(P8(5)), H(P8(6)) )
1087:
1088:         if( JPTDT.ne.0 .and. NPDET.gt.0 ) then
1089:             call NEESUM(IOW, NBATCH, NSKIP, NGROUP ,
1090: & NPDET, NRESP, NGENE0, WSUMB, A(LRESP) ,
1091: & H(LFLPD ), H(LSFLPD),
1092: & H(LREPD ), H(LSREPD) )
1093:         end if
1094: C
1095: C ..... OPTIMIZATION OF KMEMO ARRAY .....
1096: C
1097:         if( NBATCH.le.NMEMOP ) then
1098:             call MEMMEM(
1099: & IOW, H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
1100:         end if
1101: C
1102: C ..... Restart file output after specified batch or history .....
1103: C
1104:         if( JRSTF.ne.0.and.NTASK.eq.1.and.NBATCH.gt.0.and.NRSINT.ne.0 )
1105: & then

```

```

1106:         if( NRSINT.gt.0.and. mod(NBATCH, NRSINT).eq.0
1107: & .or. (NRSINT.lt.0.and.mod(NTHIST,abs(NRSINT)).lt.NHIST) )
1108: & then
1109:             write(IOW,7040) NBATCH, NTHIST
1110:             call RESTO0(IOW,H,0,TITLE, NTHIST, NBATCH)
1111:             call RESTOT(IOW,H,0,IDTASK,TITLE)
1112:             call RWIND( IROT )
1113:         end if
1114:         end if
1115: 7040 format(1x,' === RESTART FILE OUTPUT AFTER BATCH ',i6,
1116: & ', HISTORY',I12,' ===')
1117: C
1118: C ..... Check elapsed time and CPU time
1119: C
1120:         call TOKEI(TE1,0)
1121: C/#IF PARA(SX*)
1122: *         call PTCLOCK(DT)
1123: *         T1 = DT
1124: C/#ELSEIF PARA(CRAY*)
1125: *         call TSECND(T1)
1126: C/#ELSE
1127:         call CPUTM(T1)
1128: C/#ENDIF
1129:         NBT1 = NBATCH+1
1130:
1131: C/#IF PARA(SX* CRAY*)
1132: *         call MVPSYNC_LOCK(2)
1133: C/#ENDIF
1134:
1135:         if ( JBPRNT.ne.0 ) then
1136:             write(IOW,7060) IDTASK, NBT1, T1, TE1
1137: 7060 format(/' - TASK ',I5,' BATCH ',I6,
1138: & ' --- CPU ',E11.4,' (SEC) ELAPSED ',E11.4,' (SEC) ---- ')
1139:         end if
1140:
1141: C/#IF PARA(SX* CRAY*)
1142: *         call MVPSYNC_UNLOCK(2)
1143: C/#ENDIF
1144:
1145:         if( TCPU.ne.0.0.and.T1.gt.TCPU*60.0 ) then
1146:             write(IOW,7080) NPART,T1,TCPU
1147:             goto 9999
1148: 7080 format(/2x,8('%%%%%%%%%')/
1149: & 3x,'Run terminated before running ',i10,
1150: & ' histories, because'/
1151: & 2x,'used cpu time (' ,E11.4,' sec) ',
1152: & 'exceeded program limit TCPU (' ,E11.4,' min.).'/
1153: & 2x,8('%%%%%%%%%')/
1154: & )
1155:         end if
1156: C-----
1157: C .... Check status of usr signal-1 and terminate calculation
1158: C if SIGUSR1 was caught .....
1159: C-----
1160: C/#IF .NOT.PARA.AND.UNIX
1161: ISUSR1 = 0
1162: call FGETUSRSIG( 1, ISUSR1 )
1163: if ( ISUSR1 .ne. 0 ) then
1164:     write(IOW,7100) NTHIST
1165:     goto 9999
1166: 7100 format(/2x,8('%%%%%%%%%')/
1167: & 3x,'Run terminated after running ',I10,
1168: & ' histories, because'/
1169: & 3x,'a signal to terminate calculation is caught.'/
1170: & 2x,8('%%%%%%%%%')/
1171: & )

```

src/gmvp/action.f

```

1171:         end if
1172: C/#ENDIF
1173: C
1174: C
1175: C ... update remaining history number and finished history number
1176: C for next batch
1177: C
1178:         NGBAT = MIN( NPART-NTHIST, NHIST )
1179:         NHIST1 = NGBAT
1180:         NGENE0 = 0
1181:         WSUMB = 0.0D0
1182: C
1183: C ... select fission source of next batch for eigenvalue problem ...
1184: C
1185: C if ( JEIGN.ne.0 ) then
1186: C     call FISSET( IOW, H(LFISB), WFFACT,
1187: C &         IRAND, NFISB, NHIST, NHIST1, NHSUB, NBANK,
1188: C &         NZONE, NFBANK, NFBANK0, NBATCH, NREG, NEST,
1189: C &         A(LWGTF), A(LKZREG),
1190: C &         H(LXSOC), NLENG,
1191: C &         H(LXXXF), H(LYYF), H(LZZZF), H(LIZZF),
1192: C &         H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
1193: C &         H(LIBRGF), H(LIBSPF), H(LKLSFF),
1194: C &         H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK,
1195: C &         H(LXXXF2), H(LYYF2), H(LZZZF2), H(LIZZF2),
1196: C &         H(LLEVLF2), H(LLZZF2), H(LLPOSF2), H(LLCRSF2),
1197: C &         H(LIBRGF2), H(LIBSPF2), H(LKLSFF2),
1198: C &         H(LDFBK2), KDFBK2, MDFBK2,
1199: C &         H(LIFBK2), KIFBK2, MIFBK2,
1200: C &         H(P10(1)) )
1201: C     end if
1202: C -----
1203: C All histories are finished
1204: C -----
1205:         else if( MACT.eq.99 ) then
1206: C
1207: C ..... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR INITIAL BATCH
1208: C
1209:         call TALSUM(IOW,A,H,CHA,
1210: C & NPkind,NBATCH,NSKIP,NTMINT,NGROUP,NBPINT,JBRN,
1211: C & NREG, NRESP,NGENE0, WSUMB, NTREG, NEVENT,
1212: C & A(LIPTRG),A(LLPTRG), H(LWSUM),H(LWLEK),
1213: C & H(LFLTR ), H(LSFLTR), H(LFLCL ), H(LSFLCL),
1214: C & A(LRESP ), H(LSRETR), H(LSRECL), H(LTFLH), H(LTFLHS),
1215: C & H(LXSOC ), H(LXKEF ), H(LMCNTR), H(LWCNTR), NFISB ,
1216: C T H(LIETAL),NETALY,A(LIDTAL),NDTALY,H(LETALY),NLETAL,
1217: C T H(LDTALY),NLDTAL, NETRV, METRV, A(LIETRV), H(LETRV),
1218: C & H(P8(1)), H(P8(2)), H(P8(3)), H(P8(4)), H(P8(5)), H(P8(6)) )
1219: C
1220:         if( JPTDT.ne.0 .and. NPDET.gt.0 ) then
1221:             call NEESUM(IOW, NBATCH, NSKIP, NGROUP,
1222: C & NPDET, NRESP, NGENE0, WSUMB, A(LRESP),
1223: C & H(LFLPD ), H(LSFLPD),
1224: C & H(LREPD ), H(LSREPD) )
1225:         end if
1226: C
1227: C ..... OPTIMIZATION OF KMEMO ARRAY .....
1228: C
1229:         if( NBATCH.le.NMEMOP ) then
1230:             call MEMMEM(
1231: C & IOW,H(LKMEMO),H(LMEMC),H(LMEMZ),NZONE,NMEMO,NBATCH)
1232:         end if
1233:         goto 9999
1234:     end if
1235: C

```

```

1236:         goto 2222
1237: C
1238: C
1239: C -----
1240: C ..... END OF RANDOM WALK .....
1241: C -----
1242: C
1243: C
1244: C
1245: 9999 continue
1246:     call TOKEI(TE1,0)
1247: C/#IF PARA( SX* )
1248: *     call PTCLOCK( DT )
1249: *     T1 = DT
1250: C/#ELSEIF PARA( CRAY* )
1251: *     call TSECND ( T1 )
1252: C/#ELSE
1253:     call CPUTM(T1)
1254: C/#ENDIF
1255: C
1256: C
1257: C ... report for each task
1258: C
1259: C/#IF PARA(SX* CRAY*)
1260: *     call MVPSYNC_LOCK(2)
1261: *     call REPORT(IDTASK,NTGEN, T1-T0, TE1-TE0, H(LNLOST) )
1262: *     call MVPSYNC_UNLOCK(2)
1263: C/#ENDIF
1264: C
1265: C/#IF UNIX
1266:     call FLUSHSTD()
1267: C/#ENDIF
1268: C/#IF PARA(PVM MPI)
1269: *     call MVPCOM_BARRIER(INFO)
1270: *     IT0 = 1
1271: *     if( IDTASK.eq.IT0 ) then
1272: *         call REPORT(IDTASK,NTGEN,T1-T0,TE1-TE0,H(LNLOST))
1273: *         TTCPU = T1-T0
1274: *         do 349 I=2,NTASK
1275: *             call MVPCOM_RECV_R4(I,100,TDCPU, 1,INFO)
1276: *             call MVPCOM_RECV_R4(I,101,TDELP, 1,INFO)
1277: *             call MVPCOM_RECV_I4(I,102,NTGENI, 1,INFO)
1278: *             call MVPCOM_RECV_I4(I,103,NLOSTI, 1,INFO)
1279: C
1280: *             call REPORT(I,NTGENI, TDCPU, TDELP, NLOSTI )
1281: *             TTCPU = TTCPU + TDCPU
1282: * 349 continue
1283: *             write(IPR,'(1X,A,E12.5,A)')
1284: *             & ' CPU TIME SUMMED OVER ALL TASKS: ',TTCPU,' SEC'
1285: *         else
1286: *             IT0 = 1
1287: *             call MVPCOM_SEND_R4(IT0,100,T1-T0, 1,INFO)
1288: *             call MVPCOM_SEND_R4(IT0,101,TE1-TE0, 1,INFO)
1289: *             call MVPCOM_SEND_I4(IT0,102,NTGEN, 1,INFO)
1290: *             call MVPCOM_SEND_I4(IT0,103,H(LNLOST), 1,INFO)
1291: *         endif
1292: C/#ENDIF
1293: C
1294:     return
1295: end

```

src/gmvp/actmpp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTMPP( TITLE, A, H, CHA )
3:   C
4:   C/#IF PARA(PVM MPI)
5:   C
6:   C=<GMVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C           ( For message passing multi task mode )
9:   C CALLED IN: CENTER
10:  C CALLS:  ACTION,HEADER,VMNTRF
11:  C=====
12:  C character TITLE(2)*72
13:  CCC   include '../shared/INC/_ARRAY'
14:  C real A(*)
15:  C real H(*)
16:  C character*4 CHA(*)
17:  C
18:  C   include 'INC/_KPIDS'
19:  C   include 'INC/_NGPS'
20:  C
21:  C   include 'INC/_FLAGS'
22:  C   include '../shared/INC/_SIZES'
23:  C   include 'INC/_SIZES2'
24:  C   include 'INC/_XBANK'
25:  C   include 'INC/_STACK'
26:  C   include 'INC/_XWORK'
27:  C   include '../shared/INC/_PGEOM'
28:  C   include 'INC/_PXSEC'
29:  C   include '../shared/INC/_PVRED'
30:  C   include 'INC/_PSOUR'
31:  C   include '../shared/INC/_PTALY0'
32:  C   include 'INC/_PTALY'
33:  C   include '../shared/INC/_PTLSP'
34:  C   include '../shared/INC/_STALY'
35:  C
36:  Ccc   include '../shared/INC/_COUNTS'
37:  C
38:  C/#IF PARA(PVM)
39:  C   include '../shared/INC/_PVMPARA'
40:  C/#ELSEIF PARA(MPI)
41:  C   *   include 'mpif.h'
42:  C/#ENDIF
43:  C   include '../shared/INC/_TASKDT'
44:  C   include '../shared/INC/_IOUNIT'
45:  C   include 'INC/_IOUNIT2'
46:  C
47:  C ... working array pointers PCA(*) for CEL2AB
48:  C   include 'INC/_WKC2A'
49:  C
50:  C ... local variables ...
51:  C
52:  C/#IF INTEGER8
53:  C   *   integer*8 KTHIST, IDUM8
54:  C/#ELSE
55:  C   integer KTHIST
56:  C/#ENDIF
57:  C
58:  C-----
59:  C
60:  C
61:  C-----
62:  C ..... Input Body of restart file
63:  C           (header is already input in INTRO2 routine)
64:  C-----
65:

```

```

66:   if ( JREST.ne.0 ) then
67:   C
68:   C ... task 1 inputs data for himself and pass data to other task.
69:   C
70:   C   if ( IDTASK.eq.1 ) then
71:   C       call RESTIN( IOW, H, IDTASK, TITLE )
72:   C       do 100 ITSK = 2, NTASK
73:   C           write(IOW,*) '... Input restart record for task ', ITSK
74:   C           call RESTIN( IOW, H, ITSK, TITLE )
75:   C       100 continue
76:   C   else
77:   C       call RESTIN( IOW, H, IDTASK, TITLE )
78:   C   end if
79:   C
80:   C   end if
81:   C
82:   C-----
83:   C ..... Monte Carlo run
84:   C-----
85:   C
86:   C   if( JNRUN.eq.0 ) then
87:   C       call ACTION( TITLE, A, H, CHA )
88:   C   end if
89:   C
90:   C-----
91:   C   stop task controller process if necessary
92:   C-----
93:   C
94:   C/#IF PARA(PVM)
95:   C   if ( JREPR.eq.0 ) then
96:   C       if ( JTSKR.ne.0 ) then
97:   C           if ( IDTASK.eq.1.and.NTASK.gt.1 ) then
98:   C               call PVMFKILL( ICTASK, INFO )
99:   C               if ( INFO.ge.0 ) then
100:  C                   write(IPR,7000)
101:  C                   format(' TASK CONTROLLER PROCESS TERMINATED',
102:  C                       & ' SUCCESSFULLY.'/)
103:  C               else
104:  C                   call PVMFPERROR( 'TERMINATION OF TASK CONTROLLER.', INFO
105:  C                       & )
106:  C               end if
107:  C           end if
108:  C       end if
109:  C/#ENDIF
110:  C
111:  C-----
112:  C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
113:  C-----
114:  C
115:  C   if ( JVMNT.ne.0 ) then
116:  C       call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
117:  C       call VMNTRF( IOW )
118:  C   end if
119:  C
120:  C   call MVPCOM_BARRIER( INFO )
121:  C/#IF UNIX
122:  C   call FLUSHSTD()
123:  C/#ENDIF
124:  C
125:  C-----
126:  C ..... Output Body of restart file
127:  C-----
128:  C   if( JRSTF.ne.0 ) then
129:  C
130:  C ... take sum of NTHIST tempoary on KTHIST1

```

src/gmvp/actmpp.f

```

131: C
132:       KTHIST = NTHIST
133: C/#IF INTEGER8
134: *       call TSKSMI( 'NTHIST', IDUM8, KTHIST, 1 )
135: C/#ELSE
136:       call TSKSMI( 'NTHIST', IDUMMY, KTHIST, 1 )
137: C/#ENDIF
138: C
139: C     ... check number of batches
140: C
141:       if ( JEIGN.ne.0 ) then
142:         KBATCH = NBATCH
143:       else
144:         KBATCH = NBATCH
145:         call TSKSMI( 'NBATCH', IDUMMY, KBATCH, 1 )
146:       end if
147: C
148: C     ... output header records ...
149: C
150:       if ( IDTASK.eq.1 ) then
151:         call RWIND( IROT )
152:         call RESTO0( IOW, H,1, TITLE, KTHIST, KBATCH )
153:       end if
154: C
155: C     ... output data body ...
156: C
157: C     ... task 1 outputs data for himself and get data from other task.
158: C
159:       if ( IDTASK.eq.1 ) then
160:         call RESTOT( IOW, H, 1, IDTASK, TITLE )
161:         do 110 ITSK = 2, NTASK
162:           call RESTOT( IOW, H, 1,ITSK, TITLE )
163:         110 continue
164:         call RWIND( IROT )
165:       else
166:         call RESTOT( IOW, H, 1,IDTASK, TITLE )
167:       end if
168:     end if
169: C
170: C-----
171: C     .... Take summation of data on each task
172: C-----
173: C
174:       call MTSUMS( TITLE, H )
175: C
176:       call MVPCOM_BARRIER( INFO )
177: C/#IF UNIX
178:       call FLUSHSTD()
179: C/#ENDIF
180: C
181: C-----
182: C     .... Output fission source file
183: C-----
184: C
185:       if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
186: C
187: C     ... convert in-cell coordinates into absolute coordinates ...
188: C     (XXXF,YYYF,ZZZF)
189: C
190:       if ( JLATT.ne.0 ) then
191:         call CEL2AB( IOW, NFISB, NFBNK0,
192:           &         A, JDEBG, H(LXXXF), H(LYYYF),
193:           &         H(LZZZF), H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
194:           &         H(LDFBK),KDFBK,MDFBK,
195:           &         H(PCA(1)), H(PCA(2)), H(PCA(3)) )

```

```

196:       end if
197: C
198: C     ... gather fission neutron data on task #1
199: C     value of NFISB is changed to the sum of NFISB over tasks.
200: C
201:       if ( NTASK.gt.1 ) then
202:         call MPPFBK( NTASK, IDTASK, NFISB, NFBNK0, NEST, JTLLT,
203:           &         H(LXXXF), H(LYYYF), H(LZZZF), H(LIZZF), H(LIBRGF),
204:           &         H(LIBSPF),
205:           &         H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK,MIFBK )
206:       end if
207: C
208:       NUSE = NFISB
209: C
210:       if ( IDTASK.eq.1 ) then
211:         call SCFOUT( 'GMVP', TITLE, NUSE, NFBNK0, NEST, NREG,
212:           &         A(LWGTF), NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF),
213:           &         H(LYYYF), H(LZZZF), DUMMY, H(LIZZF), DUMMY,
214:           &         H(LIBRGF), H(LIBSPF), ' ', 1, IERR )
215:       end if
216: C
217:       end if
218: C
219: C-----
220: C     ... tasks other than main task exits here ...
221: C-----
222: C
223:       call MVPCOM_BARRIER( INFO )
224: C/#IF UNIX
225:       call FLUSHSTD()
226: C/#ENDIF
227: C
228:       IT0 = 1
229:       if ( IDTASK.ne.IT0 ) then
230:         write(IMG,*) '=== Task ', IDTASK, ' Stopped.'
231:       end if
232: C
233:       call MVPCOM_EXIT( INFO )
234: C
235:       if ( IDTASK.ne.IT0 ) then
236:         stop
237:       end if
238: C
239: C/#ENDIF
240: C
241:       return
242:     end
243: C
244: C-----
245: C
246: C=====
247: C     Routines to gather array data on the first task
248: C     history : programmed by M.Sasaki (Jul 1997)
249: C=====
250: C
251: C/#IF PARA(PVM MPI)
252:       subroutine MPPFBK( NTASK, IDTASK,NFISB, NFBNK0,NEST, JTLLT, XXXF,
253:         &         YYYF, ZZZF, IZZF, IBRGF, IBSPF,
254:         &         DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
255: C=====
256: C     purpose: gather neutron bank data on task ITSK in
257: C     parallel mode on PVM, MPI
258: C     history: programmed by M.Sasaki ( 18 Jul 1997 )
259: C     10 Feb 1999: gather optional fission bank data
260: C=====

```

src/gmvp/actmpp.f

```

261:
262:     real*8 XXXF(NFBNK0)
263:     real*8 YYYY(NFBNK0)
264:     real*8 ZZZF(NFBNK0)
265:     integer IZZF(NFBNK0)
266:     integer IBRGF(NFBNK0)
267:     integer IBSPF(NFBNK0,NEST)
268: C
269:     real*8 DFBK(NFBNK0,*)
270:     integer KDFBK(0:MDFBK)
271:     integer IFBK(NFBNK0,*)
272:     integer KIFBK(0:MIFBK)
273: C -----
274:     call MPPGR8( XXXF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
275:     call MPPGR8( YYYY, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
276:     call MPPGR8( ZZZF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
277:     call MPPGI4( IZZF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
278:     if ( JTLT.ne.0 ) then
279:         call MPPGI4( IBRGF, NFBNK0, NFISB, NF, NTASK, IDTASK, IERR )
280:         do 100 J = 1, NEST
281:             call MPPGI4( IBSPF(1,J), NFBNK0, NFISB, NF, NTASK, IDTASK,
282: & IERR )
283: 100     continue
284:     end if
285:
286:     if ( KDFBK(0).gt.0 ) then
287:         do 130 K = 1, KDFBK(0)
288:             call MPPGR8( DFBK(1,K), NFBNK0, NFISB,
289: & NF, NTASK, IDTASK, IERR )
290: 130     continue
291:     end if
292:
293:     if ( KIFBK(0).gt.0 ) then
294:         do 150 K = 1, KIFBK(0)
295:             call MPPGI4( IFBK(1,K), NFBNK0, NFISB,
296: & NF, NTASK, IDTASK, IERR )
297: 150     continue
298:     end if
299: C
300:     NFISB = NF
301: C
302:     return
303: end
304: C/#ENDIF
305:
306: C=====
307: C  gather data on an array  on task #1.
308: C  Data size may differ in each task.
309: C-----
310: C arguments ( i=input, o=output, w=work )
311: C i o IA(na),RA(na),Da(na) : integer,real*4,real*8 arrays
312: C i na : array dimension size
313: C i nd : data size on calling task.
314: C o ndd : returned as sum of data size on all tasks.
315: C i ntask : number of tasks
316: C i itsk : calling task #
317: C=====
318:     subroutine MPPGI4( IA, NA, ND, NDD, NTASK, ITSK, IERR
319: & )
320: C
321: C/#IF PARA( PVM MPI)
322:     integer IA(NA)
323:     parameter( MSGTG1 = 987 )
324:     parameter( MSGTG2 = 988 )
325:     parameter( MSGTG3 = 989 )

```

```

326: C
327:     include '../shared/INC/_IOUNIT'
328: C
329:     IERR = 0
330: C
331:     if ( ITSK.eq.1 ) then
332:         ND0 = ND
333:         do 100 IT = 2, NTASK
334:             call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
335:
336:             if ( ND0+NN.gt.NA ) then
337:                 write(IPR,*)
338: & 'XXX(MPPGI4) Too many data are being gathered.',
339: & ' from task ', IT, ' size ', ND,
340: & ' gathered data so far ', ND0, ' limit : ', NA
341:                 IERR = 1
342:             end if
343:
344:             call MVPCOM_RECV_I4( IT, MSGTG2, IA(ND0+1), NN, INFO )
345:             ND0 = ND0 + NN
346: 100     continue
347:         NDD = ND0
348:         call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
349:     else
350:         call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
351:         call MVPCOM_SEND_I4( 1, MSGTG2, IA, ND, INFO )
352:         call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
353:         if ( IERR.ne.0 ) then
354:             write(IPR,*) 'XXX(MPPGI4) Error on Task #1.'
355:         end if
356:     end if
357: C/#ENDIF
358:     return
359: end
360: C
361: C-----
362: C
363:     subroutine MPPGR4( RA, NA, ND, NDD, NTASK, ITSK, IERR
364: & )
365: C
366: C/#IF PARA( PVM MPI)
367:     real RA(NA)
368:     parameter( MSGTG1 = 987 )
369:     parameter( MSGTG2 = 988 )
370:     parameter( MSGTG3 = 989 )
371: C
372:     include '../shared/INC/_IOUNIT'
373: C
374:     IERR = 0
375: C
376:     if ( ITSK.eq.1 ) then
377:         ND0 = ND
378:         do 100 IT = 2, NTASK
379:             call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
380:
381:             if ( ND0+NN.gt.NA ) then
382:                 write(IPR,*)
383: & 'XXX(MPPGR4) Too many data are being gathered.',
384: & ' from task ', IT, ' size ', ND,
385: & ' gathered data so far ', ND0, ' limit : ', NA
386:                 IERR = 1
387:             end if
388:
389:             call MVPCOM_RECV_I4( IT, MSGTG2, RA(ND0+1), NN, INFO )
390:             ND0 = ND0 + NN

```


src/gmvp/actmpp.f

```
391: 100 continue
392: NDD = ND0
393: call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
394: else
395: call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
396: call MVPCOM_SEND_R4( 1, MSGTG2, RA, ND, INFO )
397: call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
398: if ( IERR.ne.0 ) then
399: write(IPR,*) 'XXX(MPPGR4) Error on Task #1.'
400: end if
401: end if
402: C/#ENDIF
403: return
404: end
405: C
406: C-----
407: C
408: subroutine MPPGR8( DA, NA, ND, NDD, NTASK, ITSK, IERR
409: &
410: )
411: C/#IF PARA( PVM MPI)
412: real*8 DA(NA)
413: parameter( MSGTG1 = 987 )
414: parameter( MSGTG2 = 988 )
415: parameter( MSGTG3 = 989 )
416: C
417: include '../shared/INC/_IOUNIT'
418: C
419: IERR = 0
420: C
421: if ( ITSK.eq.1 ) then
422: ND0 = ND
423: do 100 IT = 2, NTASK
424: call MVPCOM_RECV_I4( IT, MSGTG1, NN, 1, INFO )
425:
426: if ( ND0+NN.gt.NA ) then
427: write(IPR,*)
428: & 'XXX(MPPGR8) Too many data are being gathered.',
429: & ' from task ', IT, ' size ', NN,
430: & ' gathered data so far ', ND0, ' limit : ', NA
431: IERR = 1
432: end if
433:
434: call MVPCOM_RECV_R8( IT, MSGTG2, DA(ND0+1), NN, INFO )
435: ND0 = ND0 + NN
436: 100 continue
437: NDD = ND0
438: call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
439: else
440: call MVPCOM_SEND_I4( 1, MSGTG1, ND, 1, INFO )
441: call MVPCOM_SEND_R8( 1, MSGTG2, DA, ND, INFO )
442: call MVPCOM_BCAST_I4( 1, MSGTG3, IERR, 1, INFO )
443: if ( IERR.ne.0 ) then
444: write(IPR,*) 'XXX(MPPGR8) Error on Task #1.'
445: end if
446: end if
447: C/#ENDIF
448: return
449: end
```

src/gmvp/actsgl.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTSGL( TITLE, A, H, CHA )
3:   C
4:   C/#IF .NOT.PARA
5:   C
6:   C=<GMVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C CALLED IN: CENTER
9:   C CALLS:   ACTION,HEADER,VMNTRF
10:  C=====
11:  character TITLE(2)*72
12:  CCC include '../shared/INC/_ARRAY'
13:  real I(*)
14:  real H(*)
15:  character*4 CHA(*)
16:  C
17:  include 'INC/_KPID$'
18:  include 'INC/_NGPS'
19:  C
20:  include 'INC/_FLAGS'
21:  include '../shared/INC/_SIZES'
22:  include 'INC/_SIZES2'
23:  include 'INC/_XBANK'
24:  include 'INC/_STACK'
25:  include 'INC/_XWORK'
26:  include '../shared/INC/_PGEOM'
27:  include 'INC/_PXSEC'
28:  include '../shared/INC/_PURED'
29:  include 'INC/_PSOUR'
30:  include '../shared/INC/_PTALY0'
31:  include 'INC/_PTALY'
32:  include '../shared/INC/_PTLSP'
33:  include '../shared/INC/_STALY'
34:  C
35:  CCC include '../shared/INC/_COUNTS'
36:  C
37:  include '../shared/INC/_TASKDT'
38:  include '../shared/INC/_IOUNIT'
39:  include 'INC/_IOUNIT2'
40:  C
41:  C ... working array pointers PCA(*) for CEL2AB
42:  include 'INC/_WKC2A'
43:  C
44:  C ... local variables ...
45:  C
46:  C/#IF INTEGER8
47:  *   integer*8 KTHIST
48:  C/#ELSE
49:  integer KTHIST
50:  C/#ENDIF
51:  C
52:  C-----
53:  C
54:  IDTASK = 1
55:  ITASK0 = 1
56:  C
57:  C-----
58:  C ..... Input Body of restart file
59:  C (header is already input in INTRO2 routine)
60:  C-----
61:  C
62:  if( JREST.ne.0 ) then
63:    call RESTIN(IOW, H, IDTASK,TITLE)
64:  end if
65:

```

```

66:  C-----
67:  C ..... Monte Carlo run
68:  C-----
69:  C
70:  if( JNRUN.eq.0 ) then
71:    call ACTION( TITLE, A, H, CHA )
72:  end if
73:  C
74:  C-----
75:  C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
76:  C-----
77:  C
78:  if( JVMNT .ne. 0 ) then
79:    call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
80:    call VMNTRF( IOW )
81:  end if
82:  C
83:  C-----
84:  C ..... Output Body of restart file
85:  C-----
86:  C
87:  if( JRSTF .ne. 0 ) then
88:  C ... output header records ...
89:  C
90:    KTHIST = NTHIST
91:    KBATCH = NBATCH
92:  C
93:    call RWIND( IROT )
94:    call REST00(IOW, H, 1, TITLE, KTHIST, KBATCH)
95:  C
96:  C ... output data body ...
97:  C
98:    call RESTOT(IOW, H, 1, IDTASK,TITLE)
99:    call RWIND( IROT )
100:  end if
101:  C
102:  C-----
103:  C ..... Output fission source file
104:  C-----
105:  C
106:  if( JEIGN.ne.0 .and. JOSRC.ne.0 ) then
107:  C
108:  C ... convert in-cell coordinates into absolute coordinates ...
109:  C (XXXF,YYYF,ZZZF)
110:  C
111:  if( JLATT.ne.0 ) then
112:    call CEL2AB( IOW, NFISB, NFBK0,
113:  &      A, JDEBG,
114:  &      H(LXXXF), H(LYYYF), H(LZZZF),
115:  &      H(LLEVL), H(LLZZF), H(LLPOSF),H(LLCRSF),
116:  &      H(LDFBK),KDFBK,MDFBK,
117:  &      H(PCA(1)),H(PCA(2)),H(PCA(3)))
118:  end if
119:  C
120:  NUSE = NFISB
121:  C
122:  C ... EEEF, INUF and NUCID are dummy for GMVP
123:  C
124:  call SCFOUT( 'GMVP', TITLE, NUSE, NFBK0, NEST, NREG,
125:  &      A(LWGT), NZONE, A(LKZREG), JDEBG, JTLLT,
126:  &      H(LXXXF), H(LYYYF), H(LZZZF),
127:  &      dummy, H(LIZZF), dummy,
128:  &      H(LIBRGF),H(LIBSPF),
129:  &      ' ',1, IERR )
130:  C

```

src/gmvp/actsgl.f

```
131:      end if
132: C
133: C/#ENDIF
134: C
135:      return
136:      end
```

SAFE

src/gmvp/actsmp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTSMP( TITLE,
3:     &               IDTASK1,      IAINFL1,      LIMITL1,
4:     &               IRANT, NRSTEP,NBATCH1,      NPART1,
5:     &               NHIST1, NHSUB1, NBANK1, IMPMAX1, NTHIST1,
6:     &               NFBANK1,      NFBNK01,      SRCSP, WSUM, WLEK,
7:     &               XAVT,  AAVT,  EAVT,  WCNTR, NCNTR, NLOST,
8:     &               SFLTR, SFLCL, SRETR, SRECL, SFLPD, SREPD, XSOC,
9:     &               XSXV,  XKEF,  IETAL, ETALY, TFLHS, ETRV,
10:    &               NCLSN, WCLSN, NLEAK, WLEAK, NABSB, WABSB,
11:    &               NECUT, WECUT, NTCUT, WTCUT, NKILD, WKILD,
12:    &               NSURV, WSURV, NSPLT, WSPLT, IRSUT, IRSMT, IRSLT
13:    &               )
14: C
15: C/#IF PARA(SX* CRAY*)
16: C
17: C=<GMVP>=====
18: C PURPOSE:  Control of restart I/O & monte carlo run in MVP code
19: C           ( For shared memory multi task mode )
20: C CALLED IN:  CMITER
21: C CALLS:  ACTION,HEADER,VMNTRF
22: C=====
23:   character TITLE(2)*72
24:   include '../shared/INC/_ARRAY'
25:   integer IAINFL(3), LIMITL
26:   real A(*)
27:   real H(*)
28:   character*4 CHA(*)
29: C
30:   include 'INC/_KPIDS'
31:   include 'INC/_NGPS'
32: C
33:   include '../shared/INC/_TASKDT'
34:   include 'INC/_FLAGS'
35:   include '../shared/INC/_SIZES'
36:   include 'INC/_SIZES2'
37:   include 'INC/_XBANK'
38:   include 'INC/_STACK'
39:   include 'INC/_XWORK'
40:   include '../shared/INC/_PGEOM'
41:   include 'INC/_PXSEC'
42:   include '../shared/INC/_PVRED'
43:   include 'INC/_PSOUR'
44:   include '../shared/INC/_PTALY0'
45:   include 'INC/_PTALY'
46:   include '../shared/INC/_PTLSP'
47:   include '../shared/INC/_STALY'
48: C
49: Ccc include '../shared/INC/_COUNTS'
50:
51:   include '../shared/INC/_IOUNIT'
52:   include 'INC/_IOUNIT2'
53: C
54: C ... working array pointers PCA(*) for CEL2AB
55:   include 'INC/_WKC2A'
56:
57:   include '../shared/INC/_RAND'
58: C
59:   integer SRCSP(NSRCSP)
60:   real*8 WSUM, WLEK
61:   real XAVT(3), AAVT(3), EAVT
62:   integer IAINFL1(3)
63: C/#IF INTEGER8
64: *   integer*8 NPART1, NTHIST1
65: C/#ELSE

```

```

66:   integer  NPART1, NTHIST1
67: C/#ENDIF
68: C
69: C ... local variables ...
70: C
71:   real*8 DT
72: C/#IF INTEGER8
73: *   integer*8 KKHIST, KTHIST
74: C/#ELSE
75:   integer  KKHIST, KTHIST
76: C/#ENDIF
77: C-----
78: C
79: C ... idtask : task # passed as user defined task value
80: C ... tcp0 : used task CPU time at startup
81: C
82:   IDTASK = IDTASK1
83: C
84:   call RNINIT( NRSTEP )
85: C
86:   if ( IDTASK.ne.ITASK0 ) then
87:     NPART = NPART1
88:     NHIST = NHIST1
89:     NHSUB = NHSUB1
90:     NBANK = NBANK1
91:     IMPMAX = IMPMAX1
92:     NFBANK = NFBANK1
93:     NFBNK0 = NFBNK01
94:     do 100 I = 1, 3
95:       IAINFL(I) = IAINFL1(I)
96: 100   continue
97:     LIMITL = LIMITL1
98: C
99: C ... clear task local data & set random number seed ...
100: C
101:   call PUTVI( H(IAINFL(1)), LIMITL-IAINFL(1)+1, 0 )
102:   call ATRANS( SRCSP, H(LSRCSP), NSRCSP )
103: C
104: C ... copy source specification data ...
105: C
106:   call ATRANS( SRCSP, H(LSRCSP), NSRCSP )
107: C
108: C ... copy special tally information data specification data ...
109: C
110:   call ATRANS( IETAL, H(LIETAL), NETALY )
111: end if
112: C
113:   call PUTVD( H(LWSUM), 1, 0.0D0 )
114:   call PUTVD( H(LWLEK), 1, 0.0D0 )
115:   call PUTV( H(LXAVT), 3, 0.0E0 )
116:   call PUTV( H(LAAVT), 3, 0.0E0 )
117:   call PUTV( H(LEAVT), 1, 0.0E0 )
118: C
119:   if ( JREST.eq.0 ) then
120:     NBATCH = 0
121:     NTHIST = 0
122:     IRAND = IRANT
123:     IRNSU = IRSUT
124:     IRNSM = IRSMT
125:     IRNSL = IRSLT
126:   end if
127: C
128:   NCOLS = 0
129:   NDEAD = 0
130:   IDEFER = 0

```

src/gmvp/actsmp.f

```

131:      NFISB   = 0
132:      NIMPT   = 0
133:      IMDEFR  = 0
134: C
135: C-----
136: C
137: C
138: C-----
139: C ..... Input Body of restart file
140: C      (header is already input in INTRO2 routine)
141: C-----
142: C
143: C      if ( JREST.ne.0 ) then
144: C
145: C          if ( IDTASK.gt.1 ) then
146: C              call MVPSYNC_WAIT( IDTASK-1 )
147: C          end if
148: C
149: C          call MVPSYNC_LOCK( 1 )
150: C
151: C          call RESTIN( IOW, H, IDTASK, TITLE )
152: C
153: C          call MVPSYNC_UNLOCK( 1 )
154: C
155: C          if ( NTASK.gt.1.and.IDTASK.lt.NTASK ) then
156: C              call MVPSYNC_POST( IDTASK )
157: C          end if
158: C
159: C          call MVPSYNC_BARRIER( 1 )
160: C
161: C      end if
162: C
163: C-----
164: C ..... Monte Carlo run
165: C-----
166: C
167: C      if( JNRUN.eq.0 ) then
168: C          call ACTION( TITLE, A, H, CHA )
169: C      end if
170: C
171: C-----
172: C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED ....
173: C-----
174: C
175: C      if ( JVMNT.ne.0 ) then
176: C          call MVPSYNC_LOCK( 1 )
177: C
178: C          call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
179: C          call VMNTRF( IOW )
180: C
181: C          call MVPSYNC_UNLOCK( 1 )
182: C      end if
183: C
184: C-----
185: C ..... Output Body of restart file
186: C-----
187: C      if( JRSTF .ne. 0 ) then
188: C
189: C          ... output header records ...
190: C
191: C
192: C          ... take sum of NTHIST tempoary on NTHIST1 and restore value
193: C          of NTHIST1.
194: C
195: C          KKHIST = NTHIST1

```

```

196:
197:      call MVPSYNC_BARRIER( 1 )
198:      call MVPSYNC_LOCK( 1 )
199:
200:      if ( IDTASK.ne.IDTASK0 ) then
201:          call TSKSMI( 'NTHIST', NTHIST1, NTHIST, 1 )
202:      end if
203:
204:      call MVPSYNC_UNLOCK( 1 )
205:      call MVPSYNC_BARRIER( 1 )
206: C
207:      KTHIST = NTHIST1
208:      NTHIST1 = KKHIST
209: C
210: C ... check number of batches
211: C
212: C      if ( JEIGN.ne.0 ) then
213: C          KBATCH = NBATCH
214: C      else
215: C          KK      = NBATCH1
216: C
217: C          call MVPSYNC_BARRIER( 1 )
218: C          call MVPSYNC_LOCK( 1 )
219: C
220: C          if ( IDTASK.ne.IDTASK0 ) then
221: C              call TSKSMI( 'NBATCH', NBATCH1, NBATCH, 1 )
222: C          end if
223: C
224: C          call MVPSYNC_UNLOCK( 1 )
225: C          call MVPSYNC_BARRIER( 1 )
226: C
227: C          KBATCH = NBATCH1
228: C          NBATCH1 = KK
229: C      end if
230: C
231: C      if ( IDTASK.eq.1 ) then
232: C          call RWIND( IROT )
233: C          call RESTO0( IOW, H, 1, TITLE, KTHIST, KBATCH )
234: C      end if
235: C
236: C ... output data body ...
237: C
238: C      if ( IDTASK.gt.1 ) then
239: C          call MVPSYNC_WAIT( IDTASK-1 )
240: C      end if
241: C
242: C      call MVPSYNC_LOCK( 1 )
243: C
244: C      call RESTOT( IOW, H, 1, IDTASK, TITLE )
245: C
246: C      call MVPSYNC_UNLOCK( 1 )
247: C
248: C      if ( NTASK.gt.1.and.IDTASK.lt.NTASK ) then
249: C          call MVPSYNC_POST( IDTASK )
250: C      end if
251: C
252: C      call MVPSYNC_BARRIER( 1 )
253: C
254: C      if ( IDTASK.eq.1 ) then
255: C          call RWIND( IROT )
256: C      end if
257: C      end if
258: C
259: C-----
260: C ..... Take summation of data on each task

```

src/gmvp/actsmp.f

```

261: C-----
262:
263:       call MTSUMS( TITLE, H, IDTASK1, IAINFL1, LIMITL1, IRANT, NRSTEP,
264: &               NBATCH1, NPART1, NHIST1, NBANK1, IMPMAX1, NTHIST1,
265: &               NFBANK1, NFBNK01, SRCSP, WSUM, WLEK, XAVT, AAVT, EAVT,
266: &               WCNTR, NCNTR, NLOST, SFLTR, SFLCL, SRETR, SRECL, SFLPD,
267: &               SREPD, XSOC, XSXV, XKEF, IETAL, ETALY, TFLHS,
268: &               NETRV, NSKIP, METRV, ETRV,
269: &               NCLSN, WCLSN, NLEAK, WLEAK, NABSB, WABSB, NECUT, WECUT,
270: &               NTCUT, WTCUT, NKILD, WKILD, NSURV, WSURV, NSPLT, WSPLT )
271: C
272: C-----
273: C ... Output fission source file
274: C-----
275: C
276:       if ( JELSN.ne.0.and.JOSRC.ne.0 ) then
277: C
278: C ... convert in-cell coordinates into absolute coordinates ...
279: C       (XXXF,YYYF,ZZZF)
280: C
281:       if ( JLATT.ne.0 ) then
282:         call CEL2AB( IOW, NFISB, NFBNK0,
283: &               A, JDEBG, H(LXXXF), H(LYYYF),
284: &               H(LZZZF), H(LLEWLF), H(LLZZF), H(LLPOSF), H(LLCRBF),
285: &               H(LDFBK), KDFBK, MDFBK,
286: &               H(PCA(1)), H(PCA(2)), H(PCA(3)) )
287:       end if
288: C
289: C ... use I/O unit IUB to collect fission neutron data
290: C
291:       if ( IDTASK.eq.1 ) call RWIND( IUB )
292: C
293:       call MVPSYNC_BARRIER( 1 )
294: C
295:       if ( IDTASK.gt.1 ) then
296: C
297:         call MVPSYNC_LOCK( 1 )
298: C
299:         call SMPFOT( IUB, IDTASK, NFISB, NFBNK0, NEST, JTLLT,
300: &               H(LXXXF), H(LYYYF), H(LZZZF), H(LLZZF), H(LIBRGF),
301: &               H(LIBSPF),
302: &               H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
303: C
304:         call MVPSYNC_UNLOCK( 1 )
305: C
306:       end if
307: C
308:       call MVPSYNC_BARRIER( 1 )
309: C
310:       if ( IDTASK.eq.1 ) then
311:         call RWIND( IUB )
312:         call SMPFIN( IUB, NTASK, NFISB, NFBNK0, NEST, JTLLT,
313: &               H(LXXXF), H(LYYYF), H(LZZZF), H(LLZZF), H(LIBRGF),
314: &               H(LIBSPF),
315: &               H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
316: C
317:         NUSE = NFISB
318:         call SCFOUT( 'GMVP', TITLE, NUSE, NFBNK0, NEST, NREG,
319: &               A(LWGTF), NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF),
320: &               H(LYYYF), H(LZZZF), DUMMY, H(LLZZF), DUMMY,
321: &               H(LIBRGF), H(LIBSPF), ' ', 1, IERR )
322:       end if
323:     end if
324: C
325: C/#ENDIF

```

```

326: C
327:       return
328:     end
329: C
330: C
331:       subroutine SMPFOT( IUB, ITSK, NFISB, NFBNK0,NEST, JTLLT, XXXF,
332: &               YYYF, ZZZF, IZZF, IBRGF, IBSPF,
333: &               DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
334: C=====
335: C purpose: output fission neutron bank data on task ITSK in
336: C SMP Parallel mode (SX, CRAY ...)
337: C
338: C Task #1 does not call this routine.
339: C
340: C history: programmed by M.Sasaki ( 19 Jul 1997 )
341: C 10 Feb 1999: output optional fission bank data
342: C=====
343: C/#IF PARA(SX* CRAY*)
344: C
345:       real*8 XXXF(NFBNK0)
346:       real*8 YYYF(NFBNK0)
347:       real*8 ZZZF(NFBNK0)
348:       integer IZZF(NFBNK0)
349:       integer IBRGF(NFBNK0)
350:       integer IBSPF(NFBNK0,NEST)
351: C
352:       real*8 DFBK(NFBNK0,*)
353:       integer KDFBK(0:MDFBK)
354:       integer IFBK(NFBNK0,*)
355:       integer KIFBK(0:MIFBK)
356: C-----
357:       write(IUB) ITSK, NFISB
358:       write(IUB) (XXXF(I),I=1,NFISB)
359:       write(IUB) (YYYF(I),I=1,NFISB)
360:       write(IUB) (ZZZF(I),I=1,NFISB)
361:       write(IUB) (IZZF(I),I=1,NFISB)
362: C
363:       if ( JTLLT.ne.0 ) then
364:         write(IUB) (IBRGF(I),I=1,NFISB)
365:         write(IUB) ((IBSPF(I,J),I=1,NFISB),J=1,NEST)
366:       end if
367: C
368:       if ( KDFBK(0).gt.0 ) then
369:         do 100 K=1,KDFBK(0)
370:           write(IUB) (DFBK(I,K),I=1,NFISB)
371: 100       continue
372:         end if
373: C
374:       if ( KIFBK(0).gt.0 ) then
375:         do 110 K=1,KIFBK(0)
376:           write(IUB) (IFBK(I,K),I=1,NFISB)
377: 110       continue
378:         end if
379: C
380: C/#ENDIF
381:       return
382:     end
383: C
384: C
385:       subroutine SMPFIN( IUB, NTASK, NFISB, NFBNK0,NEST, JTLLT, XXXF,
386: &               YYYF, ZZZF, IZZF, IBRGF, IBSPF,
387: &               DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
388: C=====
389: C purpose: input fission neutron bank data on scratch file in
390: C SMP parallel mode (SX, CRAY ...)

```

src/gmvp/actsmp.f

```
391: C
392: C   Only task 1 calls this routine.
393: C
394: C history: programmed by M.Sasaki (Jul 1997)
395: C 10 Feb 1999: input optional fission bank data
396: C=====
397: C/#IF PARA(SX* CRAY*)
398:
399:     real*8 XXXF(NFBNK0)
400:     real*8 YYF(NFBNK0)
401:     real*8 ZZZF(NFBNK0)
402:     integer IZZF(NFBNK0)
403:     integer IBRGF(NFBNK0)
404:     integer IBSPF(NFBNK0,NEST)
405: C
406:     real*8 DFBK(NFBNK0,*)
407:     integer KDFBK(0:MDFBK)
408:     integer IFBK(NFBNK0,*)
409:     integer KIFBK(0:MIFBK)
410: C-----
411:     do 120 N = 2, NTASK
412:         read(IUB) ITSK, NFISB0
413:         K1      = NFISB + 1
414:         NFISB    = NFISB + NFISB0
415:
416:         read(IUB) (XXXF(I),I=K1,NFISB)
417:         read(IUB) (YYF(I),I=K1,NFISB)
418:         read(IUB) (ZZZF(I),I=K1,NFISB)
419:         read(IUB) (IZZF(I),I=K1,NFISB)
420:
421:         if ( JTLLT.ne.0 ) then
422:             read(IUB) (IBRGF(I),I=K1,NFISB)
423:             read(IUB) ((IBSPF(I,J),I=K1,NFISB),J=1,NEST)
424:         end if
425:
426:         if ( KDFBK(0).gt.0 ) then
427:             do 100 K=1,KDFBK(0)
428:                 read(IUB) (DFBK(I,K),I=K1,NFISB)
429:             continue
430:         end if
431:
432:         if ( KIFBK(0).gt.0 ) then
433:             do 110 K=1,KIFBK(0)
434:                 read(IUB) (IFBK(I,K),I=K1,NFISB)
435:             continue
436:         end if
437:     120 continue
438: C
439: C/#ENDIF
440: C
441:     return
442: end
```

src/gmvp/actvpp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTVPP( TITLE, A, H, CHA )
3:   C
4:   C/#IF PARA(VPP)
5:   C
6:   C=<GMVP>=====
7:   C PURPOSE:  Control of restart I/O & monte carlo run in MVP-code
8:   C           ( For VPP Fortran multi task mode )
9:   C CALLED IN: CENTER
10:  C CALLS:  ACTION,HEADER,VMNTRF
11:  C=====
12:  C character TITLE(2)*72
13:  CCCC include '../shared/INC/_ARRAY'
14:  C real A(*)
15:  C real H(*)
16:  C character*4 CHA(*)
17:  C
18:  C include 'INC/_KPIDS'
19:  C include 'INC/_NGPS'
20:  C
21:  C include '../shared/INC/_TASKDT'
22:  C include 'INC/_FLAGS'
23:  C include '../shared/INC/_SIZES'
24:  C include 'INC/_SIZES2'
25:  C include 'INC/_XBANK'
26:  C include 'INC/_STACK'
27:  C include 'INC/_XWORK'
28:  C include '../shared/INC/_PCBOM'
29:  C include 'INC/_PXSEC'
30:  C include '../shared/INC/_PVRED'
31:  C include 'INC/_PSOUR'
32:  C include '../shared/INC/_PTALY0'
33:  C include 'INC/_PTALY'
34:  C include '../shared/INC/_PTLSP'
35:  C include '../shared/INC/_STALY'
36:  C
37:  C include '../shared/INC/_COUNTS'
38:  C
39:  C include '../shared/INC/_VPPPARA'
40:  C
41:  C include '../shared/INC/_IOUNIT'
42:  C include 'INC/_IOUNIT2'
43:  C
44:  C ... working array pointers PCA(*) for CEL2AB
45:  C include 'INC/_WKC2A'
46:  C
47:  C ... local variables ...
48:  C
49:  C/#IF INTEGER8
50:  C * integer*8 KTHIST
51:  C/#ELSE
52:  C integer KTHIST
53:  C/#ENDIF
54:  C real*8 TT0, TT1, TT2, TT3
55:  C character*137 BUFF
56:  C
57:  C!xocl processor p( mpe )
58:  C
59:  C
60:  C-----
61:  C
62:  C=====
63:  C In current implimentation, each process uses its own message
64:  C output I/O unit (IOW).
65:  C=====

```

```

66:  C
67:  C do 100 ITSK = 1, NTASK
68:  C   if( JVPPS(1).ne.0 ) then
69:  C     IVPPPR(ITSK) = IPR
70:  C   else
71:  C     IVPPPR(ITSK) = 75 + ITSK
72:  C   ... unnamed file
73:  C     open( IVPPPR(ITSK), form='FORMATTED',status='SCRATCH' )
74:  C   ... named file
75:  C     BUFF = ' '
76:  C     write(BUFF,(''vppprt.'' ,i2)) ITSK + 75
77:  C     call CCOMP( BUFF, LEN(BUFF), BUFF, II )
78:  C     open( IVPPPR(ITSK),form='FORMATTED',file =BUFF(:ICLEN(BUFF)),
79:  C       & status='UNKNOWN' )
80:  C
81:  C   end if
82:  C
83:  C   write(ipr,*) ' VPP multi task : opened I/O unit ',IVPPPR(ITSK)
84:  C 100 continue
85:  C
86:  C   LOCK1 = 1
87:  C   LOCK2 = 2
88:  C
89:  C   ... I/O unit iub is used to collect fission source data if necessary
90:  C
91:  C   call RWIND( IUB )
92:  C
93:  C ==== Parallel region starts here =====
94:  C
95:  C   call GETTOD( TT2 )
96:  C
97:  C!xocl parallel region
98:  C
99:  C   IDTASK = IDVPROC()
100:  C
101:  C-----
102:  C ... setting of printout I/O unit (IOW) and random number.
103:  C-----
104:  C
105:  C   call PRPARA( NTASK, A(LIRANT), A(LIRSUT), A(LIRSMT), A(LIRSLT) )
106:  C
107:  C-----
108:  C ..... Input Body of restart file
109:  C (header is already input in INTRO2 routine)
110:  C-----
111:  C   call GETTOD( TT0 )
112:  C
113:  C   if ( JREST.ne.0 ) then
114:  C!xocl spread do
115:  C   do 110 ITSK = 1, NTASK
116:  C
117:  C!xocl lockon LOCK1
118:  C   call RESTIN( IOW, H, ITSK, TITLE )
119:  C!xocl end lockon
120:  C
121:  C 110 continue
122:  C!xocl end spread
123:  C   end if
124:  C   call GETTOD( TT1 )
125:  C   TMINPP(IDVPROC()) = TT1 - TT0
126:  C
127:  C   if ( JSTOP.eq.0 .and. JNRUN.eq.0 ) then
128:  C     do I = 1, 1000
129:  C       write(IOW,*) I, ' %%% start action task ', IDVPROC(),
130:  C       & ' iow = ', IOW

```


src/gmvp/actvpp.f

```

131: c          end do
132:
133: !xocl spread do
134:   do 120 ITSK = 1, NTASK
135:     call GETTOD( TT0 )
136:     write(IOW,*) '##### PE      : ', IDVPROC()
137:     write(IOW,*) '##### IRAND : ', IRAND
138:     CCC      IDTASK = IDVPROC()
139: C-----
140: C ..... Monte Carlo run
141: C-----
142:   call ACTION( TITLE, A, H, CHA )
143: C-----
144: C ..... OUTPUT CPU-TIME INFORMATION (EVENT-WISE) IF REQUIRED .....
145: C-----
146:   if ( JVMNT.ne.0 ) then
147:     call HEADER( IOW, 'DETAILED CPU TIME MONITOR' )
148:     call VMNTRF( IOW )
149:   end if
150: C
151:   call GETTOD( TT1 )
152:   TMACT(IDVPROC()) = TT1 - TT0
153:   120   continue
154:
155: !xocl end spread
156:
157:   end if
158: C
159:   call GETTOD( TT0 )
160: C
161: C-----
162: C ..... Output Body of restart file
163: C-----
164:   if( JRSTF .ne. 0 ) then
165: C
166: C   ... take sum of NTHIST tempoary on KTHIST
167: C
168:     KTHIST = 0
169: !xocl spread do
170:   do 130 I = 1, NTASK
171:     KTHIST = KTHIST + NTHIST
172:   130   continue
173: !xocl end spread sum(KTHIST)
174: c
175: c
176:   if ( JEIGN.ne.0 ) then
177:     KBATCH = NBATCH
178:   else
179: !xocl spread do
180:   do 140 I = 1, NTASK
181:     KBATCH = KBATCH + NBATCH
182:   140   continue
183: !xocl end spread sum(KBATCH)
184:   end if
185: c
186: c   ... output header records ...
187: c
188: C   .... Using spread do only to use barrier ...
189: C
190: !xocl spread do
191:   do 142 itsk=1,ntask
192:     if ( IDTASK.eq.1 ) then
193:       call RWIND(IROT)
194:       call REST00( IOW, H, 1, TITLE, KTHIST, KBATCH )
195:     end if

```

```

196:   142   continue
197: !xocl end spread
198: C
199: C
200: C ... using barrier may cause error , but why ?
201: C
202: C!xocl barrier
203: c
204: c   ... output data body ...
205: c
206: !xocl spread do
207:   do 150 ITSK = 1, NTASK
208:
209: !xocl lockon LOCK2
210:   call RESTOT( IOW, H, 1, ITSK, TITLE )
211: !xocl end lockon
212:
213:   150   continue
214: !xocl end spread
215: C
216: !xocl spread do
217:   do 152 i=1,ntask
218:     if ( IDTASK.eq.1 ) then
219:       call RWIND(IROT)
220:     end if
221:   152   continue
222: !xocl end spread
223: C
224:   end if
225:   call GETTOD( TT1 )
226:   TMOUTP(IDVPROC()) = TT1 - TT0
227: C
228: C-----
229: C ..... Take sum of data on each task
230: C-----
231: C
232:   call GETTOD( TT0 )
233:
234:   call MTSUMS( TITLE, H )
235:
236:   call GETTOD( TT1 )
237:   TMSUM(IDVPROC()) = TT1 - TT0
238: C
239: C
240: C-----
241: C ... convert in-cell coordinates into absolute coordinates
242: C   for fission file output ...
243: C   (XXXF,YYYF,ZZZF)
244: C-----
245:   call GETTOD( TT0 )
246:   if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
247: C
248:   if ( JLATT.ne.0 ) then
249:     call CEL2AB( IOW, NFISB, NFBNK0,
250:       &          A, JDEBG, H(LXXXF), H(LYYYF),
251:       &          H(LZZZF), H(LLEVLf), H(LLZZF), H(LLPOSF), H(LLCRSF),
252:       &          H(LDFBK),KDFBK,MDFBK,
253:       &          H(PCA(1)), H(PCA(2)), H(PCA(3)) )
254:   end if
255: C
256: C ... XXXF,YYYF,ZZZF,INUF,EEEF & IBRGF on file
257: C
258: !xocl spread do
259:   do 160 ITSK = 1, NTASK
260:     if ( ITSK.gt.1 ) then

```

src/gmvp/actvpp.f

```

261: !xocl lockon LOCK2
262:      call VPPFOT( IUB, ITSK, NFISB, NFBNK0, NEST, JTLLT,
263:      &          H(LXXXF), H(LYYYF), H(LZZZF), H(LIZZF),
264:      &          H(LIBRGF), H(LIBSPF),
265:      &          H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
266: !xocl end lockon
267:      end if
268:      160      continue
269: !xocl end spread
270: C
271:      end if
272:
273:      call GETTOD( TT1 )
274:      TMOUTP( IDVPROC() ) = TMOUTP( IDVPROC() ) + TT1 - TT0
275: C
276:      call PESUM8( TMINPP(1), NTASK )
277:      call PESUM8( TMACT(1), NTASK )
278:      call PESUM8( TMOUTP(1), NTASK )
279:      call PESUM8( TMSUM(1), NTASK )
280: C
281:      if( IOW.ne.6 ) call rwind(IOW)
282:      IOW      = 6
283: C
284: !xocl end parallel
285: C
286:      call GETTOD( TT3 )
287:      TMPARA = TT3 - TT2
288: C
289: C-----
290: C ..... Output fission source file (out side of parallel region )
291: C-----
292: C
293:      if ( JEIGN.ne.0.and.JOSRC.ne.0 ) then
294: C
295: C          ... gather fission neutron bank data of tsak 2 - Ntask
296: C          output on I/O unit IUB
297: C
298:      call RWIND( IUB )
299:      call VPPFIN( IUB, NTASK, NFISB, NFBNK0, NEST, JTLLT, H(LXXXF),
300:      &          H(LYYYF), H(LZZZF), H(LIZZF), H(LIBRGF), H(LIBSPF),
301:      &          H(LDFBK), KDFBK, MDFBK, H(LIFBK), KIFBK, MIFBK )
302: C
303:      NUSE = NFISB
304:      call SCFOUT( 'GMVP', TITLE, NUSE, NFBNK0, NEST, NREG, A(LWGIF),
305:      &          NZONE, A(LKZREG), JDEBG, JTLLT, H(LXXXF), H(LYYYF),
306:      &          H(LZZZF), DUMMY, H(LIZZF), DUMMY, H(LIBRGF), H(LIBSPF),
307:      &          ' ', 1, IERR )
308:      end if
309: C
310: C-----
311: C ... collect printout on each task
312: C-----
313: C
314:      call GETTOD( TT0 )
315:      if( JVPSP(1).eq.0 ) then
316:      do 190 ITSK = 1, MPE
317: C
318:      call RWIND( IVPPPR(ITSK) )
319: C
320:      write(IOW,7001) ITSK
321: 7001      format(/lx,'@@@ OUPUT FROM TASK #',i3,' ON VPP-FORTRAN'//)
322: 170      BUFF = ' '
323:      read(IVPPPR(ITSK),'(A)',end =180) BUFF
324:      write(IOW,*) BUFF(:ICLEN2(BUFF))
325:      go to 170

```

```

326:
327:      180      close( IVPPPR(ITSK) )
328:      190      continue
329:      end if
330:      call GETTOD( TT1 )
331:      TMPRINT = TT1 - TT0
332: C
333: C/#ENDIF
334: C
335:      return
336:      end
337: C
338: C
339: C
340:      subroutine VPPFOT( IUB, ITSK, NFISB, NFBNK0, NEST, JTLLT, XXXF,
341:      &          YYYF, ZZZF, IZZF, IBRGF, IBSPF,
342:      &          DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK )
343: C=====
344: C purpose: output fission neutron bank data on task ITSK in
345: C Parallel mode on VPP (VPP-Fortran)
346: C
347: C Task #1 does not call this routine.
348: C
349: C history: programmed by M.Sasaki ( 18 Jul 1997 )
350: C 10 Feb 1999: output optional fission bank data
351: C=====
352: C/#IF PARA(VPP)
353: C
354:      real*8 XXXF(NFBNK0)
355:      real*8 YYYF(NFBNK0)
356:      real*8 ZZZF(NFBNK0)
357:      integer IZZF(NFBNK0)
358:      integer IBRGF(NFBNK0)
359:      integer IBSPF(NFBNK0,NEST)
360: C
361:      real*8 DFBK(NFBNK0,*)
362:      integer KDFBK(0:MDFBK)
363:      integer IFBK(NFBNK0,*)
364:      integer KIFBK(0:MIFBK)
365: C-----
366:      write(IUB) ITSK, NFISB
367:      write(IUB) (XXXF(I),I=1,NFISB)
368:      write(IUB) (YYYF(I),I=1,NFISB)
369:      write(IUB) (ZZZF(I),I=1,NFISB)
370:      write(IUB) (IZZF(I),I=1,NFISB)
371:      if ( JTLLT.ne.0 ) then
372:      write(IUB) (IBRGF(I),I=1,NFISB)
373:      write(IUB) ((IBSPF(I,N),I=1,NFISB),N=1,NEST)
374:      end if
375: C
376:      if ( KDFBK(0).gt.0 ) then
377:      do 100 K=1,KDFBK(0)
378:      write(IUB) (DFBK(I,K),I=1,NFISB)
379: 100      continue
380:      end if
381: C
382:      if ( KIFBK(0).gt.0 ) then
383:      do 110 K=1,KIFBK(0)
384:      write(IUB) (IFBK(I,K),I=1,NFISB)
385: 110      continue
386:      end if
387: C
388: C/#ENDIF
389:      return
390:      end

```

src/gmvp/actvpp.f

```
391: C
392: C
393:     subroutine VPPFIN( IUB,  NTASK, NFISB, NFBNK0,NEST,  JTLLT, XXXF,
394:     &                  YYYYF, ZZZF, IZZF,  IBRGF, IBSPF,
395:     &                  DFBK,KDFBK,MDFBK,IFBK,KIFBK,MIFBK )
396: C=====
397: C purpose: input fission neutron bank data on scratch file in
398: C   Parallel mode on VPP (VPP-Fortran)
399: C
400: C   Only task 1 calls this routine.
401: C
402: C history: programmed by M.Sasaki ( 18 Jul 1997 )
403: C=====
404: C/#IF PARA(VPP)
405:
406:     real*8 XXXF(NFBNK0)
407:     real*8 YYYYF(NFBNK0)
408:     real*8 ZZZF(NFBNK0)
409:     integer IZZF(NFBNK0)
410:     integer IBRGF(NFBNK0)
411:     integer IBSPF(NFBNK0,NEST)
412: C
413:     real*8 DFBK(NFBNK0,*)
414:     integer KDFBK(0:MDFBK)
415:     integer IFBK(NFBNK0,*)
416:     integer KIFBK(0:MIFBK)
417: C-----
418:     do 120 N = 2, NTASK
419:         read(IUB) ITSK, NFISB0
420:         K1      = NFISB + 1
421:         NFISB   = NFISB + NFISB0
422:
423:         read(IUB) (XXXF(I),I=K1,NFISB)
424:         read(IUB) (YYYYF(I),I=K1,NFISB)
425:         read(IUB) (ZZZF(I),I=K1,NFISB)
426:         read(IUB) (IZZF(I),I=K1,NFISB)
427:
428:         if ( JTLLT.ne.0 ) then
429:             read(IUB) (IBRGF(I),I=K1,NFISB)
430:             read(IUB) ((IBSPF(I,J),I=K1,NFISB),J=1,NEST)
431:         end if
432:
433:         if ( KDFBK(0).gt.0 ) then
434:             do 100 K=1,KDFBK(0)
435:                 read(IUB) (DFBK(I,K),I=K1,NFISB)
436:             100 continue
437:         end if
438:
439:         if ( KIFBK(0).gt.0 ) then
440:             do 110 K=1,KIFBK(0)
441:                 read(IUB) (IFBK(I,K),I=K1,NFISB)
442:             110 continue
443:         end if
444:
445:     120 continue
446: C
447: C/#ENDIF
448:     return
449: end
```

src/gmvp/adjrev.f

```

1:      subroutine ADJREV( X,      NGP,      N )
2: C=====
3: C  PURPOSE: REVERSAL OF FIRST INDECES OF A REAL  ARRAY X(NGP,N)
4: C  ENTRY: FISREV
5: C          USED FOR ADJOINT REVERSAL FISSION SPECTRUM AND SIGF
6: C          (SIGF IS ALREADY REVERSED WITH RESPECT TO ENERGY)
7: C=====
8:      real      X(NGP,N), XX
9: C
10:     NGPH      = NGP/2
11: C
12: *VOCL LOOP,NOVREC
13: do 110 J = 1, N
14:   do 100 I = 1, NGPH
15:     XX      = X(I,J)
16:     X(I,J)  = X(NGP+1-I,J)
17:     X(NGP+1-I,J)  = XX
18:   100 continue
19: 110 continue
20: C
21:   return
22: end
23: C
24:   subroutine FISREV( X,      NGP,      N,      STOTX, SNUFX, SNAPX, SGPX,
25: &      SNFPX, S2DPX, S2PPX, NMEDX, NSPX, JEIGN,
26: &      JFISS, JNEUT, JPHOT )
27: C
28: C**** EXCHANGE OF FISSION SPECTRUM AND NU*FISSION CROSS SECTION
29: C
30:   real      X(NGP,N), XX, STX, STS, S2DP2
31:   real      STOTX(NGP,1), SNUFX(NGP,1), SNAPX(NGP,1), SGPX(NGP,1),
32: &      SNFPX(NGP,1), S2DPX(2,NGP,1), S2PPX(3,NGP,1)
33:   include ' ../shared/INC/_IUNIT'
34: C
35:   NN      = MIN(N,NMEDX)
36: C
37:   do 160 J = 1, NN
38:     STX      = 0.0
39:     STS      = 0.0
40:     do 100 I = 1, NGP
41:       STX      = STX + SNUFX(I,J)
42:       STS      = STS + X(I,J)
43:   100 continue
44: C
45:   if ( STX.gt.0. ) then
46:     write(IPR,7000) J, STS
47: 7000   format(1X,' == SUMMATION OF FISSION SPECTRUM OF ',
48: &      'FISSIONABLE MATERIAL (MAT=',I2,' ) = ',1PE12.4)
49:     do 110 I = 1, NGP
50:       SNUFX(I,J)  = SNUFX(I,J) /STX
51:   110 continue
52:   else
53:     write(IPR,*) ' == NON-FISSIONABLE MATERIAL (MAT= ', J, ' )'
54:     STX      = 0.0
55:     do 120 I = 1, NGP
56:       SNUFX(I,J)  = 0.0
57:   120 continue
58:   end if
59: C
60:   if ( STS.gt.0. ) then
61:     do 130 I = 1, NGP
62:       X(I,J)  = X(I,J) /STS
63:   130 continue
64:   else
65:     do 140 I = 1, NGP

```

```

66:       X(I,J)  = 0.0
67:   140 continue
68:   end if
69: C
70:   do 150 I = 1, NGP
71:     XX      = X(I,J)
72:     X(I,J)  = SNUFX(I,J)
73:     SNUFX(I,J)  = XX*STX
74:     if ( STOTX(I,J).gt.0. ) then
75:       SNFPX(I,J)  = SNUFX(I,J) / STOTX(I,J)
76:     else
77:       SNFPX(I,J)  = 0.0
78:     end if
79: C
80:     S2DP2      = SNAPX(I,J)
81:     if ( I.le.NGPX.and.JPHOT*JNEUT.ne.0 ) then
82:       S2DP2      = S2DP2 + SGPX(I,J)
83:     end if
84:     S2DPX(2,I,J)  = S2DP2
85:     if ( JEIGN.eq.0.and.JFISS.ne.0 ) then
86:       S2DPX(1,I,J)  = S2DP2 + SNFPX(I,J)
87:     else
88:       S2DPX(1,I,J)  = S2DP2
89:     end if
90:     if ( S2DPX(1,I,J).gt.0. ) then
91:       S2PPX(1,I,J)  = SNAPX(I,J) / S2DPX(1,I,J)
92:       S2PPX(2,I,J)  = S2DP2 / S2DPX(1,I,J)
93:     else
94:       S2PPX(1,I,J)  = 1.0
95:       S2PPX(2,I,J)  = 1.0
96:     end if
97:     if ( S2DPX(2,I,J).gt.0. ) then
98:       S2PPX(3,I,J)  = SNAPX(I,J) / S2DPX(2,I,J)
99:     else
100:       S2PPX(3,I,J)  = 1.0
101:     end if
102:   150 continue
103:   160 continue
104: C
105:   return
106: end

```

src/gmvp/angles.f

```

1:      subroutine ANGLES( IG1,   JG1,   MX )
2: C=====
3: C  PURPOSE: TO CALCULATE GENERALIZED GAUSSIAN QUADRATURE.
4: C      ( FROM MORSE CODE
5: C      MODIFIED TO FORTRAN77 STYLE BY M.SASAKI 3/16/88 )
6: C  CALLED IN : JNPUT
7: C  CALLS: GETMUS,FIND,Q,BADMOM
8: C=====
9: C
10:      include '../shared/INC/_IOUNIT'
11: C
12:      common /MEANS/  NM,      NV,      XMU(21),      VAR(20),
13:      & XNORML(20)
14:      common /RESULT/ POINT(21),  WEIGHT(21),  ROOT(21,21)
15:      common /MOMENT/ NMOM,  XMOMNT(25),  NF,  F(25)
16:      common /LOCSIG/ ISTART,  ISCGOG,  INABOG,  IGABOG,  IFPORG,  IFNGP,
17:      & IPFPOG,  IDSGOG,  IPRBNG,  IPRBGG,  ISCANG,  ISCAGG,  ISPORG,
18:      & ISPORT,  INPBUF,  ISIGOG,  INFPOG,  IABSOG,  ITOTSG,  NGP,
19:      & NDS,  NGG,  NDSG,  INGP,  INDS,  NMED,  NELEM,
20:      & NMIX,  NCOEF,  NSCT,  NTS,  NTG,  NDSNGP,  NDSNGG,
21:      & IADJ,  NME,  LOC,  INGS,  INSG,  IL,  IO,
22:      & KKK,  IXTAPE,  IDEL,  ITEMEL,  ITEMG,  IRSG,  IRDSG,
23:      & ISTR,  IPRIN,  IFMU,  IMOM,  IDTF,  ISTAT,  IPUN,
24:      & NUS,  NGN,  IHT,  INUS,  INUSN,  INGN,  INGNP,
25:      & INNN,  IGGG
26: C
27: C  NM= NUMBER OF MU VALUES ACCEPTED
28: C  NV= NUMBER OF VAR VALUES ACCEPTED
29: C  NM= NV OR NV+1
30: C  NP= NUMBER OF ANGLES IN DISCRETE DISTRIBUTION
31: C  NACC=NUMBER OF MOMENTS ACCEPTED
32: C
33:      do 100 K = 1, NSCT
34:          WEIGHT(K) = 0.0
35:          POINT(K) = 0.0
36:      100 continue
37: C
38: C  .... calculate parameters to evaluate orthogonal polynomials.
39: C
40:      call GETMUS
41: C
42:      if ( IMOM.gt.0 ) write(IO,7000) (XMOMNT(I),I=1,NMOM)
43:      7000 format(1X,'MOMENTS ',4X,1P10E12.5/(1X,11E12.5))
44: C
45: C
46: cc 20 IF(ABS(XMU(1)).GT.1.0) GOTO 125
47: C
48:      if ( ABS(XMU(1)).gt.1.0 ) then
49:          JG2 = IG1 + JG1 - 1 - NUS
50:          write(IO,7020) XMU(1), IG1, JG2, MX
51:      7020 format('// FIRST MOMENT REJECTED.  M(1)=' ,E10.3,5X,'IN=' ,I3,5X,
52:      & 'OUT=' ,I3,5X,'MIXTURE=' ,I3)
53:          NM = 0
54:          NV = 0
55:          call BADMOM
56: C
57: C  .... RETRY ...
58: C
59:          NM = 1
60:          if ( ABS(XMU(1)).gt.1.001 ) then
61:              XMU(1) = 1.0
62:              INUSN = 1 + INUSN
63:          elseif ( XMU(1).gt.1.0 ) then
64:              XMU(1) = 1.0
65:              call CNTERR('WARNING')
66:
67:          write(IMG,*) '!!! FIRST MOMENT IS MODIFIED TO 1.0'
68:          elseif ( XMU(1).lt.-1.0 ) then
69:              XMU(1) = -1.0
70:              call CNTERR('WARNING')
71:              write(IMG,*) '!!! FIRST MOMENT IS MODIFIED TO -1.0'
72:          endif
73:      end if
74: C
75:      ROOT(1,1) = XMU(1)
76: C
77:      if ( NM.gt.1 ) then
78:          do 110 L = 2, NM
79:              call FIND( L, NCK )
80:              cccccccccc IF(NCK.GT.0) GOTO 120
81:              if ( NCK.ne.0 ) then
82:                  JG2 = IG1+JG1-1 -NUS
83:                  write(IMG,7040) L, NCK, MX, IG1, JG2
84:                  call CNTERR( 'WARNING' )
85:                  NM = L - 1
86:                  NV = L - 1
87:                  go to 120
88:              end if
89:          110 continue
90:      end if
91: C
92:      if ( NM.gt.NV ) go to 130
93: C
94:      120 XMU(NV+1) =
95:      & -VAR(NV)*(Q(NV-1,1.0)/Q(NV,1.0)+Q(NV-1,-1.0)/Q(NV,-1.0))/2.
96:      call FIND( NV+1, NCK )
97:      if ( NCK.ne.0 ) then
98:          JG2 = IG1+JG1-1 -NUS
99:          write(IMG,7040) NV + 1, NCK, MX, IG1, JG2
100:          call CNTERR( 'WARNING' )
101:          NV = NV - 1
102:      end if
103: C
104:      7040 format(1X,'!!! ROOT OF ',I2,'TH ORDER ORTHOGONAL POLYNOMIAL',
105:      & ' IS OUTSIDE OF (-1,1). ( ',I2,': +1/-1=>1.0/<-1.0)',
106:      & ' MAT=' ,I5,' GRP. ',I3,' TO ',I3)
107: C
108:      130 NP = NV + 1
109:      NACC = NM + NV
110:      do 140 K = 1, NP
111:          POINT(K) = ROOT(K,NP)
112:      140 continue
113:      do 160 K = 1, NP
114:          SUM = 1.0
115:          if ( NV.le.0.0 ) then
116:              WEIGHT(1) = 1.
117:              go to 170
118:          else
119:              do 150 L = 1, NV
120:                  SUM = SUM + (Q(L,POINT(K)))**2/XNORML(L)
121:              150 continue
122:              WEIGHT(K) = 1.0/SUM
123:          end if
124:      160 continue
125: C
126:      170 do 190 K = 1, NP
127:          BIG = WEIGHT(K)
128:          J = K
129:          do 180 L = K, NP
130:              if ( WEIGHT(L).gt.BIG ) then
131:                  BIG = WEIGHT(L)

```

src/gmvp/angles.f

```
131:          J          = L
132:          end if
133: 180    continue
134:          WEIGHT(J)    = WEIGHT(K)
135:          WEIGHT(K)    = BIG
136:          SPOINT      = POINT(K)
137:          POINT(K)    = POINT(J)
138:          POINT(J)    = SPOINT
139: 190    continue
140: C
141:    if ( NACC-NMOM.lt.0.and.IPUN.gt.0 ) then
142:      call BADMOM
143:      JG2      = IG1 + JG1 - 1 - NUS
144:      write(I0,7060) NACC, IG1, JG2, MX
145: 7060    format(// 'NUMBER OF MOMENTS ACCEPTED = ',I3,'      IN=',I3,
146: &          '      OUT=',I3,5X,'MIXTURE=',I3,//
147: &          ' * * * * * ')
148:    end if
149: C
150:    return
151: C
152: c 120 NM=L-1
153: c    NV=L-1
154: c    GO TO 45
155: C
156: C
157: c 125 JG2=IG1+JG1-1 -NUS
158: c    WRITE(I0,1030) XMU(1),IG1,JG2,MX
159: c1030 FORMAT(// 'FIRST MOMENT REJECTED.  M(1)=' ,E10.3,5X, 'IN=' ,
160: c    1      I3,5X, 'OUT=' ,I3,5X, 'MIXTURE=' ,I3)
161: c    NM=0
162: c    NV=0
163: c    CALL BADMOM
164: C*****
165: C      (RETRY)
166: c    NM=1
167: c    XMU(1)=1.0
168: c    INUSN=1 +INUSN
169: c    GO TO 20
170: C*****
171: C
172:    end
```

src/gmvp/apmatx.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine APMATX( SIGT, NSIG )
3: C=====
4: C PURPOSE: TO RETRIEV PROCESSED MACRO-DDX DATA FROM UNIT IOMDX
5: C   (WRITTEN IN 'MACRO4')
6: C CALLED IN: JNPUT
7: C CALLS: none
8: C=====
9:   common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
10:  & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPOG,
11:  & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
12:  & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
13:  & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
14:  & IADJ, NME, LOC, INGS, INSG, II, IO,
15:  & KKK, IXTAPE, IDEL, ITEMG, IRSG, IRDSG,
16:  & ISTR, IPRIN, IFMU, IMOM, IDTF, ISTAT, IPUN,
17:  & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
18:  & INNN, IGGG, IPNGP, IPSPOG, IPSGOG, IANG, INUFIS
19: CCCCC COMMON/DDX/ IDDX,MAXU,MAXSD,IMAX,NLASTD,MAXDDX
20:   real SIGT(1)
21:   integer NSIG(1)
22: C
23:   include '../shared/INC/_IOUNIT'
24:   include 'INC/_IOUNIT2'
25: C
26: C   NGP : PRIMARY GROUP
27: C   NGG : SECONDARY GROUP
28: C   INGP: TOTAL GROUP = NGG+NGP = NTG
29: C
30:   NSRT = ISPORT + 1
31:   NLSTD = ISPORT + INGP*NSCT + INGP + 6 + NSCT + NGG
32: C   NLASTD = ISPORT + INGP*NSCT*3
33:   NBMC = (NDSNGP+NDSNGG)*NSCT*3
34:   call RWIND( IOMDX )
35:   do 150 MED = 1, NMED
36:     I1 = (MED-1)*NBMC + IPRBNG
37:     I3 = (MED-1)*INGP*NSCT + ISPOG
38:     I5 = (MED-1)*INGP*NSCT*3 + ISCANG
39:     I7 = (MED-1)*NGP*NGG*3 + IPNGP
40:     K1 = (MED-1)*NTG + ISTART
41:     K2 = (MED-1)*NTG + ISCCOG
42:     K3 = (MED-1)*NTG + INUFIS
43:     K4 = (MED-1)*NTG + INUS
44:     K5 = (MED-1)*NGP + IGABOG
45:   do 140 IG = 1, INGP
46:     read(IOMDX,end =9000) (SIGT(I),I=NSRT,NLSTD)
47:     ITE = NSIG(INGS+IG)
48:     NDSK = NSIG(INNN+IG)
49:     SIGT(K1+IG) = SIGT(NSRT+4)
50:     SIGT(K2+IG) = SIGT(NSRT)
51:     SIGT(K3+IG) = SIGT(NSRT+3)
52:     SIGT(K4+IG) = 0.0
53:     if ( IG.le.NGP ) SIGT(K5+IG) = SIGT(NSRT+5)
54: C
55:     if ( ISTAT.gt.0 ) then
56:       I4 = I3 + (IG-1)*NSCT
57:       J = NSRT + 5 + INGP + INGP*NSCT
58:       do 100 MU = 1, NSCT
59: C         DOMECA = 6.283185307*(SIGT(IANG+MU)-SIGT(IANG+MU+1))
60:         DOMECA = SIGT(IANG+MU) - SIGT(IANG+MU+1)
61:         SIGT(I4+MU) = SIGT(J+MU) /DOMECA*2.0
62:       100 continue
63:     end if
64: C
65:     NLST = ISPORT + NDSK*NSCT*3

```

```

66:     read(IOMDX,end =9000) (SIGT(I),I=NSRT,NLST)
67:     I2 = I1 + ITE*NSCT*3
68:     do 110 I = 1, NDSK*NSCT*3
69:       SIGT(I2+I) = SIGT(ISPORT+I)
70:     110 continue
71: C
72:     if ( ISTAT.gt.0 ) then
73:       NLST = ISPORT + NSCT*3
74:       read(IOMDX,end =9000) (SIGT(I),I=NSRT,NLST)
75:       I6 = I5 + (IG-1)*NSCT*3
76:       do 120 I = 1, NSCT*3
77:         SIGT(I6+I) = SIGT(ISPORT+I)
78:     120 continue
79:     end if
80: C
81:     if ( NGG.ne.0 ) then
82:       NLST = ISPORT + NGG*3
83:       read(IOMDX,end =9000) (SIGT(I),I=NSRT,NLST)
84:       if ( IG.le.NGP ) then
85:         I8 = I7 + (IG-1)*NGG*3
86:         do 130 I = 1, NGG*3
87:           SIGT(I8+I) = SIGT(ISPORT+I)
88:     130 continue
89:         end if
90:       end if
91: C
92:     140 continue
93:     150 continue
94:     return
95: C
96: 9000 call PRSTOP( 1, 'END OF FILE IOMDX IN SUB.APMATX' )
97:     stop 888
98:     end

```

src/gmvp/badmom.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BADMOM
3: C=====
4: C PURPOSE: TO PROVIDE ERROR PRINTOUT FOR MOMENTS WHICH HAVE BEEN
5: C REJECTED. (LEGENDRE EXPANSION MODE)
6: C CALLED IN: ANGLES
7: C CALLS: MAMENT
8: C=====
9:   real MU, MOMENT, NORM, MUT, MUB, MOM, MOMT, MOMB, L
10:  common /MOMENT/ NMOM, MOMENT(25), NF, F(25)
11:  common /MEANS/ NN, N, MU(21), VAR(20), NORM(20)
12:  common /QAL/ QR(21), A(20,21), L(21)
13:  common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
14: & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPOG,
15: & ISPORT, INPBIF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
16: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
17: & NMLX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
18: & IADJ, NME, LOC, INGS, INSG, IL, IO,
19: & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRBSG,
20: & IPRIN, IPRIN, IFMU, IMOM, IDTE, ISTAT, IPUN,
21: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
22: & INNN, IGGG
23: c
24: c
25:   NM = N + NN
26:   NBAD = NM + 1
27:   NM1 = N - 1
28:   NP1 = N + 1
29: c
30:   write(I0,9000) NBAD
31: 9000 format(/1X,'-----MOMENT(' ,I2,' ) IS BAD, OUTPUT FROM BADMOM')
32: c
33:   if ( N.eq.0 ) then
34:     if ( NN.eq.0 ) then
35:       MUT = 1.
36:       MUB = -1.
37:       MU(1) = MOMENT(1)
38:       MOMT = 1.
39:       MOMB = -1.
40:       go to 120
41:     end if
42:     VART = 2./ (1./Q(1,1.)-1./Q(1,-1.))
43:     VARB = 0.
44:     MOMB = MOMENT(1)**2
45:     MOMT = MOMB + VART
46:     go to 150
47:   end if
48:   if ( N.ne.NN ) go to 130
49: c
50: 100 MUT = 1. - VAR(N)*Q(NM1,1.) /Q(N,1.)
51: c
52: c THE 2N+1 MOMENT IS BAD
53: c
54:   MUB = -1. - VAR(N)*Q(NM1,-1.) /Q(N,-1.)
55:   MU(NP1) = QR(NP1) - QR(N)
56:   MOM = NORM(N)*QR(N)
57:   do 110 K = 1, N
58:     MOM = MOM - A(N,K)*MOMENT(N+K)
59: 110 continue
60:   MOMT = MOM + NORM(N)*MUT
61:   MOMB = MOM + NORM(N)*MUB
62: c
63: c READY TO OUTPUT MOMB,MOMENT(NBAD),MOMT,MUB,MU(NP1),MUT
64: c
65: 120 write(I0,9020) MUB, NP1, MU(NP1), MUT

```

```

66: 9020 format(' MUBOT =',F15.9,' MU(' ,I2,' ) = ',F14.9,' MUTOP =',F16.9)
67:   go to 160
68: c
69: 130 VART = 2./ (Q(N,1.) /Q(NP1,1.)-Q(N,-1.) /Q(NP1,-1.))
70: c
71: c NBAD IS 2N+2, VAR(N+1) IS BAD
72: c
73:   MOMB = 0.
74:   VARB = 0.
75:   do 140 K = 1, NP1
76:     MOMB = MOMB - A(NP1,K)*MOMENT(N+K)
77: 140 continue
78:   MOMT = MOMB + VART*NORM(N)
79: c
80: c READY TO OUTPUT MOMB,MOMENT(NBAD),MOMT,VAR(NP1),VART
81: c
82: 150 write(I0,9040) VARB, NP1, VAR(NP1), VART
83: 9040 format(' VARBOT =',F15.9,' VAR(' ,I2,' ) =',F15.9,' VARTOP = ',F15.9
84: & )
85: 160 MOM = MOMENT(NBAD)
86:   FA = F(NBAD)
87:   MOMENT(NBAD) = MOMT
88:   call MAMENT( NBAD )
89:   FT = F(NBAD)
90:   MOMENT(NBAD) = MOMB
91:   call MAMENT( NBAD )
92:   FB = F(NBAD)
93:   MOMENT(NBAD) = MOM
94:   if ( MOM.gt.MOMT ) then
95:     DELM = MOM - MOMT
96:     DELF = FA - FT
97:   else
98:     DELM = MOM - MOMB
99:     DELF = FA - FB
100:   end if
101:   RANGEM = MOMT - MOMB
102:   RANGEF = FT - FB
103: c
104: c NOW READY TO OUTPUT FB,FA,FT
105: c
106:   write(I0,9060) MOMB, NBAD, MOMENT(NBAD), MOMT, RANGEM, DELM
107: 9060 format(' MOMBOT =',F15.9,' MOM(' ,I2,' ) =',F15.9,' MOMTOP = ',
108: & F15.9,5X,'RANGE = ',F9.6,' ERROR = ',F9.6)
109:   write(I0,9080) FB, NBAD, FA, FT, RANGEF, DELF
110: 9080 format(' FBOT =',F15.9,' F(' ,I2,' ) =',F15.9,' FTOP = ',
111: & F16.9,5X,'RANGE = ',F9.6,' ERROR = ',F9.6)
112:   return
113: end

```


src/gmvp/bprogv.f

```
1:      subroutine BPROGV
2: C=<GMVP>=====
3: C purpose: initialization of "PROGV" strings (program version
4: C      description)
5: C=====
6: C*
7: C*..... DESCRIPTION OF PROGRAM VERSION  ETC. (GMVP).....
8: C*
9:      include 'INC/_PROGV'
10: C
11:      character*64 PROGV0(40)
12: C
13:      data (PROGV0(I),I=1,15)/
14: 1'      GMVP : JAEA Monte Carlo transport code
15: 2'      with multi-group cross section library
16: 3'
17: 4'
18: 5'      *** Main capability of GMVP ***
19: 6'
20: 7'      PROBLEM TYPE      : Fixed source & eigenvalue.
21: 9'      CROSS SECTION    : Multigroup DDX or PL.
22: A'      GEOMETRY         : Combinatorial Geometry with
23: 1'                        rectangular & hexagonal lattice.
24: 2'      GEOMETRY CHECK   : 2-D slice image with characters.
25: 3'      FIXED SOURCE     : POINT, SPHERE & RECTANGULAR GEOMETRY,
26: 4'                        ISOTROPIC & MONODIRECTIONAL.
27: 5'      ESTIMATORS       : COLLISION & TRACK LENGTH ESTIMATOR.
28: 6'                        POINT ESTIMATOR.
29:
30: CCC 8'      TRACKING PROCEDURE : Event selection or zone selection.
31: C*
32:      data (PROGV0(I),I=16,30)/
33: 7'      BOUNDARY CONDITION : VACUUM, PERFECT REFLECTION &
34: 8'                        WHITE REFLECTION & PERIODIC.
35: 9'
36: A'      MATERIAL NUMBER ASSIGNMENT :
37: 1'
38: 2'          .GE. 1 : REAL MEDIUM.
39: 3'          0 : INNER VOID.
40: 4'          -1 TO -998 : LATTICE.
41: 5'          -1000 : OUTER VOID.
42: 6'          -1001 TO -1999 : ALBEDO MATERIAL (NOT IN USE)
43: 7'          -2000 : PERFECT REFLECTOR (MIRROR).
44: 8'          -3000 : WHITE REFLECTOR.
45: 9'          -4000 : PERIODIC BOUNDARY.
46: A'
47: A'
48: C*
49:      data (PROGV0(I),I=31,40)/
50: 1'      FILES : RESTART DATA READ FROM I/O UNIT # 10
51: 2'                        WRITTEN ON I/O UNIT # 20
52: 3'      BINARY OUTPUT WRITTEN ON I/O UNIT # 30
53: 4'
54: 5'
55: 6'
56: 7'
57: 8'
58: 9'
59: A'RESTART FILE TYPE 3.3 (AUG 2003)
60: C A'RESTART FILE TYPE 3.2 (FEB 1999)
61: CCC A'RESTART FILE TYPE 3.0 (MAY 1997)
62: C A'RESTART FILE TYPE 2.1 (MAY 1994)
63: C*CCCCC 40'TH LINE OF PROGV IS USED AS RESTART FILE INFORMATION
64: C
65:
66:      NLines = 40
67:      do 100 I=1,NLines
68:          PROGV(I) = PROGV0(I)
69:      100 continue
70: C
71:      return
72:      end
```

src/gmvp/cadenz.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine CADENZ( TITLE, A, H, CHA, LIMIT, LIMITL, LIMITC )
3: C=====
4: C  PURPOSE:  PROCESSING AFTER MONTE CARLO RUN
5: C  CALLED IN: CENTER
6: C  CALLS: RESTO,TALLYO
7: C=====
8:      real A(*)
9:      real H(*)
10:     character*4 CHA(*)
11: C
12:     character TITLE(2)*72
13: C
14: CCCC include '../shared/INC/_ARRAY'
15: C include '../shared/INC/_LISTOFF'
16: C
17:     include 'INC/_KPIDS'
18:     include 'INC/_NGPS'
19: C
20:     include 'INC/_FLAGS'
21:     include '../shared/INC/_SIZES'
22:     include 'INC/_SIZES2'
23:     include '../shared/INC/_PGEOM'
24:     include 'INC/_XWORK'
25:     include '../shared/INC/_PTALYO'
26:     include 'INC/_PTALY'
27:     include '../shared/INC/_PTLSP'
28:     include '../shared/INC/_STALY'
29:     include 'INC/_XBANK'
30:     include 'INC/_STACK'
31:     include 'INC/_WKTLO'
32:     include '../shared/INC/_LISTON'
33: C
34:     include '../shared/INC/_WORDL'
35:     include '../shared/INC/_IOUNIT'
36:     include 'INC/_IOUNIT2'
37: C
38: C ... local variables ...
39: c##<2007/03/14:PN3:
40:     character*1 NUCPN_CDMY
41:     integer      NUCPN_DMY
42:     NUCPN_DMY = 0
43: c##
44: C
45: C ..... OUTPUT TO RESTART FILE .....
46: C
47: c      call RESTO( NGROUP, NRESP, NREG, NTREG, NTHIST, NLENG, H(LWSUM),
48: c &              H(LWLEK), IRAND, TITLE, NBATCH, NMEMO, NZONE, NSKIP,
49: c &              H(LKMEMO), H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
50: c &              H(LNCNTR), H(LWCNTR), H(LXAVT), H(LAAVT), H(LEAVT),
51: c &              NFISB,NFBANK,NFBNK0,
52: c &              H(LXXXF), H(LYYYF), H(LZZZF), H(LIZZF), H(LXSOC),
53: c &              H(LXSXV), H(LXKEF), NLATT, NEST, H(LLEVLF), H(LLZZF),
54: c &              H(LLPOSF), H(LLCRSF), H(LIBRGF), H(LIBSPF), NPDET,
55: c &              H(LSFLPD), H(LSREPD), NETALY, H(LETALY), NLETAL )
56: C
57: C ... do tally output etc. in customized version.
58: C
59:     call ZZTALO( H(LWSUM), IOT, A, A, H, H, CHA )
60: C
61: C ..... PRINT OUT OF TALLY DATA .....
62: C
63:     call TALLYO( TITLE, A, CHA,
64: &              NTHIST, H(LWSUM), H(LWLEK), NPKIND, NGROUP,
65: &              NREG, NTREG, NRESP, NBATCH, NLENG, NGP1, NGP2, H(LSFLTR),

```

```

66: &              H(LSFLCL), H(LSRETR), H(LSRECL), A(LENGYB), A(LRVOL),
67: &              A(LTRVOL), NSKIP, NMXLG, NEITER,
68: &              H(LWCNTR), H(LXSOC), H(LXSXV), H(LXKEF),
69: &              NPDET, H(LSFLPD), H(LSREPD), H(LTFLHS), A(LITRNM),
70: &              CHA(LTNAMS), NNames,NSTALY, A(LTIMEB), NTIME,
71: &              H(P9(1)), H(P9(2)), H(P9(3)), H(P9(4)),
72: &              H(P9(5)), H(P9(6)), H(P9(7)),
73: &              H(P9(13)), H(P9(14)), H(P9(15)), H(P9(16)), H(P9(17)) )
74: C
75: C ..... Output of special tally data .....
76: C
77:     if( NSTALY.gt.0 ) then
78:         call STLOUT( H(LIETAL),NIETAL,NETALY,H(LETALY),NLETAL,
79: &              JEIGN, NBATCH,NSKIP, H(LWSUM),
80: &              A(LENGYB), KENGP, KPLIM, A(LTIMEB), A(LANGLB),
81: &              NETRV, METRV, NBETRV,A(LIETRV), H(LETRV), NMXLG,NEITER,
82: &              H(P9(11)), CHA(LRNM), NBINMX, NEDTMX, MXRGBN,
83: &              H(P9(15)),H(P9(16)),
84: c##<2007/03/14:PN3:
85: &              NUCPN_DMY, NUCPN_DMY, NUCPN_DMY, NUCPN_DMY, NUCPN_DMY,
86: &              NUCPN_DMY, NUCPN_CDMY, NUCPN_DMY, NUCPN_CDMY, NUCPN_DMY)
87: c##>
88:     end if
89: C
90: C ..... PRINT OUT OF MONITORING DATA .....
91: C
92:     call PRMNTR( IOT,A, CHA, JMNTR, NGROUP, NREG, JTIME, H(LNCLSN),
93: &              H(LWCLSN), H(LNLEAK), H(LWLEAK), H(LNABSB), H(LWABSB),
94: &              H(LNECUT), H(LWECUT), H(LNLCUT), H(LWLCUT), H(LNKILD),
95: &              H(LWKILD), H(LNSURV), H(LWSURV), H(LNCNTR), H(LWCNTR) )
96: C
97: C
98:     return
99: end

```

[illegible]

```

66: C
67: C ... initiate multitasking mode ( PVM )
68: C
69: C/#IF PARA( PVM )
70: *      call TSKPVM( NTASK0, TSKINF )
71: C/#ELSEIF PARA( MPI )
72: *      call TSKMPI( NTASK0, TSKINF )
73: C/#ELSEIF PARA( SX* CRAY* )
74: *      ITASK0 = 1
75: *      if( NTASK0 .eq. 0 ) NTASK0 = 1
76: *      IDTASK = 1
77: C/#ELSEIF PARA( VPP )
78: *      ITASK0 = 1
79: *      NTASK0 = MPE
80: *      IDTASK = IDVPROC()
81: C/#ELSE
82:      ITASK0 = 1
83:      NTASK0 = 1
84:      IDTASK = 1
85: C/#ENDIF
86: C
87: C ..... initialize option flags .....
88: C
89: ccc      call FLAGIN
90: C
91: C ..... PRINT DATE, TIME & PROGRAM DESCRIPTION .....
92: C
93:      call HIZUKE( XDATE )
94:      call JIKAN( TIME8 )
95:      call TOKEI( T00, 1 )
96:      call CPUTM( TC0 )
97: C
98:      call VERSTR
99:      call GETVER( VER, RDATE )
100: C
101:      BLANK = ' '
102:      write(IPR,'(/(8X,a,a))') (BLANK(:8-I),GMVP(i),I=1,7)
103: C
104:      write(IPR,7000) VER(:ICLEN2(VER)), RDATE(:ICLEN2(RDATE)),
105:      &              XDATE,TIME8
106: 7000 format(/8X,' VERSION : ',A, '      RELEASE DATE : ',A/
107:      &              8X,' EXECUTION DATE : ',A,'      TIME : ',A/)
108: C
109: C ==== PRINT COMMENTS TO PROGRAM ( VERSION, OPTIONS ETC.)
110: C      (main task only or if SUBTASK-PRINT(1)==2)
111: C
112: C << STYLE >>
113: C
114: C      *****
115: C      *
116: C      *      PROGV(1)
117: C      *      PROGV(2)
118: C      *      ....
119: C      *
120: C      *****
121: C
122:      if ( JSTPRN(1).ge.2.or.IDTASK.eq.1 ) then
123:      write(IPR,7010)
124: 7010      format(8X,'*',7('*****'),'*')
125: C
126:      write(IPR, '(8X,','*',5X,A60,5X,','*')')
127:      &      ' '
128:      &      (PROGV(I),I=1,40),
129:      &      ' '
130: C

```

```

66: C
67: C ... initiate multitasking mode ( PVM )
68: C
69: C/#IF PARA( PVM )
70: *      call TSKPVM( NTASK0, TSKINF )
71: C/#ELSEIF PARA( MPI )
72: *      call TSKMPI( NTASK0, TSKINF )
73: C/#ELSEIF PARA( SX* CRAY* )
74: *      ITASK0 = 1
75: *      if( NTASK0 .eq. 0 ) NTASK0 = 1
76: *      IDTASK = 1
77: C/#ELSEIF PARA( VPP )
78: *      ITASK0 = 1
79: *      NTASK0 = MPE
80: *      IDTASK = IDVPROC()
81: C/#ELSE
82:      ITASK0 = 1
83:      NTASK0 = 1
84:      IDTASK = 1
85: C/#ENDIF
86: C
87: C ..... initialize option flags .....
88: C
89: ccc      call FLAGIN
90: C
91: C ..... PRINT DATE, TIME & PROGRAM DESCRIPTION .....
92: C
93:      call HIZUKE( XDATE )
94:      call JIKAN( TIME8 )
95:      call TOKEI( T00, 1 )
96:      call CPUTM( TC0 )
97: C
98:      call VERSTR
99:      call GETVER( VER, RDATE )
100: C
101:      BLANK = ' '
102:      write(IPR,'/(8X,a,a)') (BLANK(:8-I),GMVP(i),I=1,7)
103: C
104:      write(IPR,7000) VER(:ICLEN2(VER)), RDATE(:ICLEN2(RDATE)),
105:      & XDATE,TIME8
106: 7000 format(/8X,' VERSION : ',A, '          RELEASE DATE : ',A/
107:      & 8X,' EXECUTION DATE : ',A,'      TIME : ',A/)
108: C
109: C==== PRINT COMMENTS TO PROGRAM ( VERSION, OPTIONS ETC.)
110: C      (main task only or if SUBTASK-PRINT(1)==2)
111: C
112: C << STYLE >>
113: C
114: C      *****
115: C      *
116: C      *      PROGV(1)
117: C      *      PROGV(2)
118: C      *      ....
119: C      *
120: C      *****
121: C
122:      if ( JSTPRN(1).ge.2.or.IDTASK.eq.1 ) then
123:      write(IPR,7010)
124: 7010      format(8X,'*',7('*****'),'')
125: C
126:      write(IPR, '(8X,\'***\',5X,A60,5X,\'***\')' )
127:      & ' ',
128:      & (PROGV(I),I=1,40),
129:      & ' '
130: C

```

src/gmvp/center.f

```

131:      write(IPR,7010)
132:      end if
133: C
134: C ==== OUTPUT RUNNIG SYSTEM & MODE ====
135: C
136:      call STAMP
137: C
138: C/#IF PARA(VPP)
139:      write(IPR,'(30X,'THE NUMBER OF PROCESSORS:',I3)') NTASK0
140:      call GETTOD(TT0)
141: C/#ENDIF
142: C
143: C ... printout I/O unit management for sub tasks
144: C
145:      call STPRN1('ON',JSTPRN,IDTASK)
146: C
147: C -----
148: C ..... PREPARATION FOR MONTE CARLO RUN
149: C -----
150: C
151:      call INTRO(TITLE,NTASK0,MLIMIT)
152: C
153: C ... disable printout I/O unit management for sub tasks
154: C
155:      call STPRN1('OFF',JSTPRN,IDTASK)
156: C
157: C/#IF PARA(MPI PVM)
158:      call MVPCOM_BARRIER(IERR)
159: C/#ENDIF
160: C/#IF UNIX
161:      call FLUSHSTD()
162: C/#ENDIF
163: C
164: C/#IF PARA(VPP)
165: *      call GETTOD(TT1)
166: *      TMINTR = TT1 - TT0
167: C
168: C=====
169: C FACOM VPP: access to an I/O unit is always synchronized,
170: C so error messages from a process may block all other processes!!
171: C
172: C In current implimentation, each process uses its own message
173: C output I/O unit (IOW).
174: C=====
175: C
176: C      do 4544 I=1,NTASK
177: C          open( I+75, form='FORMATTED',status='SCRATCH' )
178: C 4544 continue
179: C/#ENDIF
180: C
181: C
182: C -----
183: C ..... MONTE CARLO RUN
184: C -----
185: C
186: C
187: C ----- For Shared Memory Multi Task run -----
188: C
189: C/#IF PARA( CRAY* SX* )
190: *      if( JSTOP.eq.0 )
191: *          & call MTTASK( TITLE, A, H, CHA, IAINFL, LIMITL,
192: *          & A(LITASK),NTASK,MNTASK,ITASK0,
193: *          & A(LIRANT),ILOCK, IBARR, A(LIRSUT), A(LIRSMT), A(LIRSLT) )
194: C
195: C ----- For VPP Fortran -----

```

```

196: C
197: C/#ELSEIF PARA(VPP)
198: C
199:      call ACTVPP( TITLE, A, H, CHA )
200: C
201: C/#ELSEIF PARA(PVM MPI)
202: C
203: C ----- For Distributed Memory MultiTask -----
204: C
205:      if( JSTOP.EQ. 0 ) then
206:          call ACTMPP( TITLE, A, H, CHA )
207:      end if
208: C
209: C/#ELSE
210: C
211: C ----- For Single Task -----
212: C
213:      if( JSTOP.EQ. 0 ) then
214:          call ACTSGL( TITLE, A, H, CHA )
215:      end if
216: C
217: C/#ENDIF
218: C
219: C
220: C -----
221: C ..... TALLY DATA ANALYSIS & OUTPUT .....
222: C -----
223: C
224: C
225:      if( JSTOP.eq.0 ) then
226: C
227: C/#IF PARA(VPP)
228:          call GETTOD(TT0)
229: C/#ENDIF
230: C
231:          call CADENZ(TITLE, A, H, CHA, LIMIT, LIMITL, LIMITC )
232: C
233: C/#IF PARA(VPP)
234:          call GETTOD(TT1)
235:          TMCNZ = TT1 - TT0
236: C/#ENDIF
237: C
238:      end if
239: C
240:      call TOKEI( TT1, 0 )
241:      call CPUTM( TC1 )
242:      TEL1 = TT1-TT0
243:      TCP1 = TC1-TC0
244: C
245:      write(IPR,7200) TEL1,
246:      & int(TEL1/3600.0),int(mod(TEL1,3600.0)/60.0),int(mod(TEL1,60.0)),
247:      & TCP1,
248:      & int(TCP1/3600.0),int(mod(TCP1,3600.0)/60.0),int(mod(TCP1,60.0))
249: C
250: 7200 format(/5x,' === Elapsed time : ',1p,e12.5,' sec. ',
251:      & '(',i4,' hours ',i2,' min ',i2,' sec ) ==='
252:      & //5x,' === CPU time : ',1p,e12.5,' sec. ',
253:      & '(',i4,' hours ',i2,' min ',i2,' sec ) ===')
254: C
255:      if( TEL1.ne.0.0 ) then
256:          write(IPR,7210) TCP1/TEL1*100.0
257: 7210 format( 5x,' (',f10.4,'%')' )
258:      end if
259: C
260: C/#IF SYSTEM( FACOMAP3K )

```

src/gmvp/center.f

```
261: *      call MVPCOM_EXIT( INFO )
262: C/#ENDIF
263:
264:      return
265:      end
```

SAFE

src/gmvp/check.f

```

1:      SUBROUTINE CHECK( COMENT )
2: C=====
3: C PURPOSE : CHECK WRITE OF ALL COMMON PARAMETERS ( GMVP )
4: C CALLED IN : VARIOUS ROUTINES
5: C=====
6:      include '../shared/INC/_LISTOFF'
7:      include '../shared/INC/_IOUNIT'
8:      include '../shared/INC/_SIZES'
9:      include 'INC/_SIZES2'
10:     include 'INC/_FLAGS'
11:     include '../shared/INC/_PGEOM'
12:     include 'INC/_PMNTR'
13:     include 'INC/_PSOUR'
14:     include '../shared/INC/_PTALY0'
15:     include 'INC/_PTALY'
16:     include '../shared/INC/_PVRED'
17:     include 'INC/_PXSEC'
18:     include 'INC/_STACK'
19:     include 'INC/_XBANK'
20:     include 'INC/_XWORK'
21: C
22:     include 'INC/_WKCOL'
23:     include 'INC/_WKFLA'
24:     include 'INC/_WKFL1'
25:     include 'INC/_WKLAT'
26:     include 'INC/_WKMR'
27: CCCC include 'INC/_WKNXA'
28: CCCC include 'INC/_WKNXC'
29: CCCC include 'INC/_WKNXU'
30:     include 'INC/_WKNXT'
31:     include 'INC/_WKSEA'
32:     include 'INC/_WKSEL'
33:     include 'INC/_WKSOU'
34:     include '../shared/INC/_LISTON'
35: C
36:     CHARACTER*(*) COMENT
37:     WRITE(IPR, '(/1x, '+++++ CHECK WRITE !! +++++' /1x, A/) ) COMENT
38: C
39: C .... SIZE & LIMIT PARAMETERS
40: C=====
41: C== COMMON /SIZES/
42: C=====
43:     WRITE(IPR,*) ' ==== COMMON /SIZES/      ==='
44: CCC   WRITE(IPR,7000)
45:     WRITE(IPR,*)
46:     &'NPART ',NPART, 'NHIST ',NHIST, 'NBANK ',NBANK, 'NGROUP',NGROUP,
47:     &'NZONE ',NZONE, 'NINPZ ',NINPZ, 'NSURF ',NSURF, 'NSDA ',NSDA,
48:     &'NZDA ',NZDA, 'NMEMO ',NMEMO, 'NMAT ',NMAT, 'NREG ',NREG,
49:     &'NSCT ',NSCT, 'NTIME ',NTIME, 'NSOUR ',NSOUR, 'NRESP ',NRESP,
50:     &'TCPU ',TCPU, 'IRAND ',IRAND, 'NTHIST',NTHIST, 'ECUT ',ECUT,
51:     &'TCUT ',TCUT, 'SUPPLY',SUPPLY, 'NFBANK',NFBANK, 'NSKIP ',NSKIP,
52:     &'NBATCH',NBATCH, 'NPICT ',NPICT, 'NREFS ',NREFS, 'NMEMS ',NMEMS,
53:     &'NMEMOP',NMEMOP, 'NLATT ',NLATT, 'NLLEV ',NLLEV, 'NCELL ',NCELL,
54:     &'NEST ',NEST, 'NLBZ ',NLBZ, 'NRSKIP',NRSKIP, 'NGP1 ',NGP1,
55:     &'NGP2 ',NGP2, 'IMPMAX',IMPMAX
56:     7000 FORMAT(1x,5(1x,A,1x,G9.3))
57: CC
58: C .... OPTION FLAGS
59: C=====
60: C== COMMON /FLAGS /
61: C=====
62:     WRITE(IPR,*) ' ==== COMMON /FLAGS /      ==='
63:     WRITE(IPR,7100)
64:     &'JREST ',JREST, 'JNEUT ',JNEUT, 'JPHOT ',JPHOT, 'JIMAG ',JIMAG,
65:     &'JTIME ',JTIME, 'JDELT ',JDELT, 'JFISS ',JFISS, 'JEIGN ',JEIGN,

```

```

66:     &'JFIXD ',JFIXD, 'JADJM ',JADJM, 'JDDX ',JDDX, 'JIMPT ',JIMPT,
67:     &'JWWD ',JWWD, 'JPSTR ',JPSTR, 'JFCOL ',JFCOL, 'JBIAS ',JBIAS,
68:     &'JRESP ',JRESP, 'JMNT ',JMNT, 'JVMNT ',JVMNT,
69:     &'JALLZ ',JALLZ, 'JONEZ ',JONEZ, 'JSIMP ',JSIMP, 'JPICT ',JPICT,
70:     &'JREFL ',JREFL, 'JLATT ',JLATT, 'JHLAT ',JHLAT, 'JFPRT ',JFPRT,
71:     &'JRRIT ',JRRIT, 'JSTOP ',JSTOP
72:     WRITE(IPR,*) 'JPRTS(20)',JPRTS
73:     WRITE(IPR,7100)
74:     &'JSTATX',JSTATX, 'JPTDT ',JPTDT
75:     7100 FORMAT(1x,9(A6,I2,1x))
76:     WRITE(IPR,*) 'JDEBG ',JDEBG
77: C
78: C .... GEOMETRY ARRAY POINTERS .....
79: C=====
80: C== COMMON /PGEOM/
81: C=====
82:     WRITE(IPR,*) ' ==== COMMON /PGEOM/      ==='
83:     WRITE(IPR,7200)
84:     &'LSDA ',LSDA, 'LKMAT ',LKMAT, 'LKREG ',LKREG, 'LKZMAT',LKZMAT,
85:     &'LKZREG',LKZREG, 'LKINPZ',LKINPZ, 'LKZDA ',LKZDA, 'LKZAA ',LKZAA,
86:     &'LKMEMO',LKMEMO, 'LVOL ',LVOL, 'LPAPER',LPAPER, 'LKSREF',LKSREF,
87:     &'LKKREF',LKKREF, 'LMEMC ',LMEMC, 'LMEMZ ',LMEMZ, 'LKCELL',LKCELL,
88:     &'LIDCEL',LIDCEL, 'LICTYP',LICTYP, 'LIPCEL',LIPCEL, 'LCELSZ',LCELSZ,
89:     &'LIDLAT',LIDLAT, 'LLTYP ',LLTYP, 'LNLVAT',LNLVAT, 'LSZLAT',LSZLAT,
90:     &'LIPLAT',LIPLAT, 'LKLATT',LKLATT, 'LKSLAT',LKSLAT, 'LKZLBZ',LKZLBZ,
91:     &'LMLBZZ',LMLBZZ, 'LNKZAA',LNKZAA, 'LDALT ',LDALT, 'LKDALT',LKDALT,
92:     &'LKHLAT',LKHLAT
93:     7200 FORMAT(1x,5(1x,A6,I8) )
94: CC
95:     WRITE(IPR,*) ' == COMMON /PGEOM2/ DEPS ',DEPS, ' DINF ',DINF
96: C .... CPU & VECTOR LENGTH MONITOR .....
97: C
98: C & CPUSOR,CPUSEL,CPUFY,CPUSCH,CPUCOL,
99: C & NEXSOR,NVSOR,VLSOR,VL2SOR,MIVSOR,MAVSOR,
100: C & NEXSEL,NVSEL,VLSEL,VL2SEL,MIVSEL,MAVSEL,
101: C & NEXFLY,NVFLY,VFLY,VL2FLY,MIVFLY,MAVFLY,
102: C & NEXSCH,NVSCH,VLSCH,VL2SCH,MIVSCH,MAVSCH,
103: C & NEXCOL,NVCOL,VLCOL,VL2COL,MIVCOL,MAVCOL
104: C
105: C CPU??? R4 CPU TIME OF SUBROUTINE
106: C NEX??? I4 NUMBER OF EXECUTION OF SUBROUTINE
107: C NV??? I4 TOTAL NUMBER OF VECTOR EVENT
108: C VL??? R4 SUM OF VECTOR LENGTH
109: C VL2??? R4 SQUARE SUM OF VECTOR LENGTH
110: C MI??? I4 MINIMUM VECTOR LENGTH
111: C MA??? I4 MAXIMUM VECTOR LENGTH
112: C
113: C '???' IS THE SYMBOL FOR EACH SUBROUTINE.
114: C SOR = SOURCE, SEL = SELECT, FLY = FLIGHT, SCH = SEARCH, COL = COLISN
115: C
116: C .... SOURCE DATA & ARRAY POINTERS
117: C=====
118: C== COMMON /PSOUR/
119: C=====
120:     WRITE(IPR,*) ' ==== COMMON /PSOUR/      ==='
121:     WRITE(IPR,7200)
122:     &'LKSOUR',LKSOUR, 'LISZON',LISZON, 'LSOUR ',LSOUR, 'LPSPAC',LPSPAC,
123:     &'LPENRG',LPENRG, 'LKENRG',LKENRG, 'LNSTIM',LNSTIM, 'LSTIM ',LSTIM,
124:     &'LPSTIM',LPSTIM, 'LISTIM',LISTIM, 'LNSANG',LNSANG, 'LSANG ',LSANG,
125:     &'LSAXIS',LSAXIS, 'LPSANG',LPSANG, 'LISANG',LISANG, 'LIFISM',LIFISM,
126:     &'LMEMZN',LMEMZN
127: C
128: C .... TALLY ARRAY POINTERS & PARAMETERS .....
129: C=====
130: C== COMMON /PTALY/

```

src/gmvp/check.f

```

131: C=====
132: WRITE(IPR,*) ' ==== COMMON /PTALY/      ==='
133: CCC WRITE(IPR,7000)
134: WRITE(IPR,*)
135: &'LWSUM ' ,LWSUM , 'LWLEK ' ,LWLEK
136: WRITE(IPR,*) 'LNLOST ' ,LNLOST
137: WRITE(IPR,7200) 'LRESP ' ,LRESP ,
138: &'LFLTR ' ,LFLTR , 'LSFLTR' ,LSFLTR , 'LFLCL ' ,LFLCL ,
139: &'LSFLCL' ,LSFLCL , 'LRETR ' ,LRETR , 'LSRETR' ,LSRETR ,
140: &'LRECL ' ,LRECL , 'LSRECL' ,LSRECL ,
141: &'LXSOC ' ,LXSOC , 'LXSXV ' ,LXSXV ,
142: &'LXKEF ' ,LXKEF , 'LNCLSN' ,LNCLSN , 'LWCLSN' ,LWCLSN , 'LNLEAK' ,LNLEAK ,
143: &'LWLEAK' ,LWLEAK , 'LNABSB' ,LNABSB , 'LWABSB' ,LWABSB , 'LNECUT' ,LNECUT ,
144: &'LWECUT' ,LWECUT , 'LNTCUT' ,LNTCUT , 'LWTCUT' ,LWTCUT , 'LNKILD' ,LNKILD ,
145: &'LWKILD' ,LWKILD , 'LNSURV' ,LNSURV , 'LWSURV' ,LWSURV , 'LNSPLT' ,LNSPLT ,
146: &'LWSPLT' ,LWSPLT , 'LENGYB' ,LENGYB , 'LTIMEB' ,LTIMEB , 'LRVOL ' ,LRVOL ,
147: &'LNCNTR' ,LNCNTR , 'LWCNTR' ,LWCNTR ,
148: c WRITE(IPR,7000)
149: c $ 'XAVT(1)' ,XAVT(1) , 'XAVT(2)' ,XAVT(2) , 'XAVT(3)' ,XAVT(3) ,
150: c $ 'AAVT(1)' ,AAVT(1) , 'AAVT(2)' ,AAVT(2) , 'AAVT(3)' ,AAVT(3) ,
151: c $ 'EAVT ' ,EAVT
152: C=====
153: C== COMMON /PTALYD/
154: C=====
155: WRITE(IPR,*) ' ==== COMMON /PTALYD/      ==='
156: WRITE(IPR,7200)
157: &'LFLPD ' ,LFLPD , 'LSFLPD' ,LSFLPD , 'LREPD ' ,LREPD ,
158: &'LSREPD' ,LSREPD
159: C=====
160: C== COMMON /PTDET /
161: C=====
162: WRITE(IPR,*) ' ==== COMMON /PTDET /      ==='
163: WRITE(IPR,7200)
164: &'NPDET ' ,NPDET , 'LXPDET' ,LXPDET , 'LIPDET' ,LIPDET , 'LIPDET2' ,LIPDET2
165: C
166: C .... VARIANCE REDUCTION ARRAY POINTERS
167: C
168: C=====
169: C== COMMON /PVRED/
170: C=====
171: WRITE(IPR,*) ' ==== COMMON /PVRED/      ==='
172: WRITE(IPR,7200)
173: &'LXIMP ' ,LXIMP , 'LWKIL ' ,LWKIL , 'LWSRV ' ,LWSRV
174: WRITE(IPR,*) ' === COMMON /PVRED/ === '
175: C .... CROSS SECTION ARRAY POINTERS
176: C=====
177: C== COMMON /PXSEC/
178: C=====
179: WRITE(IPR,*) ' ==== COMMON /PXSEC/      ==='
180: WRITE(IPR,7200)
181: &'NGPX ' ,NGPX , 'NGGX ' ,NGGX , 'NTGX ' ,NTGX , 'NDSPX ' ,NDSPX ,
182: &'NDSGX ' ,NDSGX , 'NDSTX ' ,NDSTX , 'NUSX ' ,NUSX , 'NSCTX ' ,NSCTX ,
183: &'NCOEFX' ,NCOEFX , 'NMEDX ' ,NMEDX , 'ITAPEX' ,ITAPEX , 'LRSGX ' ,LRSGX ,
184: &'LANGX ' ,LANGX , 'LNGSX ' ,LNGSX , 'LNSGX ' ,LNSGX , 'LDELX ' ,LDELX ,
185: &'LNNNX ' ,LNNNX , 'LNNGX ' ,LNNGX , 'LSTOTX' ,LSTOTX , 'LSSCX ' ,LSSCX ,
186: &'LSNUFX' ,LSNUFX , 'LSNAPX' ,LSNAPX , 'LSGPX ' ,LSGPX , 'LSNFPX' ,LSNFPX ,
187: &'LSNUSX' ,LSNUSX , 'LSGPBX' ,LSGPBX , 'LSSCBX' ,LSSCBX , 'LSDBBX' ,LSDBBX ,
188: &'LSANGX' ,LSANGX , 'LSGPPX' ,LSGPPX , 'LSSCPX' ,LSSCPX , 'LSPORT' ,LSPORT ,
189: &'LWGTG ' ,LWGTG , 'LWGTFI' ,LWGTFI , 'LFKAI ' ,LFKAI , 'LKKAI ' ,LKKAI ,
190: &'LWGTG ' ,LWGTG , 'LWGTGI' ,LWGTGI , 'LS2DPX' ,LS2DPX , 'LS2PPX' ,LS2PPX
191: C .... PARTICLE STACK POINTERS & LENGTHS
192: C=====
193: C== COMMON /STACK/
194: C=====
195: WRITE(IPR,*) ' ==== COMMON /STACK/      ==='

```

```

196: WRITE(IPR,7200)
197: &'LLSFFL' ,LLSFFL , 'LNFFL ' ,LNFFL , 'LIZFFL' ,LIZFFL , 'LLSCOL' ,LLSCOL ,
198: &'NCOLS ' ,NCOLS , 'LLSSRC' ,LLSSRC , 'LNNXT ' ,LNNXT , 'LIZNXT' ,LIZNXT ,
199: &'IDEFER' ,IDEFER , 'LLSDED' ,LLSDED , 'NDEAD ' ,NDEAD , 'NFISB ' ,NFISB ,
200: &'LLSREF' ,LLSREF , 'LIZREF' ,LIZREF , 'LISREF' ,LISREF , 'LNBREF' ,LNBREF ,
201: &'LLSLAT' ,LLSLAT , 'LIZLAT' ,LIZLAT , 'LNXL' ,LNXL , 'NIMPT ' ,NIMPT ,
202: &'NDIMPT' ,NDIMPT , 'LIMSFL' ,LIMSFL , 'LIMNFL' ,LIMNFL , 'LIMZFL' ,LIMZFL ,
203: &'LIMSNX' ,LIMSNX , 'LIMNNX' ,LIMNNX , 'LIMZNX' ,LIMZNX , 'IMDEFR' ,IMDEFR ,
204: &'LIMSLT' ,LIMSLT , 'LIMNLT' ,LIMNLT , 'LIMZLT' ,LIMZLT , 'LIMDED' ,LIMDED
205: C
206: C .... PARTICLE BANK POINTERS
207: C=====
208: C== COMMON /XBANK/
209: C=====
210: WRITE(IPR,*) ' ==== COMMON /XBANK/      ==='
211: WRITE(IPR,7200)
212: &'LXXX ' ,LXXX , 'LYYY ' ,LYYY , 'LZZZ ' ,LZZZ , 'LAAA ' ,LAAA ,
213: &'LBBB ' ,LBBB , 'LCCC ' ,LCCC , 'LWWW ' ,LWWW , 'LIZZ ' ,LIZZ ,
214: &'LIGG ' ,LIGG , 'LTTT ' ,LTTT , 'LXXXF' ,LXXXF , 'LYYYF' ,LYYYF ,
215: &'LZZZF' ,LZZZF , 'LIZZF' ,LIZZF , 'LLEVL' ,LLEVL , 'LLZZ ' ,LLZZ ,
216: &'LLPOS ' ,LLPOS , 'LXIM ' ,LXIM , 'LLEVL' ,LLEVL , 'LLZZF' ,LLZZF ,
217: &'LLPOSF' ,LLPOSF , 'LLCRS ' ,LLCRS , 'LLCRSF' ,LLCRSF , 'LKLSF ' ,LKLSF ,
218: &'LKLSFF' ,LKLSFF
219: C=====
220: C== COMMON /XIBANK/
221: C=====
222: WRITE(IPR,*) ' ==== COMMON /XIBANK/      ==='
223: WRITE(IPR,7200)
224: &'LXXXI ' ,LXXXI , 'LYYYI ' ,LYYYI , 'LZZZI ' ,LZZZI , 'LAAAI ' ,LAAAI ,
225: &'LBBBI ' ,LBBBI , 'LCCCI ' ,LCCCI , 'LWWWI ' ,LWWWI , 'LIZZI ' ,LIZZI ,
226: &'LIGGI ' ,LIGGI , 'LTTTI ' ,LTTTI , 'LLEVLI' ,LLEVLI , 'LLZZI ' ,LLZZI ,
227: &'LLPOSI' ,LLPOSI , 'LLCRSI' ,LLCRSI , 'LKLSFI' ,LKLSFI , 'LKDETP' ,LKDETP ,
228: &'LOPTI ' ,LOPTI , 'LPATH ' ,LPATH
229: C
230: C .... VECTOR WORKING AREA POINTERS .....
231: WRITE(IPR,*) ' ==== COMMON /XWORK/ ===== '
232: WRITE(IPR,7200)
233: &'NWORK ' ,NWORK , 'LWORK ' ,LWORK , 'NWORKB' ,NWORKB , 'LWORKB' ,LWORKB
234: C
235: C=====
236: C== COMMON /WKCOL/
237: C=====
238: WRITE(IPR,*) ' ==== COMMON /WKCOL/      ==='
239: WRITE(IPR,*)
240: &'P3(12)' ,P3
241: C=====
242: C== COMMON /WKFLA/
243: C=====
244: WRITE(IPR,*) ' ==== COMMON /WKFLA/      ==='
245: WRITE(IPR,*)
246: &'P6(30)' ,P6
247: C=====
248: C== COMMON /WKFL1/
249: C=====
250: WRITE(IPR,*) ' ==== COMMON /WKFL1/      ==='
251: WRITE(IPR,*)
252: &'P0(22)' ,P0
253: C=====
254: C== COMMON /WKLAT/
255: C=====
256: WRITE(IPR,*) ' ==== COMMON /WKLAT/      ==='
257: WRITE(IPR,*)
258: &'P5(10)' ,P5
259: C=====
260: C== COMMON /WKMR/

```

src/gmvp/check.f

```
261: C=====
262:      WRITE(IPR,*) ' ==== COMMON /WKMir/      ==='
263:      WRITE(IPR,*)
264:      &'P4(22)',P4
265: C=====
266: C== COMMON /WKNXT/
267: C=====
268:      WRITE(IPR,*) ' ==== COMMON /WKNXT/      ==='
269:      WRITE(IPR,*)
270:      &' PC(30) ',PC
271:      WRITE(IPR,*)
272:      &' PF(50) ',PF
273:      WRITE(IPR,*)
274:      &' PS(20) ',PS,' PL(20) ',PL
275: C=====
276: C== COMMON /WKNXA/
277: C=====
278: C      WRITE(IPR,*) ' ==== COMMON /WKNXA/      ==='
279: C      WRITE(IPR,*)
280: C      &' PFA(30) ',PFA,' PSA(20) ',PSA
281: C=====
282: C== COMMON /WKNXC/
283: C=====
284: C      WRITE(IPR,*) ' ==== COMMON /WKNXC/      ==='
285: C      WRITE(IPR,*)
286: C      &' PC(30) ',PC,' PFC(30) ',PFC,' PSC(20) ',PSC
287: C=====
288: C== COMMON /WKNXU/
289: C=====
290: C      WRITE(IPR,*) ' ==== COMMON /WKNXU/      ==='
291: C      WRITE(IPR,*)
292: C      &' PFU(30) ',PFU,' PSU(20) ',PSU
293: C=====
294: C== COMMON /WKSEA/
295: C=====
296: C      WRITE(IPR,*) ' ==== COMMON /WKSEA/      ==='
297: C      WRITE(IPR,*)
298: C      &' P7(30) ',P7
299: C=====
300: C== COMMON /WKSE1/
301: C=====
302: C      WRITE(IPR,*) ' ==== COMMON /WKSE1/      ==='
303: C      WRITE(IPR,*)
304: C      &' P2(12) ',P2
305: C=====
306: C== COMMON /WKSOU/
307: C=====
308: C      WRITE(IPR,*) ' ==== COMMON /WKSOU/      ==='
309: C      WRITE(IPR,*)
310: C      &' P1(15) ',P1
311: C
312:      RETURN
313:      END
```


src/gmvp/chkenv.f

```
1:      subroutine CHKENV( PROG )
2:
3:      C/#IF .NOT.NOGETENV
4:
5:      C=<MVP>=====
6:      C Purpose: Check enviromnet variables
7:      C=====
8:      C argument :
9:      C
10:     C i prog : code name ( "MVP" or "GMVP" )
11:     C
12:     C=====
13:     C/#IF CUTIL.AND.MS_VISUAL
14:     C
15:     C ... This interface block is for CUTIL & MS-Visual tools. ...
16:     C
17:     * interface
18:     *      subroutine FUNBUF()
19:     *CDEC$      ATTRIBUTES C :: FUNBUF
20:     *      end subroutine
21:     *      end interface
22:     C/#ENDIF
23:     C
24:     include '../shared/INC/_IOUNIT'
25:     C/#IF PARA(VPP)
26:     include '../shared/INC/_VPPPARA'
27:     include '../shared/INC/_TASKDT'
28:     C/#ENDIF
29:     C
30:     C ... external data ...
31:     C
32:     character*(*) PROG
33:     C
34:     C ... local data ...
35:     C
36:     character*64 VAR
37:     character*128 STRING
38:     C
39:     C-----
40:     C
41:     C ==== check MVPUNBUF : set standard output unbufferd
42:     C ==== check MVP_UNBUF : set standard output unbufferd
43:     C      ("line buffered" saying exactly )
44:     C      Other than null or blank should be defined for the variable
45:     C      to activate it.
46:     C
47:     C      STRING = ' '
48:     C
49:     C      call ENVGET( 'MVPUNBUF', STRING )
50:     C      if( STRING.eq.' ' ) then
51:     C          call ENVGET( 'MVP_UNBUF', STRING )
52:     C      end if
53:     C/#IF CUTIL
54:     C      if( STRING.ne.' ' ) then
55:     C          call FUNBUF()
56:     C          write(IPR,*) '==(CHKENV) Standard output is line buffered.'
57:     C      end if
58:     C/#ENDIF
59:     C
60:     C/#IF PARA(VPP)
61:     C
62:     C ==== check MVPVPPRT : set standard output unbufferd
63:     C ==== check MVP_VPP_PRT : set standard output unbufferd
64:     C
65:     C      VPPPRT//<I/O unit #> is a file name of printout
```

```
66:     C      from each task.
67:     C
68:     C      VPPPRT = ' '
69:     C      STRING = ' '
70:     C      call ENVGET( 'MVPVPPRT', STRING )
71:     C      if( STRING.eq.' ' ) then
72:     C          call ENVGET( 'MVP_VPP_PRT', STRING )
73:     C      end if
74:     C      if( STRING.ne.' ' ) then
75:     C          VPPPRT = STRING
76:     C          write(IPR,*) '==(CHKENV) VPP printout file name base <',
77:     C      &          VPPPRT(:iclen2(VPPPRT)), '>'
78:     C      end if
79:     C/#ENDIF
80:     C
81:     C/#ENDIF
82:     C
83:     C      return
84:     C      end
```

src/gmvp/chknum.f

```

1:      subroutine CHKNUM( JNEUT, JPHOT, JFISS, JADJM, NGROUP,NGP1,  NGP2,
2:      &
3:      &
4: C=====
5: C  PURPOSE: CHECK THE INTEGER PARAMETERS
6: C=====
7:      integer KMAT(1)
8: C
9:      include '../shared/INC/_IUNIT'
10: C
11: C -----
12: C
13:      ICK      = 0
14: C
15:      if ( NGROUP.le.0 ) then
16:      write(IMG,'(1x,a,i5,a)') 'XXX NGROUP <= 0 (= ', NGROUP,
17:      &
18:      call CNTERR( 'FATAL' )
19:      end if
20:
21:      if ( JNEUT*JPHOT.ne.0 ) then
22:
23:      if ( (NGP1+NGP2).eq.0 ) then
24:
25:      write(IMG,'(1x,a,a,i4,a,i4,a)')
26:      &
27:      &
28:      call CNTERR( 'FATAL' )
29:
30:      else if ( NGP2.eq.0 ) then
31:
32:      NGP2      = NGROUP - NGP1
33:      write(IMG,'(1x,a,a,i4,a)')
34:      &
35:      &
36:      call CNTERR( 'WARNING' )
37:      if ( NGP2.le.0 ) then
38:      write(IMG,*) 'XXX NGP2 <= 0, STOP !!!'
39:      call CNTERR( 'FATAL' )
40:      end if
41:
42:      else if ( NGP1.eq.0 ) then
43:
44:      NGP1      = NGROUP - NGP2
45:      write(IMG,'(1x,a,a,i4,a)')
46:      &
47:      &
48:      call CNTERR( 'WARNING' )
49:      if ( NGP1.le.0 ) then
50:      write(IMG,*) 'XXX NGP1 <= 0, STOP !!!'
51:      call CNTERR( 'FATAL' )
52:      end if
53:
54:      end if
55:
56:      if ( (NGP1+NGP2).ne.NGROUP ) then
57:
58:      write(IMG,'(1x,a,i4,a,i4,a)') 'XXX NGP1+NGP2 (= ', NGP1
59:      &
60:      &
61:      call CNTERR( 'FATAL' )
62:      ICK      = ICK + 1
63:
64:      end if
65: C

```

```

66:      if ( JADJM.ne.0 ) then
67:      NT      = NGP1
68:      NGP1     = NGP2
69:      NGP2     = NT
70:      end if
71: C
72:      else if ( JNEUT.ne.0 ) then
73:
74:      if ( NGP1.ne.NGROUP ) then
75:
76:      write(IMG,'(1x,a,i4,a,i4,a)') '!!! NGP1 (= ', NGP1,
77:      &
78:      &
79:      call CNTERR( 'WARNING' )
80:      NGP1     = NGROUP
81:
82:      end if
83:      NGP2     = 0
84: C
85: CC ... This process is moved to XSGMVP. (2004/06/22)
86: CC      else if ( JPHOT.ne.0 ) then
87: CC
88: CC      if ( NGP2.ne.NGROUP ) then
89: CC
90: CC      write(IMG,'(1x,a,i4,a,i4,a)') '!!! NGP2 (= ', NGP2,
91: CC      &
92: CC      &
93: CC      call CNTERR( 'WARNING' )
94: CC      NGP2     = NGROUP
95: CC
96: CC      end if
97: CC
98: CC      NGP1     = NGP2
99: CC      NGP2     = 0
100: CC      write(IMG,'(a,i3,a,i3,a)') '!!! NGP1 (= ', NGP1,
101: CC      &
102: CC      &
103: CC      call CNTERR( 'WARNING' )
104: CC
105: CC      end if
106: C
107:      if ( NGP1.gt.NGPX ) then
108:
109:      write(IMG,*) 'XXX PRIMARY GROUP NUMBER TO BE ANALYZED(', NGP1,
110:      &
111:      &
112:      call CNTERR( 'FATAL' )
113:
114:      end if
115: C
116:      if ( NGP2.gt.NGGX ) then
117:
118:      write(IMG,*) 'XXX PRIMARY GROUP NUMBER TO BE ANALYZED(', NGP2,
119:      &
120:      &
121:      call CNTERR( 'FATAL' )
122:
123:      end if
124: C
125:      if ( (NGPX+NGGX).ne.NTGX ) then
126:
127:      write(IMG,*) 'XXX THERE IS DISCREPANCY BETWEEN GROUP',
128:      &
129:      call CNTERR( 'FATAL' )
130:

```

src/gmvp/chknum.f

```
131:         end if
132: C
133:         NT      = 0
134:         do 100 I = 1, NINPZ
135:           if ( KMAT(I).gt.NT ) NT = KMAT(I)
136: 100 continue
137: C
138:         if ( (NMAT.eq.0.or.NMAT.lt.NT).and.JFISS.ne.0 ) then
139:
140:           write(IMG,*) 'XXX FISSION SPECTRUM IS NOT GIVEN BECAUSE ',
141: &           'NMAT=0 OR LESS THAN MATERIAL USED (NMAT=', NMAT,
142: &           ',USED MAT=', NT, ' '), STOP !!!'
143:           call CNTERR( 'FATAL' )
144:
145:         end if
146: C
147:         if ( NT.gt.NMEDX ) then
148:
149:           write(IMG,*) 'XXX NUMBER OF PREPARED CROSS SECTION (=' NMEDX,
150: &           ') IS', ' LESS THAN MATERIAL USED (=' NT,
151: &           '), STOP !!!'
152:           call CNTERR( 'FATAL' )
153:
154:         end if
155: C
156: C      IF( ICK.NE.0 ) THEN
157: C          STOP 888
158: C      ENDIF
159: C
160:         return
161:     end
```

src/gmvp/chsymb.f

```
1:      character*2 function CHSYMB(NA)
2: C=<GMVP>=====
3: C  Purpose : Dummy routine for mvp/chsymb. GMVP doesn't require this
4: C            routine for actual calculations but for compilation.
5: C            This routine is required since the photonuclear capability
6: C            was added.
7: C  Called in: STLPR?
8: C=====
9:      write(6,*) 'XXX(CHSYMB) GMVP does not use this routine.'
10:     stop
11:     end
```

DATA

src/gmvp/colisn.f

```

1: C/#IF ARGSAVE
2: * SUBROUTINE COLISN( IOW,
3: * 4 NGROUP, NGP1, NGP2, NZONE, NMAT, NREG,
4: * 5 NTIME, NBANK, NFBANK, NFBANK0, NEST,
5: * 6 KZMAT, KZREG, XIMP, WKIL, WSRV, WTIME, WLLIM, MXPGEN,
6: * 7 WGTG, WGTFI, FKAI, KKAI, WGTG, WGTGI,
7: * 8 NGPX, NGGX, NTGX, NDSGX, NDSGX, NDSTX,
8: * 9 NUSX, NSCTX, NCOEFX, NMEDX,
9: * A ANGX, NGSX, NNNX,
10: * 1 STOTX, SSCX, SNAPX, SGPX, SNFPX,
11: * 2 SNUSX, SGPBX, IGPBX, SSCBX, ISCBX,
12: * 3 SDBX, IDDBX, S2DBX, S2PPX,
13: * 4 SANGX, IANGX, SGPPX, SSCPX, SPORT,
14: * 5 IRAND,
15: * 6 LEVL, LZZ, LPOS, LCRS, KLSF, IBREG, IBSPC,
16: * 7 LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF,
17: * T NSTAL, NRESP, RESP, NMKREG, MKREG,
18: * T JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY )
19: C/#ELSE
20: subroutine COLISN( IOW, NGROUP, NGP1, NGP2, NZONE, NMAT, NREG, NTIME,
21: & NBANK, NFBANK, NFBANK0, NEST, KZMAT, KZREG,
22: & XIMP, WKIL, WSRV, WTIME, WLLIM, MXPGEN, WGTG, WGTFI,
23: & FKAI, KKAI, WGTG,
24: & WGTGI, NGPX, NGGX, NTGX, NDSGX, NDSGX,
25: & NDSTX, NUSX, NSCTX, NCOEFX, NMEDX, ANGX, NGSX,
26: & NNNX, STOTX, SSCX, SNAPX, SGPX, SNFPX,
27: & SNUSX, SGPBX, IGPBX, SSCBX, ISCBX, SDBX,
28: & IDDBX, S2DBX, S2PPX, SANGX, IANGX, SGPPX,
29: & SSCPX, SPORT, IRAND, LEVL, LZZ, LPOS, LCRS,
30: & KLSF, IBREG, IBSPC, LEVLF, LZZF, LPOSF,
31: & LCRSF, IBRGF, IBSPF,
32: T NSTAL, NRESP, RESP, NMKREG, MKREG,
33: T JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, DTALY,
34: & LSFFL, NFFL, IZFFL,
35: & LSCOL, NCOLS, LSDDED, NDEAD, XXX, YYY, ZZZ,
36: & AAA, BBB, CCC, WWW, IZZ, IGG, ITT, ITT,
37: & XIM, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
38: & DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK,
39: & XXXF, YYYF, ZZZF, IZZF,
40: & NFISB, FLCL,
41: & NCLSN, WCLSN, NABSB, WABSB, NECUT, NECUT,
42: & NKILD, WKILD, NSURV, WSRV, NSPLT, NSPLT,
43: & NCNTR, WCNTR, IWK1, IWK2, IWK3, IWK4, IWK5,
44: & IWK6, IWK7, IWK8, IWK9, RP, R, IWRK,
45: & DWK1, DWK2
46: & )
47: C/#ENDIF
48: C=====
49: C PURPOSE: HANDLING COLLISION & REACTION
50: C CALLED IN: ACTION
51: C CALLS: VMNTR1, RANU2, MCOLS
52: C=====
53: C
54: C === INTER-STACK DATA FLOW ===
55: C
56: C COLLISION STACK -----+-----> FREE-FLIGHT STACK
57: C (LSCOL, NCOLS) | | (LSFFL, IZFFL, NFFL)
58: C DEAD PARTICLE STACK ---+-----> DEAD PARTICLE STACK
59: C (LSDDED, NDEAD) (LSDDED, NDEAD)
60: C
61: C IF THERE ARE ANY FISSION-PARTICLES IN NON-EIGENVALUE PROBLEMS,
62: C THE FOLLOWING DATA FLOW OCCURS.
63: C
64: C DEAD PARTICLE STACK -----> COLISION STACK
65: C

```

```

66: C === BANK DATA TO BE UPDATED ===
67: C
68: C (XXX,YYY,ZZZ), (AAA,BBB,CCC) : POSITION, DIRECTION
69: C IGG : ENERGY GROUP. WWW : WEIGHT
70: C
71: C === BANK DATA ADDED NEWLY (SPLITTING, FISSION) ===
72: C
73: C (XXX,YYY,ZZZ), (AAA,BBB,CCC), IZZ : POSITION, DIRECTION & ZONE #
74: C IGG, WWW : ENERGY GROUP & WEIGHT
75: C LEVL, LZZ, LPOS, LCRS : LATTICE-PARAMETER
76: C
77: C === FISSION BANK (TO BE CREATED IN FISGEN) ===
78: C
79: C (XXXF,YYYF,ZZZF), IZZF : POSITION & ZONE #
80: C LEVLF, LZZF, LPOSF, LCRSF : LATTICE-PARAMETER
81: C
82: C=====
83: C implicit real*8(A-H,O-Z)
84: C
85: C include 'INC/_FLAGS'
86: C
87: C**** GEOMETRY ARRAY DATA
88: C
89: C integer KZMAT(NZONE,2), KZREG(NZONE)
90: C
91: C**** VARIANCE REDUCTION DATA
92: C
93: C real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
94: C real WTIME(NTIME)
95: C real WLLIM
96: C
97: C**** CROSS SECTION DATA
98: C
99: C real ANGX(*), STOTX(*), SSCX(*), SNAPX(*), SGPX(*), SNFPX(*),
100: C & SNUSX(*), SGPBX(*), SSCBX(*), SDBX(*), SGPPX(*),
101: C & SSCPX(*), SPORT(*), SANGX(NSCTX,3,NTGX,NMEDX), WGTG(*),
102: C & WGTFI(*), FKAI(NTGX,*), WGTG(*), WGTGI(*)
103: C integer NGSX(*), NNNX(*), IGPBX(*), ISCBX(*), IDDBX(*),
104: C & KKAI(2,NTGX,*), IANGX(NSCTX,3,NTGX,NMEDX)
105: C
106: C .... ISCBX & IDDBX HAVE THE SAME ADDRESSES AS SSCBX & SDBX .....
107: C IANGX HAVE THE SAME ADDRESSES AS SANGX .....
108: C
109: C**** STACK
110: C
111: C integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE), LSCOL(NBANK),
112: C & LSDDED(NBANK)
113: C
114: C**** PARTICLE BANK
115: C
116: C real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
117: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
118: C integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
119: C & LPOS(NBANK,NEST), LCRS(NBANK,NEST)
120: C integer KLSF(NBANK), IBREG(NBANK), IBSPC(NBANK,0:NEST)
121: C real*8 ITT(NBANK)
122: C integer ITT(NBANK)
123: C real WWW(NBANK), XIM(NBANK)
124: C
125: C CCC real SGTL(NBANK,NSTAL)
126: C
127: C ... optional bank parameters
128: C
129: C real*8 DBNK(NBANK,*)
130: C integer KDBNK(0:MDBNK)

```

src/gmvp/colisn.f

```

131:      integer IBNK(NBANK,*)
132:      integer KIBNK(0:MIBNK)
133: C
134: C**** Fission neutron bank
135: C
136:      real*8 XXXF(NFBNK0), YYYY(NFBNK0), ZZZF(NFBNK0)
137:      integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
138:      &      LCRSF(NFBNK0,NEST)
139:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
140: C
141:      real*8 DFBK(NFBNK0,*)
142:      integer KDFBK(0:MDFBK)
143:      integer IFBK(NFBNK0,*)
144:      integer KIFBK(0:MIFBK)
145: C
146: C**** TALLY ARRAY
147: C
148:      real*8 FLCL(NGROUP,NREG)
149:      real*8 WCNTR(*), NCNTR(*)
150:      real*8 WCLSN(NGROUP,NREG), WABSB(NGROUP,NREG), WECUT(NREG),
151:      &      WKILD(NGROUP,NREG), WSURV(NGROUP,NREG),
152:      &      WSPLT(NGROUP,NREG), NCLSN(NGROUP,NREG),
153:      &      NABSB(NGROUP,NREG), NECUT(NREG), NKILD(NGROUP,NREG),
154:      &      NSURV(NGROUP,NREG), NSPLT(NGROUP,NREG)
155: C
156:      real RESP(NGROUP,NREG)
157:      integer MKREG(NREG,2)
158: C
159:      integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
160:      integer IDTAL(*)
161:      real*8 DTALY(*)
162: C
163: C
164: C**** WORKING AREA
165: C
166:      integer IWK1(NBANK), IWK2(NBANK), IWK3(NBANK), IWK4(NBANK),
167:      &      IWK5(NBANK), IWK6(NBANK), IWK7(NBANK), IWK8(NBANK),
168:      &      IWK9(NBANK), IWRK(NBANK)
169:      real RP(NBANK), R(4*NBANK)
170:      real*8 DWK1(NBANK), DWK2(NBANK)
171: C
172: C 0..NBK.2NBK.3NBK.4NBK.5NBK.6NBK.7NBK.8NBK.9NBK,10NBK,11NBK
173: C IWK1 IWK2 IWK3 IWK4 IWK5 IWK6 IWK7 IWK8 IWK9 RP R,IWRK
174: C****
175: C
176:      real*8 PI2
177:      parameter( PI2 = 2*3.14159265358979D0 )
178: C
179:      NSCT3 = NSCTX*3
180:      NSCT3N = NSCT3*NDSTX
181:      NDST3 = NDSTX*3
182:
183: C/#IF ARGSAVE
184: *      return
185: C
186: *      entry COLISN ( LSFFL, NFFL , IZFFL,
187: *      2          LSCOL, NCOLS, LSDED, NDEAD,
188: *      3          XXX , YYY , ZZZ , AAA , BBB , CCC ,
189: *      4          WWW , IZZ , IGG , TTT , ITT , XIM ,
190: *      5          DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
191: *      6          DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
192: *      7          XXXF , YYYY , ZZZF , IZZF ,
193: *      8          NFISB, FLCL , NCLSN, WCLSN,
194: *      9          NABSB, WABSB, NECUT, WECUT, NKILD, WKILD,
195: *      10         NSURV, WSURV, NSPLT, WSPLT, NCNTR, WCNTR,

```

```

196: *      8          IWK1 , IWK2 , IWK3 , IWK4, IWK5, IWK6 ,
197: *      9          IWK7 , IWK8 , IWK9 , RP , R , IWRK,
198: *      10         DWK1, DWK2 )
199: C/#ENDIF
200:
201:      if ( JVMNT.ne.0 ) call VMNTR1( 5, NCOLS )
202: C
203:      WMIN = WLLIM
204: C
205: C**** GATHER IREG AND IMAT----> IWK1 AND IWK2
206: C
207:      do 100 I = 1, NCOLS
208:      IZ = IZZ(LSCOL(I))
209:      IWK1(I) = KZREG(IZ)
210:      if ( JTLT.ne.0 ) IWK1(I) = IBREG(LSCOL(I))
211:      IWK2(I) = KZMAT(IZ,1)
212:      IWK3(I) = IGG(LSCOL(I))
213: 100 continue
214: C
215: C ... material # in current zone may be negative in STGR base
216: C
217:      if ( JSTGR.ne.0 ) then
218:      do 110 I = 1, NCOLS
219:      if ( IWK2(I).lt.0 ) then
220:      IWK2(I) = KZMAT(IZZ(LSCOL(I)),2)
221:      end if
222: 110 continue
223:      end if
224: C
225: C**** TAKE TALLY
226: C
227:      do 120 I = 1, NCOLS
228:      IP = LSCOL(I)
229:      WCNTR(4) = WCNTR(4) + WWW(IP)
230:      RP(I) = WWW(IP) /STOTX((IWK2(I)-1)*NTGX+IWK3(I))
231: 120 continue
232:      NCNTR(4) = NCNTR(4) + NCOLS
233: C
234:      if ( JMNTR.ne.0 ) then
235:      do 130 I = 1, NCOLS
236:      NCLSN(IWK3(I),IWK1(I)) = NCLSN(IWK3(I),IWK1(I)) + 1
237:      WCLSN(IWK3(I),IWK1(I)) = WCLSN(IWK3(I),IWK1(I))
238:      &      + WWW(LSCOL(I))
239: 130 continue
240:      end if
241: C
242: C ..... FLUX .....
243: C
244: C
245:      do 140 I = 1, NCOLS
246:      FLCL(IWK3(I),IWK1(I)) = FLCL(IWK3(I),IWK1(I)) + RP(I)
247: 140 continue
248: C
249: C**** Take "spectial" tally by neutron collision estimator
250: C
251:      if ( NTEVE.gt.0.and.(JTEVE(10).gt.0 .or. JTEVE(11).gt.0 ) ) then
252: C
253:      do 150 I = 1, NCOLS
254:      DWK1(I) = RP(I)
255:      if ( JTIME.ne.0 ) then
256:      IWK7(I) = ITT(LSCOL(I))
257:      DWK2(I) = TTT(LSCOL(I))
258:      end if
259: 150 continue
260: C

```

src/gmvp/colisn.f

```

261: C      === neutron collision ===
262: C
263: C      do 180 J = 0, JTEVE(10) - 1
264: C
265: C      ... pointer to direct tally (idt) & d-tally # (it) ...
266: C      IDT      = KTEVE(LTEVE(10)+J) - 1
267: C      IT       = IDTAL(IDT+9)
268: C
269: C      ... skip if current region does not need this tally ...
270: C
271: C      do 160 I = 1, NCOLS
272: C      if ( JDTRG(IWK2(I),IT).ne.0 ) go to 170
273: C 160      continue
274: C      go to 180
275: C
276: C 170      continue
277: C
278: C      JPTIM2 = 0
279: C      MZONE0 = 0
280: C
281: C      << comment on Feb 23 2000 >>
282: C      ... TCUT00 is used for TCUT not passed to this routine,
283: C      and it should not be used in this call.
284: C      ... TIMEB0 is also a spacer for real TIME(*)
285: C      ... Currently treatment for JPTIM > 0 in STALN1 may be
286: C      incorrect for collision estimator
287: C
288: C      ... SGATL is not supported in GMVP currently.
289: C
290: C      call STALN1( IOW, JTIME, JPTIM, MZONE0, IDTAL(IDT+1), DWK1,
291: C      &          NCOLS, IWK3, IWK1, LSCOL, IWK7, DWK2, TDUMY0,
292: C      &          NGROUP, NGP1, NREG, NRESP, NBANK, NSTAL, NZONE,
293: C      &          NTIME, RESP, TIMEB0, NMKREG, MKREG, DTALY, SGATL,
294: C      &          TCUT00, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, R,
295: C      &          IWK8, IWK9, JPTIM2 )
296: C 180      continue
297: C
298: C      ==== photon collision ====
299: C
300: C      do 210 J = 0, JTEVE(11) - 1
301: C
302: C      ... pointer to direct tally (idt) & d-tally # (it) ...
303: C      IDT      = KTEVE(LTEVE(11)+J) - 1
304: C      IT       = IDTAL(IDT+9)
305: C
306: C      ... skip if current region does not need this tally ...
307: C
308: C      do 190 I = 1, NCOLS
309: C      if ( JDTRG(IWK2(I),IT).ne.0 ) go to 200
310: C 190      continue
311: C      go to 210
312: C
313: C 200      continue
314: C
315: C      JPTIM2 = 0
316: C      MZONE0 = 0
317: C
318: C      call STALN1( IOW, JTIME, JPTIM, MZONE0, IDTAL(IDT+1), DWK1,
319: C      &          NCOLS, IWK3, IWK1, LSCOL, IWK7, DWK2, TDUMY0,
320: C      &          NGROUP, NGP1, NREG, NRESP, NBANK, NSTAL, NZONE,
321: C      &          NTIME, RESP, TIMEB0, NMKREG, MKREG, DTALY, SGATL,
322: C      &          TCUT00, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, R,
323: C      &          IWK8, IWK9, JPTIM2 )
324: C 210      continue
325: C

```

```

326: C      end if
327: C
328: C
329: C      N2      = 2*NCOLS
330: C      N3      = 3*NCOLS
331: C
332: C
333: C**** FISSON ANALYSIS AND FISSON NEUTRON GENERATION
334: C
335: C
336: C      if ( JFISS.ne.0 ) then
337: C
338: C      (IWK1,IWK2 & IWK3 are IREG, IMAT & IGRP in FISGEN )
339: C
340: C      call FISGEN( NGROUP, NZONE, NMAT, NREG, NTIME, NBANK, NFBANK,
341: C      &          NFBNK0, IWK1, IWK2, IWK3, WGTG, WGTGI, WTIME, WLLIM,
342: C      &          MXPGEN, FKAI, KKAI, NGPX, NGGX, NTGX, NMEDX, SNFPX,
343: C      &          SNAPX, S2DPX, S2PPX, LSCOL, NCOLS, LSDED, NDEAD, LSFFL,
344: C      &          IZFFL, NFFL, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IZZ,
345: C      &          IGG, TTT, ITT, XIM, KLSF, DBNK, KDBNK, MDBNK, IBNK,
346: C      &          KIBNK, MIBNK, XXXF, YYF, ZZZF, IZZF, DFBK, KDFBK,
347: C      &          MDFBK, IFBK, KIFBK, MIFBK, NFISB, NCNTR, WCNTR, IWK4,
348: C      &          IWK5, IWK6, IWK7, IWK8, IWK9, RP, R, IWRK, IRAND, NEST,
349: C      &          LEVL, LZZ, LPOS, LCRS, LEVLF, LZZF, LPOSF, LCRSF,
350: C      &          IBREG, IBSPC, IBRGF, IBSPF )
351: C      end if
352: C
353: C**** SECONDARY PHOTON GENERATION
354: C
355: C      if ( JNEUT*JPHOT.ne.0 ) then
356: C      call GAMGEN( NGROUP, NZONE, NMAT, NREG, NTIME, NBANK, NGP1,
357: C      &          NGP2, IWK1, IWK2, IWK3, WGTG, WGTGI, WTIME, WLLIM,
358: C      &          MXPGEN, NGPX, NGGX, NTGX, NMEDX, SGPX, SGPBX, IGPBX,
359: C      &          S2DPX, S2PPX, LSCOL, NCOLS, LSDED, NDEAD, LSFFL, IZFFL,
360: C      &          NFFL, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IZZ, IGG, TTT,
361: C      &          ITT, XIM, KLSF, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
362: C      &          NGAM, NCNTR, WCNTR, IWK4, IWK5, IWK6, IWK7, IWK8, IWK9,
363: C      &          RP, R, IWRK, IRAND, NEST, LEVL, LZZ, LPOS, LCRS, IBREG,
364: C      &          IBSPC )
365: C      end if
366: C
367: C**** DETERMINATION OF OUTGOING PARTICLE PARAMETERS ( IGG,AAA,BBB,CCC)
368: C
369: C      if ( JDDX.ne.0 ) then
370: C
371: C      DETERMINATION OF OUTGOING PARTICLE PARAMETER( ENERGY GROUP AND
372: C      DIRECTION BY USING DDX
373: C      IWK2: MEDIA NUMBER, IWK3: INCIDENT ENERGY GROUP
374: C      IWK4: OUTGOING ENERGY GROUP
375: C      IWK5: OUTGOING DIRECTION
376: C
377: C      if ( JSTATX.eq.0 ) then
378: C      call RANU2( IRAND, R, NCOLS*2, ICON )
379: C      *VOCL LOOP,NOVREC
380: C      do 220 I = 1, NCOLS
381: C      IP      = LSCOL(I)
382: C
383: C      ..... L0: HEAD POSITION OF MATRIX
384: C      L0      = (IWK2(I)-1)*NSCT3N + NGSX(IWK3(I))*NSCT3
385: C
386: C      ..... DISCRETE SAMPLING OF OUTGOING ANGLE AND ENERGY GROUP
387: C
388: C      N1      = NNNX(IWK3(I))*NSCTX
389: C      C/#IF ROUNDOFF(NEAREST)
390: C      L1      = MIN(INT(N1*R(I))+1,N1)

```

src/gmvp/colisn.f

```

391: C/#ELSE
392: *          L1      = N1*R(I) + 1
393: C/#ENDIF
394:          RR      = R(I+NCOLS) - SDBBX(L1+L0)
395: C/#IF ROUNDOFF(NEAREST)
396:          if ( SDBBX(L1+L0).ge.1. ) RR      = -0.5
397:          L2      = 2*L1 + N1 + L0
398:          L3      = MIN(INT(L2+RR),L2)
399: ccccccccc L3      = INT(2*L1+RR) + N1 + L0
400: C/#ELSE
401: *          L2      = 2*L1 + N1 + L0
402: *          L3      = L2 + RR
403: C/#ENDIF
404:          IPP      = IDDBX(L3)
405:          IGS      = (IPP-1)/NSCTX
406:          IWK4(I) = IGS + IWK3(I) - NUSX
407:          IWK5(I) = IPP - NSCTX*IGS
408: C          WEIGHT REDUCTION
409:          WWW(IP) = WWW(IP)*SNAPX((IWK2(I)-1)*NTGX+IWK3(I))
410: 220      continue
411: c
412: C ..... JSTATX > 0
413: c
414:          else
415:          call RANU2( IRAND, R, NCOLS*4, ICON )
416: *VOCL LOOP,NOVREC
417:          do 230 I = 1, NCOLS
418:             IP      = LSCOL(I)
419:
420: C ..... DISCRETE SAMPLING OF DIRECTION
421:
422: C/#IF ROUNDOFF(NEAREST)
423:          L4      = MIN(INT(NSCTX*R(I))+1,NSCTX)
424: C/#ELSE
425: *          L4      = NSCTX*R(I)+1
426: C/#ENDIF
427:
428:          RR      = R(I+NCOLS) - SANGX(L4,1,IWK3(I),IWK2(I))
429:
430: C/#IF ROUNDOFF(NEAREST)
431:          L5      = 2*L4 + NSCTX
432:          L6      = MIN(INT(L5+RR),L5)
433: ccccccccc L6      = INT(2*L4 + RR) + NSCTX
434: C/#ELSE
435: *          L5      = 2*L4 + NSCTX
436: *          L6      = L5 + RR
437: C/#ENDIF
438:
439:          IWK5(I) = IANGX(L6,1,IWK3(I),IWK2(I))
440:
441: C ..... DISCRETE SAMPLING OF OUTGOING ENERGY GROUP
442:
443:          NNNX1   = NNNX(IWK3(I))
444:          L0      = (IWK2(I)-1)*NSCT3N + NGSX(IWK3(I))*NSCT3
445:          &       + (IWK5(I)-1)*NNNX1*3
446:
447: C/#IF ROUNDOFF(NEAREST)
448:          L1      = MIN(INT(NNNX1*R(I+N2))+1,NNNX1)
449: C/#ELSE
450: *          L1      = NNNX1*R(I+N2) + 1
451: C/#ENDIF
452:
453:          RR      = R(I+N3) - SDBBX(L1+L0)
454:
455: C/#IF ROUNDOFF(NEAREST)

```

```

456:          if ( SDBBX(L1+L0).ge.1. ) RR      = -0.5
457:          L2      = 2*L1 + NNNX1 + L0
458:          L3      = MIN(INT(L2+RR),L2)
459: ccccccc L3      = INT(2*L1+RR) + NNNX1 + L0
460: C/#ELSE
461: *          L2      = 2*L1 + NNNX1 + L0
462: *          L3      = L2 + RR
463: C/#ENDIF
464:
465:          IWK4(I) = IDDBX(L3) + IWK3(I) - NUSX - 1
466: C
467: C          WEIGHT REDUCTION
468: C
469:          WWW(IP) = WWW(IP)*SNAPX((IWK2(I)-1)*NTGX+IWK3(I))
470: 230      continue
471:          end if
472: C
473: C .... ENERGY CUT OFF .....
474: C
475:          if ( NTGX.gt.NGROUP ) then
476: C
477: C .... FOR DETAILED MONITORING ONLY ....
478: C
479:          if ( JMNTR.ne.0 ) then
480: *VOCL LOOP,SCALAR
481:          do 240 I = 1, NCOLS
482:             if ( IWK4(I).gt.NGROUP ) then
483:                NECUT(IWK1(I)) = NECUT(IWK1(I)) + 1
484:                WECUT(IWK1(I)) = WECUT(IWK1(I)) + WWW(LSCOL(I))
485:             end if
486: 240      continue
487:          end if
488: c
489:          NN      = 0
490: *VOCL LOOP,NOVREC
491:          do 250 I = 1, NCOLS
492:             IP      = LSCOL(I)
493:             if ( IWK4(I).gt.NGROUP ) then
494:                NDEAD = NDEAD + 1
495:                LSDDED(NDEAD) = IP
496:                NCNTR(8) = NCNTR(8) + 1
497:                WCNTR(8) = WCNTR(8) + WWW(IP)
498:             else
499:                NN      = NN + 1
500:                LSCOL(NN) = IP
501:                IWK1(NN) = IWK1(I)
502:                IWK3(NN) = IWK3(I)
503:                IWK4(NN) = IWK4(I)
504:                IWK5(NN) = IWK5(I)
505:             end if
506: 250      continue
507: C
508:          NCOLS   = NN
509:          end if
510: C
511:          call RANU2( IRAND, R, NCOLS*2, ICON )
512: C
513: C ..... SCATTERING ANGLE .....
514: C
515: *VOCL LOOP,NOVREC
516:          do 260 I = 1, NCOLS
517:             IP      = LSCOL(I)
518:             FM      = ANGX(IWK5(I))*R(I) + ANGX(IWK5(I)+1)*(1.-R(I))
519: C
520:          AAAAA0 = AAA(IP)

```


src/gmvp/colisn.f

```

521:      BBBB0 = BBB(IP)
522:      CCCCC0 = CCC(IP)
523:      SINPSI = SQRT(1.-FM*FM)
524:      ETA = PI2*R(NCOLS+I)
525:      SINETA = SIN(ETA)
526:      COSETA = COS(ETA)
527:      STHETA = 1. - AAAAA0*AAAAA0
528: *VOCL STMT,IF(100)
529:      if ( STHETA.gt.0.0 ) then
530:          STHETA = SQRT(STHETA)
531:          COSPHI = BBBB0/STHETA
532:          SINPHI = CCCCC0/STHETA
533:      else
534:          STHETA = 0.0
535:          COSPHI = 1.0
536:          SINPHI = 0.0
537:      end if
538: C
539:      BBBB0 = BBBB0*FM + (AAAAA0*COSPHI*COSETA
540: &          -SINPHI*SINETA)*SINPSI
541:      CCCCC0 = CCCCC0*FM + (AAAAA0*SINPHI*COSETA
542: &          +COSPHI*SINETA)*SINPSI
543:      AAAAA0 = AAAAA0*FM - COSETA*SINPSI*STHETA
544:      S = 1./SQRT(AAAAA0*AAAAA0+BBBBB0*BBBBB0+CCCCC0*CCCCC0)
545: C
546:      AAA(IP) = AAAAA0*S
547:      BBB(IP) = BBBB0*S
548:      CCC(IP) = CCCCC0*S
549:      IGG(IP) = IWK4(I)
550: 260      continue
551: C
552: C
553:      else
554: C
555: C**** DETERMINATION OF OUTGOING PARTICLE PARAMETER( ENERGY GROUP AND
556: C      DIRECTION WHEN ANGULAR DISTRIBUTION IS GIVEN BY PL MOMENTS
557: C
558:      call RANU2( IRAND, R, NCOLS*2, ICON )
559: *VOCL LOOP,NOVREC
560:      do 270 I = 1, NCOLS
561:          IP = LSCOL(I)
562:          L0 = (IWK2(I)-1)*NDST3 + NGSX(IWK3(I))*3
563:          NNNX1 = NNNX(IWK3(I))
564:
565: C/#IF ROUND OFF(NEAREST)
566:          L1 = MIN(INT(NNNX1*R(I))+1,NNNX1)
567: C/#ELSE
568:          L1 = NNNX1*R(I) + 1
569: C/#ENDIF
570:
571:          RR = R(I+NCOLS) - SSCBX(L1+L0)
572:
573: C/#IF ROUND OFF(NEAREST)
574:          if ( SSCBX(L1+L0).ge.1. ) RR = -0.5
575:          L2 = 2*L1 + NNNX1 + L0
576:          L3 = MIN(INT(L2+RR),L2)
577: cccc          L3 = INT(2*L1+RR) + NNNX1 + L0
578: C/#ELSE
579:          L2 = 2*L1 + NNNX1 + L0
580:          L3 = L2 + RR
581: C/#ENDIF
582:          IWK4(I) = ISCBX(L3) + IWK3(I) - NUSX - 1
583: C      WEIGHT REDUCTION
584:          WWW(IP) = WWW(IP)*SNAPX((IWK2(I)-1)*NTGX+IWK3(I))
585: 270      continue

```

```

586: C
587: C ..... ENERGY CUT OFF (PL)
588: C
589:      if ( NTGX.gt.NGROUP ) then
590: C
591: c      .... FOR DETAILED MONITORING ONLY ....
592: C
593:      if ( JMNTR.ne.0 ) then
594: *VOCL LOOP,SCALAR
595:          do 280 I = 1, NCOLS
596:              if ( IWK4(I).gt.NGROUP ) then
597:                  NECUT(IWK1(I)) = NECUT(IWK1(I)) + 1
598:                  WECUT(IWK1(I)) = WECUT(IWK1(I)) + WWW(LSCOL(I))
599:              end if
600: 280          continue
601:          end if
602: C
603:          NN = 0
604: *VOCL LOOP,NOVREC
605:          do 290 I = 1, NCOLS
606:              IP = LSCOL(I)
607:              if ( IWK4(I).gt.NGROUP ) then
608:                  NDEAD = NDEAD + 1
609:                  LSDED(NDEAD) = IP
610:                  NCNTR(8) = NCNTR(8) + 1
611:                  WCNTR(8) = WCNTR(8) + WWW(IP)
612:              else
613:                  NN = NN + 1
614:                  LSCOL(NN) = IP
615:                  IWK1(NN) = IWK1(I)
616:                  IWK2(NN) = IWK2(I)
617:                  IWK3(NN) = IWK3(I)
618:                  IWK4(NN) = IWK4(I)
619:              end if
620: 290          continue
621:          NCOLS = NN
622:          end if
623: C
624: C ..... SCATTERING ANGLE (PL) .....
625: C
626:      if ( NSCTX.gt.0 ) then
627:          N2 = NCOLS*2
628:          call RANU2( IRAND, R, NCOLS*3, ICON )
629: *VOCL LOOP,NOVREC
630:          do 300 I = 1, NCOLS
631:              IP = LSCOL(I)
632: C
633:              ETA = PI2*R(N2+I)
634:              SINETA = SIN(ETA)
635:              COSETA = COS(ETA)
636: C
637:              IG = IWK4(I) - IWK3(I) + NUSX
638:              L0 = (IWK2(I)-1)*NSCT3N + NGSX(IWK3(I))*NSCT3
639:              &          + IG*NSCT3
640:              if ( SDBBX(L0+1).ge.0. ) then
641:
642: C/#IF ROUND OFF(NEAREST)
643:                  L1 = MIN(INT(NSCTX*R(I))+1,NSCTX)
644: C/#ELSE
645:                  L1 = NSCTX*R(I) + 1
646: C/#ENDIF
647:
648:                  RR = R(NCOLS+I) - SDBBX(L1+L0)
649:
650: C/#IF ROUND OFF(NEAREST)

```

src/gmvp/colisn.f

```

651:          L2      = 2*L1 + NSCTX + L0
652:          L3      = MIN(INT(L2+RR),L2)
653:          ccccc   L3      = INT(2*L1+RR) + NSCTX + L0
654: C/#ELSE
655: *              L2      = 2*L1 + NSCTX + L0
656: *              L3      = L2 + RR
657: C/#ENDIF
658:          FM       = SDBBX(L3)
659: C
660:          AAAAA0   = AAA(IP)
661:          BBBB0    = BBB(IP)
662:          CCCCC0   = CCC(IP)
663:          SINPSI   = SQRT(1.-FM*FM)
664:          STHETA   = 1. - AAAAA0*AAAAA0
665: *VOCL STMT,IP(100)
666:          if ( STHETA.gt.0.0 ) then
667:             STHETA = SQRT(STHETA)
668:             COSPHI = BBBB0/STHETA
669:             SINPHI = CCCCC0/STHETA
670:          else
671:             STHETA = 0.0
672:             COSPHI = 1.0
673:             SINPHI = 0.0
674:          end if
675: C
676:          BBBB0    = BBBB0*FM
677:          &        + (AAAAA0*COSPHI*COSETA-SINPHI*SINETA)*SINPSI
678:          CCCCC0   = CCCCC0*FM
679:          &        + (AAAAA0*SINPHI*COSETA+COSPHI*SINETA)*SINPSI
680:          AAAAA0   = AAAAA0*FM - COSETA*SINPSI*STHETA
681:          else
682: C
683: C      ISOTROPIC SCATTERING SAMPLING
684: C
685:          CCCCC0   = 1. - 2.*(R(I))
686:          A        = SQRT(1.-CCCCC0*CCCCC0)
687:          AAAAA0   = A*COSETA
688:          BBBB0    = A*SINETA
689:          end if
690: C
691:          S        = 1./
692:          &        SQRT(AAAAA0*AAAAA0+BBBBB0*BBBBB0+CCCCC0*CCCCC0)
693:          AAA(IP)  = AAAAA0*S
694:          BBB(IP)  = BBBB0*S
695:          CCC(IP)  = CCCCC0*S
696:          IGG(IP)  = IWK4(I)
697:          300      continue
698:          else
699: C
700: C      .... ISOTROPIC SCATTERING SAMPLING (NSCTX=0)
701: C
702:          call RANU2( IRAND, R, NCOLS*2, ICON )
703: *VOCL LOOP,NOVREC
704:          do 310 I = 1, NCOLS
705:             IP    = LSCOL(I)
706: C
707:             CCCCC0 = 1. - 2.*(R(I))
708:             A      = SQRT(1.-CCCCC0*CCCCC0)
709:             ETA    = PI2*R(NCOLS+I)
710:             AAAAA0 = A*COS(ETA)
711:             BBBB0  = A*SIN(ETA)
712:             S      = 1./
713:             &      SQRT(AAAAA0*AAAAA0+BBBBB0*BBBBB0+CCCCC0*CCCCC0)
714:             AAA(IP) = AAAAA0*S
715:             BBB(IP) = BBBB0*S

```

```

716:          CCC(IP) = CCCCC0*S
717:          IGG(IP) = IWK4(I)
718:          310      continue
719:          end if
720:          end if
721: C
722: C
723: C
724: C**** SPLITTING ON THE BASIS OF IMPORTANCE
725: C
726: C      IWK4(I) : enrgy group
727: C      IWK1(I) : region #
728: C
729: C      IWK3(I) : number of particles increased after splitting or R.R.
730: C              (it may be negative)
731: C
732: C      RP(I)   : new/old importance ratio.
733: C      IWRK(I) :
734: C
735:          if ( JIMPT.gt.0 ) then
736:             call RANU2( IRAND, R, NCOLS, ICON )
737:             NSPLTT = 0
738: *VOCL LOOP,NOVREC
739:             do 320 I = 1, NCOLS
740:                XIMPN = XIMP(IWK4(I),IWK1(I))
741:                RP(I) = XIMPN/XIM(LSCOL(I))
742:                XIM(LSCOL(I)) = XIMPN
743:                if ( RP(I).lt.1. ) then
744:                   NP = 1 + RP(I) - R(I)
745:                   IWK3(I) = NP - 1
746:                   NCNTR(10) = NCNTR(10) + NP
747:                   W = NP*WWW(LSCOL(I))
748:                   WCNTR(10) = WCNTR(10) + W
749:                else
750:                   NP1 = RP(I)
751:                   P = RP(I) - NP1
752:                   IWK3(I) = NP1 + P - R(I)
753:                   NSPLTT = NSPLTT + IWK3(I)
754:                end if
755:             320      continue
756: C
757: C      .... for detailed monitoring only ....
758: C
759:          if ( JMNTR.ne.0 ) then
760: *VOCL LOOP,SCALAR
761:             do 330 I = 1, NCOLS
762:                if ( RP(I).lt.1. ) then
763:                   NP = 1 + RP(I) - R(I)
764:                   W = NP*WWW(LSCOL(I))
765:                   NSURV(IWK4(I),IWK1(I)) = NSURV(IWK4(I),IWK1(I)) + NP
766:                   WSURV(IWK4(I),IWK1(I)) = WSURV(IWK4(I),IWK1(I)) + W
767:                end if
768:             330      continue
769:             end if
770: C
771:          if ( JMNTR.ne.0 ) then
772: *VOCL LOOP,SCALAR
773:             do 340 I = 1, NCOLS
774:                if ( IWK3(I).lt.0 ) then
775:                   NKILD(IWK4(I),IWK1(I)) = NKILD(IWK4(I),IWK1(I)) + 1
776:                   WKILD(IWK4(I),IWK1(I)) = WKILD(IWK4(I),IWK1(I))
777:                   &      + WWW(LSCOL(I))
778:                end if
779:             340      continue
780:             end if

```

src/gmvp/colisn.f

```

781: C
782:     NN      = 0
783: *VOCL LOOP,NOVREC
784:     do 350 I = 1, NCOLS
785:         IP    = LSCOL(I)
786:         if ( IWK3(I).lt.0 ) then
787:             NDEAD = NDEAD + 1
788:             LSDED(NDEAD) = IP
789:             NCNTR(9) = NCNTR(9) + 1
790:             WCNTR(9) = WCNTR(9) + WWW(IP)
791:         else
792:             NN      = NN + 1
793:             LSCOL(NN) = IP
794:             IWK1(NN) = IWK1(I)
795:             IWK3(NN) = IWK3(I)
796:             IWK4(NN) = IWK4(I)
797:             RP(NN)  = RP(I)
798:         end if
799: 350 continue
800: C
801:     if ( NSPLTT.le.NDEAD ) then
802: C         N = 0
803: C         DO 2300 I=1,NN
804: C             DO 2300 II = 1,IWK3(I)
805: C                 N = N + 1
806: C                 IWRK(N) = I
807: C2300 CONTINUE
808: C         MA = 0
809: C         do 360 I = 1, NN
810: C             MA = MAX(MA,IWK3(I))
811: 360 continue
812: C         N = 0
813: C         do 380 K = 1, MA
814: C             do 370 I = 1, NN
815: C                 if ( IWK3(I).ge.K ) then
816: C                     N = N + 1
817: C                     IWRK(N) = I
818: C                 end if
819: C             continue
820: 380 continue
821: C             I1 = NN
822: C
823: C ..... SPLITTING PREVENTED BECAUSE OF INSUFFICIENT BANK SIZE .....
824: C
825: C         else
826: C             NSPLTT = 0
827: C             N = 0
828: C             I1 = 0
829: C             do 400 I = 1, NN
830: C                 K = IWK3(I)
831: C                 NSPLTT = NSPLTT + K
832: C                 if ( NSPLTT.gt.NDEAD ) go to 410
833: C                 I1 = I
834: C                 do 390 II = 1, K
835: C                     N = N + 1
836: C                     IWRK(N) = I
837: C                 continue
838: C             continue
839: C             go to 430
840: C
841: C         410 NSPLTT = N
842: C
843: C         if ( I1.lt.NN ) then
844: C *VOCL LOOP,NOVREC
845: C             do 420 I = I1 + 1, NN

```

```

846: C             IP    = LSCOL(I)
847: C             IF( IWK3(I).EQ.0 ) THEN
848: C                 if ( RP(I).lt.1. ) then
849: C                     WWW(IP) = WWW(IP) /RP(I)
850: C                 else if ( IWK3(I).gt.0 ) then
851: C                     NCNTR(6) = NCNTR(6) + 1
852: C                     WCNTR(6) = WCNTR(6) + WWW(IP)
853: C                 end if
854: 420 continue
855: C             end if
856: 430 continue
857: C             end if
858: C
859: C             W = 0.0
860: *VOCL LOOP,NOVREC
861: C             do 440 I = 1, I1
862: C                 IP    = LSCOL(I)
863: C                 WWW(IP) = WWW(IP) /RP(I)
864: C                 RP(I) = WWW(IP)*IWK3(I)
865: C                 W = W + RP(I)
866: 440 continue
867: C                 NCNTR(5) = NCNTR(5) + NSPLTT
868: C                 WCNTR(5) = WCNTR(5) + W
869: C
870: C             .... FOR DETAILED MONITORING ONLY ....
871: C
872: C             if ( JMNTR.ne.0 ) then
873: C *VOCL LOOP,SCALAR
874: C             do 450 I = 1, I1
875: C                 NSPLT(IWK4(I),IWK1(I)) = NSPLT(IWK4(I),IWK1(I)) +
876: C                 & IWK3(I)
877: C                 WSPLT(IWK4(I),IWK1(I)) = WSPLT(IWK4(I),IWK1(I)) + RP(I)
878: 450 continue
879: C             end if
880: C
881: C             NDD0 = NDEAD
882: *VOCL LOOP,NOVREC
883: C             do 460 I = 1, NSPLTT
884: C                 I0 = IWRK(I)
885: C                 L0 = LSCOL(I0)
886: C                 L1 = LSDED(NDD0)
887: C                 NDD0 = NDD0 - 1
888: C                 NN = NN + 1
889: C                 LSCOL(NN) = L1
890: C                 IWK1(NN) = IWK1(I0)
891: C                 IWK4(NN) = IWK4(I0)
892: C                 AAA(L1) = AAA(L0)
893: C                 BBB(L1) = BBB(L0)
894: C                 CCC(L1) = CCC(L0)
895: C                 XXX(L1) = XXX(L0)
896: C                 YYY(L1) = YYY(L0)
897: C                 ZZZ(L1) = ZZZ(L0)
898: C                 WWW(L1) = WWW(L0)
899: C                 IGG(L1) = IGG(L0)
900: C                 IZZ(L1) = IZZ(L0)
901: C                 TTT(L1) = TTT(L0)
902: C                 XIM(L1) = XIM(L0)
903: C                 if ( JLATT.ne.0 ) LEVL(L1) = LEVL(L0)
904: C                 CCCC if ( JTLT.ne.0 ) IBREG(L1) = IBREG(L0)
905: C                 IBREG(L1) = IBREG(L0)
906: C                 if ( JTIME.ne.0 ) ITT(L1) = ITT(L0)
907: 460 continue
908: C             if ( JLATT.ne.0.and.NSPLTT.gt.0 ) then
909: C                 do 480 K = 1, NEST
910: C                     NDD1 = NDEAD

```

src/gmvp/colisn.f

```

911: *VOCL LOOP,NOVREC
912:       do 470 I = 1, NSPLTT
913:         IO = IWRK(I)
914:         L0 = LSCOL(IO)
915:         L1 = LSDED(NDD1)
916:         NDD1 = NDD1 - 1
917:         LZZ(L1,K) = LZZ(L0,K)
918:         LPOS(L1,K) = LPOS(L0,K)
919:         if ( JHLAT.ne.0 ) LCRS(L1,K) = LCRS(L0,K)
920:         if ( JTLT.ne.0 ) IBSPC(L1,K) = IBSPC(L0,K)
921:       470 continue
922:       480 continue
923:     end if
924: C
925: C
926: C .... optional bank parameters
927: C
928:   if ( NSPLTT.gt.0 ) then
929:     do 500 K = 1, KDBNK(0)
930:       NDD1 = NDEAD
931:       do 490 I = 1, NSPLTT
932:         IO = IWRK(I)
933:         L0 = LSCOL(IO)
934:         L1 = LSDED(NDD1)
935:         NDD1 = NDD1 - 1
936:         DBNK(L1,K) = DBNK(L0,K)
937:       490 continue
938:     500 continue
939:     do 520 K = 1, KIBNK(0)
940:       NDD1 = NDEAD
941:       do 510 I = 1, NSPLTT
942:         IO = IWRK(I)
943:         L0 = LSCOL(IO)
944:         L1 = LSDED(NDD1)
945:         NDD1 = NDD1 - 1
946:         IBNK(L1,K) = IBNK(L0,K)
947:       510 continue
948:     520 continue
949:   end if
950: C
951:   NDEAD = NDD0
952:   NCOLS = NN
953:
954: end if
955: C
956: C
957: C**** END OF SPLITTING BASED ON IMPORTANCE
958: C
959: C
960: C**** WEIGHT KILL OR WEIGHT WINDOW
961: C
962: C
963: C ***** Russin roulette only ****
964: C
965:   if ( JWWND.eq.0.and.JRRLT.ne.0 ) then
966:     call RANU2( IRAND, R, NCOLS, ICON )
967: C
968: C .... FOR DETAILED MONITORING ONLY .....
969: C
970:   if ( JMNTR.gt.0 ) then
971: *VOCL LOOP,SCALAR
972:       do 530 I = 1, NCOLS
973:         IP = LSCOL(I)
974:         WW = WWW(IP)
975:         WK = WKIL(IWK4(I),IWK1(I))

```

```

976:         WS = WSRV(IWK4(I),IWK1(I))
977:         if ( JWTIM.ne.0 ) then
978:           ITTII = ITT(LSCOL(I))
979:           WK = WK*WTIME(ITTII)
980:           WS = WS*WTIME(ITTII)
981:         end if
982:         if ( JRWVR.ne.0 ) then
983:           WK = WK*DBNK(IP,KDBNK(1))
984:           WS = WS*DBNK(IP,KDBNK(1))
985:         end if
986: c temporary #####
987:         WS = MAX(10d-10,WS)
988:         WK = MAX(10d-10,WK)
989: c #####
990:         if ( WW.lt.WK ) then
991:           P = WW/WS
992:           if ( R(I).lt.P ) then
993:             NSURV(IWK4(I),IWK1(I)) = NSURV(IWK4(I),IWK1(I)) +
994:             & 1
995:             WSURV(IWK4(I),IWK1(I)) = WSURV(IWK4(I),IWK1(I))
996:             & + WW
997:           else
998:             NKILD(IWK4(I),IWK1(I)) = NKILD(IWK4(I),IWK1(I)) +
999:             & 1
1000:             WKILD(IWK4(I),IWK1(I)) = WKILD(IWK4(I),IWK1(I))
1001:             & + WW
1002:           end if
1003:         end if
1004:       530 continue
1005:     end if
1006: C
1007: C
1008:   NN = 0
1009:   if ( JWTIM.eq.0.and.JRWVR.eq.0 ) then
1010: *VOCL LOOP,NOVREC
1011:       do 540 I = 1, NCOLS
1012:         IP = LSCOL(I)
1013:         WWW = WWW(IP)
1014:         WW = WWW(LSCOL(I))
1015:         WK = WKIL(IWK4(I),IWK1(I))
1016:         WS = WSRV(IWK4(I),IWK1(I))
1017: C
1018:         if ( WW.lt.WK ) then
1019:           P = WW/WS
1020:           if ( R(I).lt.P ) then
1021:             NCNTR(12) = NCNTR(12) + 1
1022:             WCNTR(12) = WCNTR(12) + WW
1023:             WWW(LSCOL(I)) = WS
1024:           else
1025:             NDEAD = NDEAD + 1
1026:             LSDED(NDEAD) = IP
1027:             LSDED(NDEAD) = LSCOL(I)
1028:             LSCOL(I) = 0
1029:             NCNTR(11) = NCNTR(11) + 1
1030:             WCNTR(11) = WCNTR(11) + WW
1031:           end if
1032:         end if
1033:         if ( LSCOL(I).gt.0 ) then
1034:           NN = NN + 1
1035:           LSCOL(NN) = LSCOL(I)
1036:         end if
1037:       540 continue
1038:     else
1039: *VOCL LOOP,NOVREC
1040:       do 550 I = 1, NCOLS

```

src/gmvp/colisn.f

```

1041:          IP      = LSCOL(I)
1042:          WW      = WWW(IP)
1043: C
1044:          WK      = WKIL(IWK4(I),IWK1(I))
1045:          WS      = WSRV(IWK4(I),IWK1(I))
1046:          if ( JWTIM.ne.0 ) then
1047:            ITTII  = ITT(IP)
1048: CCC          WK      = WKIL(IWK4(I),IWK1(I)) * WTIME(ITTII)
1049:          WK      = WKIL(IWK4(I),IWK1(I))*WTIME(ITTII)
1050:          WS      = WSRV(IWK4(I),IWK1(I))*WTIME(ITTII)
1051:          end if
1052:          if ( JRWVR.ne.0 ) then
1053:            WK      = WK*DBNK(IP,KDBNK(1))
1054:            WS      = WS*DBNK(IP,KDBNK(1))
1055:          end if
1056: c temporary #####
1057:          WS      = MAX(10d-10,WS)
1058:          WK      = MAX(10d-10,WK)
1059: c #####
1060:          if ( WW.lt.WK ) then
1061:            P      = WW/WS
1062:            if ( R(I).lt.P ) then
1063:              NCNTR(12) = NCNTR(12) + 1
1064:              WCNTR(12) = NCNTR(12) + WW
1065:              WWW(IP) = WS
1066:            else
1067:              NDEAD  = NDEAD + 1
1068:              LSDED(NDEAD) = IP
1069:              LSCOL(I) = 0
1070:              NCNTR(11) = NCNTR(11) + 1
1071:              WCNTR(11) = WCNTR(11) + WW
1072:            end if
1073:          end if
1074:          if ( IP.gt.0 ) then
1075:            NN      = NN + 1
1076:            LSCOL(NN) = LSCOL(I)
1077:          end if
1078: 550          continue
1079:          end if
1080:          NCOLS    = NN
1081: C
1082:          else if ( JWWND.ne.0 ) then
1083: C
1084: C**** WEIGHT WINDOW
1085: C
1086:          call RANU2( IRAND, R, NCOLS, ICON )
1087: C
1088: C .... FOR DETAILED MONITORING ONLY .....
1089: C
1090:          if ( JMNTR.gt.0 ) then
1091: *VOCL LOOP,SCALAR
1092:          do 560 I = 1, NCOLS
1093:            IP      = LSCOL(I)
1094:            WW      = WWW(IP)
1095:            WK      = WKIL(IWK4(I),IWK1(I))
1096:            WS      = WSRV(IWK4(I),IWK1(I))
1097:            if ( JWTIM.ne.0 ) then
1098:              ITTII  = ITT(LSCOL(I))
1099:              WK      = WK*WTIME(ITTII)
1100:              WS      = WS*WTIME(ITTII)
1101:            end if
1102:            if ( JRWVR.ne.0 ) then
1103:              WK      = WK*DBNK(IP,KDBNK(1))
1104:              WS      = WS*DBNK(IP,KDBNK(1))
1105:            end if

```

```

1106:          WS      = MAX(WMIN,WS)
1107:          WK      = MAX(WMIN,WK)
1108:          RP(I)   = WW/WS
1109:          if ( WW.lt.WK ) then
1110:            NP      = 1 + RP(I) - R(I)
1111:            NSURV(IWK4(I),IWK1(I)) = NSURV(IWK4(I),IWK1(I)) + NP
1112:            WSURV(IWK4(I),IWK1(I)) = WSURV(IWK4(I),IWK1(I))
1113:              &      + NP*WW
1114:          end if
1115: 560          continue
1116:          end if
1117: C
1118:          NSPLTT  = 0
1119: C
1120:          if ( JWTIM.eq.0.and.JRWVR.eq.0 ) then
1121: *VOCL LOOP,NOVREC
1122:          do 570 I = 1, NCOLS
1123:            IP      = LSCOL(I)
1124:            WW      = WWW(IP)
1125:            RP(I)   = WW/WSRV(IWK4(I),IWK1(I))
1126:            if ( WW.lt.WKIL(IWK4(I),IWK1(I)) ) then
1127:              NP      = 1 + RP(I) - R(I)
1128:              IWK3(I) = NP - 1
1129:              NCNTR(10) = NCNTR(10) + NP
1130:              WCNTR(10) = WCNTR(10) + NP*WW
1131:            else if ( RP(I).gt.2. ) then
1132:              NP1     = RP(I)
1133:              P       = RP(I) - NP1
1134:              IWK3(I) = NP1 + P - R(I)
1135:              NSPLTT  = NSPLTT + IWK3(I)
1136:            else
1137:              RP(I)   = 1.0
1138:              IWK3(I) = 0
1139:            end if
1140: 570          continue
1141:          else
1142: *VOCL LOOP,NOVREC
1143:          do 580 I = 1, NCOLS
1144:            IP      = LSCOL(I)
1145:            WW      = WWW(IP)
1146:            WK      = WKIL(IWK4(I),IWK1(I))
1147:            WS      = WSRV(IWK4(I),IWK1(I))
1148:            if ( JWTIM.ne.0 ) then
1149:              ITTII  = ITT(IP)
1150:              WK      = WK*WTIME(ITTII)
1151:              WS      = WS*WTIME(ITTII)
1152:            end if
1153:            if ( JRWVR.ne.0 ) then
1154:              WK      = WK*DBNK(IP,KDBNK(1))
1155:              WS      = WS*DBNK(IP,KDBNK(1))
1156:            end if
1157:            WS      = MAX(WMIN,WS)
1158:            WK      = MAX(WMIN,WK)
1159:
1160:            RP(I)   = WW/WS
1161:
1162:            if ( WW.lt.WK ) then
1163:              NP      = 1 + RP(I) - R(I)
1164:              IWK3(I) = NP - 1
1165:              NCNTR(10) = NCNTR(10) + NP
1166:              WCNTR(10) = WCNTR(10) + NP*WW
1167:            else if ( RP(I).gt.2. ) then
1168:              NP1     = RP(I)
1169:              P       = RP(I) - NP1
1170:              IWK3(I) = NP1 + P - R(I)

```

src/gmvp/colisn.f

```

1171:          NSPLTT = NSPLTT + IWK3(I)
1172:        else
1173:          RP(I) = 1.0
1174:          IWK3(I) = 0
1175:        end if
1176: 580      continue
1177:    end if
1178:  C
1179:  C .... FOR DETAILED MONITORING ONLY .....
1180:  C
1181:    if ( JMNTR.gt.0 ) then
1182:  *VOCL LOOP,SCALAR
1183:    do 590 I = 1, NCOLS
1184:      if ( IWK3(I).lt.0 ) then
1185:        NKILD(IWK4(I),IWK1(I)) = NKILD(IWK4(I),IWK1(I)) + 1
1186:        WKILD(IWK4(I),IWK1(I)) = WKILD(IWK4(I),IWK1(I))
1187:      &      + WWW(LSCOL(I))
1188:    end if
1189: 590    continue
1190:  end if
1191:  C
1192:  C
1193:    NN = 0
1194:  *VOCL LOOP,NOVREC
1195:    do 600 I = 1, NCOLS
1196:      IP = LSCOL(I)
1197:      if ( IWK3(I).ge.0 ) then
1198:        NN = NN + 1
1199:        LSCOL(NN) = IP
1200:        IWK3(NN) = IWK3(I)
1201:        RP(NN) = RP(I)
1202:      else
1203:        NDEAD = NDEAD + 1
1204:        LSDED(NDEAD) = IP
1205:        NCNTR(9) = NCNTR(9) + 1
1206:        WCNTR(9) = WCNTR(9) + WWW(IP)
1207:      end if
1208: 600    continue
1209:  C
1210:    if ( NSPLTT.le.NDEAD ) then
1211:      N = 0
1212:      DO 3800 I=1,NN
1213:      DO 3800 II = 1,IWK3(I)
1214:      N = N + 1
1215:      IWRK(N) = I
1216:  C3800    CONTINUE
1217:      N = 0
1218:      MA = 0
1219:      do 610 I = 1, NN
1220:        MA = MAX(MA,IWK3(I))
1221: 610      continue
1222:      do 630 K = 1, MA
1223:        do 620 I = 1, NN
1224:          if ( IWK3(I).ge.K ) then
1225:            N = N + 1
1226:            IWRK(N) = LSCOL(I)
1227:          end if
1228: 620        continue
1229: 630      continue
1230:      I1 = NN
1231:    else
1232:      NSPLTT = 0
1233:      N = 0
1234:      I1 = 0
1235:      do 650 I = 1, NN

```

```

1236:      K = IWK3(I)
1237:      NSPLTT = NSPLTT + K
1238:      if ( NSPLTT.gt.NDEAD ) go to 660
1239:      I1 = I
1240:      do 640 II = 1, K
1241:        N = N + 1
1242:        IWRK(N) = LSCOL(I)
1243: 640      continue
1244: 650    continue
1245:    go to 680
1246:  C
1247: 660    NSPLTT = N
1248:    if ( I1.lt.NN ) then
1249:  *VOCL LOOP,NOVREC
1250:      do 670 I = I1 + 1, NN
1251:        IP = LSCOL(I)
1252:  CM      IF( IWK3(I).EQ.0) THEN
1253:        if ( RP(I).lt.1.0 ) then
1254:          WWW(IP) = WWW(IP) /RP(I)
1255:        else
1256:          NCNTR(6) = NCNTR(6) + 1
1257:          WCNTR(6) = WCNTR(6) + WWW(IP)
1258:        end if
1259: 670      continue
1260:    end if
1261:  C
1262: 680    continue
1263:  end if
1264:  C
1265:    W = 0.
1266:  *VOCL LOOP,NOVREC
1267:    do 690 I = 1, I1
1268:      IP = LSCOL(I)
1269:      WWW(IP) = WWW(IP) /RP(I)
1270:      RP(I) = WWW(IP)*IWK3(I)
1271:      W = W + RP(I)
1272: 690    continue
1273:  C
1274:      NCNTR(5) = NCNTR(5) + NSPLTT
1275:      WCNTR(5) = WCNTR(5) + W
1276:  C
1277:  C .... FOR DETAILED MONITORING ONLY .....
1278:  C
1279:    if ( JMNTR.gt.0 ) then
1280:      do 700 I = 1, I1
1281:        NSPLT(IWK4(I),IWK1(I)) = NSPLT(IWK4(I),IWK1(I)) +
1282:      &      IWK3(I)
1283:        WSPLT(IWK4(I),IWK1(I)) = WSPLT(IWK4(I),IWK1(I)) + RP(I)
1284: 700      continue
1285:    end if
1286:  C
1287:    NDD0 = NDEAD
1288:  *VOCL LOOP,NOVREC
1289:    do 710 I = 1, NSPLTT
1290:      L0 = IWRK(I)
1291:      L1 = LSDED(NDD0)
1292:      NDD0 = NDD0 - 1
1293:      NN = NN + 1
1294:      LSCOL(NN) = L1
1295:      AAA(L1) = AAA(L0)
1296:      BBB(L1) = BBB(L0)
1297:      CCC(L1) = CCC(L0)
1298:      XXX(L1) = XXX(L0)
1299:      YYY(L1) = YYY(L0)
1300:      ZZZ(L1) = ZZZ(L0)

```

src/gmvp/colisn.f

```

1301:      WWW(L1) = WWW(L0)
1302:      IGG(L1) = IGG(L0)
1303:      IZZ(L1) = IZZ(L0)
1304:      TTT(L1) = TTT(L0)
1305:      if ( JLATT.ne.0 ) LEVL(L1) = LEVL(L0)
1306: CCCCCCCC if ( JTLT.ne.0 ) IBREG(L1) = IBREG(L0)
1307:      IBREG(L1) = IBREG(L0)
1308:      if ( JTIME.ne.0 ) ITT(L1) = ITT(L0)
1309:      710 continue
1310: C
1311:      if ( JLATT.ne.0.and.NSPLTT.gt.0 ) then
1312:      do 730 K = 1, NEST
1313:      NDD1 = NDEAD
1314: *VOCL LOOP,NOVREC
1315:      do 720 I = 1, NSPLTT
1316:      L0 = IWRK(I)
1317:      L1 = LSDED(NDD1)
1318:      NDD1 = NDD1 - 1
1319:      LZZ(L1,K) = LZZ(L0,K)
1320:      LPOS(L1,K) = LPOS(L0,K)
1321:      if ( JHLAT.ne.0 ) LCRS(L1,K) = LCRS(L0,K)
1322:      if ( JTLT.ne.0 ) IBSPC(L1,K) = IBSPC(L0,K)
1323:      720 continue
1324:      730 continue
1325:      end if
1326: C
1327: C .... optional bank parameters
1328: C
1329:      if ( NSPLTT.gt.0 ) then
1330:      do 750 K = 1, KDBNK(0)
1331:      NDD1 = NDEAD
1332:      do 740 I = 1, NSPLTT
1333:      L0 = IWRK(I)
1334:      L1 = LSDED(NDD1)
1335:      NDD1 = NDD1 - 1
1336:      DBNK(L1,K) = DBNK(L0,K)
1337:      740 continue
1338:      750 continue
1339:      do 770 K = 1, KIBNK(0)
1340:      NDD1 = NDEAD
1341:      do 760 I = 1, NSPLTT
1342:      L0 = IWRK(I)
1343:      L1 = LSDED(NDD1)
1344:      NDD1 = NDD1 - 1
1345:      IBNK(L1,K) = IBNK(L0,K)
1346:      760 continue
1347:      770 continue
1348:      end if
1349: C
1350:      NDEAD = NDD0
1351:      NCOLS = NN
1352:      end if
1353: C
1354: C .... SEND PARTICLES TO FREE-FLIGHT STACK ....
1355: C
1356:      NN = NFFL(NZONE+1)
1357:      do 780 I = 1, NCOLS
1358:      IP = LSCOL(I)
1359:      LSFFL(NN+I) = IP
1360:      IZFFL(NN+I) = IZZ(IP)
1361:      780 continue
1362:      if ( KIBNK(5).ne.0 ) then
1363:      do 790 I = 1, NCOLS
1364:      IP = LSCOL(I)
1365:      IBNK(IP,KIBNK(5)) = 0
1366:      790 continue
1367:      end if
1368: *VOCL LOOP,SCALAR
1369:      do 800 I = 1, NCOLS
1370:      NFFL(IZFFL(NN+I)) = NFFL(IZFFL(NN+I)) + 1
1371:      800 continue
1372:      NFFL(NZONE+1) = NN + NCOLS
1373:      NCOLS = 0
1374: C
1375:      return
1376:      end

```

src/gmvp/dumpbk.f

```
1:      subroutine DUMPBK( N1,N2,IPRT,CHAR, A,      IA,      H,      IH )
2: C=====
3: C  PURPOSE : PRINTOUT OF PARTICLE BANK. (FOR DEBUG ONLY)
4: C      (BANK #  N1 TO N2 )
5: C  CALLED IN : VARIOUS ROUTINES
6: C=====
7: CCC  include '../shared/INC/_ARRAY'
8: C
9:      real A(*)
10:     integer IA(*)
11:     real H(*)
12:     integer IH(*)
13: C
14:     character*(*) CHAR
15: C
16:     include 'INC/_XBANK'
17:     include 'INC/_FLAGS'
18:     include '../shared/INC/_SIZES'
19:     include 'INC/_SIZES2'
20: C
21: C ..... COMMENT .....
22: C
23:     write(IPRT,'(/1x,A)') CHAR
24: C
25: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP,
26: C
27:     call DUMPXX( N1,N2,IPRT, H(LXXX),H(LYYY),H(LZZZ), H(LAAA),H(LBBB),
28:     &           H(LCCC), H(LWWW), H(LIZZ), H(LIGG) )
29: C
30: C === LATTICE GEOMETRY ===
31: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP, LEVEL, POS, CRS
32: C
33:     if ( JLATT.ne.0 ) then
34:         ML = 25
35:         do 110 K = 1, NEST
36:             write(IPRT,7000) K
37:             do 100 L = N1, N2, ML
38:                 L2 = MIN(L+ML-1,N2)
39:                 K1 = NBANK*(K-1) + L - 1
40:                 K2 = NBANK*(K-1) + L2 - 1
41:                 write(IPRT,7020) ' --- ', (I,I=L,L2)
42:                 write(IPRT,7020) 'LEVL ', (IH(LLEVL+I),I=K1,K2)
43:                 write(IPRT,7020) 'LZZ ', (IH(LLZZ+I),I=K1,K2)
44:                 write(IPRT,7020) 'LPOS ', (IH(LLPOS+I),I=K1,K2)
45:                 write(IPRT,7020) 'LCRS ', (IH(LLCRS+I),I=K1,K2)
46:             100 continue
47:             110 continue
48:         end if
49: C
50:     7000 format(/1x,('      LEVEL ',I2,'      ',10X:))
51:     7020 format(1X,A5,25I5)
52: C
53:     return
54: end
```


src/gmvp/dumpst.f

```

1:      subroutine DUMPST( CHAR, IPRT, H,      IH )
2: C=====
3: C PURPOSE : PRINTOUT OF STACK DATA. (FOR DEBUG ONLY)
4: C CALLED IN : VARIOUS ROUTINES
5: C=====
6:      real H(*)
7:      integer IH(*)
8: C
9: C      include '../shared/INC/_ARRAY'
10: CC/#IF DYNAMIC
11: CC/# IF NOPINTER
12: C*      integer ia(1)
13: C*      equivalence( h, ia )
14: CC/# ELSE
15: C      INTEGER ia
16: C      pointer ( iph,ia(1) )
17: CC/# ENBIF
18: CC/#ELSE
19: C*      integer ia(1)
20: C*      equivalence( h, ia )
21: CC/#ENDIF
22: CCC
23: C
24:      character*(*) CHAR
25: C
26:      include 'INC/_FLAGS'
27:      include 'INC/_STACK'
28:      include '../shared/INC/_SIZES'
29:      include 'INC/_SIZES2'
30:      include 'INC/_XBANK'
31: C
32: CC/#IF DYNAMIC
33: C      iph = lph
34: CC/#ENDIF
35: C
36: C ..... COMMENT .....
37:      write(IPRT,'(/lx,A)') CHAR
38: C
39: C ..... FLIGHT STACK ....
40: C
41:      ML      = 20
42:      write(IPRT,7000) 'NFFL ', (IH(LNFFL+I),I=0,NZONE)
43:      MM      = IH(LNFFL+NZONE)
44:      do 100 K1 = 1, MM, ML
45:          K2      = MIN(K1+ML-1,MM) - 1
46:          write(IPRT,7020) 'LSFFL', (IH(LLSFFL+I),I=K1-1,K2)
47:          write(IPRT,7020) 'IZFFL', (IH(LIZFFL+I),I=K1-1,K2)
48:          write(IPRT,7020) 'IZZ ', (IH(LIZZ+IH(LLSFFL+I)-1),I=K1-1,K2)
49:      100 continue
50: C
51: C ..... SEARCH STACK ....
52: C
53:      write(IPRT,7000) 'NNXT ', (IH(LNNXT+I),I=0,NZONE)
54:      MM      = IH(LNNXT+NZONE)
55:      do 110 K1 = 1, MM, ML
56:          K2      = MIN(K1+ML-1,MM) - 1
57:          write(IPRT,7020) 'LSSRC', (IH(LLSSRC+I),I=K1-1,K2)
58:          write(IPRT,7020) 'IZNXT', (IH(LIZNXT+I),I=K1-1,K2)
59:          write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSSRC+I)-1),I=K1-1,K2)
60:      110 continue
61: C
62: C ..... COLLISION STACK ....
63: C
64:      write(6,7000) 'NCOLS', NCOLS
65:      MM      = NCOLS

```

```

66:      do 120 K1 = 1, MM, ML
67:          K2      = MIN(K1+ML-1,MM) - 1
68:          write(IPRT,7020) 'LSCOL', (IH(LLSCOL+I),I=K1-1,K2)
69:          write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSCOL+I)-1),I=K1-1,K2)
70:      120 continue
71: C
72: C ..... DEAD PARTICLE STACK ....
73: C
74:      write(IPRT,7000) 'NDEAD', NDEAD
75: C      MM = NDEAD
76: C      DO 260 K1=1,MM,ML
77: C          K2 = MIN(K1+ML-1,MM) - 1
78: C          WRITE(IPRT,7100) 'LSDED', (IH(LLSDED+I),I=K1-1,K2)
79: C      260 CONTINUE
80: C
81: C ..... REFLECTION STACK
82: C
83:      if ( JREFL.ne.0 ) then
84:          write(IPRT,7000) 'NBREF', (IH(LNBREF+I),I=0,NREFS)
85:          MM      = IH(LNBREF+NREFS)
86:          do 130 K1 = 1, MM, ML
87:              K2      = MIN(K1+ML-1,MM) - 1
88:              write(IPRT,7020) 'LSREF', (IH(LLSREF+I),I=K1-1,K2)
89:              write(IPRT,7020) 'IZREF', (IH(LIZREF+I),I=K1-1,K2)
90:              write(IPRT,7040) 'ISREF', (IH(LISREF+I),I=K1-1,K2)
91:          130 continue
92:          end if
93: C
94: C ..... LATTICE SEARCH STACK .....
95: C
96:      if ( JLATT.ne.0 ) then
97:          write(IPRT,7000) 'NXLT ', (IH(LNXLT+I),I=0,NLBZ)
98:          MM      = IH(LNXLT+NLBZ)
99:          do 140 K1 = 1, MM, ML
100:              K2      = MIN(K1+ML-1,MM) - 1
101:              write(IPRT,7020) 'LSLAT', (IH(LLSLAT+I),I=K1-1,K2)
102:              write(IPRT,7040) 'IZLAT', (IH(LIZLAT+I),I=K1-1,K2)
103:          140 continue
104:          end if
105: C
106: C
107: 7000 format(1X,'* ',A6,': ',20I4/(1X,9X,20I4))
108: 7020 format(1X,A5,20I5)
109: 7040 format(1X,A5,20I5/)
110:      return
111:      end

```

src/gmvp/dumpxx.f

```
1: C
2: C ===== AUXILIARY SUBROUTINE =====
3: C
4:       subroutine DUMPXX(N1,N2,IPRT,XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,IZZ,IGG)
5: C
6: C
7:       real*8   XXX(*),YYY(*),ZZZ(*),AAA(*),BBB(*),CCC(*)
8:       real     WWW(*)
9:       integer  IZZ(*),IGG(*)
10: C
11: C
12:       write(IPRT,7001)
13: C
14: C       write(IPRT,7000) (I, XXX(I),YYY(I),ZZZ(I),
15: C &   AAA(I),BBB(I),CCC(I),WWW(I),IZZ(I),IGG(I), I=N1,N2)
16: C
17: C7000 FORMAT(/lx,' # ', ' XXX      ', ' YYY      ', ' ZZZ      ',
18: C &   ' AAA      ', ' BBB      ', ' CCC      ',
19: C &   ' WWW      ', ' IZZ ', ' IGG ' //
20: C & (lx ,I5,1X,1P,6E11.4,E11.4,2I5))
21: C
22: C       7001 format(/lx,' # ', ' XXX      ', ' YYY      ', ' ZZZ      ',
23: C &   ' AAA      ', ' BBB      ', ' CCC      ',
24: C &   ' WWW      ', ' IZZ ', ' IGG ' / )
25: C       7002 format(lx ,I5,1X,1P,6E11.4,E11.4,2I5)
26: C
27:       do 100 I=N1,N2
28: C       if(I.EQ.5.OR.I.EQ.15.OR.I.EQ.58.OR.I.EQ.70.OR.I.EQ.91) then
29:           write(IPRT,7002) I, XXX(I),YYY(I),ZZZ(I),
30:           &   AAA(I),BBB(I),CCC(I),WWW(I),IZZ(I),IGG(I)
31: C       end if
32:       100 continue
33: C
34:       return
35:       end
```

src/gmvp/ffread.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine FFREAD( IN,      K,      V,      NF,      N5,      N6,
3:      &                  IPRTRG )
4: C=====
5: C  PURPOSE:  'FIDO' FORMAT INPUT PROCESSING
6: C  CALLED IN: READSG
7: C
8: C=====
9:      include '../shared/INC/_IUNIT'
10: C
11:      character*4 C, NY(77), K(37), NYNCC
12:      integer NYC
13:      integer IN(37)
14:      real V(37)
15:      character*10 NUMS
16:
17:      real*8 VNUM, NUM
18:
19:      data NUMS /'0123456789'/
20:      data IFREE /1/
21: C
22: C-----
23: C
24:      100 continue
25: C
26:      do 110 I = 1, 37
27:          IN(I) = 0
28:          V(I) = 0.0
29:          K(I) = ' '
30:      110 continue
31:
32:      IEXP = 0
33:      NSCL = 0
34:      NDPN = 0
35:      IEX = 0
36:      NUM = 0.0D0
37:      ISGN = 1
38:      IBLNK = 0
39:      NF = 1
40:      NCC = 1
41:      IFC = 0
42:      IDS = 0
43: C
44:      120 read(N5,fmt = '(76A1,I4)',end =130) (NY(I),I=1,76), NYC
45:      write(NY(77),'(i4)') NYC
46:      if ( IPRTRG.gt.0 ) write(N6,'(40X,76A1,I4)') (NY(I),I=1,76), NYC
47:      C = NY(1)
48:      if ( C.eq.'T'.or.C.eq.' ' .or.C.eq.'+' .or.C.eq.'-'
49:      & .or.C.eq.'E'.or.C.eq.'C'.or.C.eq.'O'.or.C.eq.'&'
50:      & .or.C.eq.'Q'.or.C.eq.'M'.or.C.eq.'N'.or.C.eq.'F'
51:      & .or.C.eq.'A'.or.(C.eq.'.'.and.NY(2).ne.'') ) go to 140
52:      N1 = INDEX(NUMS,C(1:1))
53:      if ( N1.ne.0 ) go to 140
54:
55:      write(N6,'(2X,74A1,I4)') (NY(I),I=3,76), NYC
56:      go to 120
57: C
58: C
59:      130 write(IMG,'(1X,'XXX (FFREAD) END OF FILE ON INPUT''))
60: C
61:      call PRSTOP( 1, 'UNEXPECTED END OF INPUT DATA' )
62:      stop 888
63: C
64:      140 NYNCC = NY(NCC)
65:      NUMM = INDEX(NUMS,NYNCC(1:1))

```

```

66: C
67:      NUMM = NUMM - 1
68:      if ( IFREE.ne.0 ) then
69:          IFC = IFC + 1
70:          if ( IFC.gt.12 ) IFC = 1
71:          if ( IFC.eq.3 ) go to 150
72:          if ( IFC.eq.11.and.IEX.ne.0.and.NYNCC.eq.' ' ) go to 210
73:          if ( NYNCC.eq.'E'.and.IFC.gt.4 ) then
74:              IEX = 1
75:              go to 210
76:          end if
77:      end if
78: C
79:      if ( NYNCC.eq.' ' ) go to 170
80: C
81:      if ( NYNCC.eq.'+' .or. NYNCC.eq.'&' ) then
82:          if ( NUM.ne.0.0D0 ) then
83:              IEX = 1
84:          else
85:              ISGN = 1
86:          end if
87:          go to 210
88:      else if ( NYNCC.eq.'-' ) then
89:          if ( NUM.ne.0.0D0 ) then
90:              IEX = -1
91:          else
92:              ISGN = -1
93:          end if
94:          go to 210
95:      else if ( NYNCC.eq.'E'.and.IBLNK.eq.1 ) then
96:          IEX = 1
97:          go to 210
98:      else
99:          IBLNK = 1
100:          if ( IEX.ne.0 ) then
101:              IEXP = IEXP*10 + IEX*NUMM
102:              go to 160
103:          else if ( NYNCC.eq.'.' ) then
104:              NDPN = -1
105:              go to 160
106:          end if
107:      end if
108: C
109:      150 if ( NUMM.ge.0.and.NUMM.le.9 ) then
110:          NUM = NUM*10D0 + dble(NUMM)
111:          NSCL = NSCL + NDPN
112:      else
113:
114:          if ( K(NF).ne.' ' .and.NYNCC.eq.' ' ) go to 170
115:
116:          K(NF) = NYNCC
117:          IN(NF) = NUM
118:          NUM = 0.0D0
119:          IBLNK = 0
120:
121:          if ( NYNCC.eq.' ' ) go to 210
122:          if ( K(NF).ne.'$'.and.K(NF).ne.'*' ) then
123:              if ( IFREE.eq.0.and.NYNCC.ne.'R'.and.NYNCC.ne.'I'
124:              & .and.NYNCC.ne.'F'.and.NYNCC.ne.'A'.and.NYNCC.ne.'L'
125:              & .and.NYNCC.ne.'K'.and.NYNCC.ne.'Q'.and.NYNCC.ne.'M'
126:              & .and.NYNCC.ne.'N' ) go to 190
127:              if ( IFREE.eq.1.and.NYNCC.eq.'S' ) go to 190
128:              if ( NCC.eq.72 ) write(N6,7000) (NY(I),I=1,76), NYC
129:              go to 210
130:          else

```

src/gmvp/ffread.f

```
131:         IFREE = 0
132:         IDS = 1
133:         if ( NF.eq.1 ) go to 190
134:         if ( K(NF).ne.K(NF-1) ) go to 190
135:         IDS = 0
136:         K(NF) = ' '
137:         NF = NF - 1
138:         go to 210
139:     end if
140: end if
141: C
142: 160 if ( IFC.ne.12 ) go to 210
143: go to 180
144: C
145: 170 if ( IDS.eq.1 ) then
146:     IFREE = 1
147:     IDS = 0
148:     go to 200
149: else if ( IFREE.ne.1 .or. IFC.ne.12 .or. K(NF).eq.' ' ) then
150:     if ( IBLNK.eq.0 ) go to 210
151:     if ( IFREE.eq.1.and.IFC.ne.12 ) go to 210
152: end if
153: C
154: 180 VNUM = ISGN*NUM
155: CCCC V(NF) = VNUM*10.0** ( IEXP+NSCL )
156: V(NF) = VNUM*(10.0D0** ( IEXP+NSCL ))
157: NDPN = 0
158: IEX = 0
159: IEXP = 0
160: NSCL = 0
161: IBLNK = 0
162: NUM = 0.0D0
163: ISGN = 1
164: C
165: 190 NF = NF + 1
166: if ( NYNCC.eq.'S' ) go to 210
167: C
168: 200 IFC = 0
169: if ( IFREE.eq.1 ) NCC = ((NCC+11)/12)*12
170: if ( K(NF-1).eq.'T' ) go to 220
171: C
172: 210 if ( NCC.lt.73 ) then
173:     NCC = NCC + 1
174:     if ( NCC.gt.72 ) go to 170
175:     go to 140
176: end if
177: C
178: 220 NF = NF - 1
179: C
180: if ( NF.le.0 ) go to 100
181: C
182: C980821
183: C read(NY(77),'(A4)') IN(37)
184: read(NY(77),'(I4)') IN(37)
185: return
186: 7000 format(/' ***** INCOMPLETE FIELD AT END OF CARD, SOME DATA MAY ',
187: & 'BE LOST '//' CARD IMAGE FOLLOWS'/1X,76A1,I4)
188: end
```

src/gmvp/find.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine FIND( L, NF )
3: C=====
4: C  PURPOSE: TO DETERMINE THE ROOTS OF Q,THE LTH ORDER ORTHOGONAL
5: C    POLYNOMIAL, LYING IN THE RANGE (-1,+1)
6: C  CALLED IN: ANGLES
7: C  CALLS:   Q
8: C=====
9: C  arguments ( o=output, i=input )
10: C  i  L : order of orthogonal polynomial.
11: C  o  NF : 0 = successfully found roots in range (-1,+1).
12: C      -1 = roots out of range.
13: C      +1 : > 1.0
14: C      -1 : < -1.0
15: C=====
16:   common /RESULT/ POINT(21),   WEIGHT(21),   ROOT(21,21)
17:   common /LOCSIG/  ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
18:   & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCSGG, ISPORC,
19:   & ISPORT, INPBUE, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
20:   & NBS,   NGS,   NDSG,   INGP,   INDS,   NMED,   NELEM,
21:   & NMIX,   NCOEF, NSCT,   NTS,   NTG,   NDSNGP, NDSNGG,
22:   & IADJ,   NME,   LOC,   INGS,   INSG,   I1,   I0,
23:   & KKK,   IXTAPE, IDEL,   ITEML, ITEMG, IRSG, IRDSG,
24:   & ISTR,   IPRIN, IFMU,   IMOM,   IDTF,   ISTAT, IPUN,
25:   & NUS,   NGN,   IHT,   INUS,   INUSN, INGN,   INGMP,
26:   & INNN,   IGGG
27: C
28:   real   VALUE(20)
29: C
30:   LM1    = L - 1
31:   do 100 I = 1, LM1
32:     VALUE(I) = Q(L,ROOT(I,LM1))
33:   100 continue
34: C
35:   ROOT(L,LM1) = 1.0
36:   QTOP = Q(L,1.0)
37:   if ( QTOP.le.0.0 ) then
38: C     IF(IPUN.LE.0) WRITE(I0,1010)L
39: c1010   FORMAT('0-----FIND'' ROOTS OF Q('',I2,'') EXTEND BEYOND +1')
40:     NF    = 1
41:     return
42:   end if
43: C
44:   XLOW = -1.
45:   QLOW = Q(L,-1.)
46:   if ( QLOW*VALUE(1).gt.0.0 ) then
47: C     IF(IPUN.GT.0) WRITE(I0,1020)L
48: c1020   FORMAT('0-----FIND'' ROOTS OF Q('',I2,'') EXTEND BEYOND -1')
49:     NF    = -1
50:     return
51:   end if
52: C
53:   do 120 K = 1, L
54:     XUP = ROOT(K,LM1)
55:     NSP = 0
56: C ..... BINARY SEARCH OF A ROOT .....
57:   110   continue
58:     NSP = NSP + 1
59:     XTRY = (XLOW+XUP)*0.5
60:     QTRY = Q(L,XTRY)
61:     if ( QTRY*QLOW.gt.0.0 ) then
62:       XLOW = XTRY
63:       QLOW = QTRY
64:     else if ( QTRY*XUP.lt.0.0 ) then
65:       XUP = XTRY

```

```

66:       end if
67:       if ( XUP.ne.XLOW.and.NSP.lt.48 ) go to 110
68: C
69:       ROOT(K,L) = XTRY
70:       XLOW = ROOT(K,LM1)
71:       QLOW = VALUE(K)
72:   120 continue
73: C
74:       ROOT(L,LM1) = 0.0
75:       NF = 0
76: C
77:       return
78:     end

```

src/gmvp/fisgen.f

```

1:      subroutine FISGEN( NGROUP,NZONE, NMAT,  NREG,  NTIME, NBANK,
2:      &                  NFBANK,NFBNK0,IREG,  IMAT,  IGRP,  WGTF,
3:      &                  WGTFI,WTIME,WLLIM,MXPGEN,FKAI,KKAI,NGPX,  NGGX,
4:      &                  NTGX,  NMEDX,  SNFPX,  SNAPX,  S2DPX,  S2PPX,
5:      &                  LSCOL,  NCOLS,  LSDED,  NDEAD,  LSFFL,  IZFFL,  NFFL,
6:      &                  XXX,  YYY,  ZZZ,  AAA,  BBB,  CCC,  WWW,
7:      &                  IZZ,  IGG,  TTT,  ITT,  XIM,  KLSF,  DBNK,
8:      &                  KDBNK,  MDBNK,  IBNK,  KIBNK,  MIBNK,  XXXF,  YYYYF,
9:      &                  ZZZF,  IZZF,
10:     &                  DFBK,KDFBK,MDFBK,  IFBK,  KIFBK,MIFBK,
11:     &                  NFISB,  NCNTR,  WCNTR,  IWK4,  IWK5,
12:     &                  IWK6,  IWK7,  IWK8,  RW,  RP,  R,  IWRK,
13:     &                  IRAND,  NEST,  LEVL,  LZZ,  LPOS,  LCRS,
14:     &                  LEVLF,  LZZF,  LPOSF,  LCRSF,  IBREG,  IBSPC,
15:     &                  IBRGF,  IBSPF )
16: C=====
17: C  PURPOSE: FISSION NEUTRON GENERATION
18: C  CALLED IN: COLISN
19: C  CALLS: RANU2
20: C=====
21: C
22: C    === INTER-STACK DATA FLOW ===
23: C
24: C    IF THERE ARE ANY FISSION-PARTICLES IN NON-EIGENVALUE PROBLEMS,
25: C    THE FOLLOWING DATA FLOW OCCURS.
26: C
27: C    DEAD PARTICLE STACK ---> (PARTICLE BANK) ---> FLIGHT STACK
28: C
29: C    === BANK DATA TO BE UPDATED ===
30: C
31: C    (NONE)
32: C
33: C    === BANK DATA ADDED NEWLY (FISSION) ===
34: C
35: C    (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #
36: C    IGG, WWW : ENERGY GROUP & WEIGHT
37: C    LEVL,LZZ,LPOS,LCRS : LATTICE-PARAMETER
38: C
39: C    === FISSION BANK (EIGENVALUE CALCULATION) ===
40: C
41: C    (XXXF,YYYYF,ZZZF),IZZF : POSITION & ZONE #
42: C    LEVLF,LZZF,LPOSF,LCRSF : LATTICE-PARAMETER
43: C
44: C=====
45: C    implicit real*8(A-H,O-Z)
46: C
47: C    include 'INC/_FLAGS'
48: C
49: C    integer JDEBG(*)
50: C
51: C**** region #, material # & group # ...
52: C
53: C    integer IREG(NCOLS), IMAT(NCOLS), IGRP(NCOLS)
54: C
55: C**** CROSS SECTION DATA
56: C
57: C    real SNFPX(NTGX,*), WGTF(*), WGTFI(*), FKAI(NTGX,*)
58: C    real SNAPX(NTGX,*), S2DPX(2,NTGX,*), S2PPX(3,NTGX,*)
59: C    real WTIME(NTIME)
60: C    real WLLIM
61: C    integer KKAI(2,NTGX,*)
62: C
63: C**** TALLY BIN DATA
64: C
65: C**** STACK
66: C
67: C    integer LSCOL(NBANK), LSDED(NBANK)
68: C    integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE)
69: C
70: C**** PARTICLE BANK
71: C
72: C    real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
73: C    real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
74: C    integer IZZ(NBANK), IGG(NBANK)
75: C    integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST)
76: C    integer LCRS(NBANK,NEST)
77: C    integer KLSF(NBANK)
78: C    real*8 TTT(NBANK)
79: C    integer ITT(NBANK)
80: C    real WWW(NBANK), XIM(NBANK)
81: C    integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
82: C
83: C ... optional bank parameters
84: C
85: C    real*8 DBNK(NBANK,*)
86: C    integer KDBNK(0:MDBNK)
87: C    integer IBNK(NBANK,*)
88: C    integer KIBNK(0:MIBNK)
89: C
90: C**** fission particle bank
91: C
92: C    real*8 XXXF(NFBNK0), YYYYF(NFBNK0), ZZZF(NFBNK0)
93: C    integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
94: C    & LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST)
95: C    integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
96: C
97: C    real*8 DFBK(NFBNK0,*)
98: C    integer KDFBK(0:MDFBK)
99: C    integer IFBK(NFBNK0,*)
100: C    integer KIFBK(0:MIFBK)
101: C
102: C**** TALLY ARRAY
103: C
104: C    real*8 NCNTR(*)
105: C    real*8 WCNTR(*)
106: C
107: C**** WORK AREA
108: C
109: C    integer IWK4(NBANK), IWK5(NBANK), IWK6(NBANK), IWK7(NBANK),
110: C    & IWK8(NBANK), IWRK(NBANK)
111: C    real RW(NBANK), RP(NBANK), R(4*NBANK)
112: C
113: C    0..NBK.2NBK.3NBK.4NBK.5NBK.6NBK.7NBK.8NBK.9NBK,10NBK,11NBK
114: C    IWK4 IWK5 IWK6 IWK7 IWK8 RW RP R,IWORK
115: C
116: C-----
117: C
118: C    WMIN = WLLIM
119: C
120: C***** FISSION NEUTRON GENERATION
121: C
122: C    call RANU2( IRAND, R, NCOLS, ICON )
123: C
124: C    NFISS = 0
125: C    if ( JWTIM.eq.0.and.JRWVR.eq.0 ) then
126: C    do 100 I = 1, NCOLS
127: C        IP = LSCOL(I)
128: C        RP(I) = WWW(IP)*SNFPX(IGRP(I),IMAT(I))
129: C        WW = RP(I)*WGTFI(IREG(I))
130: C        NP = WW

```

src/gmvp/fisgen.f

```

131: C/#IF ROUND OFF(NEAREST)
132:   if ( WW.ne.0.0 ) then
133:     IWK4(I) = MIN(NP+1,INT(WW+R(I)))
134:   else
135:     IWK4(I) = 0
136:   end if
137: C/#ELSE
138: *   P = WW - NP
139: *   IWK4(I) = NP + 1 + P - R(I)
140: C/#ENDIF
141:   NFISS = NFISS + IWK4(I)
142: 100 continue
143: else
144: do 110 I = 1, NCOLS
145:   IP = LSCOL(I)
146:   RP(I) = WWW(IP)*SNFPX(IGRP(I),IMAT(I))
147:
148:   WW = WGTF(IREG(I))
149:   if ( JWTIM.ne.0. ) WW = WW*WTIME(ITT(IP))
150:   if ( JRWVR.ne.0. ) WW = WW*DBNK(IP,KDBNK(1))
151:   WW = MAX(WW,WMIN)
152:
153:   WW = RP(I) /WW
154:
155:   NP = WW
156: C/#IF ROUND OFF(NEAREST)
157:   if ( WW.ne.0.0 ) then
158:     IWK4(I) = MIN(NP+1,INT(WW+R(I)))
159:   else
160:     IWK4(I) = 0
161:   end if
162: C/#ELSE
163: *   P = WW - NP
164: *   IWK4(I) = NP + 1 + P - R(I)
165: C/#ENDIF
166:   NFISS = NFISS + IWK4(I)
167: 110 continue
168: end if
169: C
170: C
171: C***** FIXED SOURCE PROBLEM *****
172: C   IWK5 : COLLIDING NEUTRON OF FISSION NEUTRONS
173: C   IWK7 : ENERGY GROUP
174: C   FISSION NEUTRON ----> PARTICLE BANK(XXX,YYY,.....)
175: C   ----> SEND TO FREE-FLIGHT STACK
176: C
177:   if ( JEIGN.eq.0 ) then
178: C   ... check generation cutoff
179: C
180: C   if ( NFISS.gt.0.and.MXPGEN.gt.0 ) then
181: C
182: C     NGCUT = 0
183: C     do 120 I = 1, NCOLS
184: C       IP = LSCOL(I)
185: C       if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IWK4(I).gt.0 )
186: C         & NGCUT = NGCUT + 1
187: C       continue
188: C 120
189: C
190: C   if ( NGCUT.gt.0 ) then
191: C     NCUT = 0
192: C     WWC = 0.0D0
193: C     do 130 I = 1, NCOLS
194: C       IP = LSCOL(I)
195: C       if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IWK4(I).gt.0 )

```

```

196:   &   then
197:     NCUT = NCUT + IWK4(I)
198:     WWC = WWC + RP(I)
199:     IWK4(I) = 0
200:   end if
201: 130 continue
202:   NCNTR(27) = NCNTR(27) + NCUT
203:   WCNTR(27) = WCNTR(27) + WWC
204:   NFISS = NFISS - NCUT
205: end if
206: end if
207: C
208:   if ( NFISS.le.0 ) then
209:     NFISB = 0
210:     return
211:   end if
212: C
213:   if ( NFISS.le.NDEAD ) then
214:     MA = 0
215:     do 140 I = 1, NCOLS
216:       MA = MAX(MA,IWK4(I))
217: 140 continue
218:     N = 0
219:     do 160 K = 1, MA
220:       do 150 I = 1, NCOLS
221:         if ( IWK4(I).ge.K ) then
222:           N = N + 1
223:           IWK5(N) = I
224:           RW(N) = WGTF(IREG(I))
225:         end if
226: 150 continue
227: 160 continue
228: C
229: C   ... apply WTIME (time dependent weighting factor)
230: C
231:   if ( JWTIM.ne.0 ) then
232:     do 170 J = 1, N
233:       RW(J) = RW(J)*WTIME(ITT(LSCOL(IWK5(J))))
234: 170 continue
235:   end if
236: C
237: C   ... WGTF is relative to weight on birth
238: C
239:   if ( JRWVR.ne.0 ) then
240:     do 180 J = 1, N
241:       RW(J) = RW(J)*DBNK(LSCOL(IWK5(J)),KDBNK(1))
242: 180 continue
243:   end if
244:   do 190 J = 1, N
245:     RW(J) = MAX(RW(J),WLLIM)
246: 190 continue
247: C
248:   else
249: C
250: C***** not all fission netrons can be stored in bank
251: C
252:     N = 0
253:     II = 0
254:     do 210 I = 1, NCOLS
255:       K = IWK4(I)
256:       NFISS = NFISS + K
257:       if ( NFISS.gt.NDEAD ) go to 220
258:       II = I
259:       do 200 II = 1, K
260:         N = N + 1

```

src/gmvp/fisgen.f

```

261:          IWK5(N) = I
262:          RW(N)   = WGTF(IREG(I))
263:      200      continue
264:      210      continue
265: C
266: C      ( program never comes here (because NFISS > NDEAD) )
267: C      go to 290
268: C
269: C      .... TOO MANY FISSION NEUTRON ....
270: C
271: C      220      NFISS = N
272: C
273: C      .. apply WTIME (time dependent weighting factor)
274: C
275: C      if ( JWTIM.ne.0 ) then
276: C      do 230 J = 1, N
277: C          RW(J) = RW(J)*WTIME(ITT(LSCOL(IWK5(J))))
278: C      230      continue
279: C      end if
280: C
281: C      ... WGTF is relative to weight on birth
282: C
283: C      if ( JRWVR.ne.0 ) then
284: C      do 240 J = 1, N
285: C          RW(J) = RW(J)*DBNK(LSCOL(IWK5(J)),KDBNK(1))
286: C      240      continue
287: C      end if
288: C      do 250 J = 1, N
289: C          RW(J) = MAX(RW(J),WLLIM)
290: C      250      continue
291: C
292: C      I2 = I1
293: C      I3 = 0
294: C      call RANU2( IRAND, R, NCOLS-I1, ICON )
295: C      *VOCL LOOP,NOVREC
296: C      do 260 I = I1 + 1, NCOLS
297: C          NCNTR(I2) = NCNTR(I2) + IWK4(I)
298: C          WCNTR(I2) = WCNTR(I2) + RP(I)
299: C          IP = LSCOL(I)
300: C          S1 = SNAPX( IGRP(I),IMAT(I) )
301: C          ST = SNFPX( IGRP(I),IMAT(I) ) + S1
302: C          IF(ST.GT.0.0) THEN
303: C              IR = S1 / ST + R(I-I1)
304: C          ELSE
305: C              IR = 1
306: C              S1 = 1.0
307: C          ENDIF
308: C      ..... IR=0/1 = FISSION/SCATTERING
309: C          WWW(IP) = WWW(IP) * ST
310: C          S1 = S2PPX(2,IGRP(I),IMAT(I))
311: C
312: C          IF(IR.GE.1) THEN
313: C          if ( R(I-I1).le.S1 ) then
314: C              I2 = I2 + 1
315: C              LSCOL(I2) = IP
316: C              IREG(I2) = IREG(I)
317: C              IMAT(I2) = IMAT(I)
318: C              IGRP(I2) = IGRP(I)
319: C              WWW(IP) = WWW(IP) /S1
320: C
321: C          ... parent neutron dies ...
322: C
323: C          else
324: C              I3 = I3 + 1
325: C              IWK6(I3) = IP
326: C
327: C          IWK8(I3) = IMAT(I)
328: C          WWW(IP) = WWW(IP)*S2DPX(1,IGRP(I),IMAT(I))
329: C      end if
330: C      260      continue
331: C
332: C          NCOLS = I2
333: C
334: C          if ( I3.gt.0 ) then
335: C              call RANU2( IRAND, R, 4*I3, ICON )
336: C              I32 = I3*2
337: C              I33 = I3*3
338: C              NN = NFFL(NZONE+1)
339: C
340: C          C..... Fission neutron generation which replaces parent neutron
341: C          *VOCL LOOP,NOVREC
342: C          do 270 I = 1, I3
343: C              IP = IWK6(I)
344: C          C/#IF ROUNDOFF(NEAREST)
345: C              LL = MIN(INT(NTGX*R(I))+1,NTGX)
346: C          C/#ELSE
347: C              LL = NTGX*R(I) + 1
348: C          C/#ENDIF
349: C              RR = R(I3+I) - FKAI(LL,IWK8(I))
350: C          C97/11/06 LL2 = 2 + RR
351: C          C/#IF ROUNDOFF(NEAREST)
352: C              LL2 = MIN(INT(2+RR),2)
353: C          C/#ELSE
354: C              LL2 = 2 + RR
355: C          C/#ENDIF
356: C              IGG(IP) = KKAI(LL2,LL,IWK8(I))
357: C
358: C          ... change weight on birth if necessary
359: C          if ( KDBNK(1).ne.0 ) then
360: C              DBNK(IP,KIBNK(1)) = WWW(IP)
361: C          end if
362: C          ... change time of birth if necessary
363: C          if ( KDBNK(2).ne.0 ) then
364: C              DBNK(IP,KIBNK(2)) = TTT(IP)
365: C          end if
366: C          ... increment particle generation count ...
367: C          if ( KIBNK(4).ne.0 ) then
368: C              IBNK(IP,KIBNK(4)) = IBNK(IP,KIBNK(4)) + 1
369: C          end if
370: C
371: C          if ( IGG(IP).le.NGROUP ) then
372: C              W = 1. - 2.*R(I32+I)
373: C              A = SQRT(1.-W*W)
374: C              PHI = 6.283185307*R(I33+I)
375: C              U = A*COS(PHI)
376: C              V = A*SIN(PHI)
377: C              S = 1./SQRT(U*U+V*V+W*W)
378: C              AAA(IP) = U*S
379: C              BBB(IP) = V*S
380: C              CCC(IP) = W*S
381: C          C----- SEND TO FLIGHT STACK -----
382: C              NN = NN + 1
383: C              LSFFL(NN) = IP
384: C              IZFFL(NN) = IZZ(IP)
385: C          C-----
386: C          else
387: C              NDEAD = NDEAD + 1
388: C              LSDED(NDEAD) = IP
389: C              NCNTR(8) = NCNTR(8) + 1
390: C              WCNTR(8) = WCNTR(8) + WWW(IP)

```


src/gmvp/fisgen.f

```

391:          end if
392: 270      continue
393: C
394:          N1      = NFFL(NZONE+1) + 1
395: *VOCL LOOP,SCALAR
396:          do 280 I = N1, NN
397:              NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
398: 280      continue
399:          NFFL(NZONE+1) = NN
400: C
401:          end if
402: C
403: 290      continue
404:          end if
405: C
406: C
407:          if ( NFISS.le.0 ) then
408:              NFISB = 0
409:              return
410:          end if
411: C
412:          W      = 0.0
413:          do 300 I = 1, NFISS
414:              W      = W + RW(I)
415: 300      continue
416:          NCNTR(2) = NCNTR(2) + NFISS
417:          WCNTR(2) = WCNTR(2) + W
418: C
419:          call RANU2( IRAND, R, 2*NFISS, ICON )
420: C
421: C      FISSION NEUTRON ENERGY ---> IWK7
422: C
423:          do 310 I = 1, NFISS
424:              IO      = IWK5(I)
425: C/#IF ROUNDOFF(NEAREST)
426:              LL      = MIN(INT(NTGX*R(I))+1,NTGX)
427: C/#ELSE
428:              *      LL      = NTGX*R(I) + 1
429: C/#ENDIF
430:              RR      = R(NFISS+I) - FKAI(LL,IMAT(IO))
431: c97/11/07 LL2      = 2 + RR
432: C/#IF ROUNDOFF(NEAREST)
433:              LL2      = MIN(INT(2+RR),2)
434: C/#ELSE
435:              *      LL2      = 2 + RR
436: C/#ENDIF
437:              IWK7(I) = KKAI(LL2,LL,IMAT(IO))
438: 310      continue
439: C
440:          NFISB = NFISS
441: C
442: C      ENERGY CUT OFF
443: C
444:          if ( NTGX.gt.NGROUP ) then
445:              NN      = 0
446: *VOCL LOOP,NOVREC
447:          do 320 I = 1, NFISS
448:              if ( IWK7(I).gt.NGROUP ) then
449:                  NCNTR(8) = NCNTR(8) + 1
450:                  WCNTR(8) = WCNTR(8) + RW(I)
451:              else
452:                  NN      = NN + 1
453:                  IWK5(NN) = IWK5(I)
454:                  IWK7(NN) = IWK7(I)
455:                  RW(NN) = RW(I)

```

```

456:          end if
457: 320      continue
458:          NFISB = NN
459:          if ( NFISB.eq.0 ) return
460:          end if
461: C
462:          call RANU2( IRAND, R, 2*NFISB, ICON )
463: C
464:          NN      = NFFL(NZONE+1)
465:          NDEAD = NDEAD - NFISB
466: *VOCL LOOP,NOVREC
467:          do 330 I = 1, NFISB
468:              IO      = IWK5(I)
469:              LO      = LSCOL(IO)
470:              L1      = LSDED(NDEAD+I)
471: C
472:              W      = 1. - 2.*R(I)
473:              A      = SQRT(1.-W*W)
474:              PHI     = 6.283185307*R(NFISB+I)
475:              U      = A*COS(PHI)
476:              V      = A*SIN(PHI)
477:              S      = 1./SQRT(U*U+V*V+W*W)
478:              AAA(L1) = U*S
479:              BBB(L1) = V*S
480:              CCC(L1) = W*S
481:              XXX(L1) = XXX(LO)
482:              YYY(L1) = YYY(LO)
483:              ZZZ(L1) = ZZZ(LO)
484:              WWW(L1) = RW(I)
485:              IGG(L1) = IWK7(I)
486:              IZZ(L1) = IZZ(LO)
487:              TTT(L1) = TTT(LO)
488:              KLSF(L1) = 0
489:              if ( JIMPT.ne.0 ) XIM(L1) = XIM(LO)
490:              if ( JLATT.ne.0 ) LEVL(L1) = LEVL(LO)
491:              if ( JTLLT.ne.0 ) IBREG(L1) = IBREG(LO)
492:              IBREG(L1) = IBREG(LO)
493:              if ( JTIME.ne.0 ) ITT(L1) = ITT(LO)
494: C
495: C---- SEND TO FREE-FLIGHT STACK
496: C
497:              LSFFL(NN+I) = L1
498:              IZFFL(NN+I) = IZZ(L1)
499: 330      continue
500: C
501:          if ( JLATT.ne.0 ) then
502:              do 350 K = 1, NEST
503: *VOCL LOOP,NOVREC
504:              do 340 I = 1, NFISB
505:                  LO      = LSCOL(IWK5(I))
506:                  L1      = LSDED(NDEAD+I)
507:                  LZZ(L1,K) = LZZ(LO,K)
508:                  LPOS(L1,K) = LPOS(LO,K)
509:                  if ( JHLAT.ne.0 ) LCRS(L1,K) = LCRS(LO,K)
510:                  if ( JTLLT.ne.0 ) IBSPC(L1,K) = IBSPC(LO,K)
511: 340          continue
512: 350          continue
513:              end if
514: C
515: C      ... copy or change optional bank parameters
516: C
517:              do 390 K = 1, KDBNK(0)
518:                  if ( K.eq.KDBNK(1) ) then
519:
520: C      ... birth weight is current weight !!!

```

src/gmvp/fisgen.f

```

521:
522: *VOCL LOOP,NOVREC
523:       do 360 I = 1, NFISB
524:         L1 = LSDED(NDEAD+I)
525:         DBNK(L1,KDBNK(1)) = WWW(L1)
526:       360 continue
527:
528: C       ... time of birth is just now !!!
529:
530:       else if ( K.eq.KDBNK(2) ) then
531: *VOCL LOOP,NOVREC
532:       do 370 I = 1, NN
533:         L1 = LSDED(NDEAD+I)
534:         DBNK(L1,KDBNK(2)) = TTT(L1)
535:       370 continue
536:
537: C       ... other optional bank parameters are only copy of those
538: C       of parents
539:
540:       else
541: *VOCL LOOP,NOVREC
542:       do 380 I = 1, NFISB
543:         L0 = LSCOL(IWK5(I))
544:         L1 = LSDED(NDEAD+I)
545:         DBNK(L1,K) = DBNK(L0,K)
546:       380 continue
547:       end if
548: 390 continue
549: C
550:       do 420 K = 1, KIBNK(0)
551: C
552: C       ... increment "particle generation" if necessary
553: C
554:       if ( K.eq.KIBNK(4) ) then
555: *VOCL LOOP,NOVREC
556:       do 400 I = 1, NFISB
557:         L0 = LSCOL(IWK5(I))
558:         L1 = LSDED(NDEAD+I)
559:         IBNK(L1,KIBNK(4)) = IBNK(L0,KIBNK(4)) + 1
560:       400 continue
561:       else
562: *VOCL LOOP,NOVREC
563:       do 410 I = 1, NFISB
564:         L0 = LSCOL(IWK5(I))
565:         L1 = LSDED(NDEAD+I)
566:         IBNK(L1,K) = IBNK(L0,K)
567:       410 continue
568:       end if
569: 420 continue
570: C
571: C---- SEND FISSION NEUTRON TO FREE-FLIGHT STACK
572: C
573: *VOCL LOOP,SCALAR
574:       do 430 I = NN + 1, NN + NFISB
575:         NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
576:       430 continue
577:         NFFL(NZONE+1) = NN + NFISB
578: C
579: C
580:       else
581: C
582: C
583: C***** EIGENVALUE PROBLEM *****
584: C       FISSION NEUTRON ----> FISSION BANK(XXXF,YYYF,ZZZF,IZZF)
585: C

```

```

586: C***** TAKE TALLY FOR KEFF
587: C
588:       do 440 I = 1, NCOLS
589:         if ( RP(I).gt.0. ) then
590:           WCNTR(14) = WCNTR(14) + RP(I)
591:           NCNTR(14) = NCNTR(14) + 1
592:         end if
593:         WCNTR(19) = WCNTR(19) + WWW(LSCOL(I))*
594:           & (1.-SNAPX(IGRP(I),IMAT(I)))
595:       440 continue
596: C
597:       if ( NFISS.le.0 ) return
598: C
599:       NCS = 1
600: C
601: C
602: C
603: 450 continue
604: C
605: C
606: C .... accept all fission neutron ....
607: C
608:       if ( NFISS.le.NBANK ) then
609:         NFISR = 0
610: C
611:         MA = 0
612:         do 460 I = NCS, NCOLS
613:           MA = MAX(MA,IWK4(I))
614:       460 continue
615:         N = 0
616:         do 480 K = 1, MA
617:           do 470 I = NCS, NCOLS
618:             if ( IWK4(I).ge.K ) then
619:               N = N + 1
620:               IWK5(N) = LSCOL(I)
621:             end if
622:           470 continue
623:         480 continue
624: C
625: C .... select fission neutron ....
626: C
627:       else
628:         NNN = 0
629:         N = 0
630:         do 500 I = NCS, NCOLS
631:           NNN = NNN + IWK4(I)
632:           ICS = I
633:           if ( NNN.gt.NBANK ) go to 510
634:           do 490 K = 1, IWK4(I)
635:             N = N + 1
636:             IWK5(N) = LSCOL(I)
637:           490 continue
638:         500 continue
639:         NFISR = 0
640:         go to 520
641: C
642: 510 continue
643:         NFISR = NFISS - N
644:         NFISS = N
645:         NCS = ICS
646:       520 continue
647:       end if
648: C
649: C
650:       if ( NFISB.ge.NFBANK ) then

```

src/gmvp/fisgen.f

```

651:      call RANU2( IRAND, R, NFISS, ICON )
652:      IPNF = 0
653:      *VOCL LOOP,NOVREC
654:      do 530 I = 1, NFISS
655:      C/#IF ROUNDOFF(NEAREST)
656:      NCN2 = NCNTR(2) + I
657:      IP = MIN(INT(NCN2*R(I))+1,NCN2)
658:      C/#ELSE
659:      *      IP = (NCNTR(2)+I)*R(i) + 1
660:      C/#ENDIF
661:      if ( IP.le.NFBANK ) then
662:      XXXF(IP) = XXX(IWK5(I))
663:      YYF(IP) = YY(IWK5(I))
664:      ZZZF(IP) = ZZZ(IWK5(I))
665:      IZZF(IP) = IZZ(IWK5(I))
666:      if ( JLATT.ne.0 ) then
667:      IPNF = IPNF + 1
668:      IWK6(IPNF) = IP
669:      IWK7(IPNF) = IWK5(I)
670:      LEVLF(IP) = LEVL(IWK5(I))
671:      CCCCC if ( JTLT.ne.0 ) IBRGF(IP) = IBREG(IWK5(I))
672:      IBRGF(IP) = IBREG(IWK5(I))
673:      end if
674:      C 88/10/24 %%%%%%%%%
675:      IF(JLATT.NE.0) THEN
676:      LEVLF(IP) = LEVL(IWK5(I))
677:      DO 1311 K=1,NEST
678:      LZZF(IP,K) = LZZ(IWK5(I),K)
679:      LPOSF(IP,K) = LPOS(IWK5(I),K)
680:      IF(JHLAT.NE.0)LCRSF(IP,K) = LCRS(IWK5(I),K)
681:      C1311 CONTINUE
682:      C      ENDIF
683:      C %%%%%%%%%
684:      end if
685:      530 continue
686:      C
687:      if ( JLATT.ne.0.and.IPNF.gt.0 ) then
688:      do 550 K = 1, NEST
689:      *VOCL LOOP,NOVREC
690:      do 540 I = 1, IPNF
691:      LZZF(IWK6(I),K) = LZZ(IWK7(I),K)
692:      LPOSF(IWK6(I),K) = LPOS(IWK7(I),K)
693:      if ( JHLAT.ne.0 ) LCRSF(IWK6(I),K) =
694:      & LCRS(IWK7(I),K)
695:      if ( JTLT.ne.0 ) IBSPF(IWK6(I),K) =
696:      & IBSPC(IWK7(I),K)
697:      540 continue
698:      550 continue
699:      end if
700:      C
701:      ... bank distance to lattice frame ...
702:      C if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
703:      C do 532 K = 0, NEST
704:      C do 531 I = 1, IPNF
705:      C DFBK(IWK6(I),KDFBK(4)+k) = DBNK(IWK7(I),KDBNK(4)+k)
706:      C 531 continue
707:      C 532 continue
708:      C end if
709:      C if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
710:      C do 535 K = 0, NEST-1
711:      C do 534 I = 1, IPNF
712:      C IFBK(IWK6(I),KIFBK(6)+k) = IBNK(IWK7(I),KIBNK(6)+k)
713:      C 534 continue
714:      C 535 continue
715:      C end if

```

```

716:      C
717:      C ... bank STG sphere position ...
718:      C
719:      C if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
720:      C KF = KDFBK(5)
721:      C KB = KDBNK(5)
722:      C do 570 K = 1, NEST
723:      C do 560 I = 1, IPNF
724:      C DFBK(IWK6(I),KF) = DBNK(IWK7(I),KB)
725:      C DFBK(IWK6(I),KF+1) = DBNK(IWK7(I),KB+1)
726:      C DFBK(IWK6(I),KF+2) = DBNK(IWK7(I),KB+2)
727:      C 560 continue
728:      C KF = KF + 3
729:      C KB = KB + 3
730:      C 570 continue
731:      C end if
732:      C
733:      C else
734:      C NTEMP = NFBANK - NFISB
735:      C NTEMP = MIN(NFISS,NTEMP)
736:      C do 580 I = 1, NTEMP
737:      C N = NFISB + I
738:      C XXXF(N) = XXX(IWK5(I))
739:      C YYF(N) = YY(IWK5(I))
740:      C ZZZF(N) = ZZZ(IWK5(I))
741:      C IZZF(N) = IZZ(IWK5(I))
742:      C if ( JLATT.ne.0 ) LEVLF(N) = LEVL(IWK5(I))
743:      C CCCC if ( JTLT.ne.0 ) IBRGF(N) = IBREG(IWK5(I))
744:      C IBRGF(N) = IBREG(IWK5(I))
745:      C 580 continue
746:      C if ( JLATT.ne.0 ) then
747:      C do 600 K = 1, NEST
748:      C *VOCL LOOP,NOVREC
749:      C do 590 I = 1, NTEMP
750:      C N = NFISB + I
751:      C LZZF(N,K) = LZZ(IWK5(I),K)
752:      C LPOSF(N,K) = LPOS(IWK5(I),K)
753:      C if ( JHLAT.ne.0 ) LCRSF(N,K) = LCRS(IWK5(I),K)
754:      C if ( JTLT.ne.0 ) IBSPF(N,K) = IBSPC(IWK5(I),K)
755:      C 590 continue
756:      C 600 continue
757:      C end if
758:      C
759:      C ... bank distance to lattice frame ...
760:      C if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
761:      C do 562 K = 0, NEST
762:      C do 561 I = 1, NTEMP
763:      C N = NFISB + I
764:      C DFBK(N,KDFBK(4)+k) = DBNK(IWK5(I),KDBNK(4)+k)
765:      C 561 continue
766:      C 562 continue
767:      C end if
768:      C if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
769:      C do 565 K = 0, NEST-1
770:      C do 564 I = 1, NTEMP
771:      C N = NFISB + I
772:      C IFBK(N,KIFBK(6)+k) = IBNK(IWK5(I),KIBNK(6)+k)
773:      C 564 continue
774:      C 565 continue
775:      C end if
776:      C
777:      C ... bank STG sphere position ...
778:      C
779:      C if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
780:      C KF = KDFBK(5)

```

src/gmvp/fisgen.f

```

781:      KB      = KDBNK(5)
782:      do 620 K = 1, NEST
783:        do 610 I = 1, NTEMP
784:          N      = NFISB + I
785:          DFBK(N,KF) = DBNK(IWK5(I),KB)
786:          DFBK(N,KF+1) = DBNK(IWK5(I),KB+1)
787:          DFBK(N,KF+2) = DBNK(IWK5(I),KB+2)
788:        610      continue
789:          KF      = KF + 3
790:          KB      = KB + 3
791:        620      continue
792:      end if
793: C
794:      NFISB = NFISB + NTEMP
795:      if ( NTEMP.lt.NFISS ) then
796:        call RANU2( IRAND, R, NFISS-NTEMP, ICON )
797:        IPNF = 0
798: *VOCL LOOP,NOVREC
799:        do 630 I = NTEMP + 1, NFISS
800:          IP = R(I-NTEMP)*(NFBANK-NTEMP+1) + 1
801:          if ( IP.le.NFBANK ) then
802:            XXXF(IP) = XXX(IWK5(I))
803:            YYF(IP) = YY(IWK5(I))
804:            ZZZF(IP) = ZZ(IWK5(I))
805:            IZZF(IP) = IZZ(IWK5(I))
806:            if ( JLATT.ne.0 ) then
807:              IPNF = IPNF + 1
808:              IWK6(IPNF) = IP
809:              IWK7(IPNF) = IWK5(I)
810:              LEVLF(IP) = LEVL(IWK5(I))
811: CCCC          if ( JTLT.ne.0 ) IBRGF(IP) = IBREG(IWK5(I))
812:              IBRGF(IP) = IBREG(IWK5(I))
813:            end if
814:          end if
815:        630      continue
816: C
817:        if ( JLATT.ne.0.and.IPNF.gt.0 ) then
818:          do 650 K = 1, NEST
819: *VOCL LOOP,NOVREC
820:            do 640 I = 1, IPNF
821:              LZZF(IWK6(I),K) = LZZ(IWK7(I),K)
822:              LPOSF(IWK6(I),K) = LPOS(IWK7(I),K)
823:              if ( JHLAT.ne.0 ) then
824:                LCRSF(IWK6(I),K) = LCRS(IWK7(I),K)
825:              end if
826:              if ( JTLT.ne.0 ) then
827:                IBSPF(IWK6(I),K) = IBSPC(IWK7(I),K)
828:              end if
829:            640      continue
830:          650      continue
831:        end if
832: C
833: C      ... bank distance to lattice frame ...
834: C      if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
835: C        do 592 K = 0, NEST
836: C          do 591 I = 1, IPNF
837: C            DFBK(IWK6(I),KDFBK(4)+k) =
838: C              & DBNK(IWK7(I),KDBNK(4)+k)
839: C          591      continue
840: C        592      continue
841: C      end if
842: C      if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
843: C        do 595 K = 0, NEST-1
844: C          do 594 I = 1, IPNF
845: C            IFBK(IWK6(I),KIFBK(6)+k) =
846: C              & IBNK(IWK7(I),KIBNK(6)+k)
847: C          594      continue
848: C        595      continue
849: C      end if
850: C
851: C      ... bank STG sphere position ...
852: C
853: C      if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
854: C        KF      = KDFBK(5)
855: C        KB      = KDBNK(5)
856: C        do 670 K = 1, NEST
857: C          do 660 I = 1, IPNF
858: C            DFBK(IWK6(I),KF) = DBNK(IWK7(I),KB)
859: C            DFBK(IWK6(I),KF+1) = DBNK(IWK7(I),KB+1)
860: C            DFBK(IWK6(I),KF+2) = DBNK(IWK7(I),KB+2)
861: C          660      continue
862: C            KF      = KF + 3
863: C            KB      = KB + 3
864: C          670      continue
865: C        end if
866: C      end if
867: C
868: C      end if
869: C
870: C      NCNTR(2) = NCNTR(2) + NFISS
871: C      WCNTR( 2) = WCNTR( 2) + W
872: C
873: C      if ( NFISR.gt.0 ) then
874: C        NFISS = NFISR
875: C        go to 450
876: C      end if
877: C
878: C
879: C      end if
880: C
881: C      return
882: C      end

```

src/gmvp/fisset.f

```

1:      subroutine FISSSET( IOW,  IFISB, WFFACT,
2:      &      IRAND, NFISB, NHIST, NHIST1, NHSUB, NBANK,
3:      &      NZONE, NFBANK, NFBNK0, NBATCH, NREG,  NEST,
4:      &      WGTf, KZREG,
5:      &      XSOC, NLENG,
6:      &      XXXF, YYF, ZZZF, IZZF,
7:      &      LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF, KLSFF,
8:      &      DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK,
9:      &      XXXF2, YYF2, ZZZF2, IZZF2,
10:     &      LEVLF2, LZZF2, LPOSF2, LCRSF2,
11:     &      IBRGF2, IBSPF2, KLSFF2,
12:     &      DFBK2, KDFBK2, MDFBK2, IFBK2, KIFBK2, MIFBK2,
13:     &      RWF )
14: C=<MVP=====
15: C PURPOSE:  selection of fission source before each batch of eigenvalue
16: C calculation mode.
17: C when a batch needs more than one sub-batches
18: C (NHSUB < NHIST ) make copies of fission sources.
19: C CALLED IN: SOURCE
20: C CALLS      : (none)
21: C-----
22: C i NHIST1      : current batch size (NHIST or less than it)
23: C o IFISB(NHIST1) : fission bank index selected for current batch.
24: C      If points XXXF2(*) ,YYF2(*) ,... when NHSUB < NHIST1
25: C      else points XXXF(*), YYF(*), ...
26: C o WFFACT : factor to normalize fission weight.
27: C w RWF(NHIST) : working array
28: C=====
29:      implicit real*8(A-H,O-Z)
30: C
31:      include 'INC/_FLAGS'
32: C
33:      integer IFISB(NHIST)
34:      real*8 WFFACT
35: C
36: C .... FISSION BANK .....
37: C
38:      real*8 XXXF(NFBNK0), YYF(NFBNK0), ZZZF(NFBNK0)
39:      integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
40:      &      LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST),
41:      &      KLSFF(NFBNK0)
42:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
43: C
44: C
45: C ... optional bank parameters
46: C
47:      real*8 DFBK(NFBNK0,*)
48:      integer KDFBK(0:MDFBK)
49:      integer IFBK(NFBNK0,*)
50:      integer KIFBK(0:MIFBK)
51: C
52: C ... fission weight copy
53: C
54:      real WGTf(NREG)
55: C
56: C ... fission bank copy
57: C
58:      real*8 XXXF2(NHIST), YYF2(NHIST), ZZZF2(NHIST)
59:      integer IZZF2(NHIST), LEVLF2(NHIST), LZZF2(NHIST,NEST),
60:      &      LPOSF2(NHIST,NEST), LCRSF2(NHIST,NEST),
61:      &      KLSFF2(NHIST)
62:      integer IBRGF2(NHIST), IBSPF2(NHIST,NEST)
63: C
64:      real*8 DFBK2(NHIST,*)
65:      integer KDFBK2(0:MDFBK2)

```

```

66:      integer IFBK2(NHIST,*)
67:      integer KIFBK2(0:MIFBK2)
68: C
69: C .... TALLY .....
70: C
71:      real*8 XSOC(NLENG)
72: C
73: C .... GEOMETRY ARRAY DATA
74: C
75:      integer KZREG(NZONE)
76: C
77: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 7*NBANK FOR USE
78: C      IN sef5n2 )
79: C
80:      real RWF(NHIST)
81: C
82: C ... local variable
83: C
84: C Declaring these variables used for weight normalization as
85: C single precision may the same results as former version
86: C in which FISSRC procedure is in SOURCE routine.
87: C But using double precision is better for precision of calculation,
88: C so these variables will be declared as double precision
89: C after testing phase.
90: C
91:      real WSUMF, WSUMF2
92: C
93: C-----
94: C
95:      if ( NBATCH.gt.0.and.NFISB.le.0 ) then
96:          write(IOW,7000)
97:          7000 format(/X,' !!! No fission source has been generated. ',
98:          &          ' STOP calculation !!'/X,
99:          &          ' Check values for "WGTf" to generate more fission',
100:         &          ' neutrons, or other problems in your input.')
101:         stop 888
102:     end if
103: C
104:     if ( NFISB.le.NHIST1 ) then
105:         do 100 I = 1, NFISB
106:             IFISB(I) = I
107:             continue
108:         if ( NFISB.lt.NHIST1 ) then
109:             call RANU2( IRAND, RWF, NHIST1-NFISB, ICON )
110:             do 110 I = 1, NHIST1 - NFISB
111: C/#IF ROUNDOff(NEAREST)
112:                 IP = MIN(INT(NFISB*RWF(I))+1,NFISB)
113: C/#ELSE
114:                 IP = NFISB*RWF(I) + 1
115: C/#ENDIF
116:                 IFISB(NFISB+I) = IP
117:             110 continue
118:         end if
119:     else
120:         call RANU2( IRAND, RWF, NHIST1, ICON )
121:         do 120 I = 1, NHIST1
122: C/#IF ROUNDOff(NEAREST)
123:             IP = MIN(INT(NFISB*RWF(I))+1,NFISB)
124: C/#ELSE
125:             IP = NFISB*RWF(I) + 1
126: C/#ENDIF
127:             IFISB(I) = IP
128:         120 continue
129:         end if
130: C

```

src/gmvp/fisset.f

```
131: C      .... Weight normalization factor ...
132: C
133:      WSUMF   = 0.0
134: *VOCL LOOP,NOVREC
135:      do 130 J = 1, NHIST1
136:          WSUMF   = WSUMF + WGTf(IBRGf(IFISB(J)))
137:      130 continue
138: C
139:      WSUMF2   = DBLE(NHIST1)
140:      WFFACT   = WSUMF2/WSUMF
141: C
142: C ... Update sum of weights .....
143: C
144:      XSOC(NBATCH+1) = 0.0D0
145: C
146: C ... copy selected fission bank data
147: C
148:      if ( NHSUB.lt.NHIST1 ) then
149:          do 140 I=1,NHIST1
150:              XXXF2(I) = XXXF(IFISB(I))
151:              YYF2(I) = YYF(IFISB(I))
152:              ZZZF2(I) = ZZZF(IFISB(I))
153:              IZZF2(I) = IZZF(IFISB(I))
154:              KLSFF2(I) = KLSFF(IFISB(I))
155:              IBRGF2(I) = IBRGF(IFISB(I))
156:              if ( JLATT.ne.0 ) then
157:                  LEVLF2(I) = LEVLF(IFISB(I))
158:              end if
159:          140 continue
160: C
161:          if ( JLATT.ne.0 ) then
162:              do 160 K=1,NEST
163:                  do 150 I=1,NHIST1
164:                      LZZF2(I,K) = LZZF(IFISB(I),K)
165:                      LPOSF2(I,K) = LPOSF(IFISB(I),K)
166:                      if ( JHLAT.ne.0 ) then
167:                          LCRSF2(I,K) = LCRSF(IFISB(I),K)
168:                      end if
169:                      if ( JTLT.ne.0 ) then
170:                          IBSPF2(I,K) = IBSPF(IFISB(I),K)
171:                      end if
172:                  150 continue
173:              160 continue
174:          end if
175:          do 180 K=1,KDFBK(0)
176:              do 170 I=1,NHIST1
177:                  DFBK2(I,K) = DFBK(IFISB(I),K)
178:              170 continue
179:          180 continue
180:          do 200 K=1,KIFBK(0)
181:              do 190 I=1,NHIST1
182:                  IFBK2(I,K) = IFBK(IFISB(I),K)
183:              190 continue
184:          200 continue
185:          do 210 I=1,NHIST1
186:              IFISB(I) = I
187:          210 continue
188:      end if
189: C
190: C ... clear fission bank entry number for current batch ...
191: C
192:      NFISB   = 0
193: C
194:      return
195:      end
```

src/gmvp/fissrc.f

```

1:      subroutine FISSRC( IOW, IRAND, NFISB, NGENE, NFB0,WFFACT,IFISB,
2:      &      NFBNK0, IBP,
3:      &      NHIST, NBANK, NZONE, NFBANK,
4:      &      NBATCH,NGROUP,NGP1, NREG, NEST,
5:      &      NLENG, WSUM, KZREG, KZMAT, LTYP, WGTF,
6:      &      ENGYB, XIMP, XSOC, NCNTR, WCNTR, NEVENT,
7:      &      NMAT, NTGX, KKAI, FKAI,
8:      &      NFFL, LSFFL, IZFFL, NDEAD, LSDED,
9:      &      XXX, YYY, ZZZ, AAA, BBB, CCC,
10:     &      WWW, IGG, IZZ, TTT, XIM, IBREG, IBSPC, KLSF,
11:     &      LEVL, LZZ, LPOS, LCRS,
12:     &      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
13:     &      XXXF, YYYF, ZZZF, IZZF,
14:     &      LEVLF, LZZF, LPOSF,
15:     &      LCRSF, IBRGF, IBSPF, KLSFF,
16:     &      DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
17:     &      R, IWK2, RWK1 )
18: C=<GMVP>=====
19: C PURPOSE: Set energy and other parameters necessary to start
20: C random walk for fission source of eigenvalue problem.
21: C this routine may be called more than once for a batch (probably
22: C before each sub-batch).
23: C
24: C (formerly working as "Preparation of fission source before
25: C each batch")
26: C
27: C Selection of fission sources from fission neutron bank is moved
28: C to FISSET routine wich should be called at the beginnig of each
29: C batch .
30: C
31: C CALLED IN: SOURCE
32: C CALLS :
33: C -----
34: C arguments:
35: C i NGENE : number of fission sources set in this routine.
36: C
37: C i IFISB(*) : index pointer to fission neutron bank selected for
38: C fission source of current batch.
39: C More than one elemnts of this array may be pointing
40: C the same fission bank elements.(take care for vectorization)
41: C
42: C i NFB0 : use fission bank of indecies IFISB(NFB0+1:NFB0+NGENE)
43: C
44: C i WFFACT : normalization factor of fission source weight.
45: C i NFBNK0: fission bank array length
46: C i XXXF,YYYF,ZZZF,IZZF,LEVLF,LZZF,LPOSF,LCRSF,IBRGF,IBSPF,
47: C KLSFF, DFBK,KDFBK,IFBK,KIFBK :
48: C Fission neutron bank arrays.
49: C
50: C <<Caution>>
51: C
52: C When NHSUB < NHIST, which means more than one sub-batch are
53: C necessary for a batch, another set of fission bank arrays
54: C XXXF2, YYYF2, .... are passed as XXXF, YYYF, ... and
55: C array size NFBNK0 is not the global data for XXXF, YYYF,...
56: C but that for XXXF2, YYYF2 (must be same as NHIST).
57: C
58: C o IBP(*) : bank pointer of generated source.
59: C o WSUM : source weight sum
60: C o XSOC : source weight sum of current batch id XSOC(NBATCH+1)
61: C o NCNTR(1,1), WCNTR(1,1) : total number and weight of fission source
62: C
63: C o NFFL, IZFFL, LSFFL : free flight particle stack.
64: C -----
65: C UPDATE:

```

```

66: C=====
67:      implicit real*8(A-H,O-Q,S-Z)
68:      implicit real(R)
69: C
70:      include 'INC/_FLAGS'
71: C
72:      real*8 WFFACT
73: C
74:      integer IFISB(NHIST)
75: C
76: C .... bank pointers returned ...
77: C
78:      integer IBP(NGENE)
79: C
80: C .... PARTICLE BANK .....
81: C
82:      real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
83:      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
84:      real WWW(NBANK), XIM(NBANK)
85:      real*8 TTT(NBANK)
86:      real WGTF(NREG)
87:      integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
88:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
89:      integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
90: C
91: C ... optional bank parameters
92: C
93:      real*8 DBNK(NBANK,*)
94:      integer KDBNK(0:MDBNK)
95:      integer IBNK(NBANK,*)
96:      integer KIBNK(0:MIBNK)
97: C
98: C .... FISSION BANK .....
99: C
100:      real*8 XXXF(NFBNK0), YYYF(NFBNK0), ZZZF(NFBNK0)
101:      integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
102:      &      LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST),
103:      &      KLSFF(NFBNK0)
104:      integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
105: C
106: C ... optional bank parameters
107: C
108:      real*8 DFBK(NFBNK0,*)
109:      integer KDFBK(0:MDFBK)
110:      integer IFBK(NFBNK0,*)
111:      integer KIFBK(0:MIFBK)
112: C
113: C .... TALLY .....
114: C
115:      real*8 WSUM, XSOC(NLENG), WCNTR(NEVENT,2), NCNTR(NEVENT,2)
116: C
117: C .... VARIANCE REDUCTION DATA .....
118: C
119:      real XIMP(NGROUP,NREG)
120: C
121: C .... STACKS ...(FREE FLIGHT & MORGUE) .....
122: C
123:      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSDED(NBANK)
124: C
125: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 7*NBANK FOR USE
126: C IN sef5n2 )
127: C
128:      real R(4*NBANK), RWK1(NBANK)
129:      integer IWK2(NBANK)
130: C

```

src/gmvp/fissrc.f

```

131: C .... ENERGY BOUNDARY .....
132: C
133:       real ENGYB(NGP1+1)
134: C
135: C .... GEOMETRY ARRAY DATA
136: C
137:       integer KZREG(NZONE)
138:       integer KZMAT(NZONE,2)
139:       integer LTyp(*)
140: C
141: C CROSS SECTION ..... (NEUTRON)
142: C
143:       integer KKAI(2,NTGX,NMAT)
144:       real FKAI(NTGX,NMAT)
145: C
146:       include '../shared/INC/_PMLATT'
147:       include '../shared/INC/_SPLATT'
148: C
149: C-----
150: C .... SOURCE IS TAKEN FROM FISSION BANK (JEIGN.EQ 1.AND.NBATCH>0)
151: C-----
152: C
153:       do 100 I=1,NGENE
154:           IWK2(I) = IFISB(NFB0+I)
155:       100 continue
156: C
157: C .....
158: C
159:       call RANU2( IRAND, R, 4*NGENE, ICON )
160: C
161:       IPP      = NDEAD - NGENE
162:       NGEN2    = NGENE*2
163:       NGEN3    = NGENE*3
164:       WSUMF    = 0.0
165: *VOCL LOOP,NOVREC
166:       do 130 I = 1, NGENE
167:           IBP(I) = LSDED(IPP+I)
168: C
169:           XXX(IBP(I)) = XXXF(IWK2(I))
170:           YYY(IBP(I)) = YYF(IWK2(I))
171:           ZZZ(IBP(I)) = ZZZF(IWK2(I))
172:           D          = 2.0*R(I) - 1.0
173:           AAA(IBP(I)) = D
174:           D          = SQRT(1.0-D**2)
175:           D2         = 6.283185307179586D0*R(NGENE+I)
176:           BBB(IBP(I)) = COS(D2)*D
177:           CCC(IBP(I)) = SIN(D2)*D
178:           IZ         = IZZF(IWK2(I))
179:           IZZ(IBP(I)) = IZ
180: C
181:           KLSF(IBP(I)) = KLSFF(IWK2(I))
182: C
183:           IREG      = KZREG(IZ)
184: C
185: C ... IBREG is always set (from Jan 2000)
186:           IBREG(IBP(I)) = IREG
187:           if ( JTLLT.eq.0 ) then
188:               RWK1(I) = WGTF(IREG)
189:           else
190:               IBREG(IBP(I)) = IBRGF(IWK2(I))
191:               RWK1(I) = WGTF(IBREG(IBP(I)))
192:           end if
193: C
194:           WSUMF      = WSUMF + RWK1(I)
195: C

```

```

196:           if ( JTIME.ne.0 ) TTT(IBP(I)) = 0.0D0
197: C
198: C
199:           IMAT      = KZMAT(IZZ(IBP(I)),1)
200: C
201: C/#IF ROUNDOFF(NEAREST)
202:           KK        = MIN(INT(NTGX*R(NGEN2+I))+1,NTGX)
203:           KKK        = 2*KK
204:           KKK        = MIN(KKK,INT(KKK+R(NGEN3+I)-FKAI(KK,IMAT)))
205: C/#ELSE
206:           KK        = NTGX*R(NGEN2+I) + 1
207:           KKK        = KK*2 + R(NGEN3+I) - FKAI(KK,IMAT)
208: C/#ENDIF
209: C
210:           IGG(IBP(I)) = KKAI(KKK,1,IMAT)
211: C
212:       130 continue
213: C
214: C
215: C ..... WEIGHT NORMALIZATION ...
216: C
217:       WSUMF2 = FLOAT(NGENE)
218:       RATIO  = WSUMF2/WSUMF
219: *VOCL LOOP,NOVREC
220:       do 140 I = 1, NGENE
221:           WWW(IBP(I)) = RWK1(I)*WFFACT
222:       140 continue
223: C
224: C ..... LATTICE PARAMETER BANK .....
225: C
226:       if ( JLATT.ne.0 ) then
227: *VOCL LOOP,NOVREC
228:           do 150 I = 1, NGENE
229:               LEVL(IBP(I)) = LEVLF(IWK2(I))
230:               if ( JTLLT.ne.0 ) IBSPC(IBP(I),0) = 0
231:           150 continue
232: C
233:           do 170 K = 1, NEST
234: *VOCL LOOP,NOVREC
235:               do 160 I = 1, NGENE
236:                   LZZ(IBP(I),K) = LZZF(IWK2(I),K)
237:                   LPOS(IBP(I),K) = LPOSF(IWK2(I),K)
238:                   if ( JHLAT.ne.0 ) LCRS(IBP(I),K) = LCRSF(IWK2(I),K)
239:                   if ( JTLLT.ne.0 ) IBSPC(IBP(I),K) = IBSPF(IWK2(I),K)
240:               160 continue
241:           170 continue
242:       end if
243: C
244: C ... retrieve distance to lattice frame ...
245: C
246:       if ( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
247:           KF      = KDFBK(4)
248:           KB      = KDBNK(4)
249:           do 190 K = 0, NEST
250:               do 180 I = 1, NGENE
251:                   DBNK(IBP(I),KB+K) = DFBK(IWK2(I),KF+K)
252:               180 continue
253:           190 continue
254:       end if
255: C
256:       if ( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
257:           KF      = KIFBK(6)
258:           KB      = KIBNK(6)
259:           do 210 K = 0, NEST - 1
260:               do 200 I = 1, NGENE

```


src/gmvp/fissrc.f

```

261: C          IBNK(IBP(I),KB+K) = IFBK(IWK2(I),KF+K)
262: C 200      continue
263: C 210      continue
264: C      end if
265: C
266: C ... retrieve STG sphere position ...
267: C
268: C      if ( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
269: C          KF = KDFBK(5)
270: C          KB = KDBNK(5)
271: C          do 230 K = 1, NEST
272: C              do 220 I = 1, NGENE
273: C                  DBNK(IBP(I),KB) = DFBK(IWK2(I),KF)
274: C                  DBNK(IBP(I),KB+1) = DFBK(IWK2(I),KF+1)
275: C                  DBNK(IBP(I),KB+2) = DFBK(IWK2(I),KF+2)
276: C 220          continue
277: C              KF = KF + 3
278: C              KB = KB + 3
279: C 230          continue
280: C          end if
281: C
282: C .... STG placement flag (which type of NND should be used)
283: C
284: C      Base material (matrix) of STG region needs NND type2 sampling
285: C
286: C      if ( JSTGR.ne.0 ) then
287: C          do 240 I = 1, NGENE
288: C              INND = 0
289: C              LVL = LEVL(IBP(I))
290: C              if ( LVL.gt.0 ) then
291: C                  if ( LTYP(LATTNM(KZMAT(LZZ(IBP(I),LVL),1))).eq.10 ) then
292: C                      if ( LPOS(IBP(I),LVL).eq.0 ) INND = 2
293: C                  end if
294: C              end if
295: C              IBNK(IBP(I),KIBNK(7)) = INND
296: C 240          continue
297: C          end if
298: C
299: C ... clear flight count in flight path
300: C
301: C      if ( JFISX.ne.0 ) then
302: C          do 250 I = 1, NGENE
303: C              IBNK(IBP(I),KIBNK(5)) = 0
304: C 250          continue
305: C          end if
306: C
307: C
308: C .... MAKE FREE FLIGHT STACK .....
309: C
310: C      do 290 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
311: C          LSFFL(I) = IBP(I-NFFL(NZONE+1))
312: C          IZFFL(I) = IZZ(LSFFL(I))
313: C          if ( JIMPT.ne.0 ) then
314: C              if ( JTLLT.eq.0 ) then
315: C                  IREG = KZREG(IZFFL(I))
316: C              else
317: C                  IREG = IBREG(LSFFL(I))
318: C              end if
319: C              XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),IREG)
320: C          end if
321: C      290 continue
322: C *VOCL LOOP, SCALAR
323: C      do 300 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
324: C          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
325: C      300 continue

```

```

326:          NFFL(NZONE+1) = NFFL(NZONE+1) + NGENE
327: C
328: C .... Update sum of weights .....
329: C
330: C          WSUMF2 = DBLE(NGENE)
331: C
332: C          WSUM = WSUM + WSUMF2
333: C          XSOC(NBATCH+1) = XSOC(NBATCH+1) + WSUMF2
334: C          NCNTR(1,1) = NCNTR(1,1) + NGENE
335: C          WCNTR(1,1) = WCNTR(1,1) + WSUMF2
336: C
337: C          return
338: C          end

```

src/gmvp/flagin.f

```
1:      subroutine FLAGIN
2: C=====
3: C purpose: initial setting of option flags (GMVP)
4: C called in: main
5: C=====
6:      include 'INC/_FLAGS'
7:      include 'INC/_KPIDS'
8: C
9: C      DATA
10: C & JREST/0/, JNEUT/1/, JPHOT/0/, JIMAG/0/, JTIME/0/, JDELT/0/,
11: C & JFISS/0/, JEIGN/0/, JFIXD/1/, JDDX /1/, JADJM/0/,
12: C & JRRLT/1/, JIMPT/0/, JWWND/0/, JPSTR/0/,
13: C & JFCOL/0/, JBIAS/0/, JRESP/0/, JMNTR/0/, JVMNT/0/, JDEBG/0/,
14: C & JALLZ/0/, JONEZ/1/, JPIC/0/, JLATT/0/, JTLLT/0/, JFPRT/0/,
15: C & JSTATX/0/, JPSTD/0/, JREPR /1/, JRUNM /0/
16: C      DATA JPRTS / 1,19*0 /
17: C
18:      JREST = 0
19:      JARST = 0
20:
21:      do 130 I = 1, MKPAR
22:          JKPARI(I) = 0
23: 130 continue
24:      JKPARI(KPNEUT) = 1
25:      JNEUT = 1
26:      JPHOT = 0
27:
28:      JIMAG = 0
29:      JTIME = 0
30:      JDELT = 0
31:
32:      JFISS = 0
33:      JEIGN = 0
34:      JFIXD = 1
35:      JDDX = 1
36:      JADJM = 0
37:
38:      JRRLT = 1
39:      JIMPT = 0
40:      JWWND = 0
41:      JPSTR = 0
42:      JTIM = 0
43:
44:      JFCOL = 0
45:      JNCOL = 0
46:      JBIAS = 0
47:      JRWVR = 0
48:
49:      JRESP = 0
50:      JMNTR = 0
51:      JVMNT = 0
52:      JOSRC = 0
53:      JRSTF = 1
54:      JSCRT = 1
55:
56:      JALLZ = 0
57:      JONEZ = 1
58:      JPIC = 0
59:      JLATT = 0
60:      JTLLT = 0
61:      JFPRT = 0
62:
63:      JSTATX = 0
64:      JPSTD = 0
65:      JREPR = 1
66:
67:      JTSKR = 1
68:      JRUNM = 0
69:      JUNDG = 0
70:
71:      JFISX = 0
72:      JSTGR = 0
73:
74:      JANLF = 0
75:      JPTIM = 0
76:
77:      JTSRF = 0
78:
79:      JPRTS(1) = 1
80:      do 100 I = 2, 20
81:          JPRTS(I) = 0
82: 100 continue
83:
84:      do 110 I = 1, MAXJDB
85:          JDEBG(I) = 0
86: 110 continue
87:
88:      do 120 I = 1, MXJVP
89:          JVPPI(I) = 0
90: 120 continue
91:
92:      JSTPRN(1) = 1
93: C/#IF PARA(SX* CRAY*)
94:      JSTPRN(1) = 2
95: C/#ENDIF
96:      JSTPRN(2) = 2
97:
98:      return
99:      end
```

src/gmvp/flight.f

```

1: C/IF ARGSAVE
2: * SUBROUTINE FLIGH0( IOW,
3: * N NBANK, NZONE, NSDA, NZDA, NGROUP, NTGX, NREG,
4: * N NCOLS, NDEAD, NLOST, IRAND, DINF,
5: * N NEST, NLATT, NLBZ,
6: * B XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
7: * B WWW ,IGG ,TTT ,LEVL ,LZZ ,LPOS ,
8: * B LCRS ,IBREG,
9: * S LSFFL ,NFFL ,IZFFL ,LSCOL ,LSSRC ,NNXT ,
10: * S IZNXT ,LSDED ,STOTX ,SNUFX ,SNAPX ,
11: * G SDA ,KZMAT ,KZREG ,KZDA ,KZAA ,NKZAA ,
12: * G DALT ,KDALT ,NVLAT ,SZLAT ,IPLAT ,KSLAT ,
13: * G LTYP ,MLBZZ ,KLATT ,CELSZ ,
14: * T FLTR ,
15: * T NLEAK ,WLEAK ,NCNTR ,WCNTR ,
16: * W KKSF ,JKSF ,IPZONE,
17: * W X ,Y ,Z ,AI ,BI ,CI ,
18: * W W ,DI ,IGI ,IBP ,R ,D11 ,
19: * W D22 ,DLOC1 ,DLOC2 ,LLS ,IZT ,DSDA0 ,
20: * W DSDA1 ,DSDA2 ,DSDA3 ,DSDA4 ,DSDA5 ,DSDA6 ,
21: * W DSDA7 ,DSDA8 ,DSDA9 )
22: C/ELSE
23: SUBROUTINE FLIGHT( IOW,
24: N NBANK, NZONE, NSDA, NZDA, NGROUP, NTGX, NREG,
25: N NCOLS, NDEAD, NLOST, IRAND, DINF,
26: N NEST, NLATT, NLBZ,
27: B XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
28: B WWW ,IGG ,TTT ,LEVL ,LZZ ,LPOS ,
29: B LCRS ,IBREG,
30: S LSFFL ,NFFL ,IZFFL ,LSCOL ,LSSRC ,NNXT ,
31: S IZNXT ,LSDED ,STOTX ,SNUFX ,SNAPX ,
32: G SDA ,KZMAT ,KZREG ,KZDA ,KZAA ,NKZAA ,
33: G DALT ,KDALT ,NVLAT ,SZLAT ,IPLAT ,KSLAT ,
34: G LTYP ,MLBZZ ,KLATT ,CELSZ ,
35: T FLTR ,
36: T NLEAK ,WLEAK ,NCNTR ,WCNTR ,
37: W KKSF ,JKSF ,IPZONE,
38: W X ,Y ,Z ,AI ,BI ,CI ,
39: W W ,DI ,IGI ,IBP ,R ,D11 ,
40: W D22 ,DLOC1 ,DLOC2 ,LLS ,IZT ,DSDA0 ,
41: W DSDA1 ,DSDA2 ,DSDA3 ,DSDA4 ,DSDA5 ,DSDA6 ,
42: W DSDA7 ,DSDA8 ,DSDA9 )
43: C/ENDIF
44: C===== < FLIGHT > =====
45: C PURPOSE: FREE-FLIGHT (ALL ZONE LOGIC )
46: C CALLED IN: ACTION
47: C CALLS: RANU2,VMNTR1
48: C=====
49: implicit real*8 (D)
50: include 'INC/_FLAGS'
51: include '../shared/INC/_TASKDT'
52: C
53: C .... PARTICLE BANK .....
54: C
55: real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
56: real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
57: real WWW(NBANK)
58: real*8 TTT(NBANK)
59: integer IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
60: & LPOS(NBANK,NEST), LCRS(NBANK,NEST), IBREG(NBANK)
61: C
62: C .... STACKS .....
63: C
64: integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSCOL(NBANK),
65: & LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK)

```

```

66: C
67: C .... GEOMETRY .....
68: C
69: real*8 SDA(NSDA), DALT(1), SZLAT(8,NLATT), CELSZ(6,1)
70: integer KZMAT(NZONE), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
71: & NKZAA(NZONE), KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT),
72: & IPLAT(1), KSLAT(1), KLATT(1), NVLAT(4,NLATT)
73: C
74: C .... CROSS SECTION .....
75: C
76: real STOTX(NTGX,1), SNUFX(NTGX,1), SNAPX(NTGX,1)
77: C
78: C .... TALLY .....
79: C
80: ccccc REAL RESP(NGROUP,NRESP)
81: ccccc REAL*8 FLTR(NGROUP,NREG), RETR(NREG,NRESP),
82: real*8 FLTR(NGROUP,NREG)
83: real*8 NLEAK(NGROUP,NREG), WLEAK(NGROUP,NREG)
84: real*8 NCNTR(*), WCNTR(*)
85: C
86: C .... WORKING AREA .....
87: C
88: real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
89: & CI(NBANK), W(NBANK), DI(NBANK), DSDA0(NBANK),
90: & DSDA1(NBANK), DSDA2(NBANK), DSDA3(NBANK), DSDA4(NBANK),
91: & DSDA5(NBANK), DSDA6(NBANK), DSDA7(NBANK), DSDA8(NBANK),
92: & DSDA9(NBANK), D11(NBANK), D22(NBANK), DLOC1(NBANK),
93: & DLOC2(NBANK)
94: real R(NBANK)
95: integer IGI(NBANK), IBP(NBANK), LLS(NBANK), KKSF(NZONE+1),
96: & IZT(NBANK), JKSF(NZONE+1), IPZONE(NZONE+1)
97: C
98: cccc COMMON /CPU2/ TF(10),ANZK,ANV,MCALL
99: C
100: C/IF ARGSAVE
101: * RETURN
102: C -----
103: * ENTRY FLIGHT
104: C/ENDIF
105: C
106: if ( JVMNT.ne.0 ) call VMNTR1( 3, NFFL(NZONE+1) )
107: C
108: C .... CHECK LEAKAGE ..... MATERIAL # = -1 --> OUTER VOID !
109: C NCNTR(7),WCNTR(7) : LEAKAGE MONITOR
110: C
111: NZ1 = NZONE + 1
112: C
113: C-----
114: C .... GATHER VECTORS .....
115: C-----
116: C
117: 100 continue
118: NZK = 0
119: do 110 K = 1, NZONE
120: if ( NFFL(K).ne.0 ) then
121: NZK = NZK + 1
122: JKSF(NZK) = K
123: end if
124: 110 continue
125: C
126: c ... treat many zones ...
127: c
128: if ( NZK.gt.20 ) then
129: IPZONE(1) = 1
130: do 120 NK = 1, NZK

```

src/gmvp/flight.f

```

131:      IPZONE(NK+1) = IPZONE(NK) + NFFL(JKSF(NK))
132:      120 continue
133:      *VOCL LOOP,NOVREC
134:      do 130 NK = 1, NZK
135:          KKSF(JKSF(NK)) = IPZONE(NK) - 1
136:      130 continue
137:      ***VOCL LOOP,NOVREC(IBP,IZT)
138:      c      DO 130 I=1,NFFL(NZ1)
139:      c          KKSF(IZFFL(I)) = KKSF(IZFFL(I)) + 1
140:      c          KKK = KKSF(IZFFL(I))
141:      c          IBP(KKK) = LSFFL(I)
142:      c          IZT(KKK) = IZFFL(I)
143:      c 130 CONTINUE
144:      c
145:      c      do 140 I = 1, NFFL(NZ1)
146:      c          KKSF(IZFFL(I)) = KKSF(IZFFL(I)) + 1
147:      c          IGI(I) = KKSF(IZFFL(I))
148:      c 140 CONTINUE
149:      *VOCL LOOP,NOVREC
150:      do 150 I = 1, NFFL(NZ1)
151:          IBP(IGI(I)) = LSFFL(I)
152:          IZT(IGI(I)) = IZFFL(I)
153:      150 CONTINUE
154:      c
155:      c      ... not so many zone ...
156:      c      else
157:      c          IPZONE(1) = 1
158:      c          do 160 NK = 1, NZK
159:      c              IPZONE(NK+1) = IPZONE(NK) + NFFL(JKSF(NK))
160:      c 160 CONTINUE
161:      c          KKK = 0
162:      c          do 180 NK = 1, NZK
163:      c              do 170 I = 1, NFFL(NZ1)
164:      c                  if ( IZFFL(I).eq.JKSF(NK) ) then
165:      c                      KKK = KKK + 1
166:      c                      IBP(KKK) = LSFFL(I)
167:      c                      IZT(KKK) = IZFFL(I)
168:      c                  end if
169:      c              CONTINUE
170:      c 180 CONTINUE
171:      c      end if
172:      c
173:      c      do 190 I = 1, NFFL(NZ1)
174:      c          LSFFL(I) = IBP(I)
175:      c          IZFFL(I) = IZT(I)
176:      c          X(I) = XXX(LSFFL(I))
177:      c          Y(I) = YYY(LSFFL(I))
178:      c          Z(I) = ZZZ(LSFFL(I))
179:      c          AI(I) = AAA(LSFFL(I))
180:      c          BI(I) = BBB(LSFFL(I))
181:      c          CI(I) = CCC(LSFFL(I))
182:      c          W(I) = WWW(LSFFL(I))
183:      c          IGI(I) = IGG(LSFFL(I))
184:      c          DI(I) = DINF
185:      c          DLOC1(I) = -DINF
186:      c          DLOC2(I) = DINF
187:      c 190 CONTINUE
188:      c
189:      c-----
190:      c      .... CALCULATE MAXIMUM SURFACE NUMBER OF ZONES .....
191:      c-----
192:      c      MSF = 0
193:      c      do 200 NK = 1, NZK
194:      c          K = JKSF(NK)
195:      c          MSF = MAX(NKZAA(K),MSF)

```

```

196:      200 CONTINUE
197:      c
198:      c-----
199:      c-----
200:      c      .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
201:      c-----
202:      c-----
203:      c
204:      c      do 370 N = 1, MSF
205:      c-----
206:      c      .... GATHER DATA OF SURFACES .....
207:      c-----
208:      c      do 250 NK = 1, NZK
209:      c          KKSF(NK) = 0
210:      c          K = JKSF(NK)
211:      c          KK = NKZAA(K)
212:      c          if ( KK.ge.N ) then
213:      c              LS = KZDA(1,KZAA(K)+N-1)
214:      c 210 ..... 0/1/2 = SIMPLE / ONE OF A BODY / LAST SURFACE OF A BODY
215:      c              LLG = KZDA(2,KZAA(K)+N-1)
216:      c 220 ..... SIGN: (+) .... INSIDE OF A SURFACE (-) ... OUTSIDE
217:      c              LSG = SIGN(1,LS)
218:      c              LSP = ABS(LS)
219:      c              KSF = INT(ABS(SDA(LSP)))
220:      c          KKSF(NK) = KSF
221:      c          if ( KSF.eq.1 ) then
222:      c              do 210 I = IPZONE(NK), IPZONE(NK+1) - 1
223:      c                  LLS(I) = LSG
224:      c                  IZT(I) = LLG
225:      c                  DSDA0(I) = SDA(LSP+1)
226:      c                  DSDA1(I) = SDA(LSP+2)
227:      c                  DSDA2(I) = SDA(LSP+3)
228:      c                  DSDA3(I) = SDA(LSP+4)
229:      c                  DSDA4(I) = SDA(LSP+5)
230:      c          CONTINUE
231:      c          else if ( KSF.eq.2 .or. KSF.eq.5 ) then
232:      c              do 220 I = IPZONE(NK), IPZONE(NK+1) - 1
233:      c                  LLS(I) = LSG
234:      c                  IZT(I) = LLG
235:      c                  DSDA0(I) = SDA(LSP+1)
236:      c                  DSDA1(I) = SDA(LSP+2)
237:      c                  DSDA2(I) = SDA(LSP+3)
238:      c                  DSDA3(I) = SDA(LSP+4)
239:      c 220 CONTINUE
240:      c          else if ( KSF.eq.3 ) then
241:      c              do 230 I = IPZONE(NK), IPZONE(NK+1) - 1
242:      c                  LLS(I) = LSG
243:      c                  IZT(I) = LLG
244:      c                  DSDA0(I) = SDA(LSP+1)
245:      c                  DSDA1(I) = SDA(LSP+2)
246:      c                  DSDA2(I) = SDA(LSP+3)
247:      c                  DSDA3(I) = SDA(LSP+4)
248:      c                  DSDA4(I) = SDA(LSP+5)
249:      c                  DSDA5(I) = SDA(LSP+6)
250:      c                  DSDA6(I) = SDA(LSP+7)
251:      c 230 CONTINUE
252:      c          else if ( KSF.eq.4 ) then
253:      c              do 240 I = IPZONE(NK), IPZONE(NK+1) - 1
254:      c                  LLS(I) = LSG
255:      c                  IZT(I) = LLG
256:      c                  DSDA0(I) = SDA(LSP+1)
257:      c                  DSDA1(I) = SDA(LSP+2)
258:      c                  DSDA2(I) = SDA(LSP+3)
259:      c                  DSDA3(I) = SDA(LSP+4)
260:      c                  DSDA4(I) = SDA(LSP+5)

```

src/gmvp/flight.f

```

261:          DSDA5(I) = SDA(LSP+6)
262:          DSDA6(I) = SDA(LSP+7)
263:          DSDA7(I) = SDA(LSP+8)
264:          DSDA8(I) = SDA(LSP+9)
265:          DSDA9(I) = SDA(LSP+10)
266: 240      continue
267:      end if
268:  end if
269: 250      continue
270: C
271: C
272: C-----
273: C ..... D11: SMALLER DISTANCE   D22: LARGER DISTANCE
274: C ..... DLOC1: MAXIMUM OF SMALLER DISTANCE TO A BODY
275: C ..... DLOC2: MINIMUM OF LARGER DISTANCE TO A BODY
276: C-----
277: C
278: C
279: C ..... IF A SURFACE HAS NO CROSSING POINT WITH A PARTICLE TRACK,
280: C "MINIMUM DISTANCE (D11)" BECOMES GREATER THAN "MAXIMUM DISTANCE
281: C (D22)" AND THE CODE JUDGES THAT A BODY INCLUDING THE
282: C SURFACE HAS NO CROSSING POINT WITH THE PARTICLE TRACK.
283: C
284:      do 260 I = 1, NFFL(NZ1)
285:          D11(I) = -DINF
286:          D22(I) = DINF
287: 260      continue
288: C
289:      KKK = 1
290: 270      KSF = KKSF(KKK)
291:      do 280 NK = KKK, NZK
292:          if ( KKSF(NK).ne.KSF ) go to 290
293: 280      continue
294:      NK = NZK + 1
295: 290      KKK2 = NK
296: C
297: C ..... SLAB .....
298: C
299:      if ( KSF.eq.1 ) then
300:          do 300 I = IPZONE(KKK), IPZONE(KKK2) - 1
301:              DUV = DSDA0(I)*AI(I) + DSDA1(I)*BI(I) + DSDA2(I)*
302:              & CI(I)
303:              DUP = -(DSDA0(I)*X(I)+DSDA1(I)*Y(I)+DSDA2(I)*Z(I))
304:              D1 = DSDA3(I) + DUP
305:              D2 = DSDA4(I) + DUP
306:              if ( DUV.ne.0.0 ) then
307:                  D1 = D1/DUV
308:                  D2 = D2/DUV
309:                  D11(I) = MIN(D1,D2)
310:                  D22(I) = MAX(D1,D2)
311: C ..... NEVER CROSSING CASE WITH BODY ( IZT(I) > 0 ) .....
312: C (PARTICLE POSITION IS OUTSIDE OF SLAB)
313: *VOCL STMT,IF(0)
314:      else if ( IZT(I).gt.0 ) then
315: CM1992-1-17      IF(D1*D2.GT.0.0) THEN
316:                  if ( D1*D2.ge.0.0 ) then
317:                      D11(I) = DINF
318:                      D22(I) = -DINF
319:                  end if
320:              end if
321: 300      continue
322:      end if
323: C
324: C ..... SPHERE .....
325: C

```

```

326:      if ( KSF.eq.2 ) then
327:          do 310 I = IPZONE(KKK), IPZONE(KKK2) - 1
328:              DPCX = X(I) - DSDA0(I)
329:              DPCY = Y(I) - DSDA1(I)
330:              DPCZ = Z(I) - DSDA2(I)
331:              DVPC = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
332:              DH = DSDA3(I) - DPCX*DPCX - DPCY*DPCY - DPCZ*DPCZ
333:              DD = DVPC*DVPC + DH
334:              if ( DD.ge.0.0 ) then
335:                  D11(I) = -DVPC - SQRT(DD)
336:                  D22(I) = -D11(I) - 2.0D0*DVPC
337:              else if ( IZT(I).gt.0 ) then
338: C
339: C ..... NEVER CROSSING CASE WITH BODY ( IZT(I) > 0 ) .....
340: C (PARTICLE POSITION IS OUTSIDE OF SPHERE)
341: CM1992-1-17      IF(DH.LT.0.0) THEN
342:                  D11(I) = DINF
343:                  D22(I) = -DINF
344: CM1992-1-17      ENDIF
345:              end if
346: 310      continue
347:      end if
348: C
349: C ..... CYLINDER .....
350: C
351:      if ( KSF.eq.3 ) then
352:          do 320 I = IPZONE(KKK), IPZONE(KKK2) - 1
353:              DPCX = X(I) - DSDA0(I)
354:              DPCY = Y(I) - DSDA1(I)
355:              DPCZ = Z(I) - DSDA2(I)
356:              DUV = DSDA3(I)*AI(I) + DSDA4(I)*BI(I) + DSDA5(I)*
357:              & CI(I)
358:              DA = 1.0D0 - DUV*DUV
359:              DUPC = DSDA3(I)*DPCX + DSDA4(I)*DPCY + DSDA5(I)*DPCZ
360:              DB = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ - DUV*DUPC
361:              DH = DPCX**2 + DPCY**2 + DPCZ**2 - DSDA6(I)
362:              & - DUPC**2
363:              DD = DB*DB - DA*DH
364:              if ( DA.gt.0.0.and.DD.ge.0.0 ) then
365:                  DD = SQRT(DD)
366:                  D11(I) = (-DB-DD) /DA
367:                  D22(I) = (-DB+DD) /DA
368:              else if ( IZT(I).gt.0 ) then
369: CM1992-1-17      IF(DH.GT.0.0) THEN
370:                  if ( DH.ge.0.0 ) then
371:                      D11(I) = DINF
372:                      D22(I) = -DINF
373:                  end if
374:              end if
375: 320      continue
376:      end if
377: C
378: C .... QUADRATIC EXPRESSION .....
379: C S0*X**2 + S1*Y**2 + S2*Z**2 + S3*X*Y + S4*Y*Z + S5*Z*X
380: C S6*X + S7*Y + S8*Z + S9 = 0
381: C
382:      if ( KSF.eq.4 ) then
383:          do 330 I = IPZONE(KKK), IPZONE(KKK2) - 1
384: *VOCL STMT,IF(50)
385:              if ( DLOC2(I).gt.0 ) then
386:                  DRV1 = DSDA0(I)*AI(I) + DSDA3(I)*BI(I) + DSDA5(I)*
387:                  & CI(I)
388:                  DRV2 = DSDA1(I)*BI(I) + DSDA4(I)*CI(I)
389:                  DRV3 = DSDA2(I)*CI(I)
390:                  DRW1 = DSDA0(I)*X(I) + DSDA3(I)*Y(I) + DSDA5(I)*

```

src/gmvp/flight.f

```

391:      &          Z(I) + DSDA6(I)
392:      DRW2      = DSDA1(I)*Y(I) + DSDA4(I)*Z(I) + DSDA7(I)
393:      DRW3      = DSDA2(I)*Z(I) + DSDA8(I)
394:      DA        = AI(I)*DRV1 + BI(I)*DRV2 + CI(I)*DRV3
395:      DB        =
396:      &          (AI(I)*DRW1+BI(I)*DRW2+CI(I)*DRW3+X(I)*DRV1
397:      &          +Y(I)*DRV2+Z(I)*DRV3)*0.5D0
398:      DC        = X(I)*DRW1 + Y(I)*DRW2 + Z(I)*DRW3 + DSDA9(I)
399: *VOCL STMT,IF(100)
400:      if ( DA.ne.0.0 ) then
401:      DD        = DB**2 - DC*DA
402:      if ( DD.ge.0.0 ) then
403:      DH        = SQRT(DD)
404:      D11(I)    = (-DB+DH) /DA
405:      D22(I)    = (-DB+DH) /DA
406: C ..... CONCAVE SURFACE ( DA < 0, D11 > D22 ) .....
407:      if ( DA.lt.0.0 ) then
408:      if ( D22(I).gt.0.0.and.D22(I).gt.DLOC1(I) )
409:      &      then
410:      D11(I)    = -DINF
411:      else
412:      D22(I)    = DINF
413:      end if
414:      end if
415: C ..... NO REAL ROOT, AND PARTICLE TRACK IS OUTSIDE OF SURFACE
416:      else if ( IZT(I).gt.0.and.DA.gt.0 ) then
417:      D11(I)    = DINF
418:      D22(I)    = -DINF
419:      end if
420: C ..... ONLY ONE REAL ROOT (DA = 0)
421: *VOCL STMT,IF(0)
422:      else
423:      if ( DB.gt.0.0 ) then
424:      D22(I)    = -DC/(DB*2.0D0)
425:      else if ( DB.lt.0.0 ) then
426:      D11(I)    = -DC/(DB*2.0D0)
427: C ..... NO ROOT, AND PARTICLE TRACK IS IN OUTSIDE OF THE SURFACE
428: *VOCL STMT,IF(0)
429:      else if ( IZT(I).gt.0.and.DC.ge.0.0 ) then
430: CM1992-1-17      ELSE IF(IZT(I).GT.0.AND.DC.GT.0.0) THEN
431:      D11(I)    = DINF
432:      D22(I)    = -DINF
433:      end if
434:      end if
435:      end if
436: 330      continue
437:      end if
438: C
439: C .... PLANE (HALF SPACE) .....
440: C
441:      if ( KSF.eq.5 ) then
442:      do 340 I = IPZONE(KKK), IPZONE(KKK2) - 1
443:      DA        = DSDA0(I)*AI(I) + DSDA1(I)*BI(I) + DSDA2(I)*
444:      &          CI(I)
445:      DB        = DSDA3(I)
446:      &          - (DSDA0(I)*X(I)+DSDA1(I)*Y(I)+DSDA2(I)*Z(I))
447: *VOCL STMT,IF(100)
448:      if ( DA.ne.0.0 ) then
449:      DD        = DB/DA
450:      if ( DA.lt.0.0 ) then
451:      D11(I)    = DD
452:      else
453:      D22(I)    = DD
454:      end if
455: *VOCL STMT,IF(0)

```

```

456:      else if ( IZT(I).gt.0.and.DB.le.0.0 ) then
457: CM1992-1-17      ELSE IF(IZT(I).GT.0. AND. DB.LT.0.0 ) THEN
458:      D11(I)    = DINF
459:      D22(I)    = -DINF
460:      end if
461: 340      continue
462:      end if
463: C
464:      KKK      = KKK2
465:      if ( KKK.le.NZK ) go to 270
466: C
467: C ..... CALCULATE MINIMUM DISTANCE
468: C
469: C =====
470: C =
471: C = DISTANCE DETERMINATION FOR COMBINATORIAL GEOMETRY (88/4/12)
472: C = VALUES OF IZT(I) (=KZDA(2,NK)) IS ASSIGNED SUCH AS;
473: C =
474: C = A.AND.B.AND. .NOT.( C.AND.D.AND.E ) .AND. .NOT.( F.AND.G )
475: C = |         |         |         |         |         |
476: C = 0         0         1         1         2         1         2
477: C =
478: C = WHERE A, B, C ETC. ARE SURFACES COMPOSING A ZONE.
479: C =
480: C =====
481:      if ( JSIMP.eq.0 ) then
482:      do 350 I = 1, NFFL(NZ1)
483:      if ( IZT(I).eq.0 ) then
484:      if ( LLS(I).gt.0.and.D22(I).ge.0.0 )
485:      &      DI(I) = MIN(DI(I),D22(I))
486:      if ( LLS(I).lt.0.and.D11(I).ge.0.0 )
487:      &      DI(I) = MIN(DI(I),D11(I))
488:      else
489:      DLOC1(I)  = MAX(DLOC1(I),D11(I))
490:      DLOC2(I)  = MIN(DLOC2(I),D22(I))
491:      end if
492:      if ( IZT(I).eq.2 ) then
493: CM1992-1-17      IF(DLOC1(I).LE.DLOC2(I) .AND. DLOC1(I).GE.0.0)
494:      if ( DLOC1(I).lt.DLOC2(I).and.DLOC1(I).ge.0.0 )
495:      &      DI(I) = MIN(DI(I),DLOC1(I))
496:      DLOC1(I)  = -DINF
497:      DLOC2(I)  = DINF
498:      end if
499: 350      continue
500:      else
501:      do 360 I = 1, NFFL(NZ1)
502:      if ( LLS(I).gt.0 ) then
503:      if ( D22(I).ge.0.0 ) DI(I) = MIN(DI(I),D22(I))
504:      else
505:      if ( D11(I).ge.0.0 ) DI(I) = MIN(DI(I),D11(I))
506:      end if
507: 360      continue
508:      end if
509: 370 continue
510: C
511: C -----
512: C .... CHECK LOST PARTICLE .....
513: C -----
514: C
515:      II      = 0
516:      do 380 I = 1, NFFL(NZ1)
517:      if ( DI(I).eq.DINF ) II = II + 1
518: 380 continue
519: C
520: C .... COUNT THE NUMBER OF FREE FLIGHT

```

```

      IGI(I) = -IGI(I)
    end if
    continue
  do 400 I = NDEAD + 1, ILOST
    LL = LSDED(I)
    write(IOW 7000) XXX(LL), YYY(LL)
    CCC(LL)
    format(1X, ' X=' ,LP,E12.5, ' Y=' ,E
      E12.5, ' ETA=' ,E12.5, ' XI

```

```

1, ILOST
(I)
XXX(LL), YYY(LL), ZZZ(LL), AAA(LL), BBB(LL),
, E12.5, ' Y=' , E12.5, ' Z=' , E12.5, ' MU= ,
ETA=' , E12.5, ' XI=' , E12.5)

-----
ES .....

L(NZ1)
0 ) then
SAFE + 1
) = LSFFL(I)
) = IZFFL(I)
= DI(I)

-----
CLES AGAIN .....

NE

FE
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
LSFFL(I))
FE

```

```

586:      do 450 I = 1, NFFL(NZ1)
587:          if ( IZFFL(I).eq.K ) KK = KK + 1
588:      450      continue
589:      end if
590:      460      continue
591:          IPZONE(NZK+1) = NFFL(NZ1) + 1
592:      end if
593: C
594: C-----
595: C .... CALCULATE DISTANCE TO LATTICE FRAME & COORDINATES IN UPPER ....
596: C      IBP(I) = 1/0 = COLLISION/ NON COLLISION
597: C-----
598: C
599: C ( IZS=1, IZE=NZK, LLS AS IFC, DSDA0 TO DSDA5 AS XUP TO CUP )
600: C
601:      if ( JHLAT.ne.0 ) then
602:          if ( JVMNT.ne.0 ) call VMNTR0( 11, IOW )
603: C
604: C
605: C      call LFRAME( IOW, JDEBG, NBANK, NEST, NLATT,NLBZ,NZONE,
606: C          &      DINF, 1, NZK, IPZONE(NZK+1)-1, DI, LLS, LEVL, LZZ,
607: C          &      LPOS, LCRS, KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ,
608: C          &      IPLAT, KLATT, KSLAT, LTYP, MLBZZ, IPZONE, DSDA0, DSDA1,
609: C          &      DSDA2, DSDA3, DSDA4, DSDA5, X, Y, Z, AI, BI, CI, IBP, R
610: C          &      )
611: C      call LFRAME( IOW, JDEBG, NBANK, NEST, NLATT,NLBZ,NZONE,
612: C          &      DINF, IBP, IPZONE, 1, NZK, DI, LLS,
613: C          &      DSDA0, DSDA1, DSDA2, DSDA3, DSDA4, DSDA5,
614: C          &      LEVL, LZZ,
615: C          &      LPOS, LCRS, KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ,
616: C          &      IPLAT, KLATT, KSLAT, LTYP, MLBZZ,
617: C          &      X, Y, Z, AI, BI, CI, R )
618: C      if ( JVMNT.ne.0 ) call VMNTR2( 11 )
619:      end if
620: C
621: C-----
622: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) ....
623: C      IBP(I) = 1/0 = COLLISION/ NON COLLISION
624: C-----
625: C
626: C ---- HEXAGONAL LATTICE -----
627: C
628:      if ( JHLAT.ne.0 ) then
629:          ICOLS = 0
630:          call RANU2( IRAND, R, NFFL(NZ1), ICON )
631:          do 470 I = 1, NFFL(NZ1)
632:              IBP(I) = 0
633:      470      continue
634:              KKK = 1
635: C
636: C      480      MAT = KZMAT(JKSF(KKK))
637: C          do 490 NK = KKK, NZK
638: C              if ( KZMAT(JKSF(NK)).ne.MAT ) go to 500
639: C      490      continue
640: C          NK = NZK + 1
641: C      500      KKK2 = NK
642: C
643:      if ( MAT.gt.0 ) then
644:          do 510 I = IPZONE(KKK), IPZONE(KKK2) - 1
645:              D1 = -LOG(R(I)) /STOTX(IGI(I),MAT)
646:              if ( DI(I).gt.D1 ) then
647:                  IBP(I) = 1
648:                  DI(I) = D1
649:                  ICOLS = ICOLS + 1
650:                  LSCOL(NCOLS+ICOLS) = LSFFL(I)

```

src/gmvp/flight.f

```

651:           end if
652: 510       continue
653:       end if
654:       KKK      = KKK2
655:       if ( KKK.le.NZK ) go to 480
656: C
657: C-----
658: C .... SEND UNCOLLIDED PARTICLES TO SEARCH STACK .....
659: C-----
660:       INXT      = 0
661: *VOCL LOOP,NOVREC
662:       do 520 I = 1, NFFL(NZ1)
663:         if ( IBP(I).eq.0 ) then
664:           IZNN      = IZFFL(I)
665: C
666: C ..... IN THE CASE OF HEXAGONAL LATTICE GEOMETRY .....
667: C
668:         if ( LLS(I).ne.0 ) then
669:           IZNN      = LLS(I)
670:           LEVL(LSFFL(I)) = LEVL(LSFFL(I)) - 1
671:           IZFFL(I)   = LLS(I)
672:           X(I)       = DSDA0(I)
673:           Y(I)       = DSDA1(I)
674:           Z(I)       = DSDA2(I)
675:           AI(I)      = DSDA3(I)
676:           BI(I)      = DSDA4(I)
677:           CI(I)      = DSDA5(I)
678:         end if
679: C
680:         INXT      = INXT + 1
681:         LSSRC(NNXT(NZ1)+INXT) = LSFFL(I)
682:         IZNXT(NNXT(NZ1)+INXT) = IZNN
683:       end if
684: 520       continue
685: *VOCL LOOP,SCALAR
686:       do 530 I = 1, NFFL(NZ1)
687:         if ( IBP(I).eq.0 ) NNXT(IZFFL(I)) = NNXT(IZFFL(I)) + 1
688: 530       continue
689: C
690:         NCOLS     = NCOLS + ICOLS
691:         NNXT(NZ1) = NNXT(NZ1) + INXT
692: C
693: C ---- NO-HEXAGONAL LATTICE
694: C
695:       else
696:         ICOLS     = 0
697:         call RANU2( IRAND, R, NFFL(NZ1), ICON )
698:         do 540 I = 1, NFFL(NZ1)
699:           IBP(I)   = 0
700: 540       continue
701:         KKK      = 1
702: C
703: C
704: 550       MAT      = KZMAT(JKSF(KKK))
705:         do 560 NK = KKK, NZK
706:           if ( KZMAT(JKSF(NK)).ne.MAT ) go to 570
707: 560       continue
708:         NK      = NZK + 1
709: 570       KKK2     = NK
710: C
711:         if ( MAT.gt.0 ) then
712:           do 580 I = IPZONE(KKK), IPZONE(KKK2) - 1
713:             D1      = -LOG(R(I)) /STOTX(IGI(I),MAT)
714:             if ( DI(I).gt.D1 ) then
715:               IBP(I) = 1

```

```

716:           DI(I)     = D1
717:           ICOLS     = ICOLS + 1
718:           LSCOL(NCOLS+ICOLS) = LSFFL(I)
719:         end if
720: 580       continue
721:       end if
722:       KKK      = KKK2
723:       if ( KKK.le.NZK ) go to 550
724: C
725: C
726: C-----
727: C .... SEND UNCOLLIDED PARTICLES TO SEARCH STACK .....
728: C-----
729:       INXT      = 0
730:       do 590 I = 1, NFFL(NZ1)
731:         if ( IBP(I).eq.0 ) then
732:           INXT      = INXT + 1
733:           LSSRC(NNXT(NZ1)+INXT) = LSFFL(I)
734:           IZNXT(NNXT(NZ1)+INXT) = IZFFL(I)
735:         end if
736: 590       continue
737: *VOCL LOOP,SCALAR
738:       do 600 I = 1, NFFL(NZ1)
739:         if ( IBP(I).eq.0 ) NNXT(IZFFL(I)) = NNXT(IZFFL(I)) + 1
740: 600       continue
741: C
742:         NCOLS     = NCOLS + ICOLS
743:         NNXT(NZ1) = NNXT(NZ1) + INXT
744:       end if
745: C
746: C-----
747: C .... MOVE PARTICLES TO NEXT POINTS AND PUT VALUES IN BANK.....
748: C-----
749: *VOCL LOOP,NOVREC
750:       do 610 I = 1, NFFL(NZ1)
751:         XXX(LSFFL(I)) = X(I) + AI(I)*DI(I)
752:         YYY(LSFFL(I)) = Y(I) + BI(I)*DI(I)
753:         ZZZ(LSFFL(I)) = Z(I) + CI(I)*DI(I)
754:         DI(I)         = DI(I)*W(I)
755: 610       continue
756: C
757: C-----
758: C       FLUX & RESPONSE TALLY ( non-universe-dependent )
759: C-----
760: C
761: C
762:       if ( JTLLT.eq.0 ) then
763: C
764: C .... TAKE TALLIES (FLUX) .....
765: C
766:         do 630 NK = 1, NZK
767:           do 620 I = IPZONE(NK), IPZONE(NK+1) - 1
768:             IBP(I) = IGI(I) + (KZREG(JKSF(NK))-1)*NGROUP
769: 620         continue
770: 630       continue
771: *VOCL LOOP,SCALAR
772:       do 640 I = 1, NFFL(NZ1)
773:         FLTR(IBP(I),1) = FLTR(IBP(I),1) + DI(I)
774: 640       continue
775: C
776: C .... TAKE TALLIES (RESPONSE) .....
777: C
778: C       IF(NRESP.GT.0) THEN
779: C         DO 580 N=1,NRESP
780: C           DO 550 I=1,NFFL(NZ1)

```


src/gmvp/flight.f

```
781: c 550          W(I) = DI(I)*RESP(IGI(I),N)
782: c              DO 570 NK=1,NZK
783: c              MREG = KZREG(JKSF(NK))
784: c              DO 560 I=IPZONE(NK),IPZONE(NK+1)-1
785: c              RETR(MREG,N) = RETR(MREG,N) + W(I)
786: c              CONTINUE
787: c 570          CONTINUE
788: c 580          CONTINUE
789: c          cccccccc ENDIF
790: c
791: c-----
792: c          FLUX & RESPONSE TALLY ( universe-dependent )
793: c-----
794: c
795: c          else
796: c
797: c          .... TAKE TALLIES (FLUX) .....
798: c
799: c          *VOCL LOOP, SCALAR
800: c              do 650 I = 1, NFFL(NZ1)
801: c                  FLTR(IGI(I),IBREG(LSFFL(I))) =
802: c                  & +FLTR(IGI(I),IBREG(LSFFL(I))) + DI(I)
803: c          650      continue
804: c
805: c          .... TAKE TALLIES (RESPONSE) .....
806: c
807: c          IF(NRESP.GT.0) THEN
808: c              DO 582 N=1,NRESP
809: c              DO 552 I=1,NFFL(NZ1)
810: c              W(I) = DI(I)*RESP(IGI(I),N)
811: c              DO 572 NK=1,NZK
812: c              MREG = KZREG(JKSF(NK))
813: c              DO 562 I=IPZONE(NK),IPZONE(NK+1)-1
814: c              RETR(MREG,N) = RETR(MREG,N) + W(I)
815: c              CONTINUE
816: c          572      CONTINUE
817: c          582      CONTINUE
818: c          cccccccc ENDIF
819: c              end if
820: c
821: c-----
822: c          .... TAKE TALLIES (EIGEN VALUE) .....
823: c-----
824: c
825: c          if ( JEIGN.ne.0 ) then
826: c              do 660 I = 1, NFFL(NZ1)
827: c                  M = KZMAT(IZFFL(I))
828: c                  if ( M.gt.0 ) then
829: c                      WCNTR(15) = WCNTR(15) + DI(I)*SNUFX(IGI(I),M)
830: c                      WCNTR(20) = WCNTR(20) + DI(I)*STOTX(IGI(I),M)*
831: c                      & (1.-SNAPX(IGI(I),M))
832: c                  end if
833: c          660      continue
834: c          end if
835: c
836: c-----
837: c          .... DELETE PARTICLES FROM FLIGHT STACK .....
838: c-----
839: c
840: c              do 670 I = 1, NZ1
841: c                  NFFL(I) = 0
842: c          670      continue
843: c
844: c              return
845: c          end
```

src/gmvp/flione.f

```

1: C/IF ARGSAVE
2: *      subroutine FLION0(IOW,IRAND,NGROUP,NTGX,NGP1,NGP2,
3: *      N      NBANK,NZONE,NSDA,NZDA,NCELL,NDEAD,NREG,NCOLS,NLOST,
4: *      N      DINF,DEPS,NLATT, NUNV, NSPACE,NKTCSP,
5: *      N      NEST, NLBZ,NTIME, TCUT,NRESP,
6: *      B      XXX,YYY,ZZZ,AAA,BBB,CCC,
7: *      B      WWW,IGG,TTT,ITT,LEVL,LZZ,LPOS,LCRS,KLSF,IBSPC,IBREG,
8: *      B      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
9: *      S      LSFFL,NFFL,IZFFL,
10: *      S      LSCOL,LSSRC,NNXT,IZNXT,LSDED,
11: *      G      SDA,KZMAT,KZREG,KSPSU,KTCSP,KZDA,KZAA,
12: *      G      DALT,KDALT,NVLAT,SZLAT,IPLAT,IPCEL,
13: *      G      KSLAT,LTYP,MLBZZ,KZLBZ,KCELL,ICTYP,KLATT,CELSZ,
14: *      G      PNND,KNND,NPNND,PPPF,KPPF,NPPPF,
15: *      T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, PSALP, PSXYZ,
16: *      T      DTALY, RESP, STOTX, SNUFX, SNAPX, TIMEB, VEL, NMKREG, MKREG,
17: *      T      NTSRF, KTSRF, ITSRF, IDSRF, ANGLB, NANGLE,
18: *      T      FLTR,
19: *      T      TFLH, NCNTR, WCNTR )
20: C/ELSE
21:      subroutine FLIONE( IOW, IRAND, NGROUP,NTGX,NGP1,NGP2,
22:      &      NBANK, NZONE, NSDA, NZDA, NCELL, NDEAD, NREG, NCOLS,
23:      &      NLOST, DINF, DEPS, NLATT, NUNV, NSPACE,NKTCSP,
24:      &      NEST, NLBZ, NTIME, TCUT, NRESP,
25:      &      XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IGG,
26:      &      TTT, ITT, LEVL, LZZ, LPOS, LCRS, KLSF, IBSPC,IBREG,
27:      B      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
28:      &      LSFFL, NFFL, IZFFL, LSCOL, LSSRC, NNXT, IZNXT,LSDED,
29:      &      SDA,KZMAT,KZREG, KSPSU,KTCSP, KZDA,KZAA,
30:      &      DALT, KDALT, NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP,
31:      &      MLBZZ, KZLBZ, KCELL, ICTYP, KLATT, CELSZ,
32:      G      PNND, KNND, NPNND, PPPF, KPPF, NPPPF,
33:      T      JDTRG, IDTAL, JTEVE, LTEVE, KTEVE, NTEVE, PSALP, PSXYZ,
34:      T      DTALY,RESP,STOTX,SNUFX,SNAPX, TIMEB,VEL, NMKREG, MKREG,
35:      T      NTSRF, KTSRF, ITSRF, IDSRF, ANGLB, NANGLE,
36:      T      FLTR, TFLH, NCNTR, WCNTR,
37:      W      X, Y, Z, AI, BI, CI, W, DI,
38:      W      IGI, IBP, IFC, R, DLOC1, DLOC2, IKL, IKL2,
39:      W      XUP, YUP, ZUP, AUP, BUP, CUP, IRG, TI,
40:      W      ISTG, ILUP, ILST, XYZ1, ABC1, XYZ2, ABC2,
41:      W      DDI, IBP0, IBP1, IBP2, KPLT, JKSF,
42:      &      MZONE )
43: C/ENDIF
44: C===== < FLIGHT > =====
45: C PURPOSE: FREE-FLIGHT OF ONE-ZONE LOGIC
46: C CALLED IN: ACTION
47: C CALLS: LFRAME,RANU2,VMNTR0,VMNTR1,VMNTR2
48: C
49: C=====
50: C
51: C      === INTER-STACK DATA FLOW ===
52: C
53: C      FREE-FLIGHT STACK -----+-----> NEXT-ZONE-SEARCH STACK
54: C      (LSFFL,IZFFL,NFFL)      |      (LSSRC,IZNXT,NNXT)
55: C      +-----> COLLISION STACK
56: C      |      (LSCOL,NCOLS)
57: C      (LOST) +-----> DEAD PARTICLE STACK
58: C      |      (LSDED,NDEAD)
59: C
60: C      === BANK DATA TO BE UPDATED ===
61: C
62: C      (XXX,YYY,ZZZ) : POSITION
63: C      (AAA,BBB,CCC) : DIRECTION (IF HEXAGONAL-LATTICE EXISTS)
64: C      LEVL          : LATTICE LEVEL (IF HEXAGONAL-LATTICE EXISTS)
65: C      KLSF          : SURFACE DATA POINTER (IF ON-SURFACE)

```

```

66: C      === BANK DATA ADDED NEWLY ===
67: C
68: C      (NOTHING)
69: C
70: C=====
71: C
72: C      implicit real*8(D)
73: C
74: C      include 'INC/_FLAGS'
75: C      include '../shared/INC/_TASKDT'
76: C      include '../shared/INC/_IDXOFF'
77: C
78: C      .... PARTICLE BANK .....
79: C
80: C      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
81: C      &      CCC(NBANK)
82: C      real WWW(NBANK)
83: C      real*8 TTT(NBANK)
84: C      integer ITT(NBANK)
85: C      integer IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
86: C      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
87: C      integer IBSPC(NBANK,0:NEST)
88: C      integer IBREG(NBANK)
89: C
90: C      ... optional bank parameters
91: C
92: C      real*8 DBNK(NBANK,*)
93: C      integer KDBNK(0:MDBNK)
94: C      integer IBNK(NBANK,*)
95: C      integer KIBNK(0:MIBNK)
96: C
97: C      .... STACKS .....
98: C
99: C      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSCOL(NBANK),
100: C      &      LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK)
101: C
102: C      .... GEOMETRY .....
103: C
104: C      real*8 SDA(NSDA), DALT(*), SZLAT(8,NLATT), CELSZ(6,*)
105: C      integer IPCEL(NCELL)
106: C      integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
107: C      integer KZLBZ(NLBZ)
108: C      integer KCELL(NZONE), ICTYP(NCELL), KLATT(*), NVLAT(4,*)
109: C      integer KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT), IPLAT(*), KSLAT(*)
110: C      integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
111: C      real*8 DINF, DEPS
112: C
113: C      ... Nearest Neighbour Distribution for STGM region.
114: C
115: C      real PNND(NPNND)
116: C      integer KNND(NPNND)
117: C
118: C      ... Partial Packing Fraction for STGM-region
119: C
120: C      real PPPF(NPPPF)
121: C      integer KPPF(NPPPF)
122: C
123: C      .... CROSS SECTION .....
124: C
125: C      real STOTX(NTGX,*), SNUFX(NTGX,*), SNAPX(NTGX,*)
126: C
127: C      .... TALLY .....
128: C
129: C      integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
130: C      real PSALP(NREG)

```

src/gmvp/flione.f

```

131:      real*8 PSXYZ(3)
132:      integer IDTAL(*)
133:      real*8 DTALY(*)
134:      real RESP(NGROUP,NRESP)
135: C
136:      real*8 FLTR(NGROUP,NREG)
137:      real*8 TFLH(*)
138:      real*8 NCNTR(*), WCNTR(*)
139: C
140:      real TIMEB(NTIME+1)
141:      real VEL(NGROUP)
142:      integer MKREG(NREG,2)
143: C
144:      ... Tally surface .....
145: C
146:      integer KTSRF(2,*), ITSRF(NTSRF+1,2), IDSRF(NTSRF)
147: C
148:      ... angle bin (cosine) ....
149: C
150:      real*8 ANGLB(NANGLE)
151: C
152: C .... WORKING AREA (LENGTH = NBANK) .....
153: C
154:      real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
155: &      CI(NBANK), W(NBANK), DI(NBANK), DLOC1(NBANK),
156: &      DLOC2(NBANK), TI(NBANK)
157:      real R(3*NBANK)
158:      integer IGI(NBANK), IBP(NBANK), IFC(NBANK)
159:      integer IRG(NBANK)
160:      integer ILST(NBANK)
161: C
162: C ..... USED FOR HEXAGONAL LATTICE GEOMETRY and STG region ...
163: C
164:      integer ILUP(NBANK)
165:      real*8 XUP(NBANK), YUP(NBANK), ZUP(NBANK), AUP(NBANK), BUP(NBANK),
166: &      CUP(NBANK)
167: C
168:      integer ISTG(NBANK)
169:      integer IBP0(NBANK), IBP1(NBANK), IBP2(NBANK)
170:      integer KPLT(NLBZ+1)
171:      integer JKSF(NLBZ)
172:      real*8 XYZ1(NBANK,3), ABC1(NBANK,3)
173:      real*8 XYZ2(NBANK,3), ABC2(NBANK,3)
174:      real*8 DDI(NBANK)
175: C
176: C ..... FOR SURFACE POINTER ...
177: C
178:      integer IKL(NBANK), IKL2(NBANK)
179:      integer IPZZ(2)
180: C
181:      real*8 SMALL
182:      parameter( SMALL = 1.0D-35 )
183: C
184: C .... statement function to check lattice zone
185: C
186:      include '../shared/INC/_PMLATT'
187:      include '../shared/INC/_SFLATT'
188:
189: C/#IF ARGSAVE
190: *      return
191: C -----
192: *      entry      FLIONE( MZONE,
193: *      W          X      ,Y      ,Z      ,AI      ,BI      ,CI      ,
194: *      W          W      ,DI      ,IGI      ,IBP      ,IFC      ,R      ,DLOC1 ,
195: *      W          DLOC2 ,IKL      ,IKL2      ,XUP      ,YUP      ,ZUP      ,

```

```

196: *      W          AUP      ,BUP      ,CUP      ,IRG      ,TI      ,
197: *      W          ISTG, ILUP, ILST, XYZ1, ABC1, XYZ2, ABC2,
198: *      W          DDI, IBP0, IBP1, IBP2, KPLT, JKSF)
199: C/#ENDIF
200:
201:      if ( JVMNT.ne.0 ) call VMNTRI( 3, NFFL(MZONE) )
202:      NZ1 = NZONE + 1
203: C
204: C-----
205: C .... Material #
206: C-----
207: C
208:      MAT = KZMAT(MZONE,1)
209: C
210: C ... this is a base material (matrix) region of STG region.
211: C
212:      JJJNND = 0
213:      if ( MAT.lt.0 ) then
214:        if ( ISLATT(MAT).and.LTYP(LATTNM(MAT)).eq.10 ) then
215:          MAT = KZMAT(MZONE,2)
216:          JJJNND = 1
217:        else
218:          write(IOW,*) 'XXX(FLIONE) Program error ? Invalid zone :',
219: &          ' MZONE=', MZONE, ' KZMAT(MZONE,*) ',
220: &          KZMAT(MZONE,1), KZMAT(MZONE,2), ' LTYP ',
221: &          LTYP(LATTNM(MAT))
222:          stop 666
223:        end if
224:      end if
225: C
226: C-----
227: C .... GATHER VECTORS .....
228: C-----
229: C
230:      II = 0
231:      do 100 I = 1, NFFL(NZ1)
232:        if ( IZFFL(I).eq.MZONE ) then
233:          II = II + 1
234:          IBP(II) = LSFFL(I)
235:        end if
236:      100 continue
237:
238:      do 110 I = 1, NFFL(MZONE)
239:        X(I) = XXX(IBP(I))
240:        Y(I) = YYY(IBP(I))
241:        Z(I) = ZZZ(IBP(I))
242:        AI(I) = AAA(IBP(I))
243:        BI(I) = BBB(IBP(I))
244:        CI(I) = CCC(IBP(I))
245:        W(I) = WWW(IBP(I))
246:        IGI(I) = IGG(IBP(I))
247:        IKL(I) = KLSF(IBP(I))
248:        IKL2(I) = 0
249: C
250:        DI(I) = DINF
251:        ILST(I) = 0
252:      110 continue
253: C
254: C-----
255: C .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
256: C-----
257: C
258: C      for matrix zone of STG region, distance to zone boundary
259: C      is that to STG region boundary and not calculated here
260: C      but in DFRAME. (also means that arrival to STG region frame

```

src/gmvp/flione.f

```

261: C      must be treated as a movement to an upper lattice level.)
262: C
263: C      if ( JFISX.eq.0 .or. JJJNND.eq.0 ) then
264: C          JSS = 0
265: C          call SPEAR1( MZONE, KZDA, KZAA, SDA, NFFL(MZONE), NBANK, JSS,
266: C      &              X, Y, Z, AI, BI, CI, DI, DLOC1, DLOC2, IKL, IKL2,
267: C      &              DUMMY1, DINF, DEPS )
268: C
269: C      ... "lost" particle index to "1"
270: C
271: C      *VOCL LOOP,NOVREC
272: C          do 120 I = 1, NFFL(MZONE)
273: C              if ( DI(I).eq.DINF ) then
274: C                  ILST(I) = 1
275: C              end if
276: C          120 continue
277: C      end if
278: C
279: C      ( DLOC1 and DLOC2 are free to use hereafter )
280: C -----
281: C ----- HEXAGONAL LATTICE -----
282: C ... CALCULATE DISTANCE TO LATTICE FRAME & COORDINATES IN UPPER LEVEL
283: C      IFC(I) = N/0 = CROSS THE OUTER FRAME (N=ZONE# IN UPPER LEVEL)/ NO
284: C      VALUE OF DI(I) MAY BE CHANGED
285: C      ( IZS=1, IZE=1, LLS AS IFC DSDA0 TO DSDA5 AS XUP TO CUP )
286: C -----
287: C
288: C      IHF = 0
289: C
290: C      if ( JFISX.eq.0.and.JHLAT.ne.0 ) then
291: C          if ( KCELL(MZONE).gt.0.and.ICTYP(KCELL(MZONE)).eq.2 ) then
292: C              IHF = 1
293: C              IPZZ(1) = 1
294: C              IPZZ(2) = NFFL(MZONE) + 1
295: C
296: C              if ( JVMNT.ne.0 ) call VMNTR0( 11, IOW )
297: C              call LFRAME( IOW, JDEBG, NBANK, NEST, NLATT, NLBZ, NZONE,
298: C      &                  DINF, IBP, IPZZ, 1, 1, DI, IFC, XUP, YUP, ZUP, AUP,
299: C      &                  BUP, CUP, LEVL, LZZ, LPOS, LCRS, KZMAT, KDALT, DALT,
300: C      &                  NVLAT, SZLAT, CELSZ, IPLAT, KLATT, KSLAT, LTYP,
301: C      &                  MLBZZ, X, Y, Z, AI, BI, CI, R )
302: C
303: C      *VOCL LOOP,NOVREC
304: C          do 130 I = 1, NFFL(MZONE)
305: C
306: C          .... this case moves only one upper lattice level
307: C
308: C              ILUP(I) = LEVL(IBP(I)) - 1
309: C
310: C          ... lost in frame distance calculation
311: C
312: C              if ( DI(I).eq.DINF ) then
313: C                  ILST(I) = ILST(I) + 2
314: C              end if
315: C          130 continue
316: C              if ( JVMNT.ne.0 ) call VMNTR2( 11 )
317: C
318: C          ... woking area IKL is never used after this call
319: C
320: C
321: C      end if
322: C
323: C      ... distance to frames until current lattice level is
324: C      calculated if flight path count is zero.
325: C

```

```

326: C      else if ( JFISX.ne.0.and.(KCELL(MZONE).gt.0.or.JJJNND.gt.0) ) then
327: C          IHF = 1
328: C          KK = 0
329: C      *VOCL LOOP,NOVREC
330: C          do 140 I = 1, NFFL(MZONE)
331: C              IFC(I) = 0
332: C              if ( IBNK(IBP(I),KIBNK(5)).eq.0 ) then
333: C                  KK = KK + 1
334: C              end if
335: C          140 continue
336: C              if ( KK.gt.0 ) then
337: C                  if ( JVMNT.ne.0 ) call VMNTR0( 11, IOW )
338: C                  call DFRAME( IOW, MZONE, NFFL(MZONE), JDEBG, NBANK, NEST,
339: C      &                      NLATT, NLBZ, NZONE, DINF, DEPS, X, Y, Z, AI, BI, CI,
340: C      &                      IBP, DI, IFC, XUP, YUP, ZUP, AUP, BUP, CUP, ILUP,
341: C      &                      ILST, LEVL, LZZ, LPOS, LCRS, DBNK, KDBNK, MDBNK,
342: C      &                      IBNK, KIBNK, MIBNK, KZMAT, KZDA, KZAA, SDA, KCELL,
343: C      &                      KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT, KLATT,
344: C      &                      KSLAT, LTYP, MLBZZ, KZLBZ, KPLT, JKSF, IBP0, IBP1,
345: C      &                      IBP2, XYZ1(1,1), XYZ1(1,2), XYZ1(1,3), ABC1(1,1),
346: C      &                      ABC1(1,2), ABC1(1,3), XYZ2(1,1), XYZ2(1,2),
347: C      &                      XYZ2(1,3), ABC2(1,1), ABC2(1,2), ABC2(1,3), DDI,
348: C      &                      IKL, IKL2, DLOC1, DLOC2 )
349: C                  if ( JVMNT.ne.0 ) call VMNTR2( 11 )
350: C              end if
351: C
352: C      ... get frame distance and upper level postion etc. from
353: C      banked data
354: C
355: C      if ( NFFL(MZONE)-KK.gt.0 ) then
356: C      *VOCL LOOP,NOVREC
357: C          do 150 I = 1, NFFL(MZONE)
358: C              KL = IBNK(IBP(I),KIBNK(6))+LEVL(IBP(I))-1
359: C              KI = KDBNK(4) + KL - 1
360: C              DDDD = DBNK(IBP(I),KI)
361: C              if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.DDDD.lt.DI(I) ) then
362: C                  IFC(I) = LZZ(IBP(I),KL)
363: C                  ILUP(I) = KL - 1
364: C                  K6 = KDBNK(6) + 6*(KL-1)
365: C                  AUP(I) = DBNK(IBP(I),K6+3)
366: C                  BUP(I) = DBNK(IBP(I),K6+4)
367: C                  CUP(I) = DBNK(IBP(I),K6+5)
368: C                  XUP(I) = DBNK(IBP(I),K6) - DDDD*AUP(I)
369: C                  YUP(I) = DBNK(IBP(I),K6+1) - DDDD*BUP(I)
370: C                  ZUP(I) = DBNK(IBP(I),K6+2) - DDDD*CUP(I)
371: C                  DI(I) = DDDD
372: C              end if
373: C          150 continue
374: C              end if
375: C          end if
376: C
377: C -----
378: C ... CHECK LOST PARTICLE .....
379: C -----
380: C
381: C      ISAFE = NFFL(MZONE)
382: C      II = 0
383: C
384: C      IF(JVMNT.NE.0) CALL VMNTR1('FLIGHT',NFFL(MZONE))
385: C
386: C      do 160 I = 1, NFFL(MZONE)
387: C          if ( ILST(I).ne.0 ) II = II + 1
388: C      160 continue
389: C
390: C      ... COUNT THE NUMBER OF FREE FLIGHT ...

```

src/gmvp/flione.f

```

391: C
392:     NCNTR(16) = NCNTR(16) + ISAFE - II
393: C
394:     if ( II.gt.0 ) then
395:
396: C/#IF PARA(SX* CRAY)
397: *     call MVPSYNC_LOCK(2)
398: C/#ENDIF
399:     write(IOW,'(/lx,a,i5,a,i5,a)') ' (FLIONE) ZONE ', MZONE, ' : ',
400: &     II, ' PARTICLES ARE LOST !! '
401:     NLOST = NLOST + II
402:     ILOST = NDEAD
403:     do 170 I = 1, NFFL(MZONE)
404:         if ( ILST(I).ne.0 ) then
405:             ILOST = ILOST + 1
406:             L$DED(ILOST) = IBP(I)
407:         end if
408:     170 continue
409:     do 180 I = NDEAD + 1, ILOST
410:         LL = L$DED(I)
411:         write(IOW,7000) LL, XXX(LL), YYY(LL), ZZZ(LL), AAA(LL),
412: &         BBB(LL), CCC(LL), KLSE(LL)
413:     180 continue
414:
415: 7000 format(1x,i6,' X=',1p,e12.5,' Y=',1p,e12.5,' Z=',1p,e12.5,' MU=',
416: &         e12.5,' ETA=',1p,e12.5,' XI=',1p,e12.5,' SURFACE=',1p,e12.5)
417:
418: C/#IF PARA(SX* CRAY)
419: *     call MVPSYNC_UNLOCK(2)
420: C/#ENDIF
421:     NDEAD = ILOST
422: C
423: C-----
424: C ..... DELETE LOST PARTICLES .....
425: C-----
426: C
427:     ISAFE = 0
428: *VOCL LOOP,NOVREC
429:     do 190 I = 1, NFFL(MZONE)
430:         if ( ILST(I).eq.0 ) then
431:             ISAFE = ISAFE + 1
432:             IBP(ISAFE) = IBP(I)
433:             DI(ISAFE) = DI(I)
434:             IKL2(ISAFE) = IKL2(I)
435:             if ( IHF.ne.0 ) then
436:                 ILUP(ISAFE) = ILUP(I)
437:                 IFC(ISAFE) = IFC(I)
438:                 XUP(ISAFE) = XUP(I)
439:                 YUP(ISAFE) = YUP(I)
440:                 ZUP(ISAFE) = ZUP(I)
441:                 AUP(ISAFE) = AUP(I)
442:                 BUP(ISAFE) = BUP(I)
443:                 CUP(ISAFE) = CUP(I)
444:             end if
445:         end if
446:     190 continue
447: C
448: C-----
449: C ..... GATHER UNLOST PARTICLES AGAIN .....
450: C-----
451: C
452:     do 200 I = 1, ISAFE
453:         X(I) = XXX(IBP(I))
454:         Y(I) = YYY(IBP(I))
455:         Z(I) = ZZZ(IBP(I))

```

```

456:         AI(I) = AAA(IBP(I))
457:         BI(I) = BBB(IBP(I))
458:         CI(I) = CCC(IBP(I))
459:         IGI(I) = IGG(IBP(I))
460:         W(I) = WWW(IBP(I))
461: 200 continue
462:     end if
463: C
464: C-----
465: C .... sampling of STG sphere position from
466: C .... Nearest Neighbour Distribution (NND)
467: C-----
468: C
469:     if ( JJJNND.ne.0 ) then
470:         MLT = LATNM(KZMAT(MZONE,1))
471:         NN = ISAFE
472: C
473: C .... ISTG (cell buffer zone of STG cell region) is returned
474: C .... IKL (cell position in STG lattice) is returned
475: C .... if sampled distance to STG is smaller than DI(I)
476: C
477: C XUP,YUP,ZUP here is not a coodinate in "upper level" in this
478: C case.
479: C
480:     call SMPSTG( IOW, MLT, X, Y, Z, AI, BI, CI, DI, ISTG, IBP,
481: &         NN, PNND, KNND, NPNND, IPLAT, KLATT, KSLAT, SZLAT,
482: &         NLATT, CELSZ, IPCEL, PPPF, KPPF, NPPPF,
483: &         IRAND, LEVL, DBNK, KDBNK, MDBNK, IBNK,
484: &         KIBNK, MIBNK, NBANK, XUP, YUP, ZUP, IKL, IBP2, TI, R )
485: C
486: C ... override frame distance flag
487: C
488:     do 210 I = 1, ISAFE
489:         if ( ISTG(I).ne.0 ) then
490:             IFC(I) = ISTG(I)
491:             AUP(I) = AI(I)
492:             BUP(I) = BI(I)
493:             CUP(I) = CI(I)
494:             ILUP(I) = LEVL(IBP(I))
495:         end if
496:     210 continue
497:     end if
498: C
499: C=====
500: C Sample flight distance on DLOC1
501: C and save path stretching factor on DLOC2 if necessary
502: C=====
503: C
504:     DAJUST = DEPS*0.1D0
505: C
506: C ... path-stretching (exponential transformation) is necessary or not
507: C
508:     JEXP = 0
509:     if ( MAT.gt.0.and.JNCOL.eq.0 ) then
510:         if ( JPSTR.eq.0
511: &         .or.
512: &         (JPSTR.ne.0.and.JTLLT.eq.0.and.PSALP(KZREG(MZONE)).eq.0.0) )
513:             then
514:                 JEXP = 0
515:             else
516:                 JEXP = 1
517:             end if
518: C
519:     call RANU2( IRAND, R, ISAFE, ICON )
520: C

```

src/gmvp/flione.f

```

521: C
522: C   ... no PATH-STRETCHING in this case
523: C
524: C       if ( JEXP.eq.0 ) then
525: C *VOCL LOOP,NOVREC
526: C       do 220 I = 1, ISAFE
527: C /#IF .NOT.SSL2
528: C         DLOC1(I) = -LOG(R(I)+SMALL) /STOTX(IGI(I),MAT)
529: C /#ELSE
530: C         DLOC1(I) = -LOG(R(I)) /STOTX(IGI(I),MAT)
531: C /#ENDIF
532: C /#IF .NOT.RANDOM( BOUND )
533: C       if ( DLOC1(I).le.0.0 ) DLOC1(I) = DAJUST
534: C /#ENDIF
535: C       220 continue
536: C
537: C   ... PATH-STRETCHING is necessary
538: C
539: C   else
540: C *VOCL LOOP,NOVREC
541: C       do 230 I = 1, ISAFE
542: C       ... IBREG is always set (from Jan 2000)
543: C       IRG1 = KZREG(MZONE)
544: C
545: C       ... get absolute (level 0) coordinates and direction
546: C       (assuming JFISX>0 for path stretching)
547: C
548: C       DXXX = X(I)
549: C       DYYY = Y(I)
550: C       DZZZ = Z(I)
551: C       DAAA = AI(I)
552: C       DBBB = BI(I)
553: C       DCCC = CI(I)
554: C       if ( JLATT.gt.0 ) then
555: C         LVL = LEVL(IBP(I))
556: C         if ( LVL.gt.0 ) then
557: C           KD4 = KDBNK(4)
558: C           KD6 = KDBNK(6)
559: C           DDDD = DBNK(IBP(I),KD4)
560: C           DAAA = DBNK(IBP(I),KD6+3)
561: C           DBBB = DBNK(IBP(I),KD6+4)
562: C           DCCC = DBNK(IBP(I),KD6+5)
563: C           DXXX = DBNK(IBP(I),KD6) - DDDD*DAAA
564: C           DYYY = DBNK(IBP(I),KD6+1) - DDDD*DBBB
565: C           DZZZ = DBNK(IBP(I),KD6+2) - DDDD*DCCC
566: C         end if
567: C       end if
568: C
569: C       DLL =
570: C       &      SQRT((DXXX-PSXYZ(1))*2+(DYYY-PSXYZ(2))*2
571: C       &      +(DZZZ-PSXYZ(3))*2)
572: C       DOO = 0.0D0
573: C       if ( DLL.ne.0.0D0 ) then
574: C         DOO =
575: C         &      (DAAA*(DXXX-PSXYZ(1))+DBBB*(DYYY-PSXYZ(2))
576: C         &      +DCCC*(DZZZ-PSXYZ(3))) /DLL
577: C       end if
578: C       DOO = PSALP(IRG1)*DOO
579: C       if ( ABS(DOO).gt.1.0D0 ) DOO = SIGN(0.99999D0,DOO)
580: C
581: C /#IF .NOT.SSL2
582: C       DLOC1(I) = -LOG(R(I)+SMALL) /STOTX(IGI(I),MAT) /
583: C       &      (1.0D0-DOO)
584: C /#ELSE
585: C       DLOC1(I) = -LOG(R(I)) /STOTX(IGI(I),MAT) /(1.0D0-DOO)

```

```

586: C /#ENDIF
587: C /#IF .NOT.RANDOM( BOUND )
588: C       if ( DLOC1(I).le.0.0 ) DLOC1(I) = DAJUST
589: C /#ENDIF
590: C       DLOC2(I) = DOO
591: C       230 continue
592: C       end if
593: C       end if
594: C
595: C   ... store IBNK(*,KIBNK(5)) for use in surface crossing tally
596: C
597: C       if ( JTSRF.ne.0 ) then
598: C         do 240 I = 1, ISAFE
599: C           IBP2(I) = IBNK(IBP(I),KIBNK(5))
600: C         240 continue
601: C       end if
602: C
603: C   ... count up path flight counter here (for collided particle
604: C   it is cleared to zero afterwards)
605: C
606: C       KI5 = KIBNK(5)
607: C       KI7 = KIBNK(7)
608: C       if ( KI5.ne.0 ) then
609: C *VOCL LOOP,NOVREC
610: C       do 250 I = 1, ISAFE
611: C         if ( IBNK(IBP(I),KI5).lt.0 ) then
612: C           IBNK(IBP(I),KI5) = 1
613: C         else
614: C           IBNK(IBP(I),KI5) = IBNK(IBP(I),KI5) + 1
615: C         end if
616: C       250 continue
617: C       end if
618: C
619: C -----
620: C   ... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) ....
621: C   (non-time dependent)
622: C -----
623: C
624: C       if ( JTIME.eq.0 ) then
625: C         ICOLS = 0
626: C
627: C =====
628: C =====
629: C ===== When hexagonal lattice or overlapping lattice is used
630: C =====
631: C =====
632: C
633: C       if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
634: C         INZ1 = NNXT(NZ1)
635: C
636: C   ... MAT > 0 : REAL MEDIUM
637: C
638: C       if ( MAT.gt.0.and.JNCOL.eq.0 ) then
639: C         INXT = 0
640: C         IHH = 0
641: C *VOCL LOOP,NOVREC
642: C       do 260 I = 1, ISAFE
643: C         D1 = DLOC1(I)
644: C
645: C         ..... COLLISION .....
646: C
647: C       if ( DI(I).gt.D1 ) then
648: C         DI(I) = D1
649: C         ICOLS = ICOLS + 1
650: C         LSCOL(NCOLS+ICOLS) = IBP(I)

```

src/gmvp/flione.f

```

651: C      .... NOT ON-SURFACE
652: C      IKL2(I) = 0
653: C
654: C      ... flight path count is cleared for collision
655: C      if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
656: C      ... next STG sampling is NND-2
657: C      if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
658: C
659: C      ..... AT BOUNDARY .....
660: C
661: C      else
662: C      IZNN = MZONE
663: C
664: C      ---- IHF = 1 : ZONE 'MZONE' IS IN A HEXAGONAL CELL. ----
665: C      if ( IHF.ne.0 ) then
666: C      if ( IFC(I).ne.0 ) then
667: C      IHH = IHH + 1
668: C      IZNN = IFC(I)
669: C      LEVL(IBP(I)) = ILUP(I)
670: C
671: C      .... make IFC(I) negative for calculation of NNXT
672: C      IFC(I) = -IFC(I)
673: C      X(I) = XUP(I)
674: C      Y(I) = YUP(I)
675: C      Z(I) = ZUP(I)
676: C      AI(I) = AUP(I)
677: C      BI(I) = BUP(I)
678: C      CI(I) = CUP(I)
679: C      ..... NOT ON-SURFACE
680: C      IKL2(I) = 0
681: C      end if
682: C      end if
683: C
684: C      INXT = INXT + 1
685: C      LSSRC(INZ1+INXT) = IBP(I)
686: C      IZNXT(INZ1+INXT) = IZNN
687: C      end if
688: C      260 continue
689: C
690: C      if ( IHF.eq.0 ) then
691: C      NNXT(MZONE) = NNXT(MZONE) + INXT
692: C      else
693: C      NNXT(MZONE) = NNXT(MZONE) + INXT - IHH
694: C
695: C      if ( IHH.gt.0 ) then
696: C      *VOCL LOOP,SCALAR
697: C      do 270 I = 1, ISAFE
698: C      if ( IFC(I).lt.0 ) NNXT(-IFC(I)) =
699: C      & NNXT(-IFC(I)) + 1
700: C      270 continue
701: C
702: C      ... "cell position" LPOS is changed here for STG cell
703: C
704: C      if ( JJJNND.ne.0 ) then
705: C      *VOCL LOOP,NOVREC
706: C      do 280 I = 1, ISAFE
707: C      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
708: C      LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
709: C      end if
710: C      280 continue
711: C      end if
712: C      if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
713: C      *VOCL LOOP,NOVREC
714: C      do 290 I = 1, ISAFE
715: C      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then

```

```

716: C      IBSPC(IBP(I),LEVL(IBP(I))) =
717: C      & KTCSP(
718: C      & KSPSU(KZREG(MZONE),
719: C      & IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
720: C      end if
721: C      290 continue
722: C      end if
723: C      end if
724: C      end if
725: C
726: C      .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
727: C      .... JNCOL .ne.0 : "collisionless" mode
728: C
729: C      else
730: C      INXT = ISAFE
731: C      if ( IHF.eq.0 ) then
732: C      do 300 I = 1, INXT
733: C      LSSRC(INZ1+I) = IBP(I)
734: C      IZNXT(INZ1+I) = MZONE
735: C      300 continue
736: C      NNXT(MZONE) = NNXT(MZONE) + INXT
737: C      else
738: C      IHHHH = 0
739: C      *VOCL LOOP,NOVREC
740: C      do 310 I = 1, INXT
741: C      LSSRC(INZ1+I) = IBP(I)
742: C
743: C      if ( IFC(I).eq.0 ) then
744: C      IZNXT(INZ1+I) = MZONE
745: C      else
746: C      IHHHH = IHHHH + 1
747: C      IZNXT(INZ1+I) = IFC(I)
748: C      LEVL(IBP(I)) = ILUP(I)
749: C
750: C      .... make IFC(I) negative for calculation of NNXT
751: C      IFC(I) = -IFC(I)
752: C      X(I) = XUP(I)
753: C      Y(I) = YUP(I)
754: C      Z(I) = ZUP(I)
755: C      AI(I) = AUP(I)
756: C      BI(I) = BUP(I)
757: C      CI(I) = CUP(I)
758: C
759: C      ..... NOT ON-SURFACE
760: C
761: C      IKL2(I) = 0
762: C      end if
763: C      310 continue
764: C
765: C      NNXT(MZONE) = NNXT(MZONE) + INXT - IHHHH
766: C
767: C      if ( IHHHH.gt.0 ) then
768: C      *VOCL LOOP,SCALAR
769: C      do 320 I = 1, INXT
770: C      if ( IFC(I).lt.0 ) NNXT(-IFC(I)) =
771: C      & NNXT(-IFC(I)) + 1
772: C      320 continue
773: C
774: C      ... "cell position" LPOS is changed here for STG cell
775: C
776: C      if ( JJJNND.ne.0 ) then
777: C      *VOCL LOOP,NOVREC
778: C      do 330 I = 1, INXT
779: C      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
780: C      LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)

```

src/gmvp/flione.f

```

781:                end if
782: 330            continue
783:            end if
784:            if ( JTLT.ne.0.and.JJNND.ne.0 ) then
785: *VOCL LOOP,NOVREC
786:                do 340 I = 1, INXT
787:                    if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
788:                        IBSPC(IBP(I),LEVL(IBP(I))) =
789: &                    KTCSPI
790: &                    KSPSU(KZREG(MZONE),
791: &                    IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
792:                    end if
793: 340            continue
794:                end if
795:            end if
796:        end if
797:    end if
798: C
799:    NCOLS = NCOLS - ICOLS
800:    NNXT(NZ1) = NNXT(NZ1) + INXT
801: C
802: C =====
803: C =====
804: C ===== When no hexagonal lattice or overlapping lattice is used
805: C =====
806: C =====
807: C
808:     else
809:         INXT = 0
810: C
811: C .... MAT > 0 : REAL MEDIUM .....
812: C
813:         if ( MAT.gt.0.and.JNCOL.eq.0 ) then
814: *VOCL LOOP,NOVREC
815:             do 350 I = 1, ISAFE
816:                 D1 = DLOC1(I)
817:                 if ( DI(I).gt.D1 ) then
818:                     DI(I) = D1
819:                     ICOLS = ICOLS + 1
820:                     LSCOL(NCOLS+ICOLS) = IBP(I)
821:
822: C .... NOT ON-SURFACE
823:
824:                     IKL2(I) = 0
825: C
826: C ... flight path count is cleared for collision
827:                 if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
828: C ... next STG sampling is NND-2
829:                 if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
830:             else
831:                 INXT = INXT + 1
832:                 LSSRC(NNXT(NZ1)+INXT) = IBP(I)
833:             end if
834: 350            continue
835: C
836: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
837: C
838:         else
839:             INXT = ISAFE
840:             do 360 I = 1, INXT
841:                 LSSRC(NNXT(NZ1)+I) = IBP(I)
842: 360            continue
843:             end if
844: C
845:             NCOLS = NCOLS + ICOLS

```

```

846:
847:         do 370 I = 1, INXT
848:             IZNXT(NNXT(NZ1)+I) = MZONE
849: 370            continue
850:
851:             NNXT(NZ1) = NNXT(NZ1) + INXT
852:             NNXT(MZONE) = NNXT(MZONE) + INXT
853:         end if
854: C
855: C -----
856: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) ....
857: C (time dependent)
858: C -----
859: C
860:         else
861:             ICOLS = 0
862: C
863: C ... time remained until time-cutoff
864: C
865:             do 380 I = 1, ISAFE
866:                 TI(I) = TCUT - TTT(IBP(I))
867: 380            continue
868: C
869: C ----- HEXAGONAL LATTICE OPTION -----
870: C
871:             if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
872:                 INZ1 = NNXT(NZ1)
873: C
874: C .... MAT > 0 : REAL MEDIUM
875: C
876:                 if ( MAT.gt.0.and.JNCOL.eq.0 ) then
877:                     INXT = 0
878:                     IHH = 0
879:                     ITTCUT = 0
880: *VOCL LOOP,NOVREC
881:                     do 390 I = 1, ISAFE
882:                         D1 = DLOC1(I)
883:                         D2 = TI(I)*VEL(IGI(I))
884:                         if ( JPTIM.ne.0 ) D2 = DINF
885: C
886: C ..... time cutoff .....
887: C
888:                     if ( D2.lt.MIN(DI(I),D1) ) then
889:                         DI(I) = D2
890:                         NDEAD = NDEAD + 1
891:                         LSDED(NDEAD) = IBP(I)
892:                         NCNTR(26) = NCNTR(26) + 1
893:                         WCNTR(26) = WCNTR(26) + W(I)
894:                         ITTCUT = ITTCUT + 1
895: C
896: C ..... COLLISION .....
897: C
898:                     else if ( DI(I).gt.D1 ) then
899:                         DI(I) = D1
900:                         ICOLS = ICOLS + 1
901:                         LSCOL(NCOLS+ICOLS) = IBP(I)
902: C .... NOT ON-SURFACE
903:                         IKL2(I) = 0
904: C
905: C ... flight path count is cleared for collision
906:                     if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
907: C ... next STG sampling is NND-2
908:                     if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
909: C
910: C ..... reached boundary .....

```


src/gmvp/flione.f

```

911: C
912:       else
913:         IZNN = MZONE
914: C
915: C ---- IHF = 1 : ZONE 'MZONE' IS IN A HEXAGONAL CELL. ----
916:       if ( IHF.ne.0.and.IFC(I).ne.0 ) then
917:         IHH = IHH + 1
918:         IZNN = IFC(I)
919: CCC       LEVL(IBP(I)) = LEVL(IBP(I)) - 1
920:         LEVL(IBP(I)) = ILUP(I)
921: C
922: C .... make IFC(I) negative for calculation of NNXT
923:       IFC(I) = -IFC(I)
924:       X(I) = XUP(I)
925:       Y(I) = YUP(I)
926:       Z(I) = ZUP(I)
927:       AI(I) = AUP(I)
928:       BI(I) = BUP(I)
929:       CI(I) = CUP(I)
930: C ..... NOT ON-SURFACE
931:       IKL2(I) = 0
932:       end if
933: C
934:       INXT = INXT + 1
935:       LSSRC(INZ1+INXT) = IBP(I)
936:       IZNXT(INZ1+INXT) = IZNN
937:       end if
938: 390       continue
939:
940:       NNXT(MZONE) = NNXT(MZONE) + INXT - IHH
941:       if ( IHF.ne.0 ) then
942:         if ( IHH.gt.0 ) then
943: *VOCL LOOP,SCALAR
944:           do 400 I = 1, ISAFE
945:             if ( IFC(I).lt.0 ) NNXT(-IFC(I)) =
946: &             NNXT(-IFC(I)) + 1
947: 400           continue
948: C
949: C ... "cell position" LPOS is changed here for STG cell
950: C
951:       if ( JJJNND.ne.0 ) then
952: *VOCL LOOP,NOVREC
953:         do 410 I = 1, ISAFE
954:           if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
955:             LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
956:           end if
957: 410         continue
958:       end if
959:       if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
960: *VOCL LOOP,NOVREC
961:         do 420 I = 1, ISAFE
962:           if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
963:             IBSPC(IBP(I),LEVL(IBP(I))) =
964: &             KTCSP(
965: &             KSPSU(KZREG(MZONE),
966: &             IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
967:           end if
968: 420         continue
969:       end if
970:       end if
971:     end if
972: C
973: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
974: C
975:       else

```

```

976:       INXT = 0
977:       ITTCUT = 0
978:       if ( IHF.eq.0 ) then
979:         do 430 I = 1, ISAFE
980:           D2 = TI(I)*VEL(IGI(I))
981:           if ( JPTIM.ne.0 ) D2 = DINF
982: C
983: C ..... time cutoff .....
984: C
985:           if ( D2.lt.DI(I) ) then
986:             DI(I) = D2
987:             NDEAD = NDEAD + 1
988:             LSDED(NDEAD) = IBP(I)
989:             NCNTR(26) = NCNTR(26) + 1
990:             WCNTR(26) = WCNTR(26) + W(I)
991:             ITTCUT = ITTCUT + 1
992:           else
993:             INXT = INXT + 1
994:             LSSRC(INZ1+INXT) = IBP(I)
995:             IZNXT(INZ1+INXT) = MZONE
996:           end if
997: 430         continue
998:         NNXT(MZONE) = NNXT(MZONE) + INXT
999:       else
1000:         IHHHH = 0
1001: *VOCL LOOP,NOVREC
1002:         do 440 I = 1, ISAFE
1003:           D2 = TI(I)*VEL(IGI(I))
1004:           if ( JPTIM.ne.0 ) D2 = DINF
1005: C
1006: C ..... time cutoff .....
1007: C
1008:           if ( D2.lt.DI(I) ) then
1009:             DI(I) = D2
1010:             NDEAD = NDEAD + 1
1011:             LSDED(NDEAD) = IBP(I)
1012:             NCNTR(26) = NCNTR(26) + 1
1013:             WCNTR(26) = WCNTR(26) + W(I)
1014:             ITTCUT = ITTCUT + 1
1015:           else
1016:             IZNN = MZONE
1017:             if ( IFC(I).ne.0 ) then
1018:               IZNN = IFC(I)
1019:               IHHHH = IHHHH + 1
1020: CCC       LEVL(IBP(I)) = LEVL(IBP(I)) - 1
1021:             LEVL(IBP(I)) = ILUP(I)
1022: C
1023: C .... make IFC(I) negative for calculation of NNXT
1024:             IFC(I) = -IFC(I)
1025:             X(I) = XUP(I)
1026:             Y(I) = YUP(I)
1027:             Z(I) = ZUP(I)
1028:             AI(I) = AUP(I)
1029:             BI(I) = BUP(I)
1030:             CI(I) = CUP(I)
1031: C ..... NOT ON-SURFACE
1032:             IKL2(I) = 0
1033:             end if
1034:
1035:             INXT = INXT + 1
1036:             LSSRC(INZ1+INXT) = IBP(I)
1037:             IZNXT(INZ1+INXT) = IZNN
1038:           end if
1039: 440         continue
1040:

```

src/gmvp/flione.f

```

1041:      NNXT(MZONE) = NNXT(MZONE) + INXT - IHFFF
1042:
1043:      if ( IHFFF.gt.0 ) then
1044: *VOCL LOOP,SCALAR
1045:      do 450 I = 1, ISAFE
1046:      if ( IFC(I).lt.0 ) NNXT(-IFC(I)) =
1047:      &      NNXT(-IFC(I)) + 1
1048:      450      continue
1049: C
1050: C      ... "cell position" LPOS is changed here for STG cell
1051: C
1052:      if ( JJJNND.ne.0 ) then
1053: *VOCL LOOP,NOVREC
1054:      do 460 I = 1, ISAFE
1055:      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
1056:      LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
1057:      end if
1058:      460      continue
1059:      end if
1060:      if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
1061: *VOCL LOOP,NOVREC
1062:      do 470 I = 1, ISAFE
1063:      if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
1064:      IBSPC(IBP(I),LEVL(IBP(I))) =
1065:      &      KTCSP(
1066:      &      KSPSU(KZREG(MZONE),
1067:      &      IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
1068:      end if
1069:      470      continue
1070:      end if
1071:      end if
1072:      end if
1073:      end if
1074: C
1075:      NCOLS = NCOLS + ICOLS
1076:      NNXT(NZ1) = NNXT(NZ1) + INXT
1077: C
1078: C ---- not in HEXAGONAL LATTICE ----
1079: C
1080:      else
1081:      INXT = 0
1082:      ITTCUT = 0
1083: C
1084: C .... MAT > 0 : REAL MEDIUM .....
1085: C
1086:      if ( MAT.gt.0.and.JNCOL.eq.0 ) then
1087: *VOCL LOOP,NOVREC
1088:      do 480 I = 1, ISAFE
1089:      D1 = DLOC1(I)
1090:      D2 = TI(I)*VEL(IGI(I))
1091:      if ( JPTIM.ne.0 ) D2 = DINF
1092:      if ( D2.lt.MIN(DI(I),D1) ) then
1093:      DI(I) = D2
1094:      NDEAD = NDEAD + 1
1095:      LSDED(NDEAD) = IBP(I)
1096:      NCNTR(26) = NCNTR(26) + 1
1097:      WCNTR(26) = WCNTR(26) + W(I)
1098:      ITTCUT = ITTCUT + 1
1099:      else if ( DI(I).gt.D1 ) then
1100:      DI(I) = D1
1101:      ICOLS = ICOLS + 1
1102:      LSCOL(NCOLS+ICOLS) = IBP(I)
1103: C
1104: C      .... NOT ON-SURFACE
1105: C

```

```

1106:      IKL2(I) = 0
1107: C
1108: C      ... flight path count is cleared for collision
1109:      if ( KI5.ne.0 ) IBNK(IBP(I),KI5) = 0
1110: C      ... next STG sampling is NND-2
1111:      if ( KI7.ne.0 ) IBNK(IBP(I),KI7) = 2
1112:      else
1113:      INXT = INXT + 1
1114:      LSSRC(NNXT(NZ1)+INXT) = IBP(I)
1115:      IZNXT(NNXT(NZ1)+INXT) = MZONE
1116:      end if
1117:      480      continue
1118: C
1119: C .... MAT = 0 : INNER VOID. TRANSFER ALL PARTICLES TO SEARCH STACK ...
1120: C
1121:      else
1122:      INXT = 0
1123:      do 490 I = 1, ISAFE
1124:      D2 = TI(I)*VEL(IGI(I))
1125:      if ( JPTIM.ne.0 ) D2 = DINF
1126:      if ( D2.lt.DI(I) ) then
1127:      DI(I) = D2
1128:      NDEAD = NDEAD + 1
1129:      LSDED(NDEAD) = IBP(I)
1130:      NCNTR(26) = NCNTR(26) + 1
1131:      WCNTR(26) = WCNTR(26) + W(I)
1132:      ITTCUT = ITTCUT + 1
1133:      else
1134:      INXT = INXT + 1
1135:      LSSRC(NNXT(NZ1)+INXT) = IBP(I)
1136:      IZNXT(NNXT(NZ1)+INXT) = MZONE
1137:      end if
1138:      490      continue
1139:      end if
1140: C
1141:      NCOLS = NCOLS + ICOLS
1142:      NNXT(MZONE) = NNXT(MZONE) + INXT
1143:      NNXT(NZ1) = NNXT(NZ1) + INXT
1144:      end if
1145: C
1146: C      ... flight time ...
1147: C
1148:      do 500 I = 1, ISAFE
1149:      TI(I) = DI(I) /VEL(IGI(I))
1150:      TFLH(1) = TFLH(1) + TI(I)*W(I)
1151:      500      continue
1152: C
1153:      end if
1154: C
1155: C-----
1156: C .... MOVE PARTICLES TO NEXT POINTS AND PUT VALUES IN BANK.....
1157: C ( Some particles may be already cut-off in time dependent problem )
1158: C-----
1159: C
1160: *VOCL LOOP,NOVREC
1161:      do 510 I = 1, ISAFE
1162:      XXX(IBP(I)) = X(I) + AI(I)*DI(I)
1163:      YYY(IBP(I)) = Y(I) + BI(I)*DI(I)
1164:      ZZZ(IBP(I)) = Z(I) + CI(I)*DI(I)
1165:      KLSF(IBP(I)) = IKL2(I)
1166:      510      continue
1167: C
1168:      if ( IHF.ne.0 ) then
1169: *VOCL LOOP,NOVREC
1170:      do 520 I = 1, ISAFE

```

src/gmvp/flione.f

```

1171:      AAA(IBP(I)) = AI(I)
1172:      BBB(IBP(I)) = BI(I)
1173:      CCC(IBP(I)) = CI(I)
1174: 520      continue
1175:      end if
1176: C
1177: C-----
1178: C .... adjust weights for path-stretching mode .....
1179: C      (is it right correcting here ?)
1180: C-----
1181: C
1182:      if ( JEXP.ne.0 ) then
1183: *VOCL LOOP,NOVREC
1184:      do 530 I = 1, ISAFE
1185: C
1186: C      ... collided in this zone ...
1187: C      if ( DLOC1(I).le.DI(I) ) then
1188: C          DEE = EXP(-DLOC2(I)*STOTX(IGI(I),MAT)*DLOC1(I)) /
1189: C      &      (1.0D0-DLOC2(I))
1190: C      else
1191: C
1192: C      ... escaped from this zone or cutoff ...
1193: C          DEE = EXP(-DLOC2(I)*STOTX(IGI(I),MAT)*DI(I))
1194: C      end if
1195: C          W(I) = W(I)*DEE
1196: C          WWW(IBP(I)) = W(I)
1197: 530      continue
1198:      end if
1199: C
1200: C-----
1201: C .... reduce weights for "collisionless" mode .....
1202: C-----
1203: C
1204:      if ( JNCOL.eq.1.and.MAT.gt.0 ) then
1205: *VOCL LOOP,NOVREC
1206:      do 540 I = 1, ISAFE
1207: C          WWW(IBP(I)) = W(I)*EXP(-DI(I)*STOTX(IGI(I),MAT))
1208: 540      continue
1209:      end if
1210: C
1211: C-----
1212: C .... collect data for tallies (region # etc.)
1213: C-----
1214: C
1215: C
1216:      if ( NTEVE.gt.0.and.(JTEVE(2).gt.0.or.JTEVE(3).gt.0) ) then
1217: C
1218: C      ..... GATHER REGION # ...
1219: C
1220: C          JREGN = 0
1221: C
1222: C          if ( JTLLT.ne.0.and.(KCELL(MZONE).ne.0.or.JJNNND.ne.0) ) then
1223: C              do 550 I = 1, ISAFE
1224: C                  IRG(I) = IBREG(IBP(I))
1225: 550          continue
1226: C
1227: C      ..... CHECK MULTI REGION / SINGLE REGION .....
1228: C          ( JREGN = 1 / 0 )
1229: C
1230: C          do 560 I = 2, ISAFE
1231: C              if ( IRG(I).ne.IRG(1) ) go to 570
1232: 560          continue
1233: C
1234: C      ..... ALL PARTICLES ARE IN THE SAME REGION. ....
1235: C          go to 580

```

```

1236:
1237: 570      JREGN = 1
1238:
1239: 580      continue
1240:      else
1241: C          do 590 I = 1, ISAFE
1242: C              IRG(I) = KZREG(MZONE)
1243: 590      continue
1244:      end if
1245: C
1246: C          NNEUT = 0
1247: C          NPHOT = 0
1248: C          if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
1249: C              do 600 I = 1, ISAFE
1250: C                  if ( IGI(I).le.NGP1 ) then
1251: C                      NNEUT = NNEUT + 1
1252: C                  else
1253: C                      NPHOT = NPHOT + 1
1254: C                  end if
1255: 600      continue
1256: C          else if ( JNEUT.ne.0 ) then
1257: C              NNEUT = ISAFE
1258: C          else if ( JPHOT.ne.0 ) then
1259: C              NPHOT = ISAFE
1260: C          end if
1261:      end if
1262: C
1263: C-----
1264: C .... Take surface tallies .....
1265: C-----
1266: C
1267: C      variables used for surface tally:
1268: C
1269: C      X(*),Y(*),Z(*) : particle position in "local" coordinates
1270: C      AI(*),BI(*),CI(*) : particle direction in "local" coordinates
1271: C      (position and direction may be changed to those of
1272: C      "global" (absolute) coordinates)
1273: C      TI(*) : time of flight
1274: C
1275: C      DI(I) : flight distance to next collision or boundary crossing point
1276: C      W(I) : particle weight
1277: C      (DI and W must not be changed their value, for use in track length
1278: C      tally.)
1279: C
1280: C      and other pre-collected data:
1281: C
1282: C      IBP(I) : bank pointers for particles to be tallied
1283: C
1284: C      IBP2(I) : flight path counter value (IBNK(*,KIBNK(5)) before
1285: C      change in collision detection procedure.
1286: C
1287: C      IGI(I) : energy group(bin) #
1288: C      IRG(I) : region #
1289: C
1290: C      available working arrays (see WRKARY routine):
1291: C
1292: C      (without any condition)
1293: C
1294: C      integer : IKL(NBANK), IKL2(NBANK), ILST(NBANK)
1295: C      real : R(3*)
1296: C      double precision: DLOC1, DLOC2
1297: C
1298: C      (when JTSRF.ne.0 or other condition)
1299: C      double precision: XUP,YUP,ZUP, AUP, BUP, CUP
1300: C      XYZ1(*,3),XYZ2(*,3), ABC1(*,3), ABC2(*,3)

```

src/gmvp/flione.f

```

1301: C          DDI
1302: C  integer : ISTG, ILUP, IBP0, IBP1, IBP2
1303: C
1304: C
1305: C      if ( NTEVE.gt.0.and.JTEVE(3).gt.0 ) then
1306: C
1307: C          do 740 ITS = 1, NTSRF
1308: C              KI10 = KIBNK(10) + ITS - 1
1309: C              KD7 = KDBNK(7) + ITS - 1
1310: C              ... first flight after birth or collision
1311: C              set surface flag to 3
1312: C              ICROSS = 0
1313: C          *VOCL LOOP,NOVREC
1314: C          do 610 I = 1, ISAFE
1315: C              if ( IBP2(I).le.0 ) then
1316: C                  IBNK(IBP(I),KI10) = 3
1317: C              end if
1318: C              if ( IBNK(IBP(I),KI10).ne.0 ) then
1319: C                  ICROSS = ICROSS + 1
1320: C              end if
1321: C          610      continue
1322: C
1323: C      ... all flight path never cross this surface !!
1324: C      if ( ICROSS.eq.0 ) go to 740
1325: C
1326: C      ICROSS = 0
1327: C  *VOCL LOOP,NOVREC
1328: C      do 620 I = 1, ISAFE
1329: C          if ( IBNK(IBP(I),KI10).ne.0 ) then
1330: C              ICROSS = ICROSS + 1
1331: C              XUP(ICROSS) = X(I)
1332: C              YUP(ICROSS) = Y(I)
1333: C              ZUP(ICROSS) = Z(I)
1334: C              AUP(ICROSS) = AI(I)
1335: C              BUP(ICROSS) = BI(I)
1336: C              CUP(ICROSS) = CI(I)
1337: C              DDI(ICROSS) = DI(I)
1338: C              if ( JTIME.ne.0 ) then
1339: C                  DLOC1(ICROSS) = TTT(IBP(I))
1340: C                  DLOC2(ICROSS) = VEL(IGI(I))
1341: C              end if
1342: C
1343: C              IBP1(ICROSS) = IBP(I)
1344: C              IBP0(ICROSS) = I
1345: C          end if
1346: C      620      continue
1347: C
1348: C      ... convert to absolute (level 0) coordinates if necessary
1349: C
1350: C      if ( JFISX.ne.0.and.JLATT.gt.0 ) then
1351: C          KD4 = KDBNK(4)
1352: C          KD6 = KDBNK(6)
1353: C      *VOCL LOOP,NOVREC
1354: C      do 630 I = 1, ICROSS
1355: C          LVL = LEVL(IBP1(I))
1356: C          if ( LVL.gt.0 ) then
1357: C              DDDD = DBNK(IBP1(I),KD4)
1358: C              AUP(I) = DBNK(IBP1(I),KD6+3)
1359: C              BUP(I) = DBNK(IBP1(I),KD6+4)
1360: C              CUP(I) = DBNK(IBP1(I),KD6+5)
1361: C              XUP(I) = DBNK(IBP1(I),KD6) - DDDD*AUP(I)
1362: C              YUP(I) = DBNK(IBP1(I),KD6+1) - DDDD*BUP(I)
1363: C              ZUP(I) = DBNK(IBP1(I),KD6+2) - DDDD*CUP(I)
1364: C          end if
1365: C      630      continue

```

```

1366: C          end if
1367: C
1368: C          ITSF = ICROSS
1369: C
1370: C      ... surface crossing judgement may need more than
1371: C      one cycle per surface, because a flight path may
1372: C      cross a surface more than once!!
1373: C
1374: C      ISCYCL = 0
1375: C      640      continue
1376: C      ISCYCL = ISCYCL + 1
1377: C      if ( ISCYCL.gt.32 ) then
1378: C          write(IOW,*)
1379: C          &          '!!!(FLIONE) Too many cycles for tally-surface',
1380: C          &          ' crossing judgement. Surface: ', ITS
1381: C          write(IOW,*) ' Number of remaining particle : ', ITSF
1382: C          go to 740
1383: C      end if
1384: C
1385: C      ... check distance to tally-surface
1386: C
1387: C      call FLSURF( ITS, NTSRF, KTSRF, ITSRF, IDSRF, ITSF, IBP1,
1388: C          &          NBANK, XUP, YUP, ZUP, AUP, BUP, CUP, SDA, NSDA,
1389: C          &          DINF, DEFS, ANGLB, NANGLE, DBNK, KDBNK, MDBNK, IBNK,
1390: C          &          KIBNK, MIBNK, XYZ1(1,1), XYZ1(1,2), XYZ1(1,3),
1391: C          &          ABC1(1,1), ABC1(1,2), ABC1(1,3), R(1), XYZ2(1,1),
1392: C          &          XYZ2(1,2), XYZ2(1,3), ABC2(1,1), ABC2(1,2),
1393: C          &          ABC2(1,3), ILUP, IKL, IKL2, ISTG )
1394: C
1395: C      ... check crossing path
1396: C
1397: C      ICROSS = 0
1398: C  *VOCL LOOP,NOVREC
1399: C      do 650 I = 1, ITSF
1400: C          III = IBP1(I)
1401: C          IFSS = IBNK(III,KI10)
1402: C
1403: C          DD0 = DBNK(III,KD7)
1404: C          DBNK(III,KD7) = DD0 - DDI(I)
1405: C
1406: C      CM2016      if ( IFSS.ne.0.and.DBNK(III,KD7).lt.0.0d0 ) then
1407: C          DERR = DD0 * 1.0d-8
1408: C          if ( IFSS.ne.0.and.DBNK(III,KD7).le.DERR ) then
1409: C              ICROSS = ICROSS + 1
1410: C
1411: C          ... crossing from outside to inside
1412: C
1413: C          if ( IFSS.lt.0 ) then
1414: C              IBNK(III,KI10) = +2
1415: C
1416: C          ... crossing from inside to outside
1417: C          ... IFSS == 1
1418: C          else
1419: C              IBNK(III,KI10) = -2
1420: C          end if
1421: C
1422: C      ... time of crossing (DLOC2: velocity)
1423: C      if ( JTIME.ne.0 ) then
1424: C          DLOC1(I) = DLOC1(I) + DD0/DLOC2(I)
1425: C      end if
1426: C
1427: C      XUP(ICROSS) = XUP(I) + DD0*AUP(I)
1428: C      YUP(ICROSS) = YUP(I) + DD0*BUP(I)
1429: C      ZUP(ICROSS) = ZUP(I) + DD0*CUP(I)
1430: C      AUP(ICROSS) = AUP(I)

```

src/gmvp/flione.f

```

1431:      BUP(ICROSS) = BUP(I)
1432:      CUP(ICROSS) = CUP(I)
1433:      DDI(ICROSS) = DDI(I) - DD0
1434:      if ( JTIME.ne.0 ) then
1435:         DLOC1I = DLOC1(I) + DD0/DLOC2(I)
1436:         DLOC1(ICROSS) = DLOC1I
1437:         DLOC2(ICROSS) = DLOC2(I)
1438:      end if
1439:      IBP1(ICROSS) = III
1440:      IBP0(ICROSS) = IBP0(I)
1441: C
1442:      end if
1443: C
1444: 650      continue
1445: C
1446:      if ( ICROSS.eq.0 ) go to 740
1447: C
1448:      ITSF = ICROSS
1449: C
1450: C-----
1451: C ... take surface tally, huhhhhh ...
1452: C-----
1453: C
1454: C ... flag set after time bin check ...
1455:      ITCHK = 0
1456: C
1457:      do 730 J = 0, JTEVE(3) - 1
1458: C ... pointer to direct tally (idt) & d-tally # (it) ...
1459:         IDT = KTEVE(LTEVE(3)+J) - 1
1460:         IT = IDTAL(IDT+9)
1461: C ... this tally is not for this surface ;-) ...
1462:         if ( ABS(IDTAL(IDT+11)).ne.ITS ) go to 730
1463: C
1464: C ... neutron or photon ?
1465:         if ( IDTAL(IDT+5).eq.1.and.NNEUT.eq.0 ) go to 730
1466:         if ( IDTAL(IDT+5).eq.2.and.NPHOT.eq.0 ) go to 730
1467: C
1468: C ... skip if current region does not need this tally ;-) ...
1469: C
1470:         if ( JREGN.eq.0 ) then
1471:            if ( JDTRG(IRG(IBP0(1)),IT).eq.0 ) go to 730
1472:         else
1473:            do 660 I = 1, ITSF
1474:               if ( JDTRG(IRG(IBP0(I)),IT).ne.0 ) go to 670
1475: 660          continue
1476:            go to 730
1477: 670          continue
1478:            end if
1479: C
1480: C ... calculate time bin on crossing
1481: C
1482:         if ( JTIME.ne.0.and.ITCHK.eq.0 ) then
1483:            ITC2 = 0
1484:            IDT2 = IDT + IDTOFF
1485:            NNBIN = IDTAL(IDT2+6)
1486:            do 680 JJ = 1, NNBIN
1487:               KBIN = IDTAL(IDT2+1)
1488:               if ( KBIN.eq.3 ) then
1489:                  ITC2 = 1
1490:                  go to 690
1491:               end if
1492: 680          continue
1493: C
1494: C ... time bin is set on ILUP(*)
1495: 690          continue

```

```

1496:      if ( ITC2.ne.0 ) then
1497:         ITCHK = 1
1498:         call BS0ISD( TIMEB, NTIME+1, DLOC1(1), ILUP(1),
1499:            & ITSF )
1500:         do 700 I = 1, ITSF
1501:            ILUP(I) = MAX(1,MIN(NTIME,ILUP(I)))
1502: 700          continue
1503:         end if
1504:      end if
1505: C
1506:      I1 = 1
1507:      I2 = ITSF
1508: C
1509:      if ( I2.gt.0 ) then
1510: C
1511: C ... put current/flux on working array XYZ1(*,1)
1512: C
1513: C ... current (just count weight!!)
1514:         if ( IDTAL(IDT+11).gt.0 ) then
1515:            KD72 = KD7 + NTSRF
1516:            do 710 I = I1, I1 + I2 - 1
1517: 710          XYZ1(I,1) = W(IBP0(I))
1518:               XYZ1(I,1) = W(IBP0(I)) *
1519:                  SIGN( 1.0d0, DBNK(IBP1(I),KD72) )
1520:            continue
1521:         else
1522:            KD72 = KD7 + NTSRF
1523:            do 720 I = I1, I1 + I2 - 1
1524:               DCC = ABS(DBNK(IBP1(I),KD72))
1525:               if ( DCC.gt.0.01d0 ) then
1526:                  XYZ1(I,1) = W(IBP0(I)) /DCC
1527:               else
1528:                  XYZ1(I,1) = W(IBP0(I)) /0.005D0
1529:               end if
1530:            continue
1531:         end if
1532: C
1533: C
1534:         call STALN3( IOW, ITS, NTSRF, JTIME, IDTAL(IDT+1),
1535:            & XYZ1(I1,1), I2, IBP1(I1), IBP0(I1), IGI, IRG,
1536:            & ILUP(I1), NGROUP, NGP1, NREG, NRESP, NBANK,
1537:            & NSTAL, NZONE, NTIME, RESP, NMKREG, MKREG,
1538:            & DTALY, SGTAL, DBNK, KDBNK, MDBNK, IBNK, KIBNK,
1539:            & MIBNK, XYZ1(1,2), R(1), R(NBANK+1) )
1540:         end if
1541: 730      continue
1542: C
1543: C ... go to next surface distance calculation cycle ...
1544: C
1545:         go to 640
1546: 740      continue
1547:      end if
1548: C
1549: C ... reduce distance to frame in lattice levels
1550: C
1551:         if ( JFISX.ne.0 ) then
1552:            MXLV = 0
1553:            KK = 0
1554:            *VOCL LOOP,NOVREC
1555:            do 750 I = 1, ISAFE
1556:               if ( IBNK(IBP(I),KIBNK(5)).ne.0 ) then
1557:                  MXLV = MAX(MXLV,LEVL(IBP(I)))
1558:                  KK = KK + 1
1559:               end if
1560: 750          continue

```

src/gmvp/flione.f

```

1561:      if ( MXLV.gt.0.and.KK.gt.0 ) then
1562:          KD4 = KDBNK(4)
1563:          do 770 LV = 1, MXLV
1564:      *VOCL LOOP,NOVREC
1565:          do 760 I = 1, ISAFE
1566:              if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.LEVL(IBP(I)).ge.LV
1567:                  &
1568:                      ) then
1569:                  DBNK(IBP(I),KD4+LV-1) = DBNK(IBP(I),KD4+LV-1)
1570:                  &
1571:                      - DI(I)
1572:              end if
1573:          continue
1574:      end if
1575: C
1576: C-----
1577: C .... Take track length tallies .....
1578: C-----
1579: C
1580: C ... change value of DI(I) for tally
1581: C
1582: C
1583:      if ( JNCOL.eq.1.and.MAT.gt.0 ) then
1584: C ... collisionless mode and non void region ...
1585: C
1586:      *VOCL LOOP,NOVREC
1587:          do 780 I = 1, ISAFE
1588:              DSIGT = STOTX(IGI(I),MAT)
1589:              DEE = DI(I)*DSIGT
1590:              if ( DEE.lt.1.0D-10 ) then
1591:                  DI(I) = DI(I)*W(I)
1592:              else
1593:                  DI(I) = ((1D0-EXP(-DEE))/DSIGT)*W(I)
1594:              end if
1595:          continue
1596:      else
1597: C
1598:      *VOCL LOOP,NOVREC
1599:          do 790 I = 1, ISAFE
1600:              DI(I) = DI(I)*W(I)
1601:          continue
1602:      end if
1603: C
1604: C-----
1605: C ... SPECIAL tallies by track length
1606: C-----
1607: C
1608:      if ( NTEVE.gt.0.and.JTEVE(2).gt.0 ) then
1609: C
1610: C ... gather current time & time bin here in X & IKL !!
1611: C
1612:      if ( JTIME.ne.0 ) then
1613:      *VOCL LOOP,NOVREC
1614:          do 800 I = 1, ISAFE
1615:              X(I) = TTT(IBP(I))
1616:              IKL(I) = ITT(IBP(I))
1617:              if ( JPTIM.ne.0 ) DLOC1(I) = TI(I)
1618:          continue
1619:      end if
1620:      do 840 J = 0, JTEVE(2) - 1
1621: C
1622: C ... pointer to direct tally (idt) & d-tally # (it) ...
1623:          IDT = KTEVE(LTEVE(2)+J) - 1
1624:          IT = IDTAL(IDT+9)
1625: C

```

```

1626: C .. neutron or photon ?
1627:      if ( IDTAL(IDT+5).eq.1.and.NNEUT.eq.0 ) go to 840
1628:      if ( IDTAL(IDT+5).eq.2.and.NPHOT.eq.0 ) go to 840
1629: C
1630: C ... skip if current region does not need this tally ...
1631: C
1632:      if ( JREGN.eq.0 ) then
1633:          if ( JDTRG(IRG(1),IT).eq.0 ) go to 840
1634:      else
1635:          do 810 I = 1, ISAFE
1636:              if ( JDTRG(IRG(I),IT).ne.0 ) go to 820
1637:          continue
1638:          go to 840
1639:      820 continue
1640:      end if
1641: C
1642:      I1 = 1
1643:      I2 = ISAFE
1644: C
1645:      if ( I2.gt.0 ) then
1646:          JPTIM2 = 0
1647: C
1648: C ... SGTL is not supported in GMVP currently.
1649: C
1650:          call STALN1( IOW, JTIME, JPTIM, MZONE, IDTAL(IDT+1),
1651:                      &
1652:                          DI(I1), I2, IGI(I1), IRG(I1), IBP(I1), IKL(I1),
1653:                          X(I1), TI(I1), NGROUP, NGP1, NREG, NRESP, NBANK,
1654:                          NSTAL, NZONE, NTIME, RESP, TIMEB, NMKREG, MKREG,
1655:                          DTALY, SGTL, TCUT, DBNK, KDBNK, MDBNK, IBNK,
1656:                          KIBNK, MIBNK, Y, Z, R, JPTIM2 )
1657: C
1658:          if ( JPTIM2.ne.0 ) then
1659:              do 830 I = I1, I2
1660:                  X(I) = TTT(IBP(I))
1661:                  IKL(I) = ITT(IBP(I))
1662:                  TI(I) = DLOC1(I)
1663:              continue
1664:          end if
1665:          840 continue
1666: C
1667:      end if
1668: C
1669: C-----
1670: C .... update time and time-bin # .....
1671: C-----
1672: C
1673:      if ( JTIME.ne.0 ) then
1674:          if ( JPTIM.eq.0 ) then
1675:      *VOCL LOOP,NOVREC
1676:          do 850 I = 1, ISAFE
1677:              TTT(IBP(I)) = TTT(IBP(I)) + TI(I)
1678:              X(I) = TTT(IBP(I))
1679:          continue
1680:      else
1681: C
1682: C ... case of PERIODIC TIME
1683:      *VOCL LOOP,NOVREC
1684:          do 860 I = 1, ISAFE
1685:              TTT(IBP(I)) = TTT(IBP(I)) + TI(I)
1686:              if ( TTT(IBP(I)).ge.TCUT ) then
1687:                  ITREP = TTT(IBP(I)) /TCUT
1688:                  TTT(IBP(I)) = TTT(IBP(I)) - TCUT*ITREP
1689:              end if
1690:              X(I) = TTT(IBP(I))

```

src/gmvp/flione.f

```

1691:      860      continue
1692:      end if
1693:
1694:      call BS0ISD( TIMEB, NTIME+1, X(1), IFC(1), ISAFE )
1695:      do 870 I = 1, ISAFE
1696:          ITT(IBP(I)) = MAX(1,MIN(NTIME,IFC(I)))
1697:      870      continue
1698:      end if
1699: C
1700: C
1701: C .... TAKE TALLIES (FLUX)
1702: C
1703: C-----
1704: C .... TAKE TALLIES (non-universe dependent tally ).....
1705: C-----
1706: C
1707: C .... TAKE TALLIES (FLUX)
1708: C
1709:      if ( JTLT.eq.0 ) then
1710:          MREG = KZREG(MZONE)
1711: *VOCL LOOP,SCALAR
1712:          do 880 I = 1, ISAFE
1713:              FLTR(IGI(I),MREG) = FLTR(IGI(I),MREG) + DI(I)
1714:      880      continue
1715:          else
1716: C
1717: C-----
1718: C .... TAKE TALLIES (universe dependent tally ).....
1719: C-----
1720: C
1721: C .... TAKE TALLIES (FLUX)
1722: C
1723: *VOCL LOOP,SCALAR
1724:          do 890 I = 1, ISAFE
1725:              FLTR(IGI(I),IBREG(IBP(I))) = FLTR(IGI(I),IBREG(IBP(I)))
1726:              & + DI(I)
1727:      890      continue
1728:          end if
1729: C
1730: C-----
1731: C .... TAKE TALLIES (EIGEN-VALUE).....
1732: C-----
1733: C
1734:      if ( JEIGN.gt.0.and.MAT.gt.0 ) then
1735:          do 900 I = 1, ISAFE
1736:              WCNTR(15) = WCNTR(15) + DI(I)*SNUFX(IGI(I),MAT)
1737:              WCNTR(20) = WCNTR(20) + DI(I)*STOTX(IGI(I),MAT)*
1738:              & (1.-SNAPX(IGI(I),MAT))
1739:      900      continue
1740:          end if
1741: C
1742: C-----
1743: C .... DELETE PARTICLES FROM FLIGHT STACK .....
1744: C-----
1745: C
1746:      910 II = 0
1747:
1748: *VOCL LOOP,NOVREC
1749:      do 920 N = 1, NFFL(NZ1)
1750:          if ( IZFFL(N).ne.MZONE ) then
1751:              II = II + 1
1752:              LSFFL(II) = LSFFL(N)
1753:              IZFFL(II) = IZFFL(N)
1754:          end if
1755:      920 continue

```

```

1756:
1757: NFFL(NZ1) = NFFL(NZ1) - NFFL(MZONE)
1758: NFFL(MZONE) = 0
1759:
1760: return
1761: end

```

src/gmvp/gamgen.f

```

1:      subroutine GAMGEN(
2:      1  NGROUP, NZONE, NREG, NTIME, NBANK, NGP1, NGP2,
3:      &  IREG, IMAT, IGRP,
4:      3  WGTG, WGTGI, WTIME, WLLIM, MXPGEN,
5:      4  NGPX, NGGX, NTGX, NMEDX, SGPX, SGPBX, IGPBX, S2DPX, S2PPX,
6:      5  LSCOL, NCOLS, LSDDED, NDEAD, LSFFL, IZFFL, NFFL,
7:      B  XXX, YYY, ZZZ, AAA, BBB, CCC,
8:      B  WWW, IZZ, IGG, TTT, ITT, XIM, KLSF,
9:      B  DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
10:     8  NGAM,
11:     9  NCNTR, WCNTR,
12:     A  IWK4, IWK5, IWK6, IWK7, IWK8,
13:     B  RW, RP, R, IWRK, IRAND,
14:     C  NEST, LEVL, LZZ, LPOS, LCRS, IBREG, IBSPC )
15: C=====
16: C  PURPOSE: SECONDARY PARTICLE (PHOTON) GENERATION
17: C  CALLED IN: COLISN
18: C  CALLS: RANU2
19: C=====
20: C
21: C    === INTER-STACK DATA FLOW ===
22: C
23: C    THE FOLLOWING DATA FLOW OCCURS.
24: C
25: C    DEAD PARTICLE STACK ---> (PARTICLE BANK) ---> FLIGHT STACK
26: C
27: C    === BANK DATA TO BE UPDATED ===
28: C
29: C    (NONE)
30: C
31: C    === BANK DATA ADDED NEWLY (FISSION) ===
32: C
33: C    (XXX,YYY,ZZZ), (AAA,BBB,CCC), IZZ : POSITION,DIRECTION & ZONE #
34: C    IGG, WWW : ENERGY GROUP & WEIGHT
35: C    LEVL,LZZ,LPOS,LCRS : LATTICE-PARAMETER
36: C
37: C=====
38: C    implicit real*8(A-H,O-Z)
39: C
40: C    include 'INC/_FLAGS'
41: C    integer JDEBG(*)
42: C
43: C
44: C    integer IREG(NCOLS), IMAT(NCOLS), IGRP(NCOLS)
45: C
46: C**** CROSS SECTION DATA
47: C
48: C    real WGTG(*), WGTGI(*)
49: C    real SGPBX(NGGX,3,NGPX,*), SGPX(NGGX,NGPX), S2DPX(2,NTGX,*),
50: C    real SGPBX(NGGX,3,NGPX,*), SGPX(NGPX,*), S2DPX(2,NTGX,*),
51: C    & S2PPX(3,NTGX,*)
52: C    integer IGPBX(NGGX,3,NGPX,*)
53: C
54: C**** Variance reduction
55: C
56: C    real WTIME(NTIME)
57: C    real WLLIM
58: C
59: C**** TALLY BIN DATA
60: C
61: C**** STACK
62: C
63: C    integer LSCOL(NBANK), LSDDED(NBANK)
64: C    integer LSFFL(NBANK), IZFFL(NBANK), NFFL(NZONE)
65: C

```

```

66: C**** PARTICLE BANK
67: C
68: C    real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
69: C    real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
70: C    integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
71: C    & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
72: C    real*8 TTT(NBANK)
73: C    integer ITT(NBANK)
74: C    real WWW(NBANK), XIM(NBANK)
75: C    integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
76: C
77: C ... optional bank parameters
78: C
79: C    real*8 DBNK(NBANK,*)
80: C    integer KDBNK(0:MDBNK)
81: C    integer IBNK(NBANK,*)
82: C    integer KIBNK(0:MIBNK)
83: C
84: C**** TALLY ARRAY
85: C
86: C    real*8 WCNTR(*), NCNTR(*)
87: C
88: C**** WORK AREA
89: C
90: C    integer IWK4(NBANK), IWK5(NBANK), IWK6(NBANK), IWK7(NBANK),
91: C    & IWK8(NBANK), IWRK(NBANK)
92: C    real RW(NBANK), RP(NBANK), R(4*NBANK)
93: C
94: C 0...NBANK..2NBANK..3NBANK..4NBANK..5NBANK..6NBANK..7NBANK..8NBANK
95: C IWK1 IWK2 IWK3 IWK4 IWK5 IWK6 IWK7 RP R,IWRK
96: C
97: C  IREG : REGION, IMAT : MATERIAL, IGRP : GROUP
98: C
99: C***** SECONDARY PARTICLE GERATION
100: C
101: C    call RANU2( IRAND, R, NCOLS, ICON )
102: C
103: C    NGAM = 0
104: C    IF( NGP1.LT.NGPX ) THEN
105: C    do 100 I = 1, NCOLS
106: C        IP = LSCOL(I)
107: C        if ( IGRP(I).le.NGP1 ) then
108: C            RP(I) = WWW(IP)*SGPX(IGRP(I),IMAT(I))
109: C            WW = RP(I)*WGTGI(IREG(I))
110: C            if ( JWTIM.ne.0 ) WW = WW/WTIME(ITT(IP))
111: C            if ( JRWVR.ne.0 ) WW = WW/DBNK(IP,KDBNK(1))
112: C            NP = WW
113: C            P = WW - NP
114: C            CCCCCCCCCC IWK4(I) = NP + 1 + P - R(I)
115: C            IWK4(I) = NP + P + R(I)
116: C            NGAM = NGAM + IWK4(I)
117: C        else
118: C            IWK4(I) = 0
119: C            RP(I) = 0.0
120: C        end if
121: C    100 continue
122: C
123: C    ELSE
124: C    DO 110 I=1, NCOLS
125: C        IP = LSCOL(I)
126: C        RP(I) = WWW(IP)*SGPX( IGRP(I),IMAT(I) )
127: C        WW = RP(I) * WGTGI( IREG(I) )
128: C        NP = WW
129: C        P = WW - NP
130: C    CCCCCCCCCC IWK4(I) = NP + 1 + P - R(I)

```


src/gmvp/gamgen.f

```

131: Cm          IWK4(I) = NP      + P + R(I)
132: Cm          NGAM      = NGAM  + IWK4(I)
133: Cm110      CONTINUE
134: Cm      ENDIF
135: C
136: C*****
137: C          IWK5 : COLLIDING NEUTRON GENERATING PHOTON
138: C          IWK7 : ENERGY GROUP
139: C          FISSION NEUTRON -----> PARTICLE BANK(XXX,YYY,.....)
140: C          -----> SEND TO FREE-FLIGHT STACK
141: C
142: C
143: C      ... check generation cutoff
144: C
145: C      if ( NGAM.gt.0.and.MXPGEN.gt.0 ) then
146: C          NGCUT = 0
147: C          do 110 I = 1, NCOLS
148: C              IP = LSCOL(I)
149: C              if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IWK4(I).gt.0 ) NGCUT =
150: C                  & NGCUT + 1
151: C          110 continue
152: C
153: C      if ( NGCUT.gt.0 ) then
154: C          NCUT = 0
155: C          WWC = 0.0D0
156: C          do 120 I = 1, NCOLS
157: C              IP = LSCOL(I)
158: C              if ( IBNK(IP,KIBNK(4)).ge.MXPGEN.and.IWK4(I).gt.0 ) then
159: C                  NCUT = NCUT + IWK4(I)
160: C                  WWC = WWC + RP(I)
161: C                  IWK4(I) = 0
162: C              end if
163: C          120 continue
164: C          NCNTR(27) = NCNTR(27) + NCUT
165: C          WCNTR(27) = WCNTR(27) + WWC
166: C          NGAM = NGAM - NCUT
167: C      end if
168: C      end if
169: C
170: C      if ( NGAM.le.0 ) then
171: C          return
172: C      end if
173: C
174: C      if ( NGAM.le.NDEAD ) then
175: C          MA = 0
176: C          do 130 I = 1, NCOLS
177: C              MA = MAX(MA,IWK4(I))
178: C          130 continue
179: C          N = 0
180: C          if ( JWTIM.eq.0 ) then
181: C              do 150 K = 1, MA
182: C                  do 140 I = 1, NCOLS
183: C                      if ( IWK4(I).ge.K ) then
184: C                          N = N + 1
185: C                          IWK5(N) = I
186: C                          RW(N) = WGTG(IREG(I))
187: C                      end if
188: C                  140 continue
189: C              150 continue
190: C          else
191: C              do 170 K = 1, MA
192: C                  do 160 I = 1, NCOLS
193: C                      if ( IWK4(I).ge.K ) then
194: C                          N = N + 1
195: C                          IWK5(N) = I
196: C                          RW(N) = WGTG(IREG(I))*WTIME(ITT(LSCOL(I)))
197: C                      end if
198: C                  160 continue
199: C              170 continue
200: C          end if
201: C          I1 = NCOLS
202: C      else
203: C          NGAM = 0
204: C          N = 0
205: C          I1 = 0
206: C          if ( JWTIM.eq.0 ) then
207: C              do 190 I = 1, NCOLS
208: C                  K = IWK4(I)
209: C                  NGAM = NGAM + K
210: C                  if ( NGAM.gt.NDEAD ) go to 220
211: C                  I1 = I
212: C                  do 180 II = 1, K
213: C                      N = N + 1
214: C                      IWK5(N) = I
215: C                      RW(N) = WGTG(IREG(I))
216: C                  180 continue
217: C              190 continue
218: C              go to 260
219: C          else
220: C              do 210 I = 1, NCOLS
221: C                  K = IWK4(I)
222: C                  NGAM = NGAM + K
223: C                  if ( NGAM.gt.NDEAD ) go to 220
224: C                  I1 = I
225: C                  do 200 II = 1, K
226: C                      N = N + 1
227: C                      IWK5(N) = I
228: C                      RW(N) = WGTG(IREG(I))*WTIME(ITT(LSCOL(I)))
229: C                  200 continue
230: C              210 continue
231: C              go to 260
232: C          end if
233: C
234: C
235: C          220 continue
236: C          NGAM = N
237: C          I2 = I1
238: C          I3 = 0
239: C          call RANU2( IRAND, R, NCOLS-I1, ICON )
240: C          *VOCL LOOP,NOVREC
241: C          do 230 I = I1 + 1, NCOLS
242: C              if ( RP(I).gt.0.0 ) then
243: C                  NCNTR(13) = NCNTR(13) + IWK4(I)
244: C                  WCNTR(13) = WCNTR(13) + RP(I)
245: C              end if
246: C              IP = LSCOL(I)
247: C              S1 = S2PPX(3,IGRP(I),IMAT(I))
248: C
249: C          if ( R(I-I1).gt.S1 ) then
250: C              I3 = I3 + 1
251: C              IWK6(I3) = IP
252: C              IWK8(I3) = IMAT(I)
253: C              IWK7(I3) = IGRP(I)
254: C              WWW(IP) = WWW(IP)*S2DPX(2,IGRP(I),IMAT(I))
255: C          end if
256: C          if ( R(I-I1).le.S1 ) then
257: C              I2 = I2 + 1
258: C              LSCOL(I2) = IP
259: C              IREG(I2) = IREG(I)
260: C              IMAT(I2) = IMAT(I)

```

src/gmvp/gamgen.f

```

261:          IGRP(I2) = IGRP(I)
262:          WWW(IP) = WWW(IP) /S1
263:          end if
264: 230      continue
265: C
266:          NCOLS = I2
267: C
268:          if ( I3.gt.0 ) then
269:              call RANU2( IRAND, R, 4*I3, ICON )
270:              I32 = I3*2
271:              I33 = I3*3
272:              NN = NFFL(NZONE+1)
273: C
274: C..... PHOTON GENERATION
275: C
276: *VOCL LOOP,NOVREC
277: do 240 I = 1, I3
278:     IP = IWK6(I)
279:
280: C/#IF ROUNDOFF(NEAREST)
281:     LL = MIN(INT(NGGX*R(I))+1,NGGX)
282: C/#ELSE
283:     *      LL = NGGX*R(I) + 1
284: C/#ENDIF
285:
286:     RR = R(I3+I) - SGPBX(LL,1,IWK7(I),IWK8(I))
287:
288:     LL2 = 2*LL + NGGX
289: C/#IF ROUNDOFF(NEAREST)
290:     LL3 = MIN(LL2,INT(LL2+RR))
291: C/#ELSE
292:     *      LL3 = LL2 + RR
293: C/#ENDIF
294: c97/11/07     LL3 = INT(2*LL+RR) + NGGX
295:
296:     IGG(IP) = IGPBX(LL3,1,IWK7(I),IWK8(I))
297: C
298:     if ( IGG(IP).le.NGROUP ) then
299:         W = 1. - 2.*R(I32+I)
300:         A = SQRT(1.-W*W)
301:         PHI = 6.283185307*R(I33+I)
302:         U = A*COS(PHI)
303:         V = A*SIN(PHI)
304:         S = 1./SQRT(U*U+V*V+W*W)
305:         AAA(IP) = U*S
306:         BBB(IP) = V*S
307:         CCC(IP) = W*S
308: C----- SEND TO FLIGHT STACK -----
309:         NN = NN + 1
310:         LSFFL(NN) = IP
311:         IZFFL(NN) = IZZ(IP)
312: C-----
313:     else
314:         NDEAD = NDEAD + 1
315:         LSDDED(NDEAD) = IP
316:         NCNTR(8) = NCNTR(8) + 1
317:         WCNTR(8) = WCNTR(8) + WWW(IP)
318:     end if
319: 240      continue
320: C
321:     N1 = NFFL(NZONE+1) + 1
322:     do 250 I = N1, NN
323:         NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
324: 250      continue
325:     NFFL(NZONE+1) = NN

```

```

326: C
327:     end if
328: C
329: 260      continue
330:     end if
331: C
332: C
333:     if ( NGAM.le.0 ) then
334:         return
335:     end if
336: C
337:     W = 0.0
338: *VOCL LOOP,NOVREC
339:     do 270 I = 1, NGAM
340:         W = W + RW(I)
341: 270      continue
342: Cm     NCNTR(2) = NCNTR(2) + NGAM
343: Cm     WCNTR(2) = WCNTR(2) + W
344:         NCNTR(3) = NCNTR(3) + NGAM
345:         WCNTR(3) = WCNTR(3) + W
346: C
347:         call RANU2( IRAND, R, 2*NGAM, ICON )
348: C
349: C     ENERGY ----> IWK7
350: C
351:         do 280 I = 1, NGAM
352:             I0 = IWK5(I)
353:
354: C/#IF ROUNDOFF(NEAREST)
355:             LL = MIN(INT(NGGX*R(I))+1,NGGX)
356: C/#ELSE
357:             *      LL = NGGX*R(I) + 1
358: C/#ENDIF
359:
360:             RR = R(NGAM+I) - SGPBX(LL,1,IGRP(I0),IMAT(I0))
361:
362: c97/11/07     LL3 = INT(2*LL+RR) + NGGX
363:             LL2 = 2*LL + NGGX
364: C/#IF ROUNDOFF(NEAREST)
365:             LL3 = MIN(LL2,INT(LL2+RR))
366: C/#ELSE
367:             *      LL3 = LL2 + RR
368: C/#ENDIF
369:
370:             IWK7(I) = IGPBX(LL3,1,IGRP(I0),IMAT(I0))
371: 280      continue
372: C
373: C     .... ENERGY CUT OFF ....
374: C
375:     if ( NTGX.gt.NGROUP ) then
376:         NN = 0
377: *VOCL LOOP,NOVREC
378:         do 290 I = 1, NGAM
379:             if ( IWK7(I).gt.NGROUP ) then
380:                 NCNTR(8) = NCNTR(8) + 1
381:                 WCNTR(8) = WCNTR(8) + RW(I)
382:             else
383:                 NN = NN + 1
384:                 IWK5(NN) = IWK5(I)
385:                 IWK7(NN) = IWK7(I)
386:                 RW(NN) = RW(I)
387:             end if
388: 290      continue
389:         NGAM = NN
390:         if ( NGAM.eq.0 ) return

```

src/gmvp/gamgen.f

```

391:      end if
392: C
393:      call RANU2( IRAND, R, 2*NGAM, ICON )
394: C
395:      NN      = NFFL(NZONE+1)
396:      NDD0    = NDEAD
397: *VOCL LOOP,NOVREC
398:      do 300 I = 1, NGAM
399:          I0   = IWK5(I)
400:          L0   = LSCOL(I0)
401:          L1   = LSDED(NDD0)
402:          NDD0 = NDD0 - 1
403: C
404:          W    = 1. - 2.*R(I)
405:          A    = SQRT(1.-W*W)
406:          PHI  = 6.283185307*R*(NGAM+I)
407:          U    = A*COS(PHI)
408:          V    = A*SIN(PHI)
409:          S    = 1./SQRT(U*U+V*V+W*W)
410:          AAA(L1) = U*S
411:          BBB(L1) = V*S
412:          CCC(L1) = W*S
413:          XXX(L1) = XXX(L0)
414:          YYY(L1) = YYY(L0)
415:          ZZZ(L1) = ZZZ(L0)
416:          WWW(L1) = RW(I)
417:          IGG(L1) = IWK7(I)
418:          IZZ(L1) = IZZ(L0)
419:          TTT(L1) = TTT(L0)
420:          KLSF(L1) = 0
421:          if ( JIMPT.ne.0 ) XIM(L1) = XIM(L0)
422:          if ( JLATT.ne.0 ) LEVL(L1) = LEVL(L0)
423:          if ( JTLT.ne.0 ) IBREG(L1) = IBREG(L0)
424:          IBREG(L1) = IBREG(L0)
425:          if ( JTIME.ne.0 ) ITT(L1) = ITT(L0)
426: C---- SEND TO FREE-FLIGHT STACK
427:          LSFFL(NN+I) = L1
428:          IZFFL(NN+I) = IZZ(L1)
429:      300 continue
430: C
431:          if ( JLATT.ne.0 ) then
432:              do 320 K = 1, NEST
433:                  NDD1 = NDEAD
434: *VOCL LOOP,NOVREC
435:                  do 310 I = 1, NGAM
436:                      L0   = LSCOL(IWK5(I))
437:                      L1   = LSDED(NDD1)
438:                      NDD1 = NDD1 - 1
439:                      LZZ(L1,K) = LZZ(L0,K)
440:                      LPOS(L1,K) = LPOS(L0,K)
441:                      if ( JHLAT.ne.0 ) LCRS(L1,K) = LCRS(L0,K)
442:                      if ( JTLT.ne.0 ) IBSPC(L1,K) = IBSPC(L0,K)
443:                  310 continue
444:                  320 continue
445:              end if
446: C
447:          ... copy or change optional bank parameters
448: C
449:          do 360 K = 1, KDBNK(0)
450: C
451:              NDD1 = NDEAD
452:              if ( K.eq.KDBNK(1) ) then
453: C
454:          ... birth weight is current weight !!!
455:

```

```

456: *VOCL LOOP,NOVREC
457:      do 330 I = 1, NGAM
458:          L1   = LSDED(NDD1)
459:          NDD1 = NDD1 - 1
460:          DBNK(L1,KDBNK(1)) = WWW(L1)
461:      330 continue
462: C
463:          ... time of birth is just now !!!
464: C
465:          else if ( K.eq.KDBNK(2) ) then
466: *VOCL LOOP,NOVREC
467:              do 340 I = 1, NGAM
468:                  L1   = LSDED(NDD1)
469:                  NDD1 = NDD1 - 1
470:                  DBNK(L1,KDBNK(2)) = TTT(L1)
471:              340 continue
472: C
473:          ... other optional bank parameters are only copy of those
474:          of parents.
475:          else
476: *VOCL LOOP,NOVREC
477:              do 350 I = 1, NGAM
478:                  L0   = LSCOL(IWK5(I))
479:                  L1   = LSDED(NDD1)
480:                  NDD1 = NDD1 - 1
481:                  DBNK(L1,K) = DBNK(L0,K)
482:              350 continue
483:          end if
484:      360 continue
485: C
486:          do 390 K = 1, KIBNK(0)
487:              NDD1 = NDEAD
488: C
489:          ... increment "particle generation" if necessary
490: C
491:          if ( K.eq.KIBNK(4) ) then
492: *VOCL LOOP,NOVREC
493:              do 370 I = 1, NGAM
494:                  L0   = LSCOL(IWK5(I))
495:                  L1   = LSDED(NDD1)
496:                  NDD1 = NDD1 - 1
497:                  IBNK(L1,KIBNK(4)) = IBNK(L0,KIBNK(4)) + 1
498:              370 continue
499:          else
500: *VOCL LOOP,NOVREC
501:              do 380 I = 1, NGAM
502:                  L0   = LSCOL(IWK5(I))
503:                  L1   = LSDED(NDD1)
504:                  NDD1 = NDD1 - 1
505:                  IBNK(L1,K) = IBNK(L0,K)
506:              380 continue
507:          end if
508:      390 continue
509: C
510: C
511:          NDEAD = NDD0
512: C
513: C---- SEND PHOTON TO FREE-FLIGHT STACK
514: C
515: *VOCL LOOP,SCALAR
516:      do 400 I = NN + 1, NN + NGAM
517:          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
518:      400 continue
519:          NFFL(NZONE+1) = NN + NGAM
520: C

```

src/gmvp/gamgen.f

```
521: C
522:      return
523:      end
```

SAE

src/gmvp/getmus.f

```

1:      subroutine GETMUS
2: C=====
3: C  PURPOSE: CALCULATES MU AND SIG WHICH ARE USED IN THE RECURRENCE
4: C          RELATION FOR ORTHOGONAL POLYNOMIALS Q.
5: C          (FROM MORSE CODE)
6: C  CALLED IN: ANGLES
7: C=====
8:      real    MOMENT, L, MU, NORM
9:      common /MOMENT/ NMOM, MOMENT(25), NF, F(25)
10:     common /MEANS/ NM, NV, MU(21), SIG(20), NORM(20)
11:     common /QAL/ Q(21), A(20,21), L(21)
12:     common /LOCSIG/ ISTART, ISGCOG, INABOG, IGABOG, IFPORG, IFNGP,
13: & IFSPOG, IDSGOG, IPRENG, IPRBGG, ISGANG, ISCAGG, ISPOG,
14: & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
15: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
16: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
17: & IADJ, NME, LOC, INGS, INSG, IL, IO,
18: & KKK, IXTAPE, IDEL, ITEMEL, ITEMG, IRSG, IRDSG,
19: & ISTR, IPRIN, IFMU, IMOM, IDTF, ISTAT, IPUN,
20: & INUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
21: & INNN, IGGG
22: c
23: c      ..... REWRITED TO FORTRAN77 STYLE 3/16/88
24: c
25:     NV = NMOM/2
26:     NM = NMOM - NV
27:
28:     do 110 I = 1, NV
29:       MU(I) = 0.0
30:       SIG(I) = 0.0
31:       NORM(I) = 0.0
32:       L(I) = 0.0
33:       Q(I) = 0.0
34:       do 100 K = 1, NM
35:         A(I,K) = 0.0
36:       100 continue
37:     110 continue
38:
39:     L(NM) = 0.
40:     Q(NM) = 0.
41:     MU(NM) = 0.0
42:     MU(1) = MOMENT(1)
43:     Q(1) = MOMENT(1)
44:     L(1) = MOMENT(1)
45:     A(1,1) = -MOMENT(1)
46:     A(1,2) = 1.0
47:     SIG(1) = MOMENT(2) - MOMENT(1)**2
48:     NORM(1) = SIG(1)
49: c
50:     if ( SIG(1).le.0.0 ) then
51:       I = 1
52:       go to 170
53:     end if
54: c
55:     if ( NV.gt.1 ) then
56:       L(2) = MOMENT(3) - MOMENT(1)*MOMENT(2)
57:       Q(2) = L(2)/NORM(1)
58:       MU(2) = Q(2) - Q(1)
59:       A(2,3) = 1.
60:       A(2,2) = -Q(2)
61:       A(2,1) = (MOMENT(1)*MOMENT(3)-MOMENT(2)**2)/SIG(1)
62:       NORM(2) = MOMENT(4) + A(2,2)*MOMENT(3) + A(2,1)*MOMENT(2)
63:
64:       SIG(2) = NORM(2)/NORM(1)
65:

```

```

66:     if ( SIG(2).le.0.0 ) then
67:       I = 2
68:       go to 170
69:     end if
70:
71:     if ( NV.gt.2 ) then
72:       do 150 I = 3, NV
73:         IM1 = I - 1
74:         IP1 = I + 1
75:
76:         do 120 K = 1, I
77:           L(I) = L(I) + A(IM1,K)*MOMENT(IM1+K)
78:         120 continue
79:
80:         Q(I) = L(I)/NORM(IM1)
81:         MU(I) = Q(I) - Q(IM1)
82:         A(I,IP1) = 1.0
83:         A(I,I) = -Q(I)
84:
85:         do 130 K = 2, IM1
86:           A(I,K) = A(IM1,K-1) - MU(I)*A(IM1,K) - SIG(IM1)*
87: & A(I-2,K)
88:         130 continue
89:
90:         A(I,1) = -MU(I)*A(IM1,1) - SIG(IM1)*A(I-2,1)
91:
92:         do 140 K = 1, IP1
93:           NORM(I) = NORM(I) + A(I,K)*MOMENT(IM1+K)
94:         140 continue
95:
96:         SIG(I) = NORM(I)/NORM(IM1)
97:         if ( SIG(I).le.0.0 ) go to 170
98:       150 continue
99:     end if
100:   end if
101:
102:   if ( NM.gt.NV.and.NV.gt.0 ) then
103:
104:     do 160 K = 1, NM
105:       L(NM) = L(NM) + A(NV,K)*MOMENT(NV+K)
106:     160 continue
107:
108:     Q(NM) = L(NM)/NORM(NV)
109:     MU(NM) = Q(NM) - Q(NV)
110:   end if
111: c
112:   if ( IFMU.ne.0 )
113: & write(I0,7000) (MU(I),SIG(I),NORM(I),L(I),Q(I),I=1,NM)
114: c
115:   return
116: c
117: 170 continue
118:   NM = I
119:   NV = I - 1
120: c
121:   if ( IPUN.gt.0 ) then
122:     write(I0,7040) I
123:     write(I0,7000) (MU(I),SIG(I),NORM(I),L(I),Q(I),I=1,NM)
124:     if ( NV.gt.0 ) then
125:       do 180 I = 1, NV
126:         IP1 = I + 1
127:         write(I0,7020) I, (A(I,K),K=1,IP1)
128:       180 continue
129:     end if
130:   end if

```

src/gmvp/getmus.f

```
131:         return
132: c
133: 7000 format(/1X,'INTERMEDIATE RESULTS OF MEANS CALCULATION'//1X,13X,
134:         &         'MEAN',19X,'VARIANCE',16X,'NORMALIZATION',
135:         &         12X,'L',23X,'Q'/(1P5E24.5))
136: 7020 format(/1X,'COEFFICIENTS OF ORTHOGONAL POLYNOMIALS'/1X,I4,10X,
137:         &         1P7E15.5/(8E15.5))
138: 7040 format(/1X,'-----GETMUS'/1X,'VARIANCE(',I2,
139:         &         ') IS NEGATIVE OR ZERO.')
140: c
141: end
```

SAFE

src/gmvp/gtdate.c

```

1: /*****
2:  * Output date and time of compilation on standard output
3:  * using ANSI C predefined macros if possible.
4:  *
5:  * 11 Jan 2001: written by Y.Nagaya
6:  *****/
7:
8: #include <stdio.h>
9: #include <string.h>
10:
11: #ifdef CRAY_C
12: #include <fortran.h>
13: #endif
14:
15: void get_cdate( char *cdate, int *cdatelen );
16: void get_ctime( char *ctime, int *ctimelen );
17:
18: void gtdate(cdate, cdatelen, ctime, ctimelen)
19: {
20:     char *cdate ;
21:     int *cdatelen ;
22:     char *ctime ;
23:     int *ctimelen ;
24: {
25:     get_cdate( cdate, cdatelen ) ;
26:     get_ctime( ctime, ctimelen ) ;
27: }
28: void gtdate_(cdate, cdatelen, ctime, ctimelen)
29: {
30:     char *cdate ;
31:     int *cdatelen ;
32:     char *ctime ;
33:     int *ctimelen ;
34: {
35:     get_cdate( cdate, cdatelen ) ;
36:     get_ctime( ctime, ctimelen ) ;
37: }
38: }
39: #ifdef CRAY_C
40: void GTDATE(cdate, cdatelen, ctime, ctimelen)
41: {
42:     _fcd cdate ;
43:     int *cdatelen ;
44:     _fcd ctime ;
45:     int *ctimelen ;
46: {
47:     get_cdate( _fcdtochp(cdate), cdatelen ) ;
48:     get_ctime( _fcdtochp(ctime), ctimelen ) ;
49: }
50: }
51: #else
52: void GTDATE(cdate, cdatelen, ctime, ctimelen)
53: {
54:     char *cdate ;
55:     int *cdatelen ;
56:     char *ctime ;
57:     int *ctimelen ;
58: {
59:     get_cdate( cdate, cdatelen ) ;
60:     get_ctime( ctime, ctimelen ) ;
61: }
62: }
63: #endif
64:
65: void _GTDATE(cdate, cdatelen, ctime, ctimelen)
66: {
67:     char *cdate ;
68:     int *cdatelen ;
69:     char *ctime ;
70:     int *ctimelen ;
71: {
72:     get_cdate( cdate, cdatelen ) ;
73:     get_ctime( ctime, ctimelen ) ;
74: }
75: }

```

```

66:     get_ctime( ctime, ctimelen ) ;
67: }
68:
69: void get_cdate( cdate, cdatelen )
70: {
71:     char *cdate ;
72:     int *cdatelen ;
73: {
74:     char *vp ;
75:     int i, nl ;
76:
77:     vp = "UNKNOWN" ;
78: #ifdef __DATE__
79:     vp = __DATE__ ;
80: #endif
81:
82:     nl = strlen(vp) ;
83:     if( nl <= *cdatelen )
84:     {
85:         memcpy( cdate, vp, nl ) ;
86:         for( i=nl; i< *cdatelen; ++i ) cdate[i] = ' ' ;
87:     }
88:     else
89:         memcpy( cdate, vp, *cdatelen ) ;
90: }
91:
92: void get_ctime( ctime, ctimelen )
93: {
94:     char *ctime ;
95:     int *ctimelen ;
96: {
97:     char *vp ;
98:     int i, nl ;
99:
100:     vp = "UNKNOWN" ;
101: #ifdef __TIME__
102:     vp = __TIME__ ;
103: #endif
104:
105:     nl = strlen(vp) ;
106:     if( nl <= *ctimelen )
107:     {
108:         memcpy( ctime, vp, nl ) ;
109:         for( i=nl; i< *ctimelen; ++i ) ctime[i] = ' ' ;
110:     }
111:     else
112:         memcpy( ctime, vp, *ctimelen ) ;
113: }

```

src/gmvp/gtndsk.f

```

1:      subroutine GTNSDK( SIGT, NSIG, ILIMIT, JNEUT, JPHOT, JSEP )
2: C=====
3: C  PURPOSE:  TO DETERMINE THE MAXIMUM NO. OF NON-ZERO DOWNSCATTERS
4: C            FOR EACH GROUP FOR ALL ELEMENTS.  (BORROWED FROM MORSE-DD)
5: C  CALLED IN: XSECIN
6: C  CALLS:    RESTOR
7: C=====
8:      include '../shared/INC/_IUNIT'
9: C
10:     common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
11: & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
12: & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
13: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
14: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
15: & IADJ, NME, LOC, INGS, INSG, I1, I0,
16: & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRDSG,
17: & ISTR, IPRIN, IPMU, IMOM, IDTF, ISTAT, IPUN,
18: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
19: & INNN, IGGG,
20: C
21:     integer NSIG(1)
22:     real    SIGT(1)
23: C
24: C ... statement function to indicate pointer calculation ....
25: C
26:     IPOINT(I,N) = I + N
27: C
28: C-----
29: C  THIS SUBROUTINE IS A COMBINATION OF READSG AND STORE
30: C  ONLY THE SCATTERING MATRIX IS STORED
31: C  TEMP STORE AT INP1
32: C  RESTORE AT IST
33: C  MAX STORAGE ISP
34: C
35:     if ( NME.gt.1 ) return
36: C
37:     NDSTMP = NDS
38:     if ( IADJ.le.0 ) then
39:         N1 = 1
40:         N2 = NGP
41:     else
42:         if ( NGG.gt.0 ) then
43:             N1 = NGP + 1
44:             N2 = NTG
45:             NDSTMP = NDSG
46:         else
47:             N1 = 1
48:             N2 = NGP
49:         end if
50:     end if
51: C
52:     IST = ISTART
53:     INP1 = IPOINT(IST,NTG*NTS)
54:     ISP = IPOINT(INP1,INGP*(INDS+IHT+NUS))
55:     if ( ISP.gt.ILIMIT ) then
56:         write(IMG,7000) LSIZ(ISP)
57: 7000    format('1','XXX(GTNSDK) NOT ENOUGH MEMORY,',
58: & ' NECESSARY AT LEAST ',I7,' WORDS.')
59:         call PRSTOP( 1, 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.'
60: & )
61:         stop 999
62:     end if
63: C
64:     do 100 I = N1, N2
65:         NSIG(INNN+I) = 0

```

```

66:     100 continue
67: C
68:     do 130 K = 1, NELEM
69:         KKK = (K-1)*NCOEF + 1
70:         call RESTOR( INP1, ISP, SIGT, NSIG, JNEUT, JPHOT, JSEP )
71:         do 120 I = N1, N2
72:             J1 = IST + (I-1)*NTS
73:             NDSK = N2 - I + 1 + NUS
74:
75:             NDSK = MIN(NUS+NDSTMP,MAX(NDSK,NUS+1))
76:
77:             M = 0
78:             do 110 J = 1, NDSK
79:                 if ( SIGT(J1+J).gt.0 ) M = J
80:             continue
81:             if ( M.ge.NSIG(INNN+I) ) then
82:                 NSIG(INNN+I) = M
83:             end if
84:         120 continue
85:     130 continue
86:     IXTAP = IABS(IXTAPE)
87: C
88:     call RWIND( IXTAP )
89: C
90:     140 return
91:     end

```


src/gmvp/gtvval.f

```

1:      subroutine GTVVAL( VNAME, VALUE, IERR )
2: C=<GMVP>=====
3: C
4: C  PURPOSE: RETURN THE VALUE OF AN INTERNAL VARIABLE NAMED VNAME:
5: C  CALLED IN:  DENTAK, PARA & OTHERS
6: C  ARGUMENTS:
7: C      VNAME : VARIABLE NAME (NOT VARIABLE ITSELF !)
8: C      VALUE : VALUE (EXPANDED TO DOUBLE PRECISION IF NECESSARY)
9: C      IERR  : ERROR CODE
10: C          0 : SUCCESSFUL
11: C          1 : NON-EXISTENT VARIABLE NAME OR VALUE FETCHING
12: C          NOT ALLOWED.
13: C
14: C          2 : PROBABLY VALUE UNDEFINED YET
15: C
16: C CAUTION: THIS ROUTINE IS CALLED FROM UTILITY ROUTINE DENTAK & PARAM
17: C          BUT CODE-DEPENDENT (MVP & GMVP USE DIFFRENT SOURCE FOR
18: C          THIS ROUTINE.
19: C
20: C=====
21:      character*(*) VNAME
22:      real*8 VALUE
23:      integer IERR
24: C
25:      include '../shared/INC/_SIZES'
26:      include 'INC/_SIZES2'
27:      include '../shared/INC/_PGEOM'
28:      include 'INC/_PXSEC'
29: C
30:      real*8 V0
31: C
32: C      .... VJ : A JUNK VALUE TO CONFIRM VALUE ASSIGNMENT ....
33: C
34:      real*8 VJ
35:      parameter( VJ = -9.9876543D37 )
36: C
37:      IERR = 0
38:      V0 = VJ
39: C
40: C  AVAILABLE VRIABLES :   ( 16 MAR 1992 )
41: C
42: C
43: C---- COMMON /SIZES/ ----
44: C
45:      if ( VNAME(1:2).ge.'NA'.and.VNAME(1:2).le.'NF' ) then
46:      if ( VNAME.eq.'NBATCH' ) V0 = NBATCH
47:      if ( VNAME.eq.'NCELL' ) V0 = NCELL
48:      if ( VNAME.eq.'NEST' ) V0 = NEST
49:      if ( VNAME.eq.'NFBANK' ) V0 = NFBANK
50:      if ( VNAME.eq.'NFBNK0' ) V0 = NFBNK0
51:      else if ( VNAME(1:2).ge.'NG'.and.VNAME(1:2).le.'NI' ) then
52:      if ( VNAME.eq.'NGP1' ) V0 = NGP1
53:      if ( VNAME.eq.'NGP2' ) V0 = NGP2
54:      if ( VNAME.eq.'NGROUP' ) V0 = NGROUP
55:      if ( VNAME.eq.'NHIST' ) V0 = NHIST
56:      if ( VNAME.eq.'NHSUB' ) V0 = NHSUB
57:      if ( VNAME.eq.'NINPZ' ) V0 = NINPZ
58:      else if ( VNAME(1:2).ge.'NJ'.and.VNAME(1:2).le.'NN' ) then
59:      if ( VNAME.eq.'NMAT' ) V0 = NMAT
60:      if ( VNAME.eq.'NMEMS' ) V0 = NMEMS
61:      if ( VNAME.eq.'NLATT' ) V0 = NLATT
62:      if ( VNAME.eq.'NLLEV' ) V0 = NLLEV
63:      if ( VNAME.eq.'NLBZ' ) V0 = NLBZ
64:      else if ( VNAME(1:2).ge.'NO'.and.VNAME(1:2).le.'NZ' ) then
65:      if ( VNAME.eq.'NPART' ) V0 = NPART

```

```

66:      if ( VNAME.eq.'NZONE' ) V0 = NZONE
67:      if ( VNAME.eq.'NREG' ) V0 = NREG
68:      if ( VNAME.eq.'NSOUR' ) V0 = NSOUR
69:      if ( VNAME.eq.'NRESP' ) V0 = NRESP
70:      if ( VNAME.eq.'NTHIST' ) V0 = NTHIST
71:      if ( VNAME.eq.'NTIME' ) V0 = NTIME
72:      if ( VNAME.eq.'NTREG' ) V0 = NTREG
73:      if ( VNAME.eq.'NPICT' ) V0 = NPICT
74: ccccc IF( VNAME .EQ. 'NPKIND' ) V0 = NPKIND
75:      else
76:      if ( VNAME.eq.'TCPU' ) V0 = TCPU
77: ccccc IF( VNAME .EQ. 'ECUT' ) V0 = BANKP
78:      end if
79:      if ( V0.ne.VJ ) go to 100
80: C
81: C      .... COMMON /PGEOM/ .....
82: C
83:      if ( VNAME.eq.'DEPS' ) V0 = DEPS
84:      if ( VNAME.eq.'DINF' ) V0 = DINF
85:      if ( V0.ne.VJ ) go to 100
86: C
87: C      .... COMMON /PXSEC/ .....
88: C
89:      if ( VNAME.eq.'NGPX' ) V0 = NGPX
90:      if ( VNAME.eq.'NGGX' ) V0 = NGGX
91:      if ( VNAME.eq.'NTGX' ) V0 = NTGX
92:      if ( V0.ne.VJ ) go to 100
93: C
94: C      .... VERIFICATION ...
95: C
96:      100 continue
97:      if ( V0.ne.VJ ) then
98:      VALUE = V0
99:      IERR = 0
100:      else
101:      IERR = 1
102:      end if
103:      return
104:      end

```

src/gmvp/gtxsec.f

```

1:      subroutine GTXSEC( A )
2:
3:      C/#!/IF PARA(PVM MPI)
4:
5:      C=====
6:      C  PURPOSE   : get CROSS-SECTION from parenttask in PVM/MPI mode
7:      C  CALLED IN : INTRO
8:      C=====
9:      C
10:     real A(*)
11:     C
12:     include '../shared/INC/_LISTOFF'
13:     cccc include '../shared/INC/_ARRAY'
14:     include 'INC/_PXSEC'
15:     include 'INC/_FLAGS'
16:     include '../shared/INC/_IOUNIT'
17:     include '../shared/INC/_LISTON'
18:     C/#!/IF PARA(PVM)
19:     include '../shared/INC/_PVMPARA'
20:     C/#!/ELSEIF PARA(MPI)
21:     * include 'mpif.h'
22:     C/#!/ENDIF
23:     include '../shared/INC/_TASKDT'
24:     C
25:     C
26:     parameter( MXSEND = 50 )
27:     integer ISEND(MXSEND)
28:     C
29:     C -----
30:     C
31:     call GTLAST( LAST )
32:     ccc if ( ITASK0.ge.0 ) then
33:     if ( IDTASK.ne.1 ) then
34:     call TOKEI( T0, 0 )
35:     C
36:     C .... get sizes & pointers
37:     C
38:     NGET = 37
39:     IT0 = 1
40:     call MVPCOM_BCAST_I4( IT0, 1, ISEND, NGET, INFO )
41:     C
42:     LSTRT = ISEND(36)
43:     NN = LAST - LSTRT
44:     LSTRT = LSTRT + NN
45:     C
46:     JDDX = ISEND(1)
47:     JSTATX = ISEND(2)
48:     NGPX = ISEND(3)
49:     NGGX = ISEND(4)
50:     NTGX = ISEND(5)
51:     NDSPX = ISEND(6)
52:     NDSGX = ISEND(7)
53:     NDSTX = ISEND(8)
54:     NUSX = ISEND(9)
55:     NSCTX = ISEND(10)
56:     NCOEFX = ISEND(11)
57:     NMEDX = ISEND(12)
58:     LRSGX = ISEND(13) + NN
59:     LANGX = ISEND(14) + NN
60:     LNGSX = ISEND(15) + NN
61:     LNSGX = ISEND(16) + NN
62:     LDELX = ISEND(17) + NN
63:     LNNNX = ISEND(18) + NN
64:     LNNGX = ISEND(19) + NN
65:     LSTOTX = ISEND(20) + NN

```

```

66:     LSSCX = ISEND(21) + NN
67:     LSNUFX = ISEND(22) + NN
68:     LSNAPX = ISEND(23) + NN
69:     LSGPX = ISEND(24) + NN
70:     LSNFPX = ISEND(25) + NN
71:     LSNUSX = ISEND(26) + NN
72:     LS2DPX = ISEND(27) + NN
73:     LS2PPX = ISEND(28) + NN
74:     LSGPBX = ISEND(29) + NN
75:     LSSCBX = ISEND(30) + NN
76:     LSDDBX = ISEND(31) + NN
77:     LSANGX = ISEND(32) + NN
78:     LSGPPX = ISEND(33) + NN
79:     LSSCPX = ISEND(34) + NN
80:     LSPORT = ISEND(35) + NN
81:     C
82:     NXDATA = ISEND(37)
83:     C
84:     call STLAST( 'XSEC', LAST+NXDATA, LAST )
85:     C
86:     if ( LSIZ(LAST).gt.LIMGT() ) then
87:     write(IMG,*)
88:     & 'XXX(GTXSEC) TASK ',IDTASK,': Not enough memory to',
89:     & ' get cross section data from parent task in PVM/MPI mode'
90:     call PRSTOP( 1, 'MEMORY OVER.' )
91:     stop 999
92:     end if
93:     C
94:     C
95:     C
96:     C .... get cross section
97:     C
98:     MSGNUM = 123
99:     NBLOCK = 2**15
100:    C
101:    do 100 K = 1, NXDATA, NBLOCK
102:    L1 = LSTRT + K - 1
103:    K2 = MIN(NXDATA,K+NBLOCK-1)
104:    NN = K2 - K + 1
105:    MSGNUM = MSGNUM + 1
106:    IT0 = 1
107:    call MVPCOM_BCAST_I4( IT0, MSGNUM, NN, 1, INFO )
108:    call MVPCOM_BCAST_I4( IT0, MSGNUM, A(L1), NN, INFO )
109:
110:    if ( INFO.lt.0 ) then
111:    write(IMG,*)
112:    & 'XXX(GTXSEC) TASK ',IDTASK,': Failed to receive ',
113:    & 'Cross section data.'
114:    call PRSTOP( 1,
115:    & 'FAILED TO GET CROSS SECTION FROM PARENT TASK IN PVM/MPI MODE.'
116:    & )
117:    stop 888
118:    end if
119:    100 continue
120:    call TOKEI( T1, 0 )
121:    C
122:    write(IPR,'(/1X,A,E12.5/1X,A,E12.5/)')
123:    & ' == Elapsed time to get cross section data : ', T1
124:    & - T0, ' Data transfer rate (word/sec) : ',
125:    & (NGET+NXDATA) /(T1-T0)
126:    C
127:    end if
128:
129:    C/#!/ENDIF
130:

```

src/gmvp/gtxsec.f

131: return
132: end



src/gmvp/igtflg.f

```
1:      function IGTFLG(NAME)
2: C==<MVP>=====
3: C  PURPOSE: GET VALUE OF OPTION PARAMETERS(FLAG) BY NAME
4: C  CALLED IN:  ANYWHERE
5: C  CALLS      : (NONE)
6: C=====
7:      include 'INC/_FLAGS'
8:      include '../shared/INC/_IOUNIT'
9: C
10:     integer      IGTFLG
11:     character*(*) NAME
12: C
13:     if ( NAME.eq.'JDEBG' ) then
14:         IGTFLG = JDEBG(1)
15:     else if ( NAME.eq.'JMCHK' ) then
16:         IGTFLG = JMCHK
17:     else if ( NAME.eq.'JTLLT' ) then
18:         IGTFLG = JTLLT
19:     else if ( NAME.eq.'JLATT' ) then
20:         IGTFLG = JLATT
21:     else
22:         write(IMG,'(/lx,a,a)')
23:         & 'XXX(IGTFLG) CANNOT GET FLAG VALUE !! (', NAME, ' )'
24:         stop 666
25:     end if
26:     return
27: end
```

src/gmvp/intro2.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine INTRO2( A,      H,      CHA,  LIMIT, LIMITL, LIMITC,
3:     &                TITLE, MAXSF, JNP,  TCUTDM )
4: C=<GMVP>=====
5: C PURPOSE: Complete preparation for random walk after data input.
6: C CALLED IN: INTRO
7: C CALLS : REFZON KEPV HEADER CRVOL KEEP2 ADJREV FISREV CRVOL1 RESTI0
8: C        RXPDET CNTERR LABEL R4PRNT WRKARY CRVOLT XSECCN RESTRT PKKAI
9: C        KEEP SRCINP NGVEL PGVEL CHKNUM
10: C-----
11: C
12: C arguments ( i = input, o = output, c =constant, w = work )
13: C
14: C c i pr : printout unit
15: C
16: C i o A(*) : dynamic memory array (task shared)
17: C i o H(*) : dynamic memory array (task local)
18: C i o CHA(*) : character dynamic memory array (task shared)
19: C i LIMIT : effective size of A(*)
20: C i LIMITL : effective size of H(*)
21: C i LIMITC : effective size of CHA(*)
22: C
23: C i title : problem title
24: C i maxsf : maximum number of surface data to be used in geometry
25: C i jnp : 1 = neutron-photon coupled problem, 0 = no
26: C o jnrun : 1 = input is non-executable because of fatal errors
27: C        0 = no fatal error
28: C-----
29: C
30:   real A(*)
31:   real H(*)
32:   character*4 CHA(*)
33: C
34:   character TITLE(2)*72
35: C
36:   include 'INC/_KPIDS'
37:   include 'INC/_KPSYMS'
38:   include 'INC/_NGPS'
39: C
40: CCCCC include './shared/INC/_ARRAY'
41:   include './shared/INC/_TASKDT'
42:   include './shared/INC/_LISTOFF'
43:   include 'INC/_FLAGS'
44:   include './shared/INC/_SIZES'
45:   include 'INC/_SIZES2'
46:   include 'INC/_XBANK'
47:   include 'INC/_FBANK2'
48:   include 'INC/_STACK'
49:   include 'INC/_XWORK'
50:   include './shared/INC/_PGEOM'
51:   include 'INC/_PXSEC'
52:   include './shared/INC/_PVRED'
53:   include 'INC/_PSOUR'
54:   include './shared/INC/_PTALY0'
55:   include 'INC/_PTALY'
56:   include './shared/INC/_PTLSP'
57:   include './shared/INC/_STALY'
58:   include './shared/INC/_LISTON'
59:   include './shared/INC/_WORDL'
60:   include './shared/INC/_IOUNIT'
61: C
62: C-----
63: C ... KEEP MEMORY AREAS FOR NON-INPUT PARAMETERS & GIVE VALUES TO THEM
64: C-----
65: C

```

```

66: CCC   call CHKNGP( 'MVP', JNEUT, NGP1, JPHOT, NGP2, NGROUP )
67:   call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2, NGROUP )
68:   call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
69: C-----
70: C      .... /PGEOM/ .....
71: C-----
72: C
73: C-----
74: C      .... TRANSLATE MAT ID # TO MAT #
75: C-----
76: C
77: C
78: C      ... set all IDMAT to material sequenatial # if not defined.
79: C
80:   if ( LIDMAT.eq.0 ) then
81:     call KEPV( A(1), 'IDMAT', LIDMAT, NMEDX, 'I4', LAST, 0, JDEBG )
82: C
83:   if ( LSTCK(0).lt.0 ) then
84:     write(IMG,7140) 'material ID to material # conversion.'
85:     call PRSTOP( 1, 'MEMORY OVER.' )
86:     stop 999
87:   end if
88: C
89: C      ... copy media # as IDMAT in this case ...
90:   call ATRANS( A(LRSGX), A(LIDMAT), NMEDX )
91:   end if
92: C
93:   call MID2NM( 'GMVP', A(LKZMAT), A(LKMAT), A(LKINPZ), NINPZ, NZONE,
94:     &          JIMAG, A(LIDMAT), DUMMY1, NMAT )
95: C
96: C-----
97: C      .... /SIZES/ .....
98: C-----
99: C
100:   if ( NSCT.eq.0 ) NSCT = NSCTX
101: C
102: C-----
103: C      .... IN ADJOINT PROBLEM, NGP1 AND NGP2 ARE CHANGED HERE
104: C-----
105: C
106:   if ( JIMAG.eq.0 ) then
107:     call CHKNUM( JNEUT, JPHOT, JFISS, JADJM, NGROUP, NGP1, NGP2,
108:       &          NTGX, NGPX, NGGX, NMAT, NMEDX, A(LKMAT), NINPZ )
109:   end if
110: C
111: C-----
112: C      .... CHECK WHETHER NMEMO WAS INPUT OR NOT .....
113: C-----
114: C
115:   if ( NMEMO.le.0 ) then
116:     NMEMO = 5
117:     write(IMG,7000) NMEMO
118:     call CNTERR( 'WARNING' )
119: C
120: 7000   format('!!! Data "NMEMO" was set to a default value (',I4,
121:     &      ' ).')
122:   &      "NMEMO" is the size of memory of next-entering-zones'
123:   &      ' for each zone, and it is desirable that "NMEMO" be greater',
124:   &      ' than'/' the maximum number of possible next-zones.'
125:   &      ' This parameter might be important to reduce time consumed',
126:   &      ' for particle tracking in some case.'//)
127: C
128:   end if
129: C
130:   call LKEPV( H(1), 'KMEMO', LKMEMO, NMEMO*NZONE, 'I4', LAST, 0,

```

src/gmvp/intro2.f

```

131:      &      JDEBG )
132:      if ( NMEMP.gt.0 ) then
133:        call LKEPV( H(1), 'MEMC', LMEMC, NMEMP*NZONE, 'I4', LAST, 0,
134:      &      JDEBG )
135:        call LKEPV( H(1), 'MEMZ', LMEMZ, NZONE, 'I4', LAST, 0, JDEBG )
136:      end if
137: C
138:      if ( LVOL.ne.0 ) write(IPR,7020) (A(LVOL+I),I=0,NINPZ-1)
139: 7020 format(/1X,130('='))' VOLUMES OF EACH INPUT-ZONE'/1X,130('=')//
140:      &      (1X,2X,10E12.5))
141: C
142: C
143: C-----
144: C .... VARIANCE REDUCTION PARAMETER .....
145: C-----
146: C .... /PVRED/ .....
147: C
148: C
149: C =====
150: call HEADER( IPR, 'VARIANCE REDUCTION PARAMETERS' )
151: C =====
152: C
153:      if ( JIMPT.ne.0.and.LXIMP.eq.0 ) then
154:        call KEPV( A(1), 'XIMP', LXIMP, NGROUP*NREG, 'R4', LAST, 1.0,
155:      &      JDEBG )
156:      end if
157:      if ( JRRLT.ne.0 .or. JWWND.ne.0 ) then
158:        if ( LWKIL.eq.0 ) then
159:          call KEPV( A(1), 'WKIL', LWKIL, NGROUP*NREG, 'R4', LAST,
160:      &      1.E-5, JDEBG )
161:        end if
162:        if ( LWSRV.eq.0 ) then
163:          call KEPV( A(1), 'WSRV', LWSRV, NGROUP*NREG, 'R4', LAST,
164:      &      1.E-4, JDEBG )
165:        end if
166:      end if
167: C
168:      if ( LSTCK(0).lt.0 ) then
169:        write(IPR,7140) 'variance reduction parameter printout.'
170:        call PRSTOP( 1, 'MEMORY OVER.' )
171:        stop 999
172:      end if
173: C
174:      if ( JIMPT.ne.0 ) then
175: C
176: C *****
177: call LABEL( IPR, 'IMPORTANCE (GROUP & REGION-WISE)' )
178: C *****
179: C
180:      call R4PRNT( 'XIMP', A(LXIMP), NGROUP, NREG, 'GROUP', 'REGION',
181:      &      IPR )
182:      if ( JADJM.ne.0 ) call ADJREV( A(LXIMP), NGROUP, NREG )
183:    end if
184: C
185:      if ( JRRLT.ne.0 .or. JWWND.ne.0 ) then
186: C
187: C *****
188: call LABEL( IPR, 'WEIGHT THRESHOULD FOR RUSSION-ROULETTE' )
189: C *****
190: C
191:      call R4PRNT( 'WKIL', A(LWKIL), NGROUP, NREG, 'GROUP', 'REGION',
192:      &      IPR )
193: C
194: C *****
195: call LABEL( IPR, 'SURVIVAL WEIGHT ON RUSSION-ROULETTE' )

```

```

196: C *****
197: C
198:      call R4PRNT( 'WSRV', A(LWSRV), NGROUP, NREG, 'GROUP', 'REGION',
199:      &      IPR )
200: C
201: C ... check consistency of WARV and WKIL
202: C
203:      call CKWGHT( A(LWSRV), A(LWKIL), NGROUP, NREG )
204: C
205:      if ( JADJM.ne.0 ) call ADJREV( A(LWKIL), NGROUP, NREG )
206:      if ( JADJM.ne.0 ) call ADJREV( A(LWSRV), NGROUP, NREG )
207:    end if
208: C
209:      if ( JTIME.ne.0.and.LWTIME.ne.0 ) then
210:        JWTIM = 1
211: C *****
212: call LABEL( IPR, 'TIME BIN WEIGHTING FACTOR' )
213: C *****
214: C
215:      call R4PRNT( 'WTIME', A(LWTIME), NTIME, 1, 'TIME', ' ', IPR )
216: C
217: C ... WTIME must be non negative ...
218:      call CKVAL4( 'WTIME', A(LWTIME), NTIME, IFL, 1 )
219:      if ( IFL.ne.0 ) then
220:        write(IMG,*) 'XXX Time bin weight factor WTIME includes ',
221:      &      IFL, ' non positive value.'
222:        call CNTERR( 'FATAL' )
223:      end if
224:    end if
225: C
226:      if ( JPSTR.ne.0 ) then
227: C *****
228: call LABEL( IPR, 'PATH STRETCHING PARAMETERS' )
229: C *****
230: CC      if ( JLATT.ne.0 ) then
231: CC        write(IPR,*)
232: CC      &      '!!! Both PATH-STRETCHING option and LATTICE option ',
233: CC      &      ' are selected.',
234: CC      &      ' Path stretching in lattice region may not be effective',
235: CC      &      ' in current version.'
236: CC      call CNTERR( 'WARNING' )
237: CC    end if
238: C
239: C ...path stretching with lattice geometry requires JFISX > 0
240: C to get absolute coordinates of particles.
241: C
242:      if ( JLATT.ne.0 ) JFISX = 1
243: C
244:      if ( LPSXYZ.eq.0 ) then
245:        write(IMG,*) 'XXX PATH-STRETCHING option is selected but',
246:      &      ' no target points PSXYZ(x y z) are given.'
247:        call CNTERR( 'FATAL' )
248:      else
249:        write(IPR,7040)
250: 7040 format(/1X,3X,'=== target point coordinates ===')
251:        call PRTXYZ( IPR, 'PSXYZ', A(LPSXYZ) )
252:      end if
253:      if ( LPSALP.eq.0 ) then
254:        write(IMG,*)
255:      &      '!!! Path stretching factor (PSALP) is not input!!',
256:      &      ' Set to 0.0'
257:        call CNTERR( 'WARNING' )
258:        call KEPV( A(1), 'PSALP', LPSALP, NREG, 'R4', LAST, 0.0,
259:      &      JDEBG )
260:      end if

```

src/gmvp/intro2.f

```

261:      call R4PRNT( 'PSALP', A(LPSALP), NREG, 1, 'REGION', ' ', IPR )
262:      call CKALP( A(LPSALP), NREG )
263:      end if
264: C
265: C-----
266: C      .... /PXSEC/ ..... FISSION WEIGHT & FISSION SPECTRUM
267: C-----
268: C
269: C
270:      if ( JFISS.ne.0 ) then
271: C
272: C      =====
273: C      call HEADER( IPR, 'DATA ABOUT FISSION REACTION' )
274: C      =====
275: C
276: C      *****
277: C      call LABEL( IPR, 'FISSION SPECTRUM' )
278: C      *****
279: C
280:      if ( LFKAI.eq.0 ) then
281:      write(IMG,*)
282:      &      'XXX Fission spectrum is not specified !! STOP'
283:      call PRSTOP( 1, 'NO FISSION SPECTRUM DATA.' )
284:      stop 888
285:      end if
286:      call R4PRNT( 'FKAI', A(LFKAI), NTGX, NMAT, 'GROUP', 'MATERIAL'
287:      &      IPR )
288: C
289:      if ( JADJM.ne.0 ) then
290:      call ADJREV( A(LFKAI), NTGX, NMAT )
291:      call FISREV( A(LFKAI), NTGX, NMAT, A(LSTOTX), A(LSNUPX),
292:      &      A(LSNAPX), A(LSGPX), A(LSNFPX), A(LS2DPX),
293:      &      A(LS2PPX), NMEDX, NGPX, JEIGN, JFISS, JNEUT, JPHOT )
294:      end if
295: C
296:      if ( LWGTF.eq.0 ) then
297:      write(IMG,*) '!!! Fission weights (WGTG) are not input !!',
298:      &      ' Set to 1.0'
299:      call CNTERR( 'WARNING' )
300:      call KEPV( A(1), 'WGTG', LWGTF, NREG, 'R4', LAST, 1.0, JDEBG
301:      &      )
302:      end if
303: C
304: C      *****
305: C      call LABEL( IPR, 'FISSION WEIGHTS' )
306: C      *****
307: C
308: C
309:      call R4PRNT( 'WGTG', A(LWGTF), NREG, 1, 'REGION', ' ', IPR )
310: C
311:      call CKVAL4( 'WGTG', A(LWGTF), NREG, IFL, 1 )
312:      if ( IFL.ne.0 ) then
313:      write(IMG,*) 'XXX Fission neutron weight WGTG includes ',
314:      &      IFL, ' non positive value.'
315:      call CNTERR( 'FATAL' )
316:      end if
317: C
318:      call KEEP( 'WGTFI', LWGTFI, NREG, 'R4', LAST, JDEBG )
319: C
320:      call ARYINV( A(LWGTF), A(LWGTFI), NREG )
321: C      do 100 I = 1, NREG
322: C      if ( A(LWGTF+I-1).ne.0 ) A(LWGTFI+I-1) = 1.0/A(LWGTF+I-1)
323: C 100 continue
324: C
325:      call KEPV( A(1), 'KKAI', LKKAI, 2*NTGX*NMAT, 'I4', LAST, 0,

```

```

326:      &      JDEBG )
327: C
328: C
329: C      ... transformation of FKAI data to BMC form is performed after
330: C      srcinp routine if source data block is not input.
331: C
332: C
333: C/#IF SOURCE(NEW)
334: C      if ( LSRCSN.ne.0 ) then
335: C/#ENDIF
336: C
337: C      call GTLAST( LTLAST )
338: C      call KEPV( A(1), 'TEMP', LTEMP, NTGX, 'I4', LTLAST, 0, JDEBG
339: C      &      )
340: C      call PKKAI( A(LKKAI), A(LFKAI), NTGX, NMAT, A(LTEMP) )
341: C      call STLAST( 'after PKKAI', LTLAST, LAST )
342: C
343: C/#IF SOURCE(NEW)
344: C      end if
345: C/#ENDIF
346: C
347: C      end if
348: C
349: C      ... NEUTRON-PHOTON MIXED PROBLEM
350: C
351: C      if ( JNEUT*JPHOT.ne.0 ) then
352: C      if ( LWGTG.eq.0 ) then
353: C      write(IMG,*)
354: C      &      '!!! Photon weights (WGTP) are not input !! Set to 1.0'
355: C      call CNTERR( 'WARNING' )
356: C      call KEPV( A(1), 'WGTP', LWGTG, NREG, 'R4', LAST, 1.0, JDEBG
357: C      &      )
358: C      end if
359: C
360: C      *****
361: C      call LABEL( IPR, 'SECONDARY PHOTON WEIGHTS' )
362: C      *****
363: C
364: C      call R4PRNT( 'WGTP', A(LWGTG), NREG, 1, 'REGION', ' ', IPR )
365: C
366: C      call CKVAL4( 'WGTP', A(LWGTG), NREG, IFL, 1 )
367: C      if ( IFL.ne.0 ) then
368: C      write(IMG,*) 'XXX Photon generation weight WGTP includes ',
369: C      &      IFL, ' non positive value.'
370: C      call CNTERR( 'FATAL' )
371: C      end if
372: C
373: C      call KEEP( 'WGTFI', LWGTFI, NREG, 'R4', LAST, JDEBG )
374: C
375: C      call ARYINV( A(LWGTG), A(LWGTFI), NREG )
376: C      end if
377: C-----
378: C      .... /PSOUR/ .....
379: C-----
380: C
381: C
382: C/#IF .NOT.SOURCE(NEW)
383: C
384: C      if ( LPENRG.eq.0 ) then
385: C      if ( JFIXD.ne.0 ) then
386: C      write(IMG,*) '!!! Source energy distribution is not ',
387: C      &      'specified (PENRG) !! OK ?'
388: C      call CNTERR( 'WARNING' )
389: C      else
390: C      call KEEP( 'PENRG', LPENRG, NGROUP*NSOUR, 'R4', LAST, JDEBG

```

src/gmvp/intro2.f

```

391:      &          )
392:      end if
393:      end if
394:      if ( JADJM.ne.0 ) call ADJREV( A(LPENRG), NGROUP, NSOUR )
395: C
396: C-----
397: C      ..... CHANGE ISZON(2,NSOUR) FROM INPUT-ZONE TO ZONE .....
398: C-----
399: C
400:      if ( LISZON.ne.0 ) then
401:      call CISZON( A(LISZON), NSOUR, A(LKINPZ), NZONE )
402:      else
403:      call KEPV( A(1), 'ISZON', LISZON, 2*NSOUR, 'I4', LAST, 0, JDEBG
404:      &          )
405:      end if
406:      if ( NMEMS.eq.0 ) NMEMS = 5
407:      call KEPV( A(1), 'MEMZN', LMEMZN, NMEMS*NSOUR, 'I4', LAST, 0,
408:      &          JDEBG )
409:      call KEEP( 'KENRG', LKENRG, 2*NGROUP*NSOUR, 'I4', LAST, JDEBG )
410: C
411: C-----
412: C      ..... PRINTOUT OF SOURCE DATA & B.M.C TRANSFORMATION .....
413: C-----
414: C
415:      if ( JEIGN.ne.0.and.LIFISM.eq.0 ) then
416:      write(IMG,*) '!!!Data "IFISM" is not specified. Set to 1 !!!'
417:      call KEPV( A(1), 'IFISM', LIFISM, NSOUR, 'I4', LAST, 1, JDEBG )
418:      end if
419: C
420:      call GTLAST( LTLAST )
421:      call KEEP( 'TEMP', LTEMP, NGROUP, 'I4', LTLAST, JDEBG )
422: C
423:      call PRSOUR( NSOUR, NGROUP, NMAT, JTIME, JEIGN, A(LKINPZ),
424:      &          A(LKSOUR), A(LISZON), A(LSOUR), A(LPSPAC), A(LPENRG),
425:      &          A(LKENRG), A(LNSTIM), A(LSTIM), A(LPSTIM), A(LISTIM),
426:      &          A(LNSANG), A(LSANG), A(LSAXIS), A(LPSANG), A(LISANG),
427:      &          A(LIFISM), A(LFKAI), A(LKKAI), A(LTEMP) )
428: C
429:      call STLAST( 'PRSOU', LTLAST, LAST )
430:
431: C/#ENDIF
432:
433: C
434: C-----
435: C      .... /PTALY/ .....
436: C-----
437: C
438: C
439: C      =====
440:      call HEADER( IPR, 'TALLY AND RESPONSE PARAMETERS' )
441: C      =====
442: C
443:      call GTLAST( LLS0 )
444:      call LGTLST( LLL0 )
445: C
446:      call LKEPV( H(1), 'WSUM', LWSUM, 1, 'R8', LAST, 0.0D0, JDEBG )
447:      call LKEPV( H(1), 'WLEK', LWLEK, 1, 'R8', LAST, 0.0D0, JDEBG )
448:      call LKEPV( H(1), 'AAVT', LAAVT, 3, 'R4', LAST, 0.0E0, JDEBG )
449:      call LKEPV( H(1), 'XAVT', LXAVT, 3, 'R4', LAST, 0.0E0, JDEBG )
450:      call LKEPV( H(1), 'EAVT', LEAVT, 1, 'R4', LAST, 0.0E0, JDEBG )
451:      call LKEPV( H(1), 'NLOST', LNLOST, 1, 'I4', LAST, 0, JDEBG )
452: C
453: C      NPKIND = 0
454: C      if ( JNEUT.ne.0 ) NPKIND = NPKIND + 1
455: C      if ( JPHOT.ne.0 ) NPKIND = NPKIND + 1

```

```

456: C
457:      if ( JRESP.eq.0.and.NRESP2.eq.0 ) NRESP = 0
458:      if ( JRESP.ne.0.and.NRESP.ne.0.and.LRESP.eq.0 ) then
459:      write(IMG,*)
460:      &      '!!! No response function is specified. All assumed to be 1.0'
461:      call CNTERR( 'WARNING' )
462:      call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4', LAST, 1.0,
463:      &          JDEBG )
464:      end if
465:      call LKEPV( H(1), 'FLTR', LFLTR, NGROUP*NREG, 'R8', LAST, 0D0,
466:      &          JDEBG )
467:      call LKEPV( H(1), 'SFLTR', LSFLTR, NGROUP*NTREG*2, 'R8', LAST,
468:      &          0D0, JDEBG )
469:      call LKEPV( H(1), 'FLCL', LFLCL, NGROUP*NREG, 'R8', LAST, 0D0,
470:      &          JDEBG )
471:      call LKEPV( H(1), 'SFLCL', LSFLCL, NGROUP*NTREG*2, 'R8', LAST,
472:      &          0D0, JDEBG )
473:      if ( JRESP.ne.0 ) then
474:      call LKEPV( H(1), 'SRETR', LSRETR, NTREG*NRESP*2, 'R8', LAST,
475:      &          0D0, JDEBG )
476:      call LKEPV( H(1), 'SRECL', LSRECL, NTREG*NRESP*2, 'R8', LAST,
477:      &          0D0, JDEBG )
478:      end if
479: C
480: C-----
481: C      For weighted time of flight
482: C-----
483: C
484:      if ( JTIME.ne.0 ) then
485:      call LKEPV( H(1), 'TFLH', LTLH, NPKIND, 'R8', LAST, 0D0, JDEBG
486:      &          )
487:      call LKEPV( H(1), 'TFLHS', LTLHS, NPKIND*2, 'R8', LAST, 0D0,
488:      &          JDEBG )
489:      end if
490: C
491: C-----
492: C      .... FOR POINT DETECTOR ....
493: C-----
494: C
495:      if ( NPDET.eq.0 ) JPTDT = 0
496:      if ( JPTDT.eq.0 ) NPDET = 0
497: C
498:      if ( JPTDT.ne.0 ) then
499: C
500: C      .... RELOCATE XPDET ARRAY FOR USE IN RANDOM WORK ....
501: C      AND CALCULATE LATTICE-RELATED PARAMETER IF NECESSARY.
502: C
503:      NPLEN = 2 + (NEST+1)*3
504:      call KEPV( A(1), 'XPDET', LTEMP, NPLEN*NPDET, 'R8', LAST, 0.0D0,
505:      &          JDEBG )
506:      call KEPV( A(1), 'IPDET', LIPDET, NPDET*2, 'I4', LAST, 0, JDEBG
507:      &          )
508:      if ( JLATT.ne.0 ) then
509:      call KEPV( A(1), 'IPDT2', LIPDT2, NEST*3*NPDET, 'I4', LAST, 0,
510:      &          JDEBG )
511:      end if
512:      if ( LJPUUSD.eq.0 ) then
513:      call KEPV( A(1), 'JPUUSD', LJPUUSD, NPDET, 'I4', LAST, 1,
514:      &          JDEBG )
515:      end if
516:      if ( LJSPDT.eq.0 ) then
517:      if ( JEIGN.ne.0 ) then
518:      call KEPV( A(1), 'JSPDT', LJSPDT, NPDET, 'I4', LAST, 0,
519:      &          JDEBG )
520:      else

```


src/gmvp/intro2.f

```

521:      call KEPV(A(1),'JSPDT', LJSPDT, NPDET, 'I4', LAST, 1,
522:      &          JDEBG )
523:      end if
524:    end if
525: C
526:      if ( LSTCK(0).lt.0 ) then
527:        write(IMSG,7140) 'point detector setting.'
528:        call PRSTOP( 1, 'MEMORY OVER.' )
529:        stop 999
530:      end if
531: C
532:      call RXPDET( A, JVMNT, JLATT, JSIMP, JHLAT, JEIGN,
533:      &          A(LXPDET), A(LTEMP), NPDET, NPLEN, NEST,
534:      &          A(LIPDET), A(LIPDT2), A(LJSPDT),
535:      &          A(LIPCEL), A(LKCELL), A(LKDALT), A(LKZMAT), A(LMLBZZ),
536:      &          A(LKZREG), A(LISUSP), A(LKSPSU), A(LKTCSP),
537:      &          NSDA, NZDA, NZONE, NCELL, NLBZ,
538:      &          NSPACE, NSUZON, NUNV, NKTCSF )
539: C
540:      LXPDET = LTEMP
541: C
542: C ..... POINT DETECTOR TALLY .....
543: C
544:      call LKEPV( H(1), 'FLPD', LFLPD, NGROUP*NPDET, 'R8', LAST, OD0,
545:      &          JDEBG )
546:      call LKEPV( H(1), 'SFLPD', LSFLPD, NGROUP*NPDET*2, 'R8', LAST,
547:      &          OD0, JDEBG )
548:      if ( JRESP.ne.0 ) then
549:        call LKEPV( H(1), 'REPD', LREPD, NPDET*NRESP, 'R8', LAST,
550:        &          OD0, JDEBG )
551:        call LKEPV( H(1), 'SREPD', LSREPD, NPDET*NRESP*2, 'R8',
552:        &          LAST, OD0, JDEBG )
553:      end if
554: C
555: C
556:    end if
557: C
558: C-----
559: C ..... for speciall tally (when $TALLY block is specified)
560: C-----
561: C
562:      if ( NDTALY.gt.0 ) then
563:        call LKEPV( H(1), 'DTALY', LDTALY, NLDTAL, 'R8', LAST, OD0,
564:        &          JDEBG )
565:      end if
566:      if ( NETALY.gt.0 ) then
567:        call LKEPV( H(1), 'ETALY', LETALY, NLETAL*2, 'R8', LAST, OD0,
568:        &          JDEBG )
569:      end if
570:      if ( LJTEVE.eq.0 ) then
571:        call KEPV( A(1), 'JTEVE', LJTEVE, NTEVE, 'I4', LAST, 0, JDEBG )
572:      end if
573: C
574: C ..... Use free-lattice-frame mode for surface tally
575: C
576:      if ( JTSRF.ne.0.and.JLATT.ne.0 ) JFISX = 1
577: C
578: C-----
579: C .... regionwise monitoring ....
580: C-----
581: C
582:      if ( JMNTR.ne.0 ) then
583:        call LKEPV( H(1), 'NCLSN', LNCLSN, NGROUP*NREG, 'R8', LAST,
584:        &          OD0, JDEBG )
585:        call LKEPV( H(1), 'WCLSN', LWCLSN, NGROUP*NREG, 'R8', LAST,

```

```

586:      &          OD0, JDEBG )
587:      call LKEPV( H(1), 'NLEAK', LNLEAK, NGROUP*NREG, 'R8', LAST,
588:      &          OD0, JDEBG )
589:      call LKEPV( H(1), 'WLEAK', LWLEAK, NGROUP*NREG, 'R8', LAST,
590:      &          OD0, JDEBG )
591:      call LKEPV( H(1), 'NABSB', LNABSB, NGROUP*NREG, 'R8', LAST,
592:      &          OD0, JDEBG )
593:      call LKEPV( H(1), 'WABSB', LWABSB, NGROUP*NREG, 'R8', LAST,
594:      &          OD0, JDEBG )
595:      call LKEPV( H(1), 'NECUT', LNECUT, NREG, 'R8', LAST, OD0, JDEBG
596:      &          )
597:      call LKEPV( H(1), 'WECUT', LWECUT, NREG, 'R8', LAST, OD0, JDEBG
598:      &          )
599:      call LKEPV( H(1), 'NTCUT', LNTCUT, NGROUP*NREG, 'R8', LAST,
600:      &          OD0, JDEBG )
601:      call LKEPV( H(1), 'WTCUT', LWTCUT, NGROUP*NREG, 'R8', LAST,
602:      &          OD0, JDEBG )
603:      call LKEPV( H(1), 'NKILD', LNKILD, NGROUP*NREG, 'R8', LAST,
604:      &          OD0, JDEBG )
605:      call LKEPV( H(1), 'WKILD', LWKILD, NGROUP*NREG, 'R8', LAST,
606:      &          OD0, JDEBG )
607:      call LKEPV( H(1), 'NSURV', LNSURV, NGROUP*NREG, 'R8', LAST,
608:      &          OD0, JDEBG )
609:      call LKEPV( H(1), 'WSURV', LWSURV, NGROUP*NREG, 'R8', LAST,
610:      &          OD0, JDEBG )
611:      call LKEPV( H(1), 'NSPLT', LNSPLT, NGROUP*NREG, 'R8', LAST,
612:      &          OD0, JDEBG )
613:      call LKEPV( H(1), 'WSPLT', LWSPLT, NGROUP*NREG, 'R8', LAST,
614:      &          OD0, JDEBG )
615:    end if
616: C
617: C-----
618: C .... event monitor ....
619: C-----
620: C
621: Cccc nevent = 20
622: C      NEVENT = 30
623: C
624:      call LKEPV( H(1), 'NCNTR', LNCNTR, NEVENT, 'R8', LAST, OD0, JDEBG
625:      &          )
626:      call LKEPV( H(1), 'WCNTR', LWCNTR, NEVENT, 'R8', LAST, OD0, JDEBG
627:      &          )
628: C
629: C-----
630: C ..... CHECK & PRINTOUT OF ENERGY & TIME BOUNDARY .....
631: C
632: C ... value range of forged lethargy does not exceed "XLOGMX"
633: C
634: C      XLOGMX = LOG(1.0E35)
635: C
636: C ... NGP1 & NGP2 might be changed above ...
637: C
638: C      NGP10 = NGP1
639: C      NGP20 = NGP2
640: C      if ( JNEUT*JPHOT.ne.0 ) then
641: C        if ( JADJM.ne.0 ) then
642: C          NGP1 = NGP20
643: C          NGP2 = NGP10
644: C        end if
645: C      else if ( JPHOT.ne.0 ) then
646: C        NGP1 = 0
647: C        NGP2 = NGP10
648: C      end if
649: C
650: C      if ( JNEUT.ne.0 ) then

```

src/gmvp/intro2.f

```

651:         if ( LENGYB.eq.0 ) then
652:             XDU = 1.0
653:             if ( REAL(NGP1).gt.XLOGMX ) then
654:                 XDU = XLOGMX/REAL(NGP1)
655:             end if
656:
657:             write(IMG, '(1X,A,A/3X,A,A,E11.4,A/)' )
658:             & '!!! No neutron energy boundary input',
659:             & ' ( ENGYB(...) or ENGYB.N(...) ).',
660:             & ' Energy structure having constant lethargy width is',
661:             & ' assumed. (WIDTH =', XDU, ' )'
662:
663:             call CNTERR( 'WARNING' )
664:
665:             NGB = NGROUP + 1
666:             if ( JPHOT.ne.0 ) NGB = NGB + 1
667:             call KEEP( 'ENGYB', LENGYB, NGP1+1, 'R4', LAST, JDEBG )
668:             call KEEP( 'ENGYB', LENGYB, NGB, 'R4', LAST, JDEBG )
669: C
670:             call LTHRGY( A(LENGYB), XDU, 1.0, NGP1+1 )
671:
672: C             do 120 I = 0, NGP1
673: C                 A(LENGYB+I) = EXP(-FLOAT(I)*XDU)
674: C 120 continue
675:             else
676:                 call CKODR4( A(LENGYB), NGP1+1, 1, INVLD )
677:                 if ( INVLD.ne.0 ) then
678:                     write(IMG,*)
679:                     & 'XXX ORDER OF NEUTRON ENERGY BOUNDARY IS INCORRECT.'
680:                     call CNTERR( 'FATAL' )
681:                 end if
682:             end if
683:         end if
684: C
685:         if ( JPHOT.ne.0 ) then
686:             if ( LENGPB.eq.0 ) then
687:                 XDU = 1.0
688:                 if ( REAL(NGP2).gt.XLOGMX ) then
689:                     XDU = XLOGMX/REAL(NGP2)
690:                 end if
691:
692:                 write(IMG, '(1X,A,A/3X,A,A,E11.4,A/)' )
693:                 & '!!! No photon energy boundary input',
694:                 & ' ( ENGPB(...) or ENGYB.P(...) ).',
695:                 & ' Energy structure having constant lethargy width is',
696:                 & ' assumed. (WIDTH =', XDU, ' )'
697:                 call CNTERR( 'WARNING' )
698:
699:                 call KEEP( 'ENGPB', LENGPB, NGP2+1, 'R4', LAST, JDEBG )
700:                 call LTHRGY( A(LENGPB), XDU, 1.0, NGP2+1 )
701: C                 do 130 I = 0, NGP2
702: C                     A(LENGPB+I) = EXP(-REAL(I)*XDU)
703: C 130 continue
704:                 NGP2S = NGP1 + 2
705:                 if ( LENGYB.eq.0 ) then
706:                     NGB = NGP2 + 1
707:                     NGP2S = 1
708:                     call KEEP( 'ENGYB', LENGYB, NGB, 'R4', LAST, JDEBG )
709:                 end if
710:                 call ATRANS( A(LENGPB), A(LENGYB+NGP2S-1), NGP2+1 )
711:
712:             else
713:                 call CKODR4( A(LENGPB), NGP2+1, 1, INVLD )
714:                 if ( INVLD.ne.0 ) then
715:                     write(IMG,*)

```

```

716:             & 'XXX ORDER OF NEUTRON ENERGY BOUNDARY IS INCORRECT.'
717:             call CNTERR( 'FATAL' )
718:         end if
719:     end if
720: end if
721: C
722: C *****
723: call LABEL( IPR, 'ENERGY BOUNDARIES (EV)' )
724: C *****
725: C
726: do 100 IK = 1, KPLIM
727:     if ( JKPAP(IK).ne.0 ) then
728:         write(IPR, '(3X, ''=== ', A, '' ==='//)' )
729:         & KPSYM(1,IK) (:ICLEN2(KPSYM(1,IK)))
730:         call R4PRNT( 'ENERGY', A(LENGYB+KENGPIK-1), NGP(IK)+1, 1,
731:             & 'BOUNDARY', ' ', IPR )
732:     end if
733: 100 continue
734:
735: C     if ( JNEUT.ne.0 ) then
736: C         write(IPR, '(3X, ''=== NEUTRON ==='//)' )
737: C         call R4PRNT( 'ENERGY', A(LENGYB), NGP1+1, 1, 'BOUNDARY', ' ',
738: C             & IPR )
739: C     end if
740: C     if ( JPHOT.ne.0 ) then
741: C         write(IPR, '(3X, ''=== PHOTON ==='//)' )
742: C         call R4PRNT( 'ENERGY', A(LENGPB), NGP2+1, 1, 'BOUNDARY', ' ',
743: C             & IPR )
744: C     end if
745: C
746: C     if ( JTIME.eq.0 ) then
747: C         NTIME = 1
748: C         call KEEP( 'TIMEB', LTIMEB, NTIME+1, 'R4', LAST, JDEBG )
749: C     end if
750: C
751: C     if ( JTIME.ne.0 ) then
752: C         if ( LTIMEB.eq.0 ) then
753: C             write(IMG,*) 'XXX NO TIME BIN BOUNDARY IS SPECIFIED.'
754: C             call CNTERR( 'FATAL' )
755: C         else
756: C             *****
757: C             call LABEL( IPR, 'TIME BOUNDARIES (SEC)' )
758: C             *****
759: C             call R4PRNT( 'TIMEB', A(LTIMEB), NTIME+1, 1, 'TIME', ' ',
760: C                 & IPR )
761: C             call CKODR4( A(LTIMEB), NTIME+1, 2, INVLD )
762: C             if ( INVLD.ne.0 ) then
763: C                 write(IMG, '(1X,A)' )
764: C             & 'XXX Time bin boundary is not in increasing order.'
765: C             call CNTERR( 'FATAL' )
766: C         end if
767: C         if ( TCUT.eq.TCUTDM ) then
768: C             TCUT = A(LTIMEB+NTIME)
769: C             write(IPR, '(1X,A)' )
770: C             & '<<MESSAGE>> Cutoff time (TCUT) is set ',
771: C             & 'to the highest time bin boundary.'
772: C             call CNTERR( 'MESSAGE' )
773: C         end if
774: C
775: C     ... adjust time boundary and TCUT for periodic time option
776: C
777: C     if ( JPTIM.ne.0 ) then

```

src/gmvp/intro2.f

```

781:         if ( TCUT.ne.A(LTIMEB+NTIME) ) then
782:             TCUT = A(LTIMEB+NTIME)
783:             write(IPR, '(//1X,A,A)')
784:         &         '<<MESSAGE>> Cutoff time (TCUT) is set',
785:         &         ' to the highest time bin boundary.'
786:         call CNTERR( 'MESSAGE' )
787:     end if
788:     if ( A(LTIMEB).gt.0.0 ) then
789:         A(LTIMEB) = 0.0
790:         write(IPR, '(//1X,A,A)')
791:     &         '<<MESSAGE>> the lowest time bin boundary',
792:     &         ' is set to be 0.0.'
793:     call CNTERR( 'MESSAGE' )
794:     end if
795: end if
796: end if
797: C
798: C ... calculate group averaged velocity ...
799: C
800:     if ( LVEL.eq.0 ) then
801:         call KEEP( 'VEL', LVEL, NGROUP, 'R4', LAST, JDEBG )
802: C
803:         if ( JNEUT.ne.0 ) then
804:             call NGVEL( A(LENGYB), A(LVEL), NGP1, NGROUP )
805:         end if
806: C
807:         if ( JPHOT.ne.0 ) then
808:             call PGVEL( A(LVEL), NGP1, NGP2, NGROUP )
809:         end if
810:     end if
811: C
812: C *****
813: call LABEL( IPR, 'GROUP AVERAGED PARTICLE VELOCITIES (CM/SEC)'
814: & )
815: C *****
816: C
817:     call R4PRNT( 'VEL', A(LVEL), NGROUP, 1, 'GROUP', ' ' IPR )
818: end if
819: C
820: C ... adjoint reversal of energy boundary & velocities ...
821: C
822:     if ( JADJM.ne.0 ) then
823:         NGB = NGROUP + 1
824:         if ( JNEUT*JPHOT.ne.0 ) NGB = NGB + 1
825:         call ADJREV( A(LENGYB), NGB, 1 )
826:         if ( JPHOT.ne.0 ) call ADJREV( A(LENGPB), NGP2+1, 1 )
827:         if ( JTIME.ne.0 ) call ADJREV( A(LVEL), NGROUP, 1 )
828:     end if
829: C
830: C-----
831: C ..... CALULATION & PRINTOUT OF REGION VOLUME .....
832: C-----
833:     if ( LTRVOL.eq.0.and.LRVOL.eq.0.and.LVOL.eq.0 ) then
834:         write(IMG,7060)
835:         7060 format(/1X,'!!! Neither tally-region volumes nor region ',
836:         &         'volumes are input, So all tally-region volumes are ',
837:         &         'set to 1.0//')
838:         call CNTERR( 'WARNING' )
839:         call KEPV( A(1), 'TRVOL', LTRVOL, NTREG, 'R4', LAST, 1.0, JDEBG
840:         & )
841:     end if
842: C
843:     if ( LRVOL.eq.0 ) then
844:         if ( LVOL.eq.0 ) then
845:             write(IPR,7080)

```

```

846: 7080 format(/1X,'<<MESSAGE>> Volumes of each input zone are not',
847:         &         ' input.//10X,
848:         &         'Their volumes are set so that volumes of each',
849:         &         ' region are equal to 1.0//')
850:     call CNTERR( 'MESSAGE' )
851: C
852:     call KEPV( A(1), 'VOL', LVOL, NINPZ, 'R4', LAST, 0.0, JDEBG
853:     & )
854:     call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 1.0, JDEBG
855:     & )
856: C
857:     if ( LSTCK(0).lt.0 ) then
858:         write(IMG,7160) 'region volume setting.'
859:         call CNTERR( 'FATAL' )
860:     else
861:         call CRVOL1( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ,
862:         &         A(LKMAT) )
863:     end if
864: else
865:     call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 0.0, JDEBG
866:     & )
867:     write(IPR,7100)
868:     7100 format(/1X,'<<MESSAGE>> Calculate region volumes from zone',
869:     &         ' volumes//')
870:     call CNTERR( 'MESSAGE' )
871: C
872:     if ( LSTCK(0).lt.0 ) then
873:         write(IMG,7160) 'region volume setting.'
874:         call CNTERR( 'FATAL' )
875:     else
876:         call CRVOL( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ )
877:     end if
878: end if
879: else
880:     if ( LVOL.eq.0 ) then
881:         call KEPV( A(1), 'VOL', LVOL, NINPZ, 'R4', LAST, 0.0, JDEBG
882:         & )
883:         if ( LSTCK(0).lt.0 ) then
884:             write(IMG,7160) 'region volume setting.'
885:             call CNTERR( 'FATAL' )
886:         else
887:             call CRVOL1( A(LRVOL), NREG, A(LVOL), A(LKREG), NINPZ,
888:             &         A(LKMAT) )
889:         end if
890:     end if
891: end if
892: C
893:     if ( LTRVOL.eq.0 ) then
894:         write(IMG,7120)
895:         7120 format(/1X,'!!! Tally-region volumes are calculated from',
896:         &         ' those of composing regions.//')
897:         call CNTERR( 'WARNING' )
898:         call KEPV( A(1), 'TRVOL', LTRVOL, NTREG, 'R4', LAST, 0.0, JDEBG
899:         & )
900: C
901:     if ( LSTCK(0).lt.0 ) then
902:         write(IMG,7160) 'region volume setting.'
903:         call CNTERR( 'FATAL' )
904:     else
905:         call CRVOL1( A(LTRVOL), NTREG, A(LRVOL), NREG, A(LIPTRG),
906:         &         A(LLPTRG) )
907:     end if
908: end if
909: C
910:     if ( LSTCK(0).lt.0 ) then

```

src/gmvp/intro2.f

```

911:      write(IMG,7160) 'region volume printout.'
912:      call CNTERR( 'FATAL' )
913:      else
914: C
915: C *****
916:      call LABEL( IPR, 'VOLUMES OF EACH INPUT ZONE' )
917: C *****
918: C
919:      call R4PRNT( 'VOL', A(LVOL), NINPZ, 1, 'INP.ZONE', ' ', IPR )
920: C
921: C *****
922:      call LABEL( IPR, 'VOLUMES OF EACH REGION' )
923: C *****
924: C
925:      call R4PRNT( 'RVOL', A(LRVOL), NREG, 1, 'REGION', ' ', IPR )
926: C
927: C *****
928:      call LABEL( IPR, 'VOLUMES OF EACH TALLY-REGION' )
929: C *****
930: C
931:      call R4PRNT( 'TRVOL', A(LTRVOL), NTREG, 1, 'T-REGION', ' ', IPR
932: &      )
933:      end if
934: C
935: C-----
936: C ..... PRINTOUT OF RESPONSE FUNCTION .....
937: C-----
938: C
939:      if ( JRESP.ne.0 .or. NRESP2.gt.0 ) then
940: C
941: C *****
942:      call LABEL( IPR,
943: &      'RESPONSE FUNCTION FOR REACTION RATE CALCULATION' )
944: C *****
945: C
946:      call R4PRNT( 'RESP', A(LRESP), NGROUP, NRESP, 'GROUP'
947: &      'RESPONSE', IPR )
948:      if ( JADJM.ne.0 ) call ADJREV( A(LRESP), NGROUP, NRESP )
949:      end if
950: C
951:      call GTLAST( LLS1 )
952:      call LGTLST( LLL1 )
953: C
954:      write(IPR, '(1X,A,I12,A,I12)')
955:      &      ' MEMORY FOR TALLY DATA (WORDS) : TASK SHARED ', LLS1
956:      &      - LLS0, ' TASK LOCAL ', LLL1 - LLL0
957:      write(IPR, '(1X,A,I12,A,I12,A)')
958:      &      ' MEMORY POSITION OF TALLY DATA: FROM A(', LLS0,
959:      &      ') TO A(', LLS1 - 1, ') '
960:      write(IPR, '(1X,A,I12,A,I12,A)')
961:      &      ' MEMORY POSITION OF TALLY DATA: FROM H(', LLL0,
962:      &      ') TO H(', LLL1 - 1, ') '
963: C
964:      call MEMUSE( 'TALLY', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL, LIMITC
965: &      )
966: C-----
967: C ..... /PSOUR/ .....
968: C-----
969: C
970: C/#IF SOURCE(NEW)
971: C
972: C ... LSRCS = 0 : probably source input is in old-fashioned way.
973: C      Convert to new type of source generation data
974: C
975:      if ( LSRCS.eq.0 ) then

```

```

976:      if ( JEIGN.ne.0.and.LIFISM.eq.0 ) then
977:      write(IMG,*)
978:      &      '!!! Data "IFISM" is not specified. Set to 1 !!!'
979:      call CNTERR( 'WARNING' )
980:      call KEPV( A(1), 'IFISM', LIFISM, NSOUR, 'I4', LAST, 1,
981: &      JDEBG )
982:      end if
983: C
984:      if ( JADJM.ne.0.and.LPENRG.ne.0 ) then
985:      do 110 INSOR = 1, NSOUR
986:          LS = LPENRG + NGROUP*(INSOR-1)
987:          if ( NGP1.ge.2 ) call ADJREV( A(LS), NGP1, 1 )
988:          if ( NGP2.ge.2 ) call ADJREV( A(LS+NGP1), NGP2, 1 )
989:      110      continue
990:      end if
991: C-----
992: C      NSOUR may be increased when problem is neutron/photon
993: C      coupled and PENRG(1:ngroup) has non zero elements for both
994: C      of neutron and photon energy group.
995: C-----
996:      if ( LSTCK(0).lt.0 .or. LSTCKL(0).lt.0 ) then
997:      write(IMG,7140) 'conversion of old type source input.'
998:      call PRSTOP( 1, 'MEMORY OVER.' )
999:      stop 999
1000:      end if
1001: C
1002:      NUC0 = 0
1003:      call SRCINP( A, H, CHA, LIMIT, LIMITL, LIMITC, LSRCS, NSRCS,
1004: &      LSOUR, LIDSR, 'GMVP', JKPAR(1), KPLIM, JFISS, JEIGN,
1005: &      JTIME, JNEUT, JPHOT, JDEBG, MWVEC, MVSTK, MKREJ,
1006: &      NERR, A(LIDMAT), ' ', NUC0, NTGX, LKSOUR,
1007: &      LISZON, LPSPAC, LPENRG, LIFISM, LFKAI, LENGYB, LENGPB,
1008: &      IDUMMY )
1009: C
1010: C .... call pk kai here for eigenvalue problem ...
1011: C
1012:      if ( JFISS.ne.0 ) then
1013:      call KEPV( A(1), 'TEMP', LTEMP, NTGX, 'I4', LTLAST, 0, JDEBG
1014: &      )
1015:      call PKKAI( A(LKKAI), A(LFKAI), NTGX, NMAT, A(LTEMP) )
1016:
1017:      call STLAST( 'after PKKAI', LTEMP, LAST )
1018:      end if
1019:      end if
1020:
1021:      if ( NMEMS.eq.0 ) NMEMS = 5
1022:      call LKEPV( H(1), 'MEMZN', LMEMZN, NMEMS*NSOUR, 'I4', LAST, 0,
1023: &      JDEBG )
1024: C/#ENDIF
1025: C
1026: C ... NGP1 & NGP2 might be changed above ...
1027: C
1028:      NGP1 = NGP10
1029:      NGP2 = NGP20
1030: C-----
1031:      if ( JADJM.ne.0 ) then
1032:      call LABEL( IPR, 'ENERGY GROUPS ARE REVERSED IN ADJOINT MODE' )
1033:      write(IMG,*)
1034:      &      '!!! All input data except $SOURCE block are reversed !!!'
1035:      write(IMG,*) '      Data in $SOURCE block are not reversed !!!'
1036:      call CNTERR( 'WARNING' )
1037:      end if
1038: C-----
1039: C
1040: C

```

src/gmvp/intro2.f

```

1041: C-----
1042: C      .... INPUT RESTART FILE & DETERMINE SIZE OF KEFF-TALLY ARRAY
1043: C-----
1044: C
1045: C      NTHIST = 0
1046: C      NBATCH = 0
1047: C      if ( JARST.ne.0 .or. JREST.ne.0 ) then
1048: C          call RESTIO( NTHIST, H(LWSUM), H(LWLEK), IRAND, NBATCH, NFISB,
1049: C      &                NGROUP, NRESP, NREG, NTREG, NMEMO, NZONE, NPART, NLATT,
1050: C      &                NEST, NPDET, NETALY, NETRV )
1051: C      end if
1052: C
1053: C-----
1054: C      call HEADER( IPR, 'Event stack, particle bank and working area' )
1055: C-----
1056: C
1057: C-----
1058: C      check batch size and sub-batch size (added Feb 2000)
1059: C      NHSUB may be given an appropriate value if not specified.
1060: C-----
1061: C
1062: C      NBANK1 = NBANK
1063: C      NHSUB1 = NHSUB
1064: C      NHIST1 = NHIST
1065: C
1066: C      call CKBATS( NHSUB, NHIST, NBANK )
1067: C
1068: C-----
1069: C      ... check and reset (if necessary) history control & bank parameters
1070: C      for multi tasking
1071: C-----
1072: C
1073: C      NPART0 = NPART
1074: C      if ( NTASK.gt.1 ) then
1075: C          call TSKCTL( 'GMVP', JEIGN, JWWND, JIMPT, JFISS, JREPR, NPART,
1076: C      &                NTHIST, NBATCH, NHIST, NHSUB, NBANK, IMEMAX, NFBANK,
1077: C      &                NFBANK0, NPART0, NHIST0, NHSUB0, NBANK0, NFBANK0 )
1078: C      end if
1079: C
1080: C      call KEEP( 'ITASK', LITASK, MNTASK*NTASK, 'I4', LAST, JDEBG )
1081: C      call KEEP( 'IRANT', LIRANT, NTASK, 'I4', LAST, JDEBG )
1082: C      call KEEP( 'IRSUT', LIRSUT, NTASK, 'I4', LAST, JDEBG )
1083: C      call KEEP( 'IRSMT', LIRSMT, NTASK, 'I4', LAST, JDEBG )
1084: C      call KEEP( 'IRSLT', LIRSLT, NTASK, 'I4', LAST, JDEBG )
1085: C
1086: C-----
1087: C      .... /STACK/ .....
1088: C-----
1089: C
1090: C      call GTLAST( LLS0 )
1091: C      call LGTLST( LLL0 )
1092: C
1093: C      if ( NBANK.eq.0 ) then
1094: C          NBANK = NHIST
1095: C          if ( JIMPT.ne.0 .or. JWWND.ne.0 .or. JFISS.ne.0 ) then
1096: C              NBANK = 2*NHIST
1097: C          end if
1098: C          write(IMG,*)
1099: C      &          '!!! Bank length (NBANK) is not defined. Set to ',
1100: C      &          NBANK
1101: C          call CNTERR( 'WARNING' )
1102: C      end if
1103: C
1104: C      RRNB = 0.0
1105: C      if ( NBANK1.gt.0.and.NHSUB1.gt.0 ) then

```

```

1106: C          RRNB = MAX(1.0D0,DBLE(NBANK1)/NHSUB1)
1107: C      end if
1108: C
1109: C      if ( NBANK.eq.0 ) then
1110: C          NBANK = NHSUB
1111: C          if ( RRNB.ne.0 ) then
1112: C              NBANK = NINT(RRNB*NHSUB)
1113: C          else if ( JIMPT.ne.0 .or. JWWND.ne.0 .or. JFISS.ne.0 ) then
1114: C          else if ( JIMPT.ne.0 .or. JWWND.ne.0 .or.
1115: C      &                (JFIXD.ne.0.and.JFISS.ne.0) ) then
1116: C              NBANK = 2*NHSUB
1117: C          end if
1118: C          write(IMG,*)
1119: C      &          '!!! Bank length (NBANK) is not defined. Set to ',
1120: C      &          NBANK
1121: C          call CNTERR( 'WARNING' )
1122: C
1123: C      else if ( NBANK.lt.NHSUB ) then
1124: C
1125: C          NBANK = NHSUB
1126: C          if ( RRNB.ne.0 ) then
1127: C              NBANK = NINT(RRNB*NHSUB)
1128: C          else if ( JIMPT.ne.0 .or. JWWND.ne.0 .or. JFISS.ne.0 ) then
1129: C          else if ( JIMPT.ne.0 .or. JWWND.ne.0 .or.
1130: C      &                (JFIXD.ne.0.and.JFISS.ne.0) ) then
1131: C              NBANK = 2*NHSUB
1132: C          end if
1133: C          write(IMG,'(1X,A,A,I10)')
1134: C      &          '!!! Particle bank length (NBANK) is smaller than "NHSUB".',
1135: C      &          ' Set to ', NBANK
1136: C          call CNTERR( 'WARNING' )
1137: C      end if
1138: C
1139: C      call LKEPV( H(1), 'LSFFL', LLSFFL, NBANK, 'I4', LAST, 0, JDEBG )
1140: C      call LKEPV( H(1), 'NFFL', LNFFL, NZONE+1, 'I4', LAST, 0, JDEBG )
1141: C      call LKEPV( H(1), 'IZFFL', LIZFFL, NBANK, 'I4', LAST, 0, JDEBG )
1142: C      call LKEPV( H(1), 'LSCOL', LLSCOL, NBANK, 'I4', LAST, 0, JDEBG )
1143: C      NCOLS = 0
1144: C
1145: C      call LKEPV( H(1), 'LSSRC', LLSSRC, NBANK, 'I4', LAST, 0, JDEBG )
1146: C      call LKEPV( H(1), 'NNXT', LNNXT, NZONE+1, 'I4', LAST, 0, JDEBG )
1147: C      call LKEPV( H(1), 'IZNXT', LIZNXT, NBANK, 'I4', LAST, 0, JDEBG )
1148: C      IDEFER = 0
1149: C
1150: C      call LKEPV( H(1), 'LSDED', LLSDED, NBANK, 'I4', LAST, 0, JDEBG )
1151: C      NDEAD = 0
1152: C-----
1153: C      .... IN THE CASE OF REFLECTIVE BOUNDARY .....
1154: C-----
1155: C      if ( JREFL.ne.0 ) then
1156: C          call KEPV( A(1), 'KSREF', LKSREF, NZDA, 'I4', LAST, 0, JDEBG )
1157: C
1158: C      ..... SEARCH REFLECTIVE BOUNDARIES .....
1159: C
1160: C      call REMAIN( 'KKREF', NLTEMP, 'I4', LAST )
1161: C      call KEEP( 'KKREF', LKKREF, NLTEMP, 'I4', LAST, JDEBG )
1162: C
1163: C      call REFZON( A(LSDA), A(LKZDA), A(LKZAA), A(LKZMAT), A(LKSREF),
1164: C      &            NZDA, NZONE, A(LKKREF), NLTEMP, NREFS, JDEBG )
1165: C
1166: C      call RESIZE( 'KKREF', LKKREF, 2*NREFS, 'I4', LAST )
1167: C
1168: C      .... refraction stack
1169: C
1170: C      call LKEPV( H(1), 'NBREF', LNBREF, NREFS+1, 'I4', LAST, 0,

```

src/gmvp/intro2.f

```

1171:      &          JDEBG )
1172:      call LKEPV( H(1), 'LSREF', LLSREF, NBANK, 'I4', LAST, 0, JDEBG
1173:      &          )
1174:      call LKEPV( H(1), 'IZREF', LIZREF, NBANK, 'I4', LAST, 0, JDEBG
1175:      &          )
1176:      call LKEPV( H(1), 'ISREF', LISREF, NBANK, 'I4', LAST, 0, JDEBG
1177:      &          )
1178:      end if
1179: C
1180: C-----
1181: C ..... IN THE CASE OF LATTICE GEOMETRY .....
1182: C-----
1183: C
1184:      if ( JLATT.ne.0 ) then
1185:      call LKEPV( H(1), 'LSLAT', LLSLAT, NBANK, 'I4', LAST, 0, JDEBG
1186:      &          )
1187:      call LKEPV( H(1), 'IZLAT', LIZLAT, NBANK, 'I4', LAST, 0, JDEBG
1188:      &          )
1189:      call LKEPV( H(1), 'LNLT', LNLTL, NLBZ+1, 'I4', LAST, 0, JDEBG )
1190:      end if
1191: C
1192: C-----
1193: C ..... IN THE CASE OF POINT DETECTOR .....
1194: C      ( STACK FOR IMAGINARY PARTICLES )
1195: C-----
1196: C
1197:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1198:      if ( IMPMAX.le.0 ) then
1199:      write(IPR,'(/1X,A,A,I10)')
1200:      &      '<<MESSAGE>> BANK LENGTH FOR IMAGINARY PARTICLES ',
1201:      &      'IS NOT SPECIFIED. SET TO 'NBANK' ! ' , NBANK
1202:      call CNTERR( 'MESSAGE' )
1203:      IMPMAX = NBANK
1204:      end if
1205:
1206:      NIMPT = 0
1207:      NDIMPT = IMPMAX
1208: C
1209:      call LKEPV( H(1), 'IMSFL', LIMSFL, IMPMAX, 'I4', LAST, 0, JDEBG
1210:      &          )
1211:      call LKEPV( H(1), 'IMNFL', LIMNFL, NZONE+1, 'I4', LAST, 0,
1212:      &          JDEBG )
1213:      call LKEPV( H(1), 'IMZFL', LIMZFL, IMPMAX, 'I4', LAST, 0, JDEBG
1214:      &          )
1215: C
1216:      call LKEPV( H(1), 'IMSNX', LIMSNX, IMPMAX, 'I4', LAST, 0, JDEBG
1217:      &          )
1218:      call LKEPV( H(1), 'IMNNX', LIMNNX, NZONE+1, 'I4', LAST, 0,
1219:      &          JDEBG )
1220:      call LKEPV( H(1), 'IMZNX', LIMZNX, IMPMAX, 'I4', LAST, 0, JDEBG
1221:      &          )
1222:      IMDEFR = 0
1223: C
1224:      if ( JLATT.ne.0 ) then
1225:      call LKEPV( H(1), 'IMSLT', LIMSLT, IMPMAX, 'I4', LAST, 0,
1226:      &          JDEBG )
1227:      call LKEPV( H(1), 'IMSLT', LIMNLT, NLBZ+1, 'I4', LAST, 0,
1228:      &          JDEBG )
1229:      call LKEPV( H(1), 'IMZLT', LIMZLT, IMPMAX, 'I4', LAST, 0,
1230:      &          JDEBG )
1231:      end if
1232:      call LKEPV( H(1), 'IMDED', LIMDED, IMPMAX, 'I4', LAST, 0, JDEBG
1233:      &          )
1234:      end if
1235: C

```

```

1236:      call GTLAST( LLS1 )
1237:      call LGTLST( LLL1 )
1238: C
1239:      write(IPR,'(/1X,A,I12,A,I12)')
1240:      &      ' MEMORY FOR STACK DATA (WORDS) : TASK SHARED ', LLS1
1241:      &      - LLS0, ' TASK LOCAL ', LLL1 - LLL0
1242:      write(IPR,'(1X,A,I12,A,I12,A)')
1243:      &      ' MEMORY POSITION OF STACK DATA: FROM A(', LLS0,
1244:      &      ') TO A(', LLS1 - 1, ') '
1245:      write(IPR,'(1X,A,I12,A,I12,A)')
1246:      &      ' MEMORY POSITION OF STACK DATA: FROM H(', LLL0,
1247:      &      ') TO H(', LLL1 - 1, ') '
1248:      call MEMUSE( 'EVENT STACK', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
1249:      &          LIMITC )
1250: C
1251: C =====
1252: C      PARTICLE BANK
1253: C =====
1254: C
1255:      call GTLAST( LLS0 )
1256:      call LGTLST( LLL0 )
1257: C
1258: C-----
1259: C ..... /XBANK/ .....
1260: C-----
1261: C
1262:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1263:      INBANK = NBANK + IMPMAX
1264:      else
1265:      INBANK = NBANK
1266:      end if
1267: C
1268:      call LKEPV( H(1), 'XXX', LXXX, INBANK, 'R8', LAST, 0D0, JDEBG )
1269:      call LKEPV( H(1), 'YYY', LYYY, INBANK, 'R8', LAST, 0D0, JDEBG )
1270:      call LKEPV( H(1), 'ZZZ', LZZZ, INBANK, 'R8', LAST, 0D0, JDEBG )
1271:      call LKEPV( H(1), 'AAA', LAAA, INBANK, 'R8', LAST, 0D0, JDEBG )
1272:      call LKEPV( H(1), 'BBB', LBBB, INBANK, 'R8', LAST, 0D0, JDEBG )
1273:      call LKEPV( H(1), 'CCC', LCCC, INBANK, 'R8', LAST, 0D0, JDEBG )
1274:      call LKEPV( H(1), 'WWW', LWWW, INBANK, 'R4', LAST, 0E0, JDEBG )
1275:      call LKEPV( H(1), 'IZZ', LIZZ, INBANK, 'I4', LAST, 0, JDEBG )
1276:      call LKEPV( H(1), 'IGG', LIGG, INBANK, 'I4', LAST, 0, JDEBG )
1277:      call LKEPV( H(1), 'TTT', LTTT, INBANK, 'R8', LAST, 0D0, JDEBG )
1278:      if ( JTIME.ne.0 ) then
1279:      call LKEPV( H(1), 'ITT', LITT, INBANK, 'I4', LAST, 0, JDEBG )
1280:      end if
1281:      call LKEPV( H(1), 'KLSF', LKLSF, INBANK, 'I4', LAST, 0, JDEBG )
1282: C
1283:      if ( JIMPT.ne.0 ) then
1284:      call LKEPV( H(1), 'XIM', LXIM, NBANK, 'R4', LAST, 0E0, JDEBG )
1285:      end if
1286:
1287:      if ( JEIGN.ne.0 ) then
1288:      if ( NFBANK.eq.0 ) then
1289:      NFBANK = NHIST
1290:      NFBANK0 = NFBANK
1291:      write(IMG,*)
1292:      &      '!!! Fission bank length (NFBANK) is not specified. Set to ',
1293:      &      NFBANK
1294:      call CNTERR( 'WARNING' )
1295:      else if ( NFBANK.lt.NHIST ) then
1296:      write(IMG,*) '!!! Fission bank length (NFBANK=' , NFBANK,
1297:      &      ') is smaller', ' than batch size (NHIST=' , NHIST,
1298:      &      ') '
1299:      call CNTERR( 'WARNING' )
1300:      end if

```

src/gmvp/intro2.f

```

1301: C
1302:     if ( NFBNK0.eq.0 ) NFBNK0 = NFBANK
1303:
1304:     call LKEPV( H(1), 'XXXF', LXXXF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1305: &
1306:     call LKEPV( H(1), 'YYYF', LYYYF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1307: &
1308:     call LKEPV( H(1), 'ZZZF', LZZZF, NFBNK0, 'R8', LAST, 0D0, JDEBG
1309: &
1310:     call LKEPV( H(1), 'IZZF', LIZZF, NFBNK0, 'I4', LAST, 0, JDEBG )
1311:     call LKEPV( H(1), 'KLSFF', LKLSFF, NFBNK0, 'I4', LAST, 0E0,
1312: & JDEBG )
1313: C
1314:     if ( JLATT.ne.0 ) then
1315:       call LKEPV( H(1), 'LEVLF', LLEVLF, NFBNK0, 'I4', LAST, 0,
1316: & JDEBG )
1317:       call LKEPV( H(1), 'LZZF', LLZZF, NFBNK0*NEST, 'I4', LAST, 0,
1318: & JDEBG )
1319:       call LKEPV( H(1), 'LPOSF', LLPOSF, NFBNK0*NEST, 'I4', LAST,
1320: & 0, JDEBG )
1321:       if ( JHLAT.ne.0 ) then
1322:         call LKEPV( H(1), 'LCRSF', LLCRSF, NFBNK0*NEST, 'I4',
1323: & LAST, 0, JDEBG )
1324:       end if
1325:     end if
1326: C
1327:     if ( JTLLT.ne.0 ) then
1328:       call LKEPV( H(1), 'IBSPF', LIBSPF, NFBNK0*NEST, 'I4', LAST,
1329: & 0, JDEBG )
1330:     end if
1331: C
1332:     ... keep IBRGF anytime.
1333:     call LKEPV( H(1), 'IBRGF', LIBRGF, NFBNK0, 'I4', LAST, 0, JDEBG
1334: &
1335: C
1336: C
1337: C     ... IFISB(NHIST) : pointer to fission bank selected for
1338: C     source of a batch.
1339: C     Sources are selected at the beginning of each batch.
1340: C     This is to manage sub-batches in eigenvalue calculation
1341:     call LKEPV( H(1), 'IFISB', LIFISB, NHIST, 'I4', LAST, 0, JDEBG
1342: &
1343: C
1344: C     ... when NHSUB < NHIST, keep memory to copy fission bank
1345: C
1346:     if ( NHSUB.lt.NHIST ) then
1347:       call LKEPV( H(1), 'XXXF2', LXXXF2, NHIST, 'R8', LAST, 0D0,
1348: & JDEBG )
1349:       call LKEPV( H(1), 'YYYF2', LYYYF2, NHIST, 'R8', LAST, 0D0,
1350: & JDEBG )
1351:       call LKEPV( H(1), 'ZZZF2', LZZZF2, NHIST, 'R8', LAST, 0D0,
1352: & JDEBG )
1353:       call LKEPV( H(1), 'IZZF2', LIZZF2, NHIST, 'I4', LAST, 0,
1354: & JDEBG )
1355:       call LKEPV( H(1), 'KLSFF2', LKLSFF2, NHIST, 'I4', LAST, 0,
1356: & JDEBG )
1357:       if ( JLATT.ne.0 ) then
1358:         call LKEPV( H(1), 'LEVLF2', LLEVLF2, NHIST, 'I4', LAST,
1359: & 0, JDEBG )
1360:         call LKEPV( H(1), 'LZZF2', LLZZF2, NHIST*NEST, 'I4',
1361: & LAST, 0, JDEBG )
1362:         call LKEPV( H(1), 'LPOSF2', LLPOSF2, NHIST*NEST, 'I4',
1363: & LAST, 0, JDEBG )
1364:         if ( JHLAT.ne.0 ) then
1365:           call LKEPV( H(1), 'LCRSF2', LLCRSF2, NHIST*NEST, 'I4',

```

```

1366: & LAST, 0, JDEBG )
1367:       end if
1368:     end if
1369:     call LKEPV( H(1), 'IBRGF2', LIBRGF2, NHIST, 'I4', LAST, 0,
1370: & JDEBG )
1371:     if ( JTLLT.ne.0 ) then
1372:       call LKEPV( H(1), 'IBSPF2', LIBSPF2, NHIST*NEST, 'I4',
1373: & LAST, 0, JDEBG )
1374:     end if
1375:   end if
1376: C
1377:   end if
1378: C
1379:   if ( JEIGN.eq.0.and.NSKIP.ne.0 ) then
1380:     call CNTERR( 'WARNING' )
1381:     write(IMG,*)
1382: & '!!! "NSKIP" is not zero in fixed source problem',
1383: & ' NSKIP is set to 0.'
1384:     NSKIP = 0
1385:   end if
1386: C-----
1387: C.....  BANK DATA FOR LATTICE GEOMETRY .....
1388: C-----
1389:   if ( JLATT.ne.0 ) then
1390:     call LKEPV( H(1), 'LEVL', LLEVL, INBANK, 'I4', LAST, 0, JDEBG )
1391:     call LKEPV( H(1), 'LZZ', LLZZ, NBANK*NEST, 'I4', LAST, 0, JDEBG
1392: &
1393:     call LKEPV( H(1), 'LPOS', LLPOS, NBANK*NEST, 'I4', LAST, 0,
1394: & JDEBG )
1395:     if ( JHLAT.ne.0 ) then
1396:       call LKEPV( H(1), 'LCRS', LLCRS, NBANK*NEST, 'I4', LAST, 0,
1397: & JDEBG )
1398:     end if
1399: C
1400:   if ( NPDET.gt.0 ) then
1401:     call LKEPV( H(1), 'LZZI', LLZZI, IMPMAX*NEST, 'I4', LAST, 0,
1402: & JDEBG )
1403:     call LKEPV( H(1), 'LPOSI', LLPOSI, IMPMAX*NEST, 'I4', LAST,
1404: & 0, JDEBG )
1405:     if ( JHLAT.ne.0 ) then
1406:       call LKEPV( H(1), 'LCRSI', LLCRSI, IMPMAX*NEST, 'I4',
1407: & LAST, 0, JDEBG )
1408:     end if
1409:   end if
1410:   end if
1411: C
1412:   if ( JTLLT.ne.0 ) then
1413:     N1 = NEST + 1
1414:     call LKEPV( H(1), 'IBSPC', LIBSPC, NBANK*N1, 'I4', LAST, 0,
1415: & JDEBG )
1416:   end if
1417: C
1418:   ... always keep IBREG (from Jan 2000)
1419:   for convenience to new features.
1420: C
1421:   call LKEPV( H(1), 'IBREG', LIBREG, NBANK, 'I4', LAST, 0, JDEBG )
1422: C
1423: C-----
1424: C.....  optional bank data .....
1425: C-----
1426: C
1427: C     ... setting of custom BANK data ...
1428: C     K[DI][BNK|FBK](11:16) is currently reserved for users.
1429: C
1430:   call ZZBANK( KDBNK, MDBNK, KIBNK, MIBNK, KDFBK, MDFBK, KIFBK,

```

src/gmvp/intro2.f

```

1431:      &          MIFBK )
1432: C
1433: C..... floating point optional bank data .....
1434: C
1435: C ... save particle weight on birth
1436:       if ( JRWVR.ne.0 ) then
1437:           KDBNK(1) = 1
1438:       end if
1439: C
1440: C ... save distance to lattice frames (NEST+1 banks)
1441:       if ( JFISX.ne.0 ) then
1442:           KDBNK(4) = NEST + 1
1443:           KDBNK(6) = 6*NEST
1444:       end if
1445: C
1446: C ... save position of STG cell (3*NEST banks)
1447:       if ( JSTGR.ne.0 ) then
1448:           KDBNK(5) = 3*NEST
1449:       end if
1450: C
1451: C ... for surface tallies (distance and angle)
1452:       if ( JTSRF.ne.0 ) then
1453:           KDBNK(7) = 2*NTSRF
1454:       end if
1455: C
1456:       NNND = 0
1457:       do 120 K = 1, MDBNK
1458:           if ( KDBNK(K).ne.0 ) then
1459:               NNND1 = NNND
1460:               NNND = NNND + KDBNK(K)
1461:               KDBNK(K) = NNND1 + 1
1462:           end if
1463:       120 continue
1464:       KDBNK(0) = NNND
1465: C
1466: C
1467:       call LKEPV( H(1), 'DBNK', LDBNK, INBANK*NNND, 'R8', LAST, OD0,
1468:       &          JDEBG )
1469: C
1470: C..... integer optional bank data .....
1471: C
1472: C
1473: C ... save particle generation number
1474: C (another case of non zero KIBNK(4) is GENERATION tally.
1475: C see TALINP routine.)
1476: C
1477:       if ( MXPGEN.gt.0 ) then
1478:           KIBNK(4) = 1
1479:       end if
1480: C
1481:       if ( JFISX.ne.0 ) then
1482: C ... save flight count in a path
1483:           KIBNK(5) = 1
1484: C ... lattice level giving the smallest distance until the level
1485:           KIBNK(6) = NEST
1486: C ... required NND type
1487:           KIBNK(7) = 1
1488:       end if
1489: C
1490: C ... marker region passing flag
1491: C
1492:       if ( KIBNK(9).gt.0 ) then
1493:           KIBNK(9) = NMKREG
1494:       end if
1495: C

```

```

1496: C ... for surface tallies (flag and angle bin)
1497:       if ( JTSRF.ne.0 ) then
1498:           KIBNK(10) = 2*NTSRF
1499: C ... save flight count in a path
1500:           KIBNK(5) = 1
1501:       end if
1502: C
1503:       NNNI = 0
1504:       do 130 K = 1, MIBNK
1505:           if ( KIBNK(K).ne.0 ) then
1506:               NNNI1 = NNNI
1507:               NNNI = NNNI + KIBNK(K)
1508:               KIBNK(K) = NNNI1 + 1
1509:           end if
1510:       130 continue
1511:       KIBNK(0) = NNNI
1512:       call LKEPV( H(1), 'IBNK', LIBNK, INBANK*NNNI, 'I4', LAST, 0, JDEBG
1513:       &          )
1514: C
1515: C-----
1516: C ... optional fission bank data ...
1517: C-----
1518: C
1519:       if ( JEIGN.ne.0 ) then
1520: C
1521: C ... save distance to lattice frames (NEST+1 banks)
1522: C
1523: C ... no need to save distances & directions in each lattice level
1524: C they are calculated newly
1525: C if ( JFISX.ne.0 ) then
1526: C     KDFBK(4) = NEST + 1
1527: C     CCCC KDFBK(6) = 6*NEST
1528: C     end if
1529: C
1530: C ... save position of STG cell (3*NEST banks)
1531: C
1532:       if ( JSTGR.ne.0 ) then
1533:           KDFBK(5) = 3*NEST
1534:       end if
1535: C
1536:       NNND1 = 0
1537:       do 140 K = 1, MDFBK
1538:           if ( KDFBK(K).ne.0 ) then
1539:               NNND1 = NNND1
1540:               NNND1 = NNND1 + KDFBK(K)
1541:               KDFBK(K) = NNND1 + 1
1542:           end if
1543:       140 continue
1544:       KDFBK(0) = NNND1
1545:       call LKEPV( H(1), 'DFBK', LDFBK, NFBNK0*NNND1, 'R8', LAST, OD0,
1546:       &          JDEBG )
1547: C
1548: C ... lattice level giving the smallest distance until the level
1549: C if ( JFISX.ne.0 ) then
1550: C     KIFBK(6) = 1
1551: C     end if
1552: C
1553:       NNNIF = 0
1554:       do 150 K = 1, MIFBK
1555:           if ( KIFBK(K).ne.0 ) then
1556:               NNNI1 = NNNIF
1557:               NNNIF = NNNIF + KIFBK(K)
1558:               KIFBK(K) = NNNI1 + 1
1559:           end if
1560:       150 continue

```


src/gmvp/intro2.f

```

1561:      KIFBK(0)      = NNNIF
1562:      call LKEPV( H(1), 'IFBK', LIFBK, NFBK0*NNNIF, 'I4', LAST, 0,
1563:      &          JDEBG )
1564: C
1565: C      if ( MDFBK2.ne.MDFBK .or. MIFBK.ne.MIFBK2 ) then
1566:      write(IMG,*) 'XXX MDFBK2.ne.MDFBK .or. MIFBK.ne.MIFBK2!!'
1567:      write(IMG,*) ' Check parameter definition in header file.'
1568:      stop 666
1569:      end if
1570:
1571: C      if ( NHSUB.lt.NHIST ) then
1572:      do 160 K = 1, MDFBK
1573:      KDFBK2(K) = KDFBK(K)
1574:      160 continue
1575:      KDFBK2(0) = KDFBK(0)
1576:      call LKEPV( H(1), 'DFBK2', LDFBK2, NHIST*KDFBK2(0), 'R8',
1577:      &          LAST, 0D0, JDEBG )
1578:      do 170 K = 1, MIFBK
1579:      KIFBK2(K) = KIFBK(K)
1580:      170 continue
1581:      KIFBK2(0) = KIFBK(0)
1582:      call LKEPV( H(1), 'IFBK2', LIFBK2, NHIST*KIFBK2(0), 'I4',
1583:      &          LAST, 0, JDEBG )
1584:      end if
1585:      end if
1586:
1587:
1588: C-----
1589: C..... BANK DATA FOR IMAGINARY PARTICLE FOR NEXT EVENT ESTIMATOR ...
1590: C-----
1591:      if ( JPTDT.ne.0.and.NPDET.gt.0 ) then
1592:      call KEEP2( 'XXI', LXXX, NBANK, 'R8', LXXXI, JDEBG )
1593:      call KEEP2( 'YYI', LYYY, NBANK, 'R8', LYYYI, JDEBG )
1594:      call KEEP2( 'ZZI', LZZZ, NBANK, 'R8', LZZZI, JDEBG )
1595:      call KEEP2( 'AAI', LAAA, NBANK, 'R8', LAAAI, JDEBG )
1596:      call KEEP2( 'BBI', LBBB, NBANK, 'R8', LBBBI, JDEBG )
1597:      call KEEP2( 'CCI', LCCC, NBANK, 'R8', LCCCI, JDEBG )
1598:      call KEEP2( 'WWI', LWWW, NBANK, 'R4', LWWWI, JDEBG )
1599:      call KEEP2( 'IZI', LIZZ, NBANK, 'I4', LIZZI, JDEBG )
1600:      call KEEP2( 'IGI', LIGG, NBANK, 'I4', LIGGI, JDEBG )
1601:      if ( JTIME.ne.0 ) then
1602:      call KEEP2( 'ITTI', LITT, NBANK, 'I4', LITTI, JDEBG )
1603:      end if
1604:      call KEEP2( 'TTTI', LTTT, NBANK, 'R8', LTTTI, JDEBG )
1605:      call KEEP2( 'KLSFI', LKLSF, NBANK, 'I4', LKLSFI, JDEBG )
1606:      call KEEP2( 'DBNKI', LDBNK, NBANK*NNND, 'R8', LDBNKI, JDEBG )
1607:      call KEEP2( 'IBNKI', LIBNK, NBANK*NNNI, 'I4', LIBNKI, JDEBG )
1608:      if ( JLATT.ne.0 ) then
1609:      call KEEP2( 'LEVLI', LLEVL, NBANK, 'I4', LLEVLI, JDEBG )
1610:      end if
1611: C
1612:      call LKEPV( H(1), 'KDETP', LKDETP, IMPMAX, 'I4', LAST, 0, JDEBG
1613:      &          )
1614:      call LKEPV( H(1), 'OPTI', LOPTI, IMPMAX, 'R8', LAST, 0.0D0,
1615:      &          JDEBG )
1616:      call LKEPV( H(1), 'PATH', LPATH, IMPMAX, 'R8', LAST, 0.0D0,
1617:      &          JDEBG )
1618:      end if
1619: C
1620:      call GTLAST( LLS1 )
1621:      call LGTLST( LLL1 )
1622: C
1623:      write(IPR,'(1X,A,I12,A,I12)')
1624:      &          ' MEMORY FOR BANK DATA (WORDS) : TASK SHARED ', LLS1
1625:      &          - LLS0, ' TASK LOCAL ', LLL1 - LLL0

```

```

1626:      write(IPR,'(1X,A,I12,A,I12,A)')
1627:      &          ' MEMORY POSITION OF BANK DATA: FROM A(' , LLS0, ' ) TO A(' ,
1628:      &          LLS1 - 1, ' )'
1629:      write(IPR,'(1X,A,I12,A,I12,A)')
1630:      &          ' MEMORY POSITION OF BANK DATA: FROM H(' , LLL0, ' ) TO H(' ,
1631:      &          LLL1 - 1, ' )'
1632:      call MEMUSE( 'PARTICLE BANK', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
1633:      &          LIMITC )
1634: C
1635: C-----
1636: C      .... INPUT RESTART FILE & DETERMINE SIZE OF KEFF-TALLY ARRAY
1637: C-----
1638: C
1639: C      if ( JREST.ne.0 ) then
1640: C      call RESTIO( NTHIST, H(LWSUM), H(LWLEK), IRAND, NBATCH, NFISB,
1641: C      &          NGROUP, NRESP, NREG, NTREG, NMEMO, NZONE, NPART, NLATT,
1642: C      &          NEST, NPDET, NETALY )
1643: C      else
1644: C      NTHIST = 0
1645: C      NBATCH = 0
1646: C      end if
1647: C
1648: C      NPART > 0 : IS ABSOLUTE HISTORY NUMBER.
1649: C      NPART < 0 : RUN NTHIST + ABS(NPART) HISTORIES.
1650: C
1651:      if ( NPART.gt.0 ) then
1652:      if ( JEIGN.ne.0 ) then
1653:      if ( NPART.le.NTHIST ) then
1654:      NLENG = NBATCH
1655:      else
1656:      NLENG = NBATCH + (NPART-NTHIST-1) /NHIST + 1
1657:      end if
1658:      end if
1659:      else if ( NPART.lt.0 ) then
1660:      if ( JEIGN.ne.0 ) NLENG = NBATCH + (ABS(NPART)-1) /NHIST + 1
1661:      NPART = NTHIST + ABS(NPART)
1662:      else
1663:      write(IMG,')(/A,A/)')
1664:      &          'XXX TOTAL NUMBER OF HISTORIES (NPART) IS NOT ',
1665:      &          'SPECIFIED OR INPUT AS ZERO.'
1666:      call CNTERR( 'FATAL' )
1667:      end if
1668: C
1669:      if ( JEIGN.ne.0.and.NLENG.lt.NSKIP+2 ) then
1670:      write(IMG,')(/1X,A,I4,A,A)')
1671:      &          '!!! Number of batches to be run ( , NLENG,
1672:      &          ' ) is too small ',
1673:      &          'to calculate standard deviations of some results.'
1674:      write(IMG,')(/1X,A/)') ' it should be greater than NSKIP+2.'
1675:      call CNTERR( 'WARNING' )
1676: C
1677:      if ( NSKIP.ne.0 ) then
1678:      NSKIP2 = NLENG - 2
1679:      if ( NSKIP2.ge.0 ) then
1680:      write(IMG,')(/1X,A,I3,A)')
1681:      &          '!!! Number of skipped batch for tally (NSKIP) is changed to ',
1682:      &          NSKIP2,
1683:      &          ' to enable calculation of standard deviation.'
1684:      call CNTERR( 'WARNING' )
1685:      NSKIP = NSKIP2
1686:      end if
1687:      end if
1688: C
1689:      end if
1690: C

```

src/gmvp/intro2.f

```

1691:      if ( JEIGN.ne.0 ) then
1692:        call LKEPV( H(1), 'XSOC', LXSOC, NLENG, 'R8', LAST, 0D0, JDEBG
1693:        &
1694:        call LKEPV( H(1), 'XSXV', LXSXV, NLENG*3, 'R8', LAST, 0D0,
1695:        & JDEBG )
1696:        call LKEPV( H(1), 'XKEF', LXKEF, NLENG*5, 'R8', LAST, 0D0,
1697:        & JDEBG )
1698:      end if
1699: C
1700: C === ETRV : special tally which requires "real variance" calculation.
1701: C
1702:      if ( METRV.gt.0 ) then
1703:        NBETRV = NLENG - NSKIP
1704:        if ( NBETRV.gt.0 ) then
1705:          call LKEPV( H(1), 'ETRV', LETRV, METRV*(NBETRV+1), 'R8',
1706:          & LAST, 0D0, JDEBG )
1707:        end if
1708:      end if
1709: C
1710: C
1711: C      if ( JREST.ne.0 ) then
1712: C
1713: C        call RESTR( NGROUP, NRESP, NREG, NTREG, NTHIST, H(LWSUM),
1714: C        & IRAND, NBATCH, NMEMO, NZONE, H(LKMEMO), H(LSFLTR),
1715: C        & H(LSFLCL), H(LSRETR), H(LSRECL), H(LNENTR), H(LWCNTR),
1716: C        & H(LXAVT), H(LAAVT), H(LCAVT), NFISB, NFBANK, NFBNK0, NLENG,
1717: C        & H(LXXXF), H(LYYYF), H(LZZZF), H(LIZZF), H(LXSOC),
1718: C        & H(LXSXV), H(LXKEF), NLATT, NEST, H(LLEVLF), H(LLZZF),
1719: C        & H(LLPOSF), H(LLCRSF), H(LIBRGF), H(LIBSPF), NDET,
1720: C        & H(LSFLPD), H(LSREPD), NETALY, H(LETALY), NLETAL )
1721: C
1722: C      end if
1723: C
1724: C
1725: C .....
1726: C
1727: C      call GTLAST( LLS1 )
1728: C      call LGTLST( LLL1 )
1729: C      write(IPR,'(1X,A,I12,A,I12)')
1730: C      & ' USED MEMORY EXCEPT WORKING AREA (WORDS): TASK SHARED '
1731: C      & LSIZ(LLS1) - 1, ' TASK LOCAL ', LSIZL(LLS1) - 1
1732: C      write(IPR,'(1X,A,I12,A,I12,A)')
1733: C      & ' LAST POSITION OF MEMORY EXCEPT WORKING AREA: A(' LLS1
1734: C      & - 1, ' ) H(' LLL1 - 1, ' )'
1735: C
1736: C -----
1737: C      .... /XWORK/ .....
1738: C -----
1739: C
1740: C ..... check NBNK2 if JUNDG is "ON" .....
1741: C
1742: C      if ( JUNDG.ne.0 ) then
1743: C        if ( NBNK2.eq.0 ) then
1744: C          NBNK2 = MIN(500,NBANK)
1745: C          write(IMG,'(1X,A,A,I7,A)')
1746: C          & '!!! Particle bank size for "underground" modules (NBNK2)',
1747: C          & ' is not specified. Set it to ', NBNK2, '.'
1748: C          call CNTERR( 'WARNING' )
1749: C        end if
1750: C      end if
1751: C
1752: C      call GTLAST( LLS0 )
1753: C      call LGTLST( LLL0 )
1754: C
1755: C      call KEEP( 'LXYZ', LLXYZ, 10, 'I4', LAST, JDEBG )

```

```

1756:      call KEEP( 'LPWRK', LLPWRK, MWVEC, 'I4', LAST, JDEBG )
1757:      call KEEP( 'LVSTK', LLVSTK, MVSTK, 'I4', LAST, JDEBG )
1758:      if ( LSTCK(0).lt.0 ) then
1759:        LLXYZ = 0
1760:        LLPWRK = 0
1761:        LLVSTK = 0
1762:      end if
1763: C
1764: C      call WRKARY( A, H, LIMIT, LIMITL, MAXSF, MWVEC, MVSTK, LLXYZ,
1765: C      & LLPWRK, LLVSTK, A(LLXYZ), A(LLPWRK), A(LLVSTK), MAXWK )
1766: C
1767: C -----
1768: C      .... for CADENZ .....
1769: C -----
1770: C
1771: C      call KEEPC( 'RNM', LRNM, NBINMX, 'C128', LAST, JDEBG )
1772: C      call KEEPC( 'RNM', LRNM, MXRGBN, 'C128', LAST, JDEBG )
1773: C
1774: C
1775: C /#IF PARA( CRAY SX* )
1776: C *      call GTLAST(LAST)
1777: C /#ELSE
1778: C      LAST = MAXWK
1779: C /#ENDIF
1780: C
1781: C      write(IPR,'(1X,A,I12,A)')
1782: C      & ' USED DYNAMIC MEMORY SIZE (TASK SHARED) = ', LSIZ(LAST)
1783: C      & - 1, '(WORD)'
1784: C      if ( LSIZ(LAST)-1.gt.LIMIT ) then
1785: C        write(IMG,*) ' XXX MEMORY OVER !! LIMIT = ', LIMIT, ' (WORD)'
1786: C        call PRSTOP( 1, 'MEMORY OVER.' )
1787: C        stop 999
1788: C      end if
1789: C
1790: C /#IF PARA( CRAY* SX* )
1791: C
1792: C      call lgtlst( llst )
1793: C ... local memory size is determined in wrkary routine : maxwk ...
1794: C
1795: C      write(IPR,'(1X,A,I12,A)')
1796: C      & ' TASK LOCAL DYNAMIC MEMORY TO BE USED = ', LSIZL(MAXWK)
1797: C      & - 1, '(WORD)'
1798: C      if ( LSIZL(MAXWK)-1.gt.LIMITL ) then
1799: C        write(IMG,*) ' XXX LOCAL MEMORY OVER !! LIMIT = ', LIMITL,
1800: C        & ' (WORD)'
1801: C        call PRSTOP( 1, 'MEMORY OVER.' )
1802: C        stop 999
1803: C      end if
1804: C      call MEMUSE( 'WORKING AREA', 0, MAXWK-LLL0, LIMIT, LIMITL, LIMITC
1805: C      & )
1806: C      call GTLAST( LLS1 )
1807: C      call MEMUSE( ' ', LSIZ(LLS1)-1, LSIZL(MAXWK)-1, LIMIT, LIMITL,
1808: C      & LIMITC )
1809: C /#ELSE
1810: C      call MEMUSE( 'WORKING AREA', LAST-LLS0, LAST-LLS0, LIMIT, LIMITL,
1811: C      & LIMITC )
1812: C      call MEMUSE( ' ', LSIZ(LAST)-1, LSIZL(LAST)-1, LIMIT, LIMITL,
1813: C      & LIMITC )
1814: C /#ENDIF
1815: C
1816: C -----
1817: C ..... CONVERSION OF CROSS SECTION IN CASE OF NTGX > NGROUP AND
1818: C NEUTRON-PHOTON MIXED PROBLEM
1819: C -----
1820: C

```

src/gmvp/intro2.f

```
1821:      if ( JIMAG.eq.0.and.JNEUT*JPHOT.ne.0 ) then
1822:      call XSECCN( IOW, JDDX, JFISS, JEIGN, JSTATX, JADJM, NGROUP,
1823:      &          NGP1, NGP2, NGPX, NGGX, NTGX, NDSPX, NDSGX, NDSTX,
1824:      &          NUSX, NSCTX, NCOEFX, NMEDX, A(LNGSX), A(LNNNX),
1825:      &          A(LSTOTX), A(LSSCX), A(LSNUFX), A(LSNAPX), A(LSGPX),
1826:      &          A(LSNFPX), A(LSNUSX), A(LSGPBX), A(LSGPBX), A(LSSCBX),
1827:      &          A(LSSCBX), A(LSDDBX), A(LSDDBX), A(LSANGX), A(LSANGX),
1828:      &          A(LSGPPX), A(LSSCPX), A(LSPORT), A(LS2DPX), A(LS2PPX),
1829:      &          A(LFKAI), A(LKKAI) )
1830:      end if
1831: C
1832: C-----
1833: C ..... INPUT OF RESTART FILE & PREPARATION FOR TALLY
1834: C-----
1835: C
1836:      if ( JREST.eq.1 ) then
1837:      if ( NPART0.gt.0.and.NTHIST.ge.NPART0 ) then
1838: Ccccccccc call CNTERR( 'FATAL' )
1839:      JNRUN = 1
1840:      else
1841:      JNRUN = 0
1842:      end if
1843:      else
1844:      NTHIST = 0
1845:      NBATCH = 0
1846: Cccccccc WSUM = 0.0
1847:      JNRUN = 0
1848:      end if
1849: Ccccc WLEK = 0.0
1850: C
1851: C
1852: 7140 format(/1X,'XXX(INTRO2) Cannot proceed input processing any more',
1853:      &      ' because of memory shortage.'/1X,' * Stop before ',A)
1854: 7160 format(/1X,'XXX(INTRO2) Cannot perform input processing ',
1855:      &      ' because of memory shortage.'/1X,' * Skip ',A)
1856: C
1857:      return
1858:      end
```

src/gmvp/intro.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine INTRO( TITLE, NTASK0,MLIMIT )
3: C=====
4: C
5: C PURPOSE:  DATA INPUT & PREPARATION FOR RANDOM WALK.
6: C CALLED IN: CENTER
7: C CALLS:   ARAYIN ASCTR4 CHECK CHKERR CKNUL0 CKVAL4 CNTERR CPUTM
8: C          DMREAD FREADS GEOMIN HEADER HVERSN INPLST INTRO2 KEEP
9: C          MEMALC NFINP OPTION PABLO PRNERR PRSIZE REGDAT RESET
10: C          RIUNIT SIZCHK SRCINP VALUE0 XSECIN
11: C=====
12:   include '../shared/INC/_ARRAY'
13:
14:   include '../shared/INC/_LISTOFF'
15: C
16:   include 'INC/_KPIDS'
17:   include 'INC/_KPSYMS'
18:   include 'INC/_NGPS'
19:
20:   include 'INC/_FLAGS'
21:   include '../shared/INC/_SIZES'
22:   include 'INC/_SIZES2'
23:   include 'INC/_XBANK'
24:   include 'INC/_STACK'
25:   include 'INC/_XWORK'
26:   include '../shared/INC/_PGEOM'
27:   include 'INC/_PXSEC'
28:   include '../shared/INC/_PVRED'
29:   include 'INC/_PSOUR'
30:   include '../shared/INC/_PTALY0'
31:   include 'INC/_PTALY'
32:   include '../shared/INC/_PTLSP'
33:   include '../shared/INC/_STALY'
34:
35:   include '../shared/INC/_LISTON'
36:
37:   include '../shared/INC/_WORDL'
38:   include '../shared/INC/_IOUNIT'
39:   include 'INC/_IOUNIT2'
40:
41: C/#IF PARA(PVM)
42:   include '../shared/INC/_PVMPARA'
43: C/#ELSEIF PARA(MPI)
44: *   include 'mpif.h'
45: C/#ENDIF
46:   include '../shared/INC/_TASKDT'
47:
48:   include '../shared/INC/_RAND'
49: C
50:   character TITLE(2)*72
51: C
52:   character VNM*6, VNAME*72, VSUBF*16
53: C
54: C ... dummy value for cutoff time
55: C
56:   parameter( TCUTDM = -1.0E30 )
57: C
58: C-----
59: C
60:   call CPUTM( T0 )
61: C
62: C ... check unopened scratch file
63: C
64:   call OPENS( 'GMVP' )
65: C

```

```

66: C ... initialize random number generator
67: C
68:   call RNINIT( 1 )
69: C
70: C ... DATA INPUT UNIT SETTING .....
71: C
72:   call RIUNIT( IOIN )
73: C
74: C .... Pass version description to HEADER routine...
75: C
76: CC   call VERSTR
77: C
78: C
79: C/#IF .NOT.PARA( PVM MPI )
80: C
81: C
82: C..... ECHOBACK PRINTING OF INPUT DATA .....
83: C
84:   call INPLST( IPR, IMMSG, INP, IOIN, NTASK0, MLIMIT )
85: C
86: C
87: C..... Send input data lines to sub task in PVM mode ...
88: C
89: C
90: C
91: C/#ELSE IF PARA( PVM MPI )
92: C
93:   call TOKEI( T01, 0 )
94:   NTASK = NTASK0
95: CC   if( ITASK0.lt. 0 ) then
96:   if ( IDTASK.eq.1 ) then
97: C
98:   call INPLST( IPR, IMMSG, INP, IOIN, NTASK0, MLIMIT )
99: C
100:   if ( NTASK0.gt.1 ) then
101:   call RWIND( IOIN )
102: C
103:   MSGNUM = 0
104:   IT0 = 1
105: C
106: 100   read(IOIN,'(A)',end =110) VNAME
107:   go to 120
108: 110   VNAME = '///END'
109: 120   MSGNUM = MSGNUM + 1
110:   call MVPCOM_BCAST_CH( IT0, MSGNUM, VNAME, LEN(VNAME), INFO )
111:   if ( VNAME.ne.'///END' ) go to 100
112:   end if
113: C
114:   else
115:   call RWIND( IOIN )
116:   MSGNUM = 0
117:   IT0 = 1
118: 130   MSGNUM = MSGNUM + 1
119:   VNAME = ' '
120:   call MVPCOM_BCAST_CH( IT0, MSGNUM, VNAME, LEN(VNAME), INFO )
121:   if ( VNAME.ne.'///END' ) then
122:   write(IOIN,'(A)') VNAME
123:   go to 130
124:   end if
125:   call RWIND( IOIN )
126:   call INPLST( IPR, IMMSG, IOIN, IOIN, NTASK0, MLIMIT )
127:   end if
128: C
129:   call RWIND( IOIN )
130: C

```

src/gmvp/intro.f

```

131:      call TOKEI( T02, 0 )
132:      write(IPR,*) 'TASK ', IDTASK, ': Elapsed ', T02 - T01,
133:      &          ' SEC FOR ', 'INPUT DATA PASSING.'
134: C
135: C/#ENDIF
136: C
137:      call HEADER( IPR, 'PROBLEM TITLE & OPTIONS' )
138: C
139: C
140: C-----
141: C ..... READ TITLE   ( 2 LINES )
142: C-----
143: C
144: C
145:      read(IOIN,'(A)') (TITLE(I),I=1,2)
146:      write(IPR,7000) (TITLE(I),I=1,2)
147: 7000 format(1X,130('='//1X,A/1X,A/1X,130('='//))
148: C
149: C
150: C-----
151: C ..... READ OPTION PARAMETERS
152: C-----
153: C
154:      NTASK = 0
155: C
156:      call OPTION( MLIMIT )
157: C
158: C .... MEMORY ALLOCATION CHECK ....
159: C      LSTT : starting position index in dynamic memory array A
160: C            in common /ARRAY/ .
161: C
162:      call MEMALC( IPR, LSTT, LSTTL, IERR )
163: C
164:      write(IPR,'()')
165: C
166:      if ( NTASK.eq.0 ) NTASK = 1
167:      if ( NTASK0.ne.0 ) NTASK = NTASK0
168: C
169: C
170: C-----
171: C ..... DEFAULT VALUES or initial setting of variables mainly in
172: C            common /SIZES/
173: C            (DEFAULT VALUES OF DEPS & DINF WILL BE GIVEN IN 'GEOMIN' )
174: C-----
175:      NMEMOP = 2
176:      DEPS = 0.0D0
177:      DINF = 0.0D0
178:      if ( JTIME.ne.0 ) then
179:          TCUT = TCUTDM
180:      else
181:          TCUT = 0.0
182:      end if
183: C
184:      NGROUP = 0
185:      do 140 I = 1, KPLIM
186:          NGP(I) = 0
187:          KNGP(I) = 0
188:          KENGP(I) = 0
189: 140 continue
190:      KNGP(KPLIM+1) = 0
191:      KENGP(KPLIM+1) = 0
192: C
193:      NZONE = 0
194:      NINPZ = 0
195:      NSURF = 0

```

```

196:      NSDA = 0
197:      NZDA = 0
198:      NMEMO = 0
199:      NMAT = 0
200:      NREG = 0
201:      NTIME = 0
202:      NSOUR = 0
203:      NMEMS = 0
204:      NRESP = 0
205:      NRESP2 = 0
206:      NRSKIP = 0
207:      NSKIP = 0
208:      NGP1 = 0
209:      NGP2 = 0
210:      NSPACE = 0
211:      NBODY = 0
212:      NTREG = 0
213:      NSTALY = 0
214:      NETALY = 0
215:      NDTALY = 0
216:      NLETAL = 0
217:      NLDTAL = 0
218:      NBINMX = 0
219:      NMKREG = 0
220: C
221:      NTSRF = 0
222:      MTSRF = 0
223:      NANGLE = 0
224: C
225:      IRAND = 0
226:      IRNSU = 0
227:      IRNSM = 0
228:      IRNSL = 0
229:      NBPINT = 1
230:      NRSINT = 0
231:      NTMINT = 10
232: C
233:      NHIST = 0
234:      NHSUB = 0
235:      NPART = 0
236:      NBANK = 0
237:      NBNK2 = 0
238:      IMPMAX = 0
239: C
240:      NFBANK = 0
241:      NFBNK0 = 0
242: C
243:      NPDET = 0
244: C
245:      MXPGEN = 0
246: C
247:      NMXLG = 10
248:      NEITER = 500
249: C
250:      WLLIM = 1.0E-10
251: C
252:      ELOOP = 0.2
253: C
254:      TCPU = 0.0
255: C
256:      NPKIND = 0
257: CC      if ( JNEUT.ne.0 ) NPKIND = NPKIND + 1
258: CC      if ( JPHOT.ne.0 ) NPKIND = NPKIND + 1
259:      KKPID = 0
260:      do 150 IK = 1, KPLIM

```

src/gmvp/intro.f

```

261:      if ( JKPAR(IK).ne.0 ) then
262:        KKPID = IK
263:        NPKIND = NPKIND + 1
264:      end if
265:      150 continue
266: C
267: C    ... KKPID is particle species ID for single particle problem.
268: C    if ( NPKIND.gt.1 ) KKPID = 0
269: C -----
270: C ..... CLEAR SOME POINTERS .....
271: C -----
272: C
273: C    LXIMP = 0
274: C    LWKIL = 0
275: C    LWSRV = 0
276: C    LWTIME = 0
277: C    LPSXYZ = 0
278: C    LPSALP = 0
279: C    LSRCSF = 0
280: C    LSOUR = 0
281: C    LKSOUR = 0
282: C    LIDSRF = 0
283: C    LPENRG = 0
284: C    LRESP = 0
285: C    LFKAI = 0
286: C    LIFISM = 0
287: C    LWGTF = 0
288: C    LWGTG = 0
289: C    LISZON = 0
290: C    LENGYB = 0
291: C    LENGPB = 0
292: C    LTIMEB = 0
293: C    LVOL = 0
294: C    LRVOL = 0
295: C    LTRVOL = 0
296: C
297: C    LIDMAT = 0
298: C
299: C    LVEL = 0
300: C    LIDTAL = 0
301: C    LIETAL = 0
302: C    LKDTAL = 0
303: C    LKETAL = 0
304: C    LJTEVE = 0
305: C    LLTEVE = 0
306: C    LKTEVE = 0
307: C    NTEVE = 0
308: C
309: C    LNSANG = 0
310: C    LSANG = 0
311: C    LPSANG = 0
312: C    LSAXIS = 0
313: C
314: C    LXPDET = 0
315: C    LJPUSD = 0
316: C    LJSPDT = 0
317: C
318: C    LIDSRF = 0
319: C    LITSRF = 0
320: C    LKTSRF = 0
321: C    LKTSFJ = 0
322: C    LANGLB = 0
323: C
324: C    ... initialize extra bank parameter index
325: C    do 160 K = 0, MDBNK

```

```

326:      KDBNK(K) = 0
327:      160 continue
328:      do 170 K = 0, MIBNK
329:        KIBNK(K) = 0
330:      170 continue
331: C
332: C    ... initialize extra fission bank parameter index
333: C    if ( JEIGN.ne.0 ) then
334: C      do 180 K = 1, MDFBK
335: C        KDFBK(K) = 0
336: C      180 continue
337: C      do 190 K = 1, MIFBK
338: C        KIFBK(K) = 0
339: C      190 continue
340: C    end if
341: C
342: C    ... temporary pointer
343: C
344: C    LKR = 0
345: C
346: C    ... local flags ...
347: C    ICHK = 0
348: C    ICK = 0
349: C
350: C    ..... JNP = 1 : MEANS NEUTRON - PHOTON JOINT PROBLEM
351: C
352: C    JNP = JNEUT*JPHOT
353: C    JNP = 0
354: C    if ( JKPAR(KPNEUT).ne.0.and.JKPAR(KPPHOT).ne.0 ) then
355: C      JNP = 1
356: C    end if
357: C
358: C
359: C    ..... checker of duplicate blocks
360: C
361: C    KKKGEO = 0
362: C    KKKXSC = 0
363: C    KKKSRC = 0
364: C    KKKTAL = 0
365: C
366: C -----
367: C ..... READ OTHER DATA .....
368: C -----
369: C
370: C
371: C    .... TO SET NULL ADDRESS ASSIGNMENT CHCKER ....
372: C
373: C    .... INITIAL VALUE OF 'LAST' IS SET IN THIS ROUTINE ...
374: C
375: C    call CKNUL0( A, LAST, LSTT )
376: C
377: C    .... NLEN : NUMBER OF NON BLANK CHARACTERS READ AS VNAME ...
378: C
379: C    200 call FREADS( VNAME, NLEN, IEND )
380: C
381: C    if ( IEND.ne.0 ) go to 210
382: C    ICALL = 0
383: C
384: C
385: C -----
386: C ..... GEOMETRY .....
387: C -----
388: C
389: C
390: C    if ( NLEN.ge.5.and.VNAME(1:5).eq.'$GEOM' ) then

```

src/gmvp/intro.f

```

391:
392:   if ( KKKGEO.ne.0 ) then
393:     write(IMG,*) 'XXX $GEOMETRY block is found more than once.'
394:     call CNTERR( 'FATAL' )
395:   end if
396:   KKKGEO = 1
397:
398:   call GTLAST( LLS0 )
399:   call LGTLST( LLL0 )
400:
401:   call CPUTM( TT0 )
402:
403:   call GEOMIN( ICALL, A, LIMIT, CHA, LIMITC, LAST, MAXSF, JDEBG,
404: &             JHLAT, JLATT, JREFL, JSIMP, JTLLT, JFISX, JSTGR )
405:
406:   call CPUTM( TT1 )
407:
408:   write(IPR, '(2x,a,e12.3,a/)' )
409: &   ' === CPU TIME FOR GEOMETRY PROCESSING ', TT1 - TT0,
410: &   ' SEC'
411: C
412:   call GTLAST( LLS1 )
413:   call LGTLST( LLL1 )
414:   call MEMUSE( 'GEOMETRY DATA', LLS1-LLS0, LLL1-LLL0, LIMIT,
415: &             LIMITL, LIMITC )
416: C
417:   go to 200
418: end if
419: C
420: C
421: C-----
422: C ..... CROSS SECTION .....
423: C-----
424: C
425: C $XSEC : cross section & material composition input as
426: C MORSE-DD (old type)
427: C
428: C $CROSS SECTION : cross section & material composition input as
429: C newly defined as GMVP type. ( $CROS is effective )
430: C
431: C
432: C
433:   if ( NLEN.ge.5
434: &   .and.(VNAME(1:5).eq.'$XSEC'.or.VNAME(1:5).eq.'$CROS') ) then
435: C
436:     if ( KKKXSC.ne.0 ) then
437:       write(IMG,*)
438: &       'XXX $CROSS SECTION block is found more than once.'
439:       call CNTERR( 'FATAL' )
440:     end if
441:     KKKXSC = 1
442:
443:     call GTLAST( LLS0 )
444:     call LGTLST( LLL0 )
445: C
446: C =====
447: C   call HEADER( IPR, 'CROSS SECTION & MATERIAL COMPOSITION' )
448: C =====
449: C/#IF PARA( PVM MPI )
450:   if ( VNAME(1:5).eq.'$XSEC' ) then
451:     write(IMG,*) 'XXX "$XSEC" type of cross section block is ',
452: &     'not allowed in "PVM/MPI" or other multitasking mode!!'
453:     write(IMG,*) ' Please input with the new format - ',
454: &     '"$CROSS SECTION ... $END CROSS SECTION"'
455:     call PRSTOP( 1, '"$XSEC" format does not match PVM/MPI.' )

```

```

456:       stop 888
457:     end if
458: C/#ENDIF
459: C
460:     call CPUTM( TT0 )
461:     call TOKEI( T01, 0 )
462: C
463: C ... pre process GMVP type input to MORSE-DD type input to use in
464: C XSECIN routine. ( ioin --> iug1 )
465: C
466:     call GTLAST( LAST )
467:
468:     if ( VNAME(1:5).eq.'$CROS' ) then
469: C
470:       call XSGMVP( A, A, A, LAST, LIMIT, LFKAI, LIDMAT, NMAT,
471: &                 NGP1, NGP2 )
472: C
473:       IIII = IUG1
474: C/#IF PARA( PVM MPI )
475: C
476: C ... send/receive IDMAT(1:NMAT)
477: C
478:       IT0 = 1
479:       if ( NTASK.gt.1 ) then
480:         call MVPCOM_BCAST_I4( IT0, 1234, NMAT, 1, INFO )
481:         if ( IDTASK.gt.1 ) then
482:           call KEEP( 'IDMAT', LIDMAT, NMAT, 'I4', LAST, JDEBG )
483:         end if
484:         call MVPCOM_BCAST_I4( IT0, 1234, A(LIDMAT), NMAT, INFO )
485:       end if
486: C/#ENDIF
487:     else
488:       LIDMAT = 0
489:       IIII = IOIN
490:     end if
491:
492:     call RIUNIT( IIII )
493: C
494:     call GTLAST( LAST )
495: C
496: C ... perform cross section input on parent task only in PVM mode.
497: C
498: C/#IF PARA(PVM MPI)
499: cc   if( ITASK0.lt.0 ) then
500:     if ( IDTASK.eq.1 ) then
501: C/#ENDIF
502:
503:       call XSECIN( ICALL, A, A, LAST, LIMIT, IPR, IIII )
504:
505: C/#IF PARA(PVM MPI)
506:     else
507:       ICALL = 1
508:       call GTXSEC( A )
509:       call GTLAST( LAST )
510:     end if
511: C/#ENDIF
512: C
513:     call STLAST( 'XSECIN', LAST, LAST )
514: C
515:     NMAT = NMEDX
516: C
517: C ... set all IDMAT to material sequenatial # if not defined.
518: C
519:     if ( LIDMAT.eq.0 ) then
520:       call KEPV( A(1), 'IDMAT', LIDMAT, NMAT, 'I4', LAST, 0, JDEBG

```

src/gmvp/intro.f

```

521:      &
522:      call ATRANS( A(LRSGX), A(LIDMAT), NMEDX )
523:      end if
524: C
525:      call CPUTM( TT1 )
526:      call TOKEI( T02, 0 )
527:      write(IPR,'(/lx/a)')
528:      &      '==== TIMES FOR CROSS SECTION PROCESSING ====='
529:      write(IPR,'(2x,a,e12.4,a)') ' CPU ', TT1 - TT0, ' SEC'
530:      write(IPR,'(2x,a,e12.4,a)') ' ELAPSED ', T02 - T01, ' SEC'
531:      write(IPR,'()')
532: C
533: C ..... RESET INPUT I/O UNIT TO STANDARD ONE ...
534: C
535:      call RIUNIT( IOIN )
536: C
537:      call GTLAST( LLS1 )
538:      call LGTLST( LLL1 )
539:      call MEMUSE( 'X-SEC DATA', LLS1-LLS0, LLL1-LLL0, LIMIT, LIMITL,
540:      &      LIMITC )
541: C
542:      go to 200
543:      end if
544: C
545: C-----
546: C "REGION"-DEPENDENT DATA
547: C-----
548: C !REGION-NAME( VNAME(DATA) ) .... )
549: C OR VNAME( !REGION-NAME( DATA ) .... )
550: C OR VNAME( DATA .... )
551: C
552:      IPPP = INDEX(VNAME(:NLEN),'.') - 1
553:      if ( IPPP.lt.0 ) IPPP = NLEN
554:      if ( VNAME(1:1).eq.'!' .or. VNAME(1:IPPP).eq.'XIMP'
555:      &      .or. VNAME(1:IPPP).eq.'WKIL' .or. VNAME(1:IPPP).eq.'WSRV'
556:      &      .or. VNAME(1:IPPP).eq.'WGTF' .or. VNAME(1:IPPP).eq.'WGTG'
557:      &      .or. VNAME(1:IPPP).eq.'WGTP' .or. VNAME(1:IPPP).eq.'LRVOL'
558:      &      .or. VNAME(1:IPPP).eq.'TRVOL' .or. VNAME(1:IPPP).eq.'PSALP' )
559:      &      then
560: C
561: C      ... WGTG (for MVP) ---> WGTG (not elegant ... ^_^; )
562: C
563:      if ( VNAME(:IPPP).eq.'WGTG' ) VNAME(:NLEN) = 'WGTG'
564: C
565: C      ... check number of energy groups.
566: C
567:      if ( VNAME(1:IPPP).eq.'XIMP' .or. VNAME(1:IPPP).eq.'WKIL'
568:      &      .or. VNAME(1:IPPP).eq.'WSRV' ) then
569:      call NGPCHK( 'GMVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
570:      &      NGROUP )
571:      call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
572:      end if
573: C
574: C      .... TEMPORARY WORKING ARRAY ....
575: C
576:      NRRR0 = MAX(NREG,NTREG)
577:      if ( LKR.eq.0 ) then
578:      call KEEP( 'KR', LKR, NREG, 'I4', LAST, JDEBG )
579:      end if
580: C
581: C###<2007/03/14:PN3:
582:      NUCPN_DMY = 0
583: C##>
584:      call REGDAT( MSG, VNAME(:NLEN), VNM, VSUBF, A, A, LIMIT, LAST,
585:      &      JNEUT, JPHOT, JDEBG, NREG, NGP(1), JKPAR(1), KPSYM,

```

```

586:      &      KPLIM, KNGP(1), NGROUP, NGP1, NGP2, LXIMP, LWKIL,
587:      &      LWSRV, LWGTF, LWGTG, LRVOL, LTRVOL, LPSALP, JTLT,
588:      &      A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
589:      &      A(LISPNM), A(LISUSP), A(LKSUZN), A(LIRGSP), NEST,
590:      &      NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAMES, A(LITRNM),
591:      &      NTREG, A(LIPTRG), A(LLPTRG), A(LKR), NRRR0,
592:      C###<2007/03/14:PN3:
593:      &      NUCPN_DMY, NUCPN_DMY, NUCPN_DMY)
594: C##>
595:      ICALL = 1
596: C
597:      end if
598: C
599: C
600: C-----
601: C ..... SOURCE .....
602: C-----
603: C
604: C/#IF SOURCE(NEW)
605: C
606:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$SOUR' ) then
607: C
608: CCC      call CHKNGP( 'GMVP',JNEUT, NGP1, JPHOT, NGP2, NGROUP )
609:      call NGPCHK( 'GMVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
610:      &      NGROUP )
611: C
612:      if ( KKKSRC.ne.0 ) then
613:      write(MSG,*) 'XXX $SOURCE block is found more than once.'
614:      call CNTERR( 'FATAL' )
615:      end if
616:      KKKSRC = 1
617: C
618:      call GTLAST( LLS0 )
619:      call LGTLST( LLL0 )
620: C
621:      LSTART = LAST
622: C
623:      NUC0 = 0
624: C
625:      call SRCINP( A, H, CHA, LIMIT, LIMITL, LIMITC, LSRCS, NSRCS,
626:      &      LSOUR, LIDSRC, 'GMVP', JKPAR(1), KPLIM, JFISS, JEIGN,
627:      &      JTIME, JNEUT, JPHOT, JUNDG, JDEBG, MWVEC, MVSTK, MXREJ,
628:      &      NERR, A(LIDMAT), ' ', NUC0, NTGX, LKSOUR,
629:      &      LISZON, LPSPAC, LPENRG, LIFISM, LFKAI, LENGYB, LENGPB,
630:      &      IDUMMY )
631: C
632:      call GTLAST( LLS1 )
633:      call LGTLST( LLL1 )
634:      call MEMUSE( 'SOURCE DATA', LLS1-LLS0, LLL1-LLL0, LIMIT,
635:      &      LIMITL, LIMITC )
636: C
637:      go to 200
638:      end if
639: C
640: C/#ENDIF
641: C
642: C
643: C-----
644: C ..... TALLY .....
645: C-----
646: C
647: C
648:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$TALL' ) then
649: C
650:      call CHKNGP( 'GMVP', JNEUT, NGP1, JPHOT, NGP2, NGROUP )

```


src/gmvp/intro.f

```

651:
652:   if ( KKKTAL.ne.0 ) then
653:     write(IMG,*) 'XXX $TALLY block is found more than once.'
654:     call CNTERR( 'FATAL' )
655:   end if
656:   KKKTAL = 1
657:
658: C
659: C     ... INCST is not used in GMVP ...
660: C
661: C     NUCDUM = 1
662: C
663: c##<2007/03/14:PN3:
664:   NUCPN_DMY = 0
665: c##>
666:   call CPUTM( TT0 )
667:   call TALINP( 'GMVP', ICALL, LICRES, NICRES, LWTIME, A, A, A,
668: &     CHA, H, H, H, NGP(1), JKPAR(1), KPSYM, KNCF(1),
669: &     KENGP(1), KPLIM, JNEUT, JPHOT, JTIME, JFISS, JEIGN,
670: &     JRESP, JLLT, JTSRF, JPTDT, JDEBG, A(LIDSRC), LIDSRC,
671: &     NUCDUM, KDBNK, MDBNK, KIBNK, MLENK,
672: &     LKR, LIMIT, LXIMP, LWKIL, LWSRV, LWGTF, LWGTP,
673: &     LPSALP,
674: c##<2007/03/14:PN3:
675: &     NUCPN_DMY, NUCPN_DMY, NUCPN_DMY, NUCPN_DMY, NUCPN_DMY )
676: c##>
677:   call CPUTM( TT1 )
678:
679:   write(IPR, '(2x,a,e12.5,a)')
680: &     ' ==== CPU TIME FOR TALLY BLOCK PROCESSING ', TT1
681: &     - TT0, ' SEC'
682:   go to 200
683: end if
684: C
685: C-----
686: C ..... EXTRACT 'VARIABLE NAME' FROM VNAME .....
687: C-----
688: C
689: C     VNM = ' '
690: C     VSUBF = ' '
691: C     IPER = INDEX(VNAME, '.')
692: C     IKPID = 0
693: C
694: C     ..... VNAME WITH SUBFIELD .....
695: C
696:   if ( IPER.ne.0 ) then
697:     VNM(1:IPER-1) = VNAME(1:IPER-1)
698:     VSUBF = VNAME(IPER+1:NLEN)
699: C
700: C     IPR2 = INDEX('NP', VSUBF(1:1))
701: C     call KPSYMB( VSUBF(1:ICLEN2(VSUBF)), '>', IKPID, 0 )
702: C     if ( IKPID.eq.0 ) then
703: C       write(IMG, '(1x,a,a,a,a,a)') 'XXX Data name suffix( ',
704: &     VSUBF(1:ICLEN2(VSUBF)), ' ) of ',
705: &     VNAME(1:ICLEN2(VNAME)), ' is incorrect.'
706: C       write(IMG,*) ' No matching particle species.'
707: C       call CNTERR( 'FATAL' )
708: C       call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
709: C       if ( IRET.ne.0 ) go to 230
710: C       go to 200
711: C
712: C     ... supported particle species but not used in this problem
713: C
714: C     else if ( JKPAR(IKPID).eq.0 ) then
715: C       write(IMG,*) '!!! ', VNAME(1:ICLEN2(VNAME)),

```

```

716: &     ' has no meanings because particle species <',
717: &     VSUBF(1:ICLEN2(VSUBF)), '> is not treated',
718: &     ' in current problem.'
719: C     call CNTERR( 'WARNING' )
720: C     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
721: C     if ( IRET.ne.0 ) go to 230
722: C     go to 200
723: C   end if
724: C
725: C     ..... VNAME WITHOUT SUBFIELD .....
726: C
727: C   else
728: C     NLLL = MIN(NLEN, LEN(VNM))
729: C     VNM(1:NLLL) = VNAME(1:NLLL)
730: C   end if
731: C
732: C   if ( VSUBF.eq.'P'.and.JPHOT.eq.0 ) then
733: C     write(IMG, '(1x,a,a,a)') '!!! ', VNAME,
734: C &     ' has no meaning for no-photon problem. Ignored !!'
735: C     call CNTERR( 'WARNING' )
736: C     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
737: C     if ( IRET.ne.0 ) go to 210
738: C     go to 180
739: C   end if
740: C   if ( VSUBF.eq.'N'.and.JNEUT.eq.0 ) then
741: C     write(IMG, '(1x,a,a,a)') '!!! ', VNAME,
742: C &     ' has no meaning for no-neutron problem. Ignored !!'
743: C     call CNTERR( 'WARNING' )
744: C     call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
745: C     if ( IRET.ne.0 ) go to 210
746: C     go to 180
747: C   end if
748: C
749: C
750: C-----
751: C ..... OTHERS (SIZE PARAMETER ETC.) .....
752: C-----
753: C
754: C
755: C
756: C   if ( VNM.eq.'DEPS' ) call VALUE0( VNAME, ICALL, 'R8', DEPS )
757: C   if ( VNM.eq.'DINF' ) call VALUE0( VNAME, ICALL, 'R8', DINF )
758: C   if ( VNM.eq.'NPART' ) call VALUE0( VNAME, ICALL, 'I4', NPART )
759: C   if ( VNM.eq.'NPART' ) call VALUE0( VNAME, ICALL, 'I8', NPART )
760: C   if ( VNM.eq.'NHIST' ) call VALUE0( VNAME, ICALL, 'I4', NHIST )
761: C   if ( VNM.eq.'NHSUB' ) call VALUE0( VNAME, ICALL, 'I4', NHSUB )
762: C   if ( VNM.eq.'NBANK' ) call VALUE0( VNAME, ICALL, 'I4', NBANK )
763: C   if ( VNM.eq.'NBK2' ) call VALUE0( VNAME, ICALL, 'I4', NBK2 )
764: C   if ( VNM.eq.'NBPINT' ) call VALUE0( VNAME, ICALL, 'I4', NBPINT )
765: C   if ( VNM.eq.'NRSINT' ) call VALUE0( VNAME, ICALL, 'I4', NRSINT )
766: C   if ( VNM.eq.'NTMINT' ) call VALUE0( VNAME, ICALL, 'I4', NTMINT )
767: C
768: C   if ( VNM.eq.'NGROUP' ) then
769: C     if ( VSUBF.eq.' ' .or. JNP.eq.0 ) then
770: C       call VALUE0( VNAME, ICALL, 'I4', NGROUP )
771: C       if ( JNEUT.ne.0 ) NGP1 = NGROUP
772: C       if ( JPHOT.ne.0 ) NGP2 = NGROUP
773: C     else if ( VSUBF.eq.'N' ) then
774: C       call VALUE0( VNAME, ICALL, 'I4', NGP1 )
775: C       if ( JPHOT.eq.0 ) NGROUP = NGP1
776: C       if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
777: C     else if ( VSUBF.eq.'P' ) then
778: C       call VALUE0( VNAME, ICALL, 'I4', NGP2 )
779: C       if ( JNEUT.eq.0 ) NGROUP = NGP2
780: C       if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2

```

src/gmvp/intro.f

```

781: C      end if
782: C      end if
783: C      if ( VNM.eq.'NGP1' ) call VALUE0( VNAME, ICALL, 'I4', NGP1 )
784: C      if ( VNM.eq.'NGP2' ) call VALUE0( VNAME, ICALL, 'I4', NGP2 )
785: C
786: C      if ( VNM.eq.'NGROUP' .or. VNM.eq.'NGP' ) then
787: C      call VALUE0( VNAME, ICALL, 'I4', NNNNG )
788: C      if ( IKPID.eq.0 ) then
789: C      if ( KKPID.ne.0 ) then
790: C      NGP(KKPID) = NNNNG
791: C      NGROUP = NNNNG
792: C      if ( KKPID.eq.KPNEUT ) NGP1 = NNNNG
793: C      if ( KKPID.eq.KPPHOT ) NGP2 = NNNNG
794: C      else
795: C      write(IMG,*) '!!! data "NGROUP()" is ambiguous for ',
796: C      & 'multi particle coupled problem (N+P etc.)'
797: C      call CNTERR( 'WARNING' )
798: C      if ( JKPAR(KPNEUT).ne.0 ) then
799: C      write(IPR,*)
800: C      & ' "NGROUP()" is taken as number of NEUTRON energy bin'
801: C      NGP(KPNEUT) = NNNNG
802: C      NGP1 = NNNNG
803: C      end if
804: C      end if
805: C      else
806: C      NGP(IKPID) = NNNNG
807: C      if ( IKPID.eq.KPNEUT ) NGP1 = NNNNG
808: C      if ( IKPID.eq.KPPHOT ) NGP2 = NNNNG
809: C      end if
810: C      end if
811: C
812: C ... NGP1 and NGP2 will be obsolete and they should be replaced
813: C with NGP(KPNEUT) and NGP(KPPHOT).
814: C The following lines are only for backward compatibility.
815: C
816: C      if ( VNM.eq.'NGP1' ) then
817: C      call VALUE0( VNAME, ICALL, 'I4', NGP1 )
818: C      NGP(KPNEUT) = NGP1
819: C      end if
820: C      if ( VNM.eq.'NGP2' ) then
821: C      call VALUE0( VNAME, ICALL, 'I4', NGP2 )
822: C      NGP(KPPHOT) = NGP2
823: C      end if
824: C
825: C      if ( NPKIND.eq.2.and.JKPAR(KPNEUT).ne.0.and.JKPAR(KPPHOT).ne.0
826: C      & .and.NGP1.gt.0.and.NGP2.gt.0 ) then
827: C      NGROUP = NGP1 + NGP2
828: C      end if
829: C
830: C      if ( VNM.eq.'NMEMO' ) call VALUE0( VNAME, ICALL, 'I4', NMEMO )
831: C      if ( VNM.eq.'NMAT' ) call VALUE0( VNAME, ICALL, 'I4', NMAT )
832: C      if ( VNM.eq.'NREG' ) then
833: C      call VALUE0( VNAME, ICALL, 'I4', NREG )
834: C      write(IMG, '(1x,a)')
835: C      & '!!! Direct input for number of region (NREG) has no meaning.'
836: C      write(IMG, '(1x,2a)')
837: C      & ' (Use "%NREG" parameter for region-number-',
838: C      & 'bounded parameters if necessary.)'
839: C      end if
840: C      if ( VNM.eq.'NSCT' ) call VALUE0( VNAME, ICALL, 'I4', NSCT )
841: C      if ( VNM.eq.'NTIME' ) call VALUE0( VNAME, ICALL, 'I4', NTIME )
842: C      if ( VNM.eq.'NSOUR' ) call VALUE0( VNAME, ICALL, 'I4', NSOUR )
843: C      if ( VNM.eq.'NRESP' ) call VALUE0( VNAME, ICALL, 'I4', NRESP )
844: C      if ( VNM.eq.'TCPU' ) call VALUE0( VNAME, ICALL, 'R4', TCPU )
845: C      if ( VNM.eq.'IRAND' ) call VALUE0( VNAME, ICALL, 'I4', IRAND )

```

```

846: C      if ( VNM.eq.'IRAND' ) then
847: C      call VALUE0( VNAME, ICALL, 'I4', IRAND )
848: C      IRNSU = int(IRAND/2.d0**42)
849: C      IRNSM = int((IRAND - IRNSU*2.d0**42)/2.d0**21)
850: C      IRNSL = nint(IRAND - IRNSU*2.d0**42 - IRNSM*2.d0**21)
851: C      end if
852: C
853: C      if ( VNM.eq.'MXPGEN' ) call VALUE0( VNAME, ICALL, 'I4', MXPGEN )
854: C      if ( VNM.eq.'TCUT' ) call VALUE0( VNAME, ICALL, 'R4', TCUT )
855: C      if ( VNM.eq.'SUPPLY' ) call VALUE0( VNAME, ICALL, 'R4', SUPPLY )
856: C      if ( VNM.eq.'WLLIM' ) call VALUE0( VNAME, ICALL, 'R4', WLLIM )
857: C      if ( VNM.eq.'ELOOP' ) call VALUE0( VNAME, ICALL, 'R4', ELOOP )
858: C      if ( VNM.eq.'NFBANK' ) call VALUE0( VNAME, ICALL, 'I4', NFBANK )
859: C      if ( VNM.eq.'NSKIP' ) call VALUE0( VNAME, ICALL, 'I4', NSKIP )
860: C      if ( VNM.eq.'NMXLAGE' ) call VALUE0( VNAME, ICALL, 'I4', NMXLAGE )
861: C      if ( VNM.eq.'NEITER' ) call VALUE0( VNAME, ICALL, 'I4', NEITER )
862: C
863: C      if ( VNM.eq.'NPICT' ) call VALUE0( VNAME, ICALL, 'I4', NPICT )
864: C      if ( VNM.eq.'NMEMS' ) call VALUE0( VNAME, ICALL, 'I4', NMEMS )
865: C      if ( VNM.eq.'NMEMOP' ) call VALUE0( VNAME, ICALL, 'I4', NMEMOP )
866: C      if ( VNM.eq.'NRSKIP' ) call VALUE0( VNAME, ICALL, 'I4', NRSKIP )
867: C
868: C      if ( VNM.eq.'NPDET' ) call VALUE0( VNAME, ICALL, 'I4', NPDET )
869: C      if ( VNM.eq.'IMPMAX' ) call VALUE0( VNAME, ICALL, 'I4', IMPMAX )
870: C
871: C
872: C-----
873: C ..... OTHERS (ARRAYS)
874: C (SIGN CHECK (CALL CKVAL4 OR CKVAL8) ADDED 1/17/1990)
875: C-----
876: C
877: C
878: C-----
879: C ...../PGEOM/.....
880: C-----
881: C
882: C      if ( VNM.eq.'KMAT' .or. VNM.eq.'KREG' ) then
883: C      write(IMG,7020)
884: C 7020 format(/1X,'XXX Old-fashioned input for mat # & region #./'
885: C      & ' specifying "KMAT" & "KREG" separately from the zone data ',
886: C      & 'block of geometry input is not allowed in this version.')
887: C      call CNTERR( 'FATAL' )
888: C      NDUM2 = NINPZ
889: C      call DMREAD( VNM, NDUM1, NDUM2, IRET )
890: C      if ( IRET.ne.0 ) go to 230
891: C      end if
892: C
893: C      if ( VNM.eq.'VOL' ) then
894: C      call ARAYIN( VNAME, A, ICALL, 'R4', LVOL, ICK, JDEBG, LAST,
895: C      & NINPZ, 'NINPZ' )
896: C      call CKVAL4( VNM, A(LVOL), NINPZ, IFL, 1 )
897: C      if ( IFL.ne.0 ) then
898: C      ICHK = ICHK + 1
899: C      write(IMG,7080) VNAME(:NLN)
900: C      call CNTERR( 'FATAL' )
901: C      end if
902: C      end if
903: C      if ( VNM.eq.'PAPER' ) then
904: C      call ARAYIN( VNAME, A, ICALL, 'R4', LPAPER, ICK, JDEBG, LAST,
905: C      & 12*NPICT, ' ' )
906: C      end if
907: C
908: C-----
909: C ...../PSOUR/.....
910: C-----

```

src/gmvp/intro.f

```

911: C
912: C
913:   if ( VNM.eq.'KSOUR' ) then
914:     call ARAYIN( VNAME, A, ICALL, 'I4', LKSOUR, ICK, JDEBG, LAST,
915: &      NSOUR, 'NSOUR' )
916:   end if
917:   if ( VNM.eq.'ISZON' ) then
918:     call ARAYIN( VNAME, A, ICALL, 'I4', LISZON, ICK, JDEBG, LAST, 2
919: &      *NSOUR, ' ' )
920:   end if
921:   if ( VNM.eq.'SOUR' ) then
922:     call ARAYIN( VNAME, A, ICALL, 'R8', LSOUR, ICK, JDEBG, LAST,
923: &      NSOUR, 'NSOUR' )
924:     call CKVAL8( VNAME, A(LSOUR), NSOUR, IFL, 0 )
925:     if ( IFL.ne.0 ) then
926:       ICHK = ICHK + 1
927:       write(IMSG,7080) VNAME(:NLEN)
928:       call CNTERR( 'FATAL' )
929:     end if
930:   end if
931:   if ( VNM.eq.'PSPAC' ) then
932:     call ARAYIN( VNAME, A, ICALL, 'R4', LPSPAC, ICK, JDEBG, LAST,
933: &      10*NSOUR, ' ' )
934:   end if
935:   if ( VNM.eq.'PENRG' ) then
936:     call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KKLIM, NSP1, NGP2,
937: &      NGROUP )
938:     call KGPSET( KNGP(1), KNGP(1), NGP(1), JKPAR(1), KKLIM )
939: C   if ( JNP.eq.0 .or. VSUBF.eq.' ' ) then
940: C     call ARAYIN( VNAME, A, ICALL, 'R4', LPENRG, ICK, JDEBG,
941: C &      LAST, NGROUP*NSOUR, ' ' )
942: C   else
943: C     if ( IKPID.eq.0 .or. KKPID.ne.0 ) then
944: C
945: C       call ARAYIN( VNAME, A, ICALL, 'R4', LPENRG, ICK, JDEBG,
946: C &      LAST, NGROUP*NSOUR, ' ' )
947: C
948: C     else if ( IKPID.ne.0 ) then
949: C
950: C       if ( LPENRG.eq.0 ) then
951: C         call KEFV( A(1), 'PENRG', LPENRG, NGROUP*NSOUR, 'R4',
952: C &      LAST, 0.0, JDEBG )
953: C       end if
954: C       if ( VSUBF.eq.'N' ) then
955: C
956: C         .... input data temporary from A(LTMPRY)
957: C
958: C         call GTLAST( LASTTM )
959: C         call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
960: C &      LAST, NGP(IKPID)*NSOUR, ' ' )
961: C
962: C         .... scatter data a(ltmpry) onto a(lpenrg) ...
963: C
964: C         call ASCTR4( VNAME, A(LPENRG), A(LTMPRY), NGROUP, NSOUR,
965: C &      NGP(IKPID), KNGP(IKPID), NGP(IKPID) )
966: C
967: C         call STLAST( 'PENRG', LASTTM, LAST )
968: C
969: C       else if ( VSUBF.eq.'P' ) then
970: C
971: C         .... input data temporary from A(LTMPRY)
972: C
973: C         call GTLAST( LASTTM )
974: C         call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
975: C &      LAST, NGP2*NSOUR, ' ' )

```

```

976: C
977: C         .... scatter data a(ltmpry) onto a(lpenrg) ...
978: C
979: C         call ASCTR4( VNAME, A(LPENRG), A(LTMPRY), NGROUP, NSOUR,
980: C &      NGP2, NGP1+1, NGP2 )
981: C
982: C         call STLAST( 'PENRG', LASTTM, LAST )
983: C
984: C       end if
985: C     end if
986: C   end if
987: C
988: C   if ( VNM.eq.'NSTIM' ) then
989: C     call ARAYIN( VNAME, A, ICALL, 'I4', LNSTIM, ICK, JDEBG, LAST,
990: C &      NSOUR, 'NSOUR' )
991: C   end if
992: C   if ( VNM.eq.'STIM' ) then
993: C     .... MNSTIM IS SUM OF NSTIM ....
994: C     call ARYSUM( A(LNSTIM), NSOUR, MNSTIM )
995: C     call ARAYIN( VNAME, A, ICALL, 'R4', LSTIM, ICK, JDEBG, LAST,
996: C &      MNSTIM, ' ' )
997: C   end if
998: C   if ( VNM.eq.'PSTIM' ) then
999: C     call ARYSUM( A(LPSTIM), NSOUR, MNSTIM )
1000: C     call ARAYIN( VNAME, A, ICALL, 'R4', LPSTIM, ICK, JDEBG, LAST,
1001: C &      MNSTIM, ' ' )
1002: C   end if
1003: C   if ( VNM.eq.'NSANG' ) then
1004: C     call ARAYIN( VNAME, A, ICALL, 'I4', LNSANG, ICK, JDEBG, LAST,
1005: C &      NSOUR, 'NSOUR' )
1006: C   end if
1007: C   if ( VNM.eq.'SANG' ) then
1008: C     .... MNSANG IS SUM OF NSANG ....
1009: C     call ARYSUM( A(LNSANG), NSOUR, MNSANG )
1010: C     call ARAYIN( VNAME, A, ICALL, 'R4', LSANG, ICK, JDEBG, LAST,
1011: C &      MNSANG, ' ' )
1012: C   end if
1013: C   if ( VNM.eq.'PSANG' ) then
1014: C     call ARYSUM( A(LNSANG), NSOUR, MNSANG )
1015: C     call ARAYIN( VNAME, A, ICALL, 'R4', LPSANG, ICK, JDEBG, LAST,
1016: C &      MNSANG, ' ' )
1017: C   end if
1018: C   if ( VNM.eq.'SAXIS' ) then
1019: C     call ARAYIN( VNAME, A, ICALL, 'I4', LSAXIS, ICK, JDEBG, LAST,
1020: C &      NSOUR*3, ' ' )
1021: C   end if
1022: C
1023: C   if ( VNM.eq.'IFISM' ) then
1024: C     call ARAYIN( VNAME, A, ICALL, 'I4', LIFISM, ICK, JDEBG, LAST,
1025: C &      NSOUR, 'NSOUR' )
1026: C   end if
1027: C
1028: C -----
1029: C ...../PXSEC/.....
1030: C -----
1031: C
1032: C   IF(VNM.EQ.'WGTF') THEN
1033: C     CALL ARAYIN(VNAME,A,ICALL,'R4', LWGTF , ICK,
1034: C +      JDEBG, LAST,      NREG , 'NREG' )
1035: C     CALL CKVAL4(VNAME,A(LWGTF),NREG,ICK,1)
1036: C   ENDIF
1037: C
1038: C   if ( VNM.eq.'FKAI' ) then
1039: C     write(IMSG,*) '!!! Input "FKAI" data separately from',
1040: C &      ' cross section specifications ($CROSS SECTION)',

```

src/gmvp/intro.f

```

1041:      &          ' is not recommended.'
1042:      call CNTERR( 'WARNING' )
1043:
1044:      call ARAYIN( VNAME, A, ICALL, 'R4', LFKAI, ICK, JDEBG, LAST,
1045:      &          NTGX*NMAT, ' ' )
1046:      call CKVAL4( VNAME, A(LFKAI), NTGX*NMAT, IFL, 0 )
1047:      if ( IFL.ne.0 ) then
1048:          ICHK = ICHK + 1
1049:          write(IMG,7080) VNAME(:NLEN)
1050:          call CNTERR( 'FATAL' )
1051:      end if
1052:  end if
1053: C IF(VNM.EQ.'WGTG') THEN
1054: C      CALL ARAYIN(VNAME,A,ICALL,'R4', LWGTG , ICK,
1055: C      +          JDEBG LAST, NREG , 'NREG' )
1056: C      CALL CKVAL4(VNAME,A(LWGTG),NREG,ICK,1)
1057: C  ENDDIF
1058: C
1059: C-----/PTALY/-----
1060: C ...../PTALY/.....
1061: C-----
1062: C
1063:      if ( VNM.eq.'RESP' ) then
1064:          call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
1065:      &          NGROUP )
1066:          call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
1067:          if ( JNP.eq.0 .or. VSUBF.eq.' ' ) then
1068:              if ( IKPID.eq.0 .or. IKPID.ne.0 ) then
1069:                  call ARAYIN( VNAME, A, ICALL, 'R4', LRESP, ICK, JDEBG, LAST,
1070:      &          NGROUP*NRESP, ' ' )
1071:                  call CKVAL4( VNM, A(LRESP), NGROUP*NRESP, IFL, 0 )
1072:                  if ( IFL.ne.0 ) then
1073:                      ICHK = ICHK + 1
1074:                      write(IMG,7080) VNAME(:NLEN)
1075:                      call CNTERR( 'FATAL' )
1076:                  end if
1077:              else
1078:                  if ( LRESP.eq.0 ) then
1079:                      call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4', LAST,
1080:      &          0.0, JDEBG )
1081:                  end if
1082:                  if ( IKPID.ne.0 ) then
1083:                      call GTLAST( LASTTM )
1084:                      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1085:      &          LAST, NGP(IKPID)*NRESP, ' ' )
1086: C
1087: C          .... scatter data a(ltmpy) onto a(lresp) ...
1088: C
1089:          call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
1090:      &          NGP(IKPID), KNGP(IKPID), NGP(IKPID) )
1091: C
1092:          call STLAST( 'RESP', LASTTM, LAST )
1093:      end if
1094:
1095:      if ( VSUBF.eq.'N' ) then
1096: C
1097: C          .... input data temporary from A(LTMPRY)
1098: C
1099: C      call GTLAST( LASTTM )
1100: C      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1101: C      &          LAST, NGP1*NRESP, ' ' )
1102: C
1103: C          .... scatter data a(ltmpy) onto a(lresp) ...
1104: C
1105: C      call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,

```

```

1106: C      &          NGP1, 1, NGP1 )
1107: C      call STLAST( 'RESP', LASTTM, LAST )
1108: C
1109: C      else if ( VSUBF.eq.'P' ) then
1110: C
1111: C          .... input data temporary from A(LTMPRY)
1112: C
1113: C      call GTLAST( LASTTM )
1114: C      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1115: C      &          LAST, NGP2*NRESP, ' ' )
1116: C
1117: C          .... scatter data a(ltmpy) onto a(lresp) ...
1118: C
1119: C      call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
1120: C      &          NGP2, NGP1+1, NGP2 )
1121: C
1122: C      call STLAST( 'RESP', LASTTM, LAST )
1123: C      end if
1124: C      end if
1125: C      end if
1126: C
1127: C      if ( VNM.eq.'ENGYB' .or. VNM.eq.'ENGPB' ) then
1128: C
1129: C          call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
1130: C      &          NGROUP )
1131: C          call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
1132: C
1133: C          if ( LENGYB.eq.0 ) then
1134: C              call KEPV( A(1), 'ENGYB', LENGYB, KENGP(KPLIM+1)-1, 'R4',
1135: C      &          LAST, 0.0, JDEBG )
1136: C
1137: C          end if
1138: C
1139: C          KP = 0
1140: C          if ( VNM.eq.'ENGPB' ) then
1141: C              KP = KPPHOT
1142: C              if ( JKPAR(KPPHOT).eq.0 ) then
1143: C                  KP = 0
1144: C              end if
1145: C          else if ( IKPID.eq.0.and.JKPAR(KPNEUT).ne.0 ) then
1146: C              KP = KPNEUT
1147: C          else if ( IKPID.ne.0 ) then
1148: C              KP = IKPID
1149: C          end if
1150: C          if ( KP.eq.0 ) then
1151: C              write(IMG,*) '!!! Energy boundary data <', VNAME(:NLEN),
1152: C      &          '> is not effective for current input.'
1153: C          call DMREAD( VNAME(:NLEN), IDUMMY, IDUMMY, IRET )
1154: C          if ( IRET.ne.0 ) go to 230
1155: C          go to 200
1156: C      end if
1157: C
1158: C      NB = NGP(KP) + 1
1159: C      call R4READ( VNAME(:NLEN), A(LENGYB+KENGP(KP)-1), NA, -NB, IRET
1160: C      &          )
1161: C      if ( NA.ne.NB ) then
1162: C          write(IMG,*) 'XXX Number of energy boundary data <',
1163: C      &          VNAME(:NLEN), '> (=, NA,
1164: C      &          ' )i is different from an expected one (=, NB, ').'
1165: C          ICHK = ICHK + 1
1166: C          call CNTERR( 'FATAL' )
1167: C      end if
1168: C
1169: C      call CKVAL4( VNM, A(LENGYB+KENGP(KP)-1), NGP(KP)+1, IFL, 1 )
1170: C      if ( IFL.ne.0 ) then
1171: C          ICHK = ICHK + 1

```

src/gmvp/intro.f

```

1171:      write(IMG,7080) VNAME(:NLEN)
1172:      call CNTERR( 'FATAL' )
1173:    end if
1174:    if ( KP.eq.KPPHOT ) then
1175:      if ( LENGPB.eq.0 ) then
1176:        call KEVP( A(1), 'ENGPB', LENGPB, NGP(KP)+1, 'R4', LAST,
1177:      &          0.0, JDEBG )
1178:      end if
1179:      call ATRANS( A(LENGYB+KENG(P(KP)-1), A(LENGPB), NGP(KP)+1 )
1180:    end if
1181:  C
1182:    ICALL = 1
1183:  end if
1184:  C
1185:  .. energy boundary data for photon in NEUTRON/PHOTON coupled mode
1186:  are stored in ENGYB(NGP1+2:NGP1+NGP2+2) and ENGPB(1:NGP2+1)
1187:  C
1188:  if ( VNM.eq.'ENGYB' ) then
1189:    NGB = NGROUP + 1
1190:    if ( JNP.ne.0 ) NGB = NGB + 1
1191:  C
1192:    if ( JNP.eq.0.and.VSUBF.eq.' ' ) then
1193:      call ARAYIN( VNAME, A, ICALL, 'R4', LENGYB, ICK, JDEBG,
1194:    &      LAST, NGB, ' ' )
1195:      call SIZCHK( VNAME, 'NGP1', NGP1, NFIMP(0)-1, ICK )
1196:  C
1197:      call CKVAL4( VNM, A(LENGYB), NGB, IFL, 1 )
1198:      if ( IFL.ne.0 ) then
1199:        ICHK = ICHK + 1
1200:        write(IMG,7080) VNAME(:NLEN)
1201:        call CNTERR( 'FATAL' )
1202:      end if
1203:    else
1204:      if ( LENGYB.eq.0 ) then
1205:        call KEVP( A(1), 'ENGYB', LENGYB, NGB, 'R4', LAST, 0.0,
1206:    &          JDEBG )
1207:      end if
1208:  C
1209:      if ( VSUBF.eq.'N' .or. VSUBF.eq.' ' ) then
1210:        .... input data temporary from A(LTMPRY)
1211:  C
1212:        call GTLAST( LASTTM )
1213:        call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
1214:    &          LAST, NGP1+1, ' ' )
1215:        call SIZCHK( VNAME, 'NGP1', NGP1, NFIMP(0)-1, ICK )
1216:        call CKVAL4( VNAME, A(LTMPRY), NGP1+1, IFL, 1 )
1217:        if ( IFL.ne.0 ) then
1218:          ICHK = ICHK + 1
1219:          write(IMG,7080) VNAME(:NLEN)
1220:          call CNTERR( 'FATAL' )
1221:        end if
1222:  C
1223:        .... scatter data a(ltmpy) onto a(lengyb) ...
1224:  C
1225:        call ASCTR4( VNAME, A(LENGYB), A(LTMPRY), NGB, 1, NGP1+1,
1226:    &          1, NGP1+1 )
1227:  C
1228:        call STLAST( 'ENGYB', LASTTM, LAST )
1229:  C
1230:      else if ( VSUBF.eq.'P' ) then
1231:  C
1232:        .... input data temporary from A(LTMPRY)
1233:  C
1234:        call GTLAST( LASTTM )
1235:        call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,

```

```

1236:  C      &          LAST, NGP2+1, ' ' )
1237:  C      call SIZCHK( VNAME, 'NGP2', NGP2, NFIMP(0)-1, ICK )
1238:  C      call CKVAL4( VNAME, A(LTMPRY), NGP2+1, IFL, 1 )
1239:  C      if ( IFL.ne.0 ) then
1240:        ICHK = ICHK + 1
1241:        write(IMG,7080) VNAME(:NLEN)
1242:        call CNTERR( 'FATAL' )
1243:      end if
1244:  C
1245:      .... scatter data a(ltmpy) onto a(lengyb) ...
1246:  C
1247:      NGP2S = NGP1 + 2
1248:      if ( NGP1.le.0 ) NGP2S = 1
1249:      call ASCTR4( VNAME, A(LENGYB), A(LTMPRY), NGB, 1, NGP2+1,
1250:    &      NGP2S, NGP2+1 )
1251:  C
1252:      call STLAST( 'ENGYB', LASTTM, LAST )
1253:  C
1254:      ... make copy of photon energy boundary in 'ENGPB' ..
1255:  C
1256:      if ( LENGPB.le.0 ) then
1257:        call KEVP( A(1), 'ENGPB', LENGPB, NGP2+1, 'R4', LAST,
1258:    &          0.0, JDEBG )
1259:      end if
1260:      call ATRANS( A(LENGYB+NGP2S-1), A(LENGPB), NGP2+1 )
1261:  C
1262:      end if
1263:    end if
1264:  end if
1265:  C
1266:  if ( VNM.eq.'ENGPB' ) then
1267:    call ARAYIN( VNAME, A, ICALL, 'R4', LENGPB, ICK, JDEBG, LAST,
1268:  &      NGP2+1, ' ' )
1269:    call SIZCHK( VNAME, 'NGP2', NGP2, NFIMP(0)-1, ICK )
1270:    call CKVAL4( VNM, A(LENGPB), NGP2+1, IFL, 1 )
1271:    if ( IFL.ne.0 ) then
1272:      ICHK = ICHK + 1
1273:      write(IMG,7080) VNAME(:NLEN)
1274:      call CNTERR( 'FATAL' )
1275:    end if
1276:  C
1277:      ... make copy of photon energy boundary in 'ENGYB' ..
1278:  C
1279:      NGB = NGROUP + 1
1280:      if ( JNP.ne.0 ) NGB = NGB + 1
1281:      NGP2S = NGP1 + 2
1282:      if ( NGP1.le.0 ) NGP2S = 1
1283:      if ( LENGYB.eq.0 ) then
1284:        call KEVP( A(1), 'ENGYB', LENGYB, NGB, 'R4', LAST, 0.0,
1285:    &          JDEBG )
1286:      end if
1287:      call ATRANS( A(LENGPB), A(LENGYB+NGP2S-1), NGP2+1 )
1288:      Cccc call atrans(a(lengpb),a(lengyb+ngp+ngp2s-1),ngp2+1)
1289:  C
1290:      end if
1291:  C
1292:      if ( VNM.eq.'TIMEB' ) then
1293:        call ARAYIN( VNAME, A, ICALL, 'R4', LTIMEB, ICK, JDEBG, LAST,
1294:    &          NTIME+1, ' ' )
1295:        call SIZCHK( VNAME, 'NTIME', NTIME, NFIMP(0)-1, ICK )
1296:      end if
1297:  C
1298:      if ( VNM.eq.'WTIME' ) then
1299:  C
1300:

```

src/gmvp/intro.f

```

1301:      call ARAYIN( VNAME, A, ICALL, 'R4', LWTIME, ICK, JDEBG, LAST,
1302:      &          NTIME, ' ' )
1303:      end if
1304:
1305: C      IF(VNM.EQ.'TRVOL') THEN
1306: C          CALL ARAYIN(VNAME,A,ICALL,'R4', LTRVOL , ICK,
1307: C      &          JDEBG, LAST ,          NTREG , 'NTREG' )
1308: C          CALL CKVAL4(VNM, A(LTRVOL),NTREG,ICK, 1)
1309: C      ENDIF
1310: C
1311: C      ... velocity averaged in group ...
1312: C
1313: C      if ( VNM.eq.'VEL' ) then
1314: C          call ARAYIN( VNAME, A, ICALL, 'R4', LVEL, ICK, JDEBG, LAST,
1315: C      &          NGROUP, 'NGROUP' )
1316: C      end if
1317: C
1318: C-----
1319: C      .... target position of path stretching .....
1320: C-----
1321: C
1322: C      if ( VNM.eq.'PSXYZ' ) then
1323: C          call ARAYIN( VNAME, A, ICALL, 'R8', LPSXYZ, ICK, JDEBG, LAST,
1324: C      &          3, ' ' )
1325: C          if ( MOD(NFINP(0),3).ne.0 ) then
1326: C              write(IMG,7040) NFINP(0)
1327: C      7040      format(/1X,'XXX Number of input data PSXYZ() is ',I4
1328: C      &          ' while a multiple of 3 is required.')
1329: C          call CNTERR( 'FATAL' )
1330: C          else if ( NFINP(0).eq.0 ) then
1331: C              call PUTVD( A(LPSXYZ), 3, 0.0D0 )
1332: C          end if
1333: C      end if
1334: C
1335: C-----
1336: C      .... POINT DETECTOR : INPUT ONLY 5 PARAMETERS HERE. ....
1337: C-----
1338: C
1339: C      if ( VNM.eq.'XPDET' ) then
1340: C          call ARAYIN( VNM, A, ICALL, 'R8', LXPDET, ICK, JDEBG, LAST,
1341: C      &          NPDET*5, ' ' )
1342: C      end if
1343: C      if ( VNM.eq.'SPDET' ) then
1344: C          call ARAYIN( VNM, A, ICALL, 'I4', LJSPDT, ICK, JDEBG, LAST,
1345: C      &          NPDET, ' ' )
1346: C      end if
1347: C
1348: C-----
1349: C      ..... INPUT FOR CUSTOMIZED VERSION .....
1350: C-----
1351: C
1352: C      $CUSTOM
1353: C      ....
1354: C      $END CUSTOM
1355: C
1356: C      ( ignored in the standard version )
1357: C
1358: C      if ( VNAME.eq.'$CUSTOM' ) then
1359: C      CCCCC      call CUSTOM( IOIN, IPR, LAST )
1360: C          call CUSTOM( IOIN, IPR, A, A, CHA )
1361: C          ICALL = 1
1362: C      end if
1363: C
1364: C-----
1365: C      .... TRIED TO INPUT INVALID ARRAY OR VARIABLES !!!! .....

```

```

1366: C-----
1367: C
1368: C      if ( ICALL.eq.0 ) then
1369: C          write(IMG,7060) VNAME(:ICLEN(VNAME))
1370: C      7060      format('!!! Input data whose name is <',A,
1371: C      &          '> does not exist or direct input is not allowed !!!')
1372: C          call CNTERR( 'WARNING' )
1373: C          call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
1374: C          if ( IRET.ne.0 ) go to 230
1375: C      end if
1376: C      go to 200
1377: C
1378: C
1379: C-----
1380: C
1381: C
1382: C      INPUT END .....
1383: C
1384: C      210 continue
1385: C
1386: C      7080 format(/1X,'XXX Data <',A,'> has a value with invalid sign.'/)
1387: C
1388: C      if ( ICHK.ne.0 ) then
1389: C          write(IMG,7100)
1390: C      7100      format('1'///1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX')/1X,
1391: C      &          ' Your input includes data of invalid sign !!'/1X,
1392: C      &          ' Please check input data or error-messages !!'/
1393: C      &          1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX'))
1394: C          call CNTERR( 'FATAL' )
1395: C      end if
1396: C      if ( ICK.ne.0 ) then
1397: C          write(IMG,7120)
1398: C      7120      format('1'///1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX')/1X,
1399: C      &          ' Your input includes data of invalid length !!'/
1400: C      &          1X,' Please check input data or error-messages !!'
1401: C      &          //1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX'))
1402: C          call CNTERR( 'FATAL' )
1403: C      end if
1404: C
1405: C-----
1406: C      PREPARE NON-INPUT PARAMETERS
1407: C-----
1408: C
1409: C      JNRUN = 0
1410: C
1411: C      call INTRO2( A, H, CHA, LIMIT, LIMITL, LIMITC, TITLE, MAXSF, JNP,
1412: C      &          TCUTDM )
1413: C
1414: C      ( IF JNRUN > 0 , NO RANDOM WALK NECESSARY )
1415: C
1416: C      call CPUTM( T1 )
1417: C      write(IPR,'(/1X,a,e12.4,a/)' )
1418: C      &          ' == CPU TIME FOR DATA PREPARATION = ', T1 - T0, ' SEC'
1419: C
1420: C-----
1421: C      .... DRAW PICTURES
1422: C-----
1423: C
1424: C      if ( JPICT.ne.0 ) then
1425: C          call PABLO( A, LIMIT, LWORK, NPICT, NZONE, NSDA, NZDA, NCELL,
1426: C      &          JLATT, JTLLT, JVMNT, JSIMP, JDEBG, A(LIDCEL),
1427: C      &          A(LIPCEL), A(LKCELL), A(LSDA), A(LKINPZ), A(LKMAT),
1428: C      &          A(LKZREG), A(LKZDA), A(LKZAA), A(LPAPER) )
1429: C      end if
1430: C      C/#IF PARA(PVM MPI)

```

src/gmvp/intro.f

```
1431: C
1432: C ... skip random number for sub tasks & reset random number parameters
1433: C
1434:       do 220 I = 0, IDTASK - 1
1435:         call RANU2( IRAND, RTEMP, 1, ICON )
1436:       220 continue
1437: C
1438:       NRSTEP = NTASK
1439:       if ( MOD(NTASK,2).eq.0 ) NRSTEP = NRSTEP + 1
1440:       call RNINIT( NRSTEP )
1441: C
1442: C/#ENDIF
1443: C
1444: C-----
1445: C ..... PRINT CONTENTS OF COMMON /SIZES/
1446: C-----
1447: C
1448:       call PRSIZE( IPR )
1449: C
1450: C-----
1451: C ..... DEBUG PRINT OF COMMONS .....
1452: C-----
1453: C
1454:       if ( JDEBG(1).ne.0 ) call CHECK( ' === END OF INTRO === ' )
1455: C
1456: C-----
1457: C ..... CHECK INPUT OR PROCESSING ERROR .....
1458: C-----
1459: C
1460:       call PRNERR( IPR, 'AFTER DATA INPUT' )
1461:       call CHKERR( 'WARNING ', KK )
1462:       if ( KK.gt.0 ) then
1463:         write(IPR,7140)
1464: 7140    format(' *** SOME WARNING MESSAGES HAVE BEEN ISSUED. ****'//
1465:              & ' I recommend you checking your input data',
1466:              & ' once more.'// ' Warning message is printed ',
1467:              & 'with "!!!" symbol on the head of printed line.'//)
1468:       end if
1469: C
1470:       JSTOP = 0
1471:       call CHKERR( 'FATAL', KK )
1472: C
1473:       if ( KK.gt.0 ) then
1474:         JSTOP = 1
1475:         write(IPR,7160)
1476: 7160    format('1','*** FATAL ERRORS HAVE BEEN DETECTED',
1477:              & ' IN INPUT-PHASE. STOP EXECUTION. ***'//
1478:              & ' Message for fatal error is printed ',
1479:              & 'with ''xxx'' symbol on the head of printed line.'//
1480:              & ' You must check your input data carefully !!')
1481: C/#IF UNIX
1482:       call FLUSHSTD( )
1483: C/#ENDIF
1484:       stop 888
1485:     end if
1486: C
1487: C-----
1488: C IF JNRUN > 0, NO RANDOM WALK WILL BE DONE.
1489: C-----
1490: C
1491: CCC   if ( JNRUN.ne.0 ) JSTOP = 1
1492: C
1493: C-----
1494: C Check flag JRUNM :
1495: C-----
```

```
1496: C
1497:       if ( MOD(JRUNM,10).ne.0 .or. MOD(JRUNM/10,10).ne.0 ) then
1498:         JSTOP = 1
1499:         write(IPR,7180) JRUNM
1500: 7180    format('1'/1X,'=== Stopped before randomwalk with ',
1501:              & ' "RUN-MODE" control option : RUN-MODE = ',I9)
1502:       end if
1503:       return
1504: C
1505: C-----
1506: C Error in 'DMREAD':
1507: C-----
1508: C
1509:       230 write(IPR,7200)
1510: 7200    format(1X,' XXX Unexpected end of data or data read error ',
1511:              & 'during skipping unnecessary or invalid data.')
1512:       call PRSTOP( 1, 'ERROR DURING SKIPPING INPUT DATA.' )
1513:       stop 888
1514: C
1515:       end
```

src/gmvp/jnput.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine JNPUT( SIGT, NSIG, IMEND, JEIGN, JFISS, JNEUT, JPHOT,
3:     & JSEP )
4: C=====
5: C PURPOSE: X-SEC INPUT & PREPROCESSING CONTROL
6: C CALLED IN: XSECIN
7: C CALLS: MACRO3,MACRO1,APMATX,READSG,STORE,LEGEND,ANGLES,BMCMK0,BMCMK1
8: C=====
9:   real TEMP(80), C1(15)
10:  common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
11:    & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
12:    & ISPORT, INPBUF, ISIGOG, INEFOG, IABSOG, ITOTSG, NGP,
13:    & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
14:    & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
15:    & IADJ, NME, LOC, INGS, INSG, I1, I0,
16:    & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRDSG,
17:    & ISTR, IPRIN, IPMU, IMOM, IDTF, ISTAT, IPUN,
18:    & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
19:    & INNN, IGGG, IPNGP, IPSOG, IPSGOG, IANG, INUFIS,
20:    & IS2DP, IS2PP
21: C * * * *
22:  common /MOMENT/ NMOM, MOMENT(25), NF, F(25)
23:  common /MEANS/ ITEST, NANS, MEAN(21), VAR(20),
24:    & NORMAL(20)
25:  common /RESULT/ POINT(21), WEIGHT(21), ROOT(21,21)
26: C
27:  common /DDX/ IDDX, MAXU, MAXSD, IMAX, NLASTD, MAXDDX
28: C
29:  COMMON /ARRAY/ SIGT(1)
30:  integer NSIG(1)
31:  real SIGT(1)
32: C
33: C ... small value replaced with zero cross section
34: C
35:  parameter( stfix = 1.0E-35 )
36: C
37:  include '../shared/INC/_IOUNIT'
38:  include 'INC/_IOUNIT2'
39: C
40: C*****
41: C
42: C ..... ZERO CLEAR OF WORKING AREA IN DDX CASE .....
43: C
44:  if ( IDDX.ne.0 ) then
45:    call MACRO3( 1, SIGT, NSIG(INNN+1) )
46:    call RWIND( IOWDX1 )
47:  end if
48: C
49:  call LABEL( I0, 'MIXING TABLE' )
50: C
51: C ----- READ DATA X-4 -----
52: C
53:  do 100 N = 1, NMIX
54:    read(I1,*) MIX, NEL, RH0
55: C
56: C   WRITE(I0,1070) MIX,NEL, nsig(idel+(abs(nel)-1)*ncoef),RH0
57: C1070  FORMAT(5X,'MEDIA ',I4,' CONTAINS ',I4,' 'TH ELEMENT (ID=',
58: C   & i8,') WITH NUMBER DENSITY ',1PE13.4 )
59: C
60:  write(I0,7000) MIX, NEL, RH0
61: 7000  format(5X,'MEDIA ',I4,' CONTAINS ',I4,' 'TH ELEMENT ',
62: C   & 'WITH NUMBER DENSITY ',1PE13.4)
63: C
64:  I9 = IRSG + N
65:  NSIG(I9) = MIX

```

```

66:  NSIG(I9+NMIX) = NEL
67:  SIGT(I9+2*NMIX) = RH0
68:  KM = MIX
69:  LLLL = NEL
70:  L = IABS(NEL)
71: C
72: C ..... MIXING & B.M.C TRANSFORMATION FOR DDX .....
73: C
74:  if ( IXTAPE.ge.0.and.IDDX.ne.0 )
75:    & call MACRO1( MIX, LLLL, RH0, NELEM, SIGT, NSIG(INNN+1) )
76: 100 continue
77: C
78:  INUSN = 0
79:  NGP3 = INGP*(INDS+IHT+NUS)
80:  NMOM = NCOEF - 1
81:  NTGMT = NTG + NME - 1
82: C
83: C-----
84: C Clearing until end of array is not recommended for virtual memory
85: C system because it may create many unused memory pages.
86: C-----
87: C
88: C*VOCL LOOP,VECTOR
89: C   do 110 I = ISTART + 1, NLASTD
90: C     SIGT(I) = 0.0
91: C   110 continue
92: C
93:  if ( IDDX.ne.0 ) call APMATX( SIGT, NSIG )
94: C
95: C ..... READ AND STORE PL CROSS SECTIONS (UNTIL LABEL 5500) ....
96: C
97:  if ( IDDX.eq.0 ) then
98: C
99: C M.SASAKI ... probably unnecessary memory clear operation
100: C   because NLASTD is the last array index of SIGT here.
101: C
102: C*VOCL LOOP,VECTOR
103: C   do 110 I = NLASTD + 1, ITOTSG
104: C     SIGT(I) = 0.0
105: C   110 continue
106: C
107:  KKK = 0
108: C
109: C I INDEX USED FOR GROUP
110: C J INDEX USED FOR DOWNSCATTER
111: C L INDEX USED FOR ELEMENT OR MOMENT
112: C M INDEX USED FOR MEDIA
113: C N INDEX USED FOR MIXTURE
114: C K INDEX USED FOR COEFFICIENT
115: C KE ALSO USED FOR ELEMENT
116: C KM ALSO USED FOR MEDIA
117: C
118:  do 180 L = 1, NELEM
119: 180  do 170 K = 1, NCOEF
120:    KKK = KKK + 1
121: C
122:  call READSG( L, K, SIGT, NSIG, JSEP )
123: C
124:  if ( IRDSG.gt.0 ) then
125: C
126: C   WRITE(I0,1010) L,nsig(idel+kkk),K
127: C1010  FORMAT(1H1,' CROSS SECTIONS AS READ FOR ELEMENT ',I3,
128: C   & ' (ID=',i10,') COEFFICIENT ',I3//1x,
129: C   & 'GP ABS XSEC NU*FISS TOTAL FROM GROUP',
130: C   & ' TRANSFER PROBABILITIES ' )

```


src/gmvp/jnput.f

```

131: C
132:       write(I0,7020) L, K
133: 7020      format('1',' CROSS SECTIONS AS READ FOR ELEMENT ',I3,
134:      &          ' COEFFICIENT ',I3//1X,
135:      &          'GP ABS XSEC NU*FISS TOTAL FROM GROUP',
136:      &          ' TRANSFER PROBABILITIES ')
137:
138:       NTU      = NTS + IHT - 1
139:       do 120 I = NME, NTGMT
140:         J1      = 1 + (I-1)*(INDS+IHT+NUS) + INPBUF
141:         J2      = MIN(J1+I+IHT-NME+NUS,J1+NTU)
142:         J        = I + NME + 1
143:         write(I0,7040) J, (SIGT(K1),K1=J1,J2)
144:         continue
145: 7040      format(I4,1P3E11.3,2X,8E10.3/(39X,8E10.3))
146:       end if
147: C
148:       call STORE( L, K, SIGT, NSIG )
149: C
150:       if ( ISTR.gt.0 ) then
151: C
152:         write(I0,7060) L, K
153: 7060      format('1',' INPUT MATERIAL NUMBER ',I3,' COEFFICIENT',
154:      &          I4// GROUP',4X,'DOWNSCATTER MATRIX')
155:         if ( K.le.1 ) then
156: C
157:         ..... P ZERO TABLE .....
158: 130      do 140 I = 1, NTG
159:           J1      = ISPORT + (L-1)*NTS*NTG + (I-1)*NTS + 1
160:           J2      = J1 + NSIG(INNN+I) - 1
161:           write(I0,7080) I, (SIGT(J3),J3=J1,J2)
162:           continue
163: 7080      format(I5,3X,1P10E11.3/(8X,10E11.3))
164:         else
165:           do 150 I = 1, NGP
166:             J1      = IABSOG + (L-1)*ITEML + NDSNGP*(K-2)
167:             &          + NSIG(INGS+I) + 1
168:             J2      = J1 + NSIG(INNN+I) - 1
169:             write(I0,7080) I, (SIGT(J3),J3=J1,J2)
170:             continue
171:           if ( NGG.gt.0 ) then
172:             do 160 I = NGP + 1, NTG
173:               J1      = ITEMG + (L-1)*ITEML + NDSNGG*(K-2)
174:               &          + NSIG(INGSG+I-NGP) + 1
175:               J2      = J1 + NSIG(INNN+I) - 1
176:               write(I0,7080) I, (SIGT(J3),J3=J1,J2)
177:               continue
178:             end if
179:           end if
180:         continue
181: 180      continue
182: C
183: C ... this memory clear is too lazy!!
184: C*VOCL LOOP,VECTOR
185: C      do 200 I = INPBUF + 1, ISPORT
186: C        SIGT(I) = 0.0
187: C 200      continue
188: C
189: C*VOCL LOOP,VECTOR
190: C      do 190 I = ISTART + 1, IMEND
191: C        SIGT(I) = 0.0
192: C 190      continue
193: C
194: C ..... MIXING LOOP (PL)
195: C

```

```

196:
197:       do 270 N = 1, NMIX
198:         I9      = IRSG + N
199:         MIX      = NSIG(I9)
200:         NEL      = NSIG(I9+NMIX)
201:         RH0      = SIGT(I9+2*NMIX)
202:         KM      = MIX
203:         LLLL     = NEL
204:         L        = IABS(NEL)
205:         RHOT     = RH0
206:         IK       = (KM-1)*NTG
207:
208:         IN      = (L-1)*NTG
209:         do 230 I = 1, NGP
210:           IN     = IN + 1
211:           IL     = IK + I
212:           IKK    = IL + ISTART
213:           INN    = IN + ISIGOG
214: C
215:         TOTAL
216:         SIGT(IKK) = SIGT(IKK) + RHOT*SIGT(INN)
217:         IKK      = IL + INUFIS
218:         INN      = IN + INFPOG
219: C
220:         NU SIGF
221:         SIGT(IKK) = SIGT(IKK) + RHOT*SIGT(INN)
222:         NDSK     = NSIG(INNN+I)
223:         NKND     = ISPORT + (L-1)*NTS*NTG + (I-1)*NTS
224:         ISS      = IK + ISCCOG + I
225:         IJK      = IFPOG + NSIG(INGS+I) + 1 + (NDSNGP+NDSNGG)*
226:         &          (KM-1)
227:         ISU      = IK + INUS + I
228:         do 200 J2 = 1, NDSK
229: C
230:         NEUTRON DOWN SCATTER
231:         IS       = NKND + J2
232:         SIGT(IJK) = SIGT(IJK) + RHOT*SIGT(IS)
233: C
234:         SUM DOWNSCATTERS + UPSCATTERS
235:         SIGT(ISS) = SIGT(ISS) + RHOT*SIGT(IS)
236:         if ( J2.le.NUS ) SIGT(ISU) = SIGT(ISU) + RHOT*
237:         &          SIGT(IS)
238:         IJK      = IJK + 1
239:         continue
240:         if ( NTG.gt.NGP ) then
241: C
242:         CALCULATE GAMMA GENERATION CROSS SECTION
243:         NKND     = ISPORT + (L-1)*NTS*NTG + (I-1)*NTS
244:         IGS3     = IGABOG + I + NGP*(KM-1)
245:         IGS      = NKND + NGP - I + 2 + NUS
246:         IGS2     = IGS + NGG - 1
247:         IGS4     = (I-1)*NGG + IFNGP + NGP*NGG*(KM-1)
248:         do 220 J2 = IGS, IGS2
249: C
250:         DETERMINE NEUTRON TO GAMMA TRANSFERS
251:         IGS4     = IGS4 + 1
252:         SIGT(IGS4) = SIGT(IGS4) + RHOT*SIGT(J2)
253:         SIGT(IGS3) = SIGT(IGS3) + RHOT*SIGT(J2)
254:         continue
255:         end if
256:         continue
257:         if ( NTG.gt.NGP ) then
258: C
259:         IK       = (KM-1)*NTG
260:         IN      = (L-1)*NTG
261: C
262:         SUM DOWNSCATTERS FOR GAMMAS
263:         do 260 I = NGP + 1, NTG
264:           IKK    = IK + I + ISTART
265:           INN    = IN + I + ISIGOG
266:           SIGT(IKK) = SIGT(IKK) + RHOT*SIGT(INN)
267:           IKK    = IK + I + INUFIS
268:           INN    = IN + I + INFPOG

```

src/gmvp/jnput.f

```

261:      SIGT(IKK) = SIGT(IKK) + RHOT*SIGT(INN)
262:      NDSK = NSIG(INNN+I)
263:      ISS = IK + I + ISCCOG
264:      IJK = IFSPG + NSIG(INSG+I-NGP) + NDSNGP
265:      &      + (NDSNGP+NDSNGG)*(KM-1)
266:      NKND = ISPORT + (L-1)*NTS*NTG + (I-1)*NTS
267:      ISU = IK + INUS + I
268:      do 250 J2 = 1, NDSK
269:        IJ = IJK + J2
270:        IS = NKND + J2
271:        SIGT(ISS) = SIGT(ISS) + RHOT*SIGT(IS)
272:        if ( J2.le.NDS ) SIGT(ISU) = SIGT(ISU)
273:        &      + RHOT*SIGT(IS)
274:        SIGT(IJ) = SIGT(IJ) + RHOT*SIGT(IS)
275:      250 continue
276:      260 continue
277:      end if
278:      270 continue
279:
280:    end if
281: C
282: C ..... DETERMINE PROBABILITIES ( PL & DDX ) .....
283: C
284:      280 continue
285:      do 310 M = 1, NMED
286:        IK = (M-1)*NTG
287:        II1 = IK + ISTAR
288:        II2 = IK + ISCCOG
289:        II3 = IK + INUFIS
290:        I7 = IK + INUS
291: C
292: C      CALCULATE SIGS/SIGT (I2)
293: C      CALCULATE NU*SIGF/SIGT (I3)
294: C      CALCULATE GAMMA GENERATION PROBABILITY (I4)
295: C      CALCULATE SECONDARY PARTICLE GENERATION (WEIGHT MODIFIER) (I5)
296: C      CALCULATE SECONDARY PARTICLE GENERATION (PARTIAL PROBABILITY) (I6)
297: C
298:        I2 = IK + INABOG
299:        I3 = IK + IFPORG
300:        I4 = IGABOG + NGP*(M-1)
301:        I5 = 2*IK + IS2DP
302:        I6 = 3*IK + IS2PP
303: *VOCL LOOP,VECTOR
304:      do 290 J = 1, NGP
305:        if ( SIGT(II1+J).ne.0.0 ) then
306:          SIGT(I4+J) = SIGT(I4+J) /SIGT(II1+J)
307:        else
308:          SIGT(I4+J) = 0.
309:        end if
310:      290 continue
311:      do 300 J = 1, NTG
312:        ATSIG = SIGT(II1+J)
313:        if ( ATSIG.ne.0. ) then
314:          SIGT(I3+J) = SIGT(II3+J) /ATSIG
315:          SIGT(I2+J) = SIGT(II2+J) /ATSIG
316:        else
317:          write(IMG,7100) M, J, STFIX
318:          7100 format(1x,'!!! (JNPUT) TOTAL CROSS-SECTION OF MEDIUM ',I3,
319:            &      ' GROUP',I5,' EQUALS ZERO. FIXED TO ',E12.5)
320:          call cnterr( 'WARNING' )
321:          SIGT(I3+J) = 0.
322:          SIGT(I2+J) = 1.
323: Cccccccccc      SIGT(II1+J)=1.E-40
324:          SIGT(II1+J) = STFIX
325:        end if

```

```

326: C
327:      S2DP2 = SIGT(I2+J)
328:      if ( J.le.NGP.and.JPHOT*JNEUT.ne.0 ) S2DP2 = S2DP2
329:      &      + SIGT(I4+J)
330:      I51 = I5 + 2*(J-1)
331:      if ( JEIGN.eq.0.and.JFISS.ne.0 ) then
332:        SIGT(I51+1) = S2DP2 + SIGT(I3+J)
333:      else
334:        SIGT(I51+1) = S2DP2
335:      end if
336:      SIGT(I51+2) = S2DP2
337: C
338:      I61 = I6 + 3*(J-1)
339:      S2DPT = SIGT(I51+1)
340:      S2DPT2 = SIGT(I51+2)
341:      if ( S2DPT.ne.0. ) then
342:        SIGT(I61+1) = SIGT(I2+J) /S2DPT
343:        SIGT(I61+2) = S2DP2/S2DPT
344:      else
345:        SIGT(I61+1) = 1.0
346:        SIGT(I61+2) = 1.0
347:      end if
348:      if ( S2DPT2.ne.0. ) then
349:        SIGT(I61+3) = SIGT(I2+J) /S2DPT2
350:      else
351:        SIGT(I61+3) = 1.0
352:      end if
353: C
354:      if ( SIGT(II2+J).ne.0. ) then
355:        SIGT(I7+J) = SIGT(I7+J) /SIGT(II2+J)
356:      else
357:        SIGT(I7+J) = 0.
358:      end if
359:      300 continue
360:      310 continue
361: C
362:      if ( IDDX.ne.0 ) then
363:        ISPORT = ISPORG
364:        go to 600
365:      end if
366: C
367: C ..... CALCULATE DISCRETE ANGLES AND PROBABILITIES NEUTRONS (PL) ...
368: C
369:      NMOM = NCOEF - 1
370:      if ( NMOM/2+1.gt.NSCT ) NMOM = 2*NSCT - 1
371: C
372:      if ( NSCT.le.0 ) then
373:        ISPORT = ISPORG
374:        go to 600
375:      end if
376: C
377:      do 400 I = 1, NGP
378:        NDSK = NSIG(INNN+I)
379:        do 390 J = 1, NDSK
380:          IEND = 1
381:          do 380 N = 1, NMIX
382:            I9 = IRSG + N
383:            if ( IEND.ne.0 ) then
384:              do 320 L = 1, NMOM
385:                F(L) = 0.
386:              320 continue
387:              IEND = 0
388:            end if
389:            KM = NSIG(I9)
390:            KE = NSIG(I9+NMIX)

```

src/gmvp/jinput.f

```

391:          RHOT      = SIGT(I9+2*NMIX)
392: C
393:          if ( KE.le.0 ) IEND = 1
394: C
395:          KE         = IABS(KE)
396:          I11        = (KM-1)*(NDSNGP+NDSNGG) + IFSPG + NSIG(ING)
397:          &          + I) + J
398:          CO         = SIGT(I11)
399:          if ( CO.eq.0. ) then
400:            if ( IEND.eq.0 ) then
401:              go to 380
402:            else
403:              do 330 L = 1, NMOM
404:                F(L) = 0.
405:              continue
406:            end if
407:          else
408:            NKND      = IABSG + (KE-1)*ITEML + NSIG(ING+I) + J
409:            do 340 L = 1, NMOM
410:              if ( IDTF.le.0 ) then
411:                DIV    = 2.*L + 1.
412:              else
413:                DIV    = 1.
414:              end if
415:              I2       = NKND + NDSNGP*(L-1)
416:              F(L)     = F(L) + RHOT*SIGT(I2)/(CO*DIV)
417:            continue
418:            if ( IEND.eq.0 ) go to 380
419:          end if
420:
421:          I2         = (KM-1)*(NDSNGP+NDSNGG)*NSCT*3 + NSIG(ING
422:          &          + I)*NSCT*3 + (J-1)*NSCT*3 + 1 + IPRBNG
423:          do 350 L = 1, NMOM
424:            if ( F(L).ne.0. ) go to 360
425:          continue
426:          SIGT(I2)    = -1.
427:          go to 380
428: C
429:          360      call LEGEND
430:          call ANGLES( I, J, KM )
431:          NP1        = NANS + 1
432:          do 370 M = NP1 + 1, NSCT
433:            POINT(M) = POINT(NP1)
434:            WEIGHT(M) = 0.0
435:          370      continue
436: C
437:          call BMCMK0( WEIGHT(1), POINT(1), SIGT(I2),
438:          &          SIGT(I2+NSCT), NSCT )
439: C
440:          380      continue
441:          390      continue
442:          400      continue
443: C
444: C ..... CALCULATE DISCRETE ANGLES AND PROBABILITIES   GAMMAS (PL).....
445: C
446:          if ( NGG.gt.0.and.NSCT.gt.0 ) then
447:            do 480 I = 1, NGG
448:              NDSK   = NSIG(IGGG+I)
449:              do 470 J = 1, NDSK
450:                IEND = 1
451:                do 460 N = 1, NMIX
452:                  I9   = IRSG + N
453:                  if ( IEND.ne.0 ) then
454:                    do 410 L = 1, NMOM
455:                      F(L) = 0.

```

```

456:          410      continue
457:          IEND      = 0
458:        end if
459:        KM         = NSIG(I9)
460:        KE         = NSIG(I9+NMIX)
461:        RHOT       = SIGT(I9+2*NMIX)
462:        if ( KE.lt.0 ) IEND = 1
463:        KE         = IABS(KE)
464:        I11        = (KM-1)*(NDSNGP+NDSNGG) + IFSPG
465:        &          + NSIG(ING+I) + NDSNGP + J
466:        CO         = SIGT(I11)
467: C
468: C
469:        if ( CO.le.0.0 ) then
470:          if ( IEND.eq.0 ) go to 460
471:          do 420 L = 1, NMOM
472:            F(L) = 0.
473:          continue
474:        else
475:          NKND      = ITEMG + (KE-1)*ITEML + NSIG(ING+I) + J
476:          DIV       = 1.0
477:          do 430 L = 1, NMOM
478:            if ( IDTF.le.0 ) DIV = 2.*L + 1.
479:            I2       = NKND + NDSNGG*(L-1)
480:            F(L)     = F(L) + RHOT*SIGT(I2)/(CO*DIV)
481:          continue
482:          if ( IEND.eq.0 ) go to 460
483:        end if
484: C
485:          440      call LEGEND
486:          IN1       = I + NGP
487:          call ANGLES( IN1, J, KM )
488:          NP1       = NANS + 1
489:          I2=(KM-1)*NDSNGP*NSCT + NSIG(ING+I)*NSCT+(J-1)*NSCT
490:          I2        = (KM-1)*(NDSNGP+NDSNGG)*NSCT*3
491:          &          + (NSIG(ING
492:          &          + I)+NDSNGP)*NSCT*3 + (J-1)*NSCT*3 + 1 + IPRBNG
493:          do 450 M = NP1 + 1, NSCT
494:            POINT(M) = POINT(NP1)
495:            WEIGHT(M) = 0.
496:          450      continue
497: C
498:          call BMCMK0( WEIGHT(1), POINT(1), SIGT(I2),
499:          &          SIGT(I2+NSCT), NSCT )
500: C
501:          460      continue
502:          470      continue
503:          480      continue
504:        end if
505: C
506: C ISTAT IS FLAG FOR SAVING LEGENDRE MOMENTS FOR MEDIA 1 IN THE
507: C INPUT BUFFER ARRAY AT POSITION OF ELEM 1 ,MEDIA 2 IN POSITION
508: C OF ELEM 2, ETC., ISTAT.GT.0 ,SAVE MOMENTS
509: C THE COEFFICIENTS HAVE 2*L+1 INCLUDED AS WELL AS G TO GPRIME PROB
510: C ABILITY
511: C
512:          490      continue
513:          if ( INUSN.gt.0 ) then
514:            500      write(IMSG,7120) INUSN
515:          C          CALL ERROR
516:          call CNTERR( 'FATAL' )
517:          Cccccccc stop 999
518:        end if
519:          7120 format('1','XXX FIRST MOMENT WAS OUTSIDE RANGE,',
520:          &          ' SEE PRINTOUT ABOVE FOR GROUPS ** ','I5,

```

src/gmvp/jnput.f

```

521:      &      ' BAD MOMENTS WERE FOUND')
522:      510 continue
523: C
524:      if ( ISTAT.eq.0 ) then
525:          ISPORT = ISPORG
526:      else
527:          NMOM = NCOEF - 1
528: C **
529: C      INFPOG, ITOTSG AND ISPORT ARE REDEFINED
530: C **
531:          ISPORT = ISPORG + (NDSNGP+NDSNGG+NGP*NGG)*NMED
532:          INFPOG = NDSNGP + NDSNGG
533:          ITOTSG = INFPOG*NMED
534:          if ( IDTF.le.0 ) then
535:              do 520 L = 1, NMOM
536:                  C1(L) = 1.
537:          520      continue
538:          else
539:              do 530 L = 1, NMOM
540:                  C1(L) = 2.*L + 1.
541:          530      continue
542:          end if
543: C
544: C      ....SET TOTAL CROSS SECTION FOR GROUP 1 AND ALL MEDIA NEGATIVE TO BE
545: C      USED AS A FLAG IN MIXING COEFFS (PL)
546: C
547:          do 540 N = 1, NMED
548:              I11 = (N-1)*NTG + ISTART + 1
549:              SIGT(I11) = -SIGT(I11)
550:          540      continue
551: C
552:          do 580 N = 1, NMIX
553:              I8 = IRSG + N
554:              KM = NSIG(I8)
555:              I7 = (KM-1)*NTG + ISTART + 1
556:              if ( SIGT(I7).le.0 ) then
557:                  KF = 0
558:                  SIGT(I7) = -SIGT(I7)
559:              else
560:                  KF = 1
561:              end if
562:              KE = IABS(NSIG(I8+NMIX))
563:              RHOT = SIGT(I8+2*NMIX)
564: C
565:          do 570 L = 1, NMOM
566:              I2 = ISPORT + (L-1)*ITOTSG + (KM-1)*INFPOG
567:              do 560 I = 1, NTG
568:                  if ( I.le.NGP ) then
569: C ..... PRIMARY .....
570:                      I3 = NSIG(INGS+I)
571:                      I11 = (KM-1)*(NDSNGP+NDSNGG) + IFSPOG + I3
572:                      NKND = IABSOG + (KE-1)*ITEML + I3 + NDSNGG*(L-1)
573:                      NKP = I2 + I3
574:                  else
575: C ..... SECONDARY .....
576:                      I3 = NSIG(ING+I-NGP) + NDSNGP
577:                      I11 = (KM-1)*(NDSNGP+NDSNGG) + IFSPOG + I3
578:                      NKND = ITEMG + (KE-1)*ITEML + I3 + NDSNGG*(L-1)
579:                      NKP = I2 + I3
580:                  end if
581:                  NDSK = NSIG(INNN+I)
582:                  do 550 J = 1, NDSK
583:                      CO = SIGT(I11+J)
584:                      if ( CO.ne.0.0 ) then
585:                          if ( KF.le.0 ) then

```

```

586:                      SIGT(NKP+J) = RHOT*SIGT(NKND+J) /CO*C1(L)
587:                  else
588:                      SIGT(NKP+J) = SIGT(NKP+J) + RHOT*SIGT(NKND
589:                      +J) /CO*C1(L)
590:                  &
591:                  end if
592:                  else
593:                      if ( KF.le.0 ) SIGT(NKP+J) = 0.
594:                  end if
595:          550      continue
596:          560      continue
597:          570      continue
598: C          ISPORT +(L-1)*ITOTSG +(N-1)*INFPOG + NSIG(INGS+I)
599: C          FOR MEDIUM N, COEFF L, GROUP I
600:          580      continue
601:              do 590 N = 1, NMED
602:                  I11 = (N-1)*NTG + ISTART + 1
603:                  if ( SIGT(I11).le.0. ) SIGT(I11) = -SIGT(I11)
604:          590      continue
605: C
606:          end if
607:          600 continue
608: C
609: C      IN THE FOLLOWING LOOPS
610: C      I INDEX FOR MEDIA
611: C      J,J3 INDEX FOR GROUPS
612: C      J2 INDEX FOR DOWNSCATTER
613: C
614:          do 810 I = 1, NMED
615: C
616: C      .... FORM NORMALIZED DISTRIBUTION FOR NEUTRON DOWNSCATTER (PL) ..
617:          if ( IDDX.eq.0 ) then
618:              do 630 J = 1, NGP
619:                  I6 = IFSPOG + NSIG(INGS+J) + (I-1)*(NDSNGP+NDSNGG)
620:                  NDSK = NSIG(INNN+J)
621:                  I8 = J + ISCCOG + (I-1)*NTG
622:                  do 610 J2 = 1, NDSK
623:                      I7 = I6 + J2
624:                      if ( SIGT(I8).eq.0. ) then
625:                          if ( J2.le.1 ) then
626:                              SIGT(I7) = 1.
627:                              write(IMG,7140) I, J
628:                              call cnterr( 'WARNING' )
629:                          else
630:                              SIGT(I7) = 0.
631:                          end if
632:                      else
633:                          SIGT(I7) = SIGT(I7) /SIGT(I8)
634:                      end if
635:          610      continue
636:                  if ( ISTAT.gt.0 ) then
637:                      do 620 J2 = 1, NDSK
638:                          SIGT(IABSOG+J2) = SIGT(I6+J2)
639:                      620      continue
640:                      I6 = IABSOG
641:                  end if
642:                  I1 = IFSPOG + (NDSNGP+NDSNGG)*3*(I-1) + NSIG(INGS
643:                  +J)*3
644:                  &
645:                  I2 = I1 + NDSK
646:                  call BMCMK1( SIGT(I6+1), SIGT(I1+1), SIGT(I2+1), NDSK )
647:          630      continue
648: C
649: C      .... FORM NORMALIZED DISTRIBUTION FOR GAMMA DOWNSCATTER (PL)
650: C
651:          if ( NTG.gt.NGP ) then

```

src/gmvp/jnput.f

```

651:      do 660 J = 1, NGG
652:        NDSK = NSIG(IGGG+J)
653:        I6 = IFSPOG + NSIG(INGG+J) + NDSNGP + (I-1)*
654:      &      (NDSNGP+NDSNGG)
655:        I8 = J + NGP + ISCCOG + (I-1)*NTG
656:        do 640 J2 = 1, NDSK
657:          I7 = I6 + J2
658:          if ( SIGT(I8).eq.0. ) then
659:            if ( J2.le.1 ) then
660:              SIGT(I7) = 1.
661:              write(1MSG,7160) I, J
662:              call CNTERR( 'WARNING' )
663:            else
664:              SIGT(I7) = 0.
665:            end if
666:          else
667:            SIGT(I7) = SIGT(I7) /SIGT(I8)
668:          end if
669:        640 continue
670:        if ( ISTAT.gt.0 ) then
671:          do 650 J2 = 1, NDSK
672:            SIGT(IABSOG+J2) = SIGT(I6+J2)
673:          650 continue
674:          I6 = IABSOG
675:        end if
676:        I1 = IFSPOG + (NDSNGP+NDSNGG)*3*(I-1)
677:      &      + (NSIG(INGG+J)+NDSNGP)*3
678:        I2 = I1 + NDSK
679:      C      call BMCMK1( SIGT(I6+1), SIGT(I1+1), SIGT(I2+1), NDSK
680:      &      )
681:      C
682:      C 660 continue
683:      C
684:      C ..... FORM NORMALIZED DISTRIBUTION FOR GAMMA PRODUCTION (PL) ....
685:      C
686:      C      do 690 J = 1, NGP
687:        NDSK = NGG
688:        I6 = IFNGP + NGG*(J-1) + (I-1)*NGP*NGG
689:        I8 = J + ISTART + (I-1)*NTG
690:        I7 = J + IGABOG + (I-1)*NGP
691:        ATSIG = SIGT(I8) + SIGT(I7)
692:        ATSIG1 = SIGT(I7)
693:        do 670 J2 = 1, NDSK
694:          I7 = I6 + J2
695:          if ( ATSIG.eq.0. .or. ATSIG1.eq.0. ) then
696:            if ( J2.le.1 ) then
697:              SIGT(I7) = 1.
698:              write(1MSG,7180) I, J
699:              call CNTERR( 'WARNING' )
700:            else
701:              SIGT(I7) = 0.
702:            end if
703:          else
704:            SIGT(I7) = SIGT(I7) /SIGT(I8)
705:          end if
706:        670 continue
707:        if ( ISTAT.gt.0 ) then
708:          do 680 J2 = 1, NDSK
709:            SIGT(IABSOG+J2) = SIGT(I6+J2)
710:          680 continue
711:          I6 = IABSOG
712:        end if
713:        I1 = IFNGP + NGP*NGG*3*(I-1) + NGG*3*(J-1)
714:        I2 = I1 + NDSK
715:

```

```

716:      call BMCMK1( SIGT(I6+1), SIGT(I1+1), SIGT(I2+1), NDSK
717:      &      )
718:      690 continue
719:    end if
720:  end if
721: C
722: 7140 format(1x,'!!! THE SCATTERING CROSS-SECTION OF MEDIUM',I3,
723:  &      ' NEUTRON GROUP',I5,' EQUALS ZERO.')
724: 7160 format(1x,'!!! THE SCATTERING CROSS-SECTION OF MEDIUM',I3,
725:  &      ' GAMMA GROUP',I5,' EQUALS ZERO.')
726: 7180 format(1x,'!!! THE GAMMA PRODUCTION CROSS SECTION OF MEDIUM',
727:  &      I3,' NEUTRON GROUP',I5,' EQUALS ZERO.')
728: C
729: C..... PRINT THE FINAL CROSS SECTION DATA FOR ISOTROPIC SCATTERING
730: C (PL & DDX)
731: C if ( ISTR.le.0 ) go to 730
732: C
733: C go to 810
734: C
735: C write(I0,7200) I
736: 7200 format('1',20X,' CROSS SECTIONS FOR MEDIA ',I5)
737: C write(I0,7220)
738: 7220 format(1X,'GROUP SIGT SIGST PNUP PNABS GAMGEN ',
739:  &      'NU*FIS DOWNSCATTER PROBABILITY')
740: C
741: C do 700 J = 1, NGP
742:   I11 = J + (I-1)*NTG
743:   I2 = I11 + ISCCOG
744:   I3 = I11 + INABOG
745:   I4 = IGABOG + J + (I-1)*NGP
746:   I5 = I11 + IFPORG
747:   I9 = I11 + INUS
748:   I6 = 1 + IFSPOG + (I-1)*(NDSNGP+NDSNGG) + NSIG(INGG+J)
749:   I8 = I11 + ISTART
750:   NDSK = NSIG(INNN+J)
751:   if ( ISTAT.le.0 .or. IDDX.ne.0 ) NDSK = 0
752:   I7 = I6 + NDSK - 1
753:   write(I0,7240) J, SIGT(I8), SIGT(I2), SIGT(I9), SIGT(I3),
754:   &      SIGT(I4), SIGT(I5), (SIGT(J3),J3=I6,I7)
755: 700 continue
756: C
757: 7240 format(1X,I4,1P2E10.3,0P4F7.4,8F8.4/(54X,8F8.4))
758: C
759: C if ( NTG.gt.NGP ) then
760:   do 710 J = NGP + 1, NTG
761:     I11 = J + (I-1)*NTG
762:     I2 = I11 + ISCCOG
763:     I3 = I11 + INABOG
764:     I6 = 1 + IFSPOG + (I-1)*(NDSNGP+NDSNGG)
765:     &      + NSIG(INGG+J) + NDSNGP
766:     I8 = I11 + ISTART
767:     NDSK = NSIG(INNN+J)
768:     if ( ISTAT.le.0 .or. IDDX.ne.0 ) NDSK = 0
769:     I7 = I6 + NDSK - 1
770:     I9 = I11 + INUS
771:     S9G = 0.
772:     I5 = I11 + IFPORG
773:     write(I0,7240) J, SIGT(I8), SIGT(I2), SIGT(I9), SIGT(I3),
774:     &      S9G, SIGT(I5), (SIGT(J3),J3=I6,I7)
775:   710 continue
776: C
777: C ..... PRINT NEUTRON TO GAMMA TRANSFER PROBABILITY .....
778: C
779: C write(I0,7260) I
780: C

```

src/gmvp/jnput.f

```

781:         if ( ISTAT.gt.0 ) then
782:             do 720 J = 1, NGP
783:                 N6 = IFNGP + (J-1)*NGG + 1 + NGP*NGG*(I-1)
784:                 N8 = IGABOG + J + NGP*(I-1)
785:                 N7 = N6 + NGG - 1
786:                 write(I0,7280) J, SIGT(N8), (SIGT(N9),N9=N6,N7)
787:             720 continue
788:         end if
789: C
790:     end if
791: C
792: 7260 format('1',10X,' NEUTRON TO GAMMA TRANSFERS FOR MEDIA',I4/1X,
793:           ' NEUT GROUP',6X,'GAMGEN',7X,'TRANSFER PROBABILITIES'/)
794: 7280 format(1X,4X,I5,5X,1P10.3,3X,0P10F8.4/(28X,10F8.4))
795: C
796: 730 if ( ISTR.le.0 ) go to 810
797: C
798:     if ( NSCT.gt.0 ) then
799:         if ( IPRIN.gt.0 ) then
800: C
801: C .....PRINT ANGLES AND PROBABILITIES .....
802: C
803:             write(I0,7300) I
804: 7300 format('1',' SCATTERING PROBABILITIES AND ANGLES ',
805:           & 'FOR MEDIA NUMBER ',I3)
806:             write(I0,7320) (' ',J=1,NSCT)
807: 7320 format('      GO TO GR',7(A1,' PROB  ANGLE ':))
808:             do 760 J = 1, NGP
809:                 NDSK = NSIG(INNN+J)
810:                 do 750 J1 = 1, NDSK
811:                     I11 = J + J1 - 1 - NUS
812: Ccccccccccccccccccccccccc IF(I11)610,610,600
813:                     if ( I11.gt.0 ) then
814:                         if ( IDDX.eq.0 ) then
815:                             I2 = (I-1)*NDSNGP*NSCT
816:                             & + NSCT*(NSIG(ING+J)-1+J1) + IPRBGG
817:                             J5 = 0
818:                         else if ( ISTAT.gt.0 ) then
819:                             I2 = (I-1)*(NDSNGP+NDSNGG)*NSCT
820:                             & + NSCT*(NSIG(ING+J)-1+J1) + ISPORT
821:                             J5 = 2*NSCT
822:                             do 740 J3 = 1, J5, 2
823:                                 J4 = J3/2 + 1
824:                                 TEMP(J3) = SIGT(I2+J4)
825:                                 TEMP(J3+1) = SIGT(IANG+J4)
826:                             740 continue
827:                         end if
828:                         write(I0,7340) J, I11, (TEMP(J6),J6=1,J5)
829:                     end if
830:                 750 continue
831:             760 continue
832: C
833: 7340 format(2I6,12F8.4/(12X,12F8.4))
834: C
835: C
836:             if ( NTG.le.NGP ) go to 810
837: C
838:             if ( NSCT.le.0 ) go to 810
839: C
840:             do 800 J = NGP + 1, NTG
841:                 NDSK = NSIG(INNN+J)
842:                 do 790 J1 = 1, NDSK
843:                     I11 = J + J1 - 1 - NUS
844:                     if ( I11.le.0 ) go to 790
845:                     if ( IDDX.eq.0 ) then

```

```

846:                                     I2 = (I-1)*NDSNGG*NSCT
847:                                     & + NSCT*(NSIG(ING+J)-1+J1) + IPRBGG
848:                                     J5 = 0
849:             else
850:                 if ( ISTAT.gt.0 ) then
851:                     I2 = (I-1)*(NDSNGP+NDSNGG)*NSCT
852:                     & + NSCT*(NSIG(ING+J)-1+J1+NDSNGP) + ISPORT
853:                     &
854:                     J5 = 2*NSCT
855:                     do 780 J3 = 1, J5, 2
856:                         J4 = J3/2 + 1
857:                         TEMP(J3) = SIGT(I2+J4)
858:                         TEMP(J3+1) = SIGT(IANG+J4)
859:                     780 continue
860:                 end if
861:             end if
862:             write(I0,7340) J, I11, (TEMP(J6),J6=1,J5)
863:             790 continue
864:             800 continue
865:             end if
866:             end if
867:             810 continue
868: C
869:             return
870:         end

```

src/gmvp/keff0.f

```

1:      subroutine KEFF0( NBATCH,IPBT,  NSKIP, JPR,   XSOC,  XKEF,
2:      &                  XKAV1, XKER1, XKAV2, XKER2, XKAV3, XKER3,
3:      &                  XKEF1, XSOCB, XKB )
4: C=====
5: C  PURPOSE: Statistical analysis of k-effective (called after some
6: C           batches for monitoring).
7: C           Calculate average form first batch to batch IPBT
8: C           and M.L.E.'s from batch NSKIP+1 to batch IPBT.
9: C  CALLED IN: TALSUM
10: C  CALL      : MTXINV
11: C-----
12: C  arguments (i=input, o=output, w=work)
13: C
14: C  i NBATCH : number of batches (probably calculated until now)
15: C  i IPBT   : batch # for which tally analysis is performed.
16: C  i NSKIP  : number of skipped batch for k-eff estimation
17: C  i JPR    : printout if non zero
18: C  i XSOC(I): neutron source weight sum until batch I
19: C  i XKEF(I): neutron production (fission) sum until batch I
20: C  o XKAV1(4),XKER1(4) : averaged K-eff and % error of 6 way
21: C                       from batch 1 to IPBT.
22: C                       ( production(track/collision/analog)
23: C                       and balance(track/collision/analog) )
24: C  o XKAV2(4),XKER2(4) : averaged K-eff and % error of 6 way
25: C                       from batch NSKIP+1 to IPBT.
26: C                       ( production(track/collision/analog)
27: C                       and balance(track/collision/analog) )
28: C  o XKAV3(5),XKER3(5) : averaged M.L.E. K-eff and % error of 6 way
29: C                       from batch NSKIP+1 to IPBT.
30: C                       ( track/collision/analog
31: C                       and production/n-balance/all )
32: C  w XKEF1(5,NBATCH),XSOCB(NBATCH),XKB(5,NBATCH): working array
33: C=====
34:      implicit real*8(A-H,O-Z)
35: C
36:      include '../shared/INC/_IOUNIT'
37:      include 'INC/_IOUNIT2'
38: C
39:      real*8 XSOC(NBATCH)
40:      real*8 XKEF(5,NBATCH)
41: C
42: C  ... until IPBT'th batch
43:      real*8 XKAV1(4)
44:      real*8 XKER1(4)
45: C  ... after NSKIP'th batch
46:      real*8 XKAV2(4)
47:      real*8 XKER2(4)
48: C  ... M.L.E. after NSKIP'th batch
49:      real*8 XKAV3(5)
50:      real*8 XKER3(5)
51: C
52: C  ... working array
53: C
54:      real*8 XKEF1(5,NBATCH)
55:      real*8 XSOCB(NBATCH)
56:      real*8 XKB(5,NBATCH)
57: C
58: C  ... criterion to reject too much correlated combination of
59: C     estimation in maximum likelihood estimation.
60: C
61:      parameter( DECORR = 1.0D-5 )
62: C
63: C  ... local data
64: C
65:      real*8 X1, X2, X3, X4, X5, X6, S1

```

```

66:      character*32 TAG
67:      character*48 ESTCMB
68: C
69:      real*8 COVXK(4,4)
70:      real*8 CORR(4,4)
71:      real*8 XKSD(4)
72:      integer IWK1(4)
73: C
74:      integer JJCC(4)
75:      integer JJCC0(4)
76: C
77:      real*8 AVG(16), SIGA(16), SIGR(16)
78:      character*48 ESTKEF(16)
79: C
80: C  ... mixing factor of M.L.E.'s ....
81: C
82: C      XKAV1(J) = Sum[I] ( XKB(I)*FMIX(I,J) )
83: C
84:      real*8 FMIX(4,4)
85: C
86: C-----
87: C
88: C-----
89: C*** STATISTICAL ANALYSIS OF K-EFFECTIVE
90: C-----
91: C
92:      XSOCB(1) = XSOC(1)
93: C
94:      XKEF1(5,1) = XKEF(5,1)
95:      XKEF1(1,1) = XKEF(1,1)
96:      XKEF1(2,1) = XKEF(2,1)
97:      XKEF1(3,1) = XKEF(3,1) + XKEF1(5,1)
98:      XKEF1(4,1) = XKEF(4,1) + XKEF1(5,1)
99: C
100:      XKB(1,1) = XKEF1(1,1) /XSOC(1)
101:      XKB(2,1) = XKEF1(2,1) /XSOC(1)
102:      XKB(3,1) = XKEF1(1,1) /XKEF1(3,1)
103:      XKB(4,1) = XKEF1(2,1) /XKEF1(4,1)
104: CC
105: CC XSOCB : SOURCE (EACH BATCH)
106: CC XKB : ESTIMATOR OF KEFF (EACH BATCH)
107: CC XKAV1 : CUMULATIVE ESTIMATOR OF KEFF TO THIS BATCH
108: CC 1: TRACK LENGTH ESTIMATOR (PRODUCTION RATE)
109: CC 2: COLLISION ESTIMATOR (PRODUCTION RATE)
110: CC 3: ANALOG ESTIMATOR (PRODUCTION RATE)
111: CC 4: TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
112: CC 5: COLLISION ESTIMATOR (NEUTRON BALANCE)
113: CC 6: ANALOG ESTIMATOR (NEUTRON BALANCE)
114: CC
115:      do 100 I = 2, IPBT
116:
117:          XSOCB(I) = XSOC(I) - XSOC(I-1)
118:
119:          XKEF1(5,I) = XKEF(5,I)
120:          XKEF1(1,I) = XKEF(1,I)
121:          XKEF1(2,I) = XKEF(2,I)
122:          XKEF1(3,I) = XKEF(3,I) + XKEF1(5,I)
123:          XKEF1(4,I) = XKEF(4,I) + XKEF1(5,I)
124:
125:          XKB(1,I) = (XKEF1(1,I)-XKEF1(1,I-1)) /XSOCB(I)
126:          XKB(2,I) = (XKEF1(2,I)-XKEF1(2,I-1)) /XSOCB(I)
127:          XKB(3,I) = (XKEF1(1,I)-XKEF1(1,I-1)) /
128:      & (XKEF1(3,I)-XKEF1(3,I-1))
129:          XKB(4,I) = (XKEF1(2,I)-XKEF1(2,I-1)) /
130:      & (XKEF1(4,I)-XKEF1(4,I-1))

```

src/gmvp/keff0.f

```

331:      continue
332:
333:          XKAV1(1)    = XKEF1(1,IPBT) /XSOC(IPBT)
334:          XKAV1(2)    = XKEF1(2,IPBT) /XSOC(IPBT)
335: c ... source weighted "balance" estimators
336:          XKAV1(3)    = 0.0D0
337:          XKAV1(4)    = 0.0D0
338:          do 110 J = 1, IPBT
339:              XKAV1(3) = XKAV1(3) + XKB(3,J)*XSOCB(J)
340:              XKAV1(4) = XKAV1(4) + XKB(4,J)*XSOCB(J)
341:          110 continue
342:          XKAV1(3)    = XKAV1(3) /XSOC(IPBT)
343:          XKAV1(4)    = XKAV1(4) /XSOC(IPBT)
344: C
345:          do 130 L = 1, 4
346:              XKER1(L) = 0.0D0
347:              if ( IPBT.gt.1 ) then
348:                  do 120 J = 1, IPBT
349:                      XKER1(L) = XKER1(L) + (XKB(L,J)-XKAV1(L))*
350:                  120 continue
351:                      XKER1(L) = SQRT(XKER1(L)/dble(IPBT)/(IPBT-1))
352:                  &           / XKAV1(L)*100.0D0
353:                  end if
354:              130 continue
355: CCC
356:              if ( JPR.ne.0 ) then
357:                  write(IOT,7001)
358:                  format(1x,'+',5('+++++',10('-----')) )
359:                  write(IOT,7000) IPBT
360:                  write(IOT,7040) (XKAV1(L),XKER1(L),L=1,4)
361:                  7000 format(1x,'+ * K-EFFECTIVE AVERAGE TO BATCH ',IS
362:                  &         '/1X','+',
363:                  &         ' TRACK LENGTH(PROD.) ',
364:                  &         ' COLLISION(PROD.)   ',
365:                  &         ' TRACK LENGTH(BAL.)   ',
366:                  &         ' COLLISION(BAL.)     ')
367:              end if
368: C
369:              IS      = NSKIP + 1
370: C
371:              if ( IPBT.gt.IS+3 ) then
372: CCC
373: CCC XSOC(I) : SOURCE. CUMULATIVE TO I'TH BATCH
374: CCC
375: CCC SIMILAR TO XKEF(4,I) KEFF (PRODUCTION & LOSS)
376: CCC
377:              XSS      = XSOC(IPBT)
378:              if ( NSKIP.gt.0 ) then
379:                  XSS    = XSOC(IPBT) - XSOC(IS-1)
380:                  XKAV2(1) = (XKEF1(1,IPBT)-XKEF1(1,IS-1)) /XSS
381:                  XKAV2(2) = (XKEF1(2,IPBT)-XKEF1(2,IS-1)) /XSS
382:              else
383:                  XSS      = XSOC(IPBT)
384:                  XKAV2(1) = XKEF1(1,IPBT) /XSS
385:                  XKAV2(2) = XKEF1(2,IPBT) /XSS
386:              end if
387: C
388: c ... source weighted "balance" estimators
389: c
390:          XKAV2(3)    = 0.0D0
391:          XKAV2(4)    = 0.0D0
392:          do 140 J = IS, IPBT
393:              XKAV2(3) = XKAV2(3) + XKB(3,J)*XSOCB(J)
394:              XKAV2(4) = XKAV2(4) + XKB(4,J)*XSOCB(J)
395:          140 continue

```

```

196:      XKAV2(3)      = XKAV2(3) / XSS
197:      XKAV2(4)      = XKAV2(4) / XSS
198: C
199: C      ..... COVXK : COVARIANCE MATRIX
200: C
201:      do 160 K = 1, 4
202:      do 150 L = 1, 4
203:          COVXK(L,K) = 0.0
204:      150      continue
205:      160      continue
206: C
207:      do 190 K = 1, 4
208:      do 180 L = 1, 4
209:          if ( L.ge.K ) then
210:              do 170 J = IS, IPBT
211:                  X1 = XSOCB(J) / (XSS-XSOCB(J))
212:                  COVXK(K,L) = COVXK(K,L) + X1*(XKB(K,J)-XKAV2(K))*
213:                      & (XKB(L,J)-XKAV2(L))
214:              170      continue
215:              else
216:                  COVXK(K,L) = COVXK(L,K)
217:              end if
218:          180      continue
219:      190      continue
220: C
221:      XX      = 1.D0/(NBATCH-IS+1)
222:      do 210 K = 1, 4
223:      do 200 L = 1, 4
224:          COVXK(K,L) = COVXK(K,L)*XX
225:      200      continue
226:      210      continue
227: C
228: C..... ESTIMATION OF KEFF BY THE PRINCIPLE OF MAXIMUM LIKELYHOOD
229: C      XKAV3 : KEFF
230: C      XKERR : STANDARD DEVIATION OF COMBINED ESTIMATION (%)
231: C      1 : TRACK LENGTH (PRODUCTION + BALANCE)
232: C      2 : COLLISION (PRODUCTION + BALANCE)
233: C      3 : PRODUCTION (TRK+COL+ANALOG)
234: C      4 : BALANCE (TRK+COL+ANALOG)
235: C      5 : ALL ESTIMATORS
236: C..... ESTIMATION OF KEFF BY EACH ESTIMATOR
237: C      XKEF : KEFF
238: C      XKSD : STANDARD DEVIATION (%)
239: C      1 : TRACK LENGTH (PRODUCTION)
240: C      2 : COLLISION (PRODUCTION)
241: C      3 : TRACK LENGTH (BALANCE)
242: C      4 : COLLISION (BALANCE)
243: C
244:      do 220 J = 1, 4
245:          XKER2(J) = SQRT(ABS(COVXK(J,J))) / XKAV2(J)*100.
246:      220      continue
247: C
248:      if ( JPR.ne.0 ) then
249:          write(IOW,7020) IS, IPBT
250:          write(IOW,7040) (XKAV2(J),XKER2(J),J=1,4)
251:          7020      format(1x,'+' /1x,'+ * K-EFFECTIVE AVERAGE ',
252:              & 'FROM BATCH ',I4,' TO ',I5,' * '
253:              & /1x,'+',
254:              & ' TRACK LENGTH(PROD.) ',
255:              & ' COLLISION(PROD.) ',
256:              & ' TRACK LENGTH(BAL.) ',
257:              & ' COLLISION(BAL.) ')
258:          7040      format(1x,'+',5(1P,E13.5,'(' ,0P,F5.3,'%')':))
259:          end if
260: C

```


src/gmvp/keff0.f

```

261: C.... clear mixing factor of estimators (used in "real variance"
262: C      calculation.
263: C
264:       do 240 J = 1, 4
265:         do 230 K = 1, 4
266:           FMIX(K,J) = 0.0D0
267:         230 continue
268:       240 continue
269: C
270: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 2 ESTIMATORS
271: C
272:       JCORR1 = 0
273:       do 280 J = 1, 4
274:         if ( J.eq.1 ) then
275:           K = 1
276:           L = 2
277: CCCCCCCC ESTCMB = 'production (track & collision)'
278:         else if ( J.eq.2 ) then
279:           K = 3
280:           L = 4
281: CCCCCCCC ESTCMB = 'balance (track & collision)'
282:         else if ( J.eq.3 ) then
283:           K = 1
284:           L = 3
285: CCCCCCCC ESTCMB = 'track length (production & balance)'
286:         else if ( J.eq.4 ) then
287:           K = 2
288:           L = 4
289: CCCCCCCC ESTCMB = 'collision (production & balance)'
290:         end if
291:
292:         CORR1 = COVXK(K,L) /SQRT(COVXK(K,K)*COVXK(L,L))
293:
294:         if ( ABS(CORR1-1.0D0).lt.DECORR ) then
295:           if ( JCORR1.eq.0 ) then
296:             JCORR1 = 1
297: C           write(IOT,(' '1'/'lx,a,a/lx,a,E12.5/'))
298: C           &           '!!! Too strongly correlated estimator pairs '
299: C           &           ', 'are found in M.L.E. calculation.',
300: C           &           '(criterion : abs(correlation coeff.-1) < ',
301: C           &           DECORR
302: C           &           write(IOT,('/a/'))
303: C           &           ' Dummy values of -1.0 will be output
304: C           &
305: C           end if
306: C           write(IOT,('1x,a,i4,a,a,a,E14.7'))
307: C           &           ' Cumulative after batch ', I, ' type : ',
308: C           &           ESTCMB, ' : correlation =', CORR1
309: C           &           XKAV3(J) = -1.0D0
310: C           &           XKER3(J) = 0.0D0
311: C         else
312: C           X1 = COVXK(K,K) - COVXK(K,L)
313: C           X2 = COVXK(L,L) - COVXK(K,L)
314: C           XKAV3(J) = (X1*XKAV2(L)+X2*XKAV2(K)) / (X1+X2)
315: C           XKER3(J) =
316: C           &           (COVXK(K,K)*COVXK(L,L)-COVXK(K,L)*COVXK(K,L)) /
317: C           &           (X1+X2)
318: C           XKER3(J) = SQRT(ABS(XKER3(J))) /XKAV3(J)*100.0
319: C
320: C           FMIX(L,J) = X1/(X1+X2)
321: C           FMIX(K,J) = X2/(X1+X2)
322: C
323: C           end if
324: C           270 continue
325: C       280 continue

```

```

326: C
327: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF ALL 4 ESTIMATORS
328: C      GET AN INVERSE MATRIX OF COVXK INTO COVXK
329: C
330:           XKER3(5) = 0.0
331:           XKAV3(5) = -1.0
332: C
333: C ... omit one estimator from too strongly correlated pair ...
334: C      ( non-diagonal elements of COVXK for omitted estimator is
335: C      cleared to 0.0 )
336: C
337:       do 320 L = 4, 1, -1
338:         JJCC(L) = 0
339:         do 300 K = 1, L - 1
340:
341:           CORR1 = COVXK(K,L) /SQRT(COVXK(K,K)*COVXK(L,L))
342:
343:           if ( ABS(CORR1-1.0D0).lt.DECORR ) then
344:             JJCC(L) = 1
345:             COVXK(L,L) = 1.0D0
346:             do 290 KK = 1, 4
347:               if ( KK.ne.L ) then
348:                 COVXK(KK,L) = 0.0D0
349:                 COVXK(L,KK) = 0.0D0
350:               end if
351:             290 continue
352:             go to 310
353:           end if
354:         300 continue
355:
356:       310 continue
357:       320 continue
358:
359:       do 330 L = 1, 4
360:         JJCC0(L) = JJCC(L)
361:       330 continue
362: C
363: C ... invert covariance matrix ...
364: C
365: C       call MTXINV( COVXK(1,1), 4, IWK1, CORR(1,1), 1.0D-16, ICON )
366: C
367: C       if ( ICON.ne.0 ) then
368: C         write(IOT,('1x,a,a'))
369: C         &           '!!! CANNOT INVERT COVARIANCE MATRIX ',
370: C         &           ' --> MAX.LIKELYHOOD ESTIMATOR PRINTED AS -1'
371: C         &
372: C
373: C         XKER3(5) = 0.0
374: C         XKAV3(5) = -1.0
375: C       else
376: C         XKER3(5) = 0.0
377: C         XKAV3(5) = 0.0
378: C       end if
379: C
380: C       if ( XKAV3(5).ne.-1.0 ) then
381: C         do 350 K = 1, 4
382: C           do 340 L = 1, 4
383: C             if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
384: C               XKER3(5) = XKER3(5) + COVXK(K,L)
385: C               XKAV3(5) = XKAV3(5) + COVXK(K,L)*XKAV2(L)
386: C             end if
387: C           340 continue
388: C         350 continue
389: C       330 continue

```

src/gmvp/keff0.f

```
391:      350      continue
392:
393:          XKER3(5)      = 1./XKER3(5)
394:          XKAV3(5)      = XKAV3(5)*XKER3(5)
395:          XKER3(5)      = SQRT(ABS(XKER3(5))) /XKAV3(5)*100.
396:      end if
397:
398: C
399:      if ( JPR.ne.0 ) then
400:          write(IOW,7060) IS, IPBT
401:          write(IOT,7040) (XKAV3(J),XKER3(J),J=1,5)
402: C
403: 7060      format(1x,'+'
404: &          /1x,'+ * M.L.E. K-EFFECTIVE BY EACH ESTIMATOR ',
405: &          'FROM BATCH ',I4,' TO ',I5,' */1X,'+',
406: &          ' PRODUCTION(MLE) ',
407: &          ' BALANCE(MLE) ',
408: &          ' TRACK LENGTH(MLE) ',
409: &          ' COLLISION(MLE) ',
410: &          ' ALL(MLE) ')
411:      end if
412: C
413:      end if
414: C
415:      if ( JPR.ne.0 ) write(IOT,7001)
416: C
417:      return
418:      end
```

src/gmvp/kpinit.f

```
1:      subroutine KPINIT
2: C=====
3: C purpose: initialize particle symbol strings for GMVP
4: C called in: MAIN
5: C calls : none
6: C-----
7: C initialization in this routine must synchronize with contents of
8: C included files _KPIDS and _KPSYMS
9: C=====
10:      include '../shared/INC/_IOUNIT'
11: C
12:      include 'INC/_KPIDS'
13:      include 'INC/_KPSYMS'
14: C
15:      include 'INC/_FLAGS'
16: C
17: C ... number of particles possible to treat in this code
18: C (this should be maximum of possible KP* )
19: C
20: C parameter ( KPLIM = ? )
21: C
22: C
23: C ... particle symbol string and their alias (or short form)
24: C
25: C character*16 KPSYM
26: C common /CKPSYM/ KPSYM(2,KPLIM)
27: C
28: C
29: C-----
30: C
31:      if ( MKPAR.lt.KPLIM ) then
32:          write(IMG,*) 'XXX(KPINIT) Array size of JKPAR(*) must be ,
33:          & 'greater than KPLIM. Program error. Check your source!!!'
34:          stop 666
35:      end if
36: C
37: C-----
38: C === possible initialization
39: C
40: C
41: C === possible initialization
42: C
43: C ... neutron and photon
44:      KPSYM(1,KPNEUT) = 'NEUTRON'
45:      KPSYM(2,KPNEUT) = 'N'
46:      KPSYM(1,KPPHOT) = 'PHOTON'
47:      KPSYM(2,KPPHOT) = 'P'
48: C ... electron and proton
49:      KPSYM(1,KPELEC) = 'ELECTRON'
50:      KPSYM(2,KPELEC) = 'EL'
51:      KPSYM(1,KPPROT) = 'PROTON'
52:      KPSYM(2,KPPROT) = 'PR'
53: C ... pi mesons (pi-0 pi+ pi- )
54:      KPSYM(1,KPPIO) = 'PIO'
55:      KPSYM(2,KPPIO) = 'PIO'
56:      KPSYM(1,KPPIP) = 'PI-PLUS'
57:      KPSYM(2,KPPIP) = 'PIP'
58:      KPSYM(1,KPPIM) = 'PI-MINUS'
59:      KPSYM(2,KPPIM) = 'PIM'
60: C ... mu mesons (mu+ mu- )
61:      KPSYM(1,KPMUP) = 'MU-PLUS'
62:      KPSYM(2,KPMUP) = 'MUP'
63:      KPSYM(1,KPMUM) = 'MU-MINUS'
64:      KPSYM(2,KPMUM) = 'MUM'
65: C
66:      return
67:      end
```

src/gmvp/kpsymb.f

```
1:      subroutine KPSYMB( PSYM, DIR,  PID,  MODE )
2: C=====
3: C purpose: conversion to/from particle symbol from/to particle ID
4: C         for CGVIEW code
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C
8: C i/o PSYM : particle symbol string
9: C i DIR : a character to specify direction of conversion
10: C      '>' : from symbol to ID
11: C          returns PID=0 if no matching symbol is found.
12: C      '<' : from ID to symbol
13: C          returns PSYM=' ' if no matching ID is found.
14: C o/i PID : particle ID number
15: C i MODE : returned symbol string when DIR='<';
16: C         MODE=0 : full symbol name
17: C         MODE=1(<>0) : short symbol name
18: C=====
19:      include '../shared/INC/_IOUNIT'
20: C
21:      include 'INC/_KPLIDS'
22:      include 'INC/_KPSYMS'
23: C
24:      character*(*) PSYM, DIR
25:      integer PID, MODE
26: C
27: C-----
28: C
29:      if ( DIR.eq.'>' ) then
30:          PID = 0
31: C
32: C      .... check full name first ....
33: C
34:          do 100 I = 1, KPLIM
35:              if ( KPSYM(1,I).eq.PSYM ) then
36:                  PID = I
37:                  go to 110
38:              end if
39:          100 continue
40:          110 continue
41:          if ( PID.gt.0 ) return
42: C
43: C      .... check short name (alias) ....
44: C
45:          do 120 I = 1, KPLIM
46:              if ( KPSYM(2,I).eq.PSYM ) then
47:                  PID = I
48:                  go to 130
49:              end if
50:          120 continue
51:          130 continue
52:          return
53: C
54:      else if ( DIR.eq.'<' ) then
55: C
56:          PSYM = ' '
57:          if ( PID.lt.1 .or. PID.gt.KPLIM ) return
58: C
59:          if ( MODE.eq.0 ) then
60:              PSYM = KPSYM(1,PID)
61:          else
62:              PSYM = KPSYM(2,PID)
63:          end if
64: C
65:      else
```

```
66:          write(IMGMSG,*) 'XXX(KPSYMB) Program error:',
67:      &          ' invalid conversion direction key ',
68:      &          '<', DIR,'> is given (must be ">" or "<").'
69:          stop 666
70:      end if
71: C
72:      return
73:      end
```

src/gmvp/legend.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine LEGEND
3: C=====
4: C  PURPOSE: TO CONVERT LEGENDRE COEFFICIENTS TO MOMENTS
5: C  CALLED IN: JNPUT          CALLS:  (NONE)
6: C=====
7:   real P1(25), P2(25)
8:   common /MOMENT/  NMOM,   XMOMNT(25),   NF,   F(25)
9:
10:  NFM   = NMOM - 1
11:  P10   = 0.0
12:  P1(1) = 1.0
13:  XMOMNT(1) = F(1)
14:
15:  if ( NMOM.lt.2 ) go to 140
16:
17:  do 100 L = 2, NMOM
18:    P1(L) = 0.0
19:  100 continue
20:
21:  do 130 N = 2, NMOM
22:    P20   = P1(1) /3.0
23:    P2(1) = P10 + 0.4*P1(2)
24:    if ( NMOM.gt.2 ) then
25:      do 110 L = 2, NFM
26:        FL   = L
27:        P2(L) = FL*P1(L-1)/(2.0*FL-1.0) + (FL+1.0)*P1(L+1) /
28:          &      (2.0*FL+3.0)
29:      110 continue
30:    end if
31:    FNF   = NMOM
32:    P2(NMOM) = FNF*P1(NFM) / (2.0*FNF-1.0)
33:    XMOMNT(N) = P20
34:    do 120 L = 1, NMOM
35:      XMOMNT(N) = XMOMNT(N) + P2(L)*F(L)
36:      P1(L) = P2(L)
37:    120 continue
38:    P10 = P20
39:  130 continue
40:  140 return
41:  end
```

src/gmvp/macrol.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine MACRO1( N97,   NEL1,  RHO,   NELEM, SIGT,  NSD )
3: C=====
4: C PURPOSE:  MAKE MACRO CROSS SECTION FROM DDX LIBRARY
5: C CALLED IN:  JNPUT,      CALLS:  MACRO2,MACRO3
6: C=====
7:   real SIGT(1)
8: C
9:   common /DDX/      IDDX,   MAXU,   MAXSD,  IMAX,   NLASTD, MAXDDX,
10:  &      ISTART,  NGP1,   NGG1
11: C
12:   integer NSD(IMAX)
13:   character*4 HOL(12)
14: C
15:   include '../shared/INC/_IUNIT'
16:   include 'INC/_IUNIT2'
17: C
18: C-----
19: C
20:   if ( IDDX.eq.0 ) return
21: C
22:   JMIX   = NGP1*NGG1
23:   NEL     = IABS(NEL1)
24:   MTBL    = MAXU*MAXSD*IMAX
25:   NSRT11  = ISTART + 1
26:   NSRT21  = NSRT11 + MTBL + 5*IMAX - 1
27: C
28:   if ( JMIX.ne.0 ) then
29:     NSRT12 = NSRT21 + 1
30:     NSRT22 = NSRT21 + IMAX
31:     NSRT13 = NSRT22 + 1
32:     NSRT23 = NSRT22 + NGG1*NGP1
33:   else
34:     NSRT12 = NSRT21 + 1
35:     NSRT13 = NSRT12
36:     NSRT23 = NSRT21
37:   end if
38: C
39:   NSRT31 = NSRT23 + 1
40:   NSRT41 = NSRT23 + IMAX*5
41:   NSRT32 = NSRT41 + 1
42: C
43:   if ( JMIX.ne.0 ) then
44:     NSRT42 = NSRT41 + IMAX
45:   else
46:     NSRT42 = NSRT41
47:   end if
48: C
49:   NSRT51 = NSRT42 + 1
50:   NSRT61 = NSRT42 + MTBL
51:   NSRT52 = NSRT61 + 1
52: C
53:   if ( JMIX.ne.0 ) then
54:     NSRT62 = NSRT61 + NGG1*NGP1
55:   else
56:     NSRT62 = NSRT61
57:   end if
58: C
59:   call RWIND( IOWDX1 )
60: C
61:   do 100 NE = 1, NELEM
62: C
63:     read(IOWDX1,end =9000) N1, N2, N3, N4, (HOL(I),I=1,12),
64:   &      (SIGT(I),I=NSRT11,NSRT21)
65:

```

```

66:     if ( JMIX.ne.0 ) then
67:       read(IOWDX1,end =9000) (SIGT(I),I=NSRT12,NSRT12+NGP1-1)
68:       read(IOWDX1,end =9000) (SIGT(I),I=NSRT13,NSRT23)
69:     end if
70:     if ( NE.eq.NEL ) go to 110
71: C
72:   100 continue
73: C
74:     write(IMSG,7000) NEL
75:   7000 format(/1X,'XXX(MACRO1) MIXTURE ELEMENT UNMATCH   DDX=',I3,
76:   &      '   NEL=',I3)
77:     call PRSTOP( 1, 'CROSS SECION ERROR.' )
78:     stop 888
79: C
80: C
81: C
82: C
83:   110 continue
84:     M97      = N97 - 1
85: C
86: C ..... MIXING OF DDX .....
87: C
88:     call MACRO2( SIGT(NSRT11), SIGT(NSRT11+5*IMAX), SIGT(NSRT12),
89:   &      SIGT(NSRT13), SIGT(NSRT31), SIGT(NSRT32), SIGT(NSRT51),
90:   &      SIGT(NSRT52), RHO )
91: C
92: C ..... NORMALIZATION & B.M.C TRANSFORMATION .....
93: C
94:     if ( NEL1.le.0 ) then
95:       call MACRO3( 0, SIGT, NSD )
96:       do 120 I = NSRT31, NSRT62
97:         SIGT(I) = 0.0
98:       120 continue
99:     end if
100:     return
101: C
102: C
103: C
104:   9000 write(IMSG,'(/1X,a,i3,a)') 'XXX END OF FILE IOWDX1(', IOWDX1,
105:   &      ' ) IN SUBROUTINE MACRO1'
106:     call PRSTOP( 1, 'CROSS SECION ERROR.' )
107:     stop 888
108:   end

```

src/gmvp/macro2.f

```
1:      subroutine MACRO2( S1,      S2,      S3,      S4,      S5,      S6,      S7,
2:      &                  S8,      RHO )
3: C=====
4: C  PURPOSE: MIXING OF DDX
5: C  CALLED IN: MACRO1,      CALLS: NONE
6: C=====
7:      common /DDX/      IDDX,      MAXU,      MAXSD,      IMAX,      NLASTD, MAXDDX,
8:      &                  ISTART, NGP1,      NGG1
9:      real S1(1), S2(MAXU,IMAX,1), S3(1), S4(NGG1,1)
10:     real S5(1), S7(MAXU,IMAX,1), S6(1), S8(NGG1,1)
11: C
12: C ..... MIX MATRIX .....
13: C
14:     do 110 IG = 1, IMAX
15: C       M1 = MAXU*(IG-1) + 1
16:       M1 = 1
17:       M2 = MAXU*MAXSD
18:       S1RHO = S1(IG)*RHO
19:       do 100 M = M1, M2
20:         S7(M,1,IG) = S7(M,1,IG) + S2(M,1,IG)*S1RHO
21:       100 continue
22:     110 continue
23: C
24: C ..... MIX 1-D DATA .....
25: C
26:     II = 5*IMAX
27:     do 120 I = 1, II
28:       S5(I) = S5(I) + S1(I)*RHO
29:     120 continue
30: C
31: C
32:     if ( NGP1*NGG1.ne.0 ) then
33: C
34: C ..... MATRIX .....
35: C
36:       do 140 IG = 1, NGP1
37:         S3RHO = S3(IG)*RHO
38:         do 130 M = 1, NGG1
39:           S8(M,IG) = S8(M,IG) + S4(M,IG)*S3RHO
40:         130 continue
41:       140 continue
42: C
43: C ..... MIXING 1-D DATA .....
44: C
45:       do 150 IG = 1, NGP1
46:         S6(IG) = S6(IG) + S3(IG)*RHO
47:       150 continue
48:     end if
49: C
50:     return
51:     end
```

src/gmvp/macro3.f

```

1:      subroutine MACRO3( KOPT, SIGT, NSD )
2: C=====
3: C PURPOSE: PREPARATION FOR NORMALIZATION OF DDX-MATRIX
4: C CALLED IN:  MACRO1,JNPUT          CALLS:  MACRO4,MACRO5
5: C=====
6:      real SIGT(1)
7:      common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
8: &      IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
9: &      ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
10: &      NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
11: &      NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
12: &      IADJ, NME, LOC, INGS, INSG, II, IO,
13: &      KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRDSG,
14: &      ISTR, IPRIN, IFMU, IMOM, IDTF, ISTAT, IPUN,
15: &      NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
16: &      KNNN, IGGG
17:      common /DDX/ IDDX, MAXU, MAXSD, IMAX, NLASTD, MAXDDX,
18: &      ISTTRT, NGP1, NGG1
19: C
20: C
21:      integer NSD(IMAX)
22: C
23:      include '../shared/INC/_IOUNIT'
24:      include 'INC/_IOUNIT2'
25: C
26:      if ( IDDX.eq.0 ) return
27: C
28:      JMX = NGP1*NGG1
29:      MTBL = MAXU*MAXSD*IMAX
30:      NSRT11 = ISTTRT + 1
31:      NSRT21 = NSRT11 + MTBL + 5*IMAX - 1
32:
33:      if ( JMX.ne.0 ) then
34:          NSRT12 = NSRT21 + 1
35:          NSRT22 = NSRT21 + IMAX
36:          NSRT13 = NSRT22 + 1
37:          NSRT23 = NSRT22 + NGG1*NGP1
38:      else
39:          NSRT12 = NSRT21 + 1
40:          NSRT13 = NSRT12
41:          NSRT23 = NSRT21
42:      end if
43:
44:      NSRT31 = NSRT23 + 1
45:      NSRT41 = NSRT23 + IMAX*5
46:      NSRT32 = NSRT41 + 1
47:      if ( JMX.ne.0 ) then
48:          NSRT42 = NSRT41 + IMAX
49:      else
50:          NSRT42 = NSRT41
51:      end if
52:      NSRT51 = NSRT42 + 1
53:      NSRT61 = NSRT42 + MTBL
54:      NSRT52 = NSRT61 + 1
55:      if ( JMX.ne.0 ) then
56:          NSRT62 = NSRT61 + NGG1*NGP1
57:      else
58:          NSRT62 = NSRT61
59:      end if
60: C
61: C ... called from JNPUT ...
62: C
63:      if ( KOPT.ne.0 ) then
64:          do 100 I = NSRT31, NSRT62
65:              SIGT(I) = 0.0

```

```

66:      100      continue
67:          call RWIND( IOMDX )
68:          return
69:      end if
70: C
71: C ... ADJOINT REVERSAL
72: C
73:      if ( IADJ.ne.0 ) then
74:          do 110 I = NSRT11, NSRT23
75:              SIGT(I) = 0.0
76:      110      continue
77:
78:          call MACRO5( SIGT(NSRT31), SIGT(NSRT32), SIGT(NSRT51),
79: &          SIGT(NSRT52), SIGT(NSRT11), SIGT(NSRT12),
80: &          SIGT(NSRT11+5*IMAX), SIGT(NSRT13), IMAX, MAXSD, MAXU,
81: &          NGP1, NGG1, JMX )
82:
83:      end if
84: C
85:      ICHK = 0
86:      NTEMP = IMAX*MAXSD
87: C
88:      if ( NUS.gt.0 ) NTEMP = MAX(NTEMP,(MAXSD+NUS)*MAXU)
89: C
90:      NSRT71 = NSRT11
91:      NSRT81 = NSRT11 + NTEMP - 1
92:      NTEMP = IMAX*MAXU
93:      if ( (NSRT81+NTEMP).lt.NSRT31 ) then
94:          NSRT72 = NSRT81 + 1
95:          NSRT82 = NSRT81 + NTEMP
96:      else
97:          NSRT72 = NSRT62 + 1
98:          NSRT82 = NSRT62 + NTEMP
99:          ICHK = NSRT72
100:      end if
101: C
102:      NTEMP = MAXU*(MAXSD+NUS)*3
103:      if ( ICHK.eq.0.and.(NSRT82+NTEMP).lt.NSRT31 ) then
104:          NSRT9 = NSRT82 + 1
105:          NSRT10 = NSRT82 + NTEMP
106:      else if ( ICHK.eq.0 ) then
107:          NSRT9 = NSRT62 + 1
108:          NSRT10 = NSRT62 + NTEMP
109:          ICHK = NSRT9
110:      else
111:          NSRT9 = NSRT82 + 1
112:          NSRT10 = NSRT82 + NTEMP
113:      end if
114: C
115:      do 120 I = NSRT11, NSRT23
116:          SIGT(I) = 0.
117:      120      continue
118:      if ( ICHK.ne.0 ) then
119:          do 130 I = ICHK, NSRT10
120:              SIGT(I) = 0.
121:      130      continue
122:      end if
123: C
124: C/#IF .NOT.SYSTEM(DECOSF*)
125:      call MACRO4( SIGT(NSRT31), SIGT(NSRT32), SIGT(NSRT51),
126: &          SIGT(NSRT52), SIGT(NSRT71), SIGT(NSRT72), SIGT(NSRT9),
127: &          NSD )
128: C/#ELSE
129: *      call MACRO4( SIGT(NSRT31), SIGT(NSRT32), SIGT(NSRT51),
130: *          SIGT(NSRT52), SIGT(NSRT71), SIGT(NSRT72), SIGT(NSRT9),

```


src/gmvp/macro3.f

```
131: *      &      NSD,  
132: *      &      SIGT(NSRT31), SIGT(NSRT32), SIGT(NSRT51),  
133: *      &      SIGT(NSRT52), SIGT(NSRT71), SIGT(NSRT72), SIGT(NSRT9) )  
134: C/#ENDIF  
135:  
136:      return  
137: C  
138:      end
```

SAFE

src/gmvp/macro4.f

```

1: C/#IF .NOT.SYSTEM(DECOSF*)
2:   subroutine MACRO4( S5, S6, S7, S8, S71, S72, S9,
3:     & NSD )
4: C/#ELSE
5: *   subroutine MACRO4(S5,S6,S7,S8,S71,S72,S9,NSD,
6:   & jjS5,jjS6,jjS7,jjS8,jjS71,jjS72,jjS9)
7: C/#ENDIF
8: C=====
9: C PURPOSE:  NORMALIZATION OF DDX-MATRIX DATA & TRANSFORMATION FOR
10: C          B.M.C. SAMPLING
11: C CALLED IN: MACRO3,          CALLS:  NONE
12: C=====
13: common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
14: & IFSPOG, IDSOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPOG,
15: & ISPORT, INPBIF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
16: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
17: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
18: & IADJ, NME, LOC, INGS, INSG, II, IO,
19: & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRS, IRBSG,
20: & ISTR, IPRIN, IFMU, IMOM, IDTE, ISTAT, IPUN,
21: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
22: & INNN, IGGG
23: common /DDX/ IDDX, MAXU, MAXSD, IMAX, NLASTD, MAXDDX,
24: & ISTTRT, NGP1, NGG1
25:
26:   real S5(IMAX,1), S6(IMAX), S7(MAXU,MAXSD,1), S8(NGG,1)
27:   real S71(MAXSD,1), S72(MAXU,1), S9(1)
28:
29: C/#IF SYSTEM(DECOSF*)
30: *   integer jjS5(IMAX,1),jjS6(IMAX),jjS7(MAXU,MAXSD,1),jjS8(NGG,1)
31: *   integer jjS71(MAXSD,1),jjS72(MAXU,1),jjS9(1)
32: *   equivalence ( SP, jjSP )
33: C/#ENDIF
34:
35:   integer NSD(IMAX)
36: C
37:   include 'INC/_IOUNIT2'
38: C
39:   if ( ISTAT.eq.0 ) then
40:     do 120 IG = 1, IMAX
41:       S5(IG,1) = 0.
42:       IGDD = IG - NUS
43:       if ( IGDD.lt.1 ) IGDD = 1
44:       do 110 IGD = IGDD, MAXSD
45:         do 100 MU = 1, MAXU
46:           S71(IGD,IG) = S71(IGD,IG) + S7(MU,IGD,IG)
47:         100 continue
48:         S5(IG,1) = S5(IG,1) + S71(IGD,IG)
49:       110 continue
50:     120 continue
51:   else
52:     do 150 IG = 1, IMAX
53:       S5(IG,1) = 0.
54:       IGDD = IG - NUS
55:       if ( IGDD.lt.1 ) IGDD = 1
56:       do 140 MU = 1, MAXU
57:         do 130 IGD = IGDD, MAXSD
58:           S72(MU,IG) = S72(MU,IG) + S7(MU,IGD,IG)
59:         130 continue
60:         S5(IG,1) = S5(IG,1) + S72(MU,IG)
61:       140 continue
62:     150 continue
63:   end if
64: C
65:   if ( NGG1*NGP1.ne.0.and.IG.le.NGP ) then

```

```

66:     do 170 IG = 1, IMAX
67:       S6(IG) = 0.0
68:       do 160 IGD = 1, NGG
69:         S6(IG) = S6(IG) + S8(IGD,IG)
70:       160 continue
71:     170 continue
72:   end if
73: C
74: C
75:   do 220 IG = 1, IMAX
76:     if ( S5(IG,1).le.0. ) go to 220
77: C
78:     if ( ISTAT.ne.0 ) then
79:       IGDD = IG - NUS
80:       if ( IGDD.lt.1 ) IGDD = 1
81:       do 190 IGD = IGDD, MAXSD
82:         do 180 MU = 1, MAXU
83:           if ( S72(MU,IG).ne.0 ) S7(MU,IGD,IG) =
84:             & S7(MU,IGD,IG) /S72(MU,IG)
85:         180 continue
86:       190 continue
87: C
88:     else
89:       M1 = MAXU*(IG-1-NUS) + 1
90:       if ( M1.lt.1 ) M1 = 1
91:       M2 = MAXU*MAXSD
92:       do 200 MU = M1, M2
93:         S7(MU,1,IG) = S7(MU,1,IG) /S5(IG,1)
94:       200 continue
95:     end if
96:     do 210 MU = 1, MAXU
97:       S72(MU,IG) = S72(MU,IG) /S5(IG,1)
98:     210 continue
99:   220 continue
100: C
101:   if ( NGP1*NGG1.ne.0 ) then
102:     do 240 IG = 1, NGP
103:       do 230 IGD = 1, NGG
104:         S8(IGD,IG) = S8(IGD,IG) /S6(IG)
105:       230 continue
106:     240 continue
107:   end if
108: C
109: C .... WRITE INTO FILE IOMDX SIGMA,SCATTERING MATRIX,ANGULAR PROB. ....
110: C
111:   do 310 IG = 1, IMAX
112:     if ( NGP1*NGG1.ne.0.and.IG.le.NGP ) then
113: C
114:       call MACRO4( IOMDX, IG, MAXU, MAXSD, IMAX, NGG,
115:         & S5, S6, S7, S8, S71, S72, S9, S6(IG),
116:         & S5, S6, S7, S8, S71, S72, S9, S6(IG) )
117: C/#IF .NOT.SYSTEM(DECOSF*)
118: C   WRITE(IOMDX) (S5(IG,J),J=1,5),S6(IG),(S71(IGD,IG),IGD=1,MAXSD),
119: C   + ((S7(MU,IGD,IG),MU=1,MAXU),IGD=1,MAXSD),
120: C   + (S72(MU,IG),MU=1,MAXU),
121: C   + (S8(IGD,IG),IGD=1,NGG)
122: C/#ELSE
123: C   WRITE(IOMDX) (jjS5(IG,J),J=1,5),jjS6(IG),
124: C   + (jjS71(IGD,IG),IGD=1,MAXSD),
125: C   + (jjS7(i,1,IG),i=1,MAXU*MAXSD),
126: C   + (jjS72(MU,IG),MU=1,MAXU),
127: C   + (jjS8(IGD,IG),IGD=1,NGG)
128: C/#ENDIF
129: C/#ENDIF
130:

```

src/gmvp/macro4.f

```

131:         else
132:         SP      = 0.
133:         call MACRO44( IOMDX, IG, MAXU, MAXSD, IMAX, NGG,
134:         &          S5, S6, S7, S8, S71, S72, S9, SP,
135:         &          S5, S6, S7, S8, S71, S72, S9, SP )
136:
137: C/#IF .NOT.SYSTEM(DECOSF*)
138: c      WRITE(IOMDX) (S5(IG,J),J=1,5),SP,
139: c      + (S71(IGD,IG),IGD=1,MAXSD),
140: c      + ((S7(MU,IGD,IG),MU=1,MAXU),IGD=1,MAXSD),
141: c      + (S72(MU,IG),MU=1,MAXU),
142: c      + (SP,IGD=1,NGG)
143: C/#ELSE
144: c      WRITE(IOMDX) (jjs5(IG,J),J=1,5),jjSP,
145: c      + (jjs71(IGD,IG),IGD=1,MAXSD),
146: c      + (jjs7(i,1,IG),i=1,MAXU*MAXSD),
147: c      + (jjs72(MU,IG),MU=1,MAXU),
148: c      + (jjSP,IGD=1,NGG)
149: C/#ENDIF
150:
151:         end if
152: C
153:         IGDD    = IG - NUS
154: C. IF(IGDD.LE.0) ---> UPSCATTERING COMPONENTS ARE NOT GIVEN IN S7
155:         MAXUSD  = MAXU*NSD(IG)
156:         do 250 I = 1, MAXUSD*3
157:             S9(I) = 0.0
158:         250 continue
159:         if ( ISTAT.eq.0 ) then
160:             if ( IGDD.gt.0 ) then
161:                 call BMCMK1( S7(1,IGDD,IG), S9(1), S9(MAXUSD+1), MAXUSD )
162:             else
163:                 INDAT = MAXU*(1-IGDD)
164:                 do 260 I = 1, MAXUSD
165:                     if ( I.le.INDAT ) then
166:                         S71(I,1) = 0.0
167:                     else
168:                         I1 = I - INDAT
169:                         S71(I,1) = S7(I1,1,IG)
170:                     end if
171:                 260 continue
172:                 call BMCMK1( S71(1,1), S9(1), S9(MAXUSD+1), MAXUSD )
173:             end if
174: C/#IF .NOT.SYSTEM(DECOSF*)
175:             write(IOMDX) (S9(I),I=1,MAXUSD*3)
176: C/#ELSE
177: *             write(IOMDX) (jjs9(I),I=1,MAXUSD*3 )
178: C/#ENDIF
179:         else
180:             do 290 MU = 1, MAXU
181:                 if ( S72(MU,IG).ne.0. ) then
182:                     do 270 IGD = 1, NSD(IG)
183:                         IGDDD = IGDD + IGD - 1
184: C..... IGDDD : SOURCE GROUP
185:                     if ( IGDDD.gt.0 ) then
186:                         S71(IGD,1) = S7(MU,IGDD+IGD-1,IG)
187:                     else
188:                         S71(IGD,1) = 0.0
189:                     end if
190:                 270 continue
191:             else
192:                 do 280 IGD = 1, NSD(IG)
193:                     S71(IGD,1) = 1.0
194:                 280 continue
195:             end if

```

```

196: C
197:         IS      = NSD(IG)*(MU-1)*3 + 1
198:         call BMCMK1( S71(1,1), S9(IS), S9(NSD(IG)+IS), NSD(IG) )
199:         290 continue
200: C/#IF .NOT.SYSTEM(DECOSF*)
201:         write(IOMDX) (S9(I),I=1,MAXUSD*3)
202: C/#ELSE
203: *             write(IOMDX) (jjs9(I),I=1,MAXUSD*3 )
204: C/#ENDIF
205:
206:         call BMCMK1( S72(1,IG), S9(1), S9(MAXU+1), MAXU )
207:
208: C/#IF .NOT.SYSTEM(DECOSF*)
209:         write(IOMDX) (S9(I),I=1,MAXU*3)
210: C/#ELSE
211: *             write(IOMDX) (jjs9(I),I=1,MAXU*3 )
212: C/#ENDIF
213:         end if
214:         if ( NGP1*NGG1.ne.0 ) then
215:             do 300 I = 1, NGG*3
216:                 S9(I) = 0.0
217:             300 continue
218:
219:             if ( IG.le.NGP ) call BMCMK1( S8(1,IG),S9(1),S9(NGG+1),NGG )
220:
221: C/#IF .NOT.SYSTEM(DECOSF*)
222:             write(IOMDX) (S9(I),I=1,NGG*3)
223: C/#ELSE
224: *             write(IOMDX) (jjs9(I),I=1,NGG*3 )
225: C/#ENDIF
226:         end if
227:         310 continue
228: c
229:         return
230:         end
231: C-----
232:         subroutine MACRO44( IOMDX, IG, MAXU, MAXSD, IMAX, NGG, S5,
233:         &          S6, S7, S8, S71, S72, S9, SP,
234:         &          JJS5, JJS6, JJS7, JJS8, JJS71, JJS72,
235:         &          JJS9, JJSP )
236: c
237: c
238: c
239:         real S5(IMAX,1), S6(IMAX), S7(MAXU,MAXSD,1), S8(NGG,1)
240:         real S71(MAXSD,1), S72(MAXU,1), S9(1)
241:
242:         integer JJS5(IMAX,1), JJS6(IMAX), JJS7(MAXU,MAXSD,1), JJS8(NGG,1)
243:         integer JJS71(MAXSD,1), JJS72(MAXU,1), JJS9(1)
244:
245: C/#IF .NOT.SYSTEM(DECOSF*)
246:         write(IOMDX) (S5(IG,J),J=1,5), SP, (S71(IGD,IG),IGD=1,MAXSD),
247:         &          ((S7(MU,IGD,IG),MU=1,MAXU),IGD=1,MAXSD),
248:         &          (S72(MU,IG),MU=1,MAXU), (S8(IGD,IG),IGD=1,NGG)
249: C/#ELSE
250: *             write(IOMDX) (jjs5(IG,J),J=1,5),jjSP,
251: *             & (jjs71(IGD,IG),IGD=1,MAXSD),
252: *             & (jjs7(i,1,IG),i=1,MAXU*MAXSD),
253: *             & (jjs72(MU,IG),MU=1,MAXU),
254: *             & (jjSP,IGD=1,NGG)
255: C/#ENDIF
256:         return
257:         end

```

src/gmvp/macro5.f

```
1:      subroutine MACRO5( S5,      S6,      S7,      S8,      S1,      S2,      S3,
2:      &                  S4,      IMAX,      MAXSD,      MAXU,      NGP1,      NGG1,      JMX
3:      &                  )
4: C=====
5: C PURPOSE: ADJOINT REVERSAL OF OF DDX-MATRIX
6: C CALLED IN:  MACRO3                  CALLS:  NONE
7: C=====
8:      real S5(IMAX,1), S6(1), S7(MAXU,MAXSD,1), S8(NGG1,1)
9:      real S1(IMAX,1), S2(1), S3(MAXU,MAXSD,1), S4(NGG1,1)
10: C
11:      II      = IMAX*5
12:      do 100 I = 1, II
13:          S1(I,1) = S5(I,1)
14:      100 continue
15:      II      = MAXU*MAXSD*IMAX
16:      do 110 I = 1, II
17:          S3(I,1,1) = S7(I,1,1)
18:      110 continue
19: C
20:      do 130 J = 1, 5
21:          do 120 I = 1, IMAX
22:              S5(I,J) = S1(IMAX-I+1,J)
23:          120 continue
24:      130 continue
25:      do 160 MU = 1, MAXU
26:          do 150 I = 1, IMAX
27:              do 140 J = 1, MAXSD
28:                  S7(MU,J,I) = S3(MU,MAXSD-I+1,IMAX-J+1)
29:              140 continue
30:          150 continue
31:      160 continue
32: C
33:      if ( JMX.ne.0 ) then
34:          do 170 I = 1, NGP1*NGG1
35:              S4(I,1) = S8(I,1)
36:          170 continue
37: C
38:          do 190 J = 1, NGG1
39:              IS      = NGP1*(J-1)
40:              do 180 I = 1, NGP1
41:                  S8(IS+I,1) = S4(NGG1-J+1,NGP1-I+1)
42:              180 continue
43:          190 continue
44:      end if
45:      return
46:      end
```

```
src/gmvp/main.f
```

```

1: *VOCL TOTAL, SCALAR
2: C
3: C   JAEA VECTORIZED MONTE CARLO TRANSPORT CODE.
4: C
5: C
6: C   _/_/_/_/_/_      _/_      _/_      _/_      _/_/_/_/_/_/_/_
7: C   _/_      _/_      _/_/_/_      _/_/_/_      _/_/_/_/_/_/_/_
8: C   _/_      _/_      _/_/_/_      _/_/_/_      _/_/_/_/_/_/_/_
9: C   _/_      _/_      _/_/_/_      _/_/_/_      _/_/_/_/_/_/_/_
10: C  _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_
11: C  _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_
12: C  _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_
13: C  _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_    _/_/_/_/_/_/_/_
14: C
15: C The multigroup neutron/photon transport Monte Carlo simulation code
16: C of Japan Atomic Research Agency (JAEA).
17: C
18: C -----
19: C History of development
20: C -----
21: C
22: C 1987:
23: C     First internal version was programmed on the FACOM VP-100 super-
24: C computer.
25: C     Most of cross section processing part was borrowed from
26: C MORSE-DD code (JAERI 1214, 1988).
27: C 1988-1989:
28: C     Lattice geometry capability useful for reactor core analysis etc.
29: C 1989:
30: C     Continuous energy version was created as the MVP code.
31: C 1991:
32: C     Point detector.
33: C 1993:
34: C     Modified as a multi-platform code which runs on many UNIX systems
35: C as well as the FACOM VP supercomputer on which this code had been
36: C developed.
37: C 1994:
38: C     Opened for public use as GMVP version 94.1.
39: C 1995:
40: C     January: Support HI/OSF - SYSTEM(HIOSF) (Hitachi S-series)
41: C
42: C -----
43: C Development staff
44: C -----
45: C
46: C Version 3:
47: C     Research Group for Reactor Physics and Standard Nuclear
48: C Code System of JAEA
49: C     Dr. Y.Nagaya, K.Okumura, Dr. T.Mori
50: C
51: C Version 2:
52: C     Reactor Physics Group of JAERI:
53: C     Y.Nagaya, K.Okumura, Dr. T.Mori, Dr. M.Nakagawa
54: C supported by
55: C     M.Sasaki & K.Kaneko
56: C
57: C Version 1:
58: C     Reactor System Laboratory of JAERI:
59: C     Dr. T.Mori, Dr. M.Nakagawa
60: C supported by
61: C     The Japan Research Institute, Limited:
62: C     M.Sasaki (coding,debugging etc.)
63: C and members of Reactor System Laboratory.
64: C
65: C =====

```

```

66: C
67: C/#IF CMAIN
68: *      subroutine FTMAIN
69: C/#ELSE
70:      program GMVP
71: C/#ENDIF
72:
73: C/#IF MS_VISUAL
74: C ... enable to use iargc, getarg for Visual Fortran.
75: *      use dflib
76: *      use dfport
77: C/#ENDIF
78:
79: C/#IF PARA(VPP)
80: *      include '../shared/INC/_VPPPARAM'
81: C/#ENDIF
82: C
83:      character*128 CSTRNG
84:      character*128 TSKINF
85: C
86: C/#IF PARA(VPP)
87: *      real*8 TTT0, TTT1
88: !xocl processor p(mpe)
89: C/#ENDIF
90: C
91:      include '../shared/INC/_IOUNIT'
92: C
93: C-----
94: C
95: CC/#IF UNBUFFER
96: CC
97: CC ... FUNBUF: set standard output unbuffered
98: CC
99: C      call FUNBUF()
100: CC/#ENDIF
101: C
102: C ... initialize some I/O unit variables.
103: C
104: C      call IIOSET
105: C
106: C ... check environment variables if possible ...
107: C
108: C (FUNBUF may be called by checking environment variable)
109: C
110: C/#IF .NOT.NOGETENV
111: C      call CHKENV( 'GMVP' )
112: C/#ENDIF
113: C
114: C/#IF PARA(VPP)
115: C
116: C      call CHROUND(3)
117: C
118: *      call GETTOD(TTT0)
119: C/#ENDIF
120: C/# IF SYSTEM(FACOMVPP)
121: C      call CHROUND(3)
122: C/# ENDIF
123: C
124: C ... common area data initialization (not to use BLOCK DATA
125: C      whih may cause data loss when linked through lib*.a archive)
126: C
127: C      call WDBL
128: C      call BPROGV
129: C      call FLAGIN
130: C      call ERINIT

```

src/gmvp/main.f

```

131:      call FRINIT(INP,IPR,10)
132:      call INDROT
133:      call KPINIT
134: C
135: C ... initialization for custom version
136: C
137:      call ZZINIT
138: C
139:      NTASK0 = 0
140:      TSKINF = ' '
141:      MLIMIT = 0
142:      IPREIN = 0
143:      call STSTFN( 'NOFILENAME' )
144: C
145: C-----
146: C Program control information from command line arguments.
147: C-----
148: C
149: C ... ARGNONE : FAT flag indicating no means to get command argument.
150: C
151: C/#IF ARGV
152: C
153: C/# IF ARGV( GETARG2 )
154: *      KJ = IARGC() - 1
155: C/# ELSEIF ARGV( MSF )
156: *      KJ = NARGS() - 1
157: C/# ELSE
158:      KJ = IARGC()
159: C/# ENDIF
160:
161:      do 100 I = 1, KJ
162:          IPREIN = IPREIN + 1
163:
164: C/# IF SYSTEM( CRAY )
165: *      call PXFGETARG(I, CSTRNG, LLLLL, ierr )
166: C/# ELSEIF ARGV( IGETARG )
167: *      LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
168: C/# ELSE
169: C/# IF ARGV( GETARG2 )
170: *      call GETARG(I+1, CSTRNG )
171: C/# ELSEIF ARGV( MSF )
172: C ... LL2 should be integer*2
173: *      call GETARG(I, CSTRNG, LL2 )
174: C/# ELSE
175:      call GETARG( I, CSTRNG )
176: C/# ENDIF
177:      LLLLL = MIN( INDEX(CSTRNG, ' ') - 1, LEN(CSTRNG) )
178:      if ( LLLLL.eq.0 ) LLLLL = LEN(CSTRNG)
179: C/# ENDIF
180: C
181: C ... quit command line check if '@@' is encountered ...
182: C
183:
184:      if ( CSTRNG(1:LLLLL) .eq. '@@' ) goto 105
185:
186:
187:      call PREIN0( IPREIN, CSTRNG(1:LLLLL), NTASK0, TSKINF, MLIMIT )
188:
189:      100 continue
190: C
191:      105 continue
192: C
193:      if ( KJ.gt.0 ) write(6,(' '1'))
194:
195: C ... END OF C/#IF ARGV ...

```

```

196: C/#ENDIF
197:
198: C
199: C-----
200: C ... pre-setting of SIGINT event handler that flushes standard
201: C      output.
202: C-----
203: C
204: C/#IF UNIX
205: C
206: C ... some implementation of MPI uses SIGUSR, SIGINT
207: C      so MVP/GMVP do not use it for them
208: C
209: C/#IF .NOT.PARA(MPI)
210:      call FSIGSET( )
211:      call FSIGUSR( )
212: C/#ENDIF
213: C/#ENDIF
214: C
215:      call AAALOC
216:      call ACALOC
217: C
218: C-----
219: C Processing Starts
220: C-----
221: C
222: C/#IF PARA
223: *      call CENTER( NTASK0, TSKINF, MLIMIT )
224: C/#ELSE
225: C/# IF .NOT.DYNAMIC
226: *      call CENTER
227: C/# ELSE
228:      call CENTER( MLIMIT )
229: C/# ENDIF
230: C/#ENDIF
231: C
232: C/#IF PARA(VPP)
233:      call GETTOD( TTT1 )
234:      TMTOTL = TTT1 - TTT0
235: C
236:      write(6,*) ' '
237:      write(6,7000) '##### VPP exec-time information'
238:      write(6,7000) '#####'
239:      write(6,7020) '##### time(intro) :', TMINTR/1.0D06,
240:      & '[sec]'
241:      write(6,7020) '##### time(para-region) :', TMPARA/1.0D06,
242:      & '[sec]'
243: C
244:      write(6,7000) '##### time(restart-input)'
245:      do 110 I = 1, MPE
246:          write(6,7040) '#####', 'PE', I, ':', TMINPP(I) /1.0D06,
247:          & '[sec]'
248:      110 continue
249: C
250:      write(6,7000) '##### time(action)'
251:      do 120 I = 1, MPE
252:          write(6,7040) '#####', 'PE', I, ':', TMACT(I) /1.0D06,
253:          & '[sec]'
254:      120 continue
255: C
256:      write(6,7000) '##### time(PE-sum)'
257:      do 130 I = 1, MPE
258:          write(6,7040) '#####', 'PE', I, ':', TMSUM(I) /1.0D06,
259:          & '[sec]'
260:      130 continue

```

src/gmvp/main.f

```
261: C
262:     write(6,7000) '#####   time(file-output)'
263:     do 140 I = 1, MPE
264:         write(6,7040) '#####', 'PE', I, ': ', TMOUTP(I) /1.0D06,
265:         &           '[sec]'
266:     140 continue
267: C
268:     write(6,7020) '#####   time(gather print): ', TMPRINT/1.0D06,
269:     &           '[sec]'
270:     write(6,7020) '#####   time(cadenz)           : ', TMCDNZ/1.0D06,
271:     &           '[sec]'
272:     write(6,7020) '#####   time(total)           : ', TMTOTL/1.0D06,
273:     &           '[sec]'
274:
275: 7000 format(A)
276: 7020 format(A,1X,F8.3,1X,A)
277: 7040 format(A,15X,A,12,2X,A,1X,F8.3,1X,A)
278:
279: C/#ENDIF
280: C
281:     stop
282:     end
```

src/gmvp/mament.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine MAMENT( NMO )
3: C=====
4: C  PURPOSE : TO CONVERT MOMENTS TO LEGENDRE COEFFICIENTS
5: C  CALLED IN: BADMOM,      CALLS: (NONE)
6: C=====
7:   common /MOMENT/ NMOM,  XMOMNT(25),  NF,  F(25)
8:   real P0(25), P1(25), P2(25)
9:
10:  do 100 I = 1, NMO
11:    P1(I) = 0.
12:    P0(I) = 0.
13:  100 continue
14:    P00 = 1.
15:    P1(1) = 1.
16:    P10 = 0.
17:    F(1) = XMOMNT(1)
18:
19:    if ( NMO.ge.2 ) then
20:      do 140 L = 2, NMO
21:        FL = L
22:        P20 = -(FL-1.) / FL * P00
23:        P2(1) = (2*FL-1.) / FL * P10 - (FL-1.) / FL * P0(1)
24:        do 110 N = 2, L
25:          P2(N) = (2*FL-1.) / FL * P1(N-1) - (FL-1.) / FL * P0(N)
26:  110      continue
27:        F(L) = P20
28:        do 120 M = 1, L
29:          F(L) = F(L) + P2(M) * XMOMNT(M)
30:  120      continue
31:        do 130 I = 1, L
32:          P0(I) = P1(I)
33:          P1(I) = P2(I)
34:  130      continue
35:          P00 = P10
36:          P10 = P20
37:  140      continue
38:        end if
39:      return
40:    end
```


src/gmvp/mntcpu.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine MNTCPU( NS )
3: C=====
4: C  PURPOSE : CHECK CPU TIME TO CALL MONITOR ROUINTES (--> VMNTR0 )
5: C  CALLED IN: ACTION
6: C  CALLS : VMNTR0,VMNTR1,VMNTR2,VMNTRC,VMNTRG
7: C=====
8:   include '../shared/INC/_IOUNIT'
9: CCC   CHARACTER*6 NS
10:   parameter( NTEST   = 100 )
11:
12: C/#IF SYSTEM(SX* ACOS)
13: *   real*8  DT0,DT1,DT3,DT4
14: C/#ENDIF
15:   real T00
16:
17:   T00   = 0.0
18:   do 100 I = 1, NTEST
19: C/#IF SYSTEM(FACOM*)
20: *       call CLOCK(T0,0,1)
21: *       call CLOCK(T1,0,1)
22: C/#ELSEIF SYSTEM(SX* ACOS)
23: *       call CLOCK(DT0)
24: *       call CLOCK(DT1)
25: C/#ELSEIF SYSTEM(CRAY*)
26: *       call SECOND(T0)
27: *       call SECOND(T1)
28: C/#ELSE
29: *       call CPUTM(T0)
30: *       call CPUTM(T1)
31: C/#ENDIF
32: C/#IF SYSTEM(SX* ACOS)
33:   T00   = T00 + DT1 - DT0
34: C/#ELSE
35:   T00   = T00 + T1 - T0
36: C/#ENDIF
37:   100 continue
38:
39:   7000 format(1X,A,E12.5,A,I2,A)
40:
41:   T0    = T00/REAL(NTEST)
42:   write(IOW,7000) ' (MNTCPU) ... T0 = ', T0, ' SEC (', NS, ' )'
43:
44:   do 110 I = 1, NTEST
45:     call VMNTR0( NS , IOW )
46:     call VMNTR2( NS )
47:   110 continue
48:
49:   call VMNTRT( T01, NS )
50:   T2    = T01/REAL(NTEST)
51:   write(IOW,7000) ' (MNTCPU) ... SCPU = ', T2, ' SEC (', NS, ' )'
52:
53:   T00   = 0.0
54:
55:   do 120 I = 1, NTEST
56: C/#IF SYSTEM(FACOM*)
57: *       call CLOCK(T3,0,1)
58: CC/#ELSEIF SYSTEM(HP* SUN* DECOSF* MIPS* NECEWS* SGI* PARAGON*)
59: C       call cputime(t3)
60: C/#ELSEIF SYSTEM(ACOS SX*)
61: *       call CLOCK(DT3)
62: C/#ELSEIF SYSTEM(CRAY*)
63: *       call SECOND(T3)
64: C/#ELSE
65: *       call CPUTM(T3)

```

```

66: C/#ENDIF
67:   call VMNTR1( NS, 9999 )
68: C/#IF SYSTEM(FACOM*)
69: *       call CLOCK(T4,0,1)
70: CC/#ELSEIF SYSTEM(HP* SUN* DECOSF* MIPS* NECEWS* SGI*)
71: C       call CPUTIME( T4 )
72: C/#ELSEIF SYSTEM(ACOS SX*)
73: *       call CLOCK(DT4)
74: C/#ELSEIF SYSTEM(CRAY*)
75: *       call SECOND(T4)
76: C/#ELSE
77: *       call CPUTM(T4)
78: C/#ENDIF
79:
80: C/#IF SYSTEM(ACOS SX*)
81: *       T00   = T00 + DT4 - DT3 - T0
82: C/#ELSE
83:   T00   = T00 + T4 - T3 - T0
84: C/#ENDIF
85:
86:   120 continue
87:
88:   T5    = T00/REAL(NTEST)
89:   write(IOW,7000) ' (MNTCPU) ... SCPU2 = ', T5, ' SEC (', NS, ' )'
90:   call VMNTRG( T2, REAL(T5), NS )
91:   call VMNTRC
92:
93:   return
94:   end

```

src/gmvp/mtsums.f

```

1: *VOCL TOTAL,SCALAR
2: C/#IF PARA(CRAY* SX*)
3: *      subroutine MTSUMS(TITLE, H, IDTASK1, IAINFL1, LIMITL1,
4: *      &      IRANT, NRSTEP, NBATCH1, NPART1, NHIST1, NBANK1, IMPMAX1,
5: *      &      NTHIST1, NFBANK1, NFBNK01, SRCSP,
6: *      &      WSUM, WLEK, XAVT, AAVT, EAVT,
7: *      &      WCNTR, NCNTR, NLOST,
8: *      &      SFLTR,      SFLCL,
9: *      &      SRETR,      SRECL,
10: *      &      SFLPD, SREPD,
11: *      &      XSOC, XSXV, XKEF,
12: *      &      IETAL, ETALY, TFLHS,
13: *      &      ETRV,
14: *      &      NCLSN, WCLSN, NLEAK, WLEAK,
15: *      &      NABSB, WABSB, NECUT, WECUT,
16: *      &      NECUT, WTCUT, NKILD, WKILD,
17: *      &      NSURV, WSRV, NSPLT, WSPLT )
18: C/#ELSE
19:      subroutine MTSUMS(TITLE,H)
20: C/#ENDIF
21: C=<GMVP>=====
22: C PURPOSE:  Take summation of data after calculation in multitask mode.
23: C
24: C      In VPP-Fortran, this routine must be called in
25: C      "!xocl parallel region ... !xocl end parallel" block
26: C
27: C CALLED IN: ACTMPP, ACTVPP, ACTSMP
28: C CALLS: TSKSM*
29: C=====
30:      character TITLE(2)*72
31: CCCC  include '../shared/INC/_ARRAY'
32:      real H(*)
33: CCC
34:      include 'INC/_FLAGS'
35:      include '../shared/INC/_SIZES'
36:      include 'INC/_SIZES2'
37:      include 'INC/_XBANK'
38:      include 'INC/_STACK'
39:      include 'INC/_XWORK'
40:      include '../shared/INC/_PGEOM'
41:      include 'INC/_PXSEC'
42:      include '../shared/INC/_PVRED'
43:      include 'INC/_PSOUR'
44:      include '../shared/INC/_PTALY0'
45:      include 'INC/_PTALY'
46:      include '../shared/INC/_PTLSP'
47:      include '../shared/INC/_STALY'
48: C
49:      include '../shared/INC/_COUNTS'
50:
51: C/#IF PARA(PVM)
52:      include '../shared/INC/_PVMPARA'
53: C/#ELSEIF PARA(MPI)
54: *      include 'mpif.h'
55: C/#ELSEIF PARA(VPP)
56:      include '../shared/INC/_VPPPARA'
57: C/#ENDIF
58:      include '../shared/INC/_TASKDT'
59:      include '../shared/INC/_TIMEDT'
60:      include '../shared/INC/_IOUNIT'
61:      include 'INC/_IOUNIT2'
62: C
63:      logical      JMEM
64: C
65: C/#IF PARA(CRAY* SX*)

```

```

66: *      integer SRCSP(NSRCSP)
67: *      real*8 WSUM,WLEK
68: *      real      XAVT(3),AAVT(3),EAVT
69: *      integer IAINFL1(3)
70: C/# IF INTEGER8
71: *      integer*8 NPART1, NTHIST1
72: C/# ELSE
73: *      integer      NPART1, NTHIST1
74: C/# ENDF
75: C
76: *      real*8 DT
77: C/#ENDIF
78: C
79: C-----
80: C
81: C/#IF PARA
82: C
83: C/#IF PARA(SX* CRAY*)
84: *      call MVPSYNC_BARRIER( 1 )
85: C/#ENDIF
86: C
87: C/# IF PARA(CRAY* SX*)
88: *      if( IDTASK.EQ.ITASK0 ) goto 678
89: C/# ENDF
90: C
91: C/#IF PARA(SX* CRAY*)
92: *      call MVPSYNC_LOCK(1)
93: C/#ENDIF
94: C
95: C -----
96: C second argument of TSKSM* is effective only in shared memory
97: C multitask mode ( local data of master task/thread )
98: C -----
99: C
100:      if( JEIGN.eq.0 ) then
101:          call TSKSMI('NBATCH', NBATCH1, NBATCH, 1 )
102:      end if
103: C
104:      call TSKSMI('NTHIST', NTHIST1, NTHIST, 1 )
105: C
106:      call TSKSMD('WSUM', WSUM, H(LWSUM), 1 )
107:      call TSKSMD('WLEK', WLEK, H(LWLEK), 1 )
108:      call TSKSMR('XAVT', XAVT, H(LXAVT), 3 )
109:      call TSKSMR('AAVT', AAVT, H(LAAVT), 3 )
110:      call TSKSMR('EAVT', EAVT, H(LEAVT), 1 )
111:      call TSKSMI('NLOST', NLOST, H(LNLOST), 1 )
112:      call TSKSMD('WCNTR', WCNTR, H(LWCNTR), NEVENT )
113:      call TSKSMD('NCNTR', NCNTR, H(LNCNTR), NEVENT )
114:      call TSKSMD('SFLTR', SFLTR, H(LSFLTR), NGROUP*NTREG*2 )
115:      call TSKSMD('SFLCL', SFLCL, H(LSFLCL), NGROUP*NTREG*2 )
116:      call TSKSMD('SRETR', SRETR, H(LSRETR), NTREG*NRESP*2 )
117:      call TSKSMD('SRECL', SRECL, H(LSRECL), NTREG*NRESP*2 )
118: C
119:      if( JPTDT.ne.0 ) then
120:          call TSKSMD('SFLPD', SFLPD, H(LSFLPD), NGROUP*NPDET*2 )
121:          call TSKSMD('SREPD', SREPD, H(LSREPD), NGROUP*NPDET*2 )
122:      endif
123: C
124:      if( NSTALY.gt.0 ) then
125:          call TSKSMD('ETALY',ETALY,H(LETALY), NLETAL*2)
126:          if ( NETRV.gt.0 ) then
127:              call TSKSMD( 'ETRV', ETRV, H(LETRV), METRV*(NBATCH-NSKIP) )
128:          end if
129:      end if
130: C

```

src/gmvp/mtsums.f

```
131:      if( JTIME.gt.0 ) then
132:        call TSKSMD('TFLHS',TFLHS,H(LTFLHS), NPKIND*2)
133:      end if
134: C
135:      if( JEIGN.ne.0 ) then
136:        call TSKSMD('XSOC', XSOC, H(LXSOC), NBATCH )
137:        call TSKSMD('XSXV', XSXV, H(LXSXV), NLENG*3 )
138:        call TSKSMD('XKEF', XKEF, H(LXKEF), 5*NBATCH )
139:      endif
140: C
141:      if( JMNTR.ne.0 ) then
142:        call TSKSMD('NCLSN', NCLSN, H(LNCLSN), NGROUP*NREG )
143:        call TSKSMD('WCLSN', WCLSN, H(LWCLSN), NGROUP*NREG )
144:        call TSKSMD('NLEAK', NLEAK, H(LNLEAK), NGROUP*NREG )
145:        call TSKSMD('WLEAK', WLEAK, H(LWLEAK), NGROUP*NREG )
146:        call TSKSMD('NABSB', NABSB, H(LNABSB), NGROUP*NREG )
147:        call TSKSMD('WABSB', WABSB, H(LWABSB), NGROUP*NREG )
148:        call TSKSMD('NECUT', NECUT, H(LNECUT), NREG )
149:        call TSKSMD('WECUT', WECUT, H(LWECUT), NREG )
150:        call TSKSMD('NTCUT', NTCUT, H(LNTCUT), NGROUP*NREG )
151:        call TSKSMD('WTCUT', WTCUT, H(LWTCUT), NGROUP*NREG )
152:        call TSKSMD('NKILD', NKILD, H(LNKILD), NGROUP*NREG )
153:        call TSKSMD('WKILD', WKILD, H(LWKILD), NGROUP*NREG )
154:        call TSKSMD('NSURV', NSURV, H(LNSURV), NGROUP*NREG )
155:        call TSKSMD('WSURV', WSURV, H(LWSURV), NGROUP*NREG )
156:        call TSKSMD('NSPLT', NSPLT, H(LNSPLT), NGROUP*NREG )
157:        call TSKSMD('WSPLT', WSPLT, H(LWSPLT), NGROUP*NREG )
158:      endif
159: C
160: C/#IF PARA(SX* CRAY*)
161: *      call MVPSYNC_UNLOCK(1)
162: C/#ENDIF
163: C
164: C
165:      678 continue
166: C/#IF PARA(PVM MPI)
167: *      call TOKEI(TE2,0)
168: *      write(IOW,'(/lx,a,i5,a)')
169: *      & 'Task:',IDTASK,' ELAPSED TIME IN "MTSUMS" ==
170: *      write(IOW,'(lx,a,e12.5)') ' BEFORE TASK SUM ',TE1-TE0
171: *      write(IOW,'(lx,a,e12.5)') ' TASK SUM ',TE2-TE1
172: *      write(IOW,'(lx,a,e12.5)') ' TOTAL ',TE2-TE0
173: C/#ENDIF
174: C
175: C/# ENDIF
176: C
177:      return
178:      end
```

src/gmvp/mttask.f

```

1:      subroutine MTTASK( TITLE, A, H, CHA, IAINFL, LIMITL,
2:      &
3:      &
4:      &
5:      C/#IF PARA( CRAY* SX* )
6:
7: C=====
8: C purpose: to control multi processing of 'ACTION' routine in GMVP code.
9: C           (for shared memory machine such as CRAY MP series or SX3 )
10: C called in : center
11: C=====
12:      integer ITASK(MNTASK,NTASK)
13:      integer IRANT(NTASK)
14:      integer ILOCK(*)
15:      integer IBARR(*)
16:
17:      integer IRSUT(NTASK), IRSMT(NTASK), IRSLT(NTASK)
18: C
19:      character*72 TITLE(2)
20: C
21: CCCC include '../shared/INC/_ARRAY'
22:      integer IAINFL(3), LIMITL
23:      real A(*)
24:      real H(*)
25:      character*4 CHA(*)
26: C
27:      include '../shared/INC/_SIZES'
28:      include '../shared/INC/_IOUNIT'
29:      include 'INC/_FLAGS'
30:      include 'INC/_PSOUR'
31:      include '../shared/INC/_PTALY0'
32:      include 'INC/_PTALY'
33:      include '../shared/INC/_STALY'
34:
35:      include '../shared/INC/_RAND'
36: C
37:      external ACTSMP
38: C
39: C
40: C
41:      NRSTEP = NTASK
42:      if( MOD(NTASK,2) .eq. 0 ) NRSTEP = NRSTEP + 1
43: C
44: C
45: C
46: C-----
47: C assign lock and barrier
48: C-----
49: C
50:      CALL MVPSYNC_ASSIGN_LOCK( 1, IERR )
51:      CALL MVPSYNC_ASSIGN_LOCK( 2, IERR )
52:      if( JREPR.eq.0 ) CALL MVPSYNC_ASSIGN_LOCK( 3, IERR )
53:
54:      call MVPSYNC_ASSIGN_BARRIER( 1, IERR )
55:
56:      do 11 I=1,NTASK
57:          call MVPSYNC_ASSIGN_EVENT( I, IERR )
58:      11 continue
59:
60:      call CPUTM( TC0 )
61:      call TOKEI( TE0,0 )
62: C
63: C-----
64: C ... Fork sub tasks .....
65: C-----

```

```

66: C
67:      ITASK0 = 1
68:      IRAN0 = IRAND
69:      IRSU0 = IRNSU
70:      IRSM0 = IRNSM
71:      IRSL0 = IRNSL
72: C
73:      do 110 N = 2, NTASK
74: C
75: C ... skip random number
76: C
77:          call RANU2( IRAND, RTEMP, 1, ICOD)
78:
79:          IRANT(N) = IRAND
80:          IRSUT(N) = IRNSU
81:          IRSMT(N) = IRNSM
82:          IRSLT(N) = IRNSL
83:      110 continue
84:
85:      do 100 N = 2, NTASK
86: C
87: C ... Pass address of tally data of main task as arguments.
88: C
89:          IRANT(N) = IRAND
90:          IRSUT(N) = IRNSU
91:          IRSMT(N) = IRNSM
92:          IRSLT(N) = IRNSL
93: C
94: C/#IF PARA( SX* )
95:      &      IK = 2
96:          ITASK(IK,N) = N
97:          call PTFORK( ITASK(1,N), ITASK(2,N),
98: C/#ELSEIF PARA( CRAY* )
99:      *      IK = 3
100:      *      ITASK(1,N) = 3
101:      *      ITASK(IK,N) = N
102:      *      call TSKSTART( ITASK(1,N),
103: C/#ENDIF
104:      &      ACTSMP, TITLE,
105:      &      ITASK(IK,N), IAINFL, LIMITL,
106:      &      IRANT(N), NRSTEP, NBATCH, NPART, NHIST, NHSUB, NBANK,
107:      &      IMPMAX, NTHIST, NFBANK, NFBNK0, H(LSRCSP),
108:      &      H(LWSUM), H(LWLEK), H(LXAVT), H(LAAVT), H(LEAVT),
109:      &      H(LWCNTR), H(LNCNTR), H(LNLOST),
110:      &      H(LSFLTR), H(LSFLCL),
111:      &      H(LSRETR), H(LSRECL),
112:      &      H(LSFLPD), H(LSREPD),
113:      &      H(LXSOC), H(LXSXV), H(LXKEF),
114:      &      H(LIETAL), H(LETALY), H(LTFLHS), H(LETRV),
115:      &      h(lNCLSN), h(lWCLSN), h(lNLEAK), h(lWLEAK),
116:      &      h(lNABSB), h(lWABSB), h(lNECUT), h(lWECUT),
117:      &      h(lNLCUT), h(lWLCUT), h(lNKILD), h(lWKILD),
118:      &      h(lNSURV), h(lWSURV), h(lNSPLT), h(lWSPLT),
119:      &      IRSUT(N), IRSMT(N), IRSLT(N) )
120: C
121:      100 continue
122: C
123: C-----
124: C ... start main task
125: C-----
126: C
127:      N = 1
128: C/#IF PARA( SX* )
129:          ITASK(2,N) = ITASK0
130: C/#ELSEIF PARA( CRAY* )

```

src/gmvp/mttask.f

```
131:      ITASK(3,N) = ITASK0
132: C/#ENDIF
133:      IRANT(N) = IRAN0
134:      IRSUT(N) = IRSU0
135:      IRSMT(N) = IRSM0
136:      IRSLT(N) = IRSLO
137: C
138:      call ACTSMP( TITLE,
139:      &          ITASK0, IAINFL, LIMITL,
140:      &          IRANT(N), NRSTEP, NBATCH, NPART, NHIST, NHSUB, NBANK,
141:      &          IMPMAX, NTHIST, NFBANK, NFBNK0, H(LSRCSP),
142:      &          H(LWSUM), H(LWLEK), H(LXAVT), H(LAAVT), H(LEAVT),
143:      &          H(LWCNTR), H(LNCNTR), H(LNLOST),
144:      &          H(LSFLTR), H(LSFLCL),
145:      &          H(LSRETR), H(LSRECL),
146:      &          H(LSFLPD), H(LSREPD),
147:      &          H(LXSOC), H(LXSXV), H(LXKEF),
148:      &          H(LIETAL), H(LETALT), H(LTFLHS), H(LETRV),
149:      &          h(lNCLSN), h(lWCLSN), h(lNLEAK), h(lWLEAK),
150:      &          h(lNABSB), h(lWABSB), h(lNECUT), h(lWECUT),
151:      &          h(lNTCUT), h(lWTCUT), h(lNKILD), h(lWKILD),
152:      &          h(lNSURV), h(lWSURV), h(lNSPLT), h(lWSPLT),
153:      &          IRSUT(N), IRSMT(N), IRSLT(N) )
154: C
155: C-----
156: C      ... wait termination of subtask
157: C-----
158: C
159:      do 200 N=2,NTASK
160: C
161: C/#IF PARA( SX* )
162:      call PTJOIN( ITASK(1,N) )
163: C/#ELSEIF PARA( CRAY* )
164:      call TSKWAIT( ITASK(1,N) )
165: C/#ENDIF
166: C
167:      200 continue
168: C
169: C
170:      call CPUTM( TC1 )
171:      call TOKEI( TE1,0 )
172: C
173: C
174:      write(IOW,7200) TE1-TE0, TC1-TC0
175:      7200 format(/1x,' ==== TIME MONITOR OF MULITTASKING ==='/
176:      &          1x,'      TOTAL CPU TIME      ',e12.5,' (SEC)'/
177:      &          1x,'      TOTAL ELAPSED TIME    ',e12.5,' (SEC)'/)
178: C
179: C
180: C/#ENDIF
181: C
182:      return
183:      end
```

src/gmvp/neesel.f

```

1:      subroutine NEESEL( MODE, IEVENT,MACT, MZONE, NIMPT, NDIMPT,
2:      & IMNFL, IMNNX, IMDEFR,IMNLT )
3: C=====
4: C PURPOSE: SELECTION OF EVENT FOR NEXT EVENT ESTIMATOR.
5: C CALLED IN: NXTEE
6: C CALLS:
7: C
8: C=====
9: C
10: C IEVENT : COUNTER OF EVENT PROCESSING IN A CALL OF THE NXTEE ROUTINE.
11: C MACT : ACTION NUMBER
12: C MZONE : SELECTED ZONE # (FOR ZONE-SELECTION TASKS)
13: C
14: C OPERATION MODE :
15: C   MODE = 1 : PROCESS ALL IMAGINARY PARTICLES
16: C             UNTIL FINISHED.
17: C   MODE = 2 : PROCESS IMAGINARY PARTICLES UNTIL
18: C             ENTRY OF BANK IS LESS THAN 'NTHIN' PARTICLES.
19: C   MODE = -1 : PROCESS N EVENTS.
20: C=====
21:      include '../shared/INC/_SIZES'
22:      include 'INC/_SIZES2'
23:      include 'INC/_FLAGS'
24: C
25:      integer IMNFL(NZONE+1), IMNNX(NZONE+1), IMNLT(NLBZ+1)
26: C
27:      IEVENT = IEVENT + 1
28: C
29: C ..... LIMITED COUNT OF OPERATIONS ( MODE < 0, -MODE OPERATIONS )
30: C
31:      if ( MODE.lt.0.and.IEVENT.gt.-MODE ) then
32:         MACT = 99
33:         return
34:      end if
35: C
36: C ..... DEFER NEXT EVENT ESTIMATION IF NUMBER OF IMAGINARY PARTICLES
37: C           FALLS BELOW A LIMIT. ( IMPMAX * SUPPLY NOW )
38: C
39:      if ( MODE.eq.2.and.NIMPT.le.SUPPLY*IMPMAX ) then
40:         MACT = 99
41:         return
42:      end if
43: C
44: C ..... FINISH IF NO PARTICLES IN BANK ....
45: C
46:      if ( NIMPT.eq.0 ) then
47:         MACT = 99
48:         return
49:      end if
50: C
51: C ..... SELECT NEXT ACTION ....
52: C
53: C ===== FREE FLIGHT =====
54:      MAX1 = 0
55:      MZONE1 = 0
56:      if ( IMNFL(NZONE+1).gt.0 ) then
57:         do 100 N = 1, NZONE
58:            if ( IMNFL(N).gt.MAX1 ) then
59:               MAX1 = IMNFL(N)
60:               MZONE1 = N
61:            end if
62: 100      continue
63:      end if
64: C
65: C ===== NEXT-ZONE SEARCH =====
66: C
67:      MAX2 = 0
68:      MZONE2 = 0
69:      if ( IMNNX(NZONE+1).gt.0 ) then
70:         do 110 N = 1, NZONE
71:            if ( IMNNX(N).gt.MAX2 ) then
72:               MAX2 = IMNNX(N)
73:               MZONE2 = N
74:            end if
75: 110      continue
76:      end if
77:      MAX3 = IMDEFR
78: C
79: C
80: C
81:      if ( JLATT.ne.0 ) then
82:         MAX6 = 0
83:         if ( IMNLT(NLBZ+1).gt.(IMPMAX
84: & -NDIMPT)*0.5 .or. MAX1.eq.0.and.MAX2.eq.0 ) then
85:            MAX6 = IMNLT(NLBZ+1)
86:         end if
87:         if ( MAX6.gt.MAX(MAX1,MAX2,MAX3) ) then
88:            MACT = 6
89:            MZONE = 0
90:            return
91:         end if
92:      end if
93: C
94: C ..... SELECT FLIGHT .....
95: C
96:      MMAX = MAX(MAX1,MAX2,MAX3)
97:      if ( MAX1.eq.MMAX ) then
98:         MACT = 1
99:         MZONE = MZONE1
100: C
101: C ..... SELECT SEARCH .....
102: C
103:      else if ( MAX2.eq.MMAX ) then
104:         MACT = 2
105:         MZONE = MZONE2
106: C
107: C ..... SELECT DEFERRED PARTICLE PROCESSING FOR SEARCH ....
108: C
109:      else if ( MAX3.eq.MMAX ) then
110:         MACT = 2
111:         MZONE = -1
112:      end if
113:      return
114: C
115:      end

```

```

      real      RESP(NGROUP,NRESP)
      NBATCH = NBATCH + 1      ( MUST BE FINISHED IN TALSUM ! /
      if ( NBATCH.eq.NSKIP ) then
        do 110 I = 1, NPDET
          do 100 IG = 1, NGROUP
            FLPD(IG,I) = 0.0
          continue
        continue
      else if ( NBATCH.gt.NSKIP ) then
        .... CALCULATE RESPONSE HERE (VALID ONLY MULTIGROUP VERSION)
        do 140 N = 1, NRESP
          do 130 I = 1, NPDET
            REPD(I,N) = 0.0
            do 120 IG = 1, NGROUP
              REPD(I,N) = REPD(I,N) + RESP(IG,N)*FLPD(IG,I)
            continue
          continue
        continue
        .... SUMMATION OF RESPONSE ....
        do 160 N = 1, NRESP
          do 150 I = 1, NPDET
            SREPD(I,N,1) = SREPD(I,N,1) + REPD(I,N)
            SREPD(I,N,2) = SREPD(I,N,2) + (REPD(I,N)**2) /WSUMB
            REPD(I,N) = 0.0
          continue
        continue
        .... SUMMATION OF FLUX ....
        do 180 I = 1, NPDET
          do 170 IG = 1, NGROUP
            SFLPD(IG,I,1) = SFLPD(IG,I,1) + FLPD(IG,I)
            SFLPD(IG,I,2) = SFLPD(IG,I,2) + (FLPD(IG,I)**2) /WSUMB
            FLPD(IG,I) = 0.0
          continue
        continue

```

```

1:      subroutine NEESUM( IOW,          NBATCH,NSKIP, NGROUP,NPDET, NRESP,
2:      &                  NGENEO,WSUMB, RESP,  FLPD,  SFLPD,  REPD,
3:      &                  SREPD )
4: C=====
5: C  PURPOSE: TAKE SUMMATION OF NEXT EVENT ESTOMATOR TALLY AFTER EACH
6: C          BATCH.
7: C  CALLED IN: ACTION
8: C=====
9:      real*8  FLPD(NGROUP,NPDET),  SFLPD(NGROUP,NPDET,2),
10:     &        REPD(NPDET,NRESP),  SREPD(NPDET,NRESP,2)
11:      real*8  WSUMB
12:      real     RESP(NGROUP,NRESP)
13: C
14: C**** NBATCH = NBATCH + 1      / MUST BE FINISHED IN TALSUM ! /
15: C
16:     if ( NBATCH.eq.NSKIP ) then
17:         do 110 I = 1, NPDET
18:             do 100 IG = 1, NGROUP
19:                 FLPD(IG,I) = 0.0
20:             100 continue
21:         110 continue
22: CC
23:     else if ( NBATCH.gt.NSKIP ) then
24: CC
25: CC         .... CALCULATE RESPONSE HERE (VALID ONLY MULTIGROUP VERSION)
26: CC
27:         do 140 N = 1, NRESP
28:             do 130 I = 1, NPDET
29:                 REPD(I,N) = 0.0
30:                 do 120 IG = 1, NGROUP
31:                     REPD(I,N) = REPD(I,N) + RESP(IG,N)*FLPD(IG,I)
32:                 120 continue
33:             130 continue
34:         140 continue
35: CC
36: CC         .... SUMMATION OF RESPONSE ....
37: CC
38:         do 160 N = 1, NRESP
39:             do 150 I = 1, NPDET
40:                 SREPD(I,N,1) = SREPD(I,N,1) + REPD(I,N)
41:                 SREPD(I,N,2) = SREPD(I,N,2) + (REPD(I,N)**2) /WSUMB
42:                 REPD(I,N) = 0.0
43:             150 continue
44:         160 continue
45: CC
46: CC         .... SUMMATION OF FLUX      ....
47: CC
48:         do 180 I = 1, NPDET
49:             do 170 IG = 1, NGROUP
50:                 SFLPD(IG,I,1) = SFLPD(IG,I,1) + FLPD(IG,I)
51:                 SFLPD(IG,I,2) = SFLPD(IG,I,2) + (FLPD(IG,I)**2) /WSUMB
52:                 FLPD(IG,I) = 0.0
53:             170 continue
54:         180 continue
55: CC
56:     end if
57: CC
58: CC
59:     return
60: end

```

src/gmvp/ngvel.f

```

1:      subroutine NGVEL( ENGXB, VEL,   NGP1,  NGROUP )
2:      C=====
3:      C  PURPOSE: calculate energy group averaged neutron velocity
4:      C  CALLED IN: INTRO2
5:      C=====
6:      c  arguments (i=input, o=output, w=work )
7:      c
8:      c i engyb(ngp1+1) : neutron energy boundaries ( eV )
9:      c o vel(ngroup)   : averaged neutron velocity ( cm/s )
10:     c i ngp1   : number of neutron energy groups
11:     c i ngroup : total number of energy groups
12:     C=====
13:     implicit real*8(D)
14:     c
15:     c
16:     real ENGXB(NGP1+1)
17:     real VEL(NGROUP)
18:     c
19:     c
20:     ... constants
21:     c
22:     --- light speed --- (cm/s)
23:     c
24:     real*8 CLIGHT
25:     parameter( CLIGHT = 2.99792458D10 )
26:     c
27:     --- neutron mass --- (gram)
28:     c
29:     real*8 NMASS
30:     parameter( NMASS = 1.6749286D-24 )
31:     c
32:     --- erg / eV --- elementary electronic charge
33:     c
34:     real*8 EECHRG
35:     parameter( EECHRG = 1.60217733D-12 )
36:     c
37:     --- mc**2 in eV
38:     c
39:     real*8 MC2
40:     parameter( MC2 = NMASS*CLIGHT**2/EECHRG )
41:     c
42:     C-----
43:     c
44:     c === calculation method ===
45:     c
46:     c
47:     c  VEL(g) = (  $\int \frac{1}{E} dE$  ) / (  $\int \frac{1}{V(E) * E} dE$  )
48:     c
49:     c
50:     c
51:     c
52:     c . Relativistic form
53:     c
54:     c  Vr(E) = clight * SQRT( E* (E + 2* nmass*clight**2 ) ) /
55:     c
56:     c           ( E + nmass*clight**2 )
57:     c
58:     c . Non-relativistic form
59:     c
60:     c  Vnr(E) = sqrt( 2 / nmass * E )
61:     c
62:     C-----
63:     c
64:     c
65:     do 100 I = 1, NGP1

```

```

66:         DE1      = ENGXB(I)
67:         DE2      = ENGXB(I+1)
68:     c
69:         D1       = LOG(DE1/DE2)
70:     c
71:     ... Relativistic form for velocity ...
72:     c
73:         D2       = LOG(DE1+MC2+SQRT(DE1*(DE1+2*MC2)))
74:         &        - SQRT(DE1*(DE1+2*MC2)) /DE1
75:         &        - LOG(DE2+MC2+SQRT(DE2*(DE2+2*MC2)))
76:         &        + SQRT(DE2*(DE2+2*MC2)) /DE2
77:     c
78:     ... Non-relativistic form for velocity ...
79:     c
80:     cc      d2 = 2D0/sqrt(2/nmass*eechrg)
81:     cc      &   * ( 1.0d0/sqrt(de1) - 1.0d0/sqrt(de2) )
82:     c
83:     c
84:         VEL(I)   = CLIGHT*ABS(D1/D2)
85:     100 continue
86:     c
87:     return
88:     end

```


src/gmvp/nid2nn.f

```
1:      subroutine NID2NN( NAME, IPART, N,      A,      CHA )
2: C=<MVP>=====
3: C purpose: a dummy routine of the same name exists in MVP.
4: C      This is only to use the same tally input processing routine for
5: C      both MVP and GMVP.
6: C called in: talin1
7: C=====
8: C
9: C i name : nuclide ID or atomic symbol
10: C i ipart : 1 = nuclide ID for neutron.
11: C      2 = Atomic symbol for photon/electron.
12: C o n      : nuclide/atom # in problem.
13: C-----
14:      character*(*) NAME
15: C
16:      real A(*)
17:      character*4 CHA(*)
18: C
19: *
20: CCCC include 'inc/shared/INC/_ARRAY'
21: *      include 'INC/_CXSEC'
22: *
23: *      n = 0
24: *      if( ipart.eq.1 ) then
25: *          call nid2n1( name, a(LNUCID), NUC, n )
26: *      else if( ipart.eq.2 ) then
27: *          call nid2n2( name, a(LNATMT), npatom, n )
28: *      endif
29: *      return
30: *      end
31: *
32: C=====
33: C
34: *      subroutine nid2n1( name, nucid, nuc, n )
35: C
36: *      character*(*) name
37: *      character*16 nucid(nuc)
38: C
39: *      do 100 i=1,nuc
40: *          if( name.eq.ncid(i) ) then
41: *              n = i
42: *              return
43: *          endif
44: * 100 continue
45: *      n = 0
46: *      return
47: *      end
48: C
49: C=====
50: C
51: *      subroutine nid2n2( name, natmt, npatom, n )
52: C
53: *      character*(*) name
54: *      integer natmt(npatom)
55: C
56: *      external natomz
57: C
58: *      n = 0
59: *      m = natomz(name)
60: C
61: *      if( m.ne.0 ) then
62: *          do 100 i=1,npatom
63: *              if( natmt(i).eq.m ) then
64: *                  n = m
65: *                  return
66: *              endif
67: * 100 continue
68: *          endif
69: *          return
70: *      end
71: C
72: *      return
73: *      end
```

src/gmvp/nucfis.f

```

1:      subroutine NUCFIS( A,      H,      IG,      NS,      IMAT, EINCD, IRND,
2:      &                  IERR, RWK )
3: C=<GMVP>=====
4: C PURPOSE: sampling energy group from fission spectrum in GMVP.
5: C CALLED IN : SYSSRC
6: C CALLS : MATFIS
7: C-----
8: C
9: C      arguments ( i= input, o = output, w = work )
10: C
11: C io  A(*) : dynamic memory array (task shared)
12: C io  H(*) : dynamic memory array (task local)
13: C o   IG(NS) : energy group sampled
14: C i   ns : number of particles whose group IG is sampled
15: C i   imat : Rmaterial #
16: C o   ierr : error code #
17: C      irnd : dummy (IRAND is given in _SIZES)
18: C w   rwk(NS,5) : working array
19: C-----
20: C
21:      real A(*)
22:      real H(*)
23: C
24:      real*8 IG(NS)
25: C
26:      real RWK(NS,5)
27: C
28:      include 'INC/_PXSEC'
29:      include '../shared/INC/_SIZES'
30:
31:      IERR      = 0
32: C
33:      call MATFIS( A(LFKAI), A(LKKAI), NTGX, NMAT, IG, NS, IMAT, IRAND,
34:      &            RWK )
35: C
36:      return
37:      end
38:
39:      subroutine MATFIS( FKAI, KKAI, NTGX, NMAT, IG, NS, IMAT,
40:      &                  IRAND, R )
41: C=<GMVP>=====
42: C PURPOSE: sampling energy group from fission spectrum in GMVP.
43: C CALLED IN : NUCFIS
44: C CALLS : none
45: C-----
46: C
47:      real*8 IG(NS)
48:      real R(NS*5)
49: C
50: C .... FISSION NEUTRON DATA .....
51: C
52:      integer KKAI(2,NTGX,NMAT)
53:      real FKAI(NTGX,NMAT)
54: C
55: C
56:      call RANU2( IRAND, R, 2*NS, ICON )
57: C
58:      do 100 I = 1, NS
59: C/#IF ROUND OFF(NEAREST)
60:          KK      = MIN( INT( NTGX*R(I) ) + 1, NTGX )
61:          KKK      = 2*KK
62:          KKK      = MIN( KKK, INT( KKK + R( NS + I ) - FKAI( KK, IMAT ) ) )
63: C/#ELSE
64:          *        KK      = NTGX*R(I) + 1
65:          *        KKK      = KK*2 + R( NS + I ) - FKAI( KK, IMAT )
66: C/#ENDIF
67:
68:          IGG      = KKAI( KKK, 1, IMAT )
69:          IG(I)    = IGG
70:      100 continue
71:      return
72:      end

```

src/gmvp/nuclst.f

```
1:      subroutine NUCLST( NAME,  INUCS, NN,      NNNUCS,A,      CHA,  
2:      &      IERR )  
3: C=====
```

4: C Purpose : Dummy routine for mvp/chsymb. GMVP doesn't require this
5: C routine for actual calculations but for compilation.
6: C This routine is required since the photonuclear capability
7: C was added.
8: C Called in: STLPR?

```
9: C=====
```

10: write(6,*) 'XXX(NUCLST) GMVP does not use this routine.'

```
11:      stop  
12:      end  
13:
```

14: subroutine CKPNMT(IWRK, ND, II, IA, IERR)
15: C=====

16: C Purpose : Dummy routine for mvp/chsymb. GMVP doesn't require this
17: C routine for actual calculations but for compilation.
18: C This routine is required since the photonuclear capability
19: C was added.
20: C Called in: STLPR?

```
21: C=====
```

22: write(6,*) 'XXX(CKPNMT) GMVP does not use this routine.'

```
23:      stop  
24:      end  
25:
```

src/gmvp/nxtec.f

```

1:      SUBROUTINE NXTEC( IOW, A, H, LSCOL,
2:      B          XXX , YYY , ZZZ , AAA , BBB , CCC ,
3:      B          WWW , IZZ , IGG , TTT , ITT ,
4:      B          LEVL , LZZ , LPOS , LCRS ,
5:      B          DBNK , KDBNK , MDBNK , IBNK , KIBNK , MIBNK,
6:      B          NDBNK , NIBNK,
7:      B          LZZI , LPOSI , LCRSI , NCNTR , WCNTR,
8:      B          OPTI , PATH , KDETP , KLSFI,
9:      X          ANGX , NGSX , NNNX , STOTX , SGPBX , IGPBX,
10:     X          SSCBX , ISCBX , SDDBX , IDDBX , FKAI , KKAI ,
11:     X          PANGX , SPORT , S2DPX , S2PPX , VEL ,
12:     C          KZMAT , KZREG , TIMEB , XPDET , IPDET , IPDT2,
13:     D          JPUSD,
14:     S          IMSFL , IMNFL , IMZFL,
15:     S          IMDED , IMSLT , IMZLT , IMNLT,
16:     W          X , Y , Z , A0 , B0 , C0 ,
17:     W          AA , BB , CC , DI , WW ,
18:     W          MATWK , IGIN , ISEL , IGO ,
19:     W          IWK1 , IWK5 , IWK6 , IWK7 , IWK9 , IWK10,
20:     W          COSL , RP , R , W , IT0 , TI )
21: C=====
22: C
23: C PURPOSE: SELECTION COLLISION POINTS FOR NEXT EVENT ESTIMATORS &
24: C CALL THE ESTIMATION MODULE
25: C CALLED IN: ACTION
26: C CALLS: VMNTRI,RANU2,LATU2,LATDW2
27: C=====
28: C
29: C === INTER-STACK DATA FLOW ===
30: C
31: C COLLISION STACK -----> FREE-FLIGHT STACK FOR IMAGINARY PARTICLE
32: C (LSCOL,NCOLS) (IMSFL,IMZFL,IMNFL) (IMSLT,IMZLT,IMNLT)
33: C (NOT REMOVED)
34: C
35: C=====
36: C implicit real*8 (A-H,O-Z)
37: C
38: C real A(*)
39: C real H(*)
40: C
41: C include '../shared/INC/_SIZES'
42: C include '../shared/INC/_PGEOM'
43: C include 'INC/_SIZES2'
44: C include 'INC/_FLAGS'
45: C include 'INC/_STACK'
46: C include 'INC/_PXSEC'
47: C include '../shared/INC/_PTALY0'
48: C include 'INC/_PTALY'
49: CCCCC include 'INC/_WKNXC'
50: C
51: C**** DETECTOR DATA FOR NEXT EVENT (POINT) ESTIMATOR
52: C
53: C real*8 XPDET(NPLEN,NPDET)
54: C integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
55: C integer JPUSD(NPDET)
56: C
57: C**** TALLY BIN DATA
58: C REAL RESP(NGROUP,NRESP)
59: C real TIMEB(*), VEL(*)
60: C
61: C**** GEOMETRY ARRAY DATA
62: C
63: C integer KZMAT(NZONE,2), KZREG(NZONE)
64: C
65: C**** CROSS SECTION DATA

```

```

66: C
67: C real ANGX(*), STOTX(NTGX,*), SGPBX(NGGX,3,NGPX,*), SSCBX(*),
68: C & SDDBX(*), PANGX(NSCTX,NTGX,NMEDX), SPORT(NDSTX,NMEDX,*),
69: C & FKAI(NTGX,*), S2DPX(2,NTGX,NMEDX), S2PPX(3,NTGX,NMEDX)
70: C integer NGSX(*), NNNX(*), IGPBX(NGGX,3,NGPX,*), ISCBX(*),
71: C & IDDBX(*), KKAI(2,NTGX,*)
72: C
73: C ..... ISCBX & IDDBX HAVE THE SAME ADDRESSES AS SSCBX & SDDBX .....
74: C IANGX HAVE THE SAME ADDRESSES AS SANGX .....
75: C PANGX HAVE THE SAME ADDRESSES AS SPORT .....
76: C
77: C**** STACK FOR IMAGINARY PARTICLE
78: C
79: C integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(NZONE+1),
80: C & IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
81: C
82: C**** STACK FOR REAL PARTICLE
83: C
84: C integer LSCOL(NBANK)
85: C
86: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
87: C
88: C real*8 XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2), AAA(NBANK,2),
89: C & BBB(NBANK,2), CCC(NBANK,2)
90: C
91: C real*8 TTT(NBANK,2)
92: C real WWW(NBANK,2)
93: C
94: C integer IZZ(NBANK,2), IGG(NBANK,2), ITT(NBANK,2), LEVL(NBANK,2),
95: C & LZZ(NBANK,NEST), LPOS(NBANK,NEST), LCRS(NBANK,NEST),
96: C & LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
97: C
98: C real*8 DBNK(NBANK,NDBNK,2)
99: C integer KDBNK(0:MDBNK)
100: C integer IBNK(NBANK,NIBNK,2)
101: C integer KIBNK(0:MIBNK)
102: C****
103: C real*8 OPTI(IMPMAX), PATH(IMPMAX)
104: C integer KDETP(IMPMAX), KLSFI(IMPMAX)
105: C
106: C**** TALLY (MONITOR) ARRAY
107: C
108: C real*8 WCNTR(*), NCNTR(*)
109: C
110: C**** WORKING AREA (THESE AREA CAN BE DESTROYED IN OTHER ROUTINES
111: C (IWK1,IWK5),(IWK6,IWK7),(IWK9,IWK10),(COSL,RP)
112: C (R ) are available as real*8 work area
113: C
114: C integer IWK1(NBANK), IWK5(NBANK), IWK6(NBANK), IWK7(NBANK),
115: C & IWK9(NBANK), IWK10(NBANK), IT0(NBANK)
116: C real COSL(NBANK), RP(NBANK), R(2*NBANK)
117: C
118: C**** WORK AREA ( CANNOT BE DESTROYED IN SUB-PROCESS (NXTEE) )
119: C
120: C integer MATWK(NBANK), IGIN(NBANK), ISEL(NBANK), IGO(NBANK)
121: C real*8 X(NBANK), Y(NBANK), Z(NBANK)
122: C real*8 A0(NBANK), B0(NBANK), C0(NBANK),
123: C & AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
124: C real W(NBANK), WW(NBANK)
125: C real*8 TI(NBANK)
126: C
127: C 0..NBK.2NBK.3NBK.4NBK.5NBK.6NBK.7NBK.8NBK.9NBK,10NBK,11NBK
128: C IWK1 MATWK IGIN ISEL IWK5 IWK6 IWK7 IGO IWK9 RP R,IWORK
129: C****
130: C

```

src/gmvp/nxtec.f

```

131: C
132:   parameter( PI2 = 2.0D0*3.1415926535897933D0 )
133:   parameter( PI4I = 1.0D0/(4*3.1415926535897933D0 ) )
134: C
135: C
136:   NSCT1 = NSCTX + 1
137:   NSCT3 = NSCTX*3
138:   NSCT3N = NSCT3*NDSTX
139:   NDST3 = NDSTX*3
140:   NPL = NCOEFX - 1
141: C
142: C
143: C**** GATHER PARTICLE DATA *****
144: C
145:   if ( JLATT.eq.0 ) then
146: C ..... NO LATTICE
147:   do 100 I = 1, NCOLS
148:     X(I) = XXX(LSCOL(I),1)
149:     Y(I) = YYY(LSCOL(I),1)
150:     Z(I) = ZZZ(LSCOL(I),1)
151:     A0(I) = AAA(LSCOL(I),1)
152:     B0(I) = BBB(LSCOL(I),1)
153:     C0(I) = CCC(LSCOL(I),1)
154:     IZ = IZZ(LSCOL(I),1)
155: C     IWK1(I) = KZREG(IZ)
156:     MATWK(I) = KZMAT(IZ,1)
157:     IGIN(I) = IGG(LSCOL(I),1)
158:     W(I) = WWW(LSCOL(I),1)*PI4I*S2DPX(1,IGIN(I),MATWK(I))
159: CTEMP ***** W(I) = WWW(LSCOL(I),1)*PI4I
160:   100 continue
161: C
162:   else
163: C
164: C ..... LATTICE
165: C ***** TRANSFORMATION OF CORDINATES AND DIRECTION
166: C FROM CELL-ORDINATE SYSTEM TO LABORATORY SYSTEM
167: C (XXX,YYY,ZZZ)&(AAA,BBB,CCC) --> (X,Y,Z)&(A0,B0,C0)
168: C
169:   JDIR = 1
170:   JLS = 0
171:   call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NNBZ,
172: & NZONE, NCOLS, NBANK, JDIR, JLS, LSCOL,
173: & X, Y, Z, A0, B0, C0,
174: & XXX, YYY, ZZZ,
175: & AAA, BBB, CCC,
176: & LEVL, LZZ, LPOS, LCRS,
177: & DBNK,KDBNK,MDBNK,
178: & A(LKZMAT),A(LKDALT),A(LDALT),A(LNVLAT),A(LSZLAT),
179: & A(LCELSZ),A(LIPLAT),A(LKLATT),A(LKSLAT),A(LLTYP),
180: & A(LICTYP),A(LMLBZZ),
181: & IWK1 )
182: check
183: C   write(6,*) '##### after latup2, ncols=',ncols
184: C   do 1112 i=1,ncols
185: C     write(6,*) '#xxx=',xxx(lscol(i),1),
186: C & ' yyy=',yyy(lscol(i),1),
187: C & ' zzz=',zzz(lscol(i),1)
188: C   write(6,*) ' aaa=',aaa(lscol(i),1),
189: C & ' bbb=',bbb(lscol(i),1),
190: C & ' ccc=',ccc(lscol(i),1)
191: C   write(6,*) ' igg :',igg(lscol(i),1),' izz :',izz(lscol(i),1),
192: C & ' levl:',levl(lscol(i),1)
193: C   write(6,*) ' lzz:',(lzz(lscol(i),1),l=1,levl(lscol(i),1))
194: C   write(6,*) ' lpos:',(lpos(lscol(i),1),l=1,levl(lscol(i),1))
195: C   if ( KDBNK(5).ne.0) then

```

```

196: C     do 1113 l=1,levl(lscol(i),1)
197: C       if(lpos(lscol(i),1).eq.0) then
198: C         k5x = kdbnk(5)+(l-1)*3
199: C         write(6,*) ' l=',l,
200: C & ' x,y,z=',(dbnk(lscol(i),k,1),k=k5x,k5x+2)
201: C       end if
202: C1113 continue
203: C     end if
204: C   write(6,*) ' x=',x(i),
205: C & ' y=',y(i),
206: C & ' z=',z(i)
207: C   write(6,*) ' a=',a0(i),
208: C & ' b=',b0(i),
209: C & ' c=',c0(i)
210: C1112 continue
211: C
212:   do 110 I = 1, NCOLS
213:     IZ = IZZ(LSCOL(I),1)
214: C     IWK1(I) = KZREG(IZ)
215:     MATWK(I) = KZMAT(IZ,1)
216:     if ( MATWK(I).lt.0) MATWK(I) = KZMAT(IZ,2)
217:     IGIN(I) = IGG(LSCOL(I),1)
218:     W(I) = WWW(LSCOL(I),1)*PI4I*S2DPX(1,IGIN(I),MATWK(I))
219:   110 continue
220:   end if
221: C
222: C
223: C
224: C
225: C-----<< DO - LOOP INDEX DEFINITION HEREAFTER >> -----
226: C
227: C
228: C
229: C I : INDICATE PARTICLES IN COLLISION STACK.
230: C
231: C << VARIABLES REFERENCED BY INDEX 'I' >>
232: C
233: C MATWK(I) : MATERIAL (MEDIA) # AT COLLISION POINT
234: C IGIN(I) : INCIDENT ENERGY GROUP
235: C X(I),Y(I),Z(I) : COLLISION POINTS
236: C W(I) : PARTICLE WEIGHTS
237: C
238: C
239: C J : INDICATE PARTICLES TO EMIT AN IMAGINARY PARTICLE
240: C TOWARD ND'TH DETECTOR.
241: C (SELECTED BY RUSSIAN ROULETTE AT THE BEGINNIG OF
242: C DETECTOR LOOP)
243: C
244: C < VARIABLES REFERENCED BY INDEX 'J' >
245: C
246: C ISEL(J) : INDICATER OF SELECTED PARTICLES IN COLLISION STACK.
247: C ( VARIABLES NATIVE TO I-INDEX ARE REFFRENCED AS
248: C MATWK(ISEL(J)) ETC. )
249: C
250: C AA(J),BB(J),CC(J) : FLIGHT VECTORS OF EMITTED IMAGINARY
251: C PARTICLES.
252: C WW(J) : PARTICLE WEIGHT.
253: C IGO(J) : ENERGY GROUP OF IMAGINARY PARTICLES.
254: C
255: C
256: C**** LOOP OVER DETECTOR (NPDET) *****
257: C
258: C
259: C
260: C

```

src/gmvp/nxtec.f

```

261:      do 360 ND = 1, NPDET
262: C
263:          if ( JPUSD(ND).eq.0 ) go to 360
264: C
265: C=====
266: C
267: C      CALCULATE DISTANCES TO ND'TH DETECTOR
268: C      & SELECT COLLISIONS COUNTED AT THE DETECTOR
269: C
270: C=====
271: C
272: C
273: C
274: C
275: C      XDA      = XPDET(3,ND)
276: C      YDA      = XPDET(4,ND)
277: C      ZDA      = XPDET(5,ND)
278: C980630 RMAX    = XPDET(ND,7)
279: C      RMAX      = XPDET(2,ND)
280: C      RMA2      = RMAX*RMAX
281: C980630 RMIN    = XPDET(ND,8)
282: C      RMIN      = XPDET(1,ND)
283: C      RMIN2     = RMIN*RMIN
284: C
285: C      call RANU2( IRAND, R, NCOLS, ICON )
286: C
287: C NN : NUMBER OF SELECTED COLLISION POINTS
288: C NBS : NUMBER OF SELECTED COLLISION POINTS WHICH REQUIRE BOUNDED SPHERE
289: C
290: C      NN      = 0
291: C      NBS     = 0
292: C
293: C      do 120 I = 1, NCOLS
294: C          AT      = XDA - X(I)
295: C          BT      = YDA - Y(I)
296: C          CT      = ZDA - Z(I)
297: C          D        = AT*AT + BT*BT + CT*CT
298: C
299: C          ..... CALCULATE CONTRIBUTION WITH PROBABILITY RMIN2/D
300: C          WHEN D > RMIN2.
301: C
302: C          if ( D.le.RMIN2 .or. D*R(I).le.RMIN2 ) then
303: C              NN      = NN + 1
304: C              ISEL(NN) = I
305: C              DI(NN)  = D
306: C              AA(NN)  = AT
307: C              BB(NN)  = BT
308: C              CC(NN)  = CT
309: C              WW(NN)  = W(I)
310: C          end if
311: C120      continue
312: C
313: C      do 130 J = 1, NN
314: C
315: C          ..... THE BOUNDED SPHERE APPROXIMATION IF DI < PMA2**2 ...
316: C
317: C          if ( DI(J).lt.RMA2 ) then
318: C              NBS      = NBS + 1
319: C
320: C
321: C          ..... DEVIDE WEIGHT BY (DISTANCE)**2 HERE ....
322: C          ..... WEIGHT CORRECTION NECESSARY IF DI > RMIN2 ...
323: C
324: C              MEANING : WW(J) = WW(J) / DI(J) *1/(RMIN2/DI(J))
325: C

```

```

326: C
327: C      else if ( DI(J).gt.RMIN2 ) then
328: C          WW(J)      = WW(J) /RMIN2
329: C      else
330: C          WW(J)      = WW(J) /DI(J)
331: C      end if
332: C
333: C      .... 'DI' MEANS DISTANCE ITSELF HEREAFTER. ....
334: C
335: C
336: C          DI(J)      = SQRT(DI(J))
337: C
338: C      .... (AA,BB,CC) IS NORMALIZED DIRECTION VECTOR HEREAFTER ...
339: C
340: C
341: C      if ( DI(J).gt.0. ) then
342: C          AA(J)      = AA(J) /DI(J)
343: C          BB(J)      = BB(J) /DI(J)
344: C          CC(J)      = CC(J) /DI(J)
345: C      else
346: C          AA(J)      = 0.
347: C          BB(J)      = 0.
348: C          CC(J)      = 0.
349: C      end if
350: C130      continue
351: C
352: C
353: C
354: C
355: C=====
356: C
357: C      SELECTION OF REACTIONS COUNTED TO DETECTOR
358: C
359: C=====
360: C
361: C
362: C      IWK5 : SCATTERING (NS)   GIVING J
363: C      IWK6 : FISSION-N (NF)   GIVING J
364: C      IWK7 : PHOTON (NP)      GIVING J
365: C
366: C
367: C      NS      = 0
368: C      NF      = 0
369: C      NP      = 0
370: C
371: C      if ( (JEIGN.ne.0.or.JFISS.eq.0).and.JPHOT*JNEUT.eq.0 ) then
372: C
373: C          ..... CONTRIBUTIONS FROM SCATTERING ONLY ....
374: C
375: C          do 140 J = 1, NN
376: C              IWK5(J) = J
377: C140          continue
378: C          NS      = NN
379: C
380: C      else
381: C
382: C          CALL RANU2( IRAND, R, NCOLS, ICON )
383: C
384: C
385: C          ..... CASE OF NO-FISSION (CONTRIBUTIONS OF FISSION-NEUTRON
386: C          IN EIGENVALUE CALCULATIONS ARE NOT CALCULATED HERE.)
387: C
388: C          if ( JEIGN.ne.0 .or. JFISS.eq.0 ) then
389: C
390: C              call RANU2( IRAND, R, NN, ICON )

```

src/gmvp/nxtec.f

```

391:         do 150 J = 1, NN
392:             I = ISEL(J)
393:             if ( R(J).le.S2PPX(3,IGIN(I),MATWK(I)) ) then
394:                 NS = NS + 1
395:                 IWK5(NS) = J
396:             else
397:                 NP = NP + 1
398:                 IWK7(NP) = J
399:             end if
400:         150 continue
401: C
402: C ..... CONTRIBUTION FROM SCATTERING, FISSION & GAMMA PRODUCTION
403: C MUST BE TAKEN IN ACCOUNT FOR THIS CASE.
404: C
405: C BEWARE THAT ONLY ONE PARTICLE IS ALLOWED TO FLY TO A DETECTOR
406: C
407: C
408: C     else
409: C         call RANU2( IRAND, R, NN, ICON )
410: C         do 160 J = 1, NN
411: C             I = ISEL(J)
412: C
413: C         >>>> SCATTERING
414: C
415: C             if ( R(J).le.S2PPX(1,IGIN(I),MATWK(I)) ) then
416: C                 NS = NS + 1
417: C                 IWK5(NS) = J
418: C
419: C             >>>> PHOTON GENERATION
420: C
421: C                 else if ( R(J).le.S2PPX(2,IGIN(I),MATWK(I)) ) then
422: C                     NP = NP + 1
423: C                     IWK7(NP) = J
424: C
425: C                 >>>> FISSION
426: C
427: C                     else
428: C                         NF = NF + 1
429: C                         IWK6(NF) = J
430: C                     end if
431: C                 160 continue
432: C
433: C             end if
434: C         end if
435: C
436: C         MATWK(I): MEDIA NUMBER, IGIN(I): INCIDENT ENERGY GROUP
437: C         IGO(J): OUTGOING ENERGY GROUP
438: C
439: C
440: C
441: C=====
442: C
443: C     SELECTION OF OUTGOING ENERGY FOR FISSION NEUTRONS
444: C
445: C=====
446: C
447: C
448: C
449: C
450: C
451: C     ..... SAMPLING BY FISSION-SPECTRUM .....
452: C
453: C
454: C
455: C         if ( NF.ne.0 ) then

```

```

456:         call RANU2( IRAND, R, 2*NF, ICON )
457: C*VOCL LOOP,NOVREC
458: C         do 170 JJ = 1, NF
459: C             J = IWK6(JJ)
460: C             I = ISEL(J)
461: C         C/#IF ROUNDOFF(NEAREST)
462: C             LL = MIN( INT(NTGX*R(JJ))+1,NTGX)
463: C         C/#ELSE
464: C             LL = NTGX*R(JJ)+1
465: C         C/#ENDIF
466: C             RR = R(NF+JJ) - FKAI(LL,MATWK(I))
467: C         c97/11/07 LL2 = 2 + RR
468: C         C/#IF ROUNDOFF(NEAREST)
469: C             LL2 = min(2,int(2 + RR))
470: C         C/#ELSE
471: C             LL2 = 2 + RR
472: C         C/#ENDIF
473: C             IGO(J) = KKAI(LL2,LL,MATWK(I))
474: C         170 continue
475: C         end if
476: C
477: C
478: C
479: C     ..... OUTGOING ENERGY FOR PHOTON .....
480: C
481: C
482: C
483: C
484: C         if ( NP.ne.0 ) then
485: C             call RANU2( IRAND, R, 2*NP, ICON )
486: C*VOCL LOOP,NOVREC
487: C             do 180 JJ = 1, NP
488: C                 J = IWK7(JJ)
489: C                 I = ISEL(J)
490: C
491: C             C/#IF ROUNDOFF(NEAREST)
492: C                 LL = MIN( INT(NGGX*R(JJ))+1,NGGX)
493: C             C/#ELSE
494: C                 LL = NGGX*R(JJ)+ 1
495: C             C/#ENDIF
496: C                 RR = R(NP+JJ) - SGPBX(LL,1,IGIN(I),MATWK(I))
497: C             LL2 = 2*LL + NGGX
498: C         c97/11/07 LL3 = LL2 + RR
499: C         C/#IF ROUNDOFF(NEAREST)
500: C             LL3 = min(LL2,int(LL2 + RR))
501: C         C/#ELSE
502: C             LL3 = LL2 + RR
503: C         C/#ENDIF
504: C             IGO(J) = IGPBX(LL3,1,IGIN(I),MATWK(I))
505: C         180 continue
506: C         end if
507: C
508: C
509: C
510: C     ..... OUTGOING ENERGY FOR SCATTERING-NEUTRON .....
511: C
512: C
513: C
514: C
515: C         if ( NS.gt.0 ) then
516: C             ..... CALCULATION OF SCATTERING ANGLE
517: C             ASSUME COSL=0.0 FOR PARTICLES WITH DI(J)=0.0
518: C
519: C             do 190 JJ = 1, NS
520: C                 J = IWK5(JJ)

```

src/gmvp/nxtec.f

```

521:          I          = ISEL(J)
522:          COSL(JJ)    = MIN(1.0D0, MAX(-1.0D0,
523:          &          A0(I)*AA(J)+B0(I)*BB(J)+C0(I)*CC(J) ))
524: 190      continue
525: C
526:          if ( JDDX.ne.0 ) then
527: C
528: C          ***** CASE OF DDX
529: C
530: C
531: C          ..... FIND ANGLE BINS -----> IWK6(JJ)
532: C
533: C
534: C          call BSVDEC( ANGX, NSCT3, COSL, IWK6, NS )
535: C
536: C          ..... SELECT OUTGOING ENERGY
537: C
538: C
539: C          MATWK(I): MEDIA NUMBER, IGIN(I): INCIDENT ENERGY GROUP
540: C          IGO(J): OUTGOING ENERGY GROUP
541: C
542: C          call RANU2( IRAND, R, NS*2, ICON )
543: *VOCL LOOP,NOVREC
544:          do 200 JJ = 1, NS
545:              J          = IWK5(JJ)
546:              I          = ISEL(J)
547:              NNNX1      = NNNX(IGIN(I))
548:              L0         = (MATWK(I)-1)*NSCT3N + NGSX(IGIN(I))*NSCT3
549:              &          + (IWK6(JJ)-1)*NNNX1*3
550:
551: C/#IF ROUNDOFF(NEAREST)
552:              L1         = MIN(INT(NNNX1*R(JJ))+1,NNNX1)
553: C/#ELSE
554:              *          L1         = NNNX1*R(JJ) + 1
555: C/#ENDIF
556:              RR         = R(JJ+NS) - SDCBX(L1+L0)
557:              L2         = 2*L1 + NNNX1 + L0
558: c97/11/07          L3         = L2 + RR
559: C/#IF ROUNDOFF(NEAREST)
560:              L3         = min(L2,int(L2 + RR))
561: C/#ELSE
562:              *          L3         = L2 + RR
563: C/#ENDIF
564:              IGO(J)     = IDDBX(L3) + IGIN(I) - NUSX - 1
565: C
566: C
567: C          ..... WEIGHT CORRECTION FOR ANGULAR DISTRIBUTION
568: C
569: C
570: C          WW(J)        = WW(J)*PANGX(IWK6(JJ),IGIN(I),MATWK(I))
571: 200      continue
572: C
573: C          else
574: C
575: C
576: C          ***** CASE OF PL EXPANSION
577: C
578: C
579: C          call RANU2( IRAND, R, NS*2, ICON )
580: *VOCL LOOP,NOVREC
581:          do 210 JJ = 1, NS
582:              J          = IWK5(JJ)
583:              I          = ISEL(J)
584:              IGOT        = IGIN(I)
585: C          ..... USED ALSO IN THE FOLLOWING LOOP

```

```

586:          IWK1(JJ)      = NGSX(IGOT)
587:          IWK6(JJ)      = MATWK(I)
588: C
589: C          L0          = (IWK6(JJ)-1)*NDST3 + IWK1(JJ)*3
590: C          NNNX1       = NNNX(IGOT)
591: C
592: C/#IF ROUNDOFF(NEAREST)
593:              L1         = MIN(INT(NNNX1*R(JJ))+1,NNNX1)
594: C/#ELSE
595:              *          L1         = NNNX1*R(JJ) + 1
596: C/#ENDIF
597:              RR         = R(JJ+NS) - SSCBX(L1+L0)
598:              L2         = 2*L1 + NNNX1 + L0
599: c97/11/07          L3         = L2 + RR
600: C/#IF ROUNDOFF(NEAREST)
601:              L3         = min(L2,int(L2 + RR))
602: C/#ELSE
603:              *          L3         = L2 + RR
604: C/#ENDIF
605:              IGD        = ISCBX(L3)
606: C          Cccccccccccccccccccccc IGO(J) = IGD + IWK7(JJ) - NUSX - 1
607:              IGO(J)     = IGD + IGOT - NUSX - 1
608:              IWK1(JJ)    = IWK1(JJ) + IGD
609: 210      continue
610: C
611: C          if ( NPL.gt.0 ) then
612:              do 220 JJ = 1, NS
613:                  RP(JJ)  = COSL(JJ)*SPORT(IWK1(JJ),IWK6(JJ),1) + 1.0
614: 220      continue
615:              if ( NPL.gt.1 ) then
616:                  do 230 JJ = 1, NS
617:                      R(JJ) = COSL(JJ)
618:                      R(JJ+NS) = 1.5*COSL(JJ)*COSL(JJ) - 0.5
619:                      RP(JJ) = RP(JJ) + R(JJ+NS)*
620:                      &          SPORT(IWK1(JJ),IWK6(JJ),2)
621: 230      continue
622: *VOCL LOOP,NOVREC
623:              do 250 IPL = 3, NPL
624:                  do 240 JJ = 1, NS
625:                      P2      = R(JJ+NS)
626:                      P3      =
627:                      &          ((2.*(IPL-1)+1.)*COSL(JJ)*P2-(IPL-1)*
628:                      &          R(JJ)) /IPL
629:                      RP(JJ)  = RP(JJ)
630:                      &          + P3*SPORT(IWK1(JJ),IWK6(JJ),IPL)
631:                      R(JJ)   = P2
632:                      R(JJ+NS) = P3
633: 240      continue
634: 250      continue
635:              end if
636: C
637: *VOCL LOOP,NOVREC
638:              do 260 JJ = 1, NS
639:                  J          = IWK5(JJ)
640:                  WW(J)      = WW(J)*RP(JJ)
641: 260      continue
642:              end if
643: C
644:              end if
645: C
646:              end if
647: C
648: C          .... ENERGY CUT OFF .....
649: C
650: C          if ( NTGX.gt.NGROUP ) then

```


src/gmvp/nxtec.f

```

651:      NNN      = 0
652: *VOCL LOOP,NOVREC
653:      do 270 J = 1, NN
654:        if ( IGO(J).le.NGROU ) then
655:          NNN      = NNN + 1
656:          ISEL(NNN) = ISEL(J)
657:          IGO(NNN)  = IGO(J)
658:          WW(NNN)   = WW(J)
659:          DI(NNN)   = DI(J)
660:          AA(NNN)   = AA(J)
661:          BB(NNN)   = BB(J)
662:          CC(NNN)   = CC(J)
663:        end if
664:      continue
665: C
666:      NN      = NNN
667:    end if
668: C
669: C .... TIME CUT OFF .....
670: C
671:      if ( JTIME.ne.0 ) then
672:        if ( JPTIM.eq.0 ) then
673:          NNN      = 0
674: *VOCL LOOP,NOVREC
675:          do 280 J = 1, NN
676:            TTTT    = TTT(LSCOL(ISEL(J)),1) + DI(J) /VEL(IGO(J))
677: C          if ( TI.le.TIMEB(NTIME+1) ) then
678:            if ( TTTT.le.TCUT ) then
679:              NNN      = NNN + 1
680:              ISEL(NNN) = ISEL(J)
681:              IGO(NNN)  = IGO(J)
682:              WW(NNN)   = WW(J)
683:              DI(NNN)   = DI(J)
684:              AA(NNN)   = AA(J)
685:              BB(NNN)   = BB(J)
686:              CC(NNN)   = CC(J)
687:              TI(NNN)   = TTTT
688:            end if
689:          continue
690: C
691:          NN      = NNN
692:        else
693:          do 290 J = 1, NN
694:            TTTT    = TTT(LSCOL(ISEL(J)),1) + DI(J) /VEL(IGO(J))
695:            if ( TTTT.ge.TCUT ) then
696:              ITREP = TTTT / TCUT
697:              TI(J) = TTTT - TCUT*ITREP
698:            end if
699:          continue
700:        end if
701: C
702:        call BS0ISD( TIMEB, NTIME+1, TI, IT0, NN )
703: C
704:        do 300 J = 1, NN
705:          IT0(J) = MAX(1,MIN(NTIME,IT0(J)))
706:        continue
707: C
708:      end if
709: C
710: C
711: C
712: C .... BOUNDED SPHERE APPROXIMATION
713: C
714: C
715: C

```

```

716:      if ( NBS.gt.0 ) then
717: *VOCL LOOP,NOVREC
718:      do 310 J = 1, NN
719: *VOCL STMT,IF(0)
720:        if ( DI(J).le.RMAX ) then
721:          I      = ISEL(J)
722:          RMD    = STOTX(IGO(J),MATWK(I))*RMAX
723:          RMI    = 1./RMD
724: C980630 CCCCCC C2      = RMA2*(RMI*RMI*2.+(1.+RMI)/(1.-EXP(RMD)))
725:          C2      = RMA2*(RMI*RMI*2.+(1.+2.0*RMI)/(1.-EXP(RMD)))
726:          WW(J)   = WW(J) /C2
727:        end if
728:      310      continue
729:        end if
730: C
731: C
732: C=====
733: C
734: C      SEND PARTICLES TO STACK FOR IMAGINARY PARTICLES
735: C
736: C=====
737: C
738: C
739:          IZD      = IPDET(ND,1)
740:          LVL      = IPDET(ND,2)
741:          if ( JLATT.ne.0.and.LVL.gt.0 ) then
742:            if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
743:            LP      = 2 + 3*LVL
744:            XDA      = XPDET(LP+1,ND)
745:            YDA      = XPDET(LP+2,ND)
746:            ZDA      = XPDET(LP+3,ND)
747:          end if
748: C
749:      320      continue
750: C
751:          if ( NN.gt.0 ) then
752: C            NNN      = MIN(NN,NDIMPT)
753: C
754: C            if ( NNN.gt.0 ) then
755:              NDIMPT = NDIMPT - NNN
756:              N      = IMNFL(NZONE+1)
757:              N1     = NN - NNN
758: *VOCL LOOP,NOVREC
759:              do 330 J = 1, NNN
760:                J1    = N1 + J
761:                IMSFL(N+J) = IMDED(NDIMPT+J)
762:                IMZFL(N+J) = IZD
763: C
764: C
765: C ---- BEWARE THAT IMAGINARY PARTICLES FLY FROM THE DETECTOR POINT
766: C      TO COLLISION POINTS. ----
767: C
768:                XXX(IMSFL(N+J),2) = XDA
769:                YYY(IMSFL(N+J),2) = YDA
770:                ZZZ(IMSFL(N+J),2) = ZDA
771:                AAA(IMSFL(N+J),2) = -AA(J1)
772:                BBB(IMSFL(N+J),2) = -BB(J1)
773:                CCC(IMSFL(N+J),2) = -CC(J1)
774:                WWW(IMSFL(N+J),2) = WW(J1)
775:                IGG(IMSFL(N+J),2) = IGO(J1)
776:                TTT(IMSFL(N+J),2) = TTT(LSCOL(ISEL(J1)),1)
777:                PATH(IMSFL(N+J)) = DI(J1)
778:                OPTI(IMSFL(N+J)) = 0.0
779:                KDETP(IMSFL(N+J)) = ND
780:                KLSFI(IMSFL(N+J)) = 0

```

src/gmvp/nxtec.f

```

781:         if ( JTIME.ne.0 ) then
782:             ITT(IMSFL(N+J),2) = IT0(J1)
783:         end if
784:         if ( KIBNK(5).ne.0 ) then
785:             IBNK(IMSFL(N+J),KIBNK(5),2) = -1
786:         end if
787: 330      continue
788: C
789:         NN2 = NNN
790:         JSTG = 0
791: C
792: C=====
793: C
794: C      In LATTICE geometry, the following parameters should be set.
795: C      LEVL, LZZI, LPOSI, LCRSI
796: C      DBNK, IBNK (JFISX.ne.0)
797: C      JSTG : flag to indicate the detector is located in STG region.
798: C      NN2 : number of remaining particles when JFISX.ne.0.
799: C
800: C=====
801: C
802:         if ( JLATT.ne.0 ) then
803:             call LATDW2(IOW, JDEBS, JHLAT, JFESX,
804: N             IMPMAX,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
805: N             DINF ,DEPS ,
806: S             IMSFL(N+1),NNN ,JSTG ,NN2 ,
807: I             IPDET(ND,2),IPDT2(1,1,ND),IPDT2(1,2,ND),
808: I             IPDT2(1,3,ND),XPDET(3,ND),
809: B             AAA(1,2),BBB(1,2),CCC(1,2) ,
810: B             LEVL(1,2),LZZI ,LPOSI ,LCRSI ,
811: B             DBNK(1,1,2),KDBNK ,MDBNK ,IBNK(1,1,2),KIBNK ,MIBNK,
812: S             IMDED ,NDIMPT,
813: G             A(LSDA) ,A(LKZDA) ,A(LKZAA) ,
814: G             A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LMLBZZ),A(LCLLSZ),
815: G             A(LSZLAT),A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),
816: G             A(LNVLAT),A(LIPCEL),A(LLTYP) ,A(LICTIP),A(LKDALT),
817: G             A(LDALT) ,A(LKZREG),
818: W             IWK1 ,IWK6 ,IWK10 ,AA(N1+1),BB(N1+1),CC(N1+1),
819: W             DI(N1+1),COSL ,R ,WW(N1+1),IGO(N1+1) )
820:         end if
821: C
822:         if ( JSTG.eq.0 ) then
823:             IMNFL(IZD) = IMNFL(IZD) + NN2
824:             IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
825: C
826: C ... detector is located in STG region. move to LATTICE stack.
827:         else
828:             NL = IMNLT(NLBZ+1)
829:             do 340 J = 1, NN2
830:                 IMSLT(NL+J) = IMSFL(N+J)
831:                 IMZLT(NL+J) = IZD
832: 340      continue
833:             IMNLT(IZD) = IMNLT(IZD) + NN2
834:             IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
835:         end if
836: C
837:         NN = NN - NNN
838:         NIMPT = NIMPT + NN2
839:     end if
840: check
841: C      write(6,*) '***(NXTEC) nn,nnn,nl,nn2,jstg=',nn,nnn,nl,nn2,jstg
842: C      write(6,*) ' nimpt,ndimpt=',nimpt,ndimpt
843: C      write(6,*) ' imded=',(imded(j),j=1,ndimpt)
844: C      if(JSTG.eq.0) then
845: C          write(6,*) ' imsfl=',(imsfl(n+j),j=1,nn2)
846: C          write(6,*) ' imzfl=',(imzfl(n+j),j=1,nn2)
847: C      else
848: C          write(6,*) ' imslt=',(imslt(nl+j),j=1,nn2)
849: C          write(6,*) ' imzlt=',(imzlt(nl+j),j=1,nn2)
850: C      endif
851: C          write(6,*) ' imnfl=',(imnfl(j),j=1,nzone+1)
852: C          write(6,*) ' imnlt=',(imnlt(j),j=1,nlbz+1)
853: C          write(6,*) ' xxx =',(xxx(imsfl(n+j),2),j=1,nn2)
854: C          write(6,*) ' yyy =',(yyy(imsfl(n+j),2),j=1,nn2)
855: C          write(6,*) ' zzz =',(zzz(imsfl(n+j),2),j=1,nn2)
856: C          write(6,*) ' aaa =',(aaa(imsfl(n+j),2),j=1,nn2)
857: C          write(6,*) ' bbb =',(bbb(imsfl(n+j),2),j=1,nn2)
858: C          write(6,*) ' ccc =',(ccc(imsfl(n+j),2),j=1,nn2)
859: C          write(6,*) ' levl =',(levl(imsfl(n+j),2),j=1,nn2)
860: C          write(6,*) ' iggi =',(igg(imsfl(n+j),2),j=1,nn2)
861: C          do 1000 l=1,levl(imsfl(n+1),2)
862: C              write(6,*) ' level =',l
863: C              write(6,*) ' lzz =',(lzzi(imsfl(n+j),1),j=1,nn2)
864: C              write(6,*) ' lpos =',(lposi(imsfl(n+j),1),j=1,nn2)
865: C              write(6,*) ' lcrs =',(lcrsi(imsfl(n+j),1),j=1,nn2)
866: C          c1000 continue
867: C
868: C *** CALL TRACKING CONTROL ROUTINE FOR IMAGINARY PARTICLES
869: C
870:         if ( NDIMPT.eq.0 ) then
871:             call NXTEE( 2, A, H, NDIMPT, NIMPT )
872:         end if
873: C
874:         go to 320
875: C
876:         end if
877: C
878: C
879: 360 continue
880: C
881: C **** END OF DETECTOR LOOP *****
882: C
883:         return
884:     end

```

src/gmvp/nxtee.f

```

1: *VOCL TOTAL,SCALAR
2:   SUBROUTINE NXTEE( MODE, A, H, NDIMP2, NIMPT2 )
3: C=====
4: C PURPOSE: TRACKING OF IMAGINARY PARTICLES FOR NEXT EVENT ESTIMATOR.
5: C CALLED IN: SOURCE,COLLISION,SELECT,SELONE
6: C CALLS:
7: C
8: C=====
9: C OPERATION MODE :
10: C MODE = 1 : PROCESS ALL IMAGINARY PARTICLES
11: C           UNTIL FINISHED.
12: C   = 2 : PROCESS IMAGINARY PARTICLES UNTIL
13: C     ENTRY OF BANK IS LESS THAN 'NTHIM' PARTICLES.
14: C   = N : PROCESS N EVENTS.
15: C=====
16: C
17:   real A(*)
18:   real H(*)
19:   include 'INC/_FLAGS'
20:   include 'INC/_SIZES'
21:   include 'INC/_SIZES2'
22:   include 'INC/_XBANK'
23:   include 'INC/_STACK'
24:   include 'INC/_XWORK'
25:   include 'INC/_PGEOM'
26:   include 'INC/_PXSEC'
27:   include 'INC/_PVRED'
28:   include 'INC/_PTALY0'
29:   include 'INC/_PTALY'
30:   include 'INC/_IOUNIT'
31:   include 'INC/_IOUNIT2'
32:   include 'INC/_WKNXT'
33:   include 'INC/_STALY'
34:   include 'INC/_PTLSP'
35: C
36:   NIMPT = NIMPT2
37:   NDIMP2 = NDIMP2
38: C
39: C .... CLEAR EVENT COUNTER ....
40: C
41:   IEVENT = 0
42: C
43: C .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
44: C
45: C 2222 CALL NEESEL(MODE, IEVENT, MACT, MZONE, NIMPT,NDIMP2,
46:   & h(LIMNFL), h(LIMNNX), IMDEFR, h(LIMNLT) )
47: C
48: C .....
49: C
50: C MACT
51: C ** = 0 : GENERATE NEXT BATCH OF PARTICLES.
52: C   = 1 : FREE FLIGHT
53: C   = 2 : NEXT ZONE SEARCH
54: C ** = 3 : COLLISION
55: C ** = 5 : REFLECTION
56: C   = 6 : LATTICE-SEARCH
57: C   = 99 : END OF RUN
58: C
59: C MZONE : SELECTED ZONE NUMBER (IF ZONEWISE LOGIC)
60: C
61: C .....
62: C
63: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT
64: C   & TAKE TALLY IF NECESSARY .....
65: C

```

```

66:   IF(MACT.EQ.1) THEN
67:     call NXTFLI(IOW,A,H, MZONE, NIMPT, NDIMP2,h(LNLOST),DEPS,DINF,
68:   B   h(LXXXI), h(LYYYI), h(LZZZI), h(LAAAI), h(LBBBI), h(LCCCI),
69:   B   h(LWWWI), h(LIGGI), h(LTTTI), h(LITTI),
70:   B   h(LLEVI),h(LLZZI), h(LLPOSI),h(LLCRSI),h(LKLSFI) ,
71:   B   h(LPATH),h(LOPTI),h(LKDETP),
72:   B   h(LDBNKI),KDBNK,MDBNK,h(LIBNKI),KIBNK,MIBNK,
73:   S   h(LIMSFL),h(LIMNFL),h(LIMZFL),
74:   S   h(LIMSNX),h(LIMNNX),h(LIMZNX),h(LIMDED),
75:   G   A(LSDA),A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),A(LDALT),
76:   G   A(LKDALT),A(LNVLAT),A(LSZLAT),A(LIPLAT),A(LIPCEL),A(LKSLAT),
77:   G   A(LLTYP),A(LMLBZZ),A(LKZLBZ),A(LKCELL),A(LICTYP),A(LKLATT),
78:   G   A(LCELSZ),A(LPNND),A(LKNND),A(LPPPF),A(LKPPF),
79:   T   NTGX,A(LSTOTX),h(LFLPD),A(LIDTAL),A(LJTEVE),A(LLTEVE),
80:   T   A(LKTEVE),NTEVE,h(LDTALY),A(LRESP),
81:   W   h(PF(1)),h(PF(2)),h(PF(3)),h(PF(4)),h(PF(5)),h(PF(6)),
82:   W   h(PF(7)),h(PF(8)),h(PF(9)),h(PF(10)),h(PF(11)),h(PF(12)),
83:   W   h(PF(13)),h(PF(14)),h(PF(15)),h(PF(16)),h(PF(17)),h(PF(18)),
84:   W   h(PF(19)),h(PF(20)),h(PF(21)),h(PF(22)),h(PF(23)),
85:   W   h(PF(24)),h(PF(25)),h(PF(26)) )
86: C* W X Y Z AI BI CI
87: C* W W DI IGI IBP IFC R
88: C* W DLOC1,DLOC2,IKL,IKL2,XUP,YUP,
89: C* W ZUP,AUP,BUP,CUP
90: C* W ISTG,ILUP,ILST )
91: C
92: C .... FIND NEXT ZONE TO ENTER .....
93: C
94:   ELSE IF(MACT.EQ.2) THEN
95:     CALL NEESEA(IOW,JVMNT,JLATT,JSIMP,MZONE,NIMPT,NDIMP2,
96:   E   h(LNLOST),DEPS,JMEM,h(LKMEMO),h(LMEMC),h(LMEMZ),
97:   B   h(LXXXI),h(LYYYI),h(LZZZI),h(LAAAI),h(LBBBI),h(LCCCI),
98:   B   h(LWWWI),h(LIZZI),h(LIGGI),h(LTTTI),h(LLEVI),h(LLZZI),
99:   B   h(LLPOSI),h(LLCRSI),h(LKLSFI),
100:  S   h(LIMSFL),h(LIMNFL),h(LIMZFL),h(LIMSNX),h(LIMNNX),h(LIMZNX),
101:  S   h(LIMDED),IMDEFR,
102:  *   h(LIMSLT),h(LIMZLT),h(LIMNLT),
103:  G   A(LSDA),A(LKZMAT),A(LKZDA),A(LKZAA),
104:  *   A(LKCELL),A(LIPCEL),A(LMLBZZ),
105:  T   h(LNCNTR),h(LWCNTR),
106:  W   h(PS(1)),h(PS(2)),h(PS(3)),h(PS(4)),h(PS(5)),h(PS(6)),
107:  W   h(PS(7)),h(PS(8)),h(PS(9)),h(PS(10)) )
108: C** W X Y Z W IBP IFI
109: C** W FXYZ,ISRF,IZI,IFL )
110: C
111: C .... LATTICE SEARCH .....
112: C
113:   ELSE IF(MACT.EQ.6) THEN
114:     JTLT1 = 0
115:     call LATTICE(JDEBG,JVMNT,JTLT1,JHLAT,JFISX,h(LNLOST),
116:   N   NBANK,NZONE,NREG,NLATT,NCELL,NLBZ,NEST,NSUZON,NSPACE,NUNV,
117:   N   DINF,DEPS,IRAND,NMKREG,
118:   B   h(LXXXI),h(LYYYI),h(LZZZI),h(LAAAI),h(LBBBI),h(LCCCI),
119:   B   h(LIZZI),h(LLEVI),h(LLZZI),h(LLPOSI),h(LIMDED),NDIMP2,
120:   B   h(LLCRSI),h(LKLSFI),h(LIBREG),h(LIBSPC),
121:   B   h(LDBNKI),KDBNK,MDBNK,h(LIBNKI),KIBNK,MIBNK,
122:   S   h(LIMSNX),h(LIMZNX),h(LIMNNX),h(LIMSLT),h(LIMZLT),h(LIMNLT),
123:   S   h(LIMSFL),h(LIMNFL),h(LIMZFL),
124:   G   A(LSDA),A(LKZDA),A(LKZAA),
125:   G   A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LMLBZZ),A(LCELSZ),A(LSZLAT),
126:   G   A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),A(LNVLAT),A(LIPCEL),
127:   G   A(LLTYP),A(LICTYP),A(LKDALT),A(LDALT),A(LKZREG),
128:   G   A(LISUSP),A(LKSPSU),A(LKTCSP),NKTCSP,A(LKHLAT),A(LMKREG),
129:   G   A(LPPPF),A(LKPPF),NPPPF,
130:   W   h(PL(1)),h(PL(2)),h(PL(3)),h(PL(4)),h(PL(5)),h(PL(6)),

```

src/gmvp/nxtee.f

```
131:      W H(PL( 7)), H(PL( 8)), H(PL( 9)), H(PL(10)),
132:      W H(PL(11)), H(PL(12)), H(PL(13)), H(PL(14)),H(PL(15)), H(PL(16)))
133:      NIMPT = IMPMAX - NDIMPT
134:
135: C
136:      ELSE IF( MACT .EQ. 99 ) THEN
137:          GOTO 9999
138:      ENDIF
139: C
140:      GOTO 2222
141: C
142: C ..... END OF TASK .....
143: C
144: 9999 NDIMP2 = NDIMPT
145:      NIMPT2 = NIMPT
146:      RETURN
147:      END
148: check
149: c      subroutine chkwrt(iggi,imsfl,imsnx,imslt,
150: c      &      imnfl,imnnx,imnlt,impmax,nzone,nlbz)
151: c      integer imsfl(impmax),imsnx(impmax),imslt(impmax)
152: c      integer imnfl(nzone+1),imnnx(nzone+1),imnlt(nlbz+1)
153: c      integer iggi(impmax)
154: c      write(6,*) ' iggi:',(iggi(i),i=1,impmax)
155: c      write(6,*) ' imsfl:',(imsfl(i),i=1,imnfl(nzone+1))
156: c      write(6,*) ' imsnx:',(imsnx(i),i=1,imnnx(nzone+1))
157: c      write(6,*) ' imslt:',(imslt(i),i=1,imnlt(nlbz+1))
158: c      return
159: c      end
```

src/gmvp/nxtfli.f

```

1:      subroutine NXTFLI( IOW, A,      H,
2:      &                  MZONE, NIMPT, NDIMPT, NLOST, DEPS, DINF,
3:      &                  XXXI, YYI, ZZI, AA, BB, CCI, WWI,
4:      &                  IGI, TTI, ITTI, LEVLI, LZ, LPOSI, LCRSI,
5:      &                  KLSFI, PATH, OPTI, KETP,
6:      &                  DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
7:      &                  IMSFL, IMNFL, IMZFL, IMSNX, IMNNX, IMZNX, IMDED,
8:      &                  SDA, KZMAT, KZREG, KZDA, KZAA, DALT, KDALT,
9:      &                  NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP, MLBZZ,
10:     &                  KZLBZ, KCELL, ICTYP, KLATT, CELSZ,
11:     &                  PNND, KNND, PPPF, KPPF,
12:     &                  NTGX, STOTX, FLPD,
13:     &                  IDTAL, JTEVE, LTEVE, KTEVE, NTEVE,
14:     &                  DTALY, RESP,
15:     &                  X, Y, Z, AI, BI, CI, OPT,
16:     &                  DI, IGI, IBP, IFC, R, DLOC1,
17:     &                  DLOC2, IKL, IKL2, XUP, YUP, ZUP, AUP,
18:     &                  BUP, CUP, SIGT, ISTG, ILUP, ILST )
19: C      W      XYZ1, ABC1, XYZ2, ABC2,
20: C      W      DB1, IBP0, IBP1, IBP2, KPLT, JKSF )
21: C===== FLIGHT =====
22: C PURPOSE:      FREE-FLIGHT OF ONE-ZONE LOGIC FOR NEXT EVENT ESTIMATOR
23: C              TAKE TALLY OF POINT DETECTOR.
24: C
25: C CALLED IN:  NXTEE
26: C CALLS:      NEEFLI, STALNS
27: C
28: C=====
29: C
30: C      === INTER-STACK DATA FLOW ===
31: C
32: C      FREE-FLIGHT STACK -----> NEXT-ZONE-SEARCH STACK
33: C      (IMSFL,IMZFL,IMNFL) | (IMSNX,IMZNX,IMNNX)
34: C      (AFTER TALLY) +-----> DEAD PARTICLE STACK
35: C                          (IMDED,NDIMPT)
36: C
37: C      === BANK DATA TO BE UPDATED ===
38: C
39: C      (XXXI,YYI,ZZI) : POSITION
40: C      (AAAI,BBBI,CCCI) : DIRECTION (IF HEXAGONAL-LATTICE EXISTS)
41: C      LEVLI : LATTICE LEVEL (IF HEXAGONAL-LATTICE EXISTS)
42: C      KLSFI : SURFACE DATA POINTER (IF ON-SURFACE)
43: C=====
44:      implicit real*8(D)
45:      include 'INC/_FLAGS'
46:      include '.../shared/INC/_SIZES'
47:      include 'INC/_SIZES2'
48:      include '.../shared/INC/_TASKDT'
49:      include '.../shared/INC/_PTALY0'
50: C
51:      real A(*)
52:      real H(*)
53: C
54: C .... PARTICLE BANK .....
55: C
56:      real*8 XXXI(IMPMAX), YYI(IMPMAX), ZZI(IMPMAX), AA(IMPMAX),
57:      &      BB(IMPMAX), CCI(IMPMAX)
58:      real WWI(IMPMAX)
59:      real*8 TTI(IMPMAX)
60:      integer ITTI(IMPMAX)
61:      integer IGI(IMPMAX), LEVLI(IMPMAX), LZ(IMPMAX,NEST),
62:      &      LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST), KLSFI(IMPMAX)
63: C
64: C      ===== IMAGINARY PARTICLES =====
65: C

```

```

66:      integer KETP(IMPMAX)
67:      real*8 OPTI(IMPMAX), PATH(IMPMAX)
68: C
69: C      ===== optional bank parameters =====
70: C
71:      real*8 DBNK(IMPMAX,*)
72:      integer KDBNK(0:MDBNK)
73:      integer IBNK(IMPMAX,*)
74:      integer KIBNK(0:MIBNK)
75: C
76: C
77: C .... STACKS .....
78: C
79:      integer IMSFL(IMPMAX), IMNFL(NZONE+1), IMZFL(IMPMAX),
80:      &      IMSNX(IMPMAX), IMNNX(NZONE+1), IMZNX(IMPMAX),
81:      &      IMDED(IMPMAX)
82: C
83: C .... GEOMETRY .....
84: C
85:      real*8 SDA(NSDA), DALT(*), SZLAT(8,NLATT), CELSZ(6,*)
86:      integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
87:      &      KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT), IPLAT(*), KSLAT(*),
88:      &      KCELL(NZONE), ICTYP(NCELL), KLATT(*), NVLAT(4,*)
89:      integer IPCEL(NCELL)
90:      integer KZLBZ(NLBZ)
91: C
92: C      ... Nearest Neighbour Distribution for STGM region.
93: C
94:      real PNND(NPNND)
95:      integer KNND(NPNND)
96: C
97: C      ... Partial Packing Fraction for STGM-region
98: C
99:      real PPPF(NPPPF)
100:     integer KPPF(NPPPF)
101: C
102: C .... CROSS SECTION .....
103: C
104:      real STOTX(NTGX,*)
105: C
106: C .... TALLY .....
107: C
108:      real*8 FLPD(NGROUP,*)
109:      integer JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
110:      integer IDTAL(*)
111:      real*8 DTALY(*)
112:      real RESP(NGROUP,NRESP)
113: C
114: C .... WORKING AREA (LENGTH = IMPMAX) .....
115: C
116:      real*8 X(IMPMAX), Y(IMPMAX), Z(IMPMAX), AI(IMPMAX), BI(IMPMAX),
117:      &      CI(IMPMAX), OPT(IMPMAX), DI(IMPMAX), DLOC1(IMPMAX),
118:      &      DLOC2(IMPMAX), XUP(IMPMAX), YUP(IMPMAX), ZUP(IMPMAX),
119:      &      AUP(IMPMAX), BUP(IMPMAX), CUP(IMPMAX)
120:      real R(3*IMPMAX)
121:      integer IGI(IMPMAX), IBP(IMPMAX), IFC(IMPMAX)
122:      integer ILST(IMPMAX), ILUP(IMPMAX), ISTG(IMPMAX)
123:      real SIGT(IMPMAX)
124: C
125: C      .... FOR SURFACE POINTER ...
126: C
127:      integer IKL(IMPMAX), IKL2(IMPMAX)
128: C
129:      include '.../shared/INC/_PMLATT'
130:      include '.../shared/INC/_SFLATT'

```

src/gmvp/nxtfli.f

```

131: C
132: C
133: C -----
134: C .... set of total cross sections ...
135: C -----
136: C
137:       MAT      = KZMAT(MZONE,1)
138: C
139: C ... this is a base material (matrix) region of STG region.
140: C
141:       JJJNND = 0
142:       if ( MAT.lt.0 ) then
143:       if ( ISLATT(MAT).and.LTYP(LATTNM(MAT)).eq.10 ) then
144:         MAT      = KZMAT(MZONE,2)
145:         JJJNND = 1
146:       else
147:         write(IOW,*) 'XXX(NEEFLI) Program error ? Invalid zone :',
148:           & ' MZONE=', MZONE, ' KZMAT(MZONE,*) ',
149:           & KZMAT(MZONE,1), KZMAT(MZONE,2), ' LTYP ',
150:           & LTYP(LATTNM(MAT))
151:         stop 666
152:       end if
153:     end if
154: C
155:       II      = 0
156:       do 100 I = 1, IMNFL(MZONE+1)
157:         if ( IMZFL(I).eq.MZONE ) then
158:           II    = II + 1
159:           IBP(II) = IMSFL(I)
160:         end if
161:       100 continue
162:       if ( MAT.gt.0 ) then
163:       *VOCL LOOP,NOVREC
164:         do 110 I = 1, II
165:           SIGT(IBP(I)) = STOTX(IGGI(IBP(I)),MAT)
166:         110 continue
167:       else
168:       *VOCL LOOP,NOVREC
169:         do 120 I = 1, II
170:           SIGT(IBP(I)) = 0.0
171:         120 continue
172:       end if
173: C
174: C
175:       call NEEFLI( IOW, JVMNT, JFISX, JHLAT, JDEBG, JJJNND,
176:         & MZONE, NIMPT, NDIMPT, NLOST, DEPS, DINF,
177:         & XXXI, YYI, ZZI, AA, BB, CCI, WNI,
178:         & IGGI, TT, ITTI, LEVLI, LZ, LPOSI, LCR,
179:         & KLSFI, PATH, OPTI, KDETP, SIGT,
180:         & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
181:         & IMSFL, IMNFL, IMZFL, IMSNX, IMNX, IMZNX, IMDED,
182:         & NDIL, NTRM,
183:         & SDA, KZMAT, KZREG, KZDA, KZAA, DAL, KDALT,
184:         & NVLAT, SZLAT, IPLAT, IPCEL, KSLAT, LTYP, MLBZZ,
185:         & KZLBZ, KCELL, ICTYP, KLATT, CELSZ,
186:         & PNND, KNND, PPPF, KPPF,
187:         & X, Y, Z, AI, BI, CI, OPT,
188:         & DI, IGI, IBP, IFC, R, DLOC1,
189:         & DLOC2, IKL, IKL2, XUP, YUP, ZUP, AUP,
190:         & BUP, CUP, ISTG, ILUP, ILST )
191: C       W       XYZ1, ABC1, XYZ2, ABC2,
192: C       W       DDI, IBP0, IBP1, IBP2, KPLT, JKSF )
193: C
194: C === data for tallies =====
195: C       bank pointer : IMDED(NDIL+1) ---> IMDED(NDIL+NTRM)

```

```

196: C       NDIL : starting bank pointer of particles to be tallied
197: C       by next event estimator.
198: C       NTRM : number of particles to be tallied by next event estimator.
199: C       OPT(1:NTRM) : contribution of i'th particle
200: C =====
201: C
202: C === TAKE TALLY (FLUX) =====
203: C
204:       *VOCL LOOP,SCALAR
205:         do 200 I = 1, NTRM
206:           IBPI = IMDED(NDIL+I)
207:           FLPD(IGGI(IBPI),KDETP(IBPI)) = FLPD(IGGI(IBPI),KDETP(IBPI))
208:           & + OPT(I)
209:         200 continue
210: C
211: C -----
212: C .... TAKE SPECIAL TALLYS .....
213: C -----
214: C
215:       if ( NTEVE.gt.0.and.JTEVE(1).gt.0 ) then
216: C
217:         NNEUT = 0
218:         NPHOT = 0
219:         if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
220:           do 210 I = 1, NTRM
221:             if ( IGGI(IMDED(NDIL+I)).le.NGP1 ) then
222:               NNEUT = NNEUT + 1
223:               IBP(NNEUT) = IMDED(NDIL+I)
224:             else
225:               NPHOT = NPHOT + 1
226:               IFC(NPHOT) = IMDED(NDIL+I)
227:             end if
228:           210 continue
229:         else if ( JNEUT.ne.0 ) then
230:           NNEUT = NTRM
231:         else if ( JPHOT.ne.0 ) then
232:           NPHOT = NTRM
233:         end if
234: C
235:         do 220 J = 0, JTEVE(1) - 1
236: C
237: C ... pointer to direct tally (idt) & d-tally # (it) ...
238:         IDT = KTEVE(LTEVE(1)+J) - 1
239:         IT = IDTAL(IDT+9)
240: C
241: C .. neutron or photon ?
242:         if ( IDTAL(IDT+5).eq.1.and.NNEUT.eq.0 ) go to 220
243:         if ( IDTAL(IDT+5).eq.2.and.NPHOT.eq.0 ) go to 220
244: C
245:         I1 = 1
246:         I2 = NTRM
247: C
248:         if ( I2.gt.0 ) then
249:           call STALN9(IOW,IDTAL(IDT+1),OPT(I1),IMDED(NDIL+1), I2,
250:             & DTALY,
251:             & IMPMAX, NGROUP, NGP1, NREG, NRESP,
252:             & NZONE, NTIME, NMKREG, A(LMKREG),
253:             & RESP,
254:             B, KDETP, IGGI, ITTI,
255:             B, DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
256:             W, AI, BI, R )
257: C
258:         end if
259:       220 continue
260: C

```

src/gmvp/nxtfli.f

```
261:      end if
262: C
263:      return
264:      end
```

SAFE

src/gmvp/option.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine OPTION( MLIMIT )
3: C=====
4: C PURPOSE: READ OPTION PARAMETERS
5: C CALLED IN:  INTRO
6: C CALLS      :
7: C
8: C=====
9:   include '../shared/INC/_ARRAY'
10:  include '../shared/INC/_TASKDT'
11: C
12:  include 'INC/_KPIDS'
13: C
14:  include 'INC/_FLAGS'
15:  include '../shared/INC/_IOUNIT'
16: C
17:  character LINE*72, CH*1, PFLAG*72
18:  parameter( MAXPRT = 10 )
19:  integer IT(MAXPRT)
20:  integer*8 ITS(MAXPRT)
21: C
22: C/#IF DYNAMIC
23: C
24: C ... reset limit size of array A in common /ARRAY/ here,
25: C   dynamic allocation of memory is available.
26: C
27:   LIMIT = MLIMIT
28: C/# IF PARA( CRAY SX* )
29: *   MLMTL = LIMITL
30: C/# ENDDIF
31: C/#ENDIF
32: C
33:   NTASK0 = NTASK
34: C
35: C ..... READ-IN NEW RECORD .....
36: C
37:   100 call FREADB( LINE, NLEN, IEND )
38:   if ( IEND.ne.0 ) go to 150
39:   JJ = 1
40: C-----
41: C ..... A 'NO-' ON HEAD MEANS NOT TO USE THIS OPTION .....
42: C-----
43:   IS = 1
44:   IE = NLEN
45:   if ( NLEN.gt.3.and.LINE(IS:IS+2).eq.'NO-' ) then
46:     JJ = 0
47:     IS = IS + 3
48:   end if
49: C   IEI = IE + 1
50: C
51:   NHYP = 0
52:   do 110 I = IS, IE
53:     if ( LINE(I:I).eq.'-' ) NHYP = NHYP + 1
54:   110 continue
55: C
56: C-----
57: C ..... CHECK OPTIONS WITH NUMERICAL DATA ...   XXXX-YYY(N)
58: C-----
59: C*****IBR = INDEX(LINE(IS:IE),'(')
60: C ... GET NEXT NON-BLANK CHARACTER IN THE INPUT FILE ....
61: C   .... ( DETECT CHARACTER OTHER THAN ' ' )
62: C
63:   IBR = 0
64:   call FPROBE( IBR, ' ', CH )
65:   if ( IBR.ne.0.and.CH.ne.'(' ) IBR = 0

```

```

66: C
67: C   .... IBRU IS SET TO 0 IF NUMERICAL DATA ARE USED ....
68: C
69:   IBRU = 1
70: C
71: C ..... CHANGE THE FLAGS .....
72: C
73: C ===== RESTART
74: C
75:   if ( NHYP.eq.0.and.IMATCH('REST*',LINE(IS:IE)).eq.1 ) then
76:     JREST = JJ
77: C
78: C ===== AUTO-RESTART
79: C
80:   else if ( NHYP.eq.1.and.IMATCH('AUTO-REST*',LINE(IS:IE)).eq.1 )
81: & then
82:     JARST = JJ
83: C
84: C ===== NEUTRON
85: C
86:   else if ( NHYP.eq.0.and.IMATCH('NEUT*',LINE(IS:IE)).eq.1 ) then
87:     JNEUT = JJ
88:     JKPAB(KPNEUT) = JNEUT
89: C
90: C ===== PHOTON
91: C
92:   else if ( NHYP.eq.0.and.IMATCH('PHOT*',LINE(IS:IE)).eq.1 ) then
93:     JPHOT = JJ
94:     JKPAB(KPPHOT) = JPHOT
95: C
96: C ===== PARTICLE
97: C
98:   else if ( NHYP.eq.0.and.IMATCH('PART*',LINE(IS:IE)).eq.1 ) then
99: C
100:   if ( IBR.ne.0 ) then
101:     PFLAG = ' '
102: 122   call CHREAD( ' ', PFLAG, ' ', IJ, ITERM, IEND )
103:   if ( IEND.ne.0 ) then
104:     write(IMG,*)'XXX Unexpected end of data in option block'
105:     call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
106:     stop 888
107:   end if
108: C
109:   if ( IJ.ne.0 ) then
110:     IBRU = 0
111: C
112: C ..... check supported particle or not .....
113: C
114:   call KPSYMB(PFLAG(:IJ), '>', IKPID, 0 )
115: C
116:   if ( IKPID.eq.0 ) then
117:     write(IMG,*)'XXX unsupported particle in PARTICLE()',
118: & ' <', PFLAG(:IJ), '>'
119:     call CNTERR('FATAL')
120:   else
121:     JKPAB(IKPID) = JJ
122:     if ( IKPID.eq.KPNEUT ) then
123:       JNEUT = JJ
124:     end if
125:     if ( IKPID.eq.KPPHOT ) then
126:       JPHOT = JJ
127:     end if
128:   end if
129:   if ( ITERM.eq.0 ) go to 122
130: C

```


src/gmvp/option.f

```

131:          IBRU      = 0
132:          end if
133: C
134: C ===== ADJOINT
135: C
136:          else if ( NHYP.eq.0.and.IMATCH('ADJO*',LINE(IS:IE)).eq.1 ) then
137:              JADJM    = JJ
138: C
139: C ===== IMAGINARY-PARTICLE
140: C
141:          else if ( NHYP.eq.1.and.IMATCH('IMAG*-PART*',LINE(IS:IE)).eq.1 )
142:              & then
143:              JIMAG    = JJ
144: C
145: C ===== TIME-DEPENDENT
146: C
147:          else if ( NHYP.eq.1.and.IMATCH('TIME*-DEPE*',LINE(IS:IE)).eq.1 )
148:              & then
149:              JTIME    = JJ
150: C
151: C ===== PERIODIC-TIME
152: C
153:          else if ( NHYP.eq.1.and.IMATCH('PERI*-TIME*',LINE(IS:IE)).eq.1 )
154:              & then
155:              JPTIM    = JJ
156: C
157: C ===== DELTA-TRACKING
158: C
159:          else if ( NHYP.eq.1.and.IMATCH('DELT*-TRAC*',LINE(IS:IE)).eq.1 )
160:              & then
161:              JDELT    = JJ
162: C
163: C ===== FISSION
164: C
165:          else if ( NHYP.eq.0.and.IMATCH('FISS*',LINE(IS:IE)).eq.1 ) then
166:              JFISS    = JJ
167: C
168: C ===== EIGEN-VALUE
169: C
170:          else if ( NHYP.eq.1.and.IMATCH('EIGE*-VALU*',LINE(IS:IE)).eq.1 )
171:              & then
172:              JEIGN    = JJ
173:              JFIXD    = 1 - JJ
174: C
175: C ===== FIXED-SOURCE
176: C
177:          else if ( NHYP.eq.1.and.IMATCH('FIXE*-SOUR*',LINE(IS:IE)).eq.1 )
178:              & then
179:              JFIXD    = JJ
180:              JEIGN    = 1 - JJ
181: C
182: C ===== RUSSIAN-ROULETTE (1989 8/29)
183: C
184:          else if ( NHYP.eq.1.and.IMATCH('RUSS*-ROUL*',LINE(IS:IE)).eq.1 )
185:              & then
186:              JRRLT    = JJ
187:              if ( JRRLT.eq.1 ) JWWND = 0
188: C
189: C ===== IMPORTANCE
190: C
191:          else if ( NHYP.eq.0.and.IMATCH('IMPO*',LINE(IS:IE)).eq.1 ) then
192:              JIMPT    = JJ
193:              if ( JIMPT.eq.1 ) JWWND = 0
194: C
195: C ===== WEIGHT-WINDOW

```

```

196: C
197:          else if ( NHYP.eq.1.and.IMATCH('WEIG*-WIND*',LINE(IS:IE)).eq.1 )
198:              & then
199:              JWWND    = JJ
200:              if ( JWWND.eq.1 ) JIMPT = 0
201:              if ( JWWND.eq.1 ) JRRLT = 0
202: C
203: C ===== PATH-STRETCHING
204: C
205:          else if ( NHYP.eq.1.and.IMATCH('PATH*-STRE*',LINE(IS:IE)).eq.1 )
206:              & then
207:              JPSTR    = JJ
208: C
209: C ===== FORCED-COLLISION
210: C
211:          else if ( NHYP.eq.1.and.IMATCH('FORC*-COLL*',LINE(IS:IE)).eq.1 )
212:              & then
213:              JFCOL    = JJ
214: C
215: C ===== COLLISION-LESS
216: C
217:          else if ( NHYP.eq.1.and.IMATCH('COLL*-LESS',LINE(IS:IE)).eq.1 )
218:              & then
219:              JNCOL    = JJ
220: C
221: C ===== SOURCE-BIASING
222: C
223:          else if ( NHYP.eq.1.and.IMATCH('SOUR*-BIAS*',LINE(IS:IE)).eq.1 )
224:              & then
225:              JBIAS    = JJ
226: C
227: C ===== RELATIVE-WEIGHT
228: C
229:          else if ( NHYP.eq.1.and.IMATCH('RELA*-WEIG*',LINE(IS:IE)).eq.1 )
230:              & then
231:              JRWVR    = JJ
232: C
233: C ===== RESPONSE
234: C
235:          else if ( NHYP.eq.0.and.IMATCH('RESP*',LINE(IS:IE)).eq.1 ) then
236:              JRESP    = JJ
237: C
238: C ===== MONITOR
239: C
240:          else if ( NHYP.eq.0.and.IMATCH('MONI*',LINE(IS:IE)).eq.1 ) then
241:              JMNTR    = JJ
242: C
243: C ===== VP-MONITOR
244: C
245:          else if ( NHYP.eq.1.and.IMATCH('VP-MONI*',LINE(IS:IE)).eq.1 ) then
246:              JVMNT    = JJ
247: C
248: C ===== DEBUG-PRINT
249: C
250:          else if ( NHYP.eq.1.and.IMATCH('DEBU*-PRIN*',LINE(IS:IE)).eq.1 )
251:              & then
252:              if ( JJ.eq.0 ) then
253:                  do 120 I = 1, MAXJDB
254:                      JDEBG(I) = 0
255:                  120 continue
256:              if ( IBR.ne.0 ) then
257:                  call DMREAD( ' ', NA, NB, IRET )
258:                  IBRU    = 0
259:              end if
260:          else if ( IBR.ne.0 ) then

```

src/gmvp/option.f

```

261:      call I4READ( 'DEBUG-PRINT', JDEBG, NA, -MAXJDB, IERR )
262:      if ( IERR.ne.0 ) then
263:        write(IMG,*)
264:        &      'XXX Invalid parameter for option "DEBUG-PRINT".'
265:        call CNTERR( 'FATAL' )
266:      end if
267:      IBRU = 0
268:    else
269:      JDEBG(1) = JJ
270:    end if
271: C
272: C ===== VPP-SPECIFIC
273: C
274:   else if ( NHYP.eq.1.and.IMATCH('VPP-SPEC*',LINE(IS:IE)).eq.1 )
275:     &      then
276:       if ( JJ.eq.0 ) then
277:         do 130 I = 1, MXJVP
278:           JVPSP(I) = 0
279: 130      continue
280:         if ( IBRU.ne.0 ) then
281:           call DMREAD( ' ', NA, NB, IRET )
282:           IBRU = 0
283:         end if
284:       else if ( IBRU.ne.0 ) then
285:         call I4READ( 'VPP-SPECIFIC', JVPSP, NA, -MAXJVP, IERR )
286:         if ( IERR.ne.0 ) then
287:           write(IMG,*)
288:           &      'XXX Invalid parameter for option "VPP-SPECIFIC".'
289:           call CNTERR( 'FATAL' )
290:         end if
291:         IBRU = 0
292:       else
293:         JVPSP(1) = JJ
294:       end if
295: C
296: C ===== PICTURE
297: C
298:   else if ( NHYP.eq.0.and.IMATCH('PICT*',LINE(IS:IE)).eq.1 ) then
299:     JPICT = JJ
300: C
301: C ===== ALL-ZONE (1988/2/25)
302: C OR EVENT-SELECTION (1989/2/14)
303: C
304:   else if ( NHYP.eq.1.and.IMATCH('ALL-ZONE',LINE(IS:IE)).eq.1
305:     &      .or. IMATCH('EVEN*-SELE*',LINE(IS:IE)).eq.1 ) then
306:     JALLZ = JJ
307:     JONEZ = 1 - JJ
308: C
309: C ===== ONE-ZONE (1988/2/25)
310: C OR ZONE-SELECTION (1989/2/14)
311: C
312:   else if ( NHYP.eq.1.and.IMATCH('ONE-ZONE',LINE(IS:IE)).eq.1
313:     &      .or. IMATCH('ZONE-SELE*',LINE(IS:IE)).eq.1 ) then
314:     JONEZ = JJ
315:     JALLZ = 1 - JJ
316: C
317: C ===== LATTICE (1988/10/19)
318: C
319:   else if ( NHYP.eq.0.and.IMATCH('LATT*',LINE(IS:IE)).eq.1 ) then
320:     JLATT = JJ
321: C
322: C ===== FREE-LATTICE-FRAME (1988/12/28)
323: C
324:   else if ( NHYP.eq.2
325:     &      .and. IMATCH('FREE*-LATT*-FRAM*',LINE(IS:IE)).eq.1 ) then

```

```

326:      JFISX = JJ
327: C
328: C ===== FLUX-PRINT (1989/08/22)
329: C
330:   else if ( NHYP.eq.1.and.IMATCH('FLUX-PRIN*',LINE(IS:IE)).eq.1 )
331:     &      then
332:       JFPRT = JJ
333: C
334: C ===== PRINT-SUPPRESS (1990/07/11)
335: C
336:   else if ( NHYP.eq.1
337:     &      .and. IMATCH('PRIN*-SUPP*',LINE(IS:IE)).eq.1 ) then
338:     if ( JJ.eq.0 ) then
339:       do I = 1, MAXPRT
340:         JPRTS(I) = 0
341:       end do
342:       if ( IBRU.ne.0 ) then
343:         call DMREAD( ' ', NA, NB, IRET )
344:         IBRU = 0
345:       end if
346:     else if ( IBRU.ne.0 ) then
347: C
348:       call I4READ( 'PRINT-SUPPRESS', IT, NA, -MAXPRT, IERR )
349:       if ( IERR.ne.0 ) then
350:         write(IMG,('/lx,a/'))
351:         &      'XXX Invalid parameter for option "PRINT-SUPPRESS".'
352:         call CNTERR( 'FATAL' )
353:       end if
354:       IBRU = 0
355: C
356:       if ( NA.gt.0 ) then
357:         do 140 I = 1, NA
358:           if ( IT(I).ge.1.and.IT(I).le.MAXPRT ) JPRTS(IT(I)) = 1
359: 140      continue
360:       else
361:         write(IMG,*)
362:         &      '!!! no data for PRINT-SUPPRESS'
363:         call CNTERR( 'WARNING' )
364:       end if
365:     else
366:       write(IMG,*)
367:       &      '!!! no data for PRINT-SUPPRESS'
368:       call CNTERR( 'WARNING' )
369:     end if
370: C
371: C ===== POINT-DETECTOR (3/14/1991)
372: C
373:   else if ( NHYP.eq.1
374:     &      .and. IMATCH('POIN*-DETE*',LINE(IS:IE)).eq.1 ) then
375: C
376:     JPTDT = JJ
377:     if ( JJ.eq.1 ) JSTATX = JJ
378: C
379: C ===== TALLY-LATTICE (1992/3/18)
380: C
381:   else if ( NHYP.eq.1.and.IMATCH('TALL*-LATT*',LINE(IS:IE)).eq.1 )
382:     &      then
383:       JTLLT = JJ
384: C
385: C ===== UNIVERSE-DEPENDENT-TALLY
386: C FRAME-DEPENDENT-TALLY
387: C ( ALIAS OF TALLY-LATTICE )
388: C
389:   else if ( NHYP.eq.2.and.IMATCH('UNIV*-DEPE*-TALL*',LINE(IS:IE))
390:     &      .eq.1 .or. IMATCH('FRAM*-DEPE*-TALL*',LINE(IS:IE)).eq.1 )

```

src/gmvp/option.f

```

391:      &      then
392:      JTLT = JJ
393:
394: C      WRITE(IPR,*) ' =CHECK  OPTION == ',JPRTS
395: C
396: C      << OPTION FOR DEBUGGING >>  CHECKK FOR /ARRAY/
397: C
398: C ===== MEMORY-CHECK
399: C
400:      else if ( NHYP.eq.1.and.JJ.eq.1
401:      &      .and.IMATCH('MEMO*-CHEC*',LINE(IS:IE)).eq.1 ) then
402:      JMCHK = JJ
403: C
404: C      ... SIZE OF DYNAMICALLY ALLOCATED MEMORY ...
405: C
406: C ===== DYNAMIC-MEMORY
407: C
408:      else if ( NHYP.eq.1
409:      &      .and.IMATCH('DYNA*-MEMO*',LINE(IS:IE)).eq.1 ) then
410:      if ( JJ.eq.0 ) then
411: C/#IF DYNAMIC
412:      LIMIT = MLIMIT
413: C/# IF PARA( CRAY SX* )
414: *      LIMITL = MLMTL
415: C/# ENDIF
416: C/#ENDIF
417:      if ( IBR.ne.0 ) then
418:      call DMREAD( ' ', NA, NB, IRET )
419:      IBRU = 0
420:      end if
421:      else if ( IBR.ne.0 ) then
422:      IT8(1) = 0
423:      IT8(2) = 0
424:      call I8READ( 'DYNAMIC-MEMORY', IT8, NA, -MAXPRT, IERR )
425:      if ( IERR.ne.0 ) then
426:      write(IMG,*)
427:      &      'XXX Invalid parameter for option "DYNAMIC-MEMORY".'
428:      call CNTERR( 'FATAL' )
429:      end if
430:      IBRU = 0
431: C/#IF DYNAMIC
432: C/# IF PARA( CRAY SX* )
433: *      if ( NA.gt.0 ) LIMIT = IT8(1)
434: *      if ( NA.gt.1 ) LIMITL = IT8(2)
435: C/# ELSE
436:      if ( NA.gt.0 ) LIMIT = IT8(1) + IT8(2)
437: C/# ENDIF
438:      if ( NA.eq.0 ) then
439:      write(IMG,*)
440:      &      'XXX No size-parameter given for DYNAMIC-MEMORY option.'
441:      call CNTERR( 'FATAL' )
442:      end if
443: C/#ELSE
444: *      write(IMG,7000) LIMIT
445: *      format(/1X,'!!! DYNAMIC-MEMORY option is not ',
446: *      &      'effective for this installation of the code.',
447: *      &      ' When you want to change memory size, please',
448: *      &      ' change "MAXMEM" parameter in the AAALOC routine and',
449: *      &      ' remake load-module.'/1X,' (current limit = ',I10,
450: *      &      ' words)')
451: *      call CNTERR( 'WARNING' )
452: C
453: *      if ( NA.gt.0.and.IT(1)+IT(2).gt.LIMIT ) then
454: *      write(IMG,*) '!!! Your demand for dynamic memory',
455: *      &      ' exceeds fixed limit of ', LIMIT, ' words of',

```

```

456: *      &      ' current load-module.'
457: *      write(IMG,*) ' It may cause error-stop before ',
458: *      &      'starting histories.'
459: *      call CNTERR( 'WARNING' )
460: *      end if
461: C/#ENDIF
462:      else
463:      write(IMG,*)
464:      &      'XXX No size-parameter given for DYNAMIC-MEMORY option.'
465:      call CNTERR( 'FATAL' )
466:      end if
467: C
468: C ===== RUN-MODE[(mode)]
469: C
470:      else if ( NHYP.eq.1.and.IMATCH('RUN-MODE*',LINE(IS:IE)).eq.1 )
471:      &      then
472:      if ( JJ.eq.0 ) then
473:      JRUNM = 0
474:      if ( IBR.ne.0 ) then
475:      call DMREAD( ' ', NA, NB, IRET )
476:      IBRU = 0
477:      end if
478:      else if ( IBR.ne.0 ) then
479:      call I4READ( 'RUN-MODE', IT, NA, -1, IERR )
480:      if ( IERR.ne.0 ) then
481:      write(IMG,*)
482:      &      'XXX Invalid parameter for option "RUN-MODE".'
483:      call CNTERR( 'FATAL' )
484:      end if
485:      IBRU = 0
486:      if ( NA.gt.0 ) then
487:      JRUNM = IT(1)
488:      else
489:      write(IMG,*)
490:      &      '!!! No data for RUN-MODE option.'
491:      call CNTERR( 'WARNING' )
492:      end if
493:      else
494:      JRUNM = 0
495:      write(IMG,*)
496:      &      '!!! No data for RUN-MODE option.'
497:      call CNTERR( 'WARNING' )
498:      end if
499: C
500: C      ... NUMBER OF TASKS FOR MULTI-TASKING MODE ...
501: C
502: C ===== MULTI-TASK
503: C
504:      else if ( NHYP.eq.1
505:      &      .and.IMATCH('MULT*-TASK*',LINE(IS:IE)).eq.1 ) then
506:      if ( JJ.eq.0 ) then
507:      NTASK = NTASK0
508:      if ( IBR.ne.0 ) then
509:      call DMREAD( ' ', NA, NB, IRET )
510:      IBRU = 0
511:      end if
512:      else if ( IBR.ne.0 ) then
513:      IT(1) = 0
514:      call I4READ( 'MULTI-TASK', IT, NA, -MAXPRT, IERR )
515:      if ( IERR.ne.0 ) then
516:      write(IMG,*)
517:      &      'XXX Invalid parameter for option "MULTI-TASKING".'
518:      call CNTERR( 'FATAL' )
519:      end if
520:      if ( NA.gt.0 ) then

```

src/gmvp/option.f

```

521:          NTASK    = IT(1)
522:        else
523:          write(IMG,*)
524:          &          'XXX No task-number given for MULTI-TASK option.'
525:          call CNTERR( 'FATAL' )
526:        end if
527:        IBRU    = 0
528:      else
529:        write(IMG,*)
530:        &          'XXX No task-number given for MULTI-TASK option.'
531:        call CNTERR( 'FATAL' )
532:      end if
533: C
534: C      ... spawn "tasker" in multitask mode if necessary (only in PVM)
535: C
536: C ===== SPAWN-TASKER
537: C
538: C      else if ( NHYP.eq.1.and.IMATCH('SPAW*-TASK*',LINE(IS:IE)).eq.1 )
539: C      &      then
540: C          JTSKR    = JJ
541: C
542: C      ... run multitasking mode in a way to obtain reproducible result
543: C
544: C ===== REPRODUCIBLE-RUN
545: C
546: C      else if ( NHYP.eq.1.and.IMATCH('REPR*-RUN',LINE(IS:IE)).eq.1 )
547: C      &      then
548: C          JREPR    = JJ
549: C
550: C      ... save source data on file
551: C      (currently data of last batch on eigenvalue problem)
552: C
553: C ===== SOURCE-OUTPUT
554: C
555: C      else if ( NHYP.eq.1.and.IMATCH('SOUR*-OUT*',LINE(IS:IE)).eq.1 )
556: C      &      then
557: C          JOSRC    = JJ
558: C
559: C      ... Open scratch I/O units if they are not opened ...
560: C
561: C ===== OPEN-SCRATCH
562: C
563: C      else if ( NHYP.eq.1.and.IMATCH('OPEN-SCRA*',LINE(IS:IE)).eq.1 )
564: C      &      then
565: C          JSCRT    = JJ
566: C
567: C      ... output restart file. Restart file is output by default,
568: C      so this option is used to suppress restart file output.
569: C
570: C ===== RESTART-FILE
571: C
572: C      else if ( NHYP.eq.1.and.IMATCH('REST*-FILE',LINE(IS:IE)).eq.1 )
573: C      &      then
574: C          JRSTF    = JJ
575: C
576: C ===== SUBTASK-PRINT
577: C      (but this option should be controled by environment variable
578: C      or command line!!)
579: C
580: C      else if ( NHYP.eq.1.and.IMATCH('SUBT*-PRIN*',LINE(IS:IE)).eq.1 )
581: C      &      then
582: C          if ( JJ.eq.0 ) then
583: C              JSTPRN(1) = 1
584: C /#IF PARA(SX* CRAY*)
585: *          JSTPRN(1) = 2

```

```

586: C /#ENDIF
587:          JSTPRN(2) = 2
588: C
589: C      if ( IBR.ne.0 ) then
590: C          call DMREAD( ' ', NA, NB, IRET )
591: C          IBRU    = 0
592: C      end if
593: C      else if ( IBR.ne.0 ) then
594: C          IT(1) = 0
595: C          call I4READ( 'SUBTASK-PRINT', IT, NA, -MAXPRT, IERR )
596: C          if ( IERR.ne.0 ) then
597: C              write(IMG,*)
598: C              &          'XXX Invalid parameter for option "SUBTASK-PRINT".'
599: C              call CNTERR( 'FATAL' )
600: C          end if
601: C          if ( NA.gt.0 ) JSTPRN(1) = IT(1)
602: C          if ( NA.gt.1 ) JSTPRN(2) = IT(2)
603: C          if ( NA.eq.0 ) then
604: C              write(IMG,*)
605: C              &          '!!! No data given to SUBTASK-PRINT() option.'
606: C              call CNTERR( 'WARNING' )
607: C          end if
608: C          IBRU    = 0
609: C      else
610: C          write(IMG,*) '!!! No data given to SUBTASK-PRINT() option.'
611: C          call CNTERR( 'WARNING' )
612: C      end if
613: C      else
614: C
615: C ..... ERROR .....
616: C
617: C          write(IMG,/'(1x,'XXX Invalid option: ',A)') LINE(IS:IE)
618: C          call CNTERR( 'FATAL' )
619: C      end if
620: C
621: C ..... SKIP UNUSED NUMERIC PARAMETERS ....
622: C
623: C      if ( IBR.ne.0.and.IBRU.ne.0 ) then
624: C          call DMREAD( ' ', NA, NB, IRET )
625: C          write(IMG,7020) LINE(IS:IE)
626: C 7020      format(/1X,'!!! Numeric parameters are given but not',
627: C              &          ' effective for option <',A,'>')
628: C          call CNTERR( 'WARNING' )
629: C      end if
630: C
631: C      go to 100
632: C
633: C 150 continue
634: C
635: C ..... ERROR OR WARNING .....
636: C          AND ADJUSTMENT OF FLAGS
637: C
638: C      if ( JEIGN.ne.0.and.JFISS.eq.0 ) then
639: C          write(IPR,/'(1x,a,a)')
640: C          &          '<<MESSAGE>> "FISSION" OPTION IS SET ACTIVE BECAUSE ',
641: C          &          'YOU ARE GOING TO SOLVE AN EIGENVALUE PROBLEM.'
642: C          JFISS = 1
643: C          call CNTERR( 'MESSAGE' )
644: C      end if
645: C
646: C      if ( JTLT.ne.0.and.JLATT.eq.0 ) then
647: C          write(IMG,/'(1x,a,a)')
648: C          &          '!!! "TALLY-LATTICE" or "FRAME-DEPENDENT-TALLY" ',
649: C          &          'option needs "LATTICE" option.'
650: C          JTLT = 0

```

src/gmvp/option.f

```

651:      call CNTERR( 'WARNING' )
652:      end if
653: C
654: C/#IF PARA
655: C
656:      if ( JREPR.eq.0.and.JEIGN.ne.0 ) then
657:          write(IMG,'(/lx,a,a)')
658:          &      '!!! Eigenvalue problems with the current multitasking ',
659:          &      'version are always reproducible.'
660:          JREPR = 1
661:          call CNTERR( 'WARNING' )
662:      end if
663: C
664: C/#ELSE
665: C
666:      if ( JREPR.eq.0 ) then
667:          write(IMG,'(/lx,a,a/lx,a)')
668:          &      '!!! In single task version, the result is always',
669:          &      'reproducible.',
670:          &      'So "NO-REPRODUCIBLE-RUN" has no meaning.'
671:          JREPR = 1
672:          call CNTERR( 'WARNING' )
673:      end if
674: C/#ENDIF
675: C
676: C
677:      if ( JFPRT.eq.1 ) JPRTS(1) = 0
678: C
679: CCCC IF( JONEZ.NE.0.AND.JALLZ.NE.0 ) THEN
680: CCC  WRITE(IPR,*) ' !!! JALLZ = 1, ALTHOUGH JONEZ = 1. JALLZ IS SET'
681: CCC +      ', TO 0, AND ONE-ZONE GEOMETRICAL TRACKING IS PERFORMED!'
682: CCC  JALLZ = 0
683: CCCC ENDIF
684: C
685: C ..... PRINTOUT OPTION PARAMETERS
686: C
687:      write(IPR,7040)
688: 7040 format(/lx,' **** OPTION PARAMETERS (1/0 = ON/OFF) **** '//
689:      &      ' FLAG      KEY-WORD      ON/OFF      INCOMPATIBLE OPTIONS'//
690:      &      )
691: 7060 format(lx,2X,A,T10,2X,A20,2X,I2,:4X,5(:' (' ,A,')'))
692: C
693:      write(IPR,7060) 'JREST', 'RESTART', 'JREST
694:      write(IPR,7060) 'JARST', 'AUTO-RESTART', 'JARST
695:      write(IPR,7060) 'JRSTF', 'RESTAR-FILE', 'JRSTF
696:      write(IPR,7060) 'JRUNM', 'RUN-MODE', 'JRUNM
697:      write(IPR,7060) 'JNEUT', 'NEUTRON', 'JNEUT
698:      write(IPR,7060) 'JPHOT', 'PHOTON', 'JPHOT
699:      write(IPR,7060) 'JADJM', 'ADJOINT CALCULATION', 'JADJM
700: CCCC WRITE(IPR,7200) 'JIMAG', 'IMAGINARY-PARTICLE', 'JIMAG,
701: Cccc &      'JNEUT', 'JPHOT'
702:      write(IPR,7060) 'JIMAG', 'IMAGINARY-PARTICLE', 'JIMAG
703:      write(IPR,7060) 'JNCOL', 'COLLISION-LESS', 'JNCOL
704:      write(IPR,7060) 'JTIME', 'TIME-DEPENDENT', 'JTIME
705:      write(IPR,7060) 'JPTIM', 'PERIODIC-TIME', 'JPTIM
706: CC  WRITE(IPR,7200) 'JDELT', 'DELTA-TRACKING', 'JDELT
707:      write(IPR,7060) 'JEIGN', 'EIGEN-VALUE', 'JEIGN, 'JFIXD'
708:      write(IPR,7060) 'JFIXD', 'FIXED-SOURCE', 'JFIXD, 'JEIGN'
709:      write(IPR,7060) 'JFISS', 'FISSION PROBLEM', 'JFISS
710:      write(IPR,7060) 'JOSRC', 'SOURCE-OUTPUT', 'JOSRC
711:      write(IPR,7060) 'JRRLT', 'RUSSIAN-ROULETTE', 'JRRLT, 'JWWND'
712:      write(IPR,7060) 'JIMPT', 'IMPORTANCE', 'JIMPT, 'JWWND'
713:      write(IPR,7060) 'JWWND', 'WEIGHT-WINDOW', 'JWWND, 'JIMPT',
714:      &      'JRRLT'
715:      write(IPR,7060) 'JPSTR', 'PATH-STRETCHING', 'JPSTR

```

```

716:      write(IPR,7060) 'JRWVR', 'RELATIVE-WEIGHT', 'JRWVR
717: CCC  WRITE(IPR,7200) 'JFCOL', 'FORCED-COLLISION', 'JFCOL
718: CCC  WRITE(IPR,7200) 'JBIAS', 'SOURCE-BIAS', 'JBIAS
719:      write(IPR,7060) 'JRESP', 'RESPONSE', 'JRESP
720:      write(IPR,7060) 'JMNTR', 'MONITOR', 'JMNTR
721:      write(IPR,7060) 'JVMNT', 'VP-MONITOR', 'JVMNT
722:      do 160 I = 1, MAXJDB
723:          if ( I.eq.1 .or. JDEBG(I).ne.0 ) then
724:              LINE = ' '
725:              write(LINE,'(''DEBUG-PRINT ('',i2,'')'')' ) I
726:              call CCOMP( LINE, LEN(LINE), LINE, IERR )
727:              write(IPR,7060) 'JDEBG', LINE(:20), JDEBG(I)
728:          end if
729: 160 continue
730:      write(IPR,7060) 'JSCRT', 'OPEN-SCRATCH', 'JSCRT
731:      write(IPR,7060) 'JALLZ', 'EVENT-SELECTION', 'JALLZ, 'JONEZ'
732:      write(IPR,7060) 'JONEZ', 'ZONE-SELECTION', 'JONEZ, 'JALLZ'
733:      write(IPR,7060) 'JREPR', 'REPRODUCIBLE-RUN', 'JREPR
734:      write(IPR,7060) 'JTSKR', 'SPAWN-TASKER', 'JTSKR
735:      write(IPR,7060) 'JLATT', 'LATTICE', 'JLATT
736:      write(IPR,7060) 'JFPRT', 'FLUX-PRINT', 'JFPRT
737:      write(IPR,7060) 'JPTDT', 'POINT-DETECTOR', 'JPTDT
738:      write(IPR,7060) 'JTLLT', 'TALLY-LATTICE', 'JTLLT
739:      do 240 I = 1, 2
740:          LINE = ' '
741:          write(LINE,'(''SUBTASK-PRINT ('',i2,'')'')' ) I
742:          call CCOMP( LINE, LEN(LINE), LINE, IERR )
743:          write(IPR,7060) 'JSTPRN', LINE(:20), JSTPRN(I)
744: 240 continue
745:      write(IPR,*) ' '
746:      if ( JPRTS(1).eq.1 ) write(IPR,*)
747:      &      ' * PRINT OF GROUP-WISE FLUX IS SUPPRESSED.'
748:      if ( JPRTS(2).eq.1 ) write(IPR,*)
749:      &      ' * PRINT OF ANGULAR FLUX IS SUPPRESSED.'
750:      if ( JPRTS(3).eq.1 ) write(IPR,*)
751:      &      ' * PRINT OF TIME-DEPENDENT FLUX IS SUPPRESSED.'
752:      if ( JPRTS(4).eq.1 ) write(IPR,*)
753:      &      ' * PRINT OF RESPONSE IS SUPPRESSED.'
754:      if ( JPRTS(5).eq.1 ) write(IPR,*)
755:      &      ' * PRINT OF TIME-DEPENDENT RESPONSE ', 'IS SUPPRESSED.'
756:      if ( JPRTS(6).eq.1 ) write(IPR,*)
757:      &      ' * PRINT OF SOURCE INFOMATION #1 ', 'IS SUPPRESSED.'
758:      if ( JPRTS(7).eq.1 ) write(IPR,*)
759:      &      ' * PRINT OF SOURCE INFOMATION #2 ', 'IS SUPPRESSED.'
760:      return
761:      end

```

src/gmvp/pgvel.f

```
1:      subroutine PGVEL( VEL,   NGP1,  NGP2,  NGROUP )
2: C=====
3: C  PURPOSE: set photon velocity
4: C  CALLED IN: INTRO2
5: C=====
6: C  arguments (i=input, o=output, w=work )
7: C
8: C  o vel(ngroup) : averaged neutron velocity ( cm/s )
9: C  i ngp1      : number of neutron energy groups
10: C  i ngp2      : number of photon energy groups
11: C  i ngroup    : total number of energy groups
12: C=====
13:      implicit real*8(D)
14: C
15: C
16:      real VEL(NGROUP)
17: C
18: C
19: C ... constants
20: C
21: C --- light speed --- (cm/s)
22: C
23:      real*8 CLIGHT
24:      parameter( CLIGHT  = 2.99792458D10 )
25: C
26: C-----
27: C
28:      do 100 I = 1, NGP2
29:          VEL(NGP1+I) = CLIGHT
30:      100 continue
31: C
32:      return
33:      end
```

src/gmvp/popara.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine POPARA
3: C=====
4: C  PURPOSE:  This routine is obsolete ( June 1997 )
5: C           Function is moved to MTSUMS.
6: C           ( summation of task tallies (FACOM VPP) )
7: C  CALLED IN:  CENTER
8: C  CALLS  PESUM8
9: C
10: C  UPDATE:
11: C=====
12: C/#!IF PARA(VPP)
13: C   include '../shared/INC/_ARRAY'
14: C   include 'INC/_FLAGS'
15: C   include '../shared/INC/_IOUNIT'
16: C   include '../shared/INC/_TASKDT'
17: C   include '../shared/INC/_VPPPARA'
18: C   include '../shared/INC/_PTALY0'
19: C   include 'INC/_PTALY'
20: C   include 'INC/_PTALY2'
21: C   include 'INC/_SIZES2'
22: C   include '../shared/INC/_SIZES'
23: C   include '../shared/INC/_STALY'
24: C
25: C!xocl processor p( mpe )
26: C   call PESUM8( TMACT(1), NTASK )
27: C
28: C   call PESUMI( NTHIST, 1 )
29: C   if ( JEIGN.eq.0 ) call PESUMI( NBATCH, 1 )
30: C   call PESUM8( A(LWSUM), 1 )
31: C   call PESUM8( A(LWLEK), 1 )
32: C   call PESUM8( A(LNCNTR), NEVENT )
33: C   call PESUM8( A(LWCNTR), NEVENT )
34: C
35: C   call PESUM8( A(LSFLTR), NGROUP*NTREG*2 )
36: C   call PESUM8( A(LSFLCL), NGROUP*NTREG*2 )
37: C
38: C   if ( JRESP.ne.0.and.NRESP.gt.0 ) then
39: C     call PESUM8( A(LSRETR), NTREG*NRESP*2 )
40: C     call PESUM8( A(LSRECL), NTREG*NRESP*2 )
41: C   end if
42: C
43: C   if ( JPTDT.ne.0 ) then
44: C     call PESUM8( A(LSFLPD), NGROUP*NPDET*2 )
45: C     if ( JRESP.ne.0 ) call PESUM8( A(LSREPD), NRESP*NPDET*2 )
46: C   end if
47: C
48: C   if ( JEIGN.eq.1 ) then
49: C     call PESUM8( A(LXSOC), NBATCH )
50: C     call PESUM8( A(LXKEF), NBATCH*5 )
51: C   end if
52: C
53: C   if ( NSTALY.gt.0 ) then
54: C     call PESUM8( A(LETALY), NLETAL*2 )
55: C   end if
56: C
57: C   if ( JTIME.gt.0 ) then
58: C     call PESUM8( A(LTFLHS), NPKIND*2 )
59: C   end if
60: C
61: C   if ( JMNTR.ne.0 ) then
62: C     call PESUM8( A(LNCLSN), NGROUP*NREG )
63: C     call PESUM8( A(LWCLSN), NGROUP*NREG )
64: C     call PESUM8( A(LNLEAK), NGROUP*NREG )
65: C     call PESUM8( A(LWLEAK), NGROUP*NREG )

```

```

66: C     call PESUM8( A(LNABSB), NGROUP*NREG )
67: C     call PESUM8( A(LWABSB), NGROUP*NREG )
68: C     call PESUM8( A(LNECUT), NREG )
69: C     call PESUM8( A(LWECUT), NREG )
70: C     call PESUM8( A(LNTCUT), NGROUP*NREG )
71: C     call PESUM8( A(LWTCUT), NGROUP*NREG )
72: C     call PESUM8( A(LNKILD), NGROUP*NREG )
73: C     call PESUM8( A(LWKILD), NGROUP*NREG )
74: C     call PESUM8( A(LNSURV), NGROUP*NREG )
75: C     call PESUM8( A(LWSURV), NGROUP*NREG )
76: C     call PESUM8( A(LNSPLT), NGROUP*NREG )
77: C     call PESUM8( A(LWSPLT), NGROUP*NREG )
78: C   end if
79: C
80: C/#!ENDIF
81: C
82: C   return
83: C   end

```

src/gmvp/prdate.c

```
1: /*****
2:  * Output date and time of compilation on standard output
3:  * using ANSI C predefined macros if possible.
4:  *
5:  * 12 Aug 1999: add fflush(stdout).
6:  *****/
7:
8: #include <stdio.h>
9:
10: void prdate()
11: {
12:     print_date_time() ;
13: }
14: void prdate_()
15: {
16:     print_date_time() ;
17: }
18: void PRDATE()
19: {
20:     print_date_time() ;
21: }
22: void _PRDATE()
23: {
24:     print_date_time() ;
25: }
26:
27: int print_date_time()
28: {
29:     int j ;
30:     j = 0 ;
31: #ifdef __DATE__
32: #ifdef __TIME__
33:     j = 1 ;
34:     printf("      COMPILATION DATE: %s  TIME: %s\n", __DATE__, __TIME__);
35: #endif
36: #endif
37:     if ( j == 0 ) {
38:         printf("      COMPILATION DATE: UNKNOWN\n");
39:     }
40:     fflush(stdout) ;
41: }
```


src/gmvp/preinp.f

```

1:      subroutine PREINP( IPREIN,CSTRNG,NTASK0,TSKINF,MLIMIT )
2:      C=<GMVP>=====
3:      C Purpose: Interpret command string for file-name/I/O-unit coupling
4:      C or default-option overriding etc.
5:      C Probably called from main routine before calling of 'CENTER'
6:      C routine, or called before reading input to MVP/GMVP.
7:      C
8:      C=====
9:      C argument :
10:     C
11:     C i iprein : counter of calling for this routine given externally.
12:     C (when iprein = 1 , initialize data )
13:     C i cstr : command string
14:     C
15:     C /nn[rfl][:file-name]
16:     C
17:     C Assign I/O unit nn to a file.
18:     C
19:     C xxx=yyy
20:     C
21:     C Execution parameter setting.
22:     C
23:     C o ntask0 : number of task effective in multiprocessing mode
24:     C
25:     C NTASK=... or ntask=...
26:     C
27:     C o tskinf : name of task control information file in multiprocessing
28:     C mode.
29:     C
30:     C TASKINFO=... or taskinfo=...
31:     C
32:     C o mlimit : memory size limit (effective only in dynamic allocation
33:     C mode )
34:     C
35:     C=====
36:     C
37:     C include '../shared/INC/_IOUNIT'
38:     C
39:     C ... external data ...
40:     C
41:     C character*(*) CSTRNG
42:     C character*(*) TSKINF
43:     C include 'INC/_FLAGS'
44:     C/#IF PARA(PVM)
45:     C include '../shared/INC/_PVMPARA'
46:     C/#ELSEIF PARA(MPI)
47:     C * include 'mpif.h'
48:     C/#ENDIF
49:     C include '../shared/INC/_TASKDT'
50:     C
51:     C ... local data ...
52:     C
53:     C character*20 CTEMP,CTEMP2
54:     C
55:     C-----
56:     C
57:     C ... initialization on first call ...
58:     C
59:     C if ( IPREIN.eq.1 ) then
60:     C JRPCPU = 0
61:     C/#IF PARA(PVM MPI)
62:     C * HOST0 = ' '
63:     C/#ENDIF
64:     C end if
65:     C

```

```

66:     C
67:     C
68:     C write(IPR,'(1x,a,a,a)') ' PARAMETER : < ', CSTRNG, '>'
69:     C
70:     C IE = 0
71:     C
72:     C .... 'CSTRNG' may be broken into some commands separated by blank.
73:     C
74:     C
75:     C 100 IS = IE + 1
76:     C call NOBLNK( CSTRNG, IS, IE, LEN(CSTRNG) )
77:     C
78:     C
79:     C ... cstrng(is:ie) : Command ....
80:     C
81:     C if ( IE.le.LEN(CSTRNG) ) then
82:     C
83:     C-----
84:     C File allocation
85:     C-----
86:     C if ( CSTRNG(IS:IS).eq.'/' ) then
87:     C
88:     C
89:     C JJJJJ = JFOPEN(CSTRNG(IS:IE))
90:     C
91:     C-----
92:     C Number of task in multitasking mode
93:     C-----
94:     C
95:     C else if ( CSTRNG(IS:IS+5).eq.'ntask='
96:     C & .or. CSTRNG(IS:IS+5).eq.'NTASK=' ) then
97:     C
98:     C CTEMP = CSTRNG(IS+6:IE)
99:     C read(CTEMP(:20),'(bn,i20)') NTASK0
100:    C
101:    C-----
102:    C Hosts name of the current task in multitasking mode
103:    C-----
104:    C
105:    C else if ( CSTRNG(IS:IS+10).eq.'MYHOSTNAME='
106:    C & .or. CSTRNG(IS:IS+10).eq.'myhostname=' ) then
107:    C
108:    C/#IF PARA(PVM MPI)
109:    C * HOST0 = CSTRNG(IS+11:IE)
110:    C/#ELSE
111:    C write(IPR,*) 'This parameter is not effective in this mode.'
112:    C/#ENDIF
113:    C
114:    C-----
115:    C taskinfo=<the name of file which contains MVP/GMVP specific task
116:    C control parameters>
117:    C ( effective in PVM , NCUBE etc. )
118:    C-----
119:    C
120:    C else if ( CSTRNG(IS:IS+8).eq.'taskinfo='
121:    C & .or. CSTRNG(IS:IS+8).eq.'TASKINFO=' ) then
122:    C TSKINF = CSTRNG(IS+9:IE)
123:    C
124:    C-----
125:    C Sub-task file name in multitasking mode
126:    C-----
127:    C
128:    C else if ( CSTRNG(IS:IS+10).eq.'SUBTASKOUT='
129:    C & .or. CSTRNG(IS:IS+10).eq.'subtaskout=' ) then
130:    C

```

src/gmvp/preinp.f

```
131: C/#IF PARA( PVM MPI )
132: *      STFN = cstrng(is+11:ie)
133: C/#ELSE
134:      write(IPR,*) '!!!(PREINP) This parameter is not effective ',
135:      &          'in this mode.'
136: C/#ENDIF
137: C
138: C-----
139: C      Override default options or sizes
140: C-----
141: C      DEBUG-PRINT ( OFF -> ON ) : [NO-]DEBUG-PRINT
142: C
143: C      VP-MONITOR ( OFF -> ON ) : [NO-]VP-MONITOR
144: C
145: C      MONITOR ( OFF -> ON ) : [NO-]MONITOR
146: C
147: C      IMAGINARY-PARTICLE( OFF->ON ) : [NO-]IMAGINARY-PARTICLE
148: C
149: C      DYNAMIC-MEMORY ( NEW_SIZE IN WORD ) : MEMORY=size
150: C.....
151: C
152:      else if ( IMATCH('*DEBU*-PRIN*',CSTRNG(IS:IE)).eq.1 ) then
153:          JDEBG(1) = 1
154:          if ( CSTRNG(IS:IS+2).eq.'NO-' ) JDEBG(1) = 0
155:          go to 110
156:      else if ( IMATCH('*VP-MONI*',CSTRNG(IS:IE)).eq.1 ) then
157:          JVMNT = 1
158:          if ( CSTRNG(IS:IS+2).eq.'NO-' ) JVMNT = 0
159:          go to 110
160:      else if ( IMATCH('*MONI*',CSTRNG(IS:IE)).eq.1 ) then
161:          JMNTR = 1
162:          if ( CSTRNG(IS:IS+2).eq.'NO-' ) JMNTR = 0
163:          go to 110
164:      else if ( IMATCH('*IMAG*-PART*',CSTRNG(IS:IE)).eq.1 ) then
165:          JIMAG = 1
166:          if ( CSTRNG(IS:IS+2).eq.'NO-' ) JIMAG = 0
167:          go to 110
168:      else if ( IMATCH('RUN-MODE=*',CSTRNG(IS:IE)).eq.1 ) then
169:          CTEMP = CSTRNG(IS+9:IE)
170:          read(CTEMP(:20),'(bn,i20)') JRUNM
171:          go to 110
172: C/#IF DYNAMIC
173:      else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
174:          CTEMP = CSTRNG(IS+7:IE)
175:          read(CTEMP(:20),'(bn,i20)') MLIMIT
176:          go to 110
177: C/#ELSE
178:      else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
179:          write(IPR,'(lx,a)')
180:          &          '<MEMORY=...> has meanings in dynamic memory mode.'
181:          go to 110
182: C/#ENDIF
183: C
184: C      SUBTASK-PRINT=i1[,i2]
185: C
186:      else if ( IMATCH('SUBT*-PRIN*',CSTRNG(IS:IE)).eq.1 ) then
187:          KK = index(CSTRNG(IS:IE),'=')
188:          if ( KK.ne.0 ) then
189:              CTEMP = CSTRNG(IS+KK:IE)
190:              K2 = index(CTEMP,',')
191:              if ( K2.eq.0 ) then
192:                  read(CTEMP(:20),'(BN,I20)') JSTPRN(1)
193:              else
194:                  CTEMP2 = CTEMP(K2+1:)
195:                  CTEMP(K2:) = ' '
```

```
196:          read(CTEMP(:20),'(BN,I20)') JSTPRN(1)
197:          read(CTEMP2(:20),'(BN,I20)') JSTPRN(2)
198:          end if
199:          end if
200:          go to 110
201: C-----
202: C      unsupported command
203: C-----
204: C
205:      else
206:          write(IMG,'(lx,a,a)')
207:          &          '!!! Unsupported command line parameter: ',CSTRNG(IS:IE)
208:          end if
209: C
210:      110      continue
211:          go to 100
212:      end if
213: C
214:      return
215:      end
```

src/gmvp/prmntr.f

```

1: *VOCL TOTAL,SCALAR
2: subroutine PRMNTR( IOT, A, CHA, JMNTR, NGROUP,NREG, JTIME, NCLSN,
3: & WCLSN, NLEAK, WLEAK, NABSB, WABSB, NECUT,
4: & WECUT, NTCUT, WTCUT, NKILD, WKILD, NSURV,
5: & WSURV, NCNTR, WCNTR )
6: C=====
7: C PURPOSE: PRINT OUT MONITORING DATAS.
8: C CALLED IN: CADENZ
9: C=====
10: implicit real*8(D,W)
11: C
12: real A(*)
13: character*4 CHA(*)
14: C
15: real*8 NCLSN(NGROUP,NREG), NLEAK(NGROUP,NREG), NABSB(NGROUP,NREG),
16: & NECUT(NREG), NTCUT(NGROUP,NREG), NKILD(NGROUP,NREG),
17: & NSURV(NGROUP,NREG)
18: real*8 WCLSN(NGROUP,NREG), WLEAK(NGROUP,NREG), WABSB(NGROUP,NREG),
19: & WECUT(NREG), WTCUT(NGROUP,NREG), WKILD(NGROUP,NREG),
20: & WSURV(NGROUP,NREG)
21: real*8 NCNTR(*), WCNTR(*)
22: C
23: call HEADER( IOT, 'MONITORING DATA' )
24: C
25: DW = WCNTR(1)
26: if (DW.eq.0.0d0) DW = 1.0d0
27: C
28: C WRITE(IOT,7000) (NCNTR(N),WCNTR(N),N=1,12)
29: write(IOT,7000) NCNTR(1),WCNTR(1),
30: & NCNTR(2),WCNTR(2), NCNTR(2)/DW,WCNTR(2)/DW,
31: & NCNTR(13), WCNTR(13), NCNTR(13)/DW, WCNTR(13)/DW,
32: & (NCNTR(N),WCNTR(N), NCNTR(N)/DW,WCNTR(N)/DW, N=3,12)
33:
34: if ( JTIME.ne.0 ) then
35: write(IOT,7020) NCNTR(26), WCNTR(26),
36: & NCNTR(26)/DW, WCNTR(26)/DW
37: end if
38: write(IOT,7030) NCNTR(27), WCNTR(27),
39: & NCNTR(27)/DW, WCNTR(27)/DW
40: write(IOT,7032) NCNTR(28), WCNTR(28),
41: & NCNTR(28)/DW, WCNTR(28)/DW
42: write(IOT,7040) (NCNTR(N),WCNTR(N)/DW,N=16,18)
43: C
44: 7000 format(/1X,58X,' TOTAL PER SOURCE WEIGHT'
45: & /1X,58X,'COUNT WEIGHT SUM COUNT WEIGHT'
46: & //10X,
47: & 'SOURCE PARTICLES ',F16.0,2X,E13.6/10X,
48: & 'FISSION REACTION (WHEN JEIGN=0) ',F16.0,3(2X,E13.6)/10X,
49: & 'FISSION REACTION PREVENTED (JEIGN=0) ',F16.0,3(2X,E13.6)/10X,
50: & '(N,GAMMA) REACTION ',F16.0,3(2X,E13.6)/10X,
51: & 'COLLISION ',F16.0,3(2X,E13.6)/10X,
52: & 'SPLITTING (IMPORTANCE OR WEIGHT WINDOW) ',F16.0,3(2X,E13.6)/10X,
53: & 'SPLITTING PREVENTED ',F16.0,3(2X,E13.6)/10X,
54: & 'LEAKAGE ',F16.0,3(2X,E13.6)/10X,
55: & 'ENERGY CUTOFF ',F16.0,3(2X,E13.6)/10X,
56: & 'KILLED (IMPORTANCE OR WEIGHT WINDOW) ',F16.0,3(2X,E13.6)/10X,
57: & 'SURVIVED (IMPORTANCE OR WEIGHT WINDOW) ',F16.0,3(2X,E13.6)/10X,
58: & 'KILLED (WEIGHT CUTOFF) ',F16.0,3(2X,E13.6)/10X,
59: & 'SURVIVED (WEIGHT CUTOFF) ',F16.0,3(2X,E13.6))
60: 7020 format(/10X,
61: & 'TIME CUTOFF ',F16.0,3(2X,E13.6))
62: 7030 format(10X,
63: & 'PARTICLE GENERATION CUTOFF ',F16.0,3(2X,E13.6))
64: 7032 format(10X,
65: & 'ENDLESS LOOP CUTOFF ',F16.0,3(2X,E13.6))

```

```

66: 7040 format(/10X,
67: & 'NUMBER OF FREE FLIGHT ',F16.0,15X,2X,E13.6/10X,
68: & 'NUMBER OF BOUNDARY CROSSING ',F16.0,15X,2X,E13.6/10X,
69: & 'NUMBER OF REFLECTION ',F16.0,15X,2X,E13.6)
70: if ( JMNTR.eq.0 ) return
71: C
72: write(IOT,'/1X,'DETAILED MONITORING DATA FROM NEXT PAGE ''')
73: write(IOT,7060) ' <COLLISION> '
74: 7060 format('1',A40)
75: call PRNUWE( IOT, NCLSN, WCLSN, NGROUP, NREG, 'GROUP ', 'REGION',
76: & A, CHA )
77: CCCC WRITE(IOT,7100) ' <LEAKAGE> '
78: CCCC CALL PRNUWE(IOT,NLEAK,WLEAK,NGROUP,NREG,'GROUP ', 'REGION',
79: CCCC & A, CHA)
80: CCCC WRITE(IOT,7100) ' <ABSORPTION> '
81: CCCC CALL PRNUWE(IOT,NABSB,WABSB,NGROUP,NREG,'GROUP ', 'REGION',
82: CCCC & A, CHA)
83: if ( JTIME.ne.0 ) then
84: write(IOT,7060) ' <KILLED PARTICLE BY TIME CUTOFF> '
85: call PRNUWE( IOT, NTCUT, WTCUT, NGROUP, NREG, 'GROUP ',
86: & 'REGION', A, CHA )
87: end if
88: CCCC WRITE(IOT,7100) ' <KILLED PARTICLE> '
89: CCCC CALL PRNUWE(IOT,NKILD,WKILD,NGROUP,NREG,'GROUP ', 'REGION',
90: CCCC & A, CHA)
91: CCCC WRITE(IOT,7100) ' <SURVIVED PARTICLE> '
92: CCCC CALL PRNUWE(IOT,NSURV,WSURV,NGROUP,NREG,'GROUP ', 'REGION',
93: CCCC & A, CHA)
94: write(IOT,7060) ' <KILLED PARTICLE BY ENERGY CUTOFF>'
95: write(IOT,7080) (I,NECUT(I),WECUT(I),I=1,NREG)
96: 7080 format(/1X,' REGION COUNTS WEIGHT '//
97: & (1X,3X,I5,2X,F12.0,5X,E13.6))
98: return
99: end

```

src/gmvp/prnspc.f

```
1:      subroutine PRNSPC( IOG, A, CHA,   ISPACE,LEVEL )
2: C=====
3: C PURPOSE: LOST PARTICLE CHECK   ( SUBSPACE NAME )
4: C CALLED IN: LOST PARTICLE PROCESSING.
5: C=====
6:      real A(*)
7:      character*4 CHA(*)
8: C
9:      include '../shared/INC/_PTLSP'
10:     include '../shared/INC/_PGEOM'
11:     include '../shared/INC/_SIZES'
12:     include 'INC/_SIZES2'
13:     include '../shared/INC/_PTALY0'
14:     include 'INC/_PTALY'
15: CCCC include '../shared/INC/_ARRAY'
16: C
17:     call PRNSPD( IOG, ISPACE, LEVEL, A(LISPNM), CHA(LTNAMS), NNMAES,
18: &              NEST, NSPACE )
19:     return
20:     end
```

src/gmvp/prsize.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine PRSIZE( IOUT )
3: C=====
4: C  PURPOSE:  PRINTOUT OF COMMON /SIZES/
5: C  CALLED IN:  INTRO
6: C=====
7:   include '../shared/INC/_SIZES'
8:   include '../shared/INC/_STALY'
9:   include 'INC/_SIZES2'
10:  include 'INC/_PXSEC'
11:  include '../shared/INC/_TASKDT'
12: C
13: C =====
14:   call HEADER( IOUT,
15: & SIZE- OR LIMIT-PARAMETERS (SOME VALUES MAY BE GIVEN BY THE CODE)
16: & )
17: C =====
18: C
19: 7000 format(/3X,'<< ',A,' >>'/)
20: 7020 format(3X,A,T11,I10,3X,A)
21: 7040 format(3X,A,T11,I10,3X,A/3X,T11,10X,3X,A)
22: 7060 format(3X,A,T11,F10.3,3X,A)
23: 7080 format(3X,A,T11,1P,E10.3,3X,A)
24: 7100 format(3X,T11,10X,3X,A)
25: C
26: C -----
27:   write(IOUT,7000) 'HISTORY & BATCH CONTROL'
28: C -----
29: C
30:   write(IOUT,7020)
31: & 'NTASK', NTASK, 'Number of tasks in this run.',
32: & 'NPART', NPART, 'Number of histories to run (per task.)',
33: & 'NHIST', NHIST, 'Number of histories per batch',
34: & 'NHSUB', NHSUB, 'Number of histories per sub-batch',
35: & 'NTHIST', NTHIST,
36: & 'Number of histories run so far (>0 in restart mode)',
37: & 'IRAND', IRAND, 'Initial random number', 'NBATCH', NBATCH,
38: & 'Number of batches run so far (>0 in restart mode)',
39: & 'NSKIP', NSKIP, 'Skipped batch in eigenvalue calculation'
40:   write(IOUT,7040)
41: & 'NLENG', NLENG,
42: & 'Number of batches expected after eigenvalue calculation',
43: & '( >= NBATCH in restart mode)'
44:   write(IOUT,7020)
45: & 'NRSKIP', NRSKIP, 'Number of random numbers skipped'
46:   write(IOUT,7100)
47: & '(< 0 : before each batch, > 0 : before first batch)'
48:   write(IOUT,7020)
49: & 'NRSINT', NRSINT, 'Restart file output interval'
50:   write(IOUT,7100)
51: & ' > 0 : After every NRSINT''th batch',
52: & ' < 0 : After every |NRSINT|''th history',
53: & ' = 0 : Only after the last batch'
54:   write(IOUT,7020)
55: & 'NBPINT', NBPINT, 'Batch monitor printout interval'
56:   write(IOUT,7060)
57: & 'TCPU', TCPU, 'Calculation CPU time limit (minute.)'
58: C
59: C -----
60:   write(IOUT,7000) 'PARTICLE BANK'
61: C -----
62: C
63:   write(IOUT,7020) 'NBANK', NBANK, 'Length of particle bank',
64: & 'NFBANK', NFBANK, 'Length of fission source bank',
65: & 'IMPMAX', IMPMAX, 'Bank kept for next event estimator'

```

```

66: C
67: C -----
68: C
69:   write(IOUT,7000) 'GEOMETRY'
70: C -----
71: C
72:   write(IOUT,7020)
73: & 'NZONE', NZONE, 'Number of zones',
74: & 'NINPZ', NINPZ, 'Number of input zones',
75: & 'NSURF', NSURF, 'Number of surfaces',
76: & 'NSDA ', NSDA, 'Length of surface data area',
77: & 'NZDA ', NZDA, 'Length of zone data area'
78:   write(IOUT,7020)
79: & 'NLATT', NLATT, 'Number of lattices',
80: & 'NCELL', NCELL, 'Number of cells',
81: & 'NEST', NEST, 'Maximum nesting level of lattice geometry',
82: & 'NLBZ', NLBZ, 'Number of lattice-buffer zones',
83: & 'NMEMO', NMEMO, 'Number of next-zone memory per zone',
84: & 'NMEMOP', NMEMOP,
85: & 'Optimize KMEMO array while NBATCH .le. NMEMOP (default=2)',
86: & 'NREFS', NREFS, 'Number of reflection surfaces',
87: & 'NMEMS', NMEMS,
88: & 'Number of memorized starting zones for each particle source',
89: & 'NPIC', NPIC, 'Number of pictures drawn'
90: C
91: C -----
92:   write(IOUT,7000) 'TALLY'
93: C -----
94: C
95:   write(IOUT,7020)
96: & 'NREG', NREG, 'Number of regions',
97: & 'NTREG', NTREG, 'Number of tally-regions',
98: & 'NSPACE', NSPACE, 'Number of subspaces excluding level 0',
99: & 'NUNV', NUNV, 'Number of input-zones which define frames',
100: & 'NSUZON', NSUZON,
101: & 'Number of zones that belongs to regions in a subspaces.',
102: & 'NGROUP', NGROUP, 'Number of energy bins (groups)',
103: & 'NGP1', NGP1, 'Number of neutron energy bins (groups)',
104: & 'NGP2', NGP2, 'Number of photon energy bins (groups)',
105: & 'NTIME', NTIME, 'Number of time bins',
106: & 'NSOUR', NSOUR, 'Number of sources',
107: & 'NRESP', NRESP, 'Number of responses',
108: & 'NSTALY', NSTALY, 'Number of special tallies'
109: C
110: C -----
111: C
112:   write(IOUT,7000) 'MATERIAL & CROSS SECTION'
113: C -----
114: C
115:   write(IOUT,7020)
116: & 'NMAT', NMAT, 'Number of materials',
117: & 'NTGX', NTGX, 'Number of energy groups (of cross section)',
118: & 'NSCT', NSCT, 'Number of scattering angle number (DDX)'
119: C
120: C -----
121: C
122:   write(IOUT,7000) 'CUTOFF & THRESHOLD'
123: C -----
124: C
125:   write(IOUT,7080) 'TCUT', TCUT, 'Cutoff particle time(age) (sec.)'
126:   return
127: end

```

src/gmvp/prsour.f

```

1: *VOCL TOTAL,SCALAR
2: subroutine PRSOUR( NSOUR, NGROUP,NMAT, JTIME, JEIGN, KINPZ,
3:   &                KSOUR, ISZON, SOUR, PSPAC, PENRG, KENRG,
4:   &                NSTIM, STIM, PSTIM, ISTIM, NSANG, SANG,
5:   &                SAXIS, PSANG, ISANG, IFISM, FKAI, KKAI, TEMP
6:   &                )
7: C=====
8: C PURPOSE: PRINTOUT & PREPROCESSING OF SOURCE DATA.
9: C CALLED IN: INTRO
10: C=====
11: integer KSOUR(NSOUR), ISZON(2,NSOUR), KENRG(2,NGROUP,NSOUR)
12: integer NSTIM(NSOUR), ISTIM(1), NSANG(NSOUR), ISANG(NSOUR)
13: integer KINPZ(1), IFISM(NSOUR), KKAI(2,NGROUP,NMAT)
14: real*8 SOUR(NSOUR)
15: real PSPAC(10,NSOUR), PENRG(NGROUP,NSOUR)
16: real STIM(1), PSTIM(1), SANG(1), SAXIS(3,NSOUR), PSANG(1)
17: real TEMP(1), FKAI(NGROUP,NMAT)
18: include '../shared/INC/_IOUNIT'
19: C
20: C =====
21: call HEADER( IPR, 'SOURCE DATA' )
22: C =====
23: C
24: write(IPR,'(///' * NUMBER OF SOURCES =',I5,') NSOUR
25: C
26: C *****
27: call LABEL( IPR, 'SOURCE FRACTION (UNNORMALIZED)' )
28: C *****
29: C
30: call R8PRNT( 'SOUR ',SOUR,NSOUR,1,'SOURCE #', ' ',IPR )
31: C
32: C *****
33: call LABEL( IPR, 'SOURCE FRACTION (NORMALIZED)' )
34: C *****
35: C
36: call PNORM8( SOUR, NSOUR )
37: call R8PRNT( 'SOUR ', SOUR, NSOUR,1, 'SOURCE #', ' ',IPR )
38: C
39: if ( JEIGN.ne.0 ) then
40: C
41: C *****
42: call LABEL( IPR, 'MAT # FOR FISSION SPECTRUM' )
43: C *****
44: C
45: call I4PRNT( 'IFISM ', IFISM, 1, NSOUR, 1, 1, 'SOURCE #',
46: & ' ',IPR )
47: end if
48: C
49: C
50: C *****
51: call LABEL( IPR, 'DATA FOR EACH SOURCE' )
52: C *****
53: C
54: do 100 N = 1, NSOUR
55: write(IPR,'(/lx,' >>>> SOURCE # ',I3,' KSOUR=',I3,
56: & ' ' FRACTION = ',E12.5)') N, KSOUR(N), SOUR(N)
57: C
58: C
59: if ( ISZON(1,N).ne.0 ) write(IPR,7000) ISZON(2,N),
60: & KINPZ(ISZON(1,N)), ISZON(1,N)
61: 7000 format(/lx,' STARTING ZONE : ',I5,'TH ZONE OF ',I5,
62: & ' 'TH INPUT-ZONE ----> ZONE ',I5)
63: write(IPR,'(/lx,' SPATIAL DATA ' '))
64: call R4PRNT( 'PSPAC ', PSPAC(1,N), 10, 1, ' DATA #',
65: & ' ',IPR )

```

```

66: if ( KSOUR(N).eq.1 .or. KSOUR(N).eq.3 .or. KSOUR(N).eq.4 ) then
67: if ( PSPAC(9,N).eq.0..and.PSPAC(10,N).eq.0. ) then
68: PSPAC(9,N) = -1.
69: PSPAC(10,N) = 1.
70: end if
71: end if
72: C
73: C
74: if ( JEIGN.eq.0 ) then
75: if ( KSOUR(N).ne.9 ) then
76: write(IPR,7020) (PENRG(K,N),K=1,NGROUP)
77: 7020 format(' ' ENERGY DISTRIBUTION (UNNORMALIZED)')//
78: & (1X,5X,10E12.5))
79: call PNORM( PENRG(1,N), NGROUP )
80: write(IPR,7040) (PENRG(K,N),K=1,NGROUP)
81: 7040 format(' ' ENERGY DISTRIBUTION (NORMALIZED)')//
82: & (1X,5X,10E12.5))
83: call BMCML( PENRG(1,N), TEMP, KENRG(1,1,N), NGROUP )
84: call ATRANS( TEMP, PENRG(1,N), NGROUP )
85: end if
86: else
87: write(IPR,7060) IFISM(N)
88: 7060 format(/lx,' ENERGY DISTRIBUTION = FISSION SPECTRUM',
89: & ' OF ',I5,'TH MATERIAL')//
90: C88/9/27 CALL ATRANS(FKAI(1,IFISM(N)),PENRG(1,N),NGROUP)
91: CCCCC CALL ATRANS(KKAI(1,1,IFISM(N)),KENRG(1,1,N),NGROUP*2)
92: end if
93: 100 continue
94: return
95: end

```

src/gmvp/q.f

```
1: *VOCL TOTAL,SCALAR
2:   function Q(ND,X)
3: C=====
4: C  PURPOSE:  TO GENERATE Q,THE VALUE AT X OF THE ORTHOGONAL POLYNOMIAL
5: C           Q OF ORDER ND
6: C  CALLED IN: ANGLES,BADMOM,FIND          CALLS: (NONE)
7: C=====
8:   common /MEANS/  N,      ITEST,  XMU(21),      VAR(20),
9:   &              XORM(20)
10: c
11: ccccc IF(ND-1)25,20,10
12: c
13:   if ( ND.le.0 ) then
14:     Q      = 1.0
15:     return
16:   end if
17: c
18:   A      = 1.0
19:   B      = X - XMU(1)
20:   if ( ND.ge.1 ) then
21:     do 100 I = 2, ND
22:       C      = (X-XMU(I))*B - VAR(I-1)*A
23:       A      = B
24:       B      = C
25:     100 continue
26:   end if
27: c
28:   Q      = B
29:   return
30: end
```

src/gmvp/readsg.f

```

1:      subroutine READSG( LEM,   KOF,   SIGT, NSIG, JSEP )
2:
3: C=====
4: C PURPOSE:  TO READ MULTIGROUP CROSS SECTIONS AND STORE THEM IN
5: C           A BUFFER REGION
6: C CALLED IN:  JNPUT          CALLS: FFREAD,XSCHLP
7: C=====
8:      include '../shared/INC/_IOUNIT'
9: C
10:     common /LOCSIG/  ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
11: & IFSPORG, IDSGOG, IPRBNG, IPRBGG, ISGANG, ISGAGG, ISPORG,
12: & ISPORT, INPBUF, ISIGOG, INEFOG, IABSOG, ITOTSG, NGP,
13: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
14: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
15: & IADJ, NME, LOC, INGS, INSG, I1, I0,
16: & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRDSG,
17: & ISTR, IPRIN, IPMU, IMOM, IDTF, ISTAT, IPUN,
18: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
19: & INNN, IGGG,
20: C
21:     common /DDX/  IDDX,  MAXU,  MAXSD,  IMAX,  NLASTD, MAXDDX
22: C
23:     real SIGT(1), V(37)
24:     integer NSIG(1), IN(37)
25:
26: C  EQUIVALENCE (SIGT(1),NSIG(1))
27:
28:     character*8 HOL(6)
29:     character*4 T(37)
30:     data IPRTRG /-1/
31: C
32: C-----
33: C
34:     IST      = 0
35: Csasa II    = INPBUF
36: M           = INGP*(INDS+IHT+NUS)
37: Csasa N     = ISPORG
38: K           = 1
39: KS          = 1
40: I3          = INPBUF + 1
41:
42: C
43: C ... Is this memory clear really necessary ??? (M.Sasaki)
44: C
45: C   do 100 I = INPBUF+1, ISPORT
46: C     SIGT(I) = 0.0
47: C 100 continue
48: C
49: C ..... INPUT CROSS SECTIONS FROM TEXT DATA (IXTAPE=0, NON-DDX).....
50: C
51: C   if ( IXTAPE.le.0 ) then
52: C
53: C     if ( IDTF.le.0 ) then
54: C       IDUM      = 0
55: C * *
56: C THE CARD SEQUENCE TEST ASSUMES THAT CARDS FOLLOW ONE OF THESE FORMAT
57: C CODE          COLS. 73-76          COLS. 77-80
58: C GAM OR MUG    GROUP NUMBER        GROUP SEQUENCE
59: C SUPERTOG OR   BLANK                GROUP SEQUENCE
60: C XSDRN
61: C ANISN         BLANK                SET SEQUENCE
62: C
63: C THE ENTIRE MESS MAY BE BYPASSED (ALONG WITH PRINT OF XSECTS AS R
64: C SETTING IRDSG TO A NEGATIVE VALUE
65: C ALPHABETIC CHARACTERS MAY BE PRESENT BUT MUST FOLLOW THE GROUP/S

```

```

66: C NO. IN COLS. 73-76/77-80 FOR SEQUENCE TEST TO REMAIN (I.E. LEADING
67: C ALPHABETIC CHARACTERS WILL RESULT IN TRAILING NUMBERS IN EACH 4 CO
68: C BEING IGNORED).
69: C * *
70: C
71: C ..... READ DATA X-6 (FREE-FORM INPUT OF ANISN TYPE CROSS SECTION) ..
72: C
73: 100      call FFREAD( IN, T, V, NF, I1, I0, IPRTRG )
74: C
75: IDUM      = IDUM + 1
76: NDUM      = IN(37)
77: C
78: if ( IRDSG.ge.0.and.NDUM.ne.IDUM.and.NDUM.ne.1.and.NDUM.ne.0
79: & ) then
80:
81: 110      write(IMG,7000) (IN(I),T(I),V(I),I=1,NF), NDUM
82: 7000      format('!!! X-SEC DATA OUT OF ORDER : ',7(I3,A1,1PE9.2))
83:
84: write(IMG,7020) NDUM, IDUM, LEM, KOF
85: 7020      format(' DATA',I5,' WAS READ WHEN DATA ',I5,' WAS ',
86: & 'EXPECTED. CHECK CROSS SECTION DATA SEQUENCE ',
87: & 'IN ELEMENT ',I5,' COEFFICIENT ',I5)
88:
89: call CNTERR( 'WARNING' )
90: IDUM      = NDUM
91: end if
92: check %%%
93: C write(*,*) '%% NF ',NF,
94: C & ' in ',(in(i),i=1,nf),' T ',(t(i),i=1,nf),' V ',(v(i),i=1,nf)
95: C %%%%%%%%%
96: C
97: do 130 I = 1, NF
98: L      = IN(I)
99: NP      = 1
100: if ( T(I).eq.'R ' ) NP      = L
101:
102: if ( T(I).eq.'Z ' ) then
103:
104: C THIS IS NECESSARY FOR THE FIXED FORM INPUT
105:
106: IN(I)   = V(I) + IN(I)
107: V(I)    = 0.0
108: NP      = L
109: end if
110:
111: do 120 J = 1, NP
112: SIGT(INPBUF+K) = V(I)
113: if ( K.gt.M ) go to 140
114:
115: if ( K.eq.M ) then
116: if ( J.ne.NP ) go to 140
117: if ( I.eq.NF ) return
118: go to 140
119: end if
120:
121: K      = K + 1
122: 120      continue
123:
124: 130      continue
125: C
126: go to 100
127: C
128: 140      write(IMG,7040) I
129: write(IMG,7060) (IN(I),T(I),V(I),I=1,NF)
130: 7040      format('/ XXX ONLY',I5,' CROSS SECTIONS WERE USED FROM ',

```


src/gmvp/readsg.f

```

131:      &          'THE FOLLOWING DATA: '//)
132: 7060      format(6(I2,A1,1PE9.2))
133:
134:          call PRSTOP( 1, 'CROSS SECTION ERROR (WHEN IDTF < 0)' )
135:          stop 888
136:      end if
137: C
138: C ..... READ DATA X-6 (DTF-IV FORMAT) .....
139: C
140: 150      K      = 1
141:          read(I1,7080) (HOL(I),I=1,6)
142: 7080      format(6A8)
143: C
144: 160      continue
145:          read(I1,7100) (V(I),I=1,6)
146: 7100      format(6E12.5)
147:
148:          do 190 I = 1, 6
149:              SIGT(INPBUF+K) = V(I)
150:              if ( K.gt.M ) go to 200
151:
152:              if ( K.eq.M ) then
153: 170                  if ( I.eq.6 ) return
154:                      LIM      = I + 1
155:
156:                  do 180 JJ = LIM, 6
157:                      if ( HOL(JJ).ne.' ' ) go to 200
158: 180                  continue
159:
160:                  return
161:              end if
162:
163:              K      = K + 1
164: 190      continue
165: C
166: C
167:          go to 160
168: C
169: C
170: 200      write(I0,7120)
171: 7120      format(//)
172:          write(IMG,7140) (V(I),I=1,6)
173: 7140      format(1X,6E12.5)
174:          call PRSTOP( 1, 'CROSS SECTION ERROR (WHEN IDTF > 0)' )
175:          stop 888
176:      end if
177: C
178: C
179: C ..... INPUT CROSS SECTIONS FROM FILE (IXTAPE .GT. 0) .....
180: C
181: C
182: C 220 read(IXTAPE,end =260) N1, N2, N3, N4, (HOL(I),I=1,6)
183: C
184: C      M2      = IDEL + KKK
185: C
186: C*** 1999/11/4
187: C
188: C 210 M2      = IDEL + KKK
189:          read(IXTAPE,end =250) N1, N2, N3, N4, (HOL(I),I=1,6)
190: C
191: C
192: C      ... skip this element when ID does not match.
193: C
194:          if ( (IDDX.ne.0.and.N3.ne.NSIG(M2))
195:      &      .or. (IDDX.eq.0.and.N3.ne.NSIG(M2).and.N4.ne.NSIG(M2)) ) then

```

```

196:
197:          if ( JSEP.eq.0 ) then
198:              read(IXTAPE,end =250)
199:          else
200:
201:              do 220 J = 1, INGP
202:                  read(IXTAPE,end =250)
203: 220          continue
204:
205:          end if
206:
207:          go to 210
208:      end if
209: C
210: 230 if ( JSEP.eq.0 ) then
211:     read(IXTAPE) (SIGT(INPBUF+L),L=1,M)
212: else
213:     KKKL      = INDS + IHT + NUS
214:     MMMM      = INPBUF
215:
216:     do 240 J = 1, INGP
217:         read(IXTAPE) (SIGT(MMMM+L),L=1,KKKL)
218:         MMMM      = MMMM + KKKL
219: 240     continue
220:
221:     end if
222: C
223:     FK      = KKK
224:     FK      = FK/NCOEF + .98
225:     KOK      = FK
226:     write(I0,7160) KOK, N4, (HOL(I),I=1,6)
227: 7160 format(/4X,'*** ELEMENT',I5,' ID=',I10,' <',6A8,'>')
228:     return
229: C
230: 250 if ( IST.le.0 ) then
231:     IST      = 1
232:     call RWIND( IXTAPE )
233:     go to 210
234: end if
235:
236: 260 I92      = NSIG(M2)
237:     write(IMG,7180) I92
238: 7180 format(/' XXX ELEMENT',I5,' CANNOT BE FOUND')
239:     call PRSTOP( 1, 'CROSS SECTION ERROR (INPUT FROM FILE)' )
240:     stop 888
241: end

```

src/gmvp/rearag.f

```

1:      subroutine REARAG( IG1,  MAXU,  MAXS,  IMAX,  NGP1,  NGG1,  II,
2:      &                  NLASTD,S1,   S2,   S3,   S4,   S5,   S6,
3:      &                  S7,   S8,   SIGT,  NSIG,  LW )
4: C=====
5: C=====
6: C
7:      common /LOCSIG/  ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
8:      &              IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
9:      &              ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
10:     &              NDS,   NGG,   NDSG,  INGP,  INDS,  NMED,  NELEM,
11:     &              NMIX,  NCOEF, NSCT,  NTS,   NTG,   NDSNGP, NDSNGG,
12:     &              IADJ,  NME,   LQG,   INGS,  INSG,  I1,   I0,
13:     &              KKK,   IXTAPE, IDEL,  ITEMPL, ITEMG,  IRSG,  IRDSG,
14:     &              ISTR,  IPRIN,  IFMU,  IMOM,  IDTF,  ISTAT,  IPUN,
15:     &              NUS,   NGN,   IHT,   INUS,  INUSN,  INGN,  INGNP,
16:     &              KNNN,  IGGS
17: C
18:      real S1(1), S2(1), S3(1), S4(1), S5(1), S6(MAXU,1),
19:      &      S7(MAXU,IMAX,1), S8(IMAX,1), SIGT(*), NSIG(*)
20:      integer LW(1)
21:      include '../shared/INC/_IOUNIT'
22: C
23: C-----
24: C
25:      do 110 IGDD = 1, MAXS
26:         IGD = IG1 + IGDD - 1 - NUS
27:         if ( IGD.gt.IMAX .or. IGD.lt.1 ) go to 110
28:         do 100 MU = 1, MAXU
29:            S7(MU,IGD,IG1) = S6(MU,IGDD)
30:         100 continue
31:      110 continue
32: C
33:      if ( IG1.lt.IMAX ) return
34: C
35: C ... for the last energy group ...
36: C
37:      120 ITBL = INDS + IHT + NUS
38:      ITBL3 = ITBL - 3
39:      MAXTBL = INGP*ITBL
40:      if ( MAXTBL+II.gt.NLASTD ) then
41:         write(IMG,7000) MAXTBL, NLASTD
42:      7000 format(1X,'XXX SUB.REARAG XXX'/10X,'NOT ENOUGH MEMORY ',I10,
43:      &          '( ',I10,')')
44:         call PRSTOP( 1, 'MEMORY SHORTAGE IN CORSS SECTION PROCESSING.'
45:      &                )
46:         stop 999
47:      end if
48: C
49:      do 130 I = 1, MAXTBL
50:         SIGT(II+I) = 0.0
51:      130 continue
52: C
53:      do 150 IG = 1, IMAX
54:         do 140 IGD = 1, IMAX
55:            S8(IGD,IG) = 0.0
56:         140 continue
57:      150 continue
58: C
59:      do 180 IG = 1, IMAX
60:         IGDMAX = LW(IG)
61:         do 170 IGDD = 1, IGDMAX
62:            IGD = IG + IGDD - 1 - NUS
63:            if ( IGD.gt.IMAX .or. IGD.lt.1 ) go to 180
64:
65:            do 160 M = 1, MAXU

```

```

66:            S8(IGD,IG) = S8(IGD,IG) + S7(M,IGD,IG)
67:      160 continue
68:
69:            S8(IGD,IG) = S8(IGD,IG)*S1(IG)
70:
71:      170 continue
72:      180 continue
73: C
74:         K1 = II + 1
75:         do 200 IG = 1, IMAX
76:            K2 = K1 + 1
77:            K3 = K2 + 1
78:            SIGT(K1) = S2(IG) + S3(IG)
79:            SIGT(K2) = S4(IG)
80:            SIGT(K3) = S5(IG)
81:            K = K3
82:            MG = IG + NUS
83:            if ( MG.gt.ITBL3 ) MG = ITBL3
84:            do 190 IGD = 1, MG
85:               J = IG - IGD + 1 + NUS
86:               K = K + 1
87:               if ( J.le.IMAX ) then
88:                  SIGT(K) = S8(IG,J)
89:               else
90:                  SIGT(K) = 0.0
91:               end if
92:            190 continue
93:            K1 = K1 + ITBL
94:         200 continue
95: C
96: C .... Output by integer form because of a bug in fortran library
97: C      in DEC/OSF pre 1.2 version.
98: C
99: C      WRITE(IXTAPE) (SIGT(II+I),I=1,MAXTBL)
100:      write(IXTAPE) (NSIG(II+I),I=1,MAXTBL)
101:
102:      do 210 I = 1, MAXTBL
103:         SIGT(II+I) = 0.0
104:      210 continue
105: C
106:      write(IPR,7020)
107: C
108:      write(IPR,7040) NGP1, NGG1, MAXU
109:      if ( IRDSG.gt.0 ) then
110:         write(IPR,7060)
111:         do 220 IG = 1, IMAX
112:            write(IPR,7080) IG, S1(IG), S2(IG), S3(IG), S4(IG), S5(IG)
113:         220 continue
114:      end if
115:
116:      backspace IXTAPE
117:      backspace IXTAPE
118:
119:      return
120: C
121:      7020 format(/1X,'DDX LIBRARY INFORMATION')
122:      7040 format(5X,'PRIMARY GROUPS = ',I5,5X,'SECONDARY GROUPS = ',I5,5X,
123:      &          'ANGLES = ',I5)
124:      7060 format(/12X,'GROUP NO',13X,'SIGMA-P',13X,'SIGMA-F',13X,'SIGMA-C',
125:      &          10X,'NU-SIGMA-F',10X,'SIGMA-T')
126:      7080 format(10X,I10,1P5E20.3)
127:      end

```

src/gmvp/resti0.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTI0( NTHIST,WSUM,  WLEK,  IRAND, NBATCH,NFISB,
3:   &                 NGROUP,NRESP, NREG,  NMEMO, NZONE,
4:   &                 NPART, NLATT, NEST,  NPDET, NETALY, NETRV )
5: C=<GMVP>=====
6: C  PURPOSE: input GMVP restart information header (records existing only
7: C           as single record even in multitask mode.) and check validity.
8: C  CALLED IN:  INTRO2
9: C  CALLS: (NONE)
10: C
11: C    I/O UNIT OF RESTART INPUT :  FT10F001  (IRIN)
12: C    I/O UNIT OF RESTART OUTPUT:  FT20F001  (IROT)
13: C
14: C=====
15:   include 'INC/_PROGV'
16:   include 'INC/_FLAGS'
17:   include '../shared/INC/_IUNIT'
18:   include 'INC/_IUNIT2'
19: C/#IF PARA(PVM)
20:   include '../shared/INC/_PVMPARA'
21: C/#ELSEIF PARA(MPI)
22: *   include 'mpif.h'
23: C/#ENDIF
24:   include '../shared/INC/_TASKDE'
25: C
26:   real*8 WSUM, WLEK
27: C/#IF INTEGER8
28: *   integer*8 NPART, NTHIST
29: C/#ELSE
30:   integer NPART, NTHIST
31: C/#ENDIF
32: C
33: C ... local variables ...
34: C
35:   character*80 HD
36:   character*72 TP(2), VERSN(40)*60
37:   integer IWRK(16)
38: C/#IF INTEGER8
39: *   integer*8 I8WRK(4)
40: C/#ELSE
41:   integer I8WRK(4)
42: C/#ENDIF
43: C
44: C-----
45: C
46:   call HEADER( IPR, 'RESTART FILE INPUT' )
47: C
48: C ... in message psssing multitask mode, only main task input header
49: C
50: C/#IF PARA(PVM MPI)
51:   if( IDTASK.eq.1 ) then
52: C/#ENDIF
53:   IPERR = 0
54: C
55: C-----
56: C ..... READ FILE HEADER (80 CHARACTERS) .....
57: C-----
58:   read(IRIN,END=777) HD
59:   write(IPR,7001) HD
60:   7001 format(3X,'=== HEADER OF RESTART FILE ==='//5X,A/)
61:   goto 888
62: C
63:   777 continue
64:   if ( JARST.eq.0 ) then
65:     write(IMG,*) 'XXX Restart file is empty !!'

```

```

66:     call CNTERR( 'FATAL' )
67:     IPERR = IPERR + 1
68:   else
69:     write(IPR,*)
70:   &   '<<MESSAGE>> (Auto-restart) Restart file is empty,',
71:   &   'so calculation is set to non-restart mode.'
72:     call CNTERR( 'MESSAGE' )
73:     JREST = 0
74:   end if
75:   return
76: C
77:   888 continue
78:   JREST = 1
79: C-----
80: C ..... READ PROBLEM TITLE IN RESTART FILE .....
81: C-----
82:   read(IRIN) (TP(I),I=1,2)
83:   write(IPR,7000) (TP(I),I=1,2)
84:   7000 format(/3X,'=== PROBLEM TITLE IN RESTART FILE ==='//5X,A/5X,A/)
85: C
86: C-----
87: C ..... READ PROGRAM VERSION DESCRIPTION. ....
88: C-----
89:   read(IRIN) (VERSN(I),I=1,40)
90:   write(IPR,'(3X,'''=== TYPE OF RESTART FILE : ''',A/))' ) VERSN(40)
91: C
92: C/#IF PARA(PVM MPI)
93: C
94: C ... end if of "if( IDTASK.eq.1 ) ..."
95: C
96:   end if
97: C
98: Check %%%%%%%%%%
99: C   do i=1,100
100: C     write(ipr,*) ' --- resti0 1 ', i, 'idtask ',idtask,' itask0 ',
101: C     &           itask0
102: C   end do
103: C %%%%%%%%%%
104: C
105:   if( NTASK.gt.1 ) then
106:     MSGNUM = 12345
107:     it0 = 1
108:     call MVPCOM_BCAST_CH(IT0,MSGNUM, HD, LEN(HD), INFO)
109:     call MVPCOM_BCAST_CH(IT0,MSGNUM, TP(1), LEN(TP(1)), INFO)
110:     call MVPCOM_BCAST_CH(IT0,MSGNUM, TP(2), LEN(TP(2)), INFO)
111:     call MVPCOM_BCAST_CH(IT0,MSGNUM,VERSN(40),LEN(VERSN(40)),INFO)
112:   endif
113: C/#ENDIF
114: C
115: C-----
116: C ..... RESTART FILE DOES NOT MATCH THIS VERSION !! .....
117: C-----
118:   if ( VERSN(40).ne.PROGV(40) ) then
119:     write(IMG,*) 'XXX I cannot read restart file of this type. '
120:     call CNTERR( 'FATAL' )
121:     return
122:   end if
123: C-----
124: C ... number of histories etc.( sum of all tasks ) ....
125: C
126: C   NRBUF0: read in buffer size of chopped restrt file records.
127: C-----
128: C/#IF PARA(PVM MPI)
129:   if( IDTASK.eq.1 ) then
130: C/#ENDIF

```

src/gmvp/resti0.f

```

131:      read(IRIN) NTASK2, NTHIST, NBATCH, NRBUFF
132: C
133: C/IF PARA(PVM MPI)
134: C
135: C      ... end if of "if( IDTASK.eq.1 ) ..."
136: C
137:      end if
138: C
139:      if( NTASK.gt.1 ) then
140:          MSGNUM = 345
141:          it0 = 1
142:          if( IDTASK.eq.IT0 ) then
143:              IWRK(1) = NTASK2
144: C/IF INTEGER8
145: *          I8WRK(2) = NTHIST
146: C/ELSE
147:          IWRK(2) = NTHIST
148: C/ENDIF
149:          IWRK(3) = NBATCH
150:          IWRK(4) = NRBUFF
151:      end if
152:
153:      call MVPCOM_BCAST_I4( IT0, MSGNUM, IWRK, 4, INFO )
154: C/IF INTEGER8
155: *      call MVPCOM_BCAST_I8( IT0, MSGNUM+1, I8WRK, 4, INFO )
156: C/ENDIF
157:
158:      if( IDTASK.ne.it0 ) then
159:          NTASK2 = IWRK(1)
160: C/IF INTEGER8
161: *          NTHIST = I8WRK(2)
162: C/ELSE
163:          NTHIST = IWRK(2)
164: C/ENDIF
165:          NBATCH = IWRK(3)
166:          NRBUFF = IWRK(4)
167:      end if
168:      endif
169: C
170: C/ENDIF
171: C
172:      write(IPR,7044) NTASK2, NTHIST, NBATCH, NRBUFF
173: 7044 format(3X,'=== STATUS OF RESTART FILE (TASK COMMON) ==='/1X,
174: &      '      NUMBER OF TASKS      (NTASK) = ',I10/1X,
175: &      '      NUMBER OF HISTORIES (NTHIST) = ',I10/1X,
176: &      '      NUMBER OF BATCHES  (NBATCH) = ',I10/1X,
177: &      '      INPUT BUFFER SIZE  (NRBUF) = ',I10/1X)
178: C
179:      if( NTASK2.ne.NTASK ) then
180:          write(IMG,7132) NTASK2, NTASK
181: 7132 format(/1X,'XXX(RESTI0) Restart file contains information of ',
182: &      'I4,' tasks and it does not match the current one ',I4,
183: &      ' ')
184:          call CNTERR( 'FATAL' )
185:      end if
186: C
187: C      ... MRBUF is defined in include file ../shared/_TASKDT currently
188: C
189:      if( NRBUFF.gt.MRBUF ) then
190:          write(IMG,7134) NRBUFF, MRBUF
191: 7134 format(/1X,'XXX(RESTI0) Input buffer size of restart file ',I6,
192: &      ' is greater than that of the program (MRBUF=',I6,')'/
193: &      'Please modify MRBUF and recompile the program.')
194:          call CNTERR( 'FATAL' )
195:      end if

```

```

196: C
197: C-----
198: C      ... VARIABLE PARAMETERS ....
199: C-----
200: C      read(IRIN) NTHIST, WSUM, IRAND, NBATCH, NFISB, WLEK
201: C
202: C      write(IPR,'(3X,'=== STATUS OF RESTART FILE ===')')
203: C      write(IPR,7020) '      HISTORY NUMBER (NTHIST) = ', NTHIST
204: C      write(IPR,7040) '      SUM OF WEIGHT  (WSUM)   = ', WSUM
205: C      write(IPR,7020) '      LAST RANDOM NUMBER (IRAND) = ', IRAND
206: C      write(IPR,7020) '      BATCH NUMBER  (NBATCH) = ', NBATCH
207: C      write(IPR,7020) '      FISSION SOURCE (NFISB)  = ', NFISB
208: C7020 format(1X,A,I15)
209: C7040 format(1X,A,1P,D15.7)
210: C
211: C
212:      if ( NPART.gt.0.and.NPART.le.NTHIST ) then
213:          write(IMG,7060) NPART, NTHIST
214: 7060      format(1X,'!!! YOU INTEND TO RUN TOTAL OF ',I9,' HISTORIES',
215: &      ' , BUT THE RESTART FILE HAS RESULTS OF ',I9,
216: &      ' HISTORIES.'/
217: &      '      SO MVP RUNS NO HISTORY AND ONLY ANALYZES TALLIES.'/
218: &      '      1X,' TO RUN MORE N HISTORIES INPUT AS NPART( -N ). '/')
219:      else if ( NPART.lt.0 ) then
220:          write(IPR,'(5X,'**** HISTORIES IN CURRENT RUN: FROM ',I10,
221: &      ' TO ',I10)') NTHIST + 1, NTHIST + ABS(NPART)
222:      else if ( NPART.gt.0 ) then
223:          write(IPR,'(5X,'**** HISTORIES TO BE RUN: FROM ',I10,
224: &      ' TO ',I10,' ('',I10,' HISTORIES)')'
225: &      ) NTHIST + 1, NPART, NPART - NTHIST
226:      end if
227: C-----
228: C      ..... READ NON-VARIABLE PARAMETERS .....
229: C-----
230: CC      IPERR = 0
231: C/IF PARA(PVM MPI)
232:      if( IDTASK.eq.1 ) then
233: C/ENDIF
234: C
235: C
236: C      ... until file version 3.0
237: C
238: C      read(IRIN) KGROUP, KNREG, KNTREG, KNRESP, KNMEMO, KNZONE, KFBANK,
239: C      &      KNLATT, KNEST, KNPDET, KNETAL,
240: C      &      KJEIGN, KJRESP, KJHLAT, KJTLT
241: C
242: C      ... from file version 3.1
243: C
244: C      read(IRIN) KGROUP, KNREG, KNTREG, KNRESP, KNMEMO, KNZONE, KFBANK,
245: C      &      KNLATT, KNEST, KNPDET, KNETAL, KNETRV
246: C      read(IRIN)
247: C      &      KJEIGN, KJRESP, KJHLAT, KJTLT
248: C
249: C/IF PARA(PVM MPI)
250: C
251: C      ... end if of "if( IDTASK.eq.1 ) ..."
252: C
253:      end if
254: C
255:      if( NTASK.gt.1 ) then
256:          MSGNUM = 2345
257:          it0 = 1
258:          if( IDTASK.eq.it0 ) then
259:              IWRK(1) = KGROUP
260:              IWRK(2) = KNREG

```

src/gmvp/resti0.f

```

261:         IWRK(3) = KNTREG
262:         IWRK(4) = KNRESP
263:         IWRK(5) = KNMEMO
264:         IWRK(6) = KNZONE
265:         IWRK(7) = KFBANK
266:         IWRK(8) = KNLATT
267:         IWRK(9) = KNEST
268:         IWRK(10) = KNPDET
269:         IWRK(11) = KNETAL
270:         IWRK(12) = KNETRV
271:         IWRK(13) = KJEIGN
272:         IWRK(14) = KJRESP
273:         IWRK(15) = KJHLAT
274:         IWRK(16) = KJTLLT
275:     endif
276:
277:     call MVPCOM BCAST_I4(IT0,MSGNUM, IWRK, 16, INFO )
278:
279:     if( IDTASK.ne.it0 ) then
280:         KGROUP = IWRK(1)
281:         KNREG = IWRK(2)
282:         KNTREG = IWRK(3)
283:         KNRESP = IWRK(4)
284:         KNMEMO = IWRK(5)
285:         KNZONE = IWRK(6)
286:         KFBANK = IWRK(7)
287:         KNLATT = IWRK(8)
288:         KNEST = IWRK(9)
289:         KNPDET = IWRK(10)
290:         KNETAL = IWRK(11)
291:         KNETRV = IWRK(12)
292:         KJEIGN = IWRK(13)
293:         KJRESP = IWRK(14)
294:         KJHLAT = IWRK(15)
295:         KJTLLT = IWRK(16)
296:     endif
297: endif
298: C/#ENDIF
299: C
300:     if ( KJEIGN.ne.JEIGN .or. KJRESP.ne.JRESP .or. KJHLAT.ne.JHLAT
301: & .or. KJTLLT.ne.JTLLT ) then
302:
303:         write(IMG,7080) KJEIGN, KJRESP, KJHLAT, KJTLLT
304: 7080     format(/1X,'XXX(RESTI0) Calculation options in restart file',
305: & ' do not match those from current input! STOP')
306: & ' JEIGN = ',I2,' JRESP = ',I2,' JHLAT = ',I2,
307: & ' JTLLT = ',I2)
308:
309:         call CNTERR( 'FATAL' )
310:         IPERR = IPERR + 1
311:     end if
312: C
313:     if ( KGROUP.ne.NGROUP .or. KNREG.ne.NREG .or. KNRESP.ne.NRESP
314: & .or. KNTREG.ne.NTREG .or. KNMEMO.ne.NMEMO .or. KNZONE.ne.NZONE
315: & .or. KNLATT.ne.NLATT .or. KNEST.ne.NEST .or. KNPDET.ne.NPDET
316: & .or. KNETAL.ne.NETALY ) then
317:
318:         write(IMG,7100) KGROUP, NGROUP, KNREG, NREG, KNRESP, NRESP,
319: & KNMEMO, NMEMO, KNZONE, NZONE, KNLATT, NLATT,
320: & KNEST, NEST, KNPDET, NPDET, KNTREG, NTREG,
321: & KNETAL, NETAL, KNETRV, NETRV
322: 7100     format(/1X,'XXX(RESTI0) One or more size parameters in',
323: & ' restart file do not match those from current input!')
324: & ' NGROUP = ',2I7,' NREG = ',2I7,' NRESP = ',2I7/
325: & ' NMEMO = ',2I7,' NZONE = ',2I7,' NLATT = ',2I7/
326: & ' NEST = ',2I7,' NPDET = ',2I7,' NTREG = ',2I7/
327: & ' NETALY = ',2I7,' NETRV = ',2I7/)
328:         call CNTERR( 'FATAL' )
329:         IPERR = IPERR + 1
330:     end if
331: C
332: C
333:     return
334: end

```

src/gmvp/restin.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine RESTIN( IOPR, H, ITSK,TITLE )
3: C=<GMVP>=====
4: C PURPOSE: Interface routine to RESTRT which inputs restart file record.
5: C CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C CALLS:
7: C
8: C-----
9: C
10: C   arguments ( i = input, o = output, c =constant, w = work )
11: C
12: C i IOPR : message printout I/O unit
13: C io H   : task local dynamic memory array
14: C i ITSK : data are stored on local data of task ITSK.
15: C        (ITSK may not be same as current IDTASK)
16: C i TITLE : problem titile
17: C-----
18:   real H(*)
19:   character TITLE(2)*72
20: C
21: CCCC include '../shared/INC/_ARRAY'
22:       include '../shared/INC/_TASKDT'
23:       include '../shared/INC/_LISTOFF'
24:       include 'INC/_FLAGS'
25:       include '../shared/INC/_SIZES'
26:       include 'INC/_SIZES2'
27:       include 'INC/_PXSEC'
28:       include 'INC/_XBANK'
29:       include 'INC/_STACK'
30:       include 'INC/_XWORK'
31:       include '../shared/INC/_PGEOM'
32:       include '../shared/INC/_PVRED'
33:       include 'INC/_PSOUR'
34:       include '../shared/INC/_PTALY0'
35:       include 'INC/_PTALY'
36:       include '../shared/INC/_STALY'
37:       include '../shared/INC/_PTLSP'
38:       include '../shared/INC/_LISTON'
39:       include '../shared/INC/_WORDL'
40: C
41: C-----
42: C
43:   call RESTRT( IOPR, ITSK, NTHIST, IRAND, H(LWSUM), H(LWLEK)
44: &             NGROUP, NRESP, NREG, NTREG, NBATCH, NMEMO,
45: &             NEVENT, NZONE, H(LKMEMO), H(LWCNTR), H(LNCNTR),
46: &             H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
47: &             H(LXAVT), H(LAAVT),H(LEAVT),
48: &             NFI8B, NFBANK,NFBK0, NLENG, H(LXXXF), H(LYYYF),
49: &             H(LZZZF), H(LIZZF), H(LIBRGF),
50: &             H(LIBSPF), H(LXSOC), H(LXSXV), H(LXKEF), NLATT, NEST,
51: &             H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
52: &             H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK, MIFBK,
53: &             NPDET, H(LSFLPD), H(LSREPD),
54: &             NETALY, H(LETALY), NLETAL,
55: &             NETRV, NSKIP, METRV, H(LETRV) )
56: C
57:   return
58: end
```

src/gmvp/resto0.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine RESTO0( IOPR, H, MONITR, TITLE, KTHIST, KBATCH )
3: C=<GMVP>=====
4: C   PURPOSE:  interface routine to restart file header output routine.
5: C   CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C   CALLS: RESTO1
7: C-----
8: C   arguments ( i = input, o = output, c =constant, w = work )
9: C
10: C i IOPR : message printout I/O unit
11: C io H : task local dynamic memory array
12: C i MONITR : do not print any message in RESTO1 if zero
13: C i TITLE : problem titile
14: C i KTHIST : total history number (sum for tasks)
15: C i KBATCH : total batch number (sum for tasks)
16: C=====
17:   real H(*)
18:   include '../shared/INC/_ARRAY'
19:   include '../shared/INC/_LISTOFF'
20:   include '../shared/INC/_TASKDT'
21:   include 'INC/_FLAGS'
22:   include '../shared/INC/_SIZES'
23:   include 'INC/_SIZES2'
24:   include '../shared/INC/_PGEOM'
25:   include 'INC/_XWORK'
26:   include '../shared/INC/_PTALY0'
27:   include 'INC/_PTALY'
28:   include '../shared/INC/_STALY'
29:   include '../shared/INC/_PTLSP'
30:   include 'INC/_XBANK'
31:   include 'INC/_STACK'
32: C
33:   include 'INC/_WKTLO'
34:   include '../shared/INC/_LISTON'
35: C
36:   character TITLE(2)*72
37:   include '../shared/INC/_WORDL'
38: C
39: C/#IF INTEGER8
40: *   integer*8 KTHIST
41: C/#ELSE
42:   integer KTHIST
43: C/#ENDIF
44: C
45: C .....IOT: PRINTOUT UNIT OF CADENZ STEP ...
46:   IOT = 6
47: C
48: C ..... OUTPUT TO RESTART FILE .....
49: C
50:   call RESTO1( IOPR, MONITR, TITLE,
51: & KTHIST,H(LWSUM), H(LWLEK), KBATCH,
52: & NGROUP,NRESP, NREG, NTREG, NMEMO,
53: & NZONE, NPART, NLATT, NEST,
54: & NPDET, NETALY, NETRV )
55: C
56:   return
57: end
```

src/gmvp/restol.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTOL( IOPR, MONITR, TITLE,
3:     &                NTHIST,WSUM, WLEK, NBATCH,
4:     &                NGROUP,NRESP, NREG, NTREG, NMEMO,
5:     &                NZONE, NPART, NLATT, NEST,
6:     &                NPDET, NETALY, NETRV )
7: C=====
8: C PURPOSE: output GMVP restart information header(records existing only
9: C           as single record even in multitask mode.)
10: C CALLED IN: RESTO0
11: C CALLS: (NONE)
12: C
13: C I/O UNIT OF RESTART INPUT : FT10F001 (IRIN)
14: C I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
15: C
16: C=====
17:   include 'INC/ PROGV'
18:   include 'INC/ FLAGS'
19:   include '../shared/INC/_IOUNIT'
20:   include 'INC/_IOUNIT2'
21: C/#IF PARA(PVM)
22:   include '../shared/INC/_PVMPARA'
23: C/#ELSEIF PARA(MPI)
24: *   include 'mpif.h'
25: C/#ENDIF
26:   include '../shared/INC/_TASKDT'
27: C
28:   character*72 TITLE(2)
29:   real*8 WSUM, WLEK
30: C/#IF INTEGER8
31: *   integer*8 NPART, NTHIST
32: C/#ELSE
33:   integer NPART, NTHIST
34: C/#ENDIF
35: C
36: C ... local variables ...
37: C
38:   character*80 HD
39:   character*12 DT, TM
40: C
41: C-----
42: C
43: ccc call HEADER( IPR, 'RESTART FILE OUTPUT' )
44: C
45: C ... in message psssing multitask mode, only main task input header
46: C
47: C/#IF PARA(PVM MPI)
48:   if( IDTASK.eq.1 ) then
49: C/#ENDIF
50: C
51: C
52: C-----
53: C ..... DATE & TIME OF OUTPUT .....
54: C-----
55: C
56:   DT = ' '
57:   TM = ' '
58:   call HIZUKE( DT )
59:   call JIKAN( TM(1:8) )
60: C
61:   HD = ' '
62: C
63:   HD = 'GMVP RESTART FILE V3.0 '//DT//' '//TM
64: C
65: C ... version 3.1 (from 2 Oct 1998)

```

```

66: C
67: C * size/flag record is separated into two record and NETRV is added.
68: C
69: C   HD = 'GMVP RESTART FILE V3.1 '//DT//' '//TM
70: C
71: C ... version 3.2 (from 1 Feb 1999)
72: C
73: C * output FDBNK and IDBNK if necessary.
74: C
75: C   HD = 'GMVP RESTART FILE V3.2 '//DT//' '//TM
76: C
77: C ... version 3.3 (from 21 Aug 2003)
78: C
79: C * output IRNSU, IRNSM, IRNSL and RNCTR. A seed value of a 63-bit RN
80: C   is split into 3 parts. RNCTR is the number of used RNs.
81: C
82:   HD = 'GMVP RESTART FILE V3.3 '//DT//' '//TM
83: C
84:   if( MONITR.ne.0 ) write(IOPR,7020) (TITLE(I),I=1,2)
85: 7020 format('//1X,' ===== OUTPUT TO RESTART FILE ===== '//1X,
86:   &        ' PROBLEM TITLE :',A72/1X,16X,A72/)
87:   if( MONITR.ne.0 ) write(IOPR,7000) HD
88: 7000 format(3X,'=== HEADER OF RESTART FILE === '//5X,A/)
89: C-----
90: C ..... write file header (80 CHARACTERS) .....
91: C-----
92:   write(IROT) HD
93: C-----
94: C ..... write problem title in restart file .....
95: C-----
96:   write(IROT) (TITLE(I),I=1,2)
97: C
98: C-----
99: C ..... write program version description. ....
100: C-----
101:   write(IROT) (PROGV(I),I=1,40)
102: C
103: C-----
104: C ... number of histories etc.( sum of all tasks ) ....
105: C
106: C MRBUF: read in buffer size of chopped restrt file records.
107: C-----
108:   write(IROT) NTASK, NTHIST, NBATCH, MRBUF
109: C-----
110: C ..... write non-variable parameters .....
111: C-----
112:   IPERR = 0
113: C
114: C .. until v3.0
115: C
116: C   write(IROT) NGROUP, NREG, NTREG, NRESP, NMEMO, NZONE,
117: C   & NFBANK, NLATT, NEST, NPDET, NETALY,
118: C   & JEIGN, JRESP, JHLAT, JTLLT
119: C
120: C .. after v3.1
121: C
122:   write(IROT) NGROUP, NREG, NTREG, NRESP, NMEMO, NZONE,
123:   & NFBANK, NLATT, NEST, NPDET, NETALY, NETRV
124:   write(IROT) JEIGN, JRESP, JHLAT, JTLLT
125: C
126: C/#IF PARA(PVM MPI)
127:   end if
128: C/#ENDIF
129:   return
130:   end

```


src/gmvp/resto.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTO( IOPR, MONITR,ITSK, NTHIST,IRAND, WSUM, WLEK,
3:   &                 NGROUP,NRESP, NREG, NTREG, NBATCH,NMEMO,
4:   &                 NEVENT,NZONE, KMEMO, WCNTR, NCNTR, SFLTR, SFLCL,
5:   &                 SRETR, SRECL, XAVT, AAVT, EAVT, NFISB,
6:   &                 NFBANK,NFBNK0,NLENG, XXXF, YYYY, ZZZF, IZZF,
7:   &                 IBRGF, IBSPF, XSOC, XSXV, XKEF, NLATT, NEST,
8:   &                 LEVLf, LZZF, LPOSF, LCRSF, DFBK, KDFBK, MDFBK,
9:   &                 IFBK, KIFBK, MIFBK, NPDET, SFLPD, SREPD,
10:  &                 NETALY,ETALY, NLETAL,NETRV, NSKIP, METRV, ETRV )
11: C=<MVP>=====
12: C  PURPOSE: Output restart information
13: C      (header - parameter must be input in RESTIO)
14: C
15: C  CALLED IN: RESTOT
16: C  CALLS: SRECO[I|R|D]
17: C
18: C  I/O UNIT OF RESTART INPUT: FT10F001 (IRIN)
19: C  I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
20: C
21: C=====
22:   include 'INC/_FLAGS'
23:   include '../shared/INC/_IOUNIT'
24:   include 'INC/_IOUNIT2'
25: C/#IF PARA(PVM)
26:   include '../shared/INC/_PVMPARA'
27: C/#ELSEIF PARA(MPI)
28: *   include 'mpif.h'
29: C/#ELSEIF PARA(VPP)
30:   include '../shared/INC/_VPPPARA'
31: C/#ENDIF
32:   include '../shared/INC/_TASKDT'
33:
34: C=<memo>= The seed value is saved as common variables
35: C      for a restart calculation. This treatment will be
36: C      modified in the future.
37:   include '../shared/INC/_RAND'
38: C
39:   real*8 WSUM, WLEK
40: C/#IF INTEGER8
41: *   integer*8 NTHIST
42: C/#ELSE
43:   integer NTHIST
44: C/#ENDIF
45: C
46: C      ..... EVENT MONITOR.....
47: C
48:   real*8 WCNTR(NEVENT), NCNTR(NEVENT)
49:   real XAVT(3), AAVT(3), EAVT
50: C
51: C      ..... NEXT-ZONE MEMORY .....
52: C
53:   integer KMEMO(NZONE,NMEMO)
54: C
55: C      ..... TALLY ARRAYS .....
56: C
57:   real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
58:   &       SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
59: C
60:   real*8 SFLPD(NGROUP,NPDET,2), SREPD(NPDET,NRESP,2)
61: C
62:   real*8 ETALY(NLETAL,2)
63: C
64:   real*8 ETRV(METRV,*)
65: C
66: C      ... EIGENVALUE TALLY .....
67: C
68:   real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(5,NLENG)
69: C
70: C      ... FISSION PARTICLE (SOURCE) BANK .....
71: C
72:   real*8 XXXF(NFBNK0), YYYY(NFBNK0), ZZZF(NFBNK0)
73:   integer LEVLf(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
74:   &       LCRSF(NFBNK0,NEST), IZZF(NFBNK0)
75:   integer IBRGF(NFBNK0), IBSPF(NFBNK0,0:NEST)
76: C
77:   real*8 DFBK(NFBNK0,*)
78:   integer KDFBK(0:MDFBK)
79:   integer IFBK(NFBNK0,*)
80:   integer KIFBK(0:MIFBK)
81: C
82: C      ..... local data .....
83: C
84:   character*32 TAG
85:   integer IWRK(32)
86:   real RWRK(32)
87:   real*8 DWRK(32)
88: C
89: C-----
90: C
91:   IERR = 0
92: C
93: C      ... Divide NTHIST into 2 parts for more than (2**31-1) histories.
94: C      This is the treatment not to change restart output format.
95: C      It should be modified in the future.
96: C
97:   I4MAX = 2147483647 ! 2**31-1
98:   IQNTH = int(NTHIST/I4MAX)
99:   IRNTH = NTHIST - IQNTH*I4MAX
100: C
101: C      ... idtask, NTHIST, IRAND, NBATCH, NFISB .....
102: C
103:   IWRK(1) = ITSK
104:   IWRK(2) = NTHIST
105:   IWRK(3) = IRAND
106:   IWRK(4) = NBATCH
107:   IWRK(5) = NFISB
108:   IWRK(6) = IRNSU
109:   IWRK(7) = IRNSM
110:   IWRK(8) = IRNSL
111:   IWRK(9) = IQNTH
112:   IWRK(10) = IRNTH
113:   NPKI = 10
114:   call SRECOI( IROT, ITSK, 'TASKHEADER', IWRK, NPKI, IERR )
115: C
116:   DWRK(1) = WSUM
117:   DWRK(2) = WLEK
118:   DWRK(3) = RNCTR
119:   NPKD = 3
120:   call SRECOI( IROT, ITSK, 'WSUM,WLEK', DWRK, NPKD, IERR )
121: C
122:   if ( MONITR.ne.0 ) write(IOPR,*) ' == Task ', IDTASK,
123:   & ' output task wise restart data header '
124: C
125:   7000 format(1X,A,I12)
126:   7020 format(1X,A,1P,E14.6)
127: C
128:   if ( MONITR.ne.0.and.IDTASK.eq.1.and.ITSK.eq.1 ) then
129:     write(IOPR,7000) ' IDTASK marked on record: ', ITSK
130:     write(IOPR,7000) ' NTHIST : ', NTHIST

```

src/gmvp/resto.f

```

131:      write(IOPR,7000) '      IRAND : ', IRAND
132:      write(IOPR,7000) '      NBATCH : ', NBATCH
133:      write(IOPR,7000) '      NFISB : ', NFISB
134:      write(IOPR,7020) '      WSUM : ', WSUM
135:      write(IOPR,7020) '      WLEK : ', WLEK
136:      write(IOPR,7000) '      IRNSU : ', IRNSU
137:      write(IOPR,7000) '      IRNSM : ', IRNSM
138:      write(IOPR,7000) '      IRNSL : ', IRNSL
139:      write(IOPR,7020) '      RNCTR : ', RNCTR
140:      write(IOPR,7000) '      IQNTH : ', IQNTH
141:      write(IOPR,7000) '      IRNTH : ', IRNTH
142:
143:      else
144:
145:      C/IF PARA(PWM MPI)
146:      MSGNUM = 111
147:      ITO = 1
148:      if ( IDTASK.ne.IT0 ) then
149:      call MVPCOM_SEND_I4( ITO, MSGNUM, IWRK, NPKI, INFO )
150:      call MVPCOM_SEND_R8( ITO, MSGNUM, DWRK, NPKD, INFO )
151:      else
152:      call MVPCOM_RECV_I4( ITSK, MSGNUM, IWRK, NPKI, INFO )
153:      call MVPCOM_RECV_R8( ITSK, MSGNUM, DWRK, NPKD, INFO )
154:      end if
155:      write(IOPR,7000) '      IDTASK marked on record: ', ITSK
156:      write(IOPR,7000) '      NTHIST : ', IWRK(2)
157:      write(IOPR,7000) '      IRAND : ', IWRK(3)
158:      write(IOPR,7000) '      NBATCH : ', IWRK(4)
159:      write(IOPR,7000) '      NFISB : ', IWRK(5)
160:      write(IOPR,7020) '      WSUM : ', DWRK(1)
161:      write(IOPR,7020) '      WLEK : ', DWRK(2)
162:      write(IOPR,7000) '      IRNSU : ', IWRK(6)
163:      write(IOPR,7000) '      IRNSM : ', IWRK(7)
164:      write(IOPR,7000) '      IRNSL : ', IWRK(8)
165:      write(IOPR,7020) '      RNCTR : ', DWRK(3)
166:      write(IOPR,7000) '      IQNTH : ', IWRK(9)
167:      write(IOPR,7000) '      IRNTH : ', IWRK(10)
168:      C/ENDIF
169:      end if
170:
171:      call SRECOI( IROT, ITSK, 'KMEMO', KMEMO, NZONE*NMEMO, IERR )
172:      call SRECOD( IROT, ITSK, 'WCNTR', WCNTR, NEVENT, IERR )
173:      call SRECOD( IROT, ITSK, 'NCNTR', NCNTR, NEVENT, IERR )
174:
175:      RWRK(1) = XAVT(1)
176:      RWRK(2) = XAVT(2)
177:      RWRK(3) = XAVT(3)
178:      RWRK(4) = AAVT(1)
179:      RWRK(5) = AAVT(2)
180:      RWRK(6) = AAVT(3)
181:      RWRK(7) = EAVT
182:      call SRECOR( IROT, ITSK, 'XAVT,AAVT,EAVT', RWRK, 7, IERR )
183:      C
184:      C ..... FLUXES .....
185:      C
186:      C      write(IROT) SFLTR
187:      C      write(IROT) SFLCL
188:      C
189:      call SRECOD( IROT, ITSK, 'SFLTR', SFLTR, NGROUP*NTREG*2, IERR )
190:      call SRECOD( IROT, ITSK, 'SFLCL', SFLCL, NGROUP*NTREG*2, IERR )
191:      C
192:      C ..... REACTION RATES .....
193:      C
194:      if ( JRESP.ne.0 ) then
195:      if ( NRESP.gt.0 ) then

```

```

196:      Cccc      write(IROT) SRETR
197:      Cccc      write(IROT) SRECL
198:      call SRECOD( IROT, ITSK, 'SRETR', SRETR, NTREG*NRESP*2, IERR
199:      &
200:      call SRECOD( IROT, ITSK, 'SRECL', SRECL, NTREG*NRESP*2, IERR
201:      &
202:      end if
203:      end if
204:      C
205:      C
206:      if ( JEIGN.ne.0 ) then
207:      C
208:      C .... EIGEN VALUE .....
209:      C
210:      Cccc      write(IROT) (XSOC(N),N=1,NBATCH)
211:      Cccc      write(IROT) (XSXV(N,1),N=1,NBATCH)
212:      Cccc      write(IROT) (XSXV(N,2),N=1,NBATCH)
213:      Cccc      write(IROT) (XSXV(N,3),N=1,NBATCH)
214:      Cccc      write(IROT) ((XKEF(J,N),J=1,7),N=1,NBATCH)
215:      call SRECOD( IROT, ITSK, 'XSOC', XSOC, NBATCH, IERR )
216:      call SRECOD( IROT, ITSK, 'XSXV1', XSXV(1,1), NBATCH, IERR )
217:      call SRECOD( IROT, ITSK, 'XSXV2', XSXV(1,2), NBATCH, IERR )
218:      call SRECOD( IROT, ITSK, 'XSXV3', XSXV(1,3), NBATCH, IERR )
219:      call SRECOD( IROT, ITSK, 'XKEF', XKEF, 5*NBATCH, IERR )
220:      C
221:      C .... FISSION SOURCE BANK .....
222:      C
223:      Ccccc      write(IROT) (XXXF(I),I=1,NFISB)
224:      Ccccc      write(IROT) (YYF(I),I=1,NFISB)
225:      Ccccc      write(IROT) (ZZF(I),I=1,NFISB)
226:      Ccccc      write(IROT) (IZZF(I),I=1,NFISB)
227:      call SRECOD( IROT, ITSK, 'XXXF', XXXF, NFISB, IERR )
228:      call SRECOD( IROT, ITSK, 'YYF', YYF, NFISB, IERR )
229:      call SRECOD( IROT, ITSK, 'ZZF', ZZF, NFISB, IERR )
230:      call SRECOI( IROT, ITSK, 'IZZF', IZZF, NFISB, IERR )
231:      if ( NLATT.ne.0 ) then
232:      Cccc      write(IROT) (LELVF(I),I=1,NFISB)
233:      Cccc      write(IROT) ((LZZF(I,J),J=1,NEST),I=1,NFISB)
234:      Cccc      write(IROT) ((LPOSF(I,J),J=1,NEST),I=1,NFISB)
235:      C
236:      call SRECOI( IROT, ITSK, 'LELVF', LELVF, NFISB, IERR )
237:      do 100 J = 1, NEST
238:      call SRECOI( IROT, ITSK, 'LZZF', LZZF(1,J), NFISB, IERR )
239:      call SRECOI( IROT, ITSK, 'LPOSF', LPOSF(1,J), NFISB, IERR
240:      &
241:      if ( JHLAT.ne.0 ) then
242:      Cccc      write(IROT) ((LCRSF(I,J),J=1,NEST),I=1,NFISB)
243:      call SRECOI( IROT, ITSK, 'LCRSF', LCRSF(1,J), NFISB,
244:      &
245:      IERR )
246:      end if
247:      continue
248:      end if
249:      C
250:      call SRECOI( IROT, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
251:      if ( JTLLT.ne.0 ) then
252:      Cccc      write(IROT) (IBRGF(I),I=1,NFISB)
253:      Cccc      write(IROT) ((IBSPF(I,K),I=1,NFISB),K=0,NEST)
254:      call SRECOI( IROT, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
255:      do 110 K = 0, NEST
256:      call SRECOI( IROT, ITSK, 'IBSPF', IBSPF(1,K), NFISB, IERR
257:      &
258:      )
259:      continue
260:      end if
261:      C
262:      C ... optional fission bank

```

src/gmvp/resto.f

```

261: C
262:     if ( KDFBK(0).gt.0 ) then
263:         do 130 K = 1, MDFBK
264:             if ( KDFBK(K).ne.0 ) then
265:                 TAG = ' '
266:                 write(TAG,(''DFBK'',I3)) K
267:                 call CCOMP( TAG, LEN(TAG), TAG, IFL )
268:
269:                 if ( K.eq.4 ) then
270:                     ND = NEST + 1
271:                 else if ( K.eq.5 ) then
272:                     ND = 3*NEST
273:                 else if ( K.eq.6 ) then
274:                     ND = 6*NEST
275:                 else
276:                     ND = 1
277:                 end if
278:
279:                 do 120 J = 0, ND - 1
280:                     call SRECOD( IROT, ITSK, TAG(:ICLEN2(TAG)),
281:                                DFBK(1,KDFBK(K)+J), NFISB, IERR )
282:                 &
283:                 120 continue
284:             end if
285:         continue
286:     if ( KIFBK(0).gt.0 ) then
287:         do 150 K = 1, MIFBK
288:             if ( KIFBK(K).ne.0 ) then
289:                 TAG = ' '
290:                 write(TAG,(''IFBK'',I3)) K
291:                 call CCOMP( TAG, LEN(TAG), TAG, IFL )
292:
293:                 if ( K.eq.6 ) then
294:                     NI = NEST
295:                 else
296:                     NI = 1
297:                 end if
298:
299:                 do 140 J = 0, NI - 1
300:                     call SRECOI( IROT, ITSK, TAG(:ICLEN2(TAG)),
301:                                IFBK(1,KIFBK(K)+J), NFISB, IERR )
302:                 &
303:                 140 continue
304:             end if
305:         continue
306:     end if
307: C
308: C .... point detector tally ....
309: C
310:     if ( NPDET.gt.0 ) then
311: Ccc write(IROT) ((SFLPD(i,j,1),i=1,ngroup),j=1,npdet)
312: Ccc write(IROT) ((SFLPD(i,j,2),i=1,ngroup),j=1,npdet)
313:         call SRECOD( IROT, ITSK, 'SFLPD', SFLPD, NGROUP*NPDET*2, IERR )
314:
315:         if ( JRESP.gt.0 ) then
316: Ccc write(IROT)(IROT) ((SREPD(i,j,1),i=1,npdet),j=1,nresp)
317: Ccc write(IROT)(IROT) ((SREPD(i,j,2),i=1,npdet),j=1,nresp)
318:             call SRECOD( IROT, ITSK, 'SREPD', SREPD, NPDET*NRESP*2, IERR
319:         &
320:         )
321:         end if
322:     end if
323: C .... SPECIAL TALLIES .....
324: C
325:     if ( NETALY.gt.0 ) then

```

```

326: Cccc write(IROT) ETALY
327:
328:         call SRECOD( IROT, ITSK, 'ETALY', ETALY, NLETAL*2, IERR )
329: C
330: C ... ETRV: added in file version 3.1 ....
331: C
332:         if ( NETRV.gt.0 ) then
333:             call SRECOD( IROT, ITSK, 'ETRV', ETRV, METRV*(NBATCH
334:             &
335:             -NSKIP), IERR )
336:         end if
337: C
338: C .... rewind ...
339: C
340: Cccc call RWIND(IROT)
341: C
342:         return
343:     end

```

src/gmvp/restor.f

```

1:      subroutine RESTOR( INP1, ISP,   SIGT, NSIG, JNEUT, JPHOT, JSEP
2:      &
3: C=====
4: C   THIS ROUTINE IS A COMBINATION OF PARTS OF READSG AND STORE.   *
5: C   IT READS ONLY THE P0 MATRIX AND TEMPORARILY STORES IT.       *
6: C   IN DDX MODE, PO MATRIX IS CALCULATED FROM DDX DATA.
7: C
8: C   NSIG(IDEL+KKK) : element ID of target material
9: C
10: C
11: C CALLED IN: GTNDSK
12: C CALLS: REARAG, PRSTOP
13: C=====
14: common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
15: & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
16: & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
17: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
18: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
19: & IADJ, NME, LOC, INGS, INSG, I1, I0,
20: & KKK, IXTAPE, IDEL, ITEML, ITEMG, IRSG, IRDSG,
21: & ISTR, IPRIN, IFMU, IMOM, IDTF, ISTAT, IPUN,
22: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
23: & INNN, IGGG, IPNGP, IPSPOG, IPSGOG, IANG, INUFIS
24: C
25: C
26: common /DDX/ IDDX, MAXU, MAXSD, IMAX, NLASTD, MAXDDX,
27: & ISTTRT, NGP1, NGG1
28: C
29: include '../shared/INC/_IOUNIT'
30: include 'INC/_IOUNIT2'
31: C
32: integer LW(300)
33: C
34: C
35: integer NSIG(1)
36: real SIGT(1)
37: character*4 HOL(12)
38: C
39: C**** EQUIVALENCE (SIGT(1),NSIG(1)) ***** ASSUMED *****
40: C
41: C
42: C ---- STATEMENT FUNCTION TO INDICATE 'POINTER' CALCULATION ! ----
43: C
44: IPOINT(IP,NSIZE) = IP + NSIZE
45: C
46: C -----
47: C
48: NME0 = NME
49: IXTAP = IABS(IXTAPE)
50: IWT = 0
51: IST = ISTART
52: II = INP1
53: N = ISP
54: K = 1
55: KS = 1
56: I3 = INP1 + 1
57: C
58: do 100 I = I3, ISP
59:   SIGT(I) = 0.0
60: 100 continue
61: C
62: C
63: C =====
64: C ===== DDX mode =====
65: C =====

```

```

66: C
67: if ( IDDX.ne.0 ) then
68:   NME = 1
69:   NGP1 = NGP
70:   NGG1 = NGG
71:   if ( IADJ.gt.0.and.JPHOT.ne.0 ) then
72:     NGG1 = NGP
73:     NGP1 = NGG
74:   end if
75:   if ( IADJ.eq.0.and.JNEUT.eq.0 ) then
76:     NGG1 = NGP
77:     NGP1 = 0
78:   end if
79: C
80:   NSRT1 = ISP + 1
81:   NSRT2 = ISP + 5*INGP
82:   NSRT3 = NSRT2 + 1
83: C***** NSRT4 = NSRT3+INGP*NSCT
84:   NSRT4 = IPOINT(NSRT3,(INDS+NUS)*NSCT)
85: C
86:   if ( NSRT4.gt.NLASTD ) then
87:     write(IMG,7000) NSRT4, NLASTD
88: 7000 format(1X,'XXX SUB.RESTOR (1) XXX'/10X,'NOT ENOUGH MEMORY ',
89:   & I10,('( ',I10,')')
90:     call PRSTOP( 1,
91:   & 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.' )
92:   stop 999
93: end if
94: C
95: C
96: if ( JNEUT.ne.0 ) then
97:   ID = IODDX
98: else
99:   ID = IOWDX2
100: end if
101: C
102: C
103: MAXU = NSCT
104: IMAX = 0
105: IMAXG = 0
106: C
107: 110 KDDX = 0
108: C
109: C
110: C ... read DDX library file & find target material (NSIG(IDEL+KKK)):
111: C
112: C
113: 120 JUMP = 1
114: C
115: C ... IMX : number of energy group
116: C maxsd : maximum of downscatter groups
117: C ic : dummy
118: C matno : material #
119: C hol : description
120: C
121: read(ID,end =250) IMX, MAXSD, IC, MATNO, (HOL(I),I=1,12)
122: C
123: write(IPR,7020) MATNO, ID
124: 7020 format(5X,' DDX-LIBRARY: MATNO =',I10,' FROM I/O UNIT ',I2)
125: C
126: C ... LW: number of downscatter energy groups for each group ...
127: C
128: if ( ID.eq.IODDX ) read(ID,end =250) (LW(I),I=1,IMX)
129: if ( ID.eq.IOWDX2 ) read(ID,end =250) (LW(I),I=NGP1+1,NGP1+IMX)
130: C

```

src/gmvp/restor.f

```

131: C
132: 7040      format(4I6,6A8)
133: 7060      format(12I6)
134: C
135: C
136: C ..... SKIP .....
137: C
138: C
139:       if ( MATNO.ne.IABS(NSIG(IDEL+KKK)) ) then
140:         JUMP      = 2
141: C
142: C ... skip cross sections ...
143:       read(ID,end =250)
144:       JUMP      = 3
145: C
146: C ... skip scattering matrices ...
147:       do 130 IG = 1, IMX
148:         read(ID,end =250)
149: 130      continue
150: C
151: C ... skip two records when input from IOWDX2 ...
152:       if ( ID.eq.IOWDX2 ) then
153:         read(ID,end =250)
154:         read(ID,end =250)
155:       end if
156: 7080      format(10E12.5)
157:       go to 120
158:     end if
159: C
160: C
161: C ..... MATERIAL FOUND .....
162: C
163: C
164:       if ( ID.eq.IODDX ) then
165:         IMAX      = IMX
166:         if ( JPHOT.ne.0 ) then
167:           ID       = IOWDX2
168:           go to 110
169:         end if
170:       else
171:         IMAXG      = IMX
172:         IMAXN      = IC
173:       end if
174: C
175:       N1          = MATNO
176:       N2          = MATNO
177:       N3          = KKK
178:       N4          = MATNO
179:       M2=IDEL+KKK
180:       NSIG(M2)=KKK
181: C
182: C ... output matno , kkk (relative # of materials input) ...
183: C
184:       write(IXTAPE) N1, N2, N3, N4, (HOL(I),I=1,12)
185: C
186:       JUMP      = 4
187:       KS1       = INGP
188:       KS2       = INGP*2
189:       KS3       = INGP*3
190:       KS4       = INGP*4
191: C
192: C
193: C
194:       ICHK      = 0
195: C

```

```

196:       if ( JPHOT*JNEUT.ne.0.and.IMAXN.lt.NGP1 ) then
197:         ICHK      = ICHK + 1
198:         write(IMG,7100) NGP1, IMAXN
199: 7100      format(1X,'XXX(RESTOR) NO. OF NEUTRON GROUPS (' ,I3,' ) IS',
200:         &        ' GREATER THAN PHOTON LIBRARY (' ,I3,' )')
201:       end if
202: C
203:       if ( NGG1.gt.IMAXG ) then
204:         ICHK      = ICHK + 1
205:         write(IMG,7120) NGG1, IMAXG
206: 7120      format(1X,'XXX(RESTOR) NO. OF PHOTON GROUPS (' ,I3,' ) IS',
207:         &        ' GREATER THAN LIBRARY (' ,I3,' )')
208:       end if
209: C
210:       NGPX1      = NGP1
211: C
212:       if ( NGP1.gt.IMAX+1 ) then
213:         ICHK      = ICHK + 1
214:         write(IMG,7140)
215: 7140      format(1X,'XXX(RESTOR) NO. OF NEUTRON GROUPS IS GREATER',
216:         &        ' THAN LIBRARY')
217:       else if ( NGP1.eq.IMAX+1 ) then
218:         NGPX1      = IMAX
219:       else
220:         if ( NSIG(IDEL+KKK).gt.0 ) go to 140
221:       end if
222: C
223:       if ( NUS.gt.0.and.NSIG(IDEL+KKK).le.0 ) then
224:         ICHK      = ICHK + 1
225:         write(IMG,*) 'XXX(RESTOR) NUS > 0 AND MATNO < 0, STOP'
226:       end if
227: C
228:       if ( NSIG(IDEL+KKK).gt.0 ) then
229:         ICHK      = ICHK + 1
230:         write(IMG,*)
231:         &        'XXX(RESTOR) CROSS SECTION INPUT FROM CARD SHOULD BE',
232:         &        ' REQUIRED (MATNO SHOULD BE < 0.)'
233:       end if
234: C
235: C ..... INPUT SUPPLEMENTARY CROSS SECTION DATA ( DDX ) .....
236: C
237: C
238: C
239:       read(IOIN,*) MAT, SIGTO, SIGPR, SIGC, SIGF, SIGFF
240: C
241:       write(IPR,7180) MAT, SIGTO, SIGPR, SIGC, SIGF, SIGFF
242: 7160      format(15,5E10.3)
243: 7180      format(' ',5X,' INPUT--MATNO,SIGTO SIGPR,SIGC,SIGF,',
244:         &        'SIGFF =' ,15,1P5E12.4)
245: C
246:       if ( MAT.ne.MATNO ) then
247:         write(IMG,*) 'XXX(RESTOR) BAD MATNO MAT = ', MAT,
248:         &        ' STOP '
249:         call PRSTOP( 1, 'BAD MATERIAL #.' )
250:         stop 888
251:       end if
252: C
253: C
254: 140      continue
255: C
256: C
257:       if ( ICHK.ne.0 ) then
258:         call PRSTOP( ICHK, 'ERRORS IN CROSS SECTION LIBRARY INPUT.'
259:         &        )
260:         stop 888

```

src/gmvp/restor.f

```

261:         end if
262: C
263: C     ... input cross sections ...
264: C
265:         if ( JNEUT.ne.0 ) then
266:             ID = IODDX
267:             read(IODDX,end =260) (SIGT(L),SIGT(L+KS1),SIGT(L+KS2),
268: &             SIGT(L+KS3),SIGT(L+KS4),L=NSRT1,NSRT1+NGPX1-1)
269:         end if
270: C
271:         if ( JPHOT.ne.0 ) then
272:             ID = IOWDX2
273:             read(IOWDX2,end =260) (SIGT(L),SIGT(L+KS1),SIGT(L+KS2),
274: &             SIGT(L+KS3),SIGT(L
275: &             +KS4),L=NSRT1+NGP1,NSRT1+NGP1+NGG1-1)
276:         end if
277: C
278:         KS1 = NSRT1
279:         KS2 = KS1 + INGP
280:         KS3 = KS2 + INGP
281:         KS4 = KS3 + INGP
282:         KS5 = KS4 + INGP
283:         KS6 = KS5 + INGP
284: C
285: C CCCCCC KSMAX = NSRT4+INGP*(INDS+IHT+NUS)*MAXU
286: C CCCCCC LSMAX = KSMAX+INGP*(INDS+IHT+NUS)
287:         KSMAX = IPOINT(NSRT4,INGP*INGP*MAXU)
288:         LSMAX = IPOINT(KSMAX,INGP*INGP)
289: C
290:         if ( LSMAX.gt.NLASTD ) then
291:             write(IMG,7200) LSMAX, NLASTD
292: 7200         format(1X,'XXX SUB.RESTOR (2) XXX'/10X,'NOT ENOUGH MEMORY ',
293: &             I10,'('',I10,'')')
294:
295:             call PRSTOP( 1,
296: &             'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.' )
297:             stop 999
298:         end if
299: C
300: 150         JUMP = 5
301:         do 160 L = NSRT4, KSMAX
302:             SIGT(L) = 0.0
303: 160         continue
304: C
305:         ID = IODDX
306:         do 190 I = 1, INGP
307: C
308: C ... in the case that cross section is not given manually in data X-4
309: C or "XTLAST"
310: C
311:         if ( I.ne.NGP1 .or. NSIG(IDEL+KKK).ge.0 ) then
312:             I9 = NGP1 - I + 1 + NUS
313:             if ( I.gt.NGP1 ) then
314:                 ID = IOWDX2
315:                 I9 = NGG1 - (I-NGP1) + 1 + NUS
316:             end if
317:             if ( I9.lt.LW(I) ) LW(I) = I9
318: C
319:             MAXS = LW(I)
320:             NSRT31 = NSRT3 + MAXS*MAXU - 1
321:             read(ID,end =260) (SIGT(L),L=NSRT3,NSRT31)
322: C
323:             call REARAG( I, MAXU, LW(I), INGP, NGP1, NGG1, LSMAX,
324: &             NLASTD, SIGT(KS1), SIGT(KS2), SIGT(KS3),
325: &             SIGT(KS4), SIGT(KS5), SIGT(NSRT3), SIGT(NSRT4),

```

```

326:         &             SIGT(KSMAX), SIGT, NSIG, LW(1) )
327: C
328:         else
329: C
330: C     THIS CASE SHOULD BE TOGATHER WITH NUS = 0
331: C
332:             LW(NGP1) = 1 + NUS
333:             NSRT31 = NSRT3 + MAXU*(1+NUS) - 1
334:             NSRT30 = NSRT3 + MAXU*NUS - 1
335: C
336:             do 170 L = NSRT3, NSRT30
337:                 SIGT(L) = 0.0
338: 170             continue
339: C
340:             do 180 L = 1, MAXU
341:                 SIGT(L+NSRT30) = (SIGT(IANG+L)-SIGT(IANG+L+1)) /2.
342: 180             continue
343: C
344:             INGP2 = NSRT1 + NGP1 - 1
345:             SIGT(INGP2) = SIGPR
346:             INGP2 = INGP2 + INGP
347:             SIGT(INGP2) = SIGF
348:             INGP2 = INGP2 + INGP
349:             SIGT(INGP2) = SIGC
350:             INGP2 = INGP2 + INGP
351:             SIGT(INGP2) = SIGFF
352:             INGP2 = INGP2 + INGP
353:             SIGT(INGP2) = SIGTO
354: C
355:             call REARAG( I, MAXU, LW(I), INGP, NGP1, NGG1, LSMAX,
356: &             NLASTD, SIGT(KS1), SIGT(KS2), SIGT(KS3),
357: &             SIGT(KS4), SIGT(KS5), SIGT(NSRT3), SIGT(NSRT4),
358: &             SIGT(KSMAX), SIGT, NSIG, LW(1) )
359:         end if
360: C
361: 190         continue
362: C
363: C
364: C     .... Output by integer form because of a bug in fortran library
365: C     in DEC/OSF pre 1.2 version.
366: C
367:         WRITE(IOWDX1) N1,N2,N3,N4,
368: C &             (HOL(I),I=1,12), (SIGT(L),L=NSRT1,NSRT2),
369: C X             (SIGT(L),L=NSRT4,KSMAX-1)
370: C
371:         write(IOWDX1) N1, N2, N3, N4, (HOL(I),I=1,12),
372: &             (NSIG(L),L=NSRT1,NSRT2), (NSIG(L),L=NSRT4,KSMAX-1)
373: C
374: C
375: Cccc         IMAX = INGP
376: Cccc         MAXSD = INGP
377: C
378:         if ( JNEUT.ne.0 ) then
379:             NGPX1 = NGP1 + 1
380:             if ( NSIG(IDEL+KKK).lt.0 ) NGPX1 = NGP1
381:             ID = IODDX
382:             do 200 I = NGPX1, IMAX
383:                 read(IODDX,end =260)
384: 200             continue
385:         end if
386: C
387:         if ( JPHOT.ne.0 ) then
388:             NGGX1 = NGG1 + 1
389:             ID = IOWDX2
390:             do 210 I = NGGX1, IMAXG

```

src/gmvp/restor.f

```

391:          read(IOWDX2,end =260)
392: 210      continue
393: C
394:          if ( JNEUT.ne.0 ) then
395: C
396:          read(IOWDX2,end =260) (SIGT(L),L=NSRT1,NSRT1+NGP1-1)
397: C
398: C .... Output by integer form because of a bug in fortran library
399: C in DEC/OSF pre 1.2 version.
400: C
401: Ccccc      WRITE(IOWDX1) (SIGT(L),L=NSRT1,NSRT1+NGP1-1)
402:          write(IOWDX1) (NSIG(L),L=NSRT1,NSRT1+NGP1-1)
403: C
404:          read(IOWDX2,end =260)
405:          &      (SIGT(L),L=NSRT1,NSRT1+IMAXG*NGP1-1)
406: C
407:          JSN      = NSRT1 - 1
408:          JSO      = NSRT1 - 1
409:          do 230 I = 1, NGP1
410:              do 220 L = 1, NGG1
411:                  SIGT(L+JSN) = SIGT(L+JSO)
412: 220          continue
413:              JSO      = JSO + IMAXG
414:              JSN      = JSN + NGG1
415: 230          continue
416: C
417: C
418: C .... Output by integer form because of a bug in fortran library
419: C in DEC/OSF pre 1.2 version.
420: C
421: Ccc          WRITE(IOWDX1) (SIGT(L),L=NSRT1,JSN)
422:          write(IOWDX1) (NSIG(L),L=NSRT1,JSN)
423:          end if
424:          end if
425: C
426:          do 240 L = NSRT1, LSMAX
427:              SIGT(L) = 0.0
428: 240          continue
429: C
430:          IMAX      = INGP
431:          MAXSD      = INGP
432: C
433:          go to 270
434: C
435: C
436: C ..... END OF CROSS SECTION LIBRARY HAS BEEN FOUND .....
437: C
438: C
439: 250      if ( KDDX.ne.1 ) then
440:          KDDX      = 1
441:          call RWIND( ID )
442:          go to 120
443:          end if
444: C
445: C
446:          I92      = NSIG(IDEL+KKK)
447:          write(IMG,7220) I92, ID, JUMP
448: 7220      format(/' XXX ELEMENT ',I5,' CANNOT BE FOUND ON DDX LIBRARY',
449:          &      ' (I/O UNIT ',I2,',',5X,'(JUMP=',I5,',')')
450:          call PRSTOP( 1, 'CROSS SECTION ERROR.' )
451:          stop 888
452: C
453: 260      write(IMG,7240) MATNO, ID, JUMP
454: 7240      format(/' XXX CROSS SECTION DATA IS INSUFFICIENT FOR',
455:          &      ' ELEMENT ',I5,' FROM I/O UNIT ',I2,' (JUMP=',I5,',')')

```

```

456:          call PRSTOP( 1, 'CROSS SECTION ERROR.' )
457:          stop 888
458: C
459: C ... end of DDX mode ...
460: C
461:          end if
462: C
463: C =====
464: C ===== DDX & PL mode =====
465: C =====
466: C
467: C 270 read(IXTAP,end =300) N1, N2, N3, N4, (HOL(I),I=1,12)
468: C      M2      = IDEL + KKK
469: C** 1999/11/4
470: 270 M2      = IDEL + KKK
471:          read(IXTAP,end =300) N1, N2, N3, N4, (HOL(I),I=1,12)
472: C
473:          if ( IDDX.eq.0.and.N4-NSIG(M2).ne.0 .or. IDDX.ne.0
474:          &      .and.N4-IABS(NSIG(IDEL+KKK)).ne.0 ) then
475: C
476:          if ( JSEP.eq.0 ) then
477:              read(IXTAP,end =300)
478:          else
479:              do 280 J = 1, INGP
480:                  read(IXTAP,end =300)
481: 280          continue
482:          end if
483: C          DUMMY READ
484:          go to 270
485:          end if
486: C
487:          if ( JSEP.eq.0 ) then
488:              read(IXTAP) (SIGT(II+L),L=1,INGP*(INDS+IHT+NUS))
489:          else
490:              MMMM      = II
491:              do 290 J = 1, INGP
492:                  read(IXTAP) (SIGT(MMMM+L),L=1,INDS+IHT+NUS)
493:                  MMMM      = MMMM + INDS + IHT + NUS
494: 290          continue
495:          end if
496: C
497:          FK          = KKK
498:          FK          = FK/NCOEF + .98
499:          KOK          = FK
500:          write(IPR,7260) KOK, N4, (HOL(I),I=1,12)
501: 7260      format(/4X,'*** ELEMENT ',I5,' ID=',I10,' <',12A4,'>')
502:          go to 310
503: C
504: C ..... END OF DATASET HAS BEEN FOUND .....
505: C
506: 300      if ( IWT.le.0 ) then
507:          IWT          = 1
508:          call RWIND( IXTAP )
509:          go to 270
510:          end if
511: C
512:          I92          = NSIG(M2)
513:          write(IMG,7280) I92, IXTAP
514: 7280      format(/1x,'XXX(RESTOR) ELEMENT ',I5,
515:          &      ' CANNOT BE FOUND ON I/O UNIT ',I3)
516:          call PRSTOP( 1, 'CROSS SECTION ERROR (INPUT FROM FILE).' )
517:          stop 888
518: C
519: 310      continue
520:          INN          = INP1

```

src/gmvp/restor.f

```
521:      NGP3      = INDS + IHT + NUS
522:      NTGMT      = NTG + NME - 1
523: C
524: C ..... FORWARD PROBLEM
525: C
526:      if ( IADJ.le.0 ) then
527:      do 330 I = NME, NTGMT
528:          NDSK      = I + NUS - NME + 1
529:          if ( NDSK-NUS-NDS-NDSG.gt.0 ) NDSK = NUS + NDS + NDSG
530:          IB          = IST
531:          do 320 J = 1, NDSK
532:              IA      = I - NME + 1 - J + NUS
533:              if ( IA-NTGMT+1.le.0 ) then
534:                  continue
535:                  IJ      = IB + IA*NTS + J
536:                  K        = (I-1)*NGP3 + J + INN + IHT
537:                  SIGT(IJ) = SIGT(K)
538:              end if
539:          320      continue
540:      330      continue
541: C
542: C ..... ADJOINT PROBLEM
543: C
544:      else
545:      do 350 I = NME, NTGMT
546:          L          = NTGMT - I + 1
547:          NDSK      = I + NUS - NME + 1
548:          if ( NDSK-NUS-NDS-NDSG.gt.0 ) NDSK = NUS + NDS + NDSG
549: C
550: C STORE PO MATRIX
551: C
552:          do 340 J = 1, NDSK
553:              K        = (I-1)*NGP3 + J + INN + IHT
554:              IJ        = IST + (L-1)*NTS + J
555:              SIGT(IJ) = SIGT(K)
556:          340      continue
557:      350      continue
558:      end if
559: C
560: C
561:      NME          = NME0
562: C
563:      return
564:      end
```


src/gmvp/restot.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine RESTOT(IOPR, H, MONITR, ITSK,TITLE )
3: C=<MVP>=====
4: C PURPOSE: Interface routine to RESTO which outputs restart file record.
5: C CALLED IN: ACTSGL,ACTMPP,ACTSMP,ACTVPP
6: C CALLS:
7: C
8: C -----
9: C
10: C   arguments ( i = input, o = output, c =constant, w = work )
11: C
12: C i IOPR : message printout I/O unit
13: C io H   : task local dynamic memory area
14: C i MONITR : do not print any message in RESTO if zero.
15: C i ITSK : data are stored on local data of task ITSK.
16: C         (ITSK may not same as current IDTASK)
17: C i TITLE : problem title
18: C -----
19:   real H(*)
20:   character TITLE(2)*72
21: C
22: CCCC include '../shared/INC/_ARRAY'
23:   include '../shared/INC/_TASKDT'
24:   include '../shared/INC/_LISTOFF'
25:   include 'INC/_FLAGS'
26:   include '../shared/INC/_SIZES'
27:   include 'INC/_SIZES2'
28:   include 'INC/_PXSEC'
29:   include 'INC/_XBANK'
30:   include 'INC/_STACK'
31:   include 'INC/_XWORK'
32:   include '../shared/INC/_PGEOM'
33:   include '../shared/INC/_PVRED'
34:   include 'INC/_PSOUR'
35:   include '../shared/INC/_PTALY0'
36:   include 'INC/_PTALY'
37:   include '../shared/INC/_STALY'
38:   include '../shared/INC/_PTLSP'
39:   include '../shared/INC/_LISTON'
40:   include '../shared/INC/_WORDL'
41: C
42: C -----
43: C
44:   call RESTO( IOPR, MONITR, ITSK, NTHIST, IRAND, H(LWSUM), H(LWLEK),
45: &
46:   &      NGROUP, NRESP, NREG, NTREG, NBATCH, NMEMO,
47:   &      NEVENT, NZONE, H(LKMEMO), H(LWCNTR), H(LNCNTR),
48:   &      H(LSFLTR), H(LSFLCL), H(LSRETR), H(LSRECL),
49:   &      H(LXAVT), H(LAAVT),H(LEAVT),
50:   &      NFISB, NFBANK,NFBNK0, NLENG, H(LXXXF), H(LYYYYF),
51:   &      H(LZZZF), H(LIZZF), H(LIBRGF),
52:   &      H(LIBSPF), H(LXSOC), H(LXSXV), H(LXKEF), NLATT, NEST,
53:   &      H(LLEVLF), H(LLZZF), H(LLPOSF), H(LLCRSF),
54:   &      H(LDFBK),KDFBK,MDFBK,H(LIFBK),KIFBK, MIFBK,
55:   &      NPDET, H(LSFLPD), H(LSREPD),
56:   &      NETALY, H(LETALY), NLETAL,
57:   &      NETRV, NSKIP, METRV, H(LETRV) )
58:   return
59:   end
```

src/gmvp/restrt.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine RESTRT( IOPR, ITSK, NTHIST,IRAND, WSUM, WLEK,
3:   &                 NGROUP,NRESP, NREG, NTREG, NBATCH,NMEMO,
4:   &                 NEVENT,NZONE, KMEMO, WCNTR, NCNTR, SFLTR,
5:   &                 SFLCL, SRETR, SRECL, XAVT, AAVT, EAVT,
6:   &                 NFISB, NFBANK,NFBNK0,NLENG, XXXF, YYYYF, ZZZF,
7:   &                 IZZF, IBRGF, IBSPF, XSOC, XSXV, XKEF,
8:   &                 NLATT, NEST, LEVLF, LZZF, LPOSF, LCRSF, DFBK,
9:   &                 KDFBK, MDFBK, IFBK, KIFBK, MIFBK, NPDET,
10:  &                 SFLPD, SREPD, NETALY,ETALY, NLETAL,NETRV,
11:  &                 NSKIP, METRV, ETRV )
12: C=<GMVP>=====
13: C PURPOSE: Input restart information
14: C   (header - parameter must be input in RESTI0)
15: C
16: C CALLED IN: ACTSGL,ACTMFP,ACTVPP,ACTSMP
17: C CALLS: SRECII[R[D]
18: C
19: C I/O UNIT OF RESTART INPUT : FT10F001 (IRIN)
20: C I/O UNIT OF RESTART OUTPUT: FT20F001 (IROT)
21: C
22: C=====
23:   include 'INC/_FLAGS'
24:   include '../shared/INC/_IOUNIT'
25:   include 'INC/_IOUNIT2'
26: C/#IF PARA(PVM)
27:   include '../shared/INC/_PVMPARA'
28: C/#ELSEIF PARA(MPI)
29: *   include 'mpi.h'
30: C/#ELSEIF PARA(VPP)
31:   include '../shared/INC/_VPPPARA'
32: C/#ENDIF
33:   include '../shared/INC/_TASKDT'
34:
35: C=<memo>= The seed value is saved as common variables
36: C   for a restart calculation. This treatment will be
37: C   modified in the future.
38:   include '../shared/INC/_RAND'
39: C
40:   real*8 WSUM, WLEK
41: C/#IF INTEGER8
42: *   integer*8 NTHIST
43: C/#ELSE
44:   integer NTHIST
45: C/#ENDIF
46: C
47: C ..... EVENT MONITOR.....
48: C
49:   real*8 WCNTR(NEVENT), NCNTR(NEVENT)
50:   real XAVT(3), AAVT(3), EAVT
51: C
52: C ..... NEXT-ZONE MEMORY .....
53: C
54:   integer KMEMO(NZONE,NMEMO)
55: C
56: C ..... TALLY ARRAYS .....
57: C
58:   real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
59:   &       SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
60: C
61:   real*8 SFLPD(NGROUP,NPDET,2), SREPD(NPDET,NRESP,2)
62: C
63:   real*8 ETALY(NLETAL,2)
64:   real*8 ETRV(METRV,*)
65: C

```

```

66: C   ... EIGENVALUE TALLY .....
67: C
68:   real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(5,NLENG)
69: C
70: C   ... FISSION PARTICLE (SOURCE) BANK .....
71: C
72:   real*8 XXXF(NFBNK0), YYYYF(NFBNK0), ZZZF(NFBNK0)
73:   integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
74:   &       LCRSF(NFBNK0,NEST), IZZF(NFBNK0)
75:   integer IBRGF(NFBNK0), IBSPF(NFBNK0,0:NEST)
76: C
77:   real*8 DFBK(NFBNK0,*)
78:   integer KDFBK(0:MDFBK)
79:   integer IFBK(NFBNK0,*)
80:   integer KIFBK(0:MIFBK)
81: C
82: C ..... local data .....
83: C
84:   character*32 TAG
85:   integer IWRK(32)
86:   real RWRK(32)
87:   real*8 DWRK(32)
88:
89: C/#IF INTEGER8
90: *   integer*8 I4MAX
91: C/#ELSE
92:   integer I4MAX
93: C/#ENDIF
94:   parameter( I4MAX = 2147483647 ) ! 2**31-1
95: C
96: C-----
97: C
98:   IERR = 0
99: C
100:   NPKI = 10
101:   call SRECII( IRIN, ITSK, 'TASKHEADER', IWRK, NPKI, IERR )
102: C
103:   if ( IERR.ne.0 ) then
104:     write(IMG,*) 'XXX(RESTRT) Cannot find an expected',
105:   &               ' task header of restart records. Task-ID: ', IDTASK,
106:   &               ' task-ID on which data are stored:', ITSK
107:   end if
108: C
109: C ... idtask, NTHIST, IRAND, NBATCH, NFISB .....
110: C
111:   if ( IDTASK.eq.ITSK ) then
112:     ITSK1 = IWRK(1)
113:     NTHIST = IWRK(2)
114:     IRAND = IWRK(3)
115:     NBATCH = IWRK(4)
116:     NFISB = IWRK(5)
117:     IRNSU = IWRK(6)
118:     IRNSM = IWRK(7)
119:     IRNSL = IWRK(8)
120:     IQNTH = IWRK(9)
121:     IRNTH = IWRK(10)
122: C
123: C ... Restore NTHIST from 2 parts for more than (2**31-1) histories.
124: C This is the treatment not to change restart output format.
125: C It should be modified in the future.
126: C
127:   NTHIST = IQNTH*I4MAX + IRNTH
128: end if
129: C
130:   NPKD = 3

```

src/gmvp/restrt.f

```

131:      call SRECID( IRIN, ITSK, 'WSUM,WLEK', DWRK, NPKD, IERR )
132: C
133:      if ( IDTASK.eq.ITSK ) then
134:          WSUM      = DWRK(1)
135:          WLEK      = DWRK(2)
136:          RNCTR     = DWRK(3)
137:      end if
138: C
139:      write(IOPR,*) ' == Task ', IDTASK,
140:      &      ' input task wise restart data header '
141: C
142:      if ( IDTASK.eq.1.and.ITSK.eq.1 ) then
143:          write(IOPR,*) ' IDTASK marked on record: ', ITSK1
144:          write(IOPR,*) ' NTHIST : ', NTHIST
145:          write(IOPR,*) ' IRAND : ', IRAND
146:          write(IOPR,*) ' NBATCH : ', NBATCH
147:          write(IOPR,*) ' NFISB : ', NFISB
148:          write(IOPR,*) ' WSUM : ', WSUM
149:          write(IOPR,*) ' WLEK : ', WLEK
150:          write(IOPR,*) ' IRNSU : ', IRNSU
151:          write(IOPR,*) ' IRNSM : ', IRNSM
152:          write(IOPR,*) ' IRNSL : ', IRNSL
153:          write(IOPR,*) ' RNCTR : ', RNCTR
154:          write(IOPR,*) ' IQNTH : ', IQNTH
155:          write(IOPR,*) ' IRNTH : ', IRNTH
156:      else
157: C/#IF PARA(PVM MPI)
158:          MSGNUM = 111
159:          IT0 = 1
160:          if ( IDTASK.ne.IT0 ) then
161:              call MVPCOM_SEND_I4( IT0, MSGNUM, IWRK, NPKI, INFO )
162:              call MVPCOM_SEND_R8( IT0, MSGNUM, DWRK, NPKD, INFO )
163:          else
164:              call MVPCOM_RECV_I4( ITSK, MSGNUM, IWRK, NPKI, INFO )
165:              call MVPCOM_RECV_R8( ITSK, MSGNUM, DWRK, NPKD, INFO )
166:          end if
167:          write(IOPR,*) ' IDTASK marked on record: ', IWRK(1)
168:          write(IOPR,*) ' NTHIST : ', IWRK(2)
169:          write(IOPR,*) ' IRAND : ', IWRK(3)
170:          write(IOPR,*) ' NBATCH : ', IWRK(4)
171:          write(IOPR,*) ' NFISB : ', IWRK(5)
172:          write(IOPR,*) ' WSUM : ', DWRK(1)
173:          write(IOPR,*) ' WLEK : ', DWRK(2)
174:          write(IOPR,*) ' IRNSU : ', IWRK(6)
175:          write(IOPR,*) ' IRNSM : ', IWRK(7)
176:          write(IOPR,*) ' IRNSL : ', IWRK(8)
177:          write(IOPR,*) ' RNCTR : ', DWRK(3)
178:          write(IOPR,*) ' IQNTH : ', IWRK(9)
179:          write(IOPR,*) ' IRNTH : ', IWRK(10)
180: C/#ENDIF
181:      end if
182:
183:      call SRECII( IRIN, ITSK, 'KMEMO', KMEMO, NZONE*NMEMO, IERR )
184:      call SRECID( IRIN, ITSK, 'WCNTR', WCNTR, NEVENT, IERR )
185:      call SRECID( IRIN, ITSK, 'NCNTR', NCNTR, NEVENT, IERR )
186:
187:      call SRECIR( IRIN, ITSK, 'XAVT,AAVT,EAVT', RWRK, 7, IERR )
188:
189:      if ( IDTASK.eq.ITSK ) then
190:          XAVT(1) = RWRK(1)
191:          XAVT(2) = RWRK(2)
192:          XAVT(3) = RWRK(3)
193:          AAVT(1) = RWRK(4)
194:          AAVT(2) = RWRK(5)
195:          AAVT(3) = RWRK(6)

```

```

196:          EAVT      = RWRK(7)
197:      end if
198: C
199: C ..... FLUXES .....
200: C
201: c      read(IRIN) SFLTR
202: c      read(IRIN) SFLCL
203: C
204:      call SRECID( IRIN, ITSK, 'SFLTR', SFLTR, NGROUP*NTREG*2, IERR )
205:      call SRECID( IRIN, ITSK, 'SFLCL', SFLCL, NGROUP*NTREG*2, IERR )
206: C
207: C ..... REACTION RATES .....
208: C
209:      if ( JRESP.ne.0 ) then
210:          if ( NRESP.gt.0 ) then
211: ccccc      read(IRIN) SRETR
212: ccccc      read(IRIN) SRECL
213:          call SRECID( IRIN, ITSK, 'SRETR', SRETR, NTREG*NRESP*2, IERR
214:          &
215:          call SRECID( IRIN, ITSK, 'SRECL', SRECL, NTREG*NRESP*2, IERR
216:          &
217:          end if
218:      end if
219: C
220:      if ( JEIGN.ne.0 ) then
221: C
222: C .... EIGEN VALUE .....
223: C
224: ccccc      read(IRIN) (XSOC(N),N=1,NBATCH)
225: ccccc      read(IRIN) (XSXV(N,1),N=1,NBATCH)
226: ccccc      read(IRIN) (XSXV(N,2),N=1,NBATCH)
227: ccccc      read(IRIN) (XSXV(N,3),N=1,NBATCH)
228: ccccc      read(IRIN) ((XKEF(J,N),J=1,7),N=1,NBATCH)
229:          call SRECID( IRIN, ITSK, 'XSOC', XSOC, NBATCH, IERR )
230:          call SRECID( IRIN, ITSK, 'XSXV1', XSXV(1,1), NBATCH, IERR )
231:          call SRECID( IRIN, ITSK, 'XSXV2', XSXV(1,2), NBATCH, IERR )
232:          call SRECID( IRIN, ITSK, 'XSXV3', XSXV(1,3), NBATCH, IERR )
233:          call SRECID( IRIN, ITSK, 'XKEF', XKEF, 5*NBATCH, IERR )
234: C
235: C .... FISSION SOURCE BANK .....
236: C
237:      if ( NFBANK.lt.NFISB ) then
238:          write(IMG,*)
239:          &          '!!!(RESTRT) LENGTH OF FISSION BANK IS LESS THAN ',
240:          &          'THAT IN RESTART FILE (NFBANK=', NFBANK, ',NFISB=',
241:          &          NFISB
242:          call CNTERR( 'WARNING' )
243:          NFBANK = NFISB
244:      end if
245: C
246: ccccc      read(IRIN) (XXXF(I),I=1,NFISB)
247: ccccc      read(IRIN) (YYYY(I),I=1,NFISB)
248: ccccc      read(IRIN) (ZZZF(I),I=1,NFISB)
249: ccccc      read(IRIN) (IZZF(I),I=1,NFISB)
250:          call SRECID( IRIN, ITSK, 'XXXF', XXXF, NFISB, IERR )
251:          call SRECID( IRIN, ITSK, 'YYYY', YYYY, NFISB, IERR )
252:          call SRECID( IRIN, ITSK, 'ZZZF', ZZZF, NFISB, IERR )
253:          call SRECII( IRIN, ITSK, 'IZZF', IZZF, NFISB, IERR )
254:          if ( NLATT.ne.0 ) then
255: ccccc      read(IRIN) (LELVF(I),I=1,NFISB)
256: ccccc      read(IRIN) ((LZZF(I,J),J=1,NEST),I=1,NFISB)
257: ccccc      read(IRIN) ((LPOSF(I,J),J=1,NEST),I=1,NFISB)
258: C
259:          call SRECII( IRIN, ITSK, 'LELVF', LEVLf, NFISB, IERR )
260:          do 100 J = 1, NEST

```

src/gmvp/restrt.f

```

261:      call SRECII( IRIN, ITSK, 'LZZF', LZZF(1,J), NFISB, IERR )
262:      call SRECII( IRIN, ITSK, 'LPOSF', LPOSF(1,J), NFISB, IERR
263:      &
264:      if ( JHLAT.ne.0 ) then
265: cccc      read(IRIN) ((LCRSF(I,J),J=1,NEST),I=1,NFISB)
266:      call SRECII( IRIN, ITSK, 'LCRSF', LCRSF(1,J), NFISB,
267:      &          IERR )
268:      end if
269: 100      continue
270: end if
271: C
272:      call SRECII( IRIN, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
273:      if ( JTLLT.ne.0 ) then
274: cccc      read(IRIN) (IBRGF(1),I=1,NFISB)
275: cccc      read(IRIN) ((IBSPF(I,K),I=1,NFISB),K=0,NEST)
276: Ccccc      call SRECII( IRIN, ITSK, 'IBRGF', IBRGF, NFISB, IERR )
277:      do 110 K = 0, NEST
278:      call SRECII( IRIN, ITSK, 'IBSPF', IBSPF(1,K), NFISB, IERR
279:      &
280: 110      continue
281:      end if
282: C
283: C      ... optional fission bank
284: C
285:      if ( KDFBK(0).gt.0 ) then
286:      do 130 K = 1, MDFBK
287:      if ( KDFBK(K).ne.0 ) then
288:      TAG = ' '
289:      write(TAG,(''DFBK'',I3)) K
290:      call CCOMP( TAG, LEN(TAG), TAG, IFL )
291:
292:      if ( K.eq.4 ) then
293:      ND = NEST + 1
294:      else if ( K.eq.5 ) then
295:      ND = 3*NEST
296:      else if ( K.eq.6 ) then
297:      ND = 6*NEST
298:      else
299:      ND = 1
300:      end if
301:
302:      do 120 J = 0, ND - 1
303:      call SRECID( IRIN, ITSK, TAG(:ICLEN2(TAG)),
304:      &          DFBK(1,KDFBK(K)+J), NFISB, IERR )
305: 120      continue
306:      end if
307: 130      continue
308:      end if
309:      if ( KIFBK(0).gt.0 ) then
310:      do 150 K = 1, MIFBK
311:      if ( KIFBK(K).ne.0 ) then
312:      TAG = ' '
313:      write(TAG,(''IFBK'',I3)) K
314:      call CCOMP( TAG, LEN(TAG), TAG, IFL )
315:
316:      if ( K.eq.6 ) then
317:      NI = NEST
318:      else
319:      NI = 1
320:      end if
321:
322:      do 140 J = 0, NI - 1
323:      call SRECII( IRIN, ITSK, TAG(:ICLEN2(TAG)),
324:      &          IFBK(1,KIFBK(K)+J), NFISB, IERR )
325: 140      continue

```

```

326:      end if
327: 150      continue
328:      end if
329:      end if
330: C
331: C      .... point detector tally ....
332: C
333:      if ( NPDET.gt.0 ) then
334: ccc      read(IRIN) ((SFLPD(i,j,1),i=1,ngroup),j=1,npdet)
335: ccc      read(IRIN) ((SFLPD(i,j,2),i=1,ngroup),j=1,npdet)
336:      call SRECID( IRIN, ITSK, 'SFLPD', SFLPD, NGROUP*NPDET*2, IERR )
337:
338:      if ( JRESP.gt.0 ) then
339: ccc      read(IRIN) ((SREPD(i,j,1),i=1,npdet),j=1,nresp)
340: ccc      read(IRIN) ((SREPD(i,j,2),i=1,npdet),j=1,nresp)
341:      call SRECID( IRIN, ITSK, 'SREPD', SREPD, NPDET*NRESP*2, IERR
342:      &
343:      end if
344:      end if
345: C
346: C      .... SPECIAL TALLIES ....
347: C
348:      if ( NETALY.gt.0 ) then
349: cccc      read(IRIN) ETALY
350:      call SRECID( IRIN, ITSK, 'ETALY', ETALY, NLETAL*2, IERR )
351: C
352: C      ... ETRV: added in file version 3.1 ....
353: C
354:      if ( NETRV.gt.0 ) then
355:      call SRECID( IRIN, ITSK, 'ETRV', ETRV,
356:      &          METRV*(NBATCH-NSKIP), IERR )
357:      end if
358:      end if
359: C
360:      return
361:      end

```

src/gmvp/seaone.f

```

1: C/IF ARGSAVE
2: * SUBROUTINE SEAONE(IOW, A, CHA,
3: * N NBANK,NZONE,NSDA,NZDA,IDEFER,NLOST,NDEAD,
4: * N NREFS,NGROUP,NGP1,NREG,NTIME,
5: * N IRAND,NLBZ,NEST,NSUZON,NSPACE,DEPS,NMKREG,
6: * B XXX,YYY,ZZZ,AAA,BBB,CCC,
7: * B WWW,IZZ,IGG,TTT,ITT,LEVL,LZZ,
8: * B LPOS,LCRS,XIM,KLSF,IBREG,IBSPC,
9: * B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
10: * S LSFFL,NFFL,IZFFL,LSSRC,NNXT,IZNXT,
11: * S LSDED,LSREF,ISREF,IZREF,NBREF,KSREF,
12: * * LSLAT,IZLAT,NXLT,
13: * G SDA,KZMAT,KZREG,KZDA,KZAA,ksfbd,
14: * * KCELL,IPCEL,MLBZZ,ISUSP,MKREG,
15: * T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD,WKILD,NSURV,
16: * T WSURV,NSPLT,WSPLT,NCNTR,WCNTR,
17: * T JDTRG,DTAL,JTEVE,LTEVE,KTEVE,NTEVE,DTALY,RESP,
18: * W X,Y,Z,W,IBP,IFI,
19: * W IZI,IWK,IFL,R,ISRF,FXYZ)
20: C/#ELSE
21: subroutine SEAONE(IOW, A, CHA,
22: N NBANK,NZONE,NSDA,NZDA,IDEFER,NLOST,NDEAD,
23: N NREFS,NGROUP,NGP1,NREG,NTIME,
24: N IRAND,NLBZ,NEST,NSUZON,NSPACE,DEPS,NMKREG,
25: B XXX,YYY,ZZZ,AAA,BBB,CCC,
26: B WWW,IZZ,IGG,TTT,ITT,LEVL,LZZ,
27: B LPOS,LCRS,XIM,KLSF,IBREG,IBSPC,
28: B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
29: S LSFFL,NFFL,IZFFL,LSSRC,NNXT,IZNXT,
30: S LSDED,LSREF,ISREF,IZREF,NBREF,KSREF,
31: * LSLAT,IZLAT,NXLT,
32: G SDA,KZMAT,KZREG,KZDA,KZAA,ksfbd,
33: * KCELL,IPCEL,MLBZZ,ISUSP,MKREG,
34: T XIMP,WKIL,WSRV,WTIME,WLLIM,NKILD,WKILD,NSURV,
35: T WSURV,NSPLT,WSPLT,NCNTR,WCNTR,
36: T JDTRG,DTAL,JTEVE,LTEVE,KTEVE,NTEVE,DTALY,RESP,
37: W X,Y,Z,W,IBP,IFI,
38: W IZI,IWK,IFL,R,ISRF,FXYZ,
39: % MZONE,JMEM,NMEMO,KMEMO,MEMC,MEMZ)
40: C/#ENDIF
41: C=====
42: C PURPOSE: FIND ZONES TO ENTER. (ONE ZONE LOGIC)
43: C CALLED IN: ACTION
44: C CALLS: JUDGE,VMNTR1,STALN7
45: C
46: C=====
47: C
48: C === INTER-STACK DATA FLOW ===
49: C
50: C NEXT-ZONE SEARCH STACK---+---+----> FREE-FLIGHT STACK
51: C (LSSRC,IZNXT,NNXT) | | (LSFFL,IZFFL,NFFL)
52: C DEAD PARTICLE STACK ----+----> LATTICE-SEARCH STACK
53: C (LSDED,NDEAD) | | (LSLAT,IZLAT,NXLT)
54: C | |
55: C | |----> REFLECTION STACK
56: C | (LSREF,ISREF,IZREF,NBREF)
57: C +----> DEAD-PARTICLE STACK
58: C (LSDED,NDEAD)
59: C
60: C === BANK DATA TO BE UPDATED ===
61: C
62: C IZZ : ZONE #
63: C WWW : WEIGHT (SPLITTING OR RUSSIAN ROULETTE)
64: C
65: C === BANK DATA ADDED NEWLY (SPLITTING) ===
66: C
67: C (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #

```

```

66: C IGG, WWW : ENERGY GROUP & WEIGHT
67: C LEVL,LZZ,LPOS,LCRS : LATTICE-PARAMETER
68: C
69: C=====
70: implicit real*8(D)
71: C
72: real A(*)
73: character*4 CHA(*)
74: C
75: include 'INC/_FLAGS'
76: include '../shared/INC/_TASKDT'
77: C
78: C ..... TALLY .....
79: C
80: integer JDTRG(NREG,*), JTEVE(NTEVE), LTEVE(NTEVE+1), KTEVE(*)
81: integer IDTAL(*)
82: real*8 DTALY(*)
83: C
84: real RESP(*)
85: C
86: C ..... BANK .....
87: C
88: real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
89: & CCC(NBANK)
90: real WWW(NBANK), XIM(NBANK)
91: real*8 TTT(NBANK)
92: integer ITT(NBANK)
93: integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
94: & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
95: integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
96: C
97: real*8 DBNK(NBANK,*)
98: integer KDBNK(0:MDBNK)
99: integer IBNK(NBANK,*)
100: integer KIBNK(0:MIBNK)
101: C
102: C ..... STACK .....
103: C
104: integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSSRC(NBANK),
105: & NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK), LSREF(NBANK),
106: & ISREF(NBANK), IZREF(NBANK), NBREF(NREFS+1), LSLAT(NBANK),
107: & IZLAT(NBANK), NXLT(*)
108: C
109: C ..... GEOMETRY .....
110: C
111: real*8 SDA(NSDA)
112: integer KZMAT(NZONE), KZREG(NZONE), KSREF(NZDA), KZDA(2,NZDA),
113: & KZAA(NZONE+1), KMEMO(NZONE,NMEMO), MEMC(NZONE,NMEMO),
114: & MEMZ(NZONE), KCELL(NZONE), IPCEL(*), MLBZZ(NZONE)
115: integer KSFBD(NZDA)
116: integer ISUSP(NSUZON,NSPACE)
117: C
118: C ..... VARIANCE REDUCTION .....
119: C
120: real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
121: real WTIME(NTIME)
122: real WLLIM
123: C
124: C ..... TALLY (MONITOR) .....
125: C
126: integer MKREG(NREG,2)
127: C
128: real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG)
129: real*8 NSPLT(NGROUP,NREG)
130: real*8 WKILD(NGROUP,NREG), WSURV(NGROUP,NREG)

```

src/gmvp/seaone.f

```

131:      real*8 WSPLT(NGROUP,NREG)
132:      real*8 NCNTR(*), WCNTR(*)
133: C
134: C ..... WORK AREA .....
135: C
136:      real*8 X(NBANK), Y(NBANK), Z(NBANK), W(NBANK)
137:      real R(NBANK), FXYZ(NBANK)
138:      integer IBP(NBANK), IZI(NBANK), IWK(NBANK), ISRF(NBANK)
139:      logical IFI(NBANK), IFL(NBANK), JMEM
140: C
141: C ..... LOCAL BUT STATIC VARIABLES (INITIALIZAED BY DATA STATEMENT)
142: C
143:      parameter( MWARN      = 10 )
144:      logical JRR
145: C
146: C ..... counter variable definition .....
147: C
148:      include '../shared/INC/_COUNTS'
149: C
150:      include '../shared/INC/_PMLATT'
151:      include '../shared/INC/_SFLATT'
152: C
153: C/#IF ARGSAVE
154: *      RETURN
155: C
156: C-----
157: C
158: *      ENTRY      SEAONE( MZONE ,JMEM ,NMEMO ,KMEMO ,MEMC,MEMZ)
159: C/#ENDIF
160: C
161: CM1999-4-9 for special tally #7
162:      NDEAD0 = NDEAD
163:      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
164:          IPAR = 0
165:      else if ( JNEUT.ne.0 ) then
166:          IPAR = 1
167:      else
168:          IPAR = 2
169:      end if
170: C
171:      WMIN = WLLIM
172: C
173:      NZ1 = NZONE + 1
174:      NR1 = NREFS + 1
175: C
176: C .... MZONE: > 0 : SEARCH FOR ZONE MZONE, < 0 : DEFERRED PARTICLES
177: C
178: C
179:      if ( MZONE.lt.0 ) go to 700
180:      if ( JVMNT.ne.0 ) call VMNTR1( 4, NNXT(MZONE) )
181: C
182: C-----
183: C .... GATHER VECTORS .....
184: C-----
185: C
186:      II = 0
187:      do 100 I = 1, NNXT(NZONE+1)
188:          if ( IZNXT(I).eq.MZONE ) then
189:              II = II + 1
190:              IBP(II) = LSSRC(I)
191:          end if
192:      100 continue
193:      do 110 I = 1, NNXT(MZONE)
194:          X(I) = XXX(IBP(I)) + DEPS*AAA(IBP(I))
195:          Y(I) = YYY(IBP(I)) + DEPS*BBB(IBP(I))

```

```

196:          Z(I) = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
197:          W(I) = WWW(IBP(I))
198:      110 continue
199: C
200: C-----
201: C .... SEARCH ZONE # KKZ+1 TO MMZ.
202: C-----
203: C
204:      if ( JLATT.ne.0 ) then
205:          ICEL = KCELL(MZONE)
206:          MMZ = IPCEL(ICEL+1) - 1
207:          if ( ICEL.gt.0 ) then
208:              KKZ = IPCEL(ICEL) - 1
209:          else
210:              KKZ = 0
211:          end if
212:      else
213:          ICEL = 0
214:          KKZ = 0
215:          MMZ = NZONE
216:      end if
217:      MMX = MIN(MMZ-KKZ,NMEMO)
218: C
219: C .... IMEMO = 1 MEANS THAT ALL ZONES IN KMEMO ARRAY HAS NOT BEEN
220: C SEARCHED.
221:      IMEMO = 1
222: C
223: C =====
224: C =====
225: C .... BEGINE NEXT-ZONE-SEARCH .....
226: C =====
227: C =====
228: C
229:      INX = NNXT(MZONE)
230:      INFL = NFFL(NZ1)
231:      IRCF = 0
232:      ILAT = 0
233:      if ( JREFL.ne.0 ) IRCF = NBREF(NR1)
234:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
235:      do 300 M = 1, MMX
236: C
237: C .... RETURN HERE IF KMEMO(MZONE,M)=0 AND THERE ARE NO PARTICLES WHICH
238: C BELONGS TO ZONE 'KZ'. DETERMINE NEXT 'KZ' WITHOUT INCREMENTING
239: C LOOP COUNTER 'M'.
240: C
241:      120 if ( INX.eq.0 ) go to 310
242: C
243: C-----
244: C .... DETERMINE ZONE TO BE CHECKED (KZ) .....
245: C-----
246: C
247:      if ( KMEMO(MZONE,M).ne.0 ) then
248:          KZ = KMEMO(MZONE,M)
249:      else
250:          IMEMO = 0
251:      130 KKZ = KKZ + 1
252:          if ( KKZ.eq.MZONE ) go to 130
253:          do 140 MM = 1, MMX
254:              if ( KKZ.eq.KMEMO(MZONE,MM) ) go to 130
255:      140 continue
256:          KZ = KKZ
257:          if ( KZ.gt.MMZ ) go to 310
258:      end if
259:      MAT = KZMAT(KZ)
260: CCCC JRR = JREFL.ne.0.and.MAT.le. -1001

```

src/gmvp/seaone.f

```

261:      JRR      = JREFL.ne.0.and.MAT.le. - 1001.and..not.ISLATT(MAT)
262: C
263: C-----
264: C .... CHECK FOR EACH PARTICLE. IFI(I) = .FALSE./TRUE. = OUT/IN
265: C-----
266: C
267:      call JUDGE( 'SEAONE', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
268:      &          NSDA, NZDA, NZ1, JVMNT, IFL, JSIMP, JRR, KSREF, ISRF,
269:      &          FXYZ )
270: C
271: C-----
272: C ... COMPRESS PARTICLE DATA & UPDATE FLIGHT STACK .....
273: C-----
274: C
275:      KK      = 0
276:      do 150 I = 1, INX
277:      if ( IFI(I) ) KK      = KK + 1
278: 150      continue
279:      if ( KK.ne.0 ) then
280:      II      = 0
281:      IFFL    = 0
282:      IDEAD   = NDEAD
283: C
284: C-----
285: C ..... SEND PARTICLES TO FLIGHT STACK .....
286: C-----
287: C
288:      if ( MAT.ge.0 ) then
289:      *VOCL LOOP,NOVREC
290:      do 160 I = 1, INX
291:      if ( IFI(I) ) then
292:      IFFL    = IFFL + 1
293:      LSFFL(INFL+IFFL) = IBP(I)
294:      end if
295: 160      continue
296: C
297: C      ... check marker region ...
298: C
299:      if ( NMKREG.ne.0 ) then
300:      IRRG    = KZREG(MZONE)
301:      IRRG2   = KZREG(KZ)
302:      if ( JTLLT.eq.0.and.MKREG(IRRG,1).ne.0
303:      &      .and.IRRG2.ne.IRRG ) then
304:      CC      = KIBNK(9) + MKREG(IRRG,1) - 1
305:      CC*VOCL LOOP,NOVREC
306:      do 170 I = 1, IFFL
307:      IP      = LSFFL(INFL+I)
308:      IBNK(IP,KG) = IBNK(IP,KG) + 1
309:      CC170    continue
310:      else if ( JTLLT.ne.0 ) then
311:      CC*VOCL LOOP,NOVREC
312:      do 180 I = 1, IFFL
313:      IP      = LSFFL(INFL+I)
314:      IRRG    = IBREG(IP)
315:      C
316:      ILV     = LEVL(IP)
317:      IRRG2   = KZREG(KZ)
318:      if ( ILV.gt.0 ) then
319:      IRRG2   = ISUSP(KZREG(KZ),IBSPC(IP,ILV))
320:      end if
321:      if ( MKREG(IRRG,1).ne.0.and.IRRG.ne.IRRG2 ) then
322:      CC      = KIBNK(9) + MKREG(IRRG,1) - 1
323:      IBNK(IP,KG) = IBNK(IP,KG) + 1
324:      end if
325:      CC180    continue

```

```

326: CC      end if
327: CC      end if
328: C
329: C      .... marker region procedure is modified in Jan 2000,
330: C      to treat marker-region in lattice geometry.
331: C
332: C      ... check marker region : set previous region # in IWK ...
333: C
334:      if ( NMKREG.ne.0 ) then
335:      *VOCL LOOP,NOVREC
336:      do 170 I = 1, IFFL
337:      IP      = LSFFL(INFL+I)
338:      IWK(I)  = IBREG(IP)
339:      C
340:      ... this is the first flight after birth as source
341:      particle, so particle has no "previous region".
342:      C
343:      if ( IBNK(IP,KIBNK(5)).lt.0 ) then
344:      IWK(I)  = 0
345:      end if
346: 170      continue
347:      end if
348: C
349: C      .... update region number for universe dependent tally ...
350: C
351:      if ( JTLLT.ne.0 ) then
352:      *VOCL LOOP,NOVREC
353:      do 180 I = 1, IFFL
354:      IP      = LSFFL(INFL+I)
355:      ILV     = LEVL(IP)
356:      if ( ILV.gt.0 ) then
357:      IBREG(IP) = ISUSP(KZREG(KZ),IBSPC(IP,ILV))
358:      else
359:      IBREG(IP) = KZREG(KZ)
360:      end if
361: 180      continue
362:      else
363:      C
364:      ... set IBREG anytime (from Jan 2000)
365:      C
366:      *VOCL LOOP,NOVREC
367:      do 190 I = 1, IFFL
368:      IP      = LSFFL(INFL+I)
369:      IBREG(IP) = KZREG(KZ)
370: 190      continue
371:      end if
372: C
373: C      ... check marker region escape : IWK is previous value of IBREG ...
374: C
375:      if ( NMKREG.ne.0 ) then
376:      *VOCL LOOP,NOVREC
377:      do 200 I = 1, IFFL
378:      IP      = LSFFL(INFL+I)
379:      IRRG    = IWK(I)
380:      IRRG2   = IBREG(IP)
381:      if ( IRRG.gt.0.and.MKREG(IRRG,1).ne.0
382:      &      .and.IRRG.ne.IRRG2 ) then
383:      CC      = KIBNK(9) + MKREG(IRRG,1) - 1
384:      IBNK(IP,KG) = IBNK(IP,KG) + 1
385:      end if
386: 200      continue
387:      end if
388: C
389:      do 210 I = 1, IFFL
390:      IZFFL(INFL+I) = KZ

```

src/gmvp/seaone.f

```

391:      210      continue
392: C
393: C      ... set region# and/or zone# on birth here !!!
394: C
395:      if ( JLATT.ne.0.and.KIBNK(5).ne.0
396:      &      .and.(KIBNK(2).ne.0.or.KIBNK(3).ne.0) ) then
397:      NMNM      = 0
398: *VOCL LOOP,NOVREC
399:      do 220 I = 1, IFFL
400:      IP      = LSFFL(INFL+I)
401:      if ( IBNK(IP,KIBNK(5)).lt.0 ) then
402:      NMNM      = NMNM + 1
403:      end if
404:      220      continue
405:
406:      if ( NMNM.gt.0.and.KIBNK(2).ne.0 ) then
407: *VOCL LOOP,NOVREC
408:      do 230 I = 1, IFFL
409:      IP      = LSFFL(INFL+I)
410:      if ( IBNK(IP,KIBNK(5)).lt.0 ) then
411:      if ( JTLT.ne.0 ) then
412:      IBNK(IP,KIBNK(2)) = IBREG(IP)
413:      else
414:      IBNK(IP,KIBNK(2)) = KZREG(KZ)
415:      end if
416:      end if
417:      230      continue
418:      end if
419:
420:      if ( NMNM.gt.0.and.KIBNK(3).ne.0 ) then
421: *VOCL LOOP,NOVREC
422:      do 240 I = 1, IFFL
423:      IP      = LSFFL(INFL+I)
424:      if ( IBNK(IP,KIBNK(5)).lt.0 ) then
425:      IBNK(IP,KIBNK(3)) = KZ
426:      end if
427:      240      continue
428:      end if
429:      end if
430: C
431:      NFFL(KZ) = NFFL(KZ) + IFFL
432:      INFL      = INFL + IFFL
433: C
434: C-----
435: C      ..... LEAKAGE .....
436: C-----
437: C
438:      else if ( MAT.eq.-1000 ) then
439: *VOCL LOOP,NOVREC
440:      do 250 I = 1, INX
441:      if ( IFI(I) ) then
442:      IDEAD = IDEAD + 1
443:      LSDED(IDEAD) = IBP(I)
444:      WCNTR(7) = WCNTR(7) + W(I)
445:      end if
446:      250      continue
447:      if ( IDEAD.gt.NDEAD ) then
448:      NCNTR(7) = NCNTR(7) + IDEAD - NDEAD
449:      NCNTR(17) = NCNTR(17) + IDEAD - NDEAD
450:      NDEAD = IDEAD
451:      end if
452: C
453: C-----
454: C      ..... periodic boundary .....
455: C-----

```

```

456: C
457:      else if ( MAT.eq.-4000 ) then
458: *VOCL LOOP,NOVREC
459:      do 260 I = 1, INX
460:      if ( IFI(I) ) then
461:      IRCF = IRCF + 1
462:      LSREF(IRCF) = IBP(I)
463:      ISREF(IRCF) = ISRF(I)
464:      IZREF(IRCF) = KZ
465:      end if
466:      260      continue
467: C
468: C-----
469: C      ..... REFLECTION .....
470: C-----
471: C
472:      CCCC      else if ( MAT.lt.-1000 ) then
473:      else if ( MAT.lt.-1000.and..not.ISLATT(MAT) ) then
474: *VOCL LOOP,NOVREC
475:      do 270 I = 1, INX
476:      if ( IFI(I) ) then
477:      IRCF = IRCF + 1
478:      LSREF(IRCF) = IBP(I)
479:      ISREF(IRCF) = ISRF(I)
480:      IZREF(IRCF) = MZONE
481:      CCCCCCCCCCCCCCCCCC      WCNTR(18) = WCNTR(18) + W(I)
482:      end if
483:      270      continue
484: C
485: C-----
486: C      ..... ESCAPING LATTICE OR CELL .....
487: C-----
488: C
489:      CCCCC      else if ( MAT.le.-1.and.MAT.ge.-999 ) then
490:      else if ( ISLATT(MAT) .or. MAT.eq.-999 ) then
491: *VOCL LOOP,NOVREC
492:      do 280 I = 1, INX
493:      if ( IFI(I) ) then
494:      ILAT = ILAT + 1
495:      LSLAT(ILAT) = IBP(I)
496:      IZLAT(ILAT) = MLBZZ(KZ)
497:      IZZ(IBP(I)) = KZ
498:      end if
499:      280      continue
500:      NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + KK
501:      end if
502: C
503: C-----
504: C      ..... COMPRESS POINTERS & TEMPORALY ARRAYS .....
505: C-----
506: C
507: *VOCL LOOP,NOVREC
508:      do 290 I = 1, INX
509:      if ( .not.IFI(I) ) then
510:      II = II + 1
511:      IBP(II) = IBP(I)
512:      X(II) = X(I)
513:      Y(II) = Y(I)
514:      Z(II) = Z(I)
515:      W(II) = W(I)
516:      end if
517:      290      continue
518:      INX = II
519: C
520: C-----

```


src/gmvp/seaone.f

```

521: C      .... SETTING OF KMEMO ARRAY .....
522: C-----
523: C
524:         if ( .not.JMEM ) then
525:             if ( IMEMO.eq.0 ) KMEMO(MZONE,M) = KZ
526:         else
527:             if ( IMEMO.eq.0 ) then
528:                 KMEMO(MZONE,M) = KZ
529:                 MEMC(MZONE,M) = MEMC(MZONE,M) + KK
530:                 MEMZ(MZONE) = M
531:             else if ( M.le.NMEMO ) then
532:                 MEMC(MZONE,M) = MEMC(MZONE,M) + KK
533:             end if
534:         end if
535: C
536:         else if ( IMEMO.eq.0 ) then
537:             go to 120
538:         end if
539: 300 continue
540: C
541: C-----
542: C .... UPDATE OF ZONE # IN BANK
543: C-----
544: C
545: 310 do 320 I = NFFL(NZ1) + 1, INFL
546:     IZZ(LSFFL(I)) = IZFFL(I)
547: 320 continue
548:     NCNTR(17) = NCNTR(17) + INFL - NFFL(NZ1)
549: C
550: C-----
551: C .... UPDATE OF REFLECTION STACK
552: C-----
553: C
554:         if ( JREFL.ne.0.and.IRCF.gt.NBREF(NR1) ) then
555:             do 330 I = NBREF(NR1) + 1, IRCF
556:                 NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
557: C980828         IZREF(I) = MZONE
558:             330 continue
559: CCCCCCCC NCNTR(18) = NCNTR(18) + IRCF - NBREF(NR1)
560:             NBREF(NR1) = IRCF
561:         end if
562: C
563: C-----
564: C .... COMPRESS SEARCH STACK .....
565: C-----
566: C
567:         II = 0
568: *VOCL LOOP,NOVREC
569:         do 340 I = 1, NNXT(NZ1)
570:             if ( IZNXT(I).ne.MZONE ) then
571:                 II = II + 1
572:                 LSSRC(II) = LSSRC(I)
573:                 IZNXT(II) = IZNXT(I)
574:             end if
575: 340 continue
576: C
577:         NDEAD1 = NDEAD
578: C
579: C=====
580: C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, I TREAT THEM
581: C AS DEFERRED PARTICLES OR LOST PARTICLES.
582: C=====
583: C
584:         if ( INX.ne.0 ) then
585: C-----

```

```

586: C      .... DEFERRED PARTICLES .....
587: C-----
588: C
589:         if ( IMEMO.eq.1.or.IMEMO.eq.0.and.KKZ.le.MMZ ) then
590: CM      &         .AND.MMX.LT.(MMZ-KKZ)) THEN
591: C
592: C
593: C      .... WARNING FOR DEFERRED PARTICLES ....
594: C
595:             MDEFR = MDEFR + 1
596:             if ( MDEFR.lt.MWARN ) then
597: C/#IF PARA(SX* CRAY)
598: *             call MVPSYNC_LOCK(2)
599: C/#ENDIF
600:                 write(IOW,7000) INX, MZONE
601: 7000             format(' == ',I5,' PARTICLES COULD NOT FIND ',
602: &                     'NEXT-ENTERING-ZONE FROM MEMORIZED DATA (KMEMO) '
603: &                     /' IN ZONE ',I5)
604: C/#IF PARA(SX* CRAY)
605: *             call MVPSYNC_UNLOCK(2)
606: C/#ENDIF
607:             end if
608: C
609:             do 350 I = 1, INX
610:                 LSSRC(II+I) = IBP(I)
611:                 IZNXT(II+I) = -MZONE
612: 350             continue
613:                 IDEFER = IDEFER + INX
614:                 NNXT(NZ1) = NNXT(NZ1) + INX
615: C
616: C-----
617: C      .... LOST PARTICLES OR TANGENTIAL PARTICLES
618: C-----
619:         else
620: C
621: C      .... IN LATTICE ZONE ... ---> LOST
622: C
623:             if ( KZMAT(MZONE).lt.0 ) then
624:                 do 360 I = 1, INX
625:                     IFI(I) = .false.
626:                     LSDED(NDEAD+I) = IBP(I)
627: 360                 continue
628:                     IDEAD = INX
629: C
630: C      .... CHECK FOR PREVIOUS ZONE. IFI(I) = .FALSE./.TRUE. = OUT/IN
631: C
632:             else
633:                 JRR = .false.
634:                 call JUDGE( 'SEAONE', MZONE, INX, X, Y, Z, IFI, SDA,
635: &                         KZDA, KZAA, NSDA, NZDA, NZ1, JVMNT, IFL, JSIMP,
636: &                         JRR, KSREF, ISRF, FXYZ )
637:                 IDEAD = 0
638:                 IFFL = 0
639: C
640:                 do 370 I = 1, INX
641: C
642: C
643: C      .... TANGENTIAL .....
644: C
645: C
646:             if ( IFI(I) ) then
647:                 IFFL = IFFL + 1
648:                 LSFFL(INFL+IFFL) = IBP(I)
649:                 IZFFL(INFL+IFFL) = MZONE
650:                 XXX(IBP(I)) = X(I)

```

src/gmvp/seaone.f

```

651:      YYY(IBP(I)) = Y(I)
652:      ZZZ(IBP(I)) = Z(I)
653: C3/11/1991 ....
654:      KLSF(IBP(I)) = 0
655: C
656: C
657: C ..... LOST .....
658: C
659: C
660:      else
661:      IDEAD = IDEAD + 1
662:      LSDED(NDEAD+IDEAD) = IBP(I)
663:      end if
664: 370 continue
665: C
666: C ... WARNING FOR TANGENTIAL PARTICLES ....
667: C
668:      MTANG = MTANG + 1
669:      if ( MTANG.lt.MWARN ) then
670:      IP1 = LSFFL(INFL+1)
671: C/#IF PARA(SX* CRAY)
672: * call MVPSYNC_LOCK(2)
673: C/#ENDIF
674:      write(IOW,7020) IFFL, MZONE, XXX(IP1), YYY(IP1),
675: & ZZZ(IP1), AAA(IP1), BBB(IP1), CCC(IP1)
676: C/#IF PARA(SX* CRAY)
677: * call MVPSYNC_UNLOCK(2)
678: C/#ENDIF
679: C
680: 7020 format(' == ',I5,' PARTICLES MAY BE TANGENTIAL',
681: & ' TO THE BOUNDARY OF ZONE ',I5,'./
682: & ' (OTHERWISE JUST ON-EDGE OR AT-CORNER ) /
683: & ' ONE OF THEM HAS THE FOLLOWING POSITION &',
684: & ' DIRECTION;/' POSITION: ',3E12.5,
685: & ' DIRECTION: ',3E12.5)
686:      end if
687: C
688:      NFFL(MZONE) = NFFL(MZONE) + IFFL
689:      INFL = INFL + IFFL
690:      end if
691: C
692:      NDEAD = NDEAD + IDEAD
693:      NLOST = NLOST + IDEAD
694: C
695: C ..... message for lost particles ...
696: C
697:      if ( IDEAD.gt.0 ) then
698: C/#IF PARA(SX* CRAY)
699: * call MVPSYNC_LOCK(2)
700: C/#ENDIF
701:      write(IOW,*) ' !! (SEAONE) ', IDEAD,
702: & ' PARTICLES ARE LOST !! ZONE # = ', MZONE
703:      do 390 I = 1, INX
704:      if ( .not.IFI(I) ) then
705:      write(IOW,7040) IBP(I), XXX(IBP(I)), YYY(IBP(I)),
706: & ZZZ(IBP(I)), AAA(IBP(I)), BBB(IBP(I)),
707: & CCC(IBP(I))
708:      if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 )
709: & write(IOW,7060)
710: & (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,LEVL(IBP(I))
711: & )
712:      if ( JTLT.ne.0 ) then
713:      do 380 L = 1, LEVL(IBP(I))
714:      call PRNSPC( IOW, A, CHA, IBSPC(IBP(I),L), L
715: & )

```

```

716: 380 continue
717:      end if
718:      end if
719: 390 continue
720: C
721: C/#IF PARA(SX* CRAY)
722: * call MVPSYNC_UNLOCK(2)
723: C/#ENDIF
724:      end if
725: 7040 format(1X,I5,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',
726: & E12.5,' ETA=',E12.5,' XI=',E12.5)
727: 7060 format(1X,' LEVL = ',I3,' LZZ = ',I5,' LPOS = ',I5)
728:      end if
729:      end if
730: C
731: C
732: C-----
733: C ... TAKE special tallies .....
734: C-----
735: C
736: C
737:      if ( NTEVE.gt.0.and.JTEVE(7).gt.0.and.NDEAD1-NDEAD0.gt.0 ) then
738:      ISAFE = NDEAD1 - NDEAD0
739: C
740:      do 460 J = 0, JTEVE(7) - 1
741: C
742: C ... pointer to direct tally (idt) & d-tally # (it) ...
743:      IDT = KTEVE(LTEVE(7)+J) - 1
744:      IT = IDTAL(IDT+9)
745: C
746: C .. neutron or photon ?
747: C
748:      NLEK = 0
749:      if ( IPAR.eq.0 ) then
750:      if ( IDTAL(IDT+5).eq.1 ) then
751:      do 400 I = 1, ISAFE
752:      IBPIII = LSDED(NDEAD0+I)
753:      if ( IGG(IBPIII).le.NGPI ) then
754:      NLEK = NLEK + 1
755:      IBP(NLEK) = IBPIII
756:      end if
757: 400 continue
758:      else if ( IDTAL(IDT+5).eq.2 ) then
759:      do 410 I = 1, ISAFE
760:      IBPIII = LSDED(NDEAD0+I)
761:      if ( IGG(IBPIII).gt.NGPI ) then
762:      NLEK = NLEK + 1
763:      IBP(NLEK) = IBPIII
764:      end if
765: 410 continue
766:      end if
767:      else if ( IPAR.eq.IDTAL(IDT+5) ) then
768:      do 420 I = 1, ISAFE
769:      IBP(I) = LSDED(NDEAD0+I)
770: 420 continue
771:      NLEK = ISAFE
772:      end if
773: C
774:      do 430 I = 1, NLEK
775:      IWK(I) = IGG(IBP(I))
776:      W(I) = WWW(IBP(I))
777:      if ( JTLT.eq.0 ) then
778:      IZI(I) = KZREG(IZZ(IBP(I)))
779:      else
780:      IZI(I) = IBREG(IBP(I))

```

src/gmvp/seaone.f

```

781:          end if
782:          if ( JTIME.gt.0 ) ISRF(I) = ITT(IBP(I))
783: 430      continue
784: C
785: C      ... skip if current region does not need this tally ...
786: C
787:          do 440 I = 1, NLEK
788:              if ( JDTRG(IZI(I),IT).ne.0 ) go to 450
789: 440      continue
790:          go to 460
791: 450      continue
792: C
793:          if ( NLEK.gt.0 ) then
794:              call STALN7( IOW, IDTAL(IDT+1), W, IBP, NLEK, IWK, IZI,
795:              &          ISRF, NGROUP, NGP1, NREG, NTIME, NBANK, DTALY,
796:              &          DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, X, Y )
797:          end if
798: 460      continue
799:          end if
800: C
801: C =====
802: C =====
803: C =====
804: C SPLITTING AND/OR RUSSIAN ROULETTE
805: C =====
806: C =====
807: C =====
808: C
809:          if ( (JIMPT.ne.0.or.JWWND.ne.0).and.INFL.gt.NFFL(NZ1) ) then
810:
811:              call RANU2( IRAND, R, INFL, ICON )
812:
813:              if ( JIMPT.ne.0 ) then
814:
815: *VOCL LOOP,NOVREC
816:              do 470 I = NFFL(NZ1) + 1, INFL
817:                  W(I) = WWW(LSFFL(I))
818:                  IGT = IGG(LSFFL(I))
819:                  if ( JTLLT.ne.0 ) then
820:                      NRG = IBREG(LSFFL(I))
821:                  else
822:                      NRG = KZREG(IZFFL(I))
823:                  end if
824: C
825:                  XIMPN = XIMP(IGT,NRG)
826:                  XLS = XIMPN/XIM(LSFFL(I))
827:                  XIM(LSFFL(I)) = XIMPN
828:                  IWK(I) = XLS + R(I)
829:                  FXYZ(I) = W(I) /XLS
830: 470      continue
831:
832:          else if ( JWWND.ne.0 ) then
833:
834:              if ( JWTIM.eq.0.and.JRWVR.eq.0 ) then
835: *VOCL LOOP,NOVREC
836:              do 480 I = NFFL(NZ1) + 1, INFL
837:                  W(I) = WWW(LSFFL(I))
838:                  IGT = IGG(LSFFL(I))
839:                  if ( JTLLT.ne.0 ) then
840:                      NRG = IBREG(LSFFL(I))
841:                  else
842:                      NRG = KZREG(IZFFL(I))
843:                  end if
844: C
845:                  IWK(I) = 1

```

```

846:                  FXYZ(I) = W(I)
847:                  WS = WSRV(IGT,NRG)
848:                  WK = WKIL(IGT,NRG)
849:                  if ( W(I).gt.2.0*WS .or. W(I).lt.WK ) then
850:                      IWK(I) = W(I) /WS + R(I)
851:                      FXYZ(I) = WS
852:                  end if
853: 480      continue
854:          else
855: *VOCL LOOP,NOVREC
856:          do 490 I = NFFL(NZ1) + 1, INFL
857:              W(I) = WWW(LSFFL(I))
858:              IGT = IGG(LSFFL(I))
859:              if ( JTLLT.ne.0 ) then
860:                  NRG = IBREG(LSFFL(I))
861:              else
862:                  NRG = KZREG(IZFFL(I))
863:              end if
864: C
865:              IWK(I) = 1
866:              FXYZ(I) = W(I)
867:              WS = WSRV(IGT,NRG)
868:              WK = WKIL(IGT,NRG)
869:              if ( JWTIM.ne.0 ) then
870:                  ITTII = ITT(LSFFL(I))
871:                  WS = WS*WTIME(ITTII)
872:                  WK = WK*WTIME(ITTII)
873:              end if
874:              if ( JRWVR.ne.0 ) then
875:                  WS = WS*DBNK(LSFFL(I),KDBNK(1))
876:                  WK = WK*DBNK(LSFFL(I),KDBNK(1))
877:              end if
878:              WS = MAX(WMIN,WS)
879:              WK = MAX(WMIN,WK)
880:              if ( W(I).gt.2.0*WS .or. W(I).lt.WK ) then
881:                  IWK(I) = W(I) /WS + R(I)
882:                  FXYZ(I) = WS
883:              end if
884: 490      continue
885:          end if
886:      end if
887: C-----
888: C ..... REMOVE KILLED PARTICLES FROM STACK & BANK ...
889: C-----
890: C
891: C          II = NFFL(NZ1)
892: *VOCL LOOP,SCALAR
893:          do 500 I = NFFL(NZ1) + 1, INFL
894:              if ( IWK(I).eq.0 ) NFFL(IZFFL(I)) = NFFL(IZFFL(I)) - 1
895: 500      continue
896: C*VOCL LOOP,NOVREC
897:          do 290 I = NFFL(NZ1) + 1, INFL
898: C
899: C          IWKI = IWK(I)
900: C          if ( IWKI.eq.0 ) then
901: C              WCNTR(9) = WCNTR(9) + W(I)
902: C              NDEAD = NDEAD + 1
903: C              LSDED(NDEAD) = LSFFL(I)
904: C          else
905: C              II = II + 1
906: C              IWK(II) = IWKI
907: C              FXYZ(II) = FXYZ(I)
908: C              W(II) = W(I)
909: C              LSFFL(II) = LSFFL(I)
910: C              IZFFL(II) = IZFFL(I)

```

src/gmvp/seaone.f

```

911: C          end if
912: C 290      continue
913: *VOCL LOOP,NOVREC
914:       do 510 I = NFFL(NZ1) + 1, INFL
915:         if ( IWK(I).eq.0 ) then
916:           WCNTR(9) = WCNTR(9) + W(I)
917:           NDEAD = NDEAD + 1
918:           LSDED(NDEAD) = LSFFL(I)
919:         end if
920:       510      continue
921: *VOCL LOOP,NOVREC
922:       do 520 I = NFFL(NZ1) + 1, INFL
923:         IWKI = IWK(I)
924:         if ( IWKI.ne.0 ) then
925:           II = II + 1
926:           IWK(II) = IWKI
927:           FXYZ(II) = FXYZ(I)
928:           W(II) = W(I)
929:           LSFFL(II) = LSFFL(I)
930:           IZFFL(II) = IZFFL(I)
931:         end if
932:       520      continue
933: C
934:       NCNTR(9) = NCNTR(9) + INFL - II
935:       INFL = II
936:       NSPLTT = 0
937:       do 530 I = NFFL(NZ1) + 1, INFL
938:         NSPLTT = NSPLTT + IWK(I)
939:       530      continue
940:       NSPLTT = NSPLTT - (INFL-NFFL(NZ1))
941: C
942: C-----
943: C .... INSUFFICIENT SPACE IN BANK TO ACCEPT ALL GENERATED PARTICLES.
944: C-----
945: C
946:       if ( NSPLTT.gt.NDEAD ) then
947:         II = 0
948:         do 540 KK = NFFL(NZ1) + 1, INFL
949:           II = II + IWK(KK) - 1
950:           if ( II.gt.NDEAD ) go to 550
951:       540      continue
952:       550      do 560 I = KK, INFL
953:         if ( W(I).lt.FXYZ(I) ) then
954:           NCNTR(10) = NCNTR(10) + 1
955:           WCNTR(10) = WCNTR(10) + W(I)
956:         else
957:           NCNTR(6) = NCNTR(6) + 1
958:           WCNTR(6) = WCNTR(6) + W(I)
959:           FXYZ(I) = W(I)
960:           IWK(I) = 1
961:         end if
962:       560      continue
963: CM       NCNTR(6) = NCNTR(6) + INFL - KK + 1
964:       end if
965: C
966: C ..... MAXIMUM SPLITTING NUMBER
967: C
968:       MX = 0
969:       do 570 I = NFFL(NZ1) + 1, INFL
970:         IWK(I) = IWK(I) - 1
971:         MX = MAX(MX,IWK(I))
972:       570      continue
973: C
974: C-----
975: C ..... NOW I CAN BEGIN SPLITTING !!

```

```

976: C-----
977: C
978:       IIM = 0
979:       do 580 I = NFFL(NZ1) + 1, INFL
980:         IIM = IIM + IWK(I)
981:         WWW(LSFFL(I)) = FXYZ(I)
982:       580      continue
983:       NCNTR(5) = NCNTR(5) + IIM
984: C
985: C ..... IZI(I) : bank pointers of particles generated by splitting.
986: C
987:       II = 0
988:       do 600 M = 1, MX
989:         do 590 I = NFFL(NZ1) + 1, INFL
990:           if ( IWK(I).ge.M ) then
991:             II = II + 1
992:             LSFFL(INFL+II) = LSDED(II)
993:             IZFFL(INFL+II) = IZFFL(I)
994:             IZI(II) = LSFFL(I)
995:             WWW(LSFFL(INFL+II)) = FXYZ(I)
996:           end if
997:         590      continue
998:       600      continue
999: C
1000:       if ( II.gt.0 ) then
1001: *VOCL LOOP,NOVREC
1002:       do 610 I = 1, II
1003:         K = LSFFL(INFL+I)
1004:         XXX(K) = XXX(IZI(I))
1005:         YYY(K) = YYY(IZI(I))
1006:         ZZZ(K) = ZZZ(IZI(I))
1007:         AAA(K) = AAA(IZI(I))
1008:         BBB(K) = BBB(IZI(I))
1009:         CCC(K) = CCC(IZI(I))
1010:         IZZ(K) = IZZ(IZI(I))
1011:         IGG(K) = IGG(IZI(I))
1012:         TTT(K) = TTT(IZI(I))
1013:         KLSF(K) = KLSF(IZI(I))
1014:         if ( JIMPT.ne.0 ) XIM(K) = XIM(IZI(I))
1015:         if ( JLATT.ne.0 ) LEVL(K) = LEVL(IZI(I))
1016: CCCCCC         if ( JTLLT.ne.0 ) IBREG(K) = IBREG(IZI(I))
1017:         IBREG(K) = IBREG(IZI(I))
1018:         if ( JTIME.ne.0 ) ITT(K) = ITT(IZI(I))
1019:       610      continue
1020: C
1021:       if ( JLATT.ne.0 ) then
1022:         do 630 NT = 1, NEST
1023: *VOCL LOOP,NOVREC
1024:         do 620 I = 1, II
1025:           K = LSFFL(INFL+I)
1026:           LZZ(K,NT) = LZZ(IZI(I),NT)
1027:           LPOS(K,NT) = LPOS(IZI(I),NT)
1028:           if ( JHLAT.ne.0 ) LCRS(K,NT) = LCRS(IZI(I),NT)
1029:           if ( JTLLT.ne.0 ) IBSPC(K,NT) = IBSPC(IZI(I),NT)
1030:       620      continue
1031:       630      continue
1032:         end if
1033: C
1034: C
1035: C .... optional bank parameters
1036: C
1037:       do 650 K = 1, KDBNK(0)
1038: *VOCL LOOP,NOVREC
1039:       do 640 I = 1, II
1040:         K1 = LSFFL(INFL+I)

```

src/gmvp/seaone.f

```

1041:          DBNK(K1,K) = DBNK(IZI(I),K)
1042:      640      continue
1043:      650      continue
1044:      do 670 K = 1, KIBNK(0)
1045: *VOCL LOOP,NOVREC
1046:          do 660 I = 1, II
1047:              K1 = LSFFL(INFL+I)
1048:              IBNK(K1,K) = IBNK(IZI(I),K)
1049:      660      continue
1050:      670      continue
1051: C
1052: *VOCL LOOP,NOVREC
1053:          do 680 I = II + 1, NDEAD
1054:              LSDED(I-II) = LSDED(I)
1055:          continue
1056:          NDEAD = NDEAD - II
1057: CM          DO 330 I=1,II
1058: CM330      WCNTR(10) = WCNTR(10) + WWW(LSFFL(INFL+I))
1059: CM          NCNTR(10) = NCNTR(10) + II
1060:          end if
1061: *VOCL LOOP,SCALAR
1062:          do 690 I = NFFL(NZ1) + 1, INFL
1063: CM340      NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + IWK(I) - 1
1064:          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + IWK(I)
1065:      690      continue
1066:          INFL = INFL + 1
1067:          end if
1068: C
1069: C-----
1070: C .... ADJUSTMENT OF NUMBER OF PARTICLE IN STACKS .....
1071: C-----
1072: C
1073:          NNXT(NZ1) = NNXT(NZ1) - NNXT(MZONE)
1074:          NNXT(MZONE) = 0
1075:          NFFL(NZ1) = INFL
1076:          if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
1077: C
1078:          return
1079: C
1080: C
1081: C
1082: C
1083: C =====
1084: C .... TREATMENT OF DEFERRED PARTICLE .....
1085: C =====
1086: C
1087: C
1088: C
1089:      700 II = 0
1090:          do 710 I = 1, NNXT(NZ1)
1091:              if ( IZNXT(I).lt.0 ) then
1092:                  II = II + 1
1093:                  IBP(II) = LSSRC(I)
1094:                  IZI(II) = -IZNXT(I)
1095:              end if
1096:          710 continue
1097: C
1098:          if ( II.ne.IDEFER ) then
1099:              write(IOW,*) ' !! (SEAONE) NUMBER OF DEFERRED PARTICLES IN ',
1100:              & ' SEARCH STACK IS DIFFERENT FROM IDEFER !! ', II,
1101:              & IDEFER
1102:              stop 666
1103:          end if
1104:          do 720 I = 1, IDEFER
1105:              X(I) = XXX(IBP(I)) + DEPS*AAA(IBP(I))

```

```

1106:          Y(I) = YYY(IBP(I)) + DEPS*BBB(IBP(I))
1107:          Z(I) = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
1108:          W(I) = WWW(IBP(I))
1109:      720 continue
1110: C
1111: C-----
1112: C .... CHECK ALL ZONES .....
1113: C-----
1114:          INX = IDEFER
1115:          IDEAD = NDEAD
1116:          IRCF = 0
1117:          ILAT = 0
1118:          if ( JREFL.ne.0 ) IRCF = NBREF(NR1)
1119:          if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
1120:          KKZ = 0
1121:          MMZ = NZONE
1122: C
1123: C
1124: C
1125:          ITANG = 0
1126: C
1127:          do 790 KZ = 1, NZONE
1128:              if ( INX.eq.0 ) go to 800
1129:              JRR = JREFL.ne.0.and.KZMAT(KZ) .le. - 1001
1130:              JRR = JREFL.ne.0.and.KZMAT(KZ) .le. - 1001
1131:              & .and..not.ISLATT(KZMAT(KZ))
1132: C
1133:              call JUDGE( 'SEARCH', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
1134:              & NSDA, NZDA, NZ1, JVMNT, IFL, JSIMP, JRR, KSREF, ISRF,
1135:              & FXYZ )
1136: CM011691 IFFL = 0
1137:          II = 0
1138:          if ( JLATT.ne.0 ) then
1139:              ICEL = KCELL(KZ)
1140:              MMZ = IPCEL(ICEL+1) - 1
1141:              if ( ICEL.gt.0 ) then
1142:                  KKZ = IPCEL(ICEL) - 1
1143:              else
1144:                  KKZ = 0
1145:              end if
1146:          end if
1147: C
1148:          if ( KZMAT(KZ).lt.0 ) then
1149: *VOCL LOOP,NOVREC
1150:              do 730 I = 1, INX
1151:                  IBPIII = IBP(I)
1152:                  if ( (.not.IFI(I)) .or. IZI(I).eq.KZ .or. IZI(I).le.KKZ
1153:                  & .or. IZI(I).gt.MMZ ) then
1154:                      II = II + 1
1155:                      IBP(II) = IBPIII
1156:                      X(II) = X(I)
1157:                      Y(II) = Y(I)
1158:                      Z(II) = Z(I)
1159:                      W(II) = W(I)
1160:                      IZI(II) = IZI(I)
1161:                  else
1162: C-----
1163: C .... LEAKAGE .....
1164: C-----
1165:                      if ( KZMAT(KZ).eq.-1000 ) then
1166:                          NDEAD = NDEAD + 1
1167:                          LSDED(NDEAD) = IBPIII
1168:                          WCNTR(7) = WCNTR(7) + W(I)
1169: C-----
1170: C .... REFLECTION .....

```

src/gmvp/seaone.f

```

1171: C-----
1172:           else if ( KZMAT(KZ).eq.-4000 ) then
1173:             IRCF = IRCF + 1
1174:             LSREF(IRCFC) = IBPIII
1175:             ISREF(IRCFC) = ISRF(I)
1176:             IZREF(IRCFC) = KZ
1177: CCCCC           else if ( KZMAT(KZ).lt.-1000 ) then
1178:           else if ( KZMAT(KZ).lt.-1000
1179:             &           .and..not.ISLATT(KZMAT(KZ)) ) then
1180:             IRCF = IRCF + 1
1181:             LSREF(IRCFC) = IBPIII
1182:             ISREF(IRCFC) = ISRF(I)
1183:             IZREF(IRCFC) = IZZ(IBPIII)
1184: CCCCCCCCCCCCCC WCNTR(18) = WCNTR(18) + W(I)
1185: C-----
1186: C      .... LATTICE-SEARCH .....
1187: C-----
1188: CCCC           else if ( KZMAT(KZ).le.-1.and.KZMAT(KZ).ge.-999 ) then
1189:           else if ( ISLATT(KZMAT(KZ)) .or. KZMAT(KZ).eq.-999 )
1190:             &           then
1191:             ILAT = ILAT + 1
1192:             LSLAT(ILAT) = IBPIII
1193:             IZLAT(ILAT) = MLBZZ(KZ)
1194: C980828 CCCCCCCCCC NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + 1
1195:           end if
1196:         end if
1197:       730       continue
1198: C
1199:       if ( JLATT.ne.0.and.ILAT.gt.NXLT(NLBZ+1) ) then
1200:         NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + ILAT - NXLT(NLBZ+1)
1201:         NXLT(NLBZ+1) = ILAT
1202:       end if
1203: C
1204: C      .... KZMAT(KZ) > 0 ....
1205: C
1206:       else
1207: C011691 ...
1208:       IFFL = 0
1209: *VOCL LOOP,NOVREC
1210:       do 740 I = 1, INX
1211:         IBPIII = IBP(I)
1212:         IZIIII = IZI(I)
1213:         if ( (.not.IFI(I)) .or. IZIIII.le.KKZ .or. IZIIII.gt.MMZ
1214:           &           ) then
1215:           II = II + 1
1216:           IBP(II) = IBPIII
1217:           X(II) = X(I)
1218:           Y(II) = Y(I)
1219:           Z(II) = Z(I)
1220:           W(II) = W(I)
1221:           IZI(II) = IZIIII
1222:         else
1223: C-----
1224: C      .... FREE-FLIGHT .....
1225: C-----
1226: C-----
1227: C
1228:       IFFL = IFFL + 1
1229:       LSFFL(NFFL(NZ1)+IFFL) = IBPIII
1230:       IZZ(IBPIII) = KZ
1231: C
1232: C      .... check marker region ...
1233: C
1234: C      if ( NMKREG.ne.0 ) then
1235: C        IRRG = KZREG(IZIIII)

```

```

1236: C
1237: C      if ( JTLLT.ne.0 ) then
1238: C        IRRG = IBREG(IBPIII)
1239: C      end if
1240: C
1241: C      IRRG2 = KZREG(KZ)
1242: C      if ( JTLLT.ne.0 ) then
1243: C        ILV = LEVL(IBPIII)
1244: C        if ( ILV.gt.0 ) then
1245: C          IRRG2 = ISUSP(KZREG(KZ),IBSPC(IBPIII,ILV))
1246: C        end if
1247: C      end if
1248: C      if ( MKREG(IRR,1).ne.0.and.IRRG.ne.IRRG2 ) then
1249: C        KG = KIBNK(9) + MKREG(IRR,1) - 1
1250: C        IBNK(IBPIII,KG) = IBNK(IBPIII,KG) + 1
1251: C      end if
1252: C      end if
1253: C
1254: C      .... marker region procedure is modified in Jan 2000,
1255: C      to treat marker-region in lattice geometry.
1256: C
1257: C      ... check previous region for MARKER-REGION flag
1258: C
1259: C      if ( NMKREG.ne.0 ) then
1260: C        ... IBREG is set always (from Jan 2000)
1261: C        IRRG = IBREG(IBPIII)
1262: C      end if
1263: C      ... this is the first flight after birth as source
1264: C      particle, so particle has no "previous region".
1265: C
1266: C      if ( IBNK(IBPIII,KIBNK(5)).lt.0 ) then
1267: C        IRRG = 0
1268: C      end if
1269: C      end if
1270: C
1271: C      .... SET REGION # FOR UNIVERSE-DEPENDENT TALLY MODE ...
1272: C
1273: C      ... IBREG is set anytime (from Jan 2000)
1274: C      IBREG(IBPIII) = KZREG(KZ)
1275: C      if ( JTLLT.ne.0 ) then
1276: C        ILV = LEVL(IBPIII)
1277: C        if ( ILV.gt.0.and.KZREG(KZ).gt.0 ) then
1278: C          IBREG(IBPIII) =
1279: C            &           ISUSP(KZREG(KZ),IBSPC(IBPIII,ILV))
1280: C        end if
1281: C      end if
1282: C
1283: C      .... check marker region escape ...
1284: C
1285: C      if ( NMKREG.ne.0.and.IRRG.ne.0 ) then
1286: C        IRRG2 = IBREG(IBPIII)
1287: C        if ( MKREG(IRR,1).ne.0.and.IRRG.ne.IRRG2 ) then
1288: C          KG = KIBNK(9) + MKREG(IRR,1) - 1
1289: C          IBNK(IBPIII,KG) = IBNK(IBPIII,KG) + 1
1290: C        end if
1291: C      end if
1292: C
1293: C-----
1294: C      .... TANGENTIAL PARTICLES
1295: C-----
1296: C
1297: C      if ( IZIIII.eq.KZ ) then
1298: C        ITANG = ITANG + 1
1299: C        XXX(IBPIII) = X(I)
1300: C        YYY(IBPIII) = Y(I)

```

src/gmvp/seaone.f

```

1301:          ZZZ(IBPIII) = Z(I)
1302:        end if
1303:      end if
1304: 740      continue
1305: C
1306:      if ( IFFL.gt.0 ) then
1307:        do 750 I = 1, IFFL
1308:          IZFFL(NFFL(NZ1)+I) = KZ
1309: 750      continue
1310: C
1311: C      ... set region# and/or zone# on birth here !!!
1312: C
1313:      if ( JLATT.ne.0.and.KIBNK(5).ne.0
1314:      & .and.(KIBNK(2).ne.0.or.KIBNK(3).ne.0) ) then
1315:        MNM = 0
1316: *VOCL LOOP,NOVREC
1317:      do 760 I = 1, IFFL
1318:        IP = LSFFL(NFFL(NZONE+1)+I)
1319:        if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1320:          MNM = MNM + 1
1321:        end if
1322: 760      continue
1323: C
1324:      if ( MNM.gt.0.and.KIBNK(2).ne.0 ) then
1325: *VOCL LOOP,NOVREC
1326:      do 770 I = 1, IFFL
1327:        IP = LSFFL(NFFL(NZONE+1)+I)
1328:        if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1329:          if ( JTLLT.ne.0 ) then
1330:            IBNK(IP,KIBNK(2)) = IBREG(IP)
1331:          else
1332:            IBNK(IP,KIBNK(2)) = KZREG(KZ)
1333:          end if
1334:        end if
1335: 770      continue
1336:      end if
1337: C
1338:      if ( MNM.gt.0.and.KIBNK(3).ne.0 ) then
1339: *VOCL LOOP,NOVREC
1340:      do 780 I = 1, IFFL
1341:        IP = LSFFL(NFFL(NZONE+1)+I)
1342:        if ( IBNK(IP,KIBNK(5)).lt.0 ) then
1343:          IBNK(IP,KIBNK(3)) = KZ
1344:        end if
1345: 780      continue
1346:      end if
1347:      end if
1348: C
1349:      NFFL(NZ1) = NFFL(NZ1) + IFFL
1350:      NFFL(KZ) = NFFL(KZ) + IFFL
1351:    end if
1352:  end if
1353: C011691 ... END
1354: C
1355:      INX = II
1356: C011691 IF(IFFL.GT.0) THEN
1357: C      DO 420 I=1,IFFL
1358: C 420      IZFFL(NFFL(NZ1)+I) = KZ
1359: C      NFFL(NZ1) = NFFL(NZ1) + IFFL
1360: C      NFFL(KZ) = NFFL(KZ) + IFFL
1361: C      ENDIF
1362: C
1363: 790 continue
1364: C
1365: C      .... WARNING FOR TANGENTIAL PARTICLES ....

```

```

1366: C
1367:      if ( ITANG.gt.0 ) then
1368:        MTANG = MTANG + 1
1369:        if ( MTANG.lt.MWARN ) then
1370: C/#IF PARA(SX* CRAY)
1371: *      call MVPSYNC_LOCK(2)
1372: C/#ENDIF
1373: C
1374:        write(IOW,7080) ITANG
1375: C
1376: C/#IF PARA(SX* CRAY)
1377: *      call MVPSYNC_UNLOCK(2)
1378: C/#ENDIF
1379: C
1380: 7080      format('/' == 'I5,' PARTICLES MAY BE TANGENTIAL',
1381:      &          ' TO THE BOUNDARIES OF ZONE. ',
1382:      &          ' (OTHERWISE JUST ON-EDGE OR AT-CORNER )')
1383:      end if
1384:    end if
1385: C
1386: C
1387: 800 NCNTR(7) = NCNTR(7) + NDEAD - IDEAD
1388: C
1389:      NDEAD1 = NDEAD
1390: C
1391: C-----
1392: C      ..... LOST PARTICLES .....
1393: C-----
1394: C
1395:      if ( INX.ne.0 ) then
1396:        do 810 I = 1, INX
1397:          LSDED(NDEAD+I) = IBP(I)
1398: 810      continue
1399:          NDEAD = NDEAD + INX
1400:          NLOST = NLOST + INX
1401: C/#IF PARA(SX* CRAY)
1402: *      call MVPSYNC_LOCK(2)
1403: C/#ENDIF
1404:          write(IOW,*) ' !! (SEAONE) ', INX, ' PARTICLES ARE LOST IN ',
1405:          &          'DEFERRED SEARCH. '
1406:          do 830 I = 1, INX
1407:            write(IOW,7040) IBP(I), XXX(IBP(I)), YYY(IBP(I)),
1408:            &          ZZZ(IBP(I)), AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I))
1409:            if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 )
1410:              &          write(IOW,7060)
1411:              &          (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,LEVL(IBP(I)))
1412:            if ( JTLLT.ne.0 ) then
1413:              do 820 L = 1, LEVL(IBP(I))
1414:                call PRNSPC( IOW, A, CHA, IBSPC(IBP(I),L), L )
1415: 820          continue
1416:            end if
1417: 830      continue
1418: C/#IF PARA(SX* CRAY)
1419: *      call MVPSYNC_UNLOCK(2)
1420: C/#ENDIF
1421:          end if
1422: C
1423: C-----
1424: C      ..... COUNT THE NUMBER OF BOUNDARY CROSSING
1425: C-----
1426: C
1427:          NCNTR(17) = NCNTR(17) + IDEFER - INX
1428: C
1429: C-----
1430: C      .... UPDATE OF REFLECTION STACK

```

src/gmvp/seaone.f

```

1431: C-----
1432: C
1433:       if ( JREFL.ne.0.and.IRCF.gt.NBREF(NR1) ) then
1434:         do 840 I = NBREF(NR1) + 1, IRCF
1435:           NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
1436:           C980828      IZREF(I) = IZZ(LSREF(I))
1437:         840 continue
1438:         CCCCCC NCNTR(18) = NCNTR(18) + IRCF - NBREF(NR1)
1439:         NBREF(NR1) = IRCF
1440:       end if
1441: C-----
1442: C-----
1443: C .... COMPRESS SEARCH STACK .....
1444: C-----
1445: C
1446:       II = 0
1447:       *VOCL LOOP,NOVREG
1448:       do 850 I = 1, NNXT(NZ1)
1449:         if ( IZNXT(I).ge.0 ) then
1450:           II = II + 1
1451:           LSSRC(II) = LSSRC(I)
1452:           IZNXT(II) = IZNXT(I)
1453:         end if
1454:       850 continue
1455:       NNXT(NZ1) = NNXT(NZ1) - IDEFER
1456:       CCCC if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
1457:       IDEFER = 0
1458: C-----
1459: C-----
1460: C-----
1461: C .... TAKE special tallies .....
1462: C-----
1463: C-----
1464: C
1465:       if ( NTEVE.gt.0.and.JTEVE(7).gt.0.and.NDEAD1-NDEAD0.gt.0 ) then
1466:         ISAFE = NDEAD1 - NDEAD0
1467: C-----
1468:         do 920 J = 0, JTEVE(7) - 1
1469: C-----
1470: C ... pointer to direct tally (idt) & d-tally # (idt) ...
1471:         IDT = KTEVE(LTEVE(7)+J) - 1
1472:         IT = IDTAL(IDT+9)
1473: C-----
1474: C .. neutron or photon ?
1475: C-----
1476:         NLEK = 0
1477:         if ( IPAR.eq.0 ) then
1478:           if ( IDTAL(IDT+5).eq.1 ) then
1479:             do 860 I = 1, ISAFE
1480:               IBPIII = LSDED(NDEAD0+I)
1481:               if ( IGG(IBPIII).le.NGP1 ) then
1482:                 NLEK = NLEK + 1
1483:                 IBP(NLEK) = IBPIII
1484:               end if
1485:             860 continue
1486:           else if ( IDTAL(IDT+5).eq.2 ) then
1487:             do 870 I = 1, ISAFE
1488:               IBPIII = LSDED(NDEAD0+I)
1489:               if ( IGG(IBPIII).gt.NGP1 ) then
1490:                 NLEK = NLEK + 1
1491:                 IBP(NLEK) = IBPIII
1492:               end if
1493:             870 continue
1494:           end if
1495:         else if ( IPAR.eq.IDTAL(IDT+5) ) then

```

```

1496:         do 880 I = 1, ISAFE
1497:           IBP(I) = LSDED(NDEAD0+I)
1498:         880 continue
1499:         NLEK = ISAFE
1500:       end if
1501: C-----
1502:       do 890 I = 1, NLEK
1503:         IWK(I) = IGG(IBP(I))
1504:         W(I) = WWW(IBP(I))
1505:         if ( JTLLT.eq.0 ) then
1506:           IZI(I) = KZREG(IZZ(IBP(I)))
1507:         else
1508:           IZI(I) = IBREG(IBP(I))
1509:         end if
1510:         if ( JTIME.gt.0 ) ISRF(I) = ITT(IBP(I))
1511:       890 continue
1512: C-----
1513: C ... skip if current region does not need this tally ...
1514: C-----
1515:       do 900 I = 1, NLEK
1516:         if ( JDTRG(IZI(I),IT).ne.0 ) go to 910
1517:       900 continue
1518:       go to 920
1519:       910 continue
1520: C-----
1521:       if ( NLEK.gt.0 ) then
1522:         call STALN7( IOW, IDTAL(IDT+1), W, IBP, NLEK, IWK, IZI,
1523:           & ISRF, NGROUP, NGP1, NREG, NTIME, NBANK, DTALY,
1524:           & DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, X, Y )
1525:       end if
1526:       920 continue
1527:     end if
1528: C-----
1529:     return
1530:   end

```


src/gmvp/search.f

```

1: C/#IF ARGSAVE
2: * subroutine SEARCH0(IOW, NBANK,NZONE,NSDA,NZDA,NREFS,NGROUP,
3: * N NREG, NLOST, NDEAD, IRAND, NLBZ, NEST, NSPACE, NSUZON, DEPS,
4: * B XXX, YYY, ZZZ, AAA, BBB, CCC,
5: * B WWW, IZZ, IGG, TTT, ITT, LEVL, LZZ,
6: * B LPOS, LCRS, XIM, IBREG, IBSPC,
7: * S LSFFL, NFFL, IZFFL, LSSRC, NNXT, IZNXT,
8: * S LSDED, LSREF, ISREF, IZREF, NBREF, KSREF,
9: * * LSLAT, IZLAT, NXLT,
10: * G SDA, KZMAT, KZREG, KZDA, KZAA, NKZAA,
11: * * KCELL, IPCEL, MLBZZ, ISUSP,
12: * T XIMP, WKIL, WSRV, NKILD, WKILD, NSURV,
13: * T WSURV, NSPLT, WSPLT, NCNTR, WCNTR,
14: * W IZT, IPZONE, MEM, JKSF, IWK2, IWK3,
15: * W IWK4, X, Y, Z, W, IWK,
16: * W IFG, LLS, IWK0, IFI, IFL, ISRF,
17: * W FXYZ, R, DSDA0, DSDA1, DSDA2, DSDA3,
18: * W DSDA4, DSDA5, DSDA6, DSDA7, DSDA8, DSDA9,
19: C/#ELSE
20: * subroutine SEARCH(IOW, NBANK,NZONE,NSDA,NZDA,NREFS,NGROUP,
21: * N NREG, NLOST, NDEAD, IRAND, NLBZ, NEST, NSPACE, NSUZON, DEPS,
22: * B XXX, YYY, ZZZ, AAA, BBB, CCC,
23: * B WWW, IZZ, IGG, TTT, ITT, LEVL, LZZ,
24: * B LPOS, LCRS, XIM, IBREG, IBSPC,
25: * S LSFFL, NFFL, IZFFL, LSSRC, NNXT, IZNXT,
26: * S LSDED, LSREF, ISREF, IZREF, NBREF, KSREF,
27: * * LSLAT, IZLAT, NXLT,
28: * G SDA, KZMAT, KZREG, KZDA, KZAA, NKZAA,
29: * * KCELL, IPCEL, MLBZZ, ISUSP,
30: * T XIMP, WKIL, WSRV, NKILD, WKILD, NSURV,
31: * T WSURV, NSPLT, WSPLT, NCNTR, WCNTR,
32: * W IZT, IPZONE, MEM, JKSF, IWK2, IWK3,
33: * W IWK4, X, Y, Z, W, IWK,
34: * W IFG, LLS, IWK0, IFI, IFL, ISRF,
35: * W FXYZ, R, DSDA0, DSDA1, DSDA2, DSDA3,
36: * W DSDA4, DSDA5, DSDA6, DSDA7, DSDA8, DSDA9,
37: * JMEM, NMEMO, KMEMO, MEMC, MEMZ)
38: C/#ENDIF
39: C=====
40: C PURPOSE: FIND ZONES TO ENTER. (ALL-ZONE LOGIC)
41: C CALLED IN: ACTION
42: C CALLS: JUDGE, VMNTR0, VMNTR1, VMNTR2
43: C=====
44: implicit real*8(D)
45: include 'INC/_FLAGS'
46: include '.../shared/INC/_TASKDT'
47: C
48: C ..... BANK .....
49: C
50: real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
51: real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
52: real WWW(NBANK), XIM(NBANK)
53: real*8 TTT(NBANK)
54: integer ITT(NBANK)
55: integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
56: & LPOS(NBANK,NEST), LCRS(NBANK,NEST)
57: integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
58: C
59: C ..... STACK .....
60: C
61: integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSSRC(NBANK),
62: & NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK), LSREF(NBANK),
63: & ISREF(NBANK), IZREF(NBANK), NBREF(1), LSLAT(NBANK),
64: & IZLAT(NBANK), NXLT(1)
65: C

```

```

66: C ..... GEOMETRY .....
67: C
68: real*8 SDA(NSDA)
69: integer KZMAT(NZONE), KZREG(NZONE), KSREF(NZDA), KZDA(2,NZDA),
70: & KZAA(NZONE+1), NKZAA(NZONE), KMEMO(NZONE,NMEMO),
71: & MEMC(NZONE,NMEMO), MEMZ(NZONE), KCELL(NZONE), IPCEL(1),
72: & MLBZZ(NZONE)
73: integer ISUSP(NSUZON,NSPACE)
74: C
75: C ..... VARIANCE REDUCTION .....
76: C
77: real XIMP(NGROUP,NREG), WKIL(NGROUP,NREG), WSRV(NGROUP,NREG)
78: C
79: C ..... TALLY (MONITOR) .....
80: C
81: real*8 NKILD(NGROUP,NREG), NSURV(NGROUP,NREG)
82: real*8 NSPLT(NGROUP,NREG)
83: real*8 WKILD(NGROUP,NREG), WSURV(NGROUP,NREG)
84: real*8 WSPLT(NGROUP,NREG)
85: real*8 NCNTR(*), WCNTR(*)
86: C
87: C ..... WORKING AREA .....
88: C
89: real*8 X(NBANK), Y(NBANK), Z(NBANK), W(NBANK), DSDA0(NBANK),
90: & DSDA1(NBANK), DSDA2(NBANK), DSDA3(NBANK), DSDA4(NBANK),
91: & DSDA5(NBANK), DSDA6(NBANK), DSDA7(NBANK), DSDA8(NBANK),
92: & DSDA9(NBANK)
93: real R(NBANK), FXYZ(NBANK)
94: integer IWK(NBANK), LLS(NBANK), IWK0(NBANK), ISRF(NBANK),
95: & IZT(NZONE+1), IPZONE(NZONE+1), MEM(NZONE+1),
96: & JKSF(NZONE+1), IWK2(NZONE+1), IWK3(NZONE+1), IWK4(NZONE+1)
97: logical IFG(NBANK), IFI(NBANK), IFL(NBANK), JRR, JMEM
98: C
99: C/#IF ARGSAVE
100: * return
101: C -----
102: * entry SEARCH ( JMEM, NMEMO, KMEMO, MEMC, MEMZ )
103: C/#ENDIF
104: C
105: NZ1 = NZONE + 1
106: C
107: if ( JVMNT.ne.0 ) call VMNTR1( 4, NNXT(NZ1) )
108: C=====
109: C ... GATHER VECTORS .....
110: C * NZK : NUMBER OF ZONES HAVING PARTICLES IN SEARCH STACK.
111: C * JKSF : ZONE # LIST OF ZONES HAVING PARTICLES IN SEARCH STACK.
112: C * IPZONE : ZONE POINTERS IN GATHERD ARRAYS (X,Y,Z ETC.)
113: C=====
114: NZK = 0
115: do 100 K = 1, NZONE
116: if ( NNXT(K).ne.0 ) then
117: NZK = NZK + 1
118: JKSF(NZK) = K
119: end if
120: 100 continue
121: C
122: if ( NZK.gt.20 ) then
123: IPZONE(1) = 1
124: *VOCL LOOP, SCALAR
125: do 110 NK = 1, NZK
126: IPZONE(NK+1) = IPZONE(NK) + NNXT(JKSF(NK))
127: 110 continue
128: *VOCL LOOP, NOVREC
129: do 120 NK = 1, NZK
130: MEM(JKSF(NK)) = IPZONE(NK) - 1

```

src/gmvp/search.f

```

131: 120 continue
132: *VOCL LOOP, SCALAR
133: do 130 I = 1, NNXT(NZ1)
134:   MEM(IZNXT(I)) = MEM(IZNXT(I)) + 1
135:   LLS(I) = MEM(IZNXT(I))
136: 130 continue
137: *VOCL LOOP, NOVREC
138: do 140 I = 1, NNXT(NZ1)
139:   IWK(LLS(I)) = LSSRC(I)
140: 140 continue
141: else
142:   IPZONE(1) = 1
143:   do 150 NK = 1, NZK
144:     IPZONE(NK+1) = IPZONE(NK) + NNXT(JKSF(NK))
145:   150 continue
146:   KKK = 0
147:   do 170 NK = 1, NZK
148:     do 160 I = 1, NNXT(NZ1)
149:       if ( IZNXT(I).eq.JKSF(NK) ) then
150:         KKK = KKK + 1
151:         IWK(KKK) = LSSRC(I)
152:       end if
153:     160 continue
154:   170 continue
155: end if
156: C
157: C ..... GATHER POSITIONS .....
158: C MOVE EACH PARTLES TO THEIR FLIGHT DIRECTIONS BY SMALL DISTANCE (DEPS)
159: C TO PREVENT MISJUDGE. (7/20/1988)
160: C
161: do 180 I = 1, NNXT(NZ1)
162:   LSSRC(I) = IWK(I)
163:   X(I) = XXX(IWK(I)) + DEPS*AAA(IWK(I))
164:   Y(I) = YYY(IWK(I)) + DEPS*BBB(IWK(I))
165:   Z(I) = ZZZ(IWK(I)) + DEPS*CCC(IWK(I))
166:   W(I) = WWW(IWK(I))
167: 180 continue
168: C=====
169: C JUDGEMENTS
170: C ..... * MEM(NK) : CONTROL ARRAYS FOR MEMORANDUM ARRAY TREATMENT
171: C = 0 : GET CANDIDATES OF NEXT ZONE FROM KMEMO ARRAY.
172: C > 0 : SET MEM(NK)'S ELEMENT OF KMEMO TO A NEWLY FOUND
173: C NEXT ZONE#.
174: C=====
175: do 190 NK = 1, NZK
176:   MEM(NK) = 0
177: 190 continue
178: C
179: C .... * INX : NUMBER OF PARTICLES WHOSE NEXT ZONES ARE SEARCHED.
180: INX = NNXT(NZ1)
181: IFFL = NFFL(NZ1)
182: if ( JREFL.ne.0 ) ICREF = NBREF(NREFS+1)
183: if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
184: C880720 DO 340 M=1,NZONE-1
185: do 720 M = 1, NZONE
186: C
187: C-----
188: C .... DETERMINE ZONE TO BE SEARCHED .....
189: C-----
190: C
191: LL = 0
192: if ( M.le.NMEMO ) then
193: do 200 NK = 1, NZK
194:   if ( KMEMO(JKSF(NK),M).ne.0 ) then
195:     IZT(NK) = KMEMO(JKSF(NK),M)

```

```

196:   LL = LL + 1
197: end if
198: 200 continue
199: end if
200: C
201: C-----
202: C .... NO CANDIDATE OF NEXT ZONE IN KMEMO .....
203: C-----
204:   if ( LL.lt.NZK ) then
205:     do 240 NK = 1, NZK
206:       K = JKSF(NK)
207:       ICEL = 0
208:       if ( JLATT.ne.0 ) ICEL = KCELL(K)
209:       if ( M.gt.NMEMO .or. MEM(NK).ne.0 .or. KMEMO(K,M).eq.0 )
210:         & then
211: C
212: C-----
213: C ..... MEM = 0 : BEGINS SEARCHING OF NEW NEXT-ZONE OTHER THAN
214: C THOSE IN ARRAY KMEMO. IF A NEW NEXT-ZONE IS FOUND,
215: C IT IS MEMORIZED AT KMEMO(JKSF(NK),MEM(NK)).
216: C-----
217: C
218:   if ( MEM(NK).eq.0 ) then
219:     MEM(NK) = M
220:     IZT(NK) = 1
221:     if ( ICEL.ne.0 ) IZT(NK) = IPCEL(ICEL)
222:   else
223:     IZT(NK) = IZT(NK) + 1
224:   end if
225:   MMZ = NZONE
226:   if ( JLATT.ne.0 ) MMZ = IPCEL(ICEL+1) - 1
227: *VOCL LOOP, SCALAR
228: do 220 MK = IZT(NK), MMZ
229:   if ( MK.eq.K ) go to 220
230:   do 210 L = 1, NMEMO
231:     if ( MK.eq.KMEMO(K,L) ) go to 220
232:   210 continue
233:   go to 230
234: 220 continue
235: C 1988/7/20 CCCCCCCC MK = NZONE
236: C ..... IF NO NEXT-ZONE WERE FOUND AFTER SEARCHING ALL ZONES EXCEPT
237: C THAT IN WHICH PARTICLES LIE (JKSF(NK)), THE ZONE JKSF(NK) WOULD
238: C BE SELECTED AS THE CANDIDATE. (I HOPE THAT THIS CASE BE RARE.)
239: C
240:   MK = K
241: C011691 MK = MMZ
242: 230 IZT(NK) = MK
243: end if
244: 240 continue
245: end if
246: C
247: C-----
248: C ..... CALCULATE MAXIMUM SURFACE NUMBER OF ZONES .....
249: C-----
250:   MSF = 0
251:   do 250 NK = 1, NZK
252:     K = IZT(NK)
253:     MSF = MAX(NKZAA(K),MSF)
254: 250 continue
255: C
256: C-----
257: C ..... SET FLAGS ( IFG = T/F : IN/OUT) .....
258: C-----
259: do 260 I = 1, INX
260:   IFG(I) = .true.

```

src/gmvp/search.f

```

261:          if ( JSIMP.eq.0 ) IFL(I)   = .false.
262:          if ( JREFL.ne.0 ) FXYZ(I)   = 1.0E30
263: 260      continue
264: C -----
265: C ..... LOOP BY SURFACE NUMBER .....
266: C -----
267: C .....
268:       do 560 N = 1, MSF
269:         do 270 I = 1, INX
270:           IFI(I) = .true.
271:           IWK(I) = 0
272: 270      continue
273:         do 320 NK = 1, NZK
274:           IWK2(NK) = 0
275:           K = IZT(NK)
276:           KK = NKZAA(K)
277:           if ( KK.ge.N ) then
278: C .....
279: C .... IWK2 : TYPE OF SURFACE   IWK3 : SIGN * (SDA POINTER)
280: C .... IWK4 : LOGICAL TREATMENT FLAG (0/1/2)
281: C .....
282:           if ( JSIMP.eq.0 ) LLG = KZDA(2,KZAA(K)+N-1)
283:           LS = KZDA(1,KZAA(K)+N-1)
284:           LSG = -SIGN(1,LS)
285:           LSP = ABS(LS)
286:           KSF = INT(ABS(SDA(LSP)))
287:           IWK2(NK) = KSF
288: C .....
289: C -----
290: C .... GATHER SURFACE DATA ....
291: C -----
292: C .....
293:       if ( KSF.eq.1 ) then
294:         do 280 I = IPZONE(NK), IPZONE(NK+1) - 1
295:           LLS(I) = LSG
296:           if ( JSIMP.eq.0 ) IWK(I) = LLG
297:           DSDA0(I) = SDA(LSP+1)
298:           DSDA1(I) = SDA(LSP+2)
299:           DSDA2(I) = SDA(LSP+3)
300:           DSDA3(I) = SDA(LSP+4)
301:           DSDA4(I) = SDA(LSP+5)
302: 280      continue
303:       else if ( KSF.eq.2 .or. KSF.eq.5 ) then
304:         do 290 I = IPZONE(NK), IPZONE(NK+1) - 1
305:           LLS(I) = LSG
306:           if ( JSIMP.eq.0 ) IWK(I) = LLG
307:           DSDA0(I) = SDA(LSP+1)
308:           DSDA1(I) = SDA(LSP+2)
309:           DSDA2(I) = SDA(LSP+3)
310:           DSDA3(I) = SDA(LSP+4)
311: 290      continue
312:       else if ( KSF.eq.3 ) then
313:         do 300 I = IPZONE(NK), IPZONE(NK+1) - 1
314:           LLS(I) = LSG
315:           if ( JSIMP.eq.0 ) IWK(I) = LLG
316:           DSDA0(I) = SDA(LSP+1)
317:           DSDA1(I) = SDA(LSP+2)
318:           DSDA2(I) = SDA(LSP+3)
319:           DSDA3(I) = SDA(LSP+4)
320:           DSDA4(I) = SDA(LSP+5)
321:           DSDA5(I) = SDA(LSP+6)
322:           DSDA6(I) = SDA(LSP+7)
323: 300      continue
324:       else if ( KSF.eq.4 ) then
325:         do 310 I = IPZONE(NK), IPZONE(NK+1) - 1

```

```

326:           LLS(I) = LSG
327:           if ( JSIMP.eq.0 ) IWK(I) = LLG
328:           DSDA0(I) = SDA(LSP+1)
329:           DSDA1(I) = SDA(LSP+2)
330:           DSDA2(I) = SDA(LSP+3)
331:           DSDA3(I) = SDA(LSP+4)
332:           DSDA4(I) = SDA(LSP+5)
333:           DSDA5(I) = SDA(LSP+6)
334:           DSDA6(I) = SDA(LSP+7)
335:           DSDA7(I) = SDA(LSP+8)
336:           DSDA8(I) = SDA(LSP+9)
337:           DSDA9(I) = SDA(LSP+10)
338: 310      continue
339:         end if
340:       end if
341:       if ( JREFL.ne.0 ) IWK3(NK) = KSREF(KZAA(IZT(NK)))+N-1)
342: 320      continue
343: C -----
344: C ..... JUDGE WHETHER PARTICLES ARE IN ZONE IZT(NK) OR NOT.
345: C ..... ZONES FROM KKK TO KKK2 HAVE SURFACES OF THE SAME TYPE.
346: C -----
347: C .....
348:       KKK = 1
349: 330      KSF = IWK2(KKK)
350:       do 340 NK = KKK, NZK
351:         if ( IWK2(NK).ne.KSF ) go to 350
352: 340      continue
353: C .....
354: C -----
355: C ..... END OF SURFACE LOOP IF ALL OF SURFACE TYPE (IWK2) = 0 ....
356: C -----
357: CCC      IF(KSF.EQ.0.AND.KKK.EQ.1) GOTO 282
358: C .....
359:       NK = NZK + 1
360: 350      KKK2 = NK
361:       JRR = .false.
362:       if ( KSF.ne.0.and.JREFL.ne.0 ) then
363:         do 360 NK = KKK, KKK2 - 1
364:           if ( IWK3(NK).ne.0 ) JRR = .true.
365: 360      continue
366:       end if
367: C .....
368: C -----
369: C .... CHECK FOR EACH PARTICLE. IFG(I) = T/F = IN/OUT
370: C -----
371:       go to(370,400,430,460,490) KSF
372:       if ( KSF.ne.0 ) go to 520
373:       go to 530
374: C .....
375: C ..... SLAB .....
376: CCCCCCCCCCCCC IF(KSF.EQ.1) THEN
377: 370      if ( JRR ) then
378:         do 380 I = IPZONE(KKK), IPZONE(KKK2) - 1
379:           D1 = DSDA0(I)*X(I) + DSDA1(I)*Y(I) + DSDA2(I)*
380:             & Z(I)
381:           D1 = (D1-DSDA3(I))*(D1-DSDA4(I))*LLS(I)
382:           IFI(I) = D1.ge.0.0
383:           D1 = ABS(D1)
384:           if ( D1.lt.FXYZ(I).and.IFI(I) ) then
385:             ISRF(I) = N
386:             FXYZ(I) = D1
387:           end if
388: 380      continue
389:       else
390:         do 390 I = IPZONE(KKK), IPZONE(KKK2) - 1

```

src/gmvp/search.f

```

391:          D1      = DSDA0(I)*X(I) + DSDA1(I)*Y(I) + DSDA2(I)*
392:          &        Z(I)
393:          IFI(I)   = (D1-DSDA3(I))*(D1-DSDA4(I))*LLS(I) .ge.0.0
394: 390      continue
395:      end if
396:      go to 530
397: C
398: C ..... SPHERE .....
399: CCCCCCCCCCCCC ELSE IF(KSF.EQ.2) THEN
400: 400      if ( JRR ) then
401:          do 410 I = IPZONE(KKK), IPZONE(KKK2) - 1
402:              D1      = ((DSDA0(I)-X(I))**2+(DSDA1(I)-Y(I))**2
403:              &        +(DSDA2(I)-Z(I))**2-DSDA3(I))*LLS(I)
404:              IFI(I)   = D1.ge.0.0
405:              D1      = ABS(D1)
406:              if ( D1.lt.FXYZ(I).and.IFI(I) ) then
407:                  ISRF(I) = N
408:                  FXYZ(I) = D1
409:              end if
410:          continue
411:      else
412:          do 420 I = IPZONE(KKK), IPZONE(KKK2) - 1
413:              IFI(I)   = ((DSDA0(I)-X(I))**2+(DSDA1(I)-Y(I))**2
414:              &        +(DSDA2(I)-Z(I))**2-DSDA3(I))*LLS(I) .ge.0.0
415:          420      continue
416:          end if
417:          go to 530
418: C
419: C ..... CYLINDER .....
420: CCCCCCCCCCCCC ELSE IF(KSF.EQ.3) THEN
421: 430      if ( JRR ) then
422:          do 440 I = IPZONE(KKK), IPZONE(KKK2) - 1
423:              DX      = X(I) - DSDA0(I)
424:              DY      = Y(I) - DSDA1(I)
425:              DZ      = Z(I) - DSDA2(I)
426:              D1      = (DX**2+DY**2+DZ**2-DSDA6(I)
427:              &        -(DSDA3(I)*DX+DSDA4(I)*DY+DSDA5(I)*DZ)**2)*
428:              &        LLS(I)
429:              IFI(I)   = D1.ge.0.0
430:              D1      = ABS(D1)
431:              if ( D1.lt.FXYZ(I).and.IFI(I) ) then
432:                  ISRF(I) = N
433:                  FXYZ(I) = D1
434:              end if
435:          440      continue
436:      else
437:          do 450 I = IPZONE(KKK), IPZONE(KKK2) - 1
438:              DX      = X(I) - DSDA0(I)
439:              DY      = Y(I) - DSDA1(I)
440:              DZ      = Z(I) - DSDA2(I)
441:              IFI(I)   = (DX**2+DY**2+DZ**2-DSDA6(I)
442:              &        -(DSDA3(I)*DX+DSDA4(I)*DY+DSDA5(I)*DZ)**2)*
443:              &        LLS(I) .ge.0.0
444:          450      continue
445:          end if
446:          go to 530
447: C
448: C ..... QUADARATIC SURFACE (GENERAL FORM) .....
449: CCCCCCCCCCCCC ELSE IF(KSF.EQ.4) THEN
450: 460      if ( JRR ) then
451:          do 470 I = IPZONE(KKK), IPZONE(KKK2) - 1
452:              D1      = (X(I)*
453:              &        (X(I)*DSDA0(I)+Y(I)*DSDA3(I)+Z(I)*DSDA5(I)
454:              &        +DSDA6(I))+Y(I)*
455:              &        (Y(I)*DSDA1(I)+Z(I)*DSDA4(I)+DSDA7(I))+Z(I)*

```

```

456:          &        (Z(I)*DSDA2(I)+DSDA8(I))+DSDA9(I))*LLS(I)
457:          IFI(I)   = D1.ge.0.0
458:          D1      = ABS(D1)
459:          if ( D1.lt.FXYZ(I).and.IFI(I) ) then
460:              ISRF(I) = N
461:              FXYZ(I) = D1
462:          end if
463: 470      continue
464:      else
465:          do 480 I = IPZONE(KKK), IPZONE(KKK2) - 1
466:              IFI(I)   = (X(I)*
467:              &        (X(I)*DSDA0(I)+Y(I)*DSDA3(I)+Z(I)*DSDA5(I)
468:              &        +DSDA6(I))+Y(I)*
469:              &        (Y(I)*DSDA1(I)+Z(I)*DSDA4(I)+DSDA7(I))+Z(I)*
470:              &        (Z(I)*DSDA2(I)+DSDA8(I))+DSDA9(I))*LLS(I) .ge.
471:              &        0.0
472:          480      continue
473:          end if
474:          go to 530
475: C
476: C ..... PLANE (HALF SPACE) .....
477: CCCCCCCCCCCCC ELSE IF(KSF.EQ.5) THEN
478: 490      if ( JRR ) then
479:          do 500 I = IPZONE(KKK), IPZONE(KKK2) - 1
480:              D1      =
481:              &        (X(I)*DSDA0(I)+Y(I)*DSDA1(I)+Z(I)*DSDA2(I)
482:              &        -DSDA3(I))*LLS(I)
483:              IFI(I)   = D1.ge.0.0
484:              D1      = ABS(D1)
485:              if ( D1.lt.FXYZ(I).and.IFI(I) ) then
486:                  ISRF(I) = N
487:                  FXYZ(I) = D1
488:              end if
489:          500      continue
490:      else
491:          do 510 I = IPZONE(KKK), IPZONE(KKK2) - 1
492:              IFI(I)   =
493:              &        (X(I)*DSDA0(I)+Y(I)*DSDA1(I)+Z(I)*DSDA2(I)
494:              &        -DSDA3(I))*LLS(I) .ge.0.0
495:          510      continue
496:          end if
497:          go to 530
498: C
499: 520      write(IOW,*) ' !! INVALID SURFACE TYPE ', KSF
500:          stop 666
501: C
502: 530      KKK      = KKK2
503:          if ( KKK.le.NZK ) go to 330
504: C
505: C-----
506: C ..... JUDGEMENT .....
507: C-----
508:          if ( JSIMP.eq.1 ) then
509:              do 540 I = 1, INX
510:                  IFG(I) = IFG(I) .and.IFI(I)
511:          540      continue
512:          else
513:              do 550 I = 1, INX
514:                  if ( IWK(I).eq.0 ) then
515:                      IFG(I) = IFG(I) .and.IFI(I)
516:                  else
517:                      IFL(I) = IFL(I) .or. IFI(I)
518:                      if ( IWK(I).eq.2 ) then
519:                          IFG(I) = IFG(I) .and.IFL(I)
520:                      IFL(I) = .false.

```

src/gmvp/search.f

```

521:                end if
522:            end if
523:    550        continue
524:        end if
525:    560        continue
526: C
527: C-----
528: C .... COMPRESS PARTICLE DATA. UPDATE FLIGHT STACK & REFLECTION STACK
529: C      KK IS THE NUMBER OF PARTICLES WHOSE NEXT ZONES ARE DETERMINED.
530: C-----
531:    570        KK      = 0
532:        do 580 I = 1, INX
533:            if ( IFG(I) ) KK      = KK + 1
534:        580        continue
535: C
536:        if ( KK.ne.0 ) then
537:            do 590 NK = 1, NZK
538:                IWK2(NK) = 0
539:    590        continue
540: C
541: C-----
542: C ... SEND PARTICLES WHOSE NEXT ZONES HAVE BEEN FOUND TO FLIGHT STACK
543: C      IF THE NEXT ZONE IS OUTER VOID, PARTICLES ARE TREATED AS LEAKED.
544: C-----
545:        do 660 NK = 1, NZK
546:            MAT      = KZMAT(IZT(NK))
547:            if ( IZT(NK).eq.JKSF(NK) ) then
548:                if ( MAT.ge.0 ) then
549: C-----
550: C      ..... KEEP ON TRACKING .....
551: C-----
552: *VOCL LOOP,NOVREC
553:        do 600 I = IPZONE(NK), IPZONE(NK+1) - 1
554:            if ( IFG(I) ) then
555: C      ..... TANGENTIAL PARTICLES
556:                IWK2(NK) = IWK2(NK) + 1
557:                IFFL      = IFFL + 1
558:                LSFFL(IFFL) = LSSRC(I)
559:                IZFFL(IFFL) = IZT(NK)
560:                XXX(LSSRC(I)) = X(I)
561:                YYY(LSSRC(I)) = Y(I)
562:                ZZZ(LSSRC(I)) = Z(I)
563: C      ..... IWK0 : OLD ZONE NUMBER (NECESSARY IN SPLITTING)
564: C      IF(JIMPT.NE.0) IWK0(IFFL) = JKSF(NK)
565:            end if
566:    600        continue
567:        else
568:            do 610 I = IPZONE(NK), IPZONE(NK+1) - 1
569:                IFG(I) = .false.
570:    610        continue
571:            end if
572: C011691        IF(MAT.GE.0) THEN
573:            else if ( MAT.ge.0 ) then
574: C-----
575: C      ..... KEEP ON TRACKING .....
576: C-----
577: *VOCL LOOP,NOVREC
578:        do 620 I = IPZONE(NK), IPZONE(NK+1) - 1
579:            if ( IFG(I) ) then
580:                IWK2(NK) = IWK2(NK) + 1
581:                IFFL      = IFFL + 1
582:                LSFFL(IFFL) = LSSRC(I)
583:                IZFFL(IFFL) = IZT(NK)
584: C      ... set IBREG anytime (from Jan 2000)
585:                IBREG(LSSRC(I)) = KZREG(IZT(NK))

```

```

586:            if ( JTLT.ne.0 ) then
587:                ILV      = LEVL(LSSRC(I))
588:                if ( ILV.gt.0 ) then
589:                    IBREG(LSSRC(I)) =
590:                        &      ISUSP(KZREG(IZT(NK))),
591:                        &      IBSPC(LSSRC(I),ILV))
592:            end if
593:        end if
594: C      ..... IWK0 : OLD ZONE NUMBER (NECESSARY IN SPLITTING)
595: C      IF(JIMPT.NE.0) IWK0(IFFL) = JKSF(NK)
596:        end if
597:    620        continue
598: C
599: C-----
600: C      ..... TERMINATE TRACKING (LEAKAGE) .....
601: C-----
602:        else if ( MAT.eq.-1000 ) then
603:            IDEAD      = NDEAD
604: *VOCL LOOP,NOVREC
605:            do 630 I = IPZONE(NK), IPZONE(NK+1) - 1
606:                if ( IFG(I) ) then
607:                    IDEAD      = IDEAD + 1
608:                    LSDED(IDEAD) = LSSRC(I)
609:                    WCNTR(7)     = WCNTR(7) + W(I)
610:                end if
611:    630        continue
612:                IWK2(NK) = IDEAD - NDEAD
613:                NCNTR(7) = NCNTR(7) + IWK2(NK)
614:                NCNTR(17) = NCNTR(17) + IWK2(NK)
615:                NDEAD     = IDEAD
616: C
617: C-----
618: C      ..... REFLECTION .....
619: C-----
620:        else if ( MAT.lt.-1000 ) then
621:            IW      = KZAA(IZT(NK)) - 1
622:            IRF      = ICREF
623:            do 640 I = IPZONE(NK), IPZONE(NK+1) - 1
624:                if ( IFG(I) ) then
625:                    IRF      = IRF + 1
626:                    LSREF(IRF) = LSSRC(I)
627:                    ISREF(IRF) = KSREF(IW+ISRF(I))
628:                    IZREF(IRF) = JKSF(NK)
629:                end if
630:    640        continue
631:            if ( IRF.gt.ICREF ) then
632:                IWK2(NK) = IRF - ICREF
633:                ICREF     = IRF
634:            end if
635: C
636: C-----
637: C      ..... ESCAPING FROM LATTICE OR CELL .....
638: C-----
639:        else if ( MAT.le.-1.and.MAT.ge.-999 ) then
640:            ILL      = ILAT
641:            do 650 I = IPZONE(NK), IPZONE(NK+1) - 1
642:                if ( IFG(I) ) then
643:                    ILL      = ILL + 1
644:                    LSLAT(ILL) = LSSRC(I)
645:                    IZLAT(ILL) = MLBZZ(IZT(NK))
646:                    IZZ(LSSRC(I)) = IZT(NK)
647:                end if
648:    650        continue
649:            if ( ILL.gt.ILAT ) then
650:                IWK2(NK) = ILL - ILAT

```

src/gmvp/search.f

```

651:          NXLT(MLBZZ(IZT(NK))) = NXLT(MLBZZ(IZT(NK)))
652:      &          + (ILL-ILAT)
653:      ILAT = ILL
654:  end if
655:  end if
656:  continue
657: *VOCL LOOP, SCALAR
658:  do 670 NK = 1, NZK
659:    if ( KZMAT(IZT(NK)).ge.0 ) NFFL(IZT(NK)) =
660:    &      NFFL(IZT(NK)) + IWK2(NK)
661:  continue
662: C
663: C-----
664: C ..... REGISTER THE NEXT-ZONE TO KMEMO ARRAY
665: C-----
666: *VOCL LOOP, NOVREC
667:  do 680 NK = 1, NZK
668:    if ( IWK2(NK).ne.0 ) then
669: C880901 ***      IF(MEM(NK).NE.0.AND.MEM(NK).LE.NMEMO) THEN
670: C                  KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
671: C                  MEM(NK) = MEM(NK) + 1
672: C*****      ENDIF
673:    if ( .not.JMEM ) then
674:      if ( MEM(NK).ne.0.and.MEM(NK).le.NMEMO ) then
675:        KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
676:        MEM(NK) = MEM(NK) + 1
677:      end if
678:    else
679:      if ( MEM(NK).ne.0.and.MEM(NK).le.NMEMO ) then
680:        KMEMO(JKSF(NK),MEM(NK)) = IZT(NK)
681:        MEMC(JKSF(NK),MEM(NK)) = MEMC(JKSF(NK),MEM(NK))
682:        &          + IWK2(NK)
683:        MEMZ(JKSF(NK)) = MEM(NK)
684:        MEM(NK) = MEM(NK) + 1
685:      else if ( MEM(NK).eq.0 ) then
686:        MEMC(JKSF(NK),M) = MEMC(JKSF(NK),M) +
687:        &          IWK2(NK)
688:      end if
689:    end if
690:  end if
691: 680  continue
692: C
693: C-----
694: C ..... RE-COUNT THE NUMBER OF NON-ZERO-PARTICLE ZONE. (NZKN - NZK)
695: C ..... CHANGE IWK2 (NUMBER OF PARTICLES WHOSE NEXT-ZONES ARE FOUND)
696: C ..... TO THE NUMBER OF PARTICLES WHOSE NEXT-ZONES ARE NOT FOUND.
697: C-----
698: C
699:      NZKN = 0
700: *VOCL LOOP, NOVREC
701:  do 690 NK = 1, NZK
702:    IWK2(NK) = IPZONE(NK+1) - IPZONE(NK) - IWK2(NK)
703:    if ( IWK2(NK).ne.0 ) then
704:      NZKN = NZKN + 1
705:      JKSF(NZKN) = JKSF(NK)
706:      IWK2(NZKN) = IWK2(NK)
707:      IZT(NZKN) = IZT(NK)
708:      MEM(NZKN) = MEM(NK)
709:    end if
710: 690  continue
711: C
712: C-----
713: C ..... COMPRESS SEARCH STACK .....
714: C-----
715:      KK = 0

```

```

716: *VOCL LOOP, NOVREC
717:  do 700 I = 1, INX
718:    if ( .not.IFG(I) ) then
719:      KK = KK + 1
720:      LSSRC(KK) = LSSRC(I)
721:      X(KK) = X(I)
722:      Y(KK) = Y(I)
723:      Z(KK) = Z(I)
724:      W(KK) = W(I)
725:    end if
726: 700  continue
727:      INX = KK
728:      NZK = NZKN
729:      IPZONE(1) = 1
730: *VOCL LOOP, SCALAR
731:  do 710 NK = 1, NZK
732:    IPZONE(NK+1) = IPZONE(NK) + IWK2(NK)
733: 710  continue
734:  end if
735:  if ( INX.eq.0 ) go to 730
736: 720  continue
737: C
738: C-----
739: C ..... COUNT UP OF REFLECTION STACK .....
740: C-----
741: 730 if ( JREFL.ne.0.and.ICREF.gt.NBREF(NREFS+1) ) then
742: *VOCL LOOP, SCALAR
743:  do 740 I = NBREF(NREFS+1) + 1, ICREF
744:    NBREF(ISREF(I)) = NBREF(ISREF(I)) + 1
745: 740  continue
746:    NBREF(NREFS+1) = ICREF
747:  end if
748: C
749: C =====
750: C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, TREAT THEM
751: C AS LOST PARTICLE.
752: C =====
753: C
754:  if ( INX.ne.0 ) then
755:    do 750 I = 1, INX
756:      LSDDED(NDEAD+I) = LSSRC(I)
757:    750  continue
758:      NDEAD = NDEAD + INX
759:      NLOST = NLOST + INX
760: C/#IF PARA(SX* CRAY)
761: *      call MVPSYNC_LOCK(2)
762: C/#ENDIF
763: C
764: C
765:      write(IOW,*) ' !! (SEARCH) ', INX, ' PARTICLES ARE LOST !! '
766:      write(IOW,7000) (X(I),Y(I),Z(I),AAA(LSSRC(I)),BBB(LSSRC(I)),
767:      &      CCC(LSSRC(I)),IZZ(LSSRC(I)),I=1,INX)
768: 7000  format(1X,' X= ',1P,E12.5,' Y= ',E12.5,' Z= ',E12.5,' MU= ',
769:      &      E12.5,' ETA=',E12.5,' XI=',E12.5,' ZONE=',I5)
770:      if ( JLATT.ne.0 ) write(IOW,7020) (LEVL(LSSRC(I)),
771:      &      LZZ(LSSRC(I),LEVL(LSSRC(I))),LPOS(LSSRC(I),LEVL(LSSRC(I))),
772:      &      I=1,INX)
773: 7020  format(1X,' LEVL = ',I3,' LZZ = ',I7,' LPOS = ',I7)
774: C/#IF PARA(SX* CRAY)
775: *      call MVPSYNC_UNLOCK(2)
776: C/#ENDIF
777: C
778: C
779:  end if
780: C

```

```
src/gmvp/search.f
```

```

781: C .....
782: C ..... UPDATING OF ZONE # IN BANK .....
783: C -----
784: C         if ( IFFL.gt.NFFL(NZ1) ) then
785: C             KFFL = NFFL(NZ1) + 1
786: C             do 760 I = KFFL, IFFL
787: C                 IZZ(LSFFL(I)) = IZFFL(I)
788: C             760 continue
789: C                 NCNTR(17) = NCNTR(17) + IFFL - NFFL(NZ1)
790: C             end if
791: C
792: C =====
793: C = BOUNDARY SPLITTING AND/OR RUSSIAN ROULETTE =
794: C =====
795: C         if ( (JIMPT.ne.0.or.JWWND.ne.0).and.IFFL.gt.NFFL(NZ1) ) then
796: C             call BANU2( IRANE, R, IFFL, ICON )
797: C             Ccc*VOCL LOOP,NOVREC(XIM)
798: C             *VOCL LOOP,NOVREC
799: C             do 770 I = KFFL, IFFL
800: C                 W(I) = WWW(LSFFL(I))
801: C                 IGT = IGG(LSFFL(I))
802: C
803: C ===== RUSSIAN ROULETTE FOR DECIMAL PART. =====
804: C         << INT(X + R) ( 0 < R < 1 ) >>
805: C
806: C         I-----I-----I ..... I-----+-----I
807: C         0         1         2         INT(X) X         INT(X)-1
808: C                     I         I         I         X+1
809: C                     I...Y...I... 1-Y I...Y...I
810: C
811: C         WHEN Y = X - INT(X) (DECIMAL PART)
812: C         * IF R < 1.0 - Y ----> INT(X+R) = INT(X)
813: C         * IF R > 1.0 - Y ----> INT(X+R) = INT(X) + 1
814: C         ==> PROBABILITY OF SURVIVAL IS Y !!!
815: C =====
816: C
817: C ... LLS(I): NUMBER OF SPLITTING, Y(I): WEIGHT OF SPLITTED OR SURVIVED
818: C                                     PARTICLES
819: C         if ( JIMPT.ne.0 ) then
820: C             XLS = XIMP(IGT,KZREG(IZFFL(I)))
821: C             & /XIMP(IGT,KZREG(IWK0(I))) )
822: C             XIMPN = XIMP(IGT,KZREG(IZFFL(I)))
823: C             XLS = XIMPN/XIM(LSFFL(I))
824: C             XIM(LSFFL(I)) = XIMPN
825: C             LLS(I) = XLS + R(I)
826: C             CCCCCCCCCC IF(XLS.ge.1.0) XLS = LLS(I)
827: C             if ( XLS.ne.0.0 ) Y(I) = W(I) /XLS
828: C             else if ( JWWND.ne.0 ) then
829: C                 LLS(I) = 1
830: C                 Y(I) = W(I)
831: C                 WS = WSRV(IGT,KZREG(IZFFL(I)))
832: C                 WK = WKIL(IGT,KZREG(IZFFL(I)))
833: C                 if ( W(I).gt.2.0*WS .or. W(I).lt.WK ) then
834: C                     LLS(I) = W(I) /WS + R(I)
835: C                     Y(I) = WS
836: C                 end if
837: C             end if
838: C             770 continue
839: C -----
840: C ..... REMOVE KILLED PARTICLES FROM STACK & BANK .....
841: C AND COUNT THE NUMBER OF PARTICLES TO BE CREATED THRU. SPLITTING
842: C -----
843: C         II = NFFL(NZ1)
844: C         Ccc*VOCL LOOP,NOVREC(LLS,Y,W,LSFFL,IZFFL)
845: C         *VOCL LOOP,NOVREC

```

```

846:      do 780 I = KFFL, IFFL
847:        if ( LLS(I).ne.0 ) then
848:          II      = II + 1
849:          LLS(II) = LLS(I)
850:          Y(II)   = Y(I)
851:          W(II)   = W(I)
852:          LSFFL(II) = LSFFL(I)
853:          IZFFL(II) = IZFFL(I)
854:        else
855:          WCNTR(9) = WCNTR(9) + W(I)
856:          NDEAD   = NDEAD + 1
857:          LSDED(NDEAD) = LSFFL(I)
858:          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) - 1
859:        end if
860:      780 continue
861:      NCNTR(9) = NCNTR(9) + IFFL - II
862:      IFFL     = II
863:      NSPLTT   = 0
864:      do 790 I = KFFL, IFFL
865:        NSPLTT = NSPLTT + LLS(I)
866:      790 continue
867:      NSPLTT = NSPLTT - (IFFL-NFFL(NZ1))
868: C
869: C-----
870: C ..... INSUFFICIENT SPACE IN BANK TO ACCEPT ALL GENERATED PARTICLES.
871: C-----
872:      if ( NSPLTT.gt.NDEAD ) then
873:        II = 0
874:        do 800 KK = KFFL, IFFL
875:          II = II + LLS(KK) - 1
876:          if ( II.gt.NDEAD ) go to 810
877:        800 continue
878:      810 do 820 I = KK, IFFL
879: CM      Y(I) = Y(I)*LLS(I)
880: CM      IF(JWWND.NE.0.AND.LLS(I).GT.1) Y(I) = W(I)
881: CM      LLS(I) = 1
882: CM      WCNTR(6) = WCNTR(6) + W(I)
883:      if ( W(I).lt.Y(I) ) then
884:        NCNTR(10) = NCNTR(10) + 1
885:        WCNTR(10) = WCNTR(10) + W(I)
886:      else
887:        NCNTR(6) = NCNTR(6) + 1
888:        WCNTR(6) = WCNTR(6) + W(I)
889:        Y(I) = W(I)
890:        LLS(I) = 1
891:      end if
892:    820 continue
893: CM      NCNTR(6) = NCNTR(6) + IFFL - KK + 1
894:      end if
895: C-----
896: C ..... FIND MAXIMUM NUMBER OF SPLITING .....
897: C-----
898:      MX = 0
899:      do 830 I = KFFL, IFFL
900:        LLS(I) = LLS(I) - 1
901:        MX = MAX(LLS(I),MX)
902:      830 continue
903: C-----
904: C ..... NOW WE CAN BEGINE SPLITTING !! .....
905: C-----
906:      II = 0
907:      do 840 I = KFFL, IFFL
908: CM      IF(LLS(I).GT.1) THEN
909:        WCNTR(5) = WCNTR(5) + Y(I)*LLS(I)
910:        NCNTR(5) = NCNTR(5) + LLS(I)

```

src/gmvp/search.f

```

911: CM          ENDIF
912:          WWW(LSFFL(I)) = Y(I)
913:      840      continue
914: C
915: C-----
916: C      ..... COPY SPLITTED PARTICLES IN BANK .....
917: C-----
918: CM          DO 820 M=2,MX
919:          do 860 M = 1, MX
920:              do 850 I = KFFL, IFFL
921:                  if ( LLS(I).ge.M ) then
922:                      II = II + 1
923:                      LSFFL(II) = LSDED(II)
924:                      IZFFL(II) = IZFFL(I)
925:                      WWW(LSFFL(II)) = Y(I)
926:                      IWK(II) = LSFFL(I)
927:                  end if
928:              850      continue
929:              860      continue
930:
931: *VOCL LOOP,NOVREC
932:          do 870 I = KFFL, IFFL
933:              K = LSFFL(II+I)
934:              XXX(K) = XXX(IWK(I))
935:              YYY(K) = YYY(IWK(I))
936:              ZZZ(K) = ZZZ(IWK(I))
937:              AAA(K) = AAA(IWK(I))
938:              BBB(K) = BBB(IWK(I))
939:              CCC(K) = CCC(IWK(I))
940:              IZZ(K) = IZZ(IWK(I))
941:              IGG(K) = IGG(IWK(I))
942:              TTT(K) = TTT(IWK(I))
943:              if ( JIMPT.ne.0 ) XIM(K) = XIM(IWK(I))
944:              if ( JLATT.ne.0 ) LEVL(K) = LEVL(IWK(I))
945:              if ( JTIME.ne.0 ) ITT(K) = ITT(IWK(I))
946:          870      continue
947: C
948:          if ( JLATT.ne.0 ) then
949:              do 890 NT = 1, NEST
950:                  do 880 I = KFFL, IFFL
951:                      K = LSFFL(II+I)
952:                      LZZ(K,NT) = LZZ(IWK(I),NT)
953:                      LPOS(K,NT) = LPOS(IWK(I),NT)
954:                      if ( JHLAT.ne.0 ) LCRS(K,NT) = LCRS(IWK(I),NT)
955:                      if ( JTLT.ne.0 ) IBSPC(K,NT) = IBSPC(IWK(I),NT)
956:                  880      continue
957:                  890      continue
958:              end if
959: C
960:          if ( II.gt.0 ) then
961: *VOCL LOOP,NOVREC
962:              do 900 I = II + 1, NDEAD
963:                  LSDED(I-II) = LSDED(I)
964:              900      continue
965:              NDEAD = NDEAD - II
966: CM          DO 840 I=1,II
967:              WCNTR(10) = WCNTR(10) + WWW(LSFFL(II+I))
968: CM          NCNTR(10) = NCNTR(10) + II
969:          end if
970: *VOCL LOOP,SCALAR
971:          do 910 I = KFFL, IFFL
972: CM          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + LLS(I) - 1
973:          NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + LLS(I)
974:          910      continue
975:          IFFL = IFFL + II

```

```

976:          end if
977: C
978: C-----
979: C      ..... UPDATING OF PARTICLE NUMBER IN STACKS .....
980: C-----
981:          NFFL(NZ1) = IFFL
982:          if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
983:          do 920 K = 1, NZ1
984:              NNXT(K) = 0
985:          920      continue
986:          return
987:          end

```


src/gmvp/select.f

```
1:      subroutine SELALZ( MACT, NZONE, NBANK, NHIST, NTGEN, NPART,
2:      &                  NGENE, NFFL, NNXT, NCOLS, NDEAD, NBREF,
3:      &                  NREFS, JREFL, JLATT, NXLT, NLBZ, JTERM )
4: C=====
5: C  PURPOSE: SELECT NEXT ACTION  (ALL ZONE GEOMETRY TRACKING)
6: C  CALLED IN: ACTION
7: C=====
8:      integer NFFL(1), NNXT(1), NBREF(1), NXLT(1), MMM(5)
9: C/#IF INTEGER8
10: *      integer*8 NPART, NTGEN
11: C/#ELSE
12:      integer NPART, NTGEN
13: C/#ENDIF
14: ccccc DATA   MMM /5*0 /
15: C
16: C .... IF ALL PARTICLES ARE DEAD, GENERATE NEXT PARTICLES ....
17: C
18:      if ( JTERM.eq.0.and.NBANK-NDEAD.le.0 ) then
19:          NGENE = MIN(NPART-NTGEN,NHIST)
20:          if ( NGENE.eq.0 ) go to 100
21:          MACT = 0
22:          MZONE = 0
23:          if ( NTGEN+NGENE.eq.NPART ) JTERM = 1
24:          return
25:      end if
26: C
27: C .... CHECK FLIGHT STACK .....
28: C
29:      100 MMM(1) = NFFL(NZONE+1)
30: C
31: C .... CHECK NEXT ZONE SEARCH STACK .....
32: C
33:      MMM(2) = NNXT(NZONE+1)
34: C
35: C .... CHECK COLLISION STACK .....
36: C
37:      MMM(3) = NCOLS
38: C
39: C .... CHECK REFLECTION STACK .....
40: C
41:      if ( JREFL.ne.0 ) then
42:          MMM(4) = NBREF(NREFS+1)
43:      else
44:          MMM(4) = 0
45:      end if
46: C
47: C .... CHECK LATTICE SEARCH STACK .....
48: C
49:      if ( JLATT.ne.0 ) then
50:          MMM(5) = NXLT(NLBZ+1)
51:      else
52:          MMM(5) = 0
53:      end if
54: C
55: C ....
56: C
57:      MMAX = 0
58: *VOCL LOOP,SCALAR
59:      do 110 M = 1, 5
60:          if ( MMM(M).gt.MMAX ) then
61:              MMAX = MMM(M)
62:              MACT = M
63:          end if
64:      110 continue
65:      if ( JTERM.eq.1.and.MMAX.eq.0 ) MACT = 99
66: C
67:      return
68:      end
```

src/gmvp/selone.f

```

1:      subroutine SELONE( MACT, JTERM, MZONE, NTGEN, NGENE,NGENE0,NGBAT,
2:      &                  H,      NFFL,
3:      &                  NNXT, NBREF, NXLT,  IMMODE,IMNFL, IMNNX,
4:      &                  IMNLT, NCNTR, WCNTR, NEVENT )
5: C=====
6: C PURPOSE: SELECT NEXT ACTION (ZONE-SELECTION GEOMETRY TRACKING)
7: C CALLED IN: ACTION
8: C
9: C <arguments: i=input o=output, w=work >
10: C
11: C o MACT : next action selected by this routine.
12: C   = 0 : GENERATE NEXT BATCH OF PARTICLES.
13: C   = 1 : FREE FLIGHT
14: C   = 2 : NEXT ZONE SEARCH
15: C   = 3 : COLLISION
16: C   = 5 : REFLECTION
17: C   = 6 : LATTICE SEARCH
18: C   = 7 : NEXT EVENT ESTIMATOR
19: C   = 88 : End of batch.
20: C   = 99 : END OF RUN
21: C o MZONE : target zone of next action ( =0 : all zones )
22: C o JTERM : set to 1 from 0 when all histories will be done after
23: C           the next batch.
24: C i JREPR : running in "NO-REPRODUCIBLE-RUN" mode if = 0
25: C           else running in "REPRODUCIBLE-RUN" mode.
26: C o IMMODE : action mode when MACT=7.
27: C
28: C
29: C (for other arguments, see description after common statements)
30: C
31: C=====
32: C/#IF INTEGER8
33: C *      integer*8 NTGEN
34: C/#ELSE
35: C      integer      NTGEN
36: C/#ENDIF
37: C
38: C      include 'INC/_FLAGS'
39: C      include '../shared/INC/_SIZES'
40: C      include 'INC/_STACK'
41: C      include 'INC/_XBANK'
42: C      include '../shared/INC/_IUNIT'
43: C      include '../shared/INC/_COUNTS'
44: C      include '../shared/INC/_TASKDT'
45: C
46: CCCC  real ELOOP
47: C      real H(*)
48: C
49: C      integer NFFL(NZONE+1), NNXT(NZONE+1), NBREF(NREFS+1), NXLT(NLBZ+1)
50: C
51: C      integer IMNFL(NZONE+1), IMNNX(NZONE+1), IMNLT(NLBZ+1)
52: C      real*8 NCNTR(NEVENT), WCNTR(NEVENT)
53: C
54: CCCC  integer JDEBG(*)
55: C
56: C      ... number of events for endless loop printout
57: C      parameter( MXEDLC = 30 )
58: C
59: C      .... data for debugging .....
60: C
61: C      parameter( NSAVEV = 30 )
62: C/#IF PARA( SX* )
63: C *      local common /SELDBG/
64: C/#ELSEIF PARA( CRAY* )
65: C *      task common /SELDBG/

```

```

66: C/#ELSE
67: C      common /SELDBG/
68: C/#ENDIF
69: C      &      MACTSV, MZONSV, MMAXSV, MMSV(7),      NFFLSV,
70: C      &      NNXTSV, IDEFSV, NBREFSV,      NXLTSV, NCOLSSV,
71: C      &      IMNFLSV, IMNNXSV,      IMNLTSV
72: C
73: C
74: C      .... LOCAL DATA
75: C
76: C      integer MMM(7), MZZ(7)
77: C      real*8 WENDL
78: C
79: Cccc DATA MZZ(3), MZZ(4),MMM(5),MMM(6) / 0,-1,0, 0 /
80: Cccc DATA MZZ(7) / 0 /
81: C
82: C-----
83: C
84: C=====
85: C .... Check consistency of stacks & bank ....
86: C=====
87: C
88: C      if ( JDEBG(5).ne.0 ) then
89: C
90: C          NSERR = 0
91: C          NALIVE = 0
92: C
93: C      ---- FLIGHT STACK ----
94: C
95: C          NALIVE = NALIVE + NFFL(NZONE+1)
96: C          NKKKK = 0
97: C          do 100 IZ = 1, NZONE
98: C              NKKKK = NKKKK + NFFL(IZ)
99: C          100 continue
100: C          if ( NFFL(NZONE+1).ne.NKKKK ) then
101: C              NSERR = NSERR + 1
102: C              write(IPR,7000) 'FLIGHT', NKKKK, NFFL(NZONE+1),
103: C              &              (NFFL(IZ),IZ=1,NZONE)
104: C          end if
105: C
106: C      ---- SEARCH STACK ----
107: C
108: C          NALIVE = NALIVE + NNXT(NZONE+1)
109: C          NKKKK = IDEFER
110: C          do 110 IZ = 1, NZONE
111: C              NKKKK = NKKKK + NNXT(IZ)
112: C          110 continue
113: C          if ( NNXT(NZONE+1).ne.NKKKK ) then
114: C              NSERR = NSERR + 1
115: C              write(IPR,7000) 'SEARCH', NKKKK, NNXT(NZONE+1),
116: C              &              (NNXT(IZ),IZ=1,NZONE), IDEFER
117: C          end if
118: C
119: C      ---- REFLECTION STACK ----
120: C
121: C          if ( JREFL.ne.0 ) then
122: C              NALIVE = NALIVE + NBREF(NREFS+1)
123: C              NKKKK = 0
124: C              do 120 IZ = 1, NREFS
125: C                  NKKKK = NKKKK + NBREF(IZ)
126: C              120 continue
127: C              if ( NBREF(NREFS+1).ne.NKKKK ) then
128: C                  NSERR = NSERR + 1
129: C                  write(IPR,7000) 'REFLECTION', NKKKK, NBREF(NREFS+1),
130: C                  &                  (NBREF(IZ),IZ=1,NREFS)

```

src/gmvp/selone.f

```

131:         end if
132:     end if
133: C
134: C ---- LATTICE-SEARCH STACK ----
135: C
136:         if ( JLATT.ne.0 ) then
137:             NALIVE = NALIVE + NXLT(NLBZ+1)
138:             NKKKK = 0
139:             do 130 IZ = 1, NLBZ
140:                 NKKKK = NKKKK + NXLT(IZ)
141: 130      continue
142:             if ( NXLT(NLBZ+1).ne.NKKKK ) then
143:                 NSERR = NSERR + 1
144:                 write(IPR,7000) 'REFLECTION', NKKKK, NXLT(NLBZ+1),
145:                 & (NXLT(IZ),IZ=1,NLBZ)
146:             end if
147:         end if
148: C
149: C --- collision stack ---
150: C
151:         NALIVE = NALIVE + NCOLS
152:         if ( NALIVE+NDEAD.ne.NBANK ) then
153:             NSERR = NSERR + 1
154:             write(IPR,7020) NALIVE, NDEAD, NBANK
155:         end if
156: C
157: C ---- imaginary particle for next event estimator ----
158: C
159:         if ( JPTDT.ne.0 ) then
160:             NALIVE = IMNFL(NZONE+1)
161:             NKKKK = 0
162:             do 140 IZ = 1, NZONE
163:                 NKKKK = NKKKK + IMNFL(IZ)
164: 140      continue
165:             if ( IMNFL(NZONE+1).ne.NKKKK ) then
166:                 NSERR = NSERR + 1
167:                 write(IPR,7000) 'NES-FLIGHT', NKKKK, IMNFL(NZONE+1),
168:                 & (IMNFL(IZ),IZ=1,NZONE)
169:             end if
170: C
171:             NALIVE = NALIVE + IMNNX(NZONE+1)
172:             NKKKK = 0
173:             do 150 IZ = 1, NZONE
174:                 NKKKK = NKKKK + IMNNX(IZ)
175: 150      continue
176:             if ( IMNNX(NZONE+1).ne.NKKKK ) then
177:                 NSERR = NSERR + 1
178:                 write(IPR,7000) 'NES-SEARCH', NKKKK, IMNNX(NZONE+1),
179:                 & (IMNNX(IZ),IZ=1,NZONE)
180:             end if
181:             if ( JLATT.ne.0 ) then
182:                 NALIVE = NALIVE + IMNLT(NLBZ+1)
183:                 NKKKK = 0
184:                 do 160 IZ = 1, NLBZ
185:                     NKKKK = NKKKK + IMNLT(IZ)
186: 160      continue
187:                 if ( IMNLT(NLBZ+1).ne.NKKKK ) then
188:                     NSERR = NSERR + 1
189:                     write(IPR,7000) 'NES-LATTICE', NKKKK, IMNLT(NLBZ+1),
190:                     & (IMNLT(IZ),IZ=1,NLBZ)
191:                 end if
192:             end if
193: C
194:             if ( NALIVE+NDIMPT.ne.IMPMAX ) then
195:                 NSERR = NSERR + 1

```

```

196:         write(IPR,7020) NALIVE, NDIMPT, IMPMAX
197:     end if
198: C
199: end if
200: C
201: if ( NSERR.gt.0 ) then
202:     write(IPR,7040) MACTSV, MZONSV, MMAXSV, MMSV
203:     if ( JPTDT.eq.0 ) then
204:         write(IPR,7060) 'STACKS ', ' NFFL ', ' NNXT ',
205:         & ' IDEFER ', ' NBREF ', ' NXLT ', ' NCOLS '
206:         write(IPR,7080) 'SAVED ', NFFLSV, NNXTSV, IDEFSV,
207:         & NBREFSV, NXLTSV, NCOLSSV
208:         write(IPR,7080) 'CURRENT ', NFFL(NZONE+1), NNXT(NZONE+1),
209:         & IDEFER, NBREF(NREFS+1), NXLT(NLBZ+1), NCOLS
210:     else
211:         write(IPR,7060) 'STACKS ', ' NFFL ', ' NNXT ',
212:         & ' IDEFER ', ' NBREF ', ' NXLT ', ' NCOLS ',
213:         & ' IMNFL ', ' IMNNX '
214:         write(IPR,7080) 'SAVED ', NFFLSV, NNXTSV, IDEFSV,
215:         & NBREFSV, NXLTSV, NCOLSSV, IMNFLSV, IMNNXSV
216:         write(IPR,7080) 'CURRENT ', NFFL(NZONE+1), NNXT(NZONE+1),
217:         & IDEFER, NBREF(NREFS+1), NXLT(NLBZ+1), NCOLS,
218:         & IMNFL(NZONE+1), IMNNX(NZONE+1)
219:     end if
220: end if
221: C
222: 7000      format(1X,'XXX(SELONE) ',A,' stack inconsistency : sum ',I8,
223:         & ' logged sum ',I8/(:1X,3X,10(:I7)))
224: 7020      format(1X,'XXX(SELONE) Sum of number of alive particles(=',I8,
225:         & ') and NDEAD (=',I10,') does not match NBANK (=',I8)
226: 7040      format(1X,'XXX(SELONE) Particle stack inconsistency detected: ',
227:         & ' the last action ID is ',I3,'. saved parameters :/1X,
228:         & ' MZONE ',I6,' MMAX ',I8/1X,' MMM ',10(I8:))
229: 7060      format(1X,A,3X,10(A:))
230: 7080      format(1X,A,1X,10(I8:))
231: C
232: end if
233: C
234: C=====
235: C .... IF ALL PARTICLES ARE DEAD, GENERATE NEXT PARTICLES ....
236: C=====
237: C
238: CM
239: 170 continue
240: CM
241: CSASA IF(jterm.EQ.0 .AND. NBANK-NDEAD.LE.NHIST*SUPPLY) THEN
242: C
243:     if ( JTERM.eq.0.and.NDEAD.ge.NBANK ) then
244: C
245: C
246:         if ( JREPR.eq.0 ) then
247: C
248: C ..... PROCESS REMAINING NEXT EVENT ESTIMATOR TASK UNTIL FINISHED ...
249: C
250:         if ( JPTDT.ne.0.and.NIMPT.ne.0 ) then
251:             MACT = 7
252:             MZONE = 0
253:             IMMODE = 1
254:             go to 240
255:         end if
256: C
257: C ... end of all history
258: C
259:         if ( JTERM.eq.1 ) then
260:             MACT = 99

```

src/gmvp/selone.f

```

261:          goto 240
262:      end if
263:  C
264:      call REQHST( NGENE, JTSKR )
265:  C
266:      if ( NGENE.eq.0 ) then
267:          MACT = 99
268:          MZONE = 0
269:          JTERM = 1
270:      else
271:          MACT = 0
272:          MZONE = 0
273:      end if
274:      go to 240
275:  C
276:  else
277:  C
278:  CCCC      NGENE = MIN(NPART-NTGEN,NHIST-NBANK+NDEAD)
279:  C          NGENE = MIN(NGBAT,NHSUB)
280:  C
281:  CCCCC      if ( NGENE.eq.0 ) go to 180
282:  C          if ( NGENE.eq.0 ) then
283:  C              MACT = 88
284:  C              go to 240
285:  C          end if
286:  C
287:  C      .... PROCESS REMAINING NEXT EVENT ESTIMATOR TASK UNTIL FINISHED ...
288:  C
289:  C          if ( JPTDT.ne.0.and.NIMPT.ne.0 ) then
290:  C              MACT = 7
291:  C              MZONE = 0
292:  C              IMMODE = 1
293:  C              go to 240
294:  C          end if
295:  C
296:  C      ... end of all history
297:  C
298:  C          if ( JTERM.eq.1 ) then
299:  C              MACT = 99
300:  C              goto 240
301:  C          end if
302:  C
303:  C          MACT = 0
304:  C          MZONE = 0
305:  C          if ( NTGEN+NGENE.eq.NPART ) JTERM = 1
306:  C          go to 240
307:  C      end if
308:  C  end if
309:  C
310:  C
311:  C
312:  C      180 MM = 0
313:  C      MZ = 0
314:  C
315:  C      .... CHECK FLIGHT STACK .....
316:  C
317:  C          if ( NFFL(NZONE+1).gt.0 ) then
318:  C              if ( JVMNT.ne.0 ) call VMNTR1( 2, NZONE )
319:  C          *VOCL LOOP,NOVREC
320:  C              do 190 KZ = 1, NZONE
321:  C                  if ( NFFL(KZ).gt.MM ) then
322:  C                      MM = NFFL(KZ)
323:  C                      MZ = KZ
324:  C                  end if
325:  C              190 continue

326:      end if
327:      MMM(1) = MM
328:      MZZ(1) = MZ
329:  C
330:  C      .... CHECK NEXT ZONE SEARCH STACK .....
331:  C
332:      MM = 0
333:      MZ = 0
334:      if ( NNXT(NZONE+1).gt.0 ) then
335:  *VOCL LOOP,NOVREC
336:          do 200 KZ = 1, NZONE
337:              if ( NNXT(KZ).gt.MM ) then
338:                  MM = NNXT(KZ)
339:                  MZ = KZ
340:              end if
341:          200 continue
342:      end if
343:      MMM(2) = MM
344:      MZZ(2) = MZ
345:  C
346:      do 210 I = 3, 7
347:          MMM(I) = 0
348:          MZZ(I) = 0
349:      210 continue
350:  C
351:  C      .... CHECK COLLISION STACK .....
352:  C
353:      MLL0 = (NBANK-NDEAD)*0.5
354:      if ( NCOLS.gt.MLL0 .or. MMM(1).eq.0.and.MMM(2).eq.0 )
355:      &    MMM(3) = NCOLS
356:  C
357:  C      .... CHECK DEFERRED PARTICLE .....
358:  C
359:      MMM(4) = IDEFER
360:      MZZ(4) = -1
361:  C
362:  C      .... CHECK REFLECTION STACK .....
363:  C
364:      if ( JREFL.ne.0 ) then
365:          if ( NBREF(NREFS+1).gt.MMM(3) ) MMM(5) = NBREF(NREFS+1)
366:      end if
367:  C
368:  C      .... CHECK LATTICE-SEARCH STACK .....
369:  C
370:      if ( JLATT.ne.0 ) then
371:          if ( NXLT(NLBZ+1).gt.MMM(3) ) MMM(6) = NXLT(NLBZ+1)
372:      end if
373:  C
374:  C      .... CHECK NEXT EVENT ESTIMATOR STACK .....
375:  C
376:      if ( JPTDT.ne.0 ) then
377:          MM = 0
378:  *VOCL LOOP,NOVREC
379:          do 220 KZ = 1, NZONE
380:              if ( IMNFL(KZ).gt.MM ) then
381:                  MM = IMNFL(KZ)
382:              end if
383:          220 continue
384:  *VOCL LOOP,NOVREC
385:          do 225 KZ = 1, NZONE
386:              if ( IMNNX(KZ).gt.MM ) then
387:                  MM = IMNNX(KZ)
388:              end if
389:          225 continue
390:          if ( JLATT.ne.0 ) then

```

src/gmvp/selone.f

```

391:         if ( IMNLT(NLBZ+1).gt.MLL0.or.MM.eq.0 )
392:         &      MM = IMNLT(NLBZ+1)
393:         end if
394:         IMMODE = -1
395:         MMM(7) = MM
396:         end if
397: C
398: C ....
399: C
400:         MMAX = 0
401:         MACT = 0
402: *VOCL LOOP, SCALAR
403:         do 230 M = 1, 7
404:         if ( MMM(M).gt.MMAX ) then
405:             MMAX = MMM(M)
406:             MACT = M
407:             MZONE = MZZ(M)
408:         end if
409: 230 continue
410: C
411:         if ( JTERM.eq.1.and.MMAX.eq.0 ) MACT = 99
412: C
413:         if ( MACT.eq.4 ) MACT = 2
414: C
415: C
416: C -----
417: C DETECTION OF ENDLESS RANDOM WALK
418: C -----
419: C
420: C JLOOP = 0 : NON-DETECTION MODE
421: C
422: C         NUMBER OF REMAINING PARTICLES ARE MORE THAN 1% OF 'NHIST'.
423: C
424: C JLOOP = 1 : DETECTION MODE
425: C
426: C         IF IN A BATCH
427: C         NUMBER OF REMAINING PARTICLES ARE LESS THAN 2 OR 1% OF 'NHIST'.
428: C
429: C         SUM OF PROCESSED-PARTICLES IN DETECTION MODE
430: C         WHEN ----- > ELOOP
431: C         SUM OF PROCESSED-PARTICLES IN THE BATCH
432: C
433: C         ENTER MONITORING-MODE.
434: C
435: C JLOOP = 2 : MONITORING MODE
436: C
437: C         Print selected event, zone etc. upto MXEDLC times.
438: C         and terminate the batch.
439: C
440: C
441: C
442: C980630 CCC
443: CCC if ( MACT.ne.0.and.MACT.ne.99 ) then
444:
445:         if ( MACT.ne.0.and.MACT.ne.99.and.MACT.ne.7 ) then
446: C
447: C         .... SUMMATION OF NUMBER OF PROCESSED PARTICLES IN THIS BATCH ..
448: C
449:         DESUM = DESUM + MMAX
450: C
451: C
452: CCC if ( JLOOP.eq.0.and.NBANK-NDEAD.le.MAX(INT(NHIST*0.01),2) )
453: C
454: C         ... NGBAT is remaining number of histories for this batch.
455: C

```

```

456: C         <<comment>> when NHSUB < NHIST, a batch includes more than
457: C         one sub-batches (new history control layer introduced in
458: C         Feb 2000) and this endless-loop detection might not
459: C         work for earlier sub-batches in a batch ...
460: C
461:         if ( JLOOP.eq.0.and.
462:         &      NGBAT+(NBANK-NDEAD).le.MAX(INT(NHIST*0.01),2) )
463:         &      then
464:             DESUM2 = 0.0
465:             JLOOP = 1
466:         end if
467: C
468:         if ( JLOOP.eq.1 ) then
469:             DESUM2 = DESUM2 + MMAX
470:             if ( ELOOP.gt.0.0.and.DESUM2/DESUM.gt.ELOOP ) then
471: C/#IF PARA(SX* CRAY)
472: *         call MVPSYNC_LOCK(2)
473: C/#ENDIF
474:             write(IOW,'(/1X,A,I4,A,I3,A/)') 'XXX ', NBANK - NDEAD,
475:             &      ' particles are probably '//
476:             &      'in endless loop. ( monitor ', MXEDLC,
477:             &      ' events )'
478: C/#IF PARA(SX* CRAY)
479: *         call MVPSYNC_UNLOCK(2)
480: C/#ENDIF
481:             JLOOP = 2
482:             NLOOP = 0
483:         end if
484:         end if
485: C
486: C         .... PROBABLE ENDLESS LOOP .....
487: C
488: C         ( MONITOR PRINT UPTO 30 EVENTS )
489: C
490:         if ( JLOOP.eq.2 ) then
491:             NLOOP = NLOOP + 1
492: C
493: C980630ccc if ( NLOOP.lt.30 ) then
494:             if ( NLOOP.lt.MXEDLC.and.MACT.ne.7 ) then
495:                 write(IOW,*) ' EVENT :', MACT, ' MZONE ', MZONE,
496:                 &      ' MMAX ', MMAX
497:             else if ( MMM(7).gt.0.and.NLOOP.eq.MXEDLC ) then
498:                 MACT = 7
499:                 MZONE = 0
500:                 IMMODE = 1
501:             else
502: C
503: C         .... More than MXEDLC events monitored. terminate batch. ....
504: C
505: C/#IF PARA(SX* CRAY)
506: *         call MVPSYNC_LOCK(2)
507: C/#ENDIF
508:             write(IOW,'(/1X,A,I5,A/)') 'XXX PROBABLY ', NBANK
509:             &      - NDEAD,
510:             &      ' PARTICLES ARE IN ENDLESS LOOP AND KILLED.'
511: C/#IF PARA(SX* CRAY)
512: *         call MVPSYNC_UNLOCK(2)
513: C/#ENDIF
514: C
515: C         .... CLEAR EVENT STACKS HERE ....
516: C
517:             NDEAD = NBANK
518:             call PUTVI( NFFL, NZONE+1, 0 )
519:             call PUTVI( NNXT, NZONE+1, 0 )
520:             if ( JREFL.ne.0 ) call PUTVI( NBREF, NREFS+1, 0 )

```

src/gmvp/selone.f

```

521:          if ( JLATT.ne.0 ) call PUTVI( NXLT, NLBZ+1, 0 )
522: C 980630 CCC
523:          if ( JPTDT.ne.0 ) then
524:            call PUTVI( IMNFL, NZONE+1, 0 )
525:            call PUTVI( IMNNX, NZONE+1, 0 )
526:            if ( JLATT.ne.0 ) call PUTVI( IMNLT, NLBZ+1, 0 )
527:            NIMPT = 0
528:          end if
529: C
530:          NCOLS = 0
531:          IDEFER = 0
532:          if ( JTERM.eq.1 ) then
533:            MACT = 99
534:          else
535:            go to 170
536:          end if
537:        end if
538:      end if
539:    end if
540: C
541: C .... END OF SELONE ...
542: C
543: C 240 continue
544: C
545: C .... RESET ENDLESS-LOOP MONITORING PARAMETERS ....
546: C
547: CCCC if ( MACT.eq.0 .or. MACT.eq.99 ) then
548:       if ( MACT.eq.88 .or. MACT.eq.99 ) then
549:         DESUM = 0.0
550:         DESUM2 = 0.0
551:         JLOOP = 0
552:         NLOOP = 0
553:       end if
554: C
555: C
556: C ... print selection information fro debugging ...
557: C
558:       if ( JDEBG(4).ge.2 ) then
559:         write(IOW,7100) MACT, MZONE, MMAX, MMM
560: 7100    format(1X,'(SELONE) mact ',I2,' mzone',I5,' mmax',I5,' mmm',10
561: &          (:I5))
562:       end if
563: C
564:       if ( JDEBG(4).ge.3 ) then
565:         write(IOW,'(1X,2(a,i7))') 'NCOLS', NCOLS, 'NDEAD', NDEAD
566:         write(IOW,7120) 'NFFL', NFFL
567:         write(IOW,7120) 'NNXT', NNXT
568:         if ( JLATT.ne.0 ) write(IOW,7120) 'NXLT', NXLT
569:         if ( JREFL.ne.0 ) write(IOW,7120) 'NBREF', NBREF
570:         if ( JPTDT.ne.0 ) then
571:           write(IOW,7120) 'IMNFL', IMNFL
572:           write(IOW,7120) 'IMNNX', IMNNX
573:           if ( JLATT.ne.0 ) write(IOW,7120) 'IMNLT', IMNLT
574:         end if
575: 7120    format(1X,'<',A,'>',10I7/(5X,10I7))
576:       end if
577: C
578: C ... save MACT etc. for debug output ....
579: C
580:       if ( JDEBG(5).ne.0 ) then
581:         MACTSV = MACT
582:         MZONSV = MZONE
583:         MMAXSV = MMAX
584:         do 250 K = 1, 7
585:           MMMSV(K) = MMM(K)
586: 250    continue
587: C
588:         NFFLSV = NFFL(NZONE+1)
589:         NNXTSV = NNXT(NZONE+1)
590:         IDEFSV = IDEFER
591:         if ( JREFL.ne.0 ) then
592:           NBREFSV = NBREF(NREFS+1)
593:         else
594:           NBREFSV = 0
595:         end if
596:         if ( JLATT.ne.0 ) then
597:           NXLTSV = NXLT(NLBZ+1)
598:         else
599:           NXLTSV = 0
600:         end if
601:         NCOLSSV = NCOLS
602: C
603:         if ( JPTDT.ne.0 ) then
604:           IMNFLSV = IMNFL(NZONE+1)
605:           IMNNXSV = IMNNX(NZONE+1)
606:           if ( JLATT.ne.0 ) IMNLTSV = IMNLT(NLBZ+1)
607:         else
608:           IMNFLSV = 0
609:           IMNNXSV = 0
610:           IMNLTSV = 0
611:         end if
612: C
613:       end if
614: C
615:       return
616:     end

```

src/gmvp/source.f

```

1: C/#IF .NOT.SOURCE(NEW)
2:   subroutine SDUMMY( )
3:     return
4: C/#ELSE
5:
6: C/#IF ARGSAVE
7: *   subroutine SOURC0(IOW, A, H, DH,
8: *   N NTGEN, NDEAD, NFISB, JBPRNT,NLOST, NMKREG,
9: *   B XXX, YYY, ZZZ, AAA, BBB, CCC, WWW,
10: *   B IZZ,IGG,TTT,ITT,KLSF,IBREG,IBSPC, XXXF,YYYY,
11: *   B ZZZF,IZZF,KLSFF, LEVL, LZZ,
12: *   B LPOS, LCRS, LEVLF, LZZF, LPOSF,
13: *   B LCRSF,IBRGF,IBSPF, XIM,
14: *   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
15: *   B DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
16: *   G MLBZZ, KZMAT, KZREG, MEMZN, SDA,
17: *   G KZDA,KZAA,IPCEL,LTYP,XIMP, LSFFL, NFFL,
18: *   S IZFFL, LSDED, LSLAT, IZLAT, NXLT,
19: *   * SOUR, SRCSP, FKAI, KKAI, NTGX,
20: *   T WSUM, XSOC, NCNTR,WCNTR, WGTf, ENGYB,ENGPB,TIMEB,
21: *   T XSXV,XAVT,AAVT,EAVT,
22: *   W LXYZ,LPWRK,LVSTK,MWVEC,MVSTK,MXREJ,
23: *   W R,IWK1,IWK2,RWK1,X,Y,Z, IFL,IFI, NNSRC, XSOUR, RWF )
24: C/#ELSE
25:   subroutine SOURCE(IOW, A, H, DH,
26:   N NTGEN, NDEAD, NFISB, JBPRNT,NLOST, NMKREG,
27:   B XXX, YYY, ZZZ, AAA, BBB, CCC, WWW,
28:   B IZZ,IGG,TTT,ITT,KLSF,IBREG,IBSPC, XXXF,YYYY,
29:   B ZZZF,IZZF,KLSFF, LEVL, LZZ,
30:   B LPOS, LCRS, LEVLF, LZZF, LPOSF,
31:   B LCRSF,IBRGF,IBSPF, XIM,
32:   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
33:   B DFBK,KDFBK,MDFBK, IFBK, KIFBK,MIFBK,
34:   G MLBZZ, KZMAT, KZREG, MEMZN, SDA,
35:   G KZDA,KZAA,IPCEL,LTYP,XIMP, LSFFL, NFFL,
36:   S IZFFL, LSDED, LSLAT, IZLAT, NXLT,
37:   * SOUR, SRCSP, FKAI, KKAI, NTGX,
38:   T WSUM, XSOC, NCNTR,WCNTR, WGTf, ENGYB,ENGPB,TIMEB,
39:   T XSXV,XAVT,AAVT,EAVT,
40:   W LXYZ,LPWRK,LVSTK,MWVEC,MVSTK,MXREJ,
41:   W R, IWK1,IWK2,RWK1, X,Y,Z, IFL,IFI, NNSRC, XSOUR,RWF,
42:   % NPDET, NPLEN, NIMPT, NDIMPT,JPUSD,
43:   % JSPDT, XPDET, IPDET, IPDT2, STOTX, VEL,
44:   S IMSFLL, IMNFL, IMZFL,
45:   S IMDED, IMSLT, IMNLT, IMZLT,
46:   B LZZI, LPOSI, LCRSI,
47:   B OPTI, PATH, KDET, KLSFI,
48:   % NGENE, NGENE0, NHIST1, NGBAT, IFISB, WFFACT, WSUMB )
49: C/#ENDIF
50: C=====
51: C PURPOSE: GENERATE PARTICLES (new type)
52: C CALLED IN: ACTION
53: C CALLS: JUDGE SYSSRC RANU2 BSVDEC SUNCL VMNTR1
54: C=====
55: C
56:   implicit real*8(D)
57: C
58:   include 'INC/_FLAGS'
59:   include '../shared/INC/_TASKDT'
60:   include '../shared/INC/_WORDL'
61: C
62:   include '../shared/INC/_SIZES'
63: C
64: C ... include _FBANK2 to access XXXF2, YYYY2 etc. in FISSRC
65: C

```

```

66:   include 'INC/_FBANK2'
67: C
68: C ... head position of dynamic memory area ....
69: C
70:   real A(*)
71:   real H(*)
72:   real*8 DH(*)
73: C
74: C/#IF INTEGER8
75: *   integer*8 NTGEN
76: C/#ELSE
77:   integer NTGEN
78: C/#ENDIF
79:   integer IFISB(NHIST)
80:   real*8 WFFACT
81: C
82: C .... PARTICLE BANK .....
83: C
84:   real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
85:   real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
86:   real WWW(NBANK), XIM(NBANK)
87:   real*8 TTT(NBANK)
88:   integer ITT(NBANK)
89:   integer IZZ(NBANK), IGG(NBANK)
90: C
91:   integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST),
92:   & LCRS(NBANK,NEST), KLSF(NBANK)
93: C
94:   integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
95: C
96:   real*8 DBNK(NBANK,*)
97:   integer KDBNK(0:MDBNK)
98:   integer IBNK(NBANK,*)
99:   integer KIBNK(0:MIBNK)
100: C
101: C .... FISSION BANK .....
102: C
103:   real*8 XXXF(NFBNK0), ZZZF(NFBNK0), YYYY(NFBNK0)
104:   integer IZZF(NBANK)
105: C
106:   integer LEVLF(NFBNK0), LZZF(NFBNK0,NEST), LPOSF(NFBNK0,NEST),
107:   & LCRSF(NFBNK0,NEST), KLSFF(NFBNK0)
108: C
109:   integer IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
110: C
111: C ... optional bank parameters
112: C
113:   real*8 DFBK(NFBNK0,*)
114:   integer KDFBK(0:MDBNK)
115:   integer IFBK(NFBNK0,*)
116:   integer KIFBK(0:MIBNK)
117: C
118: C .... TALLY .....
119: C
120:   real*8 WSUM, XSOC(NLENG), XSXV(NLENG,3), WSUMB, RSRC
121:   real*8 NCNTR(*), WCNTR(*)
122:   real XAVT(3), AAVT(3)
123: C
124: C .... VARIANCE REDUCTION .....
125: C
126:   real XIMP(NGROUP,NREG)
127: C
128: C .... STACKS ...(FREE FLIGHT & MORGUE) .....
129: C
130:   integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)

```

src/gmvp/source.f

```

131:      integer LSDED(NBANK)
132: C
133: C .... STACKS ...(LATTICE SEARCH) .....
134: C
135:      integer LSLAT(NBANK), NXLT(*), IZLAT(NBANK)
136: C
137: C .... WORKING ARRAYS   ( LENGTH OF R MUST BE  4*NBANK FOR USE
138: C                      IN FNSSOC )
139: C      R and IFL can be used as real*8 data
140:      real R(6*NBANK), RWK1(NBANK)
141:      integer IWK1(NBANK), IWK2(NBANK)
142:      logical IFL(NBANK), IFI(NBANK)
143:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
144: C
145:      real*8 XSOUR(NSOUR)
146:      integer NNSRC(NSOUR)
147:      real RWF(NHIST)
148: C
149: C .... SOURCE DATA .....
150: C
151:      integer SRCSP(*)
152:      integer MEMZN(NMEMS,NSOUR)
153:      real*8 SOUR(NSOUR)
154: C
155: C .... FISSION NEUTRON DATA .....
156: C
157:      integer KKAI(2,NTGX,NMAT)
158:      real FKAI(NTGX,NMAT), WGTF(NREG)
159: C
160: C .... GEOMETRY ARRAY DATA .....
161: C
162:      integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
163:      integer MLBZZ(NZONE), IPCEL(*), LTP(* )
164: C
165:      real*8 SDA(*)
166: C
167: C 6/19/1991  **** ADDED ARGUMENTS FIXED BY ENTRY 'SOURCE'
168: C              (FOR NEXT EVENT ESTIMATOR OPTION)
169: C
170: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR
171: C
172:      real*8 XPDET(NPLEN,NPDET)
173:      integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
174:      integer JPUSD(NPDET), JSPDT(NPDET)
175: C
176: C**** total cross section ( added in Jun, 1998 )
177: C
178:      real STOTX(NTGX,*)
179: C
180: C**** TALLY BIN DATA
181: C
182:      real ENGYB(NGP1+1)
183:      real ENGPB(NGP2+1)
184:      real TIMEB(NTIME+1)
185:      real VEL(NGROUP)
186: C
187: C**** STACK FOR IMAGINARY PARTICLE
188: C
189:      integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(*), IMDED(IMPMAX),
190:      &      IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
191: C
192: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
193: C
194:      integer LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
195: C****

```

```

196:      real*8 OPTI(IMPMAX), PATH(IMPMAX)
197:      integer KDETP(IMPMAX), KLSFI(IMPMAX)
198: C
199: C**** WORKING AREA
200: C
201: C      ... pointers to working array
202: C
203:      integer LXYZ(10)
204: C
205: C ----> integer      lx,ly,lz,la,lb1,lc, le,lg,lt,lw
206: C
207:      integer LPWRK(MWVEC), LVSTK(MVSTK)
208: C
209:      include '../shared/INC/_PMLATT'
210: C
211: C --- local data ----
212: C
213:      integer JVSMP(10)
214: C
215: C      ... local data whose value must be saved call to call
216: C
217: C/#IF PARA(SX*)
218: *      LOCAL COMMON /LSSOUR/ XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN
219: C/#ELSEIF PARA(CRAY)
220: *      TASK COMMON /LSSOUR/ XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN
221: C/#ENDIF
222:      data XAV, YAV, ZAV, AAV, BAV, CAV, EAV, DWSUMN /8*0.0d0/
223: C
224: C --- statement functions
225: C
226:      include '../shared/INC/_SFLATT'
227: C
228: C --- statement function to convert pointer to  real data to
229: C      pointer to real*8 data
230: C
231: C/#IF WORD(64)
232: *      LDBLE(L)      = L
233: C/#ELSE
234: *      LDBLE(L)      = L/MWORD + MWORD - 1
235: C/#ENDIF
236: C
237: C/#IF ARGSAVE
238: *      return
239: C
240: C -----
241: C
242: *      entry SOURCU( NPDET, NPLEN, NIMPT, NDIMPT, JPUSD,
243: *      D      JSPDT, XPDET, IPDET, IPDT2, STOTX, VEL,
244: *      S      IMSFL, IMNFL, IMZFL,
245: *      S      IMDED, IMSLT, IMNLT, IMZLT,
246: *      B      LZZI , LPOSI, LCRSI,
247: *      B      OPTI , PATH , KDETP, KLSFI  )
248: C
249: *      return
250: C
251: C -----
252: *      entry SOURCE( NGENE , NGENEO ,NHIST1, NGBAT, IFISB, WFFACT,
253: *      &      WSUMB )
254: C
255: C/#ENDIF
256: C
257: C
258: C-----
259: C .... IF ALL PARTICLES ARE DEAD, CLEAR UP DEAD PARTICLE STACK ....
260: C-----

```


src/gmvp/source.f

```

261:      if ( NDEAD.eq.NBANK ) then
262:        do 100 I = 1, NBANK
263:          LSDED(I) = I
264:        100 continue
265:      C
266:        NDIMPT = IMPMAX
267:        if ( JPTDT.ne.0 ) then
268:          do 110 I = 1, IMPMAX
269:            IMDED(I) = I
270:          110 continue
271:        end if
272:      C
273:      else
274:        write(IOW,*) ' !!!(SOURCE) NDEAD IS NOT EQUAL TO NBANK !!!',
275:        & ' NDEAD=', NDEAD, ' NBANK=', NBANK, ' NTGEN= ', NTGEN
276:        write(IOW,*) ' SOMETHING WRONG HAPPENED, STOP !!!'
277:        stop 666
278:      end if
279: C-----
280: C .... Print number of histories processed in a batch for first
281: C sub-batch only.
282: C-----
283:      if ( NGENE0.eq.0 ) then
284:        C/#IF PARA
285:        C/#IF PARA(SX* CRAY)
286:        * call MVPSYNC_LOCK(2)
287:      C/#ENDIF
288:      if ( JBPRNT.ne.0 ) then
289:        c write(IOW,7000) IDTASK, NHIST1, IRAND
290:        write(IOW,7000) IDTASK, NHIST1
291:      end if
292:      C/#IF PARA(SX* CRAY)
293:      * call MVPSYNC_UNLOCK(2)
294:      C/#ENDIF
295:      C/#ELSE
296:      if ( JBPRNT.ne.0 ) then
297:        c write(IOW,7020) NHIST1, IRAND
298:        write(IOW,7020) NHIST1
299:      end if
300:      C/#ENDIF
301:      C
302:      c7000 format(1x,' TASK ',I5,' GENERATE ',I6,' SOURCES. IRAND = ',
303:      & I11)
304:      7000 format(1x,' TASK ',I5,' GENERATE ',I6,' SOURCES.')
305:      c7020 format(1x,' GENERATE ',I6,' SOURCES. IRAND = ',I11)
306:      7020 format(1x,' GENERATE ',I6,' SOURCES.')
307:      C
308:      C ... select fission source of next batch for eigenvalue problem ...
309:      C
310:      if ( JEIGN.ne.0.and.NBATCH.gt.0 ) then
311:        call FISSET( IOW, IFISB, WFFACT, IRAND, NFISB, NHIST,
312:        & NHIST1, NHSUB, NBANK, NZONE, NFBANK, NFBNK0, NBATCH,
313:        & NREG, NEST, WGTF, KZREG, XSOC, NLENG, XXXF, YYF,
314:        & ZZZF, IZZF, LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF,
315:        & KLSFF, DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK,
316:        & H(LXXXF2), H(LYYYF2), H(LZZZF2), H(LIZZF2),
317:        & H(LLEVLF2), H(LLZZF2), H(LLPOSF2), H(LLCRSF2),
318:        & H(LIBRGF2), H(LIBSPF2), H(LKLSFF2), H(LDFBK2),
319:        & KDFBK2, MDFBK2, H(LIFBK2), KIFBK2, MIFBK2, RWF(1) )
320:      end if
321:      C
322:      C ... initialize variables for batchwise source parameter average
323:      C
324:      XAV = 0.0D0
325:      YAV = 0.0D0

```

```

326:      ZAV = 0.0D0
327:      AAV = 0.0D0
328:      BAV = 0.0D0
329:      CAV = 0.0D0
330:      EAV = 0.0D0
331:      end if
332:      C
333:      C-----
334:      C ..... NGENE0 : MEMORIZE NGENE FOR USE IN TALSUM .....
335:      C-----
336:      CCC NGENE0 = NGENE
337:      NFB0 = NGENE0
338:      NGENE0 = NGENE0 + NGENE
339:      NGBAT = NGBAT - NGENE
340:      C
341:      ILOST = 0
342:      if ( JVMNT.ne.0 ) call VMNTRI( 1, NGENE )
343:      C
344:      C
345:      C
346:      C =====
347:      C FIXED SOURCE PROBLEM OR INITIAL SOURCE FOR EIGENVALUE PROBLEM
348:      C =====
349:      C
350:      C
351:      C
352:      if ( JEIGN.eq.0 .or. NBATCH.le.0 ) then
353:      C-----
354:      C .... DETERMINE PARTICLE NUMBERS FROM EACH SOURCE IF NSOUR > 1 .....
355:      C-----
356:      if ( NSOUR.gt.1 ) then
357:        NSS = 0
358:      *VOCL LOOP,SCALAR
359:      do 120 N = 1, NSOUR
360:        NNSRC(N) = NGENE*SOUR(N)
361:        XSOUR(N) = FLOAT(NGENE)*SOUR(N) - NNSRC(N)
362:        if ( N.gt.1 ) XSOUR(N) = XSOUR(N-1) + XSOUR(N)
363:        NSS = NSS + NNSRC(N)
364:      120 continue
365:      C-----
366:      C .... IF THE SUM OF NNSRC IS LESS THAN NGENE, THE REST OF SOURCE
367:      C PARTICLES ARE GENERATED WITH PROBABILITIES PROPORTIONAL TO THE
368:      C DECIMAL PARTS OF NGENE*SOUR OF EACH SOURCE. (XSOUR).
369:      C-----
370:      NSS1 = NGENE - NSS
371:      if ( NSS1.gt.0.0 ) then
372:        call RANU2( IRAND, R, NSS1, ICON )
373:      *VOCL LOOP,SCALAR
374:      do 150 K = 1, NSS1
375:        R(K) = R(K)*XSOUR(NSOUR)
376:        do 130 L = 1, NSOUR
377:          if ( XSOUR(L).gt.R(K) ) go to 140
378:        130 continue
379:        L = NSOUR
380:        140 NNSRC(L) = NNSRC(L) + 1
381:        150 continue
382:      C
383:      C ... sum of NNSRC > NGENE: reject NSS-NGENE sources
384:      C
385:      else
386:        call RANU2( IRAND, R, ABS(NSS1), ICON )
387:        NSS2 = NSS
388:        do 180 K = 1, ABS(NSS1)
389:          IIKIL = MIN(INT(R(K)*NSS2+1),NSS2)
390:          NSS3 = 0
391:          do 160 N = 1, NSOUR

```

src/gmvp/source.f

```

391:          NSS3 = NSS3 + NNSRC(N)
392:          if ( NSS3.ge.IIKIL ) then
393:            NNSRC(N) = NNSRC(N) - 1
394:            go to 170
395:          end if
396:          160          continue
397:          170          NSS2 = NSS2 - 1
398:          180          continue
399:        end if
400:      else
401:        NNSRC(1) = NGENE
402:      end if
403: C-----
404: C .... GENERATE PARTICLES .....
405: C-----
406:      NSS = 0
407:      IST = 0
408:      IPP = NDEAD
409: C
410:      LXD = LDBLE(LXYZ(1)) - 1
411:      LYD = LDBLE(LXYZ(2)) - 1
412:      LZD = LDBLE(LXYZ(3)) - 1
413:      LAD = LDBLE(LXYZ(4)) - 1
414:      LBD = LDBLE(LXYZ(5)) - 1
415:      LCD = LDBLE(LXYZ(6)) - 1
416:      LED = LDBLE(LXYZ(7)) - 1
417:      LGD = LDBLE(LXYZ(8)) - 1
418:      LTD = LDBLE(LXYZ(9)) - 1
419:      LWD = LDBLE(LXYZ(10)) - 1
420:
421:      do 420 N = 1, NSOUR
422:
423:        NN = NNSRC(N)
424: Ccccccccccccc KS = KSOUR(N)
425: C temporary ....
426:        KS = 0
427: Ccc          IF(JMNTR.NE.0)
428: Ccc &          WRITE(IOW,*) ' SOURCE # ',N,': ',NN,' PARTICLES'
429:          IPP = IPP - NN
430: C
431: C
432: Ccccccccccccc nwrk = 5
433:          NRWK = 5
434: Ccccccccccccc MXREJ = 300
435: C
436: C ... subroutines under 'SYSSRC' can use working array other than
437: C      nnsrc,xsour & iwk1, iwk2
438: C
439: C
440:          call SYSSRC( 'GMVP', A, N, NN, NPK, JDEBG, JLATT, JTLT,
441:          &          JHLAT, SRCSP, SRCSP, SRCSP, MXREJ, H, H, DH, IRAND,
442:          &          SDA, NSDA, LXYZ(1), LXYZ(2), LXYZ(3), LXYZ(4),
443:          &          LXYZ(5), LXYZ(6), LXYZ(7), LXYZ(8), LXYZ(9),
444:          &          LXYZ(10), JVSMP, MWVEC, MVSTK, LPWRK, LVSTK, R, R,
445:          &          NRWK, NERROR )
446: C
447: C
448:          if ( JLATT.ne.0 ) then
449:            do 190 I = 1, NN
450:              LEVL(LSDED(IPP+I)) = 0
451:            190          continue
452:          end if
453: C
454:          do 200 I = 1, NN
455:            KLSF(LSDED(IPP+I)) = 0

```

```

456:      200          continue
457: C
458: C .... transfer sampled data to bank ....
459: C
460:          DWSUMN = 0.0D0
461:          do 210 I = 1, NN
462:            IWK1(NSS+I) = LSDED(IPP+I)
463: C
464:            XXX(IWK1(NSS+I)) = DH(LXD+I)
465:            YYY(IWK1(NSS+I)) = DH(LYD+I)
466:            ZZZ(IWK1(NSS+I)) = DH(LZD+I)
467: C
468:            AAA(IWK1(NSS+I)) = DH(LAD+I)
469:            BBB(IWK1(NSS+I)) = DH(LBD+I)
470:            CCC(IWK1(NSS+I)) = DH(LCD+I)
471: C
472:            WWW(IWK1(NSS+I)) = DH(LWD+I)
473:            DWSUMN = DWSUMN + WWW(IWK1(NSS+I))
474: C
475:            if ( JTIME.ne.0 ) TTT(IWK1(NSS+I)) = DH(LTD+I)
476: C
477:      210          continue
478:          if ( JBPRNT.ne.0.and.JMNTR.ne.0 ) then
479:            write(IOW, '(1x,a,i4,a,i7,a,e12.5,a,e12.5,a)')
480:              &          ' SOURCE # ', N, ': ', NN,
481:              &          ' PARTICLES. WEIGHT SUM ', DWSUMN, ' (AVERAGE ',
482:              &          DWSUMN/NN, ') '
483:          end if
484: C
485: C ... energy value is sampled ...
486: C
487:          if ( JVSMP(7).ne.0 ) then
488: C ... neutron ...
489:            if ( NPK.eq.2**0 ) then
490: Ccccccccccc      debt = ebot
491: Ccccccccccc      detp = etop
492:            if ( JADJM.eq.0 ) then
493:              DEBT = ENGYB(NGP1+1)
494:              DETP = ENGYB(1)
495:            else if ( JPHOT.eq.0 ) then
496:              DEBT = ENGYB(1)
497:              DETP = ENGYB(NGP1+1)
498:            else
499:              DEBT = ENGYB(NGP1+2)
500:              DETP = ENGYB(NGP1+NGP2+2)
501:            end if
502: C ... photon ...
503:            else if ( NPK.eq.2**1 ) then
504: Ccccccccccc      debt = ebotp
505: Ccccccccccc      detp = etopp
506:              NGPP = NGP2
507:              if ( JADJM.eq.1 .or. JNEUT.eq.0 ) NGPP = NGP1
508:              if ( JADJM.eq.0 ) then
509:                DEBT = ENGPB(NGPP+1)
510:                DETP = ENGPB(1)
511:              else
512:                DEBT = ENGPB(1)
513:                DETP = ENGPB(NGPP+1)
514:              end if
515:            end if
516:
517:          do 220 I = 1, NN
518:            R(I) = MAX(DEBT,MIN(DH(LED+I),DETP))
519:          220          continue
520:

```

src/gmvp/source.f

```

521: C
522: C    ... determine energy bins ....
523: C
524: C
525: C    if ( NPK.eq.2**0 ) then
526: C        if ( JADJM.eq.0 ) then
527: C            call BSVDEC( ENGYB, NGP1+1, R, IWK2, NN )
528: C        else if ( JPHOT.eq.0 ) then
529: C            call BSVINC( ENGYB, NGP1+1, R, IWK2, NN )
530: C        else
531: C            call BSVINC( ENGYB(NGP1+2), NGP2+1, R, IWK2, NN )
532: C
533: C            do 230 I = 1, NN
534: C                IWK2(I) = IWK2(I) + NGP1
535: C            continue
536: C        end if
537: C    else
538: C        if ( JADJM.eq.0 ) then
539: C            call BSVDEC( ENGPB, NGPP+1, R, IWK2, NN )
540: C        if ( JNEUT.ne.0 ) then
541: C            do 240 I = 1, NN
542: C                IWK2(I) = IWK2(I) + NGP1
543: C            continue
544: C        end if
545: C    else
546: C        call BSVINC( ENGPB, NGPP+1, R, IWK2, NN )
547: C    end if
548: C end if
549: C
550: C    do 250 I = 1, NN
551: C        IGG(IWK1(NSS+I)) = IWK2(I)
552: C    continue
553: C
554: C    ... energy group is sampled ...
555: C
556: C    else if ( JVSMP(8).ne.0 ) then
557: C
558: C    ... neutron ...
559: C        if ( NPK.eq.2**0 ) then
560: C            *VOCL LOOP,NOVREC
561: C            do 260 I = 1, NN
562: C                IIG = DH(LGD+I)
563: C                IGG(IWK1(NSS+I)) = IIG
564: C                if ( JADJM.ne.0.and.JPHOT.ne.0 ) then
565: C                    IGG(IWK1(NSS+I)) = IIG + NGP1
566: C                end if
567: C            continue
568: C        ... photon ...
569: C        else if ( NPK.eq.2**1 ) then
570: C            *VOCL LOOP,NOVREC
571: C            do 270 I = 1, NN
572: C                IIG = DH(LGD+I)
573: C                IGG(IWK1(NSS+I)) = IIG
574: C                if ( JADJM.eq.0.and.JNEUT.ne.0 ) then
575: C                    IGG(IWK1(NSS+I)) = IIG + NGP1
576: C                end if
577: C            continue
578: C        end if
579: C    end if
580: C
581: C    ... determine time bin ....
582: C
583: C    if ( JTIME.ne.0 ) then
584: C        call BSOISD( TIMEB, NTIME+1, DH(LTD+1), IWK2(1), NN )
585: C        do 280 I = 1, NN

```

```

586: C            ITT(IWK1(NSS+I)) = MAX(1,MIN(NTIME,IWK2(I)))
587: C        continue
588: C    end if
589: C
590: C
591: C-----
592: C === DETERMINE STARTING ZONE NUMBERS =====
593: C-----
594: C
595: C <COMMENT> 'NSST' INDICATES THE NUMBER OF REALLY GENERATED SOURCE
596: C (6/14/1991) PARTICLES ( NSST - NSS IS THE NUMBER OF 'REAL' SOURCE
597: C PARTICLES (I.E. UNLOST SOURCE PARTICLES)).
598: C TO BE USED IN THE UNCOLLIDED FLUX CALCULATION.
599: C
600: C
601: C    NSST = NSS
602: C    do 290 I = 1, NN
603: C        IWK2(I) = IWK1(NSS+I)
604: C        X(I) = XXX(IWK2(I)) + 1.0D-8*AAA(IWK2(I))
605: C        Y(I) = YYY(IWK2(I)) + 1.0D-8*BBB(IWK2(I))
606: C        Z(I) = ZZZ(IWK2(I)) + 1.0D-8*CCC(IWK2(I))
607: C    continue
608: C
609: C    II = NN
610: C    KKZ = 1
611: C    MEM = 1
612: C    MMZ = NZONE
613: C    if ( JLATT.ne.0 ) MMZ = IPCEL(1) - 1
614: C
615: C    do 370 M = 1, MMZ
616: C        if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
617: C            if ( MEM.le.NMEMS.and.MEMZN(MEM,N).ne.0 ) then
618: C                IZ = MEMZN(MEM,N)
619: C                IM = 1
620: C            else
621: C                IM = 0
622: C                do 310 KZ = KKZ, MMZ
623: C                    do 300 K = 1, NMEMS
624: C                        if ( MEMZN(K,N).ne.0.and.KZ.eq.MEMZN(K,N) )
625: C                            go to 310
626: C                    continue
627: C                    go to 320
628: C                continue
629: C                KZ = MMZ
630: C                IZ = KZ
631: C                KKZ = IZ + 1
632: C            end if
633: C        else
634: C            IZ = M
635: C        end if
636: C
637: C =====
638: C
639: C    .... JUDGE WHETHER PARTICLES BELONG TO ZONE IZ OR NOT. ....
640: C
641: C        call JUDGE( 'SOURCE', IZ, II, X, Y, Z, IFI, SDA, KZDA,
642: C                    KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
643: C                    .false., DUMMY, DUMMY, DUMMY )
644: C
645: C =====
646: C
647: C-----
648: C    ..... SEND PARTICLES TO FLIGHT STACK .....
649: C-----
650: C

```

src/gmvp/source.f

```

651:         INN      = 0
652:         if ( KZMAT(IZ,1).ge.0 ) then
653:             IFFL    = NFFL(NZONE+1)
654: *VOCL LOOP,NOVREC
655:         do 330 I = 1, II
656:             if ( IFI(I) ) then
657:                 IZZ(IWK2(I))    = IZ
658:                 INN      = INN + 1
659:                 LSFFL(IFFL+INN) = IWK2(I)
660:                 IZFFL(IFFL+INN) = IZ
661:                 IREG      = KZREG(IZ)
662: C         ... set IBREG anytime (from Jan 2000)
663:                 IBREG(IWK2(I)) = IREG
664:                 if ( JTLT.ne.0 ) then
665:                     IBSPC(IWK2(I),0) = 0
666:                 end if
667:                 if ( JIMPT.ne.0 ) then
668:                     XIM(IWK2(I))    = XIMP(IGG(IWK2(I)),IREG)
669:                 end if
670:                 if ( JPTDT.ne.0 ) IWK1(NSST+INN) = IWK2(I)
671:             end if
672: 330         continue
673: C
674:             NFFL(IZ)    = NFFL(IZ) + INN
675:             NFFL(NZONE+1) = NFFL(NZONE+1) + INN
676: C
677:             if ( JPTDT.ne.0 ) NSST = NSST + INN
678: C
679: C-----
680: C         ..... IF PARTICLES ARE IN A LATTICE, SEND THEM TO LATTICE STACK.
681: C-----
682: C
683: CCCC         else if ( KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998 ) then
684:             else if ( ISLATT(KZMAT(IZ,1)) ) then
685:                 ILAT    = NXLT(NLBZ+1)
686: *VOCL LOOP,NOVREC
687:             do 340 I = 1, II
688:                 if ( IFI(I) ) then
689:                     IZZ(IWK2(I))    = IZ
690:                     INN      = INN + 1
691:                     LSLAT(ILAT+INN) = IWK2(I)
692:                     IZLAT(ILAT+INN) = MLBZZ(IZ)
693:                     IREG      = KZREG(IZ)
694: C         ... set IBREG anytime (from Jan 2000)
695:                     IBREG(IWK2(I)) = IREG
696:                     if ( JTLT.ne.0 ) then
697:                         IBSPC(IWK2(I),0) = 0
698:                     end if
699:                     if ( JIMPT.ne.0 ) then
700:                         if ( IREG.gt.0 ) then
701:                             XIM(IWK2(I))    = XIMP(IGG(IWK2(I)),IREG)
702:                         else
703:                             XIM(IWK2(I))    = 1.0
704:                         end if
705:                     end if
706: C
707:                     if ( JPTDT.ne.0 ) IWK1(NSST+INN) = IWK2(I)
708:                 end if
709: 340         continue
710: C
711:             NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + INN
712:             NXLT(NLBZ+1)    = ILAT + INN
713: C
714:             if ( JPTDT.ne.0 ) NSST = NSST + INN
715: C

```

```

716: C
717: C-----
718: C         ..... GENERATING PARTICLES IN INVALID MATERIAL ? .....
719: C-----
720: C
721:         else
722:             do 350 I = 1, II
723:                 if ( IFI(I) ) then
724:                     INN      = INN + 1
725:                     ILOST    = ILOST + 1
726: C1999/8/5         LSDED(IPP+I)    = IWK2(I)
727:                     LSDED(IPP+INN) = IWK2(I)
728:                 end if
729: 350             continue
730:                 IPP      = IPP + INN
731: C/#IF PARA(SX* CRAY)
732: *                 call MVPSYNC_LOCK(2)
733: C/#ENDIF
734:                 if ( INN.gt.0 ) write(IOW,7040) N,KZMAT(IZ,1),IZ,INN
735: 7040             format(/1X,' !!! CAUTION: YOU ARE GOING TO',
736: &                 ' GENERATE PARTICLES IN A ZONE WHICH PARTICLES',
737: &                 ' SHOULD NOT BE IN !!! '/1X,' SOURCE NO. ',I3,
738: &                 ' MAT = ',I5,' ZONE = ',I5,' ',I8,
739: &                 ' PARTICLES. THEY ARE TREATED AS LOST!!')
740: C/#IF PARA(SX* CRAY)
741: *                 call MVPSYNC_UNLOCK(2)
742: C/#ENDIF
743:             end if
744: C
745: C-----
746: C         ..... MEMORIZE NEWLY FOUND ZONE .....
747: C-----
748: C
749:             if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
750:                 if ( IM.ne.0 ) then
751:                     MEM      = MEM + 1
752:                 else if ( INN.ne.0.and.MEM.le.NMEMS
753: &                 .and.MEMZN(MEM,N).eq.0 ) then
754:                     MEMZN(MEM,N) = IZ
755:                     MEM      = MEM + 1
756:                 end if
757:             end if
758: C
759: C-----
760: C         ..... COMPRESS .....
761: C-----
762: C
763:             if ( INN.lt.II ) then
764:                 INNN      = 0
765: *VOCL LOOP,NOVREC
766:             do 360 I = 1, II
767:                 if ( .not.IFI(I) ) then
768:                     INNN      = INNN + 1
769:                     IWK2(INNN) = IWK2(I)
770:                     X(INNN) = X(I)
771:                     Y(INNN) = Y(I)
772:                     Z(INNN) = Z(I)
773:                 end if
774: 360             continue
775:                 II      = INNN
776:             else
777:                 II      = 0
778:                 go to 380
779:             end if
780: 370         continue

```

src/gmvp/source.f

```

781: C
782: C-----
783: C ..... ZONES NOT FOUND !! (LOST) .....
784: C-----
785: C
786:   380      continue
787: C
788:       if ( II.ne.0 ) then
789: C/#IF PARA(SX* CRAY)
790: *         call MVPSYNC_LOCK(2)
791: C/#ENDIF
792:       write(IOW,7060) II, N, (XXX(IWK2(I)),YYY(IWK2(I)),
793: &         ZZZ(IWK2(I)),AAA(IWK2(I)),BBB(IWK2(I)),
794: &         CCC(IWK2(I)),I=1,II)
795:   7060      format(/1X,' **** ',15,
796: &         ' PARTICLES ARE LOST IN SOURCE ROUTINE!'/1X,
797: &         ' SOURCE NUMBER = ',I5/(1X,' X = ',1P,E12.5,
798: &         ' Y = ',1P,E12.5,' Z = ',1P,E12.5,' MU = ',1P,E12.5,
799: &         ' ETA = ',1P,E12.5,' XI = ',1P,E12.5))
800: C/#IF PARA(SX* CRAY)
801: *         call MVPSYNC_UNLOCK(2)
802: C/#ENDIF
803:       do 390 I = 1, II
804:         LSDDED(IPP+I) = IWK2(I)
805:   390      continue
806:         ILOST = ILOST + II
807:         IPP = IPP + II
808:       end if
809: C
810: C-----
811: C ..... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS .....
812: C-----
813: C
814: C ..... KSOUR2 : =0 ISOTROPIC SOURCE
815: C ..... .NE.0 SOURCE TYPE
816: C
817:       if ( JPTDT.ne.0 ) then
818:         KSOUR2 = 0
819:         if ( KS.eq.2 .or. KS.eq.9 ) KSOUR2 = KS
820:         DUMAX = 1.0D0
821:         DUMIN = -1.0D0
822:         NSORS = NSST - NSS
823: C
824:         call SUNCL( IOW, A, H, NSORS, IWK1(NSS+1), KSOUR2,
825: &         RDUMMY, DUMAX, DUMIN, NPDET, NPLEN,NIMPT, NDIMPT,
826: &         KDBNK(0),KIBNK(0),
827: &         TIMEB, KZMAT, STOTX, VEL ,
828: &         JPUSD, JSPDT, XPDET, IPDET, IPDT2, IMSFL,
829: &         IMNFL, IMZFL, IMDED, IMSLT, IMNLT, IMZLT, XXX,
830: &         YYY, ZZZ, AAA, BBB, CCC, WWW, IZZ, IGG, TTT,
831: &         ITT, LEVL, LZZ, LPOS, LCRS,
832: &         DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
833: &         LZZI, LPOSI, LCRSI, OPTI,
834: &         PATH, KDETP, KLSFI,
835: &         X, Y, Z, R , R(2*NBANK+1), R(4*NBANK+1),
836: &         H(LXYZ(1)),H(LXYZ(2)),H(LXYZ(3)),H(LXYZ(4)),
837: &         H(LXYZ(5)),H(LXYZ(6)),H(LXYZ(7)),H(LXYZ(8)),
838: &         H(LXYZ(9)),H(LXYZ(10)),
839: &         IFL, IFI, RWK1, IWK2 )
840: C
841:       end if
842: C
843: C ... save source set # if necessary as optional bank parameter
844: C
845:       if ( KIBNK(1).ne.0 ) then

```

```

846:       do 400 I = 1, NN
847:         IBNK(IWK1(NSS+I),KIBNK(1)) = N
848:   400      continue
849:       end if
850: C
851: C ... set flight path counter to negative
852: C
853:       if ( KIBNK(5).ne.0 ) then
854:         do 410 I = 1, NN
855:           IBNK(IWK1(NSS+I),KIBNK(5)) = -1
856:   410      continue
857:         end if
858: C
859:         NSS = NSS + NN
860: C
861:   420      continue
862: C
863: C-----
864: C .... UPDATE SUM OF WEIGHTS .....
865: C-----
866: C
867:         DWSUM = 0.0D0
868:         do 430 I = 1, NGENE
869:           DWSUM = DWSUM + WWW(IWK1(I))
870:   430      continue
871:           WSUM = WSUM + DWSUM
872: C
873: C ... save source weight sum of a batch (first batch only)...
874: C
875:         if ( JEIGN.ne.0.and.NGBAT.eq.0 ) XSOC(NBATCH+1) = WSUM
876: C
877:         NCNTR(1) = NCNTR(1) + NGENE
878:         WCNTR(1) = WSUM
879:         WCNTR(1) = WCNTR(1) + DWSUM
880: C
881: C
882: C =====
883: C SOURCES FROM FISSION NEUTRON BANK FOR EIGENVALUE PROBLEM
884: C =====
885: C
886: C
887:       else
888: C
889:       if ( NHSUB.lt.NHIST1 ) then
890:         call FISSRC( IOW, IRAND, NFISB, NGENE, NFB0, WFFACT, IFISB,
891: &         NHIST, IWK1, NHIST, NBANK, NZONE, NFBANK, NBATCH,
892: &         NGROUP, NGP1, NREG, NEST, NLENG, WSUM, KZREG, KZMAT,
893: &         LTYP, WGTFF, ENGYB, XIMP, XSOC, NCNTR, WCNTR, NEVENT,
894: &         NMAT, NTGX, KKAI, FKAI, NFFL, LSFFL, IZFFL, NDEAD,
895: &         LSDDED, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IGG, IZZ,
896: &         TTT, XIM, IBREG, IBSPC, KLSF, LEVL, LZZ, LPOS, LCRS,
897: &         DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, H(LXXXXF2),
898: &         H(LYYYF2), H(LZZZF2), H(LIZZF2), H(LLEVLF2),
899: &         H(LLZZF2), H(LLPOSF2), H(LLCRSF2), H(LIBRGF2),
900: &         H(LIBSPF2), H(LKLSFF2), H(LDFBK2), KDFBK2, MDFBK2,
901: &         H(LIFBK2), KIFBK2, MIFBK2, R, IWK2, RWK1 )
902: C
903:       else
904:         call FISSRC( IOW, IRAND, NFISB, NGENE, NFB0, WFFACT, IFISB,
905: &         NFBNK0, IWK1, NHIST, NBANK, NZONE, NFBANK, NBATCH,
906: &         NGROUP, NGP1, NREG, NEST, NLENG, WSUM, KZREG, KZMAT,
907: &         LTYP, WGTFF, ENGYB, XIMP, XSOC, NCNTR, WCNTR, NEVENT,
908: &         NMAT, NTGX, KKAI, FKAI, NFFL, LSFFL, IZFFL, NDEAD,
909: &         LSDDED, XXX, YYY, ZZZ, AAA, BBB, CCC, WWW, IGG, IZZ,
910: &         TTT, XIM, IBREG, IBSPC, KLSF, LEVL, LZZ, LPOS, LCRS,
911: &         DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK, XXXF, YYYF,

```

src/gmvp/source.f

```

911:      &          ZZZF, IZZF, LEVLF, LZZF, LPOSF, LCRSF, IBRGF, IBSPF,
912:      &          KLSFF, DFBK, KDFBK, MDFBK, IFBK, KIFBK, MIFBK, R,
913:      &          IWK2, RWK1 )
914:      end if
915: C
916: C
917: C ... SOURCES ARE TAKEN FROM FISSION BANK (JEIGN.EQ.1.AND.NBATCH>0)
918: C
919: C      if ( NFISB.le.0 ) then
920: C/ #IF PARA
921: C/ #IF PARA(SX* CRAY)
922: C*      call MVPSYNC_LOCK(2)
923: C/ #ENDIF
924: C      write(IOW,7040) IDTASK
925: C/ #IF PARA(SX* CRAY)
926: C*      call MVPSYNC_UNLOCK(2)
927: C/ #ENDIF
928: C/ #ELSE
929: C      write(IOW,7060)
930: C/ #ENDIF
931: C 7040      format(/IX,' TASK ',I5,
932: C      &          ' !!! NO FISSION SITE HAS BEEN SELECTED. STOP.',
933: C      &          ' !!!')
934: C 7060      format(/IX,' !!! NO FISSION SITE HAS BEEN SELECTED. STOP.',
935: C      &          ' !!!')
936: C      stop 666
937: C      end if
938: C
939: C      if ( NFISB.le.NGENE ) then
940: C      do 420 I = 1, NFISB
941: C          IWK2(I) = I
942: C      420      continue
943: C      if ( NFISB.lt.NGENE ) then
944: C          call RANU2( IRAND, R, NGENE-NFISB, ICON )
945: C          do 430 I = 1, NGENE - NFISB
946: C
947: C/ #IF ROUNDOFF(NEAREST)
948: C          IP      = MIN(INT(NFISB*R(I))+1,NFISB)
949: C/ #ELSE
950: C          IP      = NFISB*R(I) + 1
951: C/ #ENDIF
952: C          IWK2(NFISB+I) = IP
953: C      430      continue
954: C      end if
955: C      else
956: C          call RANU2( IRAND, R, NGENE, ICON )
957: C          do 440 I = 1, NGENE
958: C
959: C/ #IF ROUNDOFF(NEAREST)
960: C          IP      = MIN(INT(NFISB*R(I))+1,NFISB)
961: C/ #ELSE
962: C          IP      = NFISB*R(I) + 1
963: C/ #ENDIF
964: C
965: C          IWK2(I) = IP
966: C      440      continue
967: C      end if
968: C
969: C      call RANU2( IRAND, R, 4*NGENE, ICON )
970: C
971: C      IPP      = NDEAD - NGENE
972: C      NGEN2    = NGENE*2
973: C      NGEN3    = NGENE*3
974: C      WSUMF    = 0.0
975: C*VOCL LOOP,NOVREC

```

```

976: C      do 450 I = 1, NGENE
977: C          IWK1(I) = LSEDED(IPP+I)
978: C          .... LATTICE PARAMETER BANK ....
979: C          IF(JLATT.NE.0) THEN
980: C              LEVL(IWK1(I)) = LEVLF(IWK2(I))
981: C              DO 375 K=1,NEST
982: C                  LZZ(IWK1(I),K) = LZZF(IWK2(I),K)
983: C                  LPOS(IWK1(I),K) = LPOSF(IWK2(I),K)
984: C                  IF(JHLAT.NE.0)
985: C                      &          LCRS(IWK1(I),K) = LCRSF(IWK2(I),K)
986: C      375      CONTINUE
987: C      Cccccccccc  ENDIF
988: C      if ( JLATT.ne.0 ) LEVL(IWK1(I)) = LEVLF(IWK2(I))
989: C
990: C      XXX(IWK1(I)) = XXXF(IWK2(I))
991: C      YYY(IWK1(I)) = YYF(IWK2(I))
992: C      ZZZ(IWK1(I)) = ZZZF(IWK2(I))
993: C      D      = 2.0*R(I) - 1.0
994: C      AAA(IWK1(I)) = D
995: C      D      = SQRT(1.0-D**2)
996: C      D2     = 6.283185307179586D0*R(NGENE+I)
997: C      BBB(IWK1(I)) = COS(D2)*D
998: C      CCC(IWK1(I)) = SIN(D2)*D
999: C      IZZ(IWK1(I)) = IZZF(IWK2(I))
1000: C3/7/1991 ...
1001: C      KLSF(IWK1(I)) = 0
1002: C      IREG      = KZREG( IWK2(I) )
1003: C      RWK1(I)    = WGTf( IREG )
1004: C
1005: C      ... set IBREG anytime
1006: C      if ( JTLLT.eq.0 ) then
1007: C          IREG      = KZREG(IZZ(IWK1(I)))
1008: C          IBREG(IWK1(I)) = IREG
1009: C          RWK1(I)    = WGTf(IREG)
1010: C      else
1011: C          IBREG(IWK1(I)) = IBRGF(IWK2(I))
1012: C          RWK1(I)    = WGTf(IBREG(IWK1(I)))
1013: C      end if
1014: C
1015: C      WSUMF      = WSUMF + RWK1(I)
1016: C
1017: C      if ( JTIME.ne.0 ) TTT(IWK1(I)) = 0.0D0
1018: C
1019: C      IMAT      = KZMAT(IZZ(IWK1(I)),1)
1020: C      if ( IMAT.lt.0 ) IMAT = KZMAT(IZZ(IWK1(I)),2)
1021: C
1022: C/ #IF ROUNDOFF(NEAREST)
1023: C      KK      = MIN( INT(NTGX*R(NGEN2+I))+1,NTGX)
1024: C      KKK     = 2*KK
1025: C      KKK     = MIN(KKK,INT(KKK+R(NGEN3+I)-FKAI(KK,IMAT)))
1026: C/ #ELSE
1027: C      KK      = NTGX*R(NGEN2+I) + 1
1028: C      KKK     = KK*2 + R(NGEN3+I) - FKAI(KK,IMAT)
1029: C/ #ENDIF
1030: C
1031: C      IGG(IWK1(I)) = KKAI(KKK,1,IMAT)
1032: C 450      continue
1033: C
1034: C      WSUMF2 = FLOAT(NGENE)
1035: C      RATIO  = WSUMF2/WSUMF
1036: C      do 460 I = 1, NGENE
1037: C          WWW(IWK1(I)) = RWK1(I)*RATIO
1038: C      460      continue
1039: C
1040: C          .... LATTICE PARAMETER BANK ....

```

src/gmvp/source.f

```

1041: C
1042: C      if ( JLATT.ne.0 ) then
1043: C          do 480 K = 1, NEST
1044: C      C*VOCL LOOP,NOVREC
1045: C          do 470 I = 1, NGENE
1046: C              LZZ(IWK1(I),K) = LZZF(IWK2(I),K)
1047: C              LPOS(IWK1(I),K) = LPOSF(IWK2(I),K)
1048: C              if ( JHLAT.ne.0 ) LCRS(IWK1(I),K) = LCRSF(IWK2(I),K)
1049: C              if ( JTLT.ne.0 ) IBSPC(IWK1(I),K) = IBSPF(IWK2(I),K)
1050: C          continue
1051: C      C 480      continue
1052: C          end if
1053: C
1054: C      ... retrieve distance to lattice frame ...
1055: C
1056: C      if( KDFBK(4).gt.0.and.KDBNK(4).gt.0 ) then
1057: C          KF = KDFBK(4)
1058: C          KB = KDBNK(4)
1059: C          do 1172 K = 0, NEST
1060: C              do 1174 I = 1, NGENE
1061: C                  DBNK(IWK1(I),KB+k) = DFBK(IWK2(I),KF+k)
1062: C              continue
1063: C          continue
1064: C      end if
1065: C      if( KIFBK(6).gt.0.and.KIBNK(6).gt.0 ) then
1066: C          KF = KIFBK(6)
1067: C          KB = KIBNK(6)
1068: C          do 2172 K = 0, NEST
1069: C              do 2174 I = 1, NGENE
1070: C                  IBNK(IWK1(I),KB+k) = IFBK(IWK2(I),KF+k)
1071: C              continue
1072: C          continue
1073: C      end if
1074: C
1075: C      ... retrieve STG sphere position ...
1076: C
1077: C      if( KDFBK(5).gt.0.and.KDBNK(5).gt.0 ) then
1078: C          KF = KDFBK(5)
1079: C          KB = KDBNK(5)
1080: C          do 172 K = 1, NEST
1081: C              do 174 I = 1, NGENE
1082: C                  DBNK(IWK1(I),KB) = DFBK(IWK2(I),KF)
1083: C                  DBNK(IWK1(I),KB+1) = DFBK(IWK2(I),KF+1)
1084: C                  DBNK(IWK1(I),KB+2) = DFBK(IWK2(I),KF+2)
1085: C              continue
1086: C              KF = KF + 3
1087: C              KB = KB + 3
1088: C          continue
1089: C      end if
1090: C
1091: C      .... STG placement flag (which type of NND should be used)
1092: C
1093: C      Base material (matrix) of STG region needs NND type2 sampling
1094: C
1095: C      if( JSTGR.ne.0 ) then
1096: C          do 233 I = 1, NGENE
1097: C              INND = 0
1098: C              LVL = LEVL(IWK1(I))
1099: C              if ( LVL.gt.0 ) then
1100: C                  CCCCCCCCCC      if( LTYP(ABS(KZMAT(LZZ(IWK1(I),LVL)))) .eq.10 ) then
1101: C                      if( LTYP(LATTNM(KZMAT(LZZ(IWK1(I),LVL),1))) .eq.10 )
1102: C                          &      then
1103: C                              if ( LPOS(IWK1(I),LVL).ne.0 ) INND = 2
1104: C                          end if
1105: C                      end if

```

```

1106: C          IBNK(IWK1(I),KIBNK(7)) = INND
1107: C      233      continue
1108: C          end if
1109: C
1110: C      ... clear flight count in flight path
1111: C
1112: C      if( JFISX.ne.0 ) then
1113: C          do 243 I = 1, NGENE
1114: C              IBNK(IWK1(I),KIBNK(5)) = 0
1115: C          243      continue
1116: C      end if
1117: C
1118: C      ... IWK2 is free to use hereafter
1119: C
1120: C
1121: C=====
1122: C
1123: C      ... determine time bin ....
1124: C
1125: C      if ( JTIME.ne.0 ) then
1126: C          do 440 I = 1, NGENE
1127: C              Z(I) = TTT(IWK1(I))
1128: C          440      continue
1129: C          call BS0ISD( TIMEB, NTIME+1, Z(1), IWK2(1), NGENE )
1130: C          do 450 I = 1, NGENE
1131: C              ITT(IWK1(I)) = MAX(1,MIN(NTIME,IWK2(I)))
1132: C          450      continue
1133: C      end if
1134: C
1135: C
1136: C      .... MAKE FREE FLIGHT STACK .....
1137: C
1138: C      C*VOCL LOOP,NOVREC
1139: C          do 510 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
1140: C              LSFFL(I) = IWK1(I-NFFL(NZONE+1))
1141: C              IZFFL(I) = IZZ(IWK1(I-NFFL(NZONE+1)))
1142: C              if ( JIMPT.ne.0 ) then
1143: C                  if ( JTLT.eq.0 ) then
1144: C                      XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),KZREG(IZFFL(I)))
1145: C                  else
1146: C                      XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),IBREG(LSFFL(I)))
1147: C                  end if
1148: C              end if
1149: C          510      continue
1150: C
1151: C          do 520 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
1152: C              NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
1153: C          520      continue
1154: C
1155: C              NFFL(NZONE+1) = NFFL(NZONE+1) + NGENE
1156: C
1157: C      .... UPDATE SUM OF WEIGHTS .....
1158: C
1159: C          WSUM = WSUM + WSUMF2
1160: C          XSOC(NBATCH+1) = WSUMF2
1161: C          NCNTR(1) = NCNTR(1) + NGENE
1162: C          Ccccc      WCNTR(1) = WSUM
1163: C          WCNTR(1) = WCNTR(1) + WSUMF2
1164: C
1165: C
1166: C
1167: C=====
1168: C      .... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS ....
1169: C=====
1170: C

```

src/gmvp/source.f

```

1171: C      .... KSOUR2 = 0   ISOTROPIC SOURCE
1172: C      .NE.0   SOURCE TYPE
1173: C
1174: C
1175: C
1176: C      .... GENERATE IMAGINARY PARTICLES TO CALCULATE UNCLLIED
1177: C      CONTRIBUTION TO NEXT EVENT DETECTORS .....
1178: C
1179: C
1180: C      if ( JPTDT.ne.0 ) then
1181: C      LLLLL = NFFL(NZONE+1) - NGENE + 1
1182: C      KSOUR2 = 0
1183: C      DUMIN = -1.0D0
1184: C      DUMAX = 1.0D0
1185: C      call SUNCL( IOW, A, H, NGENE, LSFLL(LLLLL),
1186: C      &      KSOUR2, RDUMMY, DUMAX, DUMIN,
1187: C      &      NPDET, NPLEN, NIMPT, NDIMPT, KDBNK(0), KIBNK(0),
1188: C      &      TIMEB, KZMAT, STOTX, VEL ,
1189: C      &      JPUSD, JSPDT, XPDET, IPDET, IPDT2, IMSFL, IMNPL,
1190: C      &      IMZFL, IMDED, IMSLT, IMNLT, IMZLT, XXX, YYY, ZZZ,
1191: C      &      AAA, BBB, CCC, WWW, IZZ, IGG, TTT, ITT,
1192: C      &      LEVL, LZZ, LPOS, LCRS,
1193: C      &      DBNK, KDBNK, MDBNK, IBNK, KIBNK, MEBNK,
1194: C      &      LZZI, LPOSI, LCRSI, OPTI, PATH, KDETP, KLSFI,
1195: C      &      X, Y, Z, R = , R(2*NBANK+1), R(4*NBANK+1),
1196: C      &      H(LXYZ(1)), H(LXYZ(2)), H(LXYZ(3)), H(LXYZ(4)),
1197: C      &      H(LXYZ(5)), H(LXYZ(6)), H(LXYZ(7)), H(LXYZ(8)),
1198: C      &      H(LXYZ(9)), H(LXYZ(10)),
1199: C      &      IPL, IFI, RWK1, IWK2 )
1200: C
1201: C      end if
1202: C
1203: C
1204: C      NFISB = 0
1205: C
1206: C      end if
1207: C
1208: C      ... treatment for optional bank parameters ....
1209: C
1210: C      ... weight on birth
1211: C
1212: C      if ( KDBNK(1).ne.0 ) then
1213: C      do 460 I = 1, NGENE
1214: C      DBNK(IWK1(I),KDBNK(1)) = WWW(IWK1(I))
1215: C 460 continue
1216: C      end if
1217: C
1218: C      ... time on birth
1219: C
1220: C      if ( KDBNK(2).ne.0 ) then
1221: C      do 470 I = 1, NGENE
1222: C      DBNK(IWK1(I),KDBNK(2)) = TTT(IWK1(I))
1223: C 470 continue
1224: C      end if
1225: C
1226: C      ... energy on birth
1227: C
1228: C      if ( KDBNK(3).ne.0 ) then
1229: C      do 550 I = 1, NGENE
1230: C      DBNK(IWK1(I),KDBNK(3)) = EEE(IWK1(I))
1231: C 550 continue
1232: C      end if
1233: C
1234: C      ... region # on birth (but the region # may not be the real one
1235: C      in lattice geometry ...)
```

```

1236: C      if ( KIBNK(2).ne.0 ) then
1237: C      *VOCL LOOP,NOVREC
1238: C      do 480 I = 1, NGENE
1239: C      if ( JTLLT.ne.0 ) then
1240: C      IBNK(IWK1(I),KIBNK(2)) = IBREG(IWK1(I))
1241: C      else
1242: C      IBNK(IWK1(I),KIBNK(2)) = KZREG(IZZ(IWK1(I)))
1243: C      end if
1244: C 480 continue
1245: C      end if
1246: C
1247: C      ... zone # on birth (but the region # may not be the real one
1248: C      in lattice geometry ...)
1249: C      if ( KIBNK(3).ne.0 ) then
1250: C      do 490 I = 1, NGENE
1251: C      IBNK(IWK1(I),KIBNK(3)) = IZZ(IWK1(I))
1252: C 490 continue
1253: C      end if
1254: C
1255: C      ... generation of particle (1 for primary particles)
1256: C
1257: C      if ( KIBNK(4).ne.0 ) then
1258: C      do 500 I = 1, NGENE
1259: C      IBNK(IWK1(I),KIBNK(4)) = 1
1260: C 500 continue
1261: C      end if
1262: C
1263: C      ... Marker region flag
1264: C
1265: C      if ( KIBNK(9).ne.0 ) then
1266: C      do 520 J = 1, NMKREG
1267: C      KI9 = KIBNK(9) + J - 1
1268: C      do 510 I = 1, NGENE
1269: C      IBNK(IWK1(I),KI9) = 0
1270: C 510 continue
1271: C 520 continue
1272: C      end if
1273: C
1274: C      .... PARTICLES IN THE REST OF BANK ARE DEAD PARTICLES ! .....
1275: C
1276: C      NDEAD = NDEAD - NGENE + ILOST
1277: C      NLOST = NLOST + ILOST
1278: C
1279: C      .... CALCULATE & PRINT AVERAGED PARAMETERS .....
1280: C
1281: C      XAV = 0.0
1282: C      YAV = 0.0
1283: C      ZAV = 0.0
1284: C      AAV = 0.0
1285: C      BAV = 0.0
1286: C      CAV = 0.0
1287: C      EAV = 0.0
1288: C
1289: C      do 530 I = 1, NGENE
1290: C      XAV = XAV + XXX(IWK1(I))*WWW(IWK1(I))
1291: C      YAV = YAV + YYY(IWK1(I))*WWW(IWK1(I))
1292: C      ZAV = ZAV + ZZZ(IWK1(I))*WWW(IWK1(I))
1293: C      AAV = AAV + AAA(IWK1(I))*WWW(IWK1(I))
1294: C      BAV = BAV + BBB(IWK1(I))*WWW(IWK1(I))
1295: C      CAV = CAV + CCC(IWK1(I))*WWW(IWK1(I))
1296: C      EAV = EAV + FLOAT(IGG(IWK1(I)))*WWW(IWK1(I))
1297: C      WSUMB = WSUMB + WWW(IWK1(I))
1298: C 530 continue
1299: C
1300: C      NTGEN = NTGEN + NGENE
```


src/gmvp/source.f

```

1301: C
1302: C ... generated all history in a batch ...
1303: C
1304: C if ( NGBAT.eq.0 ) then
1305: C
1306: C     RSRC = DBLE(NGENE0) / WSUMB
1307: C     XAV = XAV * RSRC
1308: C     YAV = YAV * RSRC
1309: C     ZAV = ZAV * RSRC
1310: C     AAV = AAV * RSRC
1311: C     BAV = BAV * RSRC
1312: C     CAV = CAV * RSRC
1313: C     EAV = EAV * RSRC
1314: C
1315: C     XAVT(1) = XAVT(1) + XAV
1316: C     XAVT(2) = XAVT(2) + YAV
1317: C     XAVT(3) = XAVT(3) + ZAV
1318: C     AAVT(1) = AAVT(1) + AAV
1319: C     AAVT(2) = AAVT(2) + BAV
1320: C     AAVT(3) = AAVT(3) + CAV
1321: C     EAVT = EAVT + EAV
1322: C
1323: C     XAV = XAV/NGENE0
1324: C     YAV = YAV/NGENE0
1325: C     ZAV = ZAV/NGENE0
1326: C
1327: C     if ( JPRTS(6).ne.1 ) then
1328: C /#IF PARA(SX* CRAY)
1329: C * call MVPSYNC_LOCK(2)
1330: C /#ENDIF
1331: C     if ( JBPRNT.ne.0 ) then
1332: C /#IF PARA
1333: C write(IOW,'(1X,' TASK ',I5)') IDTASK
1334: C /#ENDIF
1335: C write(IOW,7080) '(BATCH ) ', XAV, YAV, ZAV, AAV/
1336: C NGENE0, BAV/NGENE0, CAV/NGENE0, EAV/NGENE0
1337: C & write(IOW,7080) '(CUMULATIVE)', XAVT(1) /NTGEN, XAVT(2) /
1338: C & NTGEN, XAVT(3) /NTGEN, AAVT(1) /NTGEN, AAVT(2) /
1339: C & NTGEN, AAVT(3) /NTGEN, EAVT/NTGEN
1340: C write(IOW,*) ' GENERATED PARTICLES IN THIS RUN = ', NTGEN
1341: C end if
1342: C /#IF PARA(SX* CRAY)
1343: C * call MVPSYNC_UNLOCK(2)
1344: C /#ENDIF
1345: C end if
1346: C end if
1347: C
1348: C 7080 format(1X,' AVERAGE ',A12,2X,' X=',1PE11.4,' Y=',E11.4,' Z=',
1349: C & E11.4,' MU=',E11.4,' ETA=',E11.4,' XI=',E11.4,' IG=',E11.4
1350: C & )
1351: C
1352: C IF(JDEBG.EQ.1) THEN
1353: C WRITE(6,7030) (I,XXX(I),YYY(I),ZZZ(I),AAA(I),BBB(I),CCC(I),
1354: C & WWW(I),IZZ(I),IGG(I),TTT(I),I=1,NGENE)
1355: C C7030 FORMAT(/1X,' GENERATED SOURCE PARAMETERS '/
1356: C & 1H , ' XXX YYY ZZZ AAA BBB',
1357: C & ' CCC WWW IZZ IGG TTT ' /
1358: C & (1H ,I6,1P6D11.4,E11.4,2I7,E11.4))
1359: C ENDIF
1360: C
1361: C ... debug dump
1362: C
1363: C if ( MOD(JDEBG(3),2).eq.1 ) then
1364: C write(IOW,7100) (I,XXX(IWK1(I)),YYY(IWK1(I)),ZZZ(IWK1(I)),
1365: C & AAA(IWK1(I)),BBB(IWK1(I)),CCC(IWK1(I)),WWW(IWK1(I)),

```

```

1366: C & IZZ(IWK1(I)),IGG(IWK1(I)),TTT(IWK1(I)),I=1,NGENE)
1367: C 7100 format(/1X,' GENERATED SOURCE PARAMETERS '/1X,
1368: C & ' XXX YYY ZZZ',
1369: C & ' AAA BBB',
1370: C & ' CCC WWW IZZ IGG TTT' /
1371: C & (1X,I5,1P,6E12.4,E12.4,I5,I3,E12.4))
1372: C end if
1373: C
1374: C if ( JEIGN.ne.0 ) then
1375: C XVA = 0.0
1376: C YVA = 0.0
1377: C ZVA = 0.0
1378: C do 540 I = 1, NGENE
1379: C XVA = XVA + XXX(IWK1(I))*XXX(IWK1(I))*WWW(IWK1(I))
1380: C YVA = YVA + YYY(IWK1(I))*YYY(IWK1(I))*WWW(IWK1(I))
1381: C ZVA = ZVA + ZZZ(IWK1(I))*ZZZ(IWK1(I))*WWW(IWK1(I))
1382: C 540 continue
1383: C XSXV(NBATCH+1,1) = XSXV(NBATCH+1,1) + XVA
1384: C XSXV(NBATCH+1,2) = XSXV(NBATCH+1,2) + YVA
1385: C XSXV(NBATCH+1,3) = XSXV(NBATCH+1,3) + ZVA
1386: C
1387: C if ( NGBAT.eq.0 ) then
1388: C
1389: C XSXV(NBATCH+1,1) = XSXV(NBATCH+1,1) * RSRC
1390: C XSXV(NBATCH+1,2) = XSXV(NBATCH+1,2) * RSRC
1391: C XSXV(NBATCH+1,3) = XSXV(NBATCH+1,3) * RSRC
1392: C
1393: C XSXV(NBATCH+1,1) = (XSXV(NBATCH+1,1)-NGENE0*XAV**2) /
1394: C & (NGENE0-1)
1395: C XSXV(NBATCH+1,2) = (XSXV(NBATCH+1,2)-NGENE0*YAV**2) /
1396: C & (NGENE0-1)
1397: C XSXV(NBATCH+1,3) = (XSXV(NBATCH+1,3)-NGENE0*ZAV**2) /
1398: C & (NGENE0-1)
1399: C
1400: C /#IF PARA(SX* CRAY)
1401: C * call MVPSYNC_LOCK(2)
1402: C /#ENDIF
1403: C
1404: C if ( JBPRNT.ne.0 ) then
1405: C if ( JPRTS(6).ne.1 ) then
1406: C write(IOW,7120) IDTASK, '(BATCH ) ',
1407: C & XSXV(NBATCH+1,1), XSXV(NBATCH+1,2),
1408: C & XSXV(NBATCH+1,3)
1409: C end if
1410: C /#ELSE
1411: C if ( JPRTS(6).ne.1 ) then
1412: C write(IOW,7140) '(BATCH ) ', XSXV(NBATCH+1,1),
1413: C & XSXV(NBATCH+1,2), XSXV(NBATCH+1,3)
1414: C end if
1415: C /#ENDIF
1416: C end if
1417: C 7120 format(1X,' TASK ',I5,' VARIANCE ',A10,2X,' X= ',1PE11.4,
1418: C & ' Y= ',E11.4,' Z= ',E11.4)
1419: C 7140 format(1X,' VARIANCE ',A10,2X,' X= ',1PE11.4,' Y= ',E11.4,
1420: C & ' Z= ',E11.4)
1421: C
1422: C /#IF PARA(SX* CRAY)
1423: C * call MVPSYNC_UNLOCK(2)
1424: C /#ENDIF
1425: C end if
1426: C end if
1427: C
1428: C return
1429: C
1430: C /#ENDIF

```

src/gmvp/source.f

1431:
1432: end



src/gmvp/srcold.f

```

1: C/#IF SOURCE(NEW)
2:   subroutine SDUMMY( )
3:   return
4: C/#ELSE
5:
6: C/#IF ARGSAVE
7: *   SUBROUTINE SOURCE(IOW,A, RH,DH, IRAND,NTGEN,NBANK,NDEAD,
8: *   N NGROUP,NZONE,NFBANK,NFBNK0,NFISB,NEST,NLOST,
9: *   N NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NMKREG,
10: *   B XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,
11: *   B IZZ,IGG,TTT,ITT,KLSF,IBREG,IBSPC, XXXF,YYYY,
12: *   B ZZZF,IZZF,KLSFF, LEVL, LZZ,
13: *   B LPOS, LCRS, LEVLF, LZZF, LPOSF,
14: *   B LCRSF,IBRGF,IBSPF, XIM,
15: *   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
16: *   G MLBZZ,KZMAT,KZREG, MEMZN, SDA
17: *   G ,KZDA,KZAA,IPCEL,XIMP,LSFFL,NFFL,
18: *   S IZFFL,LSDED,LSLAT,IZLAT,NXLT,
19: *   * KSOUR,ISZON,SOUR,PSPAC,PENRG,KENRG,
20: *   * NSTIM,STIM,PSTIM,ISTIM, NSANG,SANG,
21: *   * SAXIS,PSANG,ISANG,IFISM,FKAI,KKAI,NTGX,
22: *   T WSUM,XSOC,NCNTR,WCNTR,WGTF,ENGYB,
23: *   T NLENG,XSXV,XAVT,AAVT,EAVT,LXYZ,
24: *   W R,IWK1,IWK2,RWK1,X,Y,Z,IFL,IFI,NNSRC,XSOUR)
25: C/#ELSE
26:   SUBROUTINE SOURCE(IOW,A, RH, DH, IRAND,NTGEN,NBANK,NDEAD,
27:   N NGROUP,NZONE,NFBANK,NFBNK0,NFISB,NEST,NLOST,
28:   N NLBZ,NSOUR,NMAT,NREG,NZDA,NSDA,NMEMS,NMKREG,
29:   B XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,
30:   B IZZ,IGG,TTT,ITT,KLSF,IBREG,IBSPC, XXXF,YYYY,
31:   B ZZZF,IZZF,KLSFF, LEVL, LZZ,
32:   B LPOS, LCRS, LEVLF, LZZF, LPOSF,
33:   B LCRSF,IBRGF,IBSPF, XIM,
34:   B DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
35:   G MLBZZ,KZMAT,KZREG, MEMZN, SDA
36:   G ,KZDA,KZAA,IPCEL,XIMP,LSFFL,NFFL,
37:   S IZFFL,LSDED,LSLAT,IZLAT,NXLT,
38:   * KSOUR,ISZON,SOUR,PSPAC,PENRG,KENRG,
39:   * NSTIM,STIM,PSTIM,ISTIM, NSANG,SANG,
40:   * SAXIS,PSANG,ISANG,IFISM,FKAI,KKAI,NTGX,
41:   T WSUM,XSOC,NCNTR,WCNTR,WGTF,ENGYB,
42:   T NLENG,XSXV,XAVT,AAVT,EAVT,LXYZ,
43:   W R,IWK1,IWK2,RWK1,X,Y,Z,IFL,IFI,NNSRC,XSOUR,
44:   % IMPMAX,NPDET,NPLEN,NTIME,NIMPT,NDIMPT,JPUSD,
45:   D JSPDT,TIMB,
46:   D XPDET,IPDET,IPDT2,STOTX,VEL,
47:   S IMSFL,IMNFL,IMZFL,
48:   S IMDED,IMSLT,IMNLT,IMZLT,
49:   B LZZI,LPOS,LCRSI,
50:   B OPTI,PATH,KDETP,KLSFI,
51: C W AA,BB,CC,DI,
52: % NGENE,NGENEO,NBATCH,WSUMB)
53: C/#ENDIF
54: C=====
55: C PURPOSE: GENERATE PARTICLES (OLD TYPE. MVP v2 support new type of
56: C source input. This routine is obsolescent.)
57: C CALLED IN: ACTION
58: C CALLS: RANU2,VMNTR1
59: C
60: C=====
61:   implicit real*8(D)
62: C
63:   include 'INC/_FLAGS'
64:   include '../shared/INC/_TASKDT'
65:   include '../shared/INC/_WORDL'

```

```

66: C
67: C ... head position of dynamic memory area ....
68: C
69:   real A(*)
70:   real RH(*)
71:   real*8 DH(*)
72: C
73: C/#IF INTEGER8
74: *   integer*8 NTGEN
75: C/#ELSE
76:   integer NTGEN
77: C/#ENDIF
78: C
79: C .... PARTICLE BANK .....
80: C
81:   real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK), AAA(NBANK), BBB(NBANK),
82:   & CCC(NBANK)
83:   real WWW(NBANK), XIM(NBANK)
84:   real*8 TTT(NBANK)
85:   integer IZZ(NBANK), IGG(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
86:   & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK),
87:   & IBREG(NBANK), IBSPC(NBANK,0:NEST)
88:   integer ITT(NBANK)
89: C
90:   real*8 DBNK(NBANK,*)
91:   integer KDBNK(0:MDBNK)
92:   integer IBNK(NBANK,*)
93:   integer KIBNK(0:MIBNK)
94: C
95: C .... FISSION BANK .....
96: C
97:   real*8 XXXF(NFBNK0), ZZZF(NFBNK0), YYYY(NFBNK0)
98:   integer IZZF(NFBNK0), LEVLF(NFBNK0), LZZF(NFBNK0,NEST),
99:   & LPOSF(NFBNK0,NEST), LCRSF(NFBNK0,NEST), KLSFF(NFBNK0),
100:   & IBRGF(NFBNK0), IBSPF(NFBNK0,NEST)
101: C
102: C .... TALLY .....
103: C
104:   real*8 WSUM,XSOC(NLENG),XSXV(NLENG,3),WSUMB
105:   real*8 NCNTR(*),WCNTR(*)
106:   real XAVT(3),AAVT(3)
107: C
108: C .... VARIANCE REDUCTION .....
109: C
110:   real XIMP(NGROUP,NREG)
111: C
112: C .... STACKS ...(FREE FLIGHT & MORGUE) .....
113: C
114:   integer LSFFL(NBANK),NFFL(NZONE+1),IZFFL(NBANK),LSDED(NBANK)
115: C
116: C .... STACKS ...(LATTICE SEARCH) .....
117: C
118:   integer LSLAT(NBANK),NXLT(*),IZLAT(NBANK)
119: C
120: C .... WORKING ARRAYS ( LENGTH OF R MUST BE 4*NBANK FOR USE
121: C IN FNSSOC )
122:   real R(6*NBANK),RWK1(NBANK)
123:   real*8 XSOUR(NSOUR)
124:   logical IFI(NBANK),IFL(NBANK)
125:   real*8 X(NBANK),Y(NBANK),Z(NBANK)
126:   integer NNSRC(NSOUR)
127:   integer IWK1(NBANK),IWK2(NBANK)
128:   integer LXYZ(10)
129: C
130: C .... SOURCE DATA .....

```

src/gmvp/srcold.f

```

131: C
132:   integer KSOUR(NSOUR), ISZON(2,NSOUR)
133:   real*8 SOUR(NSOUR)
134:   real PSPAC(10,NSOUR), PENRG(NGROUP,NSOUR)
135:   integer KENRG(2,NGROUP,NSOUR)
136:   real ENGYB(*)
137:   real STIM(*), PSTIM(*)
138:   integer NSTIM(NSOUR), ISTIM(NSOUR), NSANG(NSOUR)
139:   real SANG(*), PSANG(*), SAXIS(3,NSOUR),
140:   integer ISANG(NSOUR), IFISM(NSOUR), MEMZN(NMEMS,NSOUR)
141: C
142: C .... FISSION NEUTRON DATA .....
143: C
144:   integer KKAI(2,NGROUP,NMAT)
145:   real FKAI(NGROUP,NMAT), WGTf(NREG)
146: C
147: C .... GEOMETRY ARRAY DATA .....
148: C
149:   integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
150:   &      MIBZZ(NZONE), IPCEL(*)
151:   real*8 SDA(*)
152: C
153: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR
154: C
155:   real*8 XPDET(NPLEN,NPDET)
156:   integer IPDET(NPDET,*), IPDT2(NEST,3,NPDET)
157:   integer JPUSD(NPDET), JSPDT(NPDET)
158: C
159: C**** total cross section ( added in Jun, 1998 )
160: C
161:   real STOTX(NTGX,*)
162: C
163: C**** TALLY BIN DATA & VELOCITY
164: C
165:   real TIMEB(*), VEL(NGROUP)
166: C
167: C**** STACK FOR IMAGINARY PARTICLE
168: C
169:   integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(*), IMDED(IMPMAX),
170:   &      IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
171: C
172: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE
173: C
174:   integer LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
175: C****
176:   real*8 OPTI(IMPMAX), PATH(IMPMAX)
177:   integer KDETP(IMPMAX), KLSFI(IMPMAX)
178: C
179: C**** WORKING AREA
180: C
181: C   real*8 AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
182: C
183:   include '../shared/INC/_PMLATT'
184:   include '../shared/INC/_SFLATT'
185: C
186: C/#IF ARGSAVE
187: *   return
188: C
189: C -----
190: C
191: *   entry SOURCU( IMPMAX, NPDET, NPLEN, NTIME, NIMPT, NDIMPT,JPUSD,
192: *   D           JSPDT, TIMEB,
193: *   D           XPDET, IPDET, IPDT2, STOTX, VEL,
194: *   S           IMSFL, IMNFL, IMZFL,
195: *   S           IMDED, IMSLT, IMNLT, IMZLT,

```

```

196: *   B           LZZI , LPOSI, LCRSI,
197: *   B           OPTI , PATH , KDETP, KLSFI)
198: C   B           OPTI , PATH , KDETP, KLSFI,
199: C   W           AA , BB , CC , DI )
200: C
201: C
202: *   return
203: C
204: C -----
205: *   entry      SOURCE( NGENE , NGENE0 ,NBATCH, WSUMB )
206: C/#ELSE
207: C --- ARGSAVE none ---
208: C
209: C/#ENDIF
210: CC   IF(JVMNT.NE.0) CALL VMNTR0('SOURCE')
211: C
212: C .... IF ALL PARTICLES ARE DEAD, CLEAR UP DEAD PARTICLE STACK .....
213: C
214:   if ( NDEAD.eq.NBANK ) then
215:     do 100 I = 1, NBANK
216:       LSDDED(I) = I
217:     100 continue
218: C
219:     NDIMPT = IMPMAX
220:     if ( JPTDT.ne.0 ) then
221:       do 110 I = 1, IMPMAX
222:         IMDED(I) = I
223:       110 continue
224:     end if
225: C
226:   end if
227: C
228: C ..... NGENE0, WSUMB : MEMORIZE NGENE FOR USE IN TALSUM .....
229: C
230:   NGENE0 = NGENE
231:   WSUMB = DBLE( NGENE0 )
232:   ILOST = 0
233: C
234: C/#IF PARA
235: C/#IF PARA(SX* CRAY)
236: *   call MVPSYNC_LOCK(2)
237: C/#ENDIF
238:   if ( JBPRNT.ne.0 ) then
239:     write(IOW,
240:   &'(1x,'' TASK '' ,i5,'' GENERATE '' ,i6,'' SOURCES.   IRAND = '' ,   &
241:   &i11)')
242:   &      ) IDTASK, NGENE, IRAND
243:   end if
244: C/#IF PARA(SX* CRAY)
245: *   call MVPSYNC_UNLOCK(2)
246: C/#ENDIF
247: C/#ELSE
248: C
249:   if ( JBPRNT.ne.0 ) then
250:     write(IOW,'(1x,'' GENERATE '' ,i6,'' SOURCES.   IRAND = '' ,i11)')
251:   &      ) NGENE, IRAND
252:   end if
253: C/#ENDIF
254: C
255: C
256: C
257: C =====
258: C FIXED SOURCE PROBLEM OR INITIAL SOURCE FOR EIGENVALUE PROBLEM
259: C =====
260: C

```

src/gmvp/srcold.f

```

261: C
262: C
263:       if ( JEIGN.eq.0 .or. NBATCH.le.0 ) then
264: C
265: C .... DETERMINE PARTICLE NUMBERS FROM EACH SOURCE IF NSOUR > 1 .....
266: C
267:       if ( NSOUR.gt.1 ) then
268:           NSS = 0
269: *VOCL LOOP,SCALAR
270:       do 120 N = 1, NSOUR
271:           NNSRC(N) = NGENE*SOUR(N)
272:           XSOUR(N) = FLOAT(NGENE)*SOUR(N) - NNSRC(N)
273:           if ( N.gt.1 ) XSOUR(N) = XSOUR(N-1) + XSOUR(N)
274:           NSS = NSS + NNSRC(N)
275: 120      continue
276: C
277: C .... IF THE SUM OF NNSRC IS LESS THAN NGENE, THE REST OF SOURCE
278: C      PARTICLES ARE GENERATED WITH PROBABILITIES PROPORTIONAL TO THE
279: C      DECIMAL PARTS OF NGENE*SOUR OF EACH SOURCE. (XSOUR).
280: C
281:       NSS = NGENE - NSS
282:       if ( NSS.ne.0.0 ) then
283:           call RANU2( IRAND, R, ABS(NSS), ICON )
284: *VOCL LOOP,SCALAR
285:       do 150 K = 1, ABS(NSS)
286:           R(K) = R(K)*XSOUR(NSOUR)
287:           do 130 L = 1, NSOUR
288:               if ( XSOUR(L).gt.R(K) ) go to 140
289: 130          continue
290:               L = NSOUR
291: 140          NNSRC(L) = NNSRC(L) + SIGN(1,NSS)
292: 150          continue
293:       end if
294:       else
295:           NNSRC(1) = NGENE
296:       end if
297: C
298: C .... GENERATE PARTICLES .....
299: C
300:       NSS = 0
301:       IST = 0
302:       IPP = NDEAD
303:
304:       do 420 N = 1, NSOUR
305:
306:           NN = NNSRC(N)
307:           KS = KSOUR(N)
308:           if ( JBPRINT.ne.0.and.JMNTR.ne.0 ) write(IOW,*)
309:           &      ' SOURCE # ', N, ': ', NN, ' PARTICLES'
310:           if ( JVMNT.ne.0 ) call VMNTR1( 1, NN )
311:           IPP = IPP - NN
312:           if ( JLATT.ne.0 ) then
313:               do 160 I = 1, NN
314:                   LEVL(LSDED(IPP+I)) = 0
315: 160          continue
316:               end if
317:               do 170 I = 1, NN
318:                   KLSF(LSDED(IPP+I)) = 0
319: 170          continue
320: C
321: C
322: C ===== SAMPLING OF POSITIONS =====
323: C
324: C
325: C

```

```

326: C TYPE 1 ..... ISOTROPIC POINT SOURCE .....
327: C
328: C TYPE 2 ..... MONO DIRECTIONAL POINT SOURCE .....
329: C      PSPAC(1:3) ... COORDINATE, PSPAC(4:6) ... DIRECTION
330: C
331:       if ( KS.eq.1 .or. KS.eq.2 ) then
332:           do 180 I = 1, NN
333:               IWK1(NSS+I) = LSDED(IPP+I)
334:               if ( ISZON(1,N).ne.0 ) IZZ(LSDED(IPP+I)) =
335:               &      ISZON(1,N)
336:               XXX(LSDED(IPP+I)) = PSPAC(1,N)
337:               YYY(LSDED(IPP+I)) = PSPAC(2,N)
338:               ZZZ(LSDED(IPP+I)) = PSPAC(3,N)
339: 180          continue
340: C
341: C TYPE 3 ..... RECTANGULAR SOURCE .....
342: C      PSPAC(1)<X<PSPAC(2), PSPAC(3)<Y<PSPAC(4), PSPAC(5)<Z<PSPAC(6)
343: C TYPE 4 ..... SPHERICAL SOURCE .....
344: C      PSPAC(1:3) = CENTER, PSPAC(4) = RADIUS
345: C
346:       else if ( KS.eq.3 .or. KS.eq.4 ) then
347:           call RANU2( IRAND, R, 3*NN, ICON )
348:           do 190 I = 1, NN
349:               IWK1(NSS+I) = LSDED(IPP+I)
350:               if ( ISZON(1,N).ne.0 ) IZZ(LSDED(IPP+I)) =
351:               &      ISZON(1,N)
352:               if ( KS.eq.3 ) then
353:                   XXX(LSDED(IPP+I)) = PSPAC(1,N) + R(I)*
354:                   &      (PSPAC(2,N)-PSPAC(1,N))
355:                   YYY(LSDED(IPP+I)) = PSPAC(3,N) + R(NN+I)*
356:                   &      (PSPAC(4,N)-PSPAC(3,N))
357:                   ZZZ(LSDED(IPP+I)) = PSPAC(5,N) + R(2*NN+I)*
358:                   &      (PSPAC(6,N)-PSPAC(5,N))
359:               else if ( KS.eq.4 ) then
360:                   DR = PSPAC(4,N)*(R(I)**(1/3.0D0))
361:                   D1 = 2.0*R(NN+I) - 1.0
362:                   D2 = SQRT(1.0-D1**2)
363:                   DD = 6.283185307179586D0*R(2*NN+I)
364:                   XXX(LSDED(IPP+I)) = DR*D1
365:                   YYY(LSDED(IPP+I)) = DR*D2*COS(DD)
366:                   ZZZ(LSDED(IPP+I)) = DR*D2*SIN(DD)
367:               end if
368: 190          continue
369: C
370: C TYPE 5 ..... CIRCULAR SOURCE MONO-DIRECTIONAL TEMPORARY.....
371: C      PSPAC(1:3) = CENTER, PSPAC(4) = RADIUS
372: C
373:       else if ( KS.eq.5 ) then
374:           call RANU2( IRAND, R, 2*NN, ICON )
375:           do 200 I = 1, NN
376:               IWK1(NSS+I) = LSDED(IPP+I)
377:               if ( ISZON(1,N).ne.0 ) IZZ(LSDED(IPP+I)) =
378:               &      ISZON(1,N)
379:               if ( PSPAC(5,N).eq.0. ) then
380:                   DAX = 1.0
381:                   DAY = 0.0
382:                   DAZ = 0.0
383:               else if ( PSPAC(6,N).eq.0. ) then
384:                   DAX = 0.0
385:                   DAY = 1.0
386:                   DAZ = 0.0
387:               else if ( PSPAC(7,N).eq.0. ) then
388:                   DAX = 0.0
389:                   DAY = 0.0
390:                   DAZ = 1.0

```

src/gmvp/srcold.f

```

391:      else
392:        DD1 = SQRT(PSPAC(6,N)**2+PSPAC(7,N)**2)
393:        DAX = 0.0
394:        DAY = PSPAC(7,N) /DD1
395:        DAZ = -PSPAC(6,N) /DD1
396:      end if
397:      DBX = PSPAC(6,N)*DAZ - PSPAC(7,N)*DAY
398:      DBY = PSPAC(7,N)*DAX - PSPAC(5,N)*DAZ
399:      DBZ = PSPAC(5,N)*DAY - PSPAC(6,N)*DAX
400:      DD2 = SQRT(DBX**2+DBY**2+DBZ**2)
401:      DBX = DBX/DD2
402:      DBY = DBY/DD2
403:      DBZ = DBZ/DD2
404: C
405:      DR = PSPAC(4,N)*SQRT(R(I))
406:      DD = 6.283185307179586D0*R(NN+I)
407:      D1 = COS(DD)*DR
408:      D2 = SIN(DD)*DR
409:      XXX(LSDED(IPP+I)) = DAX*D1 + DBX*D2 + PSPAC(1,N)
410:      YYY(LSDED(IPP+I)) = DAY*D1 + DBY*D2 + PSPAC(2,N)
411:      ZZZ(LSDED(IPP+I)) = DAZ*D1 + DBZ*D2 + PSPAC(3,N)
412: 200      continue
413: C
414: C TYPE 6 ..... SQUARE PLANE SOURCE MONO-DIRECTIONAL .....
415: C PSPAC(1:3) = VERTEX, PSPAC(4:6) = VECTOR OF 1 PENN
416: C PSPAC(7 ) = LENGTH OF OTHER :PSPAC(4:6)X:PSPAC(8,10)
417: C PSPAC(8:10)= MONO-DIRECTION PERPENDICULAR TO PSPAC(4:6)
418: C
419:      else if ( KS.eq.6 ) then
420:        call RANU2( IRAND, R, 2*NN, ICON )
421:        do 210 I = 1, NN
422:          IWK1(NSS+I) = LSDED(IPP+I)
423:          if ( ISZON(1,N).ne.0 ) IZZ(LSDED(IPP+I)) =
424:            & ISZON(1,N)
425:          DBX = PSPAC(5,N)*PSPAC(10,N) - PSPAC(6,N)*
426:            & PSPAC(9,N)
427:          DBY = PSPAC(6,N)*PSPAC(8,N) - PSPAC(4,N)*
428:            & PSPAC(10,N)
429:          DBZ = PSPAC(4,N)*PSPAC(9,N) - PSPAC(5,N)*
430:            & PSPAC(8,N)
431:          DD2 = SQRT(DBX**2+DBY**2+DBZ**2)
432:          DBX = DBX/DD2
433:          DBY = DBY/DD2
434:          DBZ = DBZ/DD2
435: C
436:          D1 = R(I)
437:          D2 = PSPAC(7,N)*R(I+NN)
438:          XXX(LSDED(IPP+I)) = PSPAC(4,N)*D1 + DBX*D2
439:            & + PSPAC(1,N)
440:          YYY(LSDED(IPP+I)) = PSPAC(5,N)*D1 + DBY*D2
441:            & + PSPAC(2,N)
442:          ZZZ(LSDED(IPP+I)) = PSPAC(6,N)*D1 + DBZ*D2
443:            & + PSPAC(3,N)
444: 210      continue
445: C
446:      end if
447: C
448: C
449: C ===== SAMPLING OF DIRECTIONS =====
450: C
451: C
452:      if ( KS.eq.1 ) then
453:        call RANU2( IRAND, R, 2*NN, ICON )
454: C
455: C ISOTROPIC BETWEEN PSPAC(9) AND PSPAC(10)

```

```

456: C      PSPAC(9)<= CCC <= PSPAC(10)
457: C      PSPAC(9,N) : MINIMUM ANGLE COSINE
458: C      PSPAC(10,N) : MAXIMUM ANGLE COSINE
459: C
460:      DUMIN = PSPAC(9,N)
461:      DUMAX = PSPAC(10,N)
462:      do 220 I = 1, NN
463:        D = (DUMAX-DUMIN)*R(I) + DUMIN
464:        CCC(LSDED(IPP+I)) = D
465:        D = DSQRT(1.0-D**2)
466:        D2 = 6.283185307179586D0*R(NN+I)
467:        BBB(LSDED(IPP+I)) = COS(D2)*D
468:        AAA(LSDED(IPP+I)) = SIN(D2)*D
469: 220      continue
470: C
471:      else if ( KS.eq.3 .or. KS.eq.4 ) then
472: C
473:        call RANU2( IRAND, R, 2*NN, ICON )
474: C
475: C ISOTROPIC
476: C
477:        do 230 I = 1, NN
478:          D = 2*R(I) - 1.0
479:          CCC(LSDED(IPP+I)) = D
480:          D = DSQRT(1.0-D**2)
481:          D2 = 6.283185307179586D0*R(NN+I)
482:          BBB(LSDED(IPP+I)) = COS(D2)*D
483:          AAA(LSDED(IPP+I)) = SIN(D2)*D
484: 230      continue
485: C added by Y.Nagaya 97/10/22
486: C This parameters are necessary for point detector.
487:        DUMAX = 1.0
488:        DUMIN = -1.0
489: C end of addition
490: C
491: C ..... MONO DIRECTIONAL .....
492: C
493:      else if ( KS.eq.2 .or. KS.eq.6 ) then
494:        if ( KS.eq.2 ) then
495:          DDDA = PSPAC(4,N)
496:          DDDB = PSPAC(5,N)
497:          DDDC = PSPAC(6,N)
498:        else if ( KS.eq.6 ) then
499:          DDDA = PSPAC(8,N)
500:          DDDB = PSPAC(9,N)
501:          DDDC = PSPAC(10,N)
502:        end if
503:        D = SQRT(DDDA**2+DDDB**2+DDDC**2)
504:        DDDA = DDDA/D
505:        DDDB = DDDB/D
506:        DDDC = DDDC/D
507:        do 240 I = 1, NN
508:          AAA(LSDED(IPP+I)) = DDDA
509:          BBB(LSDED(IPP+I)) = DDDB
510:          CCC(LSDED(IPP+I)) = DDDC
511: 240      continue
512: C
513: C
514: C
515: C ===== SAMPLING OF ENERGY, WEIGHT & TIME =====
516: C
517: C
518:      if ( KS.gt.0.and.KS.ne.9 ) then
519:        call RANU2( IRAND, R, 2*NN, ICON )
520:        do 250 I = 1, NN

```

src/gmvp/srcold.f

```

521:
522: C/#IF ROUND OFF(NEAREST)
523:          KK      = MIN(INT(NGROUP*R(I))+1,NGROUP)
524: C/#ELSE
525: *          KK      = NGROUP*R(I) + 1
526: C/#ENDIF
527:
528: C      .... SAMPLING FROM FISSION SPECTRUM      ....
529:          if ( JEIGN.ne.0.and.NBATCH.eq.0 ) then
530:              KKK      = KK*2 + R(NN+I) - FKAI(KK,IFISM(N))
531:              IGG(LSDED(IPP+I)) = KKAI(KKK,1,IFISM(N))
532: C      .... SAMPLING FROM PENRG      ....
533:          else
534:              KKK      = KK*2 + R(NN+I) - PENRG(KK,N)
535:              IGG(LSDED(IPP+I)) = KENRG(KKK,1,N)
536:          end if
537:          WWW(LSDED(IPP+I)) = 1.0
538:          TTT(LSDED(IPP+I)) = 0.0D0
539: 250      continue
540:      end if
541: C
542: C
543: C
544: C ===== SPECIAL CASES : POSITION, DIRECTION & ENERGY ARE SAMPLED
545: C          IN A SPECIAL ROUTINE.
546: C
547: C
548: C
549: C TYPE 9 ..... FNS NEUTRON SOURCE ( AUG. 1988  M.SASAKI ) .....
550: C   PSPAC(1:3) ... COORDINATE, PSPAC(4) ... MAXIMUM RADIUS
551: C   PSPAC(5) ... ROTATION INDEX  PSPAC(6:8) ... ROTATION ANGLES(DEGREE)
552: C
553: C
554: C
555:          if ( KS.eq.9 ) then
556:              do 260 I = 1, NN
557:                  IWK1(NSS+I) = LSDED(IPP+I)
558: C              IF(ISZON(1,N).NE.0) IZZ(LSDED(IPP+I)) = ISZON(1,N)
559: 260          continue
560: C
561: C
562:              call FNSSOC( IRAND, NN, IGG, XXX, YYY, ZZZ, AAA, BBB,
563: &                      CCC, WWW, TTT, R, RWK1, IWK2, ENGYB, NGROUP,
564: &                      IWK1(NSS+1), PSPAC(5,N), PSPAC(1,N), PSPAC(2,N),
565: &                      PSPAC(3,N), PSPAC(4,N), PSPAC(6,N), PSPAC(7,N),
566: &                      PSPAC(8,N) )
567: C
568: C
569: C          ELSE
570: C              WRITE(6,*) '!!! INVALID SOURCE TYPE ',KSOUR(N),
571: C              ' . STOP '
572: C              STOP
573: C          end if
574: C
575: C
576:          if ( KS.lt.0 ) then
577:              do 270 I = 1, NN
578:                  IWK1(NSS+I) = LSDED(IPP+I)
579: 270          continue
580:              ISTIM1 = ISTIM(N)
581:              ISTIM2 = ISTIM(N) - N + 1
582:              ISANG1 = ISANG(N)
583:              ISANG2 = ISANG(N) - N + 1
584:              if ( KS.eq.-1 ) then
585:                  call SRCU1( IOW, NN, N, IRAND, IWK1(NSS+1), NGROUP,

```

```

586: &                      NGP1, NGP2, NBANK, ENGYB, ENGPB, XXX, YYY,
587: &                      ZZZ, AAA, BBB, CCC, IGG, WWW, TTT, KSOUR(N),
588: &                      PSPAC(1,N), PENRG(1,N), KENRG(1,1,N), NSTIM,
589: &                      STIM(ISTIM1), PSTIM(ISTIM2), NSANG,
590: &                      SANG(ISANG1), PSANG(ISANG2), SAXIS(1,N), X, Y,
591: &                      Z, IWK2, IFI, IFL, RWK1, R )
592:          else if ( KS.eq.-2 ) then
593:              call SRCU2( IOW, NN, N, IRAND, IWK1(NSS+1), NGROUP,
594: &                      NGP1, NGP2, NBANK, ENGYB, ENGPB, XXX, YYY,
595: &                      ZZZ, AAA, BBB, CCC, IGG, WWW, TTT, KSOUR(N),
596: &                      PSPAC(1,N), PENRG(1,N), KENRG(1,1,N), NSTIM,
597: &                      STIM(ISTIM1), PSTIM(ISTIM2), NSANG,
598: &                      SANG(ISANG1), PSANG(ISANG2), SAXIS(1,N), X, Y,
599: &                      Z, IWK2, IFI, IFL, RWK1, R )
600:          end if
601:      end if
602: C
603: C
604: C      ==== DETERMINE STARTING ZONE #
605: C          AND SEND THEM TO APPROPRIATE STACKS. =====
606: C
607: C <COMMENT> 'NSST' INDICATES THE NUMBER OF REALLY GENERATED SOURCE
608: C (6/14/1991) PARTICLES ( NSST - NSS IS THE NUMBER OF 'REAL' SOURCE
609: C PARTICLES (I.E. UNLOST SOURCE PARTICLES)).
610: C          TO BE USED IN THE UNCOLLIDED FLUX CALCULATION.
611: C
612: C
613:          NSST      = NSS
614: C
615: C
616: C      ==== WHEN ZONE NUMBER IS SPECIFIED =====
617: C
618:          if ( ISZON(1,N).ne.0 ) then
619:              IZ      = ISZON(1,N)
620: C
621: C          ..... FREE FLIGHT STACK .....
622: C
623:          if ( KZMAT(IZ,1).ge.0 ) then
624:              do 280 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NN
625:                  LSFFL(I) = IWK1(NSS+I-NFFL(NZONE+1))
626:                  IZFFL(I) = IZ
627:                  IZZ(LSFFL(I)) = IZ
628:                  IREG      = KZREG(IZ)
629: C              ... set IBREG anytime (from Jun 2000)
630:                  IBREG(LSFFL(I)) = IREG
631:                  if ( JTLLT.ne.0 ) then
632:                      IBSPC(LSFFL(I),0) = 0
633:                  end if
634:                  if ( JIMPT.ne.0 ) then
635:                      XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),IREG)
636:                  end if
637: 280          continue
638:                  NFFL(IZ) = NFFL(IZ) + NN
639:                  NFFL(NZONE+1) = NFFL(NZONE+1) + NN
640:                  NSST      = NSST + NN
641: C
642: C          ..... INVALID MATERIAL ZONE ! .... (TREATED AS LOST)....
643: C
644: C          CCCCCC      else if ( KZMAT(IZ,1).le.-1000 ) then
645: C          else if ( KZMAT(IZ,1).le.-1000
646: C              .and..not.ISLATT(KZMAT(IZ)) ) then
647: C              write(IOW,7000) N, KZMAT(IZ,1), IZ, NN
648: C              ILOST      = ILOST + NN
649: C              IPP        = IPP + NN
650: C

```

src/gmvp/srcold.f

```

651: C      .... LATTICE SEARCH STACK ....
652: C
653: C      else
654: C      do 290 I = NXLT(NLBZ+1) + 1, NXLT(NLBZ+1) + NN
655: C          LSLAT(I) = IWK1(NSS+I-NXLT(NLBZ+1))
656: C          IZLAT(I) = MLBZZ(IZ)
657: C          IZZ(LSLAT(I)) = IZ
658: C          IREG = KZREG(IZ)
659: C      ... set IBREG anytime (from Jun 2000)
660: C      IBREG(LSLAT(I)) = IREG
661: C      if ( JTLT.ne.0 ) then
662: C          IBSPC(LSLAT(I),0) = 0
663: C      end if
664: C      if ( JIMPT.ne.0 ) then
665: C          if ( IREG.gt.0 ) then
666: C              XIM(LSLAT(I)) = XIMP(IGG(LSLAT(I)),IREG)
667: C          else
668: C              XIM(LSLAT(I)) = 1.0
669: C          end if
670: C      end if
671: C      290 continue
672: C      NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + NN
673: C      NXLT(NLBZ+1) = NXLT(NLBZ+1) + NN
674: C      NSST = NSST + NN
675: C      end if
676: C
677: C      === DETERMINE STARTING ZONE NUMBERS IF NOT SPECIFIED =====
678: C      else if ( ISZON(1,N).eq.0 ) then
679: C
680: C      do 300 I = 1, NN
681: C          IWK2(I) = LSDED(IPP+I)
682: C          X(I) = XXX(IWK2(I)) + 1.0D-8*AAA(IWK2(I))
683: C          Y(I) = YYY(IWK2(I)) + 1.0D-8*BBB(IWK2(I))
684: C          Z(I) = ZZZ(IWK2(I)) + 1.0D-8*CCC(IWK2(I))
685: C      300 continue
686: C
687: C      II = NN
688: C      KKZ = 1
689: C      MEM = 1
690: C      MMZ = NZONE
691: C      if ( JLATT.ne.0 ) MMZ = IPCEL(1) - 1
692: C
693: C      do 380 M = 1, MMZ
694: C          if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
695: C              if ( MEM.le.NMEMS.and.MEMZN(MEM,N).ne.0 ) then
696: C                  IZ = MEMZN(MEM,N)
697: C                  IM = 1
698: C              else
699: C                  IM = 0
700: C                  do 320 KZ = KKZ, MMZ
701: C                      do 310 K = 1, NMEMS
702: C                          if ( MEMZN(K,N).ne.0.and.KZ.eq.MEMZN(K,N)
703: C                              ) go to 320
704: C                      &
705: C                      310 continue
706: C                      go to 330
707: C                  320 continue
708: C                  KZ = MMZ
709: C                  330 IZ = KZ
710: C                  KKZ = IZ + 1
711: C              end if
712: C          else
713: C              IZ = M
714: C          end if
715: C

```

```

716: C =====
717: C =====
718: C =====
719: C
720: C      .... JUDGE WHETHER PARTICLES BELONG TO ZONE IZ OR NOT. ....
721: C
722: C          call JUDGE( 'SOURCE', IZ, II, X, Y, Z, IFI, SDA, KZDA,
723: C              &
724: C              &
725: C              KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
726: C              .false., DUMMY, DUMMY, DUMMY )
727: C =====
728: C =====
729: C
730: C      ..... SEND PARTICLES TO FLIGHT STACK .....
731: C
732: C      INN = 0
733: C      if ( KZMAT(IZ,1).ge.0 ) then
734: C          IFFL = NFFL(NZONE+1)
735: C          do 340 I = 1, II
736: C              if ( IFI(I) ) then
737: C                  IZZ(IWK2(I)) = IZ
738: C                  INN = INN + 1
739: C                  LSFFL(IFFL+INN) = IWK2(I)
740: C                  IZFFL(IFFL+INN) = IZ
741: C                  IREG = KZREG(IZ)
742: C              ... set IBREG anytime (from Jun 2000)
743: C              IBREG(IWK2(I)) = IREG
744: C              if ( JTLT.ne.0 ) then
745: C                  IBSPC(IWK2(I),0) = 0
746: C              end if
747: C              if ( JIMPT.ne.0 ) then
748: C                  XIM(IWK2(I)) = XIMP(IGG(IWK2(I)),IREG)
749: C              end if
750: C              if ( JPTDT.ne.0 ) IWK1(NSS+INN) = IWK2(I)
751: C              end if
752: C          340 continue
753: C
754: C          NFFL(IZ) = NFFL(IZ) + INN
755: C          NFFL(NZONE+1) = NFFL(NZONE+1) + INN
756: C
757: C          if ( JPTDT.ne.0 ) NSST = NSST + INN
758: C
759: C      ..... IF PARTICLES ARE IN A LATTICE, SEND THEM TO LATTICE STACK.
760: C
761: C      CCCC      else if ( KZMAT(IZ,1).le.-1.and.
762: C      CCCC &          KZMAT(IZ,1).ge.-998 ) then
763: C      else if ( ISLATT(KZMAT(IZ,1)) ) then
764: C          ILAT = NXLT(NLBZ+1)
765: C          do 350 I = 1, II
766: C              if ( IFI(I) ) then
767: C                  IZZ(IWK2(I)) = IZ
768: C                  INN = INN + 1
769: C                  LSLAT(ILAT+INN) = IWK2(I)
770: C                  IZLAT(ILAT+INN) = MLBZZ(IZ)
771: C                  IREG = KZREG(IZ)
772: C              ... set IBREG anytime (from Jun 2000)
773: C              IBREG(IWK2(I)) = IREG
774: C              if ( JTLT.ne.0 ) then
775: C                  IBSPC(IWK2(I),0) = 0
776: C              end if
777: C              if ( JIMPT.ne.0 ) then
778: C                  if ( IREG.gt.0 ) then
779: C                      XIM(IWK2(I)) =
780: C                      &
781: C                      XIMP(IGG(IWK2(I)),IREG)

```


src/gmvp/srcold.f

```

781:      else
782:        XIM(IWK2(I)) = 1.0
783:      end if
784:    end if
785:    if ( JPTDT.ne.0 ) IWK1(NSS+INN) = IWK2(I)
786:  end if
787: 350    continue
788: C
789:      NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + INN
790:      NXLT(NLBZ+1) = ILAT + INN
791: C
792:      if ( JPTDT.ne.0 ) NSST = NSST + INN
793: C
794: C
795: C ..... GENERATING PARTICLES IN INVALID MATERIAL ? .....
796: C
797:      else
798:        do 360 I = 1, II
799:          if ( .IFI(I) ) then
800:            INN = INN + 1
801:            ILOST = ILOST + 1
802:            LSDED(IPP+I) = IWK2(I)
803:          end if
804:        360    continue
805:            IPP = IPP + INN
806: C/#IF PARA(SX* CRAY)
807: *      call MVPSYNC_LOCK(2)
808: C/#ENDIF
809:      if ( INN.gt.0 ) write(IOW,7000) N,KZMAT(IZ,1),IZ,
810:      &      INN
811: 7000    format(/1X,' !!! CAUTION: YOU ARE GOING TO',
812:      &      ' GENERATE PARTICLES IN A ZONE WHICH PARTICLES',
813:      &      ' SHOULD NOT BE IN !!! '/1X,' SOURCE NO. ',
814:      &      I3,' MAT = ',I5,' ZONE = ',I5,' ',I8,
815:      &      ' PARTICLES. THEY ARE TREATED AS LOST!!!')
816: C/#IF PARA(SX* CRAY)
817: *      call MVPSYNC_UNLOCK(2)
818: C/#ENDIF
819:      end if
820: C
821: C ..... MEMORIZE NEWLY FOUND ZONE .....
822: C
823:      if ( JEIGN.eq.0 .or. NBATCH.gt.0 ) then
824:        if ( IM.ne.0 ) then
825:          MEM = MEM + 1
826:        else if ( INN.ne.0.and.MEM.le.NMEMS
827:      &      .and.MEMZN(MEM,N).eq.0 ) then
828:          MEMZN(MEM,N) = IZ
829:          MEM = MEM + 1
830:        end if
831:      end if
832: C
833: C ..... COMPRESS .....
834: C
835:      if ( INN.lt.II ) then
836:        INNN = 0
837: *VOCL LOOP,NOVREC
838:      do 370 I = 1, II
839:        if ( .not.IFI(I) ) then
840:          INNN = INNN + 1
841:          IWK2(INNN) = IWK2(I)
842:          X(INNN) = X(I)
843:          Y(INNN) = Y(I)
844:          Z(INNN) = Z(I)
845:        end if

```

```

846: 370    continue
847:      II = INNN
848:    else
849:      II = 0
850:      go to 390
851:    end if
852: 380    continue
853: C
854: C ..... ZONES NOT FOUND !! (LOST) .....
855: 390    continue
856: C
857:      if ( II.ne.0 ) then
858: C
859: C
860:      write(IOW,7020) II, N, (XXX(IWK2(I)),YYY(IWK2(I)),
861:      &      ZZZ(IWK2(I)),AAA(IWK2(I)),BBB(IWK2(I)),
862:      &      CCC(IWK2(I)),I=1,II)
863: 7020    format(/1X,' **** ',I5,
864:      &      ' PARTICLES ARE LOST IN SOURCE ROUTINE!'/1X,
865:      &      ' SOURCE NUMBER = ',I5/(1X,' X = ',
866:      &      1P,E12.5,' Y = ',E12.5,' Z = ',E12.5,' MU = ',
867:      &      E12.5,' ETA = ',E12.5,' XI = ',E12.5))
868: C
869: C
870:      do 400 I = 1, II
871:        LSDED(IPP+I) = IWK2(I)
872:      400    continue
873:        ILOST = ILOST + II
874:        IPP = IPP + II
875:      end if
876:    end if
877: C
878: C ..... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS ....
879: C
880: C ..... KSOUR2 : =0 ISOTROPIC SOURCE
881: C ..... .NE.0 SOURCE TYPE
882: C
883:      if ( JPTDT.ne.0 ) then
884:        KSOUR2 = 0
885:        if ( KS.eq.2 .or. KS.eq.9 ) KSOUR2 = KS
886: C
887:      call SUNCL( IOW, NSST-NSS, IWK1(NSS+1), KSOUR2,
888:      &      PSPAC(1,N), DUMAX, DUMIN, NPDET, NPLEN,
889:      &      NIMPT, NDIMPT, KDBNK(0),KIBNK(0),
890:      &      TIMEB, KZMAT, STOTX, VEL,
891:      &      JPUSD, JSPDT, XPDET, IPDET, IPDT2, IMSFL,
892:      &      IMNFL, IMZFL, IMDED, IMSLT, IMNLT, IMZLT, XXX,
893:      &      YYY, ZZZ, AAA, BBB, CCC, WWW, IZZ, IGG, TTT,
894:      &      ITT, LEVL, LZZ, LPOS, LCRS, LZZI, LPOSI, LCRSI,
895:      &      OPTI, PATH, KDETP, KLSFI,
896:      &      X, Y, Z, R, R(2*NBANK+1), R(4*NBANK+1),
897:      &      RH(LXYZ(1)),RH(LXYZ(2)),RH(LXYZ(3)),RH(LXYZ(4)),
898:      &      RH(LXYZ(5)),RH(LXYZ(6)),RH(LXYZ(7)),RH(LXYZ(8)),
899:      &      RH(LXYZ(9)),RH(LXYZ(10)),
900:      &      IFI, IFL, RWK1, IWK2 )
901:    end if
902: C
903: C ... save source set # if necessary as optional bank parameter
904: C
905:      if ( KIBNK(1).ne.0 ) then
906:        do 410 I = 1, NN
907:          IBNK(IWK1(NSS+I),KIBNK(1)) = N
908:        410    continue
909:      end if
910: C

```

src/gmvp/srcold.f

```

911:          NSS      = NSS + NN
912: C
913: 420      continue
914: C
915: C .... UPDATE SUM OF WEIGHTS .....
916: C
917:          DWSUM      = 0.0D0
918:          do 430 I = 1, NGENE
919:              DWSUM      = DWSUM + WWW(IWK1(I))
920: 430      continue
921:          WSUM      = WSUM + DWSUM
922:          if ( JEIGN.ne.0 ) XSOC(NBATCH+1) = WSUM
923:          NCNTR(1) = NCNTR(1) + NGENE
924: cccccc      WCNTR(1) = WSUM
925:          WCNTR(1) = WCNTR(1) + DWSUM
926: C
927: C
928: C =====
929: C SOURCES FROM FISSION NEUTRON BANK FOR EIGENVALUE PROBLEM
930: C =====
931: C
932: C
933:      else
934: C
935: C ... SOURCES ARE TAKEN FROM FISSION BANK (JEIGN.EQ.1.AND.NBATCH>0)
936: C
937:          if ( NFISB.le.0 ) then
938: C/#IF PARA
939: C/#IF PARA(SX* CRAY)
940: *          call MVPSYNC_LOCK(2)
941: C/#ENDIF
942:          write(IOW,7040) IDTASK
943: 7040      format(/1X,' TASK ',I5,
944: &          ' !!! NO FISSION SITE HAS BEEN SELECTED. STOP.',
945: &          ' !!!')
946: C/#IF PARA(SX* CRAY)
947: *          call MVPSYNC_UNLOCK(2)
948: C/#ENDIF
949: C/#ELSE
950:          write(IOW,7060)
951: 7060      format(/1X,' !!! NO FISSION SITE HAS BEEN SELECTED. STOP.',
952: &          ' !!!')
953: C/#ENDIF
954:          stop 666
955:      end if
956: C
957:          if ( NFISB.le.NGENE ) then
958:              do 440 I = 1, NFISB
959:                  IWK2(I) = I
960: 440      continue
961:              if ( NFISB.lt.NGENE ) then
962:                  call RANU2( IRAND, R, NGENE-NFISB, ICON )
963:                  do 450 I = 1, NGENE - NFISB
964:
965: C/#IF ROUNDOFF(NEAREST)
966:          IP      = MIN(INT(NFISB*R(I))+1,NFISB)
967: C/#ELSE
968: *          IP = NFISB*R(I) + 1
969: C/#ENDIF
970:          IWK2(NFISB+I) = IP
971: 450      continue
972:      end if
973:      else
974:          call RANU2( IRAND, R, NGENE, ICON )
975:          do 460 I = 1, NGENE

```

```

976:
977: C/#IF ROUNDOFF(NEAREST)
978:          IP      = MIN(INT(NFISB*R(I))+1,NFISB)
979: C/#ELSE
980: *          IP = NFISB*R(I) + 1
981: C/#ENDIF
982:
983:          IWK2(I) = IP
984: 460      continue
985:      end if
986: C
987:          call RANU2( IRAND, R, 4*NGENE, ICON )
988: C
989:          IPP      = NDEAD - NGENE
990:          NGEN2     = NGENE*2
991:          NGEN3     = NGENE*3
992:          WSUMF     = 0.0
993: *VOCL LOOP,NOVREC
994:          do 470 I = 1, NGENE
995:              IWK1(I) = LSDED(IPP+I)
996:              if ( JLATT.ne.0 ) LEVL(IWK1(I)) = LEVLF(IWK2(I))
997: C
998:          XXX(IWK1(I)) = XXXF(IWK2(I))
999:          YYY(IWK1(I)) = YYF(IWK2(I))
1000:          ZZZ(IWK1(I)) = ZZZF(IWK2(I))
1001:          D      = 2.0*R(I) - 1.0
1002:          AAA(IWK1(I)) = D
1003:          D      = DSQRT(1.0-D**2)
1004:          D2     = 6.283185307179586D0*R(NGENE+I)
1005:          BBB(IWK1(I)) = COS(D2)*D
1006:          CCC(IWK1(I)) = SIN(D2)*D
1007:          IZZ(IWK1(I)) = IZZF(IWK2(I))
1008:          KLSF(IWK1(I)) = 0
1009: C
1010: C ... set IBREG anytime (from Jun 2000)
1011:          if ( JTLT.eq.0 ) then
1012:              IREG = KZREG(IZZ(IWK1(I)))
1013:              IBREG(IWK1(I)) = IREG
1014:              RWK1(I) = WGTf(IREG)
1015:          else
1016:              IBREG(IWK1(I)) = IBRGF(IWK2(I))
1017:              RWK1(I) = WGTf(IBREG(IWK1(I)))
1018:          end if
1019: C
1020:          WSUMF     = WSUMF + RWK1(I)
1021: C
1022:          TTT(IWK1(I)) = 0.0D0
1023: C
1024:          IMAT      = KZMAT(IZZ(IWK1(I)),1)
1025:          if ( IMAT.lt.0 ) IMAT = KZMAT(IZZ(IWK1(I)),2)
1026:
1027: C/#IF ROUNDOFF(NEAREST)
1028:          KK      = MIN(INT(NGROUP*R(NGEN2+I))+1,NGROUP)
1029:          KKK     = 2*KK
1030:          KKK     = MIN(KKK,INT(KKK+R(NGEN3+I)-FKAI(KK,IMAT)))
1031: C/#ELSE
1032: *          KK      = NGROUP*R(NGEN2+I) + 1
1033: *          KKK     = KK*2 + R(NGEN3+I) - FKAI(KK,IMAT)
1034: C/#ENDIF
1035:
1036:          IGG(IWK1(I)) = KKAI(KKK,1,IMAT)
1037: 470      continue
1038: C
1039:          WSUMF2    = FLOAT(NGENE)
1040:          RATIO     = WSUMF2/WSUMF

```

src/gmvp/srcold.f

```

1041:      do 480 I = 1, NGENE
1042:      WWW(IWK1(I)) = RWK1(I)*RATIO
1043:      480      continue
1044: C
1045: C          .... LATTICE PARAMETER BANK ....
1046: C
1047:      if ( JLATT.ne.0 ) then
1048:      do 490 K = 1, NEST
1049: *VOCL LOOP,NOVREC
1050:      do 490 I = 1, NGENE
1051:      LZZ(IWK1(I),K) = LZZF(IWK2(I),K)
1052:      LPOS(IWK1(I),K) = LPOSF(IWK2(I),K)
1053:      if ( JHLAT.ne.0 ) LCRSI(IWK1(I),K) = LCRSF(IWK2(I),K)
1054:      if ( JTLLT.ne.0 ) IBSPC(IWK1(I),K) = IBSPF(IWK2(I),K)
1055: 490      continue
1056:      end if
1057: C
1058: C .... MAKE FREE FLIGHT STACK .....
1059: C
1060:      do 500 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
1061:      LSFFL(I) = IWK1(I-NFFL(NZONE+1))
1062:      IZFFL(I) = IZZ(IWK1(I-NFFL(NZONE+1)))
1063:      if ( JIMPT.ne.0 ) then
1064:      if ( JTLLT.eq.0 ) then
1065:      XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),KZREG(IZFFL(I)))
1066:      else
1067:      XIM(LSFFL(I)) = XIMP(IGG(LSFFL(I)),IBREG(LSFFL(I)))
1068:      end if
1069:      end if
1070: 500      continue
1071: C
1072:      do 510 I = NFFL(NZONE+1) + 1, NFFL(NZONE+1) + NGENE
1073:      NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
1074: 510      continue
1075: C
1076:      NFFL(NZONE+1) = NFFL(NZONE+1) + NGENE
1077: C
1078: C .... UPDATE SUM OF WEIGHTS .....
1079: C
1080:      WSUM = WSUM + WSUMF2
1081:      XSOC(NBATCH+1) = WSUMF2
1082:      NCNTR(1) = NCNTR(1) + NGENE
1083: cccccccc WCNTR(1) = WSUM
1084:      WCNTR(1) = WCNTR(1) + WSUMF2
1085: C
1086: C ..... UNCOLLIDED CONTRIBUTION TO NEXT EVENT ESTIMATORS .....
1087: C
1088: C ..... KSOUR2 : =0 ISOTROPIC SOURCE
1089: C          .NE.0 SOURCE TYPE
1090: C
1091:      LLLLL = NFFL(NZONE+1) - NGENE + 1
1092: C
1093: C
1094: C ..... GENERATE IMAGINARY PARTICLES TO CALCULATE UNCLLIDED
1095: C          CONTRIBUTION TO NEXT EVENT DETECTORS .....
1096: C
1097: C
1098:      if ( JPTDT.ne.0 ) then
1099:      call SUNCL( IOW, NGENE, LSFFL(NFFL(NZONE+1)-NGENE+1), 0,
1100:      &          PSPAC, +1.D0, -1.D0,
1101:      &          NPDET, NPLEN, NIMPT, NDIMPT, KDBNK(0),KIBNK(0),
1102:      &          TIMEB, KZMAT, STOTX, VEL ,
1103:      &          JPUSD, JSPDT, XPDET, IPDET, IPDT2, IMSFL, IMNFL,
1104:      &          IMZFL, IMDED, IMSLT, IMNLT, IMZLT, XXX, YYY, ZZZ,
1105:      &          AAA, BBB, CCC, WWW, IZZ, IGG, TTT, ITT,

```

```

1106:      &          LEVL, LZZ, LPOS,
1107:      &          LCRS, LZZI, LPOSI, LCRSI, OPTI, PATH, KDET, KLSFI,
1108:      &          IFI, IFL, R(1), R(NBANK+1)
1109:      &          X, Y, Z, R, R(2*NBANK+1), R(4*NBANK+1),
1110:      &          RH(LXYZ(1)),RH(LXYZ(2)),RH(LXYZ(3)),RH(LXYZ(4)),
1111:      &          RH(LXYZ(5)),RH(LXYZ(6)),RH(LXYZ(7)),RH(LXYZ(8)),
1112:      &          RH(LXYZ(9)),RH(LXYZ(10)),
1113:      &          IFI, IFL, RWK1, IWK2 )
1114:      end if
1115: C
1116: C
1117:      NFISB = 0
1118: C
1119:      end if
1120: C
1121: C
1122: C          ... treatment for optional bank parameters ....
1123: C
1124: C          ... weight on birth
1125: C
1126:      if ( KDBNK(1).ne.0 ) then
1127:      do 520 I = 1, NGENE
1128:      DBNK(IWK1(I),KDBNK(1)) = WWW(IWK1(I))
1129: 520      continue
1130:      end if
1131: C
1132: C          ... time on birth
1133: C
1134:      if ( KDBNK(2).ne.0 ) then
1135:      do 530 I = 1, NGENE
1136:      DBNK(IWK1(I),KDBNK(2)) = TTT(IWK1(I))
1137: 530      continue
1138:      end if
1139: C
1140: C          ... energy on birth
1141: C
1142:      if ( KDBNK(3).ne.0 ) then
1143:      do 481 I = 1, NGENE
1144:      DBNK(IWK1(I),KDBNK(3)) = EEE(IWK1(I))
1145: c 481      continue
1146:      end if
1147: C
1148: C          ... region # on birth (but the region # may not be the real one
1149: C          in lattice geometry ...)
1150:      if ( KIBNK(2).ne.0 ) then
1151:      do 540 I = 1, NGENE
1152:      IBNK(IWK1(I),KIBNK(2)) = IBREG(IWK1(I))
1153: 540      continue
1154:      end if
1155: C
1156: C          ... zone # on birth (but the region # may not be the real one
1157: C          in lattice geometry ...)
1158:      if ( KIBNK(3).ne.0 ) then
1159:      do 550 I = 1, NGENE
1160:      IBNK(IWK1(I),KIBNK(3)) = IZZ(IWK1(I))
1161: 550      continue
1162:      end if
1163: C
1164: C          ... generation of particle (1 for primary particles)
1165: C
1166:      if ( KIBNK(4).ne.0 ) then
1167:      do 560 I = 1, NGENE
1168:      IBNK(IWK1(I),KIBNK(4)) = 1
1169: 560      continue
1170:      end if

```

src/gmvp/srcold.f

```

1171: C
1172: C ... Marker region flag
1173: C
1174:       if ( KIBNK(9).ne.0 ) then
1175:         do 564 J = 1, NMKREG
1176:           KI9 = KIBNK(9)+J-1
1177:           do 562 I = 1, NGENE
1178:             IBNK(IWK1(I),KI9) = 0
1179:           562 continue
1180:         564 continue
1181:       end if
1182: C
1183: C .... PARTICLES IN THE REST OF BANK ARE DEAD PARTICLES ! .....
1184: C
1185:       NDEAD = NDEAD - NGENE + ILOST
1186:       NLOST = NLOST + ILOST
1187: C*      NDEAD = NBANK - NGENE
1188: C*      DO 400 I=1,NDEAD
1189: C 400      LSDDED(I) = NGENE + 1
1190: C
1191: C .... CALCULATE & PRINT AVERAGED PARAMETERS .....
1192: C
1193:       XAV = 0.0
1194:       YAV = 0.0
1195:       ZAV = 0.0
1196:       AAV = 0.0
1197:       BAV = 0.0
1198:       CAV = 0.0
1199:       EAV = 0.0
1200:       do 570 I = 1, NGENE
1201:         XAV = XAV + XXX(IWK1(I))*WWW(IWK1(I))
1202:         YAV = YAV + YYY(IWK1(I))*WWW(IWK1(I))
1203:         ZAV = ZAV + ZZZ(IWK1(I))*WWW(IWK1(I))
1204:         AAV = AAV + AAA(IWK1(I))*WWW(IWK1(I))
1205:         BAV = BAV + BBB(IWK1(I))*WWW(IWK1(I))
1206:         CAV = CAV + CCC(IWK1(I))*WWW(IWK1(I))
1207:         EAV = EAV + FLOAT(IGG(IWK1(I)))*WWW(IWK1(I))
1208:       570 continue
1209:       XAVT(1) = XAVT(1) + XAV
1210:       XAVT(2) = XAVT(2) + YAV
1211:       XAVT(3) = XAVT(3) + ZAV
1212:       AAVT(1) = AAVT(1) + AAV
1213:       AAVT(2) = AAVT(2) + BAV
1214:       AAVT(3) = AAVT(3) + CAV
1215:       EAVT = EAVT + EAV
1216:       NTGEN = NTGEN + NGENE
1217: C
1218:       XAV = XAV/NGENE
1219:       YAV = YAV/NGENE
1220:       ZAV = ZAV/NGENE
1221: C
1222:       if ( JPRTS(6).ne.1 ) then
1223: C/#IF PARA(SX* CRAY)
1224:       *      call MVPSYNC_LOCK(2)
1225: C/#ENDIF
1226:       if ( JBPRNT.ne.0 ) then
1227: C/#IF PARA
1228:       write(IOW,'(1x,' TASK ',i5)') IDTASK
1229: C/#ENDIF
1230:       write(IOW,7080) '(BATCH) ', XAV, YAV, ZAV, AAV/NGENE,
1231:       &      BAV/NGENE, CAV/NGENE, EAV/NGENE
1232:       write(IOW,7080) '(CUMULATIVE)', XAVT(1)/NTGEN, XAVT(2)/
1233:       &      NTGEN, XAVT(3)/NTGEN, AAVT(1)/NTGEN, AAVT(2)/
1234:       &      NTGEN, AAVT(3)/NTGEN, EAVT/NTGENWRITE(IOW,*)
1235:       &      ' GENERATED PARTICLES IN THIS RUN = ', NTGEN

```

```

1236: C
1237:       end if
1238: C/#IF PARA(SX* CRAY)
1239:       *      call MVPSYNC_UNLOCK(2)
1240: C/#ENDIF
1241:       end if
1242: C
1243:       7080 format(1X,' AVERAGE ',A12,2X,' X=',1PE11.4,' Y=',E11.4,' Z=',
1244:       &      E11.4,' MU=',E11.4,' ETA=',E11.4,' XI=',E11.4,' IG=',E11.4
1245:       &      )
1246: C
1247: C IF(JDEBG.EQ.1) THEN
1248: C WRITE(6,7030) (I,XXX(I),YYY(I),ZZZ(I),AAA(I),BBB(I),CCC(I),
1249: C & WWW(I),IZZ(I),IGG(I),TTT(I),I=1,NGENE)
1250: C 7030 FORMAT(/1x,' GENERATED SOURCE PARAMETERS '/
1251: C & 1x,' XXX YYY ZZZ AAA BBB',
1252: C & ' CCC WWW IZZ IGG TTT '/
1253: C & (1x ,I6,1P6D11.4,E11.4,2I7,E11.4))
1254: C ENDIF
1255: CC
1256:       if ( JEIGN.ne.0 ) then
1257:         XVA = 0.0
1258:         YVA = 0.0
1259:         ZVA = 0.0
1260:         do 580 I = 1, NGENE
1261:           XVA = XVA + (XXX(IWK1(I))-XAV)*(XXX(IWK1(I))-XAV)*
1262:           &      WWW(IWK1(I))
1263:           YVA = YVA + (YYY(IWK1(I))-YAV)*(YYY(IWK1(I))-YAV)*
1264:           &      WWW(IWK1(I))
1265:           ZVA = ZVA + (ZZZ(IWK1(I))-ZAV)*(ZZZ(IWK1(I))-ZAV)*
1266:           &      WWW(IWK1(I))
1267:         580 continue
1268:         XSXV(NBATCH+1,1) = XVA/(NGENE-1)
1269:         XSXV(NBATCH+1,2) = YVA/(NGENE-1)
1270:         XSXV(NBATCH+1,3) = ZVA/(NGENE-1)
1271:         ccccccc XSXV(NBATCH+1) = XVA/(NGENE-1)
1272:         ccccccc XSYV(NBATCH+1) = YVA/(NGENE-1)
1273:         ccccccc XSZV(NBATCH+1) = ZVA/(NGENE-1)
1274: C/#IF PARA(SX* CRAY)
1275:       *      call MVPSYNC_LOCK(2)
1276: C/#ENDIF
1277:       if ( JBPRNT.ne.0 ) then
1278: C/#IF PARA
1279:       if ( JPRTS(6).ne.1 ) write(IOW,7100) IDTASK, '(BATCH) ',
1280:       &      XSXV(NBATCH+1,1), XSXV(NBATCH+1,2), XSXV(NBATCH+1,3)
1281:       7100 format(1X,' TASK ',I5,' VARIANCE ',A10,2X,' X= ',1PE11.4,
1282:       &      ' Y= ',E11.4,' Z= ',E11.4)
1283: C/#ELSE
1284:       if ( JPRTS(6).ne.1 ) write(IOW,7120) '(BATCH) ',
1285:       &      XSXV(NBATCH+1,1), XSXV(NBATCH+1,2), XSXV(NBATCH+1,3)
1286:       7120 format(1X,' VARIANCE ',A10,2X,' X= ',1PE11.4,' Y= ',E11.4,
1287:       &      ' Z= ',E11.4)
1288: C/#ENDIF
1289:       end if
1290: C/#IF PARA(SX* CRAY)
1291:       *      call MVPSYNC_UNLOCK(2)
1292: C/#ENDIF
1293: C
1294:       end if
1295: C
1296:       if ( JTIME.ne.0 ) then
1297:         do 590 I = 1, NGENE
1298:           ITT(IWK1(I)) = 1
1299:         590 continue
1300:       end if

```

src/gmvp/srcold.f

```
1301: C
1302:      return
1303:
1304: C/#ENDIF
1305:
1306:      end
```

SAFE

src/gmvp/srcul.f

```

1:      SUBROUTINE SRCU1
2:      N ( IOW , NN , ID , IRAND, IBP ,
3:      N   NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
4:      B   XXX , YYY , ZZZ , AAA , BBB , CCC ,
5:      B   IGG , WWW , TTT ,
6:      S   KSOUR, PSPAC, PENRG, KENRG,
7:      S   NSTIM, STIM , PSTIM,
8:      S   NSANG, SANG , PSANG, SAXIS,
9:      W   X,Y,Z,IWK1,IWK2,IWK3,IWK4, R )
10: C
11: C=<GMVP>=====
12: C PURPOSE: GENERATE PARTICLES (USER SOURCE ROUTINE #1 FOR GMVP)
13: C CALLED IN: SOURCE (KSOUR=-1)
14: C -----
15: C VARIABLES      TYPE      MEANINGS
16: C IOW            I4        LOGICAL UNIT FOR PRINTOUT
17: C NN             I4        NUMBER OF PARTICLES BEING GENERATED FROM NOW
18: C ID             I4        ID NUMBER OF THIS SOURCE ( 1 =< ID <= NSOUR )
19: C IRAND          I4        INITIAL RANDOM NUMBER
20: C IBP(NN)        I4        BANK POINTER FOR PARTICLES TO BE GENERATED
21: C                I4        DESCRIPTERS FOR N'TH PARTICLE ARE STORED IN
22: C                I4        IBP(N)'TH POSITION IN THE BANK.
23: C NGROUP         I4        NUMBER OF ENERGY GROUPS FOR TALLY AND SOURCE
24: C NGP1           I4        NUMBER OF ENERGY GROUPS OF NEUTRONS
25: C NGP2           I4        NUMBER OF ENERGY GROUPS OF PHOTONS
26: C NBANK          I4        SIZE OF PARTICLE BANK
27: C -----
28: C-----< PARTICLE BANK >: THESE ARRAYS SHOULD BE ASSIGNED VALUES -----
29: C XXX(NBANK)     R8        X-POSITION (CM)
30: C YYY(NBANK)     R8        Y-POSITION (CM)
31: C ZZZ(NBANK)     R8        Z-POSITION (CM)
32: C AAA(NBANK)     R8        X-DIRECTION COSINE
33: C BBB(NBANK)     R8        Y-DIRECTION COSINE
34: C CCC(NBANK)     R8        Z-DIRECTION COSINE
35: C WWW(NBANK)     R4        PARTICLE WEIGHT
36: C IGG(NBANK)     I4        ENERGY GROUP
37: C TTT(NBANK)     R8        TIME (SEC) MEANINGLESS IN THE CODE.
38: C-----< SOURCE DATA >-----
39: C KSOUR          I4        TYPE OF THE SOURCE.
40: C PSPAC(10)      R4        SPATIAL DISTRIBUTION. (INPUT VALUES)
41: C
42: C PENRG(NGROUP)  R4        ENERGY DISTRIBUTIONS (MODIFIED DISCRETE SAMPLING)
43: C KENRG(2,NGROUP) I4        ENERGY BIN # (FOR DISCRETE SAMPLING)
44: C
45: C NSTIM          I4        TIME BIN NUMBER OF SOURCE. (INPUT VALUE)
46: C STIM(NSTIM+1)  R4        TIME BINS OF SOURCE. (INPUT VALUES)
47: C PSTIM(NSTIM)   R4        TIME DISTRIBUTION. (INPUT VALUES)
48: C
49: C NSANG          I4        ANGLE NUMBERS OF SOURCE. (INPUT VALUES)
50: C SANG(NSANG+1)  R4        ANGLES (COSINE) BINS OF SOURCE. (INPUT VALUES)
51: C PSANG(NSANG)   R4        ANGULAR DISTRIBUTION. (INPUT VALUES)
52: C SAXIS(3)       R4        DIRECTION COSINES TO SOURCE GENERATION AXES.
53: C*
54: C=====
55: C
56: C .... BANK POINTER .....
57: C
58: C      INTEGER      IBP(NN)
59: C
60: C .... PARTICLE BANK .....
61: C
62: C      REAL*8        XXX(NBANK) , ZZZ(NBANK) , YYY(NBANK) , AAA(NBANK) ,
63: C      &             BBB(NBANK) , CCC(NBANK)
64: C      REAL          WWW(NBANK)
65: C      REAL*8        TTT(NBANK)

```

```

66:      INTEGER      IGG(NBANK)
67: C
68: C .... SOURCE DATA .....
69: C
70: C      INTEGER      KSOUR, KENRG(2,NGROUP) , NSTIM, NSANG
71: C      REAL          PSPAC(10) , PENRG(NGROUP) , ENGYB(NGP1+1) , ENGPB(NGP2+1) ,
72: C      &             STIM(NSTIM+1) , PSTIM(NSTIM) ,
73: C      &             SANG(NSANG+1) , PSANG(NSANG) , SAXIS(3)
74: C
75: C .... WORKING ARRAY FOR
76: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 4*NBANK FOR USE
77: C      REAL          R(4*NBANK)
78: C      INTEGER      IWK1(NBANK) , IWK2(NBANK) , IWK3(NBANK) , IWK4(NBANK)
79: C      REAL*8        X(NBANK) , Y(NBANK) , Z(NBANK)
80: C
81: C
82: C
83: C      WRITE(IOW,*) ' '
84: C      WRITE(IOW,*)
85: C      & ' XXX YOU ARE USING USER SOURCE ROUTINE #1 (SRC1) XXX'
86: C      WRITE(IOW,*)
87: C      & ' XXX THIS JOB HAS BEEN TERMINATED BY THE CODE XXX'
88: C      WRITE(IOW,*)
89: C      & ' XXX PLEASE REPLACE SRCU1 WITH YOURS XXX'
90: C
91: C      STOP 888
92: C
93: C      RETURN
94: C      END

```

src/gmvp/srcu2.f

```

1:      SUBROUTINE SRCU2
2:      N ( IOW , NN , ID , IRAND, IBP ,
3:      N   NGROUP, NGP1, NGP2 , NBANK, ENGYB, ENGPB,
4:      B   XXX , YYY , ZZZ , AAA , BBB , CCC ,
5:      B   IGG , WWW , TTT ,
6:      S   KSOUR, PSPAC, PENRG, KENRG,
7:      S   NSTIM, STIM , PSTIM,
8:      S   NSANG, SANG , PSANG, SAXIS,
9:      W   X,Y,Z,IWK1,IWK2,IWK3,IWK4, R )
10: C
11: C=<GMVP>=====
12: C PURPOSE: GENERATE PARTICLES (USER SOURCE ROUTINE #2 FOR GMVP)
13: C CALLED IN: SOURCE (KSOUR=-2)
14: C -----
15: C VARIABLES      TYPE      MEANINGS
16: C IOW            I4        LOGICAL UNIT FOR PRINTOUT
17: C NN             I4        NUMBER OF PARTICLES BEING GENERATED FROM NOW
18: C ID             I4        ID NUMBER OF THIS SOURCE ( 1 =< ID <= NSOUR )
19: C IRAND          I4        INITIAL RANDOM NUMBER
20: C IBP(NN)        I4        BANK POINTER FOR PARTICLES TO BE GENERATED
21: C                I4        DESCRIPTERS FOR N'TH PARTICLE ARE STORED IN
22: C                I4        IBP(N)'TH POSITION IN THE BANK.
23: C NGROUP         I4        NUMBER OF ENERGY GROUPS FOR TALLY AND SOURCE
24: C NGP1           I4        NUMBER OF ENERGY GROUPS OF NEUTRONS
25: C NGP2           I4        NUMBER OF ENERGY GROUPS OF PHOTONS
26: C NBANK          I4        SIZE OF PARTICLE BANK
27: C -----
28: C-----< PARTICLE BANK >: THESE ARRAYS SHOULD BE ASSIGNED VALUES -----
29: C XXX(NBANK)     R8        X-POSITION (CM)
30: C YYY(NBANK)     R8        Y-POSITION (CM)
31: C ZZZ(NBANK)     R8        Z-POSITION (CM)
32: C AAA(NBANK)     R8        X-DIRECTION COSINE
33: C BBB(NBANK)     R8        Y-DIRECTION COSINE
34: C CCC(NBANK)     R8        Z-DIRECTION COSINE
35: C WWW(NBANK)     R4        PARTICLE WEIGHT
36: C IGG(NBANK)     I4        ENERGY GROUP
37: C TTT(NBANK)     R8        TIME (SEC) MEANINGLESS IN THE CODE.
38: C-----< SOURCE DATA >-----
39: C KSOUR          I4        TYPE OF THE SOURCE.
40: C PSPAC(10)      R4        SPATIAL DISTRIBUTION. (INPUT VALUES)
41: C
42: C PENRG(NGROUP)  R4        ENERGY DISTRIBUTIONS (MODIFIED DISCRETE SAMPLING)
43: C KENRG(2,NGROUP) I4        ENERGY BIN # (FOR DISCRETE SAMPLING)
44: C
45: C NSTIM          I4        TIME BIN NUMBER OF SOURCE. (INPUT VALUE)
46: C STIM(NSTIM+1)  R4        TIME BINS OF SOURCE. (INPUT VALUES)
47: C PSTIM(NSTIM)   R4        TIME DISTRIBUTION. (INPUT VALUES)
48: C
49: C NSANG          I4        ANGLE NUMBERS OF SOURCE. (INPUT VALUES)
50: C SANG(NSANG+1)  R4        ANGLES (COSINE) BINS OF SOURCE. (INPUT VALUES)
51: C PSANG(NSANG)   R4        ANGULAR DISTRIBUTION. (INPUT VALUES)
52: C SAXIS(3)       R4        DIRECTION COSINES TO SOURCE GENERATION AXES.
53: C*
54: C=====
55: C
56: C .... BANK POINTER .....
57: C
58: C      INTEGER      IBP(NN)
59: C
60: C .... PARTICLE BANK .....
61: C
62: C      REAL*8        XXX(NBANK) , ZZZ(NBANK) , YYY(NBANK) , AAA(NBANK) ,
63: C      &             BBB(NBANK) , CCC(NBANK)
64: C      REAL          WWW(NBANK)
65: C      REAL*8        TTT(NBANK)

```

```

66:      INTEGER      IGG(NBANK)
67: C
68: C .... SOURCE DATA .....
69: C
70: C      INTEGER      KSOUR, KENRG(2,NGROUP) , NSTIM, NSANG
71: C      REAL          PSPAC(10) , PENRG(NGROUP) , ENGYB(NGP1+1) , ENGPB(NGP2+1) ,
72: C      &             STIM(NSTIM+1) , PSTIM(NSTIM) ,
73: C      &             SANG(NSANG+1) , PSANG(NSANG) , SAXIS(3)
74: C
75: C .... WORKING ARRAY FOR
76: C .... RANDOM NUMBER ETC. .( LENGTH OF R MUST BE 4*NBANK FOR USE
77: C      REAL          R(4*NBANK)
78: C      INTEGER      IWK1(NBANK) , IWK2(NBANK) , IWK3(NBANK) , IWK4(NBANK)
79: C      REAL*8        X(NBANK) , Y(NBANK) , Z(NBANK)
80: C
81: C
82: C
83: C      WRITE(IOW,*) ' '
84: C      WRITE(IOW,*)
85: C      & ' XXX YOU ARE USING USER SOURCE ROUTINE #2 (SRC2) XXX'
86: C      WRITE(IOW,*)
87: C      & ' XXX THIS JOB HAS BEEN TERMINATED BY THE CODE XXX'
88: C      WRITE(IOW,*)
89: C      & ' XXX PLEASE REPLACE SRCU2 WITH YOURS XXX'
90: C
91: C      STOP 888
92: C
93: C      RETURN
94: C      END

```

src/gmvp/staln1.f

```

1:      subroutine STALN1( IOW, JTIME, JPTIM, MZONE, IDTALY,
2:      I          FLX, NN,
3:      T          IEG, IRG, IBP, ITP, TIM, TI,
4:      N          NGROUP,NGP1,NREG,NRESP, NBANK,
5:      N          NSTAL, NZONE, NTIME,
6:      D          RESP, TIMEB, NMKREG, MKREG,
7:      R          DTALY,
8:      X          SGTAL, TCUT,
9:      B          DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
10:     W          DNF,IPP,IPMK, JPTIM2 )
11: C=<GMVP>=====
12: C PURPOSE: Take special tallies for track length or collision tally
13: C          (neutron/photon & multiple time)
14: C
15: C CALLED IN: FLIONE, COLISN
16: C=====
17: C arguments ( i = input, o=output, w=work )
18: C
19: C i IOW : message printout I/O unit
20: C i JTIME : problem is time dependent if non zero.
21: C i JPTIM : periodic time is used if non zero.
22: C i MZONE : zone #
23: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
24: C          routine.
25: C i FLX(NN) : flux or contribution to tally
26: C i NN : NUMBER OF PARTICLES
27: C i IEG(NN) : GROUP #
28: C i IRG(NN) : REGION #
29: C i IBP(NN) : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
30: C          JREAC=1 OR JEIGN.NE.0
31: C i ITP(NN) : current time bin of each particle
32: C i TIM(NN) : current time
33: C i TI(NN) : flight time tallied
34: C o DTALY(*) : tally
35: C
36: C w DNF(NN) : working area
37: C w IPP(NN) : working area
38: C w IPMK(NN) : working area
39: C i/o JPTIM2 : set unity in this routine when periodic time is applied
40: C-----
41: C
42: C Structure of IDTAL(*)
43: C
44: C +----- direct tally loop ( NDTALY )
45: C |
46: C | (1) ID
47: C | (2) (dummy for debug: -999)
48: C | (3) pointer to DTALY(*)
49: C | (4) event #
50: C | (5) particle type
51: C | (6) number of dimensions
52: C | (7) multiplication factor type
53: C | (8) nuclide/atom # & reaction #
54: C | (9) direct tally #
55: C | (10) custom tally # (non-zero means user customized treatment)
56: C | (11) detector # (for surface event positive tally-surface #
57: C |          means particle current tally, and negative one
58: C |          means flux tally)
59: C | ( IDTOFF data so far )
60: C |
61: C | +----- tally dimension loop
62: C | | (1) dimension type
63: C | | (2) number of tallied bins
64: C | | (3) size of common bin -> tallied bin conversion table
65: C | | ( IDMOFF data so far )
66: C | | +----- common bin loop
67: C | | | tallied bin # for each common bin
68: C | | +-----
69: C | +-----
70: C +-----
71: C=====
72: C          implicit real*8(A-H,O-Z)
73: C
74: C          integer IDTALY(*)
75: C
76: C          real*8 FLX(NN)
77: C          integer IEG(NN), IBP(NN), IRG(NN)
78: C          integer ITP(NN)
79: C
80: C          real*8 DBNK(NBANK,*)
81: C          integer KDBNK(0:MDBNK)
82: C          integer IBNK(NBANK,*)
83: C          integer KIBNK(0:MIBNK)
84: C          real SGTAL(NBANK,NSTAL)
85: C
86: C          real*8 TIM(NN), TI(NN)
87: C
88: C          real*8 DTALY(*)
89: C
90: C          real RESP(NGROUP,NRESP)
91: C          real TIMEB(NTIME+1)
92: C          real TCUT
93: C          integer MKREG(NREG,2)
94: C
95: C          integer IPP(NN)
96: C          integer IPMK(NN)
97: C          real*8 DNF(NN)
98: C
99: C ... offset of direct-tally data in IDTAL(*)
100: C
101: CC parameter( IDTOFF = 12 )
102: C
103: C ... offset of tally dimension dependent data in IDTAL(*)
104: C
105: CC parameter( IDMOFF = 3 )
106: C
107: C          include '../shared/INC/_IDXOFF'
108: C
109: C-----
110: C          IRI = IRG(1)
111: C          II = 0
112: C
113: C ... neutron tally ...
114: C
115: C          if ( IDTALY(5).eq.1 ) then
116: C              do 100 I = 1, NN
117: C                  if ( IEG(I).le.NGP1 ) then
118: C                      IPP(I) = 0
119: C                      II = II + 1
120: C                  else
121: C                      IPP(I) = -1
122: C                  end if
123: C              100 continue
124: C
125: C ... photon tally ...
126: C
127: C          else if ( IDTALY(5).eq.2 ) then
128: C              do 110 I = 1, NN
129: C                  if ( IEG(I).gt.NGP1 ) then
130: C                      IPP(I) = 0

```


src/gmvp/staln1.f

```

131:      II      = II + 1
132:      else
133:      IPP(I) = -1
134:      end if
135: 110 continue
136:      end if
137: C
138:      if ( II.eq.0 ) return
139: C
140: C ... this flag shows that time dependent tally is really necessary
141: C
142:      JTIME2 = 0
143: C
144: C ... ido is to check multiple time tally is finished or not
145: C      in time dependent case.
146: C
147:      IDO = NN
148:      ICYCLE = 0
149: C
150: C ... check marker region and marker region tally bin in IPMK(I) ...
151: C
152:      if ( NMKREG.gt.0 ) then
153:      IDT = IDTOFF
154:      do 120 IB = 1, IDTALY(6)
155:      KBIN = IDTALY(IDT+1)
156: C
157: C ... this tally has marker region
158:      if ( KBIN.eq.8 ) then
159:      IDTM = IDT
160:      go to 130
161:      end if
162:      IDT = IDT + IDMOFF + IDTALY(IDT+3)
163: 120 continue
164:      go to 170
165: 130 continue
166: C
167: C ... set non-marked as default
168:      do 140 I = 1, NN
169:      IPMK(I) = 2
170: 140 continue
171: C
172:      do 160 IM = 1, NMKREG
173:      KG = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
174: C
175: C ... this marker region is used in current tally ...
176: C
177:      if ( KG.eq.1 ) then
178:      KI9 = KIBNK(9) + IM - 1
179:      do 150 I = 1, NN
180:      if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I) = 1
181: 150 continue
182:      end if
183: 160 continue
184: C
185: 170 continue
186:      end if
187: C
188: C ... in a time dependent case, more than one cycles for a loop
189: C      that starts from the following label.
190: C
191: 180 continue
192: C
193:      if ( IDO.ge.0 ) then
194: C
195: C ... calculate position in dtaly(*)

```

```

196: C
197:      IDT = IDTOFF
198:      do 200 IB = 1, IDTALY(6)
199:      KBIN = IDTALY(IDT+1)
200:      do 190 I = 1, NN
201:      if ( IPP(I).ge.0 ) then
202:      K = 0
203: C
204: C ... only 1 dtaly bin
205: C
206:      if ( IDTALY(IDT+3).eq.0 ) then
207:      K = 1
208: C
209: C ... marker region
210: C
211:      else if ( KBIN.eq.8 ) then
212:      K = IPMK(I)
213:      else
214: C
215: C ... region
216: C
217:      if ( KBIN.eq.1 ) then
218:      K = IRG(I)
219: C
220: C ... energy
221: C
222:      else if ( KBIN.eq.2 ) then
223:      K = IEG(I)
224: C
225:      ... for photon ...
226:      if ( IDTALY(5).eq.2 ) K = K - NGP1
227: C
228: C ... time
229: C
230:      else if ( KBIN.eq.3 ) then
231:      K = ITP(I) + ICYCLE
232:      JTIME2 = 1
233: C
234: C CCCCCCCCCCCCCC cc else if( kbin.eq.4 ) then
235: C CCCCCCCCCCCCCC cc k = ???
236: C
237: C ... generation
238: C
239:      else if ( KBIN.eq.5 ) then
240:      K =
241: C      & MIN( IBNK( IBP(I), KIBNK(4) ), IDTALY( IDT+3 ) )
242: C
243: C ... source set
244: C
245:      else if ( KBIN.eq.6 ) then
246:      K = IBNK( IBP(I), KIBNK(1) )
247: C
248: C ... source region
249: C
250:      else if ( KBIN.eq.7 ) then
251:      K = IBNK( IBP(I), KIBNK(2) )
252:      end if
253: C
254: C ... convert common bin to d-tally bin
255: C
256:      K = IDTALY( IDT+IDMOFF+K )
257:      end if
258: C
259: C ... IPP(I) = -1 : not in tally range !!!
260: C
261:      if ( K.eq.0 ) then

```

src/gmvp/staln1.f

```

261:             IPP(I) = -1
262:             IDO      = IDO - 1
263:             else
264:                 IPP(I) = IPP(I)*IDTALY(IDT+2) + K - 1
265:             end if
266:         end if
267:     190     continue
268:         IDT      = IDT + IDMOFF + IDTALY(IDT+3)
269:     200     continue
270: C
271:         INUC      = IDTALY(8) /1024
272:         IMIC      = IDTALY(8) - INUC*1024
273: C
274:         if ( ICYCLE.eq.0 ) then
275: C
276: C         ... response ...
277: C
278:         if ( IDTALY(7).lt.0 ) then
279:             do 210 I = 1, NN
280:                 DNF(I) = FLX(I)*RESP(IEG(I),-IDTALY(7))
281:             210     continue
282: C
283: C         ... ???? ...
284:         else if ( idtaly(7).lt.1000 ) then
285:             Ccccccc
286:             dnf(i) = dnf(i) *
287: C         ... smic ...
288: Cc only in MVP
289: C         else if ( IDTALY(7).eq.1001 ) then
290: C             DNF(I) = DNF(I)*SMIC(IBP(I),INUC,LMIC(IMIC))*
291: C             &         DNZON(INUC,MZONE)
292: C
293: C         ... smic ... (per atom)
294: Cc only in MVP
295: C         else if ( IDTALY(7).eq.1002 ) then
296: C             DNF(I) = DNF(I)*SMIC(IBP(I),INUC,LMIC(IMIC))
297: C
298: C         ... sgtaal ...
299: C
300:         else if ( IDTALY(7).eq.2000 ) then
301:             do 220 I = 1, NN
302:                 DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
303:             220     continue
304: C
305: C         ... smac ...
306: Cc only in MVP
307: C         else if ( IDTALY(7).eq.3000 ) then
308: C             DNF(I) = DNF(I)*
309: C             &         SMAC(IBP(I),IMB(I),LMAC(IDTALY(8)))
310: C
311: C
312: C
313: C         ... flux ...
314: C
315:         else
316:             do 230 I = 1, NN
317:                 DNF(I) = FLX(I)
318:             230     continue
319:         end if
320:     end if
321: C
322: C --- if tally event type is not track length,
323: C no need to multi-cycle tally for time bin
324: C
325:         if ( IDTALY(4).ne.2 ) then

```

```

326:             JTIME2 = 0
327:         end if
328: C
329: C --- weight by time-bin occupation rate ---
330: C
331:         IP0      = IDTALY(3)
332:         if ( JTIME2.eq.0 ) then
333:             do 240 I = 1, NN
334:                 if ( IPP(I).ge.0 ) then
335:                     IP      = IP0 + IPP(I)
336:                     DTALY(IP) = DTALY(IP) + DNF(I)
337:                 end if
338:             240     continue
339:         else
340:             do 250 I = 1, NN
341:                 if ( IPP(I).ge.0 ) then
342:                     ITPP     = ITP(I) + ICYCLE
343:                     if ( ABS(TI(I)).ne.0.0 ) then
344: C 1999-Jul
345: C                     DT      = MIN(TI(I),DBLE(TIMEB(ITPP+1)-TIM(I)))
346: C                     &         - MAX(0.0d0,DBLE(TIMEB(ITPP))-TIM(I))
347: C
348: C                     DT      = MIN(TI(I),DBLE(TIMEB(ITPP+1))-TIM(I))
349: C                     &         - MAX(0.0d0,DBLE(TIMEB(ITPP))-TIM(I))
350: C                     DDD      = DNF(I)*DT/TI(I)
351: C                 else
352: C                     DDD      = 0.0
353: C                 end if
354: C
355:                 IP      = IP0 + IPP(I)
356:                 DTALY(IP) = DTALY(IP) + DDD
357:             end if
358:             250     continue
359:         end if
360: C
361: C
362:         if ( JTIME2.eq.0 ) then
363:             return
364:         else if ( JPTIM.eq.0 ) then
365:             do 260 I = 1, NN
366:                 if ( IPP(I).ge.0 ) then
367:                     ITPP     = ITP(I) + ICYCLE + 1
368:                     if ( ITPP.gt.NTIME
369: C                     &         .or. TIMEB(ITP(I)+ICYCLE+1)-TIM(I).ge.TI(I) ) then
370: C                         IPP(I) = -2
371: C                         IDO      = IDO - 1
372: C                     else
373: C                         IPP(I) = 0
374: C                     end if
375:                 end if
376:             260     continue
377:             ICYCLE = ICYCLE + 1
378:             if ( IDO.gt.0 ) then
379:                 go to 180
380:             else if ( IDO.lt.0 ) then
381:                 write(IOW,*)
382: C                 &         'XXX(STALN1) Program error?: IDO is negative !!!',
383: C                 &         ' IDO=',IDO
384: C                 &         write(IOW,*) ' Something wrong happened, STOP !!!'
385:                 stop 666
386:             end if
387:         else
388:             do 270 I = 1, NN
389:                 if ( IPP(I).ge.0 ) then
390:                     ITPP     = ITP(I) + ICYCLE

```

src/gmvp/staln1.f

```
391:          TII      = TI(I)
392:          TT        = TIM(I)+TII
393:          IPP(I)     = 0
394:          TT1       = TIMEB(1)
395:          if ( ITPP.le.NTIME ) TT1 = TIMEB(ITPP+1)
396:          if ( TT.le.TT1 ) then
397:             IPP(I)  = -2
398:             IDO      = IDO - 1
399:          else if ( ITPP.ge.NTIME ) then
400:             TI(I)    = TT - TCUT
401:             TIM(I)    = 0.0D0
402:             DNF(I)    = DNF(I) * TI(I) / TII
403:             ITP(I)    = -ICYCLE
404:             JPTIM2    = 1
405:          end if
406:        end if
407: 270      continue
408:          ICYCLE = ICYCLE + 1
409:          if ( IDO.gt.0 ) then
410:             go to 180
411:          else if ( IDO.lt.0 ) then
412:             write(IOW,*)
413:             &      'XXX(STALN1) Program error?: IDO is negative !!',
414:             &      ' IDO=',IDO
415:             write(IOW,*) ' Something wrong happened, STOP !!!'
416:             stop 666
417:          end if
418:        end if
419: C
420:      end if
421: C
422:      return
423:      end
```

src/gmvp/staln3.f

```

1:      subroutine STALN3( IOW,  ISURF, NTSRF,
2:      &                  JTIME, IDTALY, FLX,  NN,
3:      &                  IBP,  IBP0,  IEG,  IRG,  ITP,
4:      &                  NGROUP, NGP1,  NREG,  NRESP, NBANK,
5:      &                  NSTAL, NZONE,
6:      &                  NTIME, RESP,  NMKREG, MKREG, DTALY,
7:      &                  SGTAL, DBNK,  KDBNK,
8:      &                  MDBNK, IBNK,  KIBNK, MIBNK, DNF,  IPP,  IPMK
9:      &                  )
10: C=<GMVP>=====
11: C PURPOSE: Take special tallies for surface crossing tally
12: C          (Surface ISURF)
13: C
14: C (can treat neutron & photon but cannot treat both particles at once)
15: C
16: C CALLED IN: FLIONE
17: C-----
18: C
19: C arguments ( i = input, o=output, w=work )
20: C
21: C i IOW : message printout I/O unit
22: C i ISURF : tally surface #.
23: C i NTSRF : total number of tally surface.
24: C i JTIME : problem is time dependent if non zero.
25: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
26: C               routine.
27: C i FLX(NN) : current/flux or any other contribution to tally
28: C i NN      : number of particles
29: C i IBP(NN) : bank pointers for each particle
30: C i IBP0(NN) : pointers to point IEG, IRG.
31: C i IEG(*) : energy group #
32: C i IRG(*) : region #
33: C i ITP(NN) : time bin on surface crossing
34: C o DTALY(*) : tally
35: C
36: C w DNF(NN) : working area
37: C w IPP(NN) : working area
38: C w IPMK(NN) : working area
39: C-----
40: C
41: C Structure of IDTAL(*)
42: C
43: C +----- direct tally loop ( NDTALY )
44: C |
45: C | (1) ID
46: C | (2) (dummy for debug: -999)
47: C | (3) pointer to DTALY(*)
48: C | (4) event #
49: C | (5) particle type
50: C | (6) number of dimensions
51: C | (7) multiplication factor type
52: C | (8) nuclide/atom # & reaction #
53: C | (9) direct tally #
54: C | (10) custom tally # (non-zero means user customized treatment)
55: C | (11) detector # (for surface event positive tally-surface #
56: C |              means particle current tally, and negative one
57: C |              means flux tally)
58: C | ( IDTOFF data so far )
59: C |
60: C | +----- tally dimension loop
61: C | | (1) dimension type
62: C | | (2) number of tallied bins
63: C | | (3) size of common bin -> tallied bin conversion table
64: C | | ( IDMOFF data so far )
65: C | | +----- common bin loop

```

```

66: C | | | tallied bin # for each common bin
67: C | | +-----
68: C | +-----
69: C +-----
70: C=====
71: C          implicit real*8(A-H,O-Z)
72: C
73: C          integer IDTALY(*)
74: C
75: C          real*8 FLX(NN)
76: C
77: C          integer IBP(NN), IBP0(NN)
78: C          integer IEG(*), IRG(*)
79: C          integer ITP(NN)
80: C
81: C          real*8 DTALY(*)
82: C
83: C          real SGTAL(NBANK, NSTAL)
84: C
85: C          real*8 DBNK(NBANK, *)
86: C          integer KDBNK(0:MDBNK)
87: C          integer IBNK(NBANK, *)
88: C          integer KIBNK(0:MIBNK)
89: C
90: C          real RESP(NGROUP, NRESP)
91: C          integer MKREG(NREG, 2)
92: C
93: C          ... working array ...
94: C
95: C          integer IPP(NN)
96: C          integer IPMK(NN)
97: C          real*8 DNF(NN)
98: C
99: C          ... offset of direct-tally data in IDTAL(*)
100: C
101: CC          parameter( IDTOFF = 12 )
102: C
103: C          ... offset of tally dimension dependent data in IDTAL(*)
104: C
105: CC          parameter( IDMOFF = 3 )
106: C
107: C          include '../shared/INC/_IDXOFF'
108: C
109: C-----
110: C
111: C          II = 0
112: C
113: C          ... neutron tally ...
114: C
115: C          if ( IDTALY(5).eq.1 ) then
116: C              do 100 I = 1, NN
117: C                  if ( IEG(I).le.NGP1 ) then
118: C                      IPP(I) = 0
119: C                      II = II + 1
120: C                  else
121: C                      IPP(I) = -1
122: C                  end if
123: C              100 continue
124: C
125: C          ... photon tally ...
126: C
127: C          else if ( IDTALY(5).eq.2 ) then
128: C              do 112 I = 1, NN
129: C                  if ( IEG(I).gt.NGP1 ) then
130: C                      IPP(I) = 0

```

src/gmvp/staln3.f

```

131:          II      = II + 1
132:          else
133:            IPP(I) = -1
134:          end if
135: 112 continue
136: end if
137: C
138: if ( II.eq.0 ) return
139: C
140: C ... check marker region and marker region tally bin in IPMK(I) ...
141: C
142: if ( NMKREG.gt.0 ) then
143:   IDT = IDTOFF
144:   do 110 IB = 1, IDTALY(6)
145:     KBIN = IDTALY(IDT+1)
146: C
147: C ... this tally has marker region
148:   if ( KBIN.eq.8 ) then
149:     IDTM = IDT
150:     go to 120
151:   end if
152:   IDT = IDT + IDMOFF + IDTALY(IDT+3)
153: 110 continue
154: go to 160
155: 120 continue
156: C
157: C ... set non-marked as default
158: do 130 I = 1, NN
159:   IPMK(I) = 2
160: 130 continue
161: C
162: do 150 IM = 1, NMKREG
163:   KG = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
164: C
165: C ... this marker region is used in current tally ...
166: C
167:   if ( KG.eq.1 ) then
168:     KI9 = KIBNK(9) + IM - 1
169:     do 140 I = 1, NN
170:       if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I) = 1
171: 140 continue
172:   end if
173: 150 continue
174: C
175: 160 continue
176: end if
177: C
178: C ... calculate position in dtaly(*)
179: C
180: IDT = IDTOFF
181: do 180 IB = 1, IDTALY(6)
182:   KBIN = IDTALY(IDT+1)
183:   do 170 I = 1, NN
184:     if ( IPP(I).ge.0 ) then
185:       K = 0
186: C
187: C ... only 1 dtaly bin
188: C
189:   if ( IDTALY(IDT+3).eq.0 ) then
190:     K = 1
191: C
192: C .. marker region
193: C
194:   else if ( KBIN.eq.8 ) then
195:     K = IPMK(I)

```

```

196: C
197:   else
198: C
199: C .. region
200: C
201:     if ( KBIN.eq.1 ) then
202:       K = IRG(IBP0(I))
203: C
204: C .. energy
205: C
206:     else if ( KBIN.eq.2 ) then
207:       K = IEG(IBP0(I))
208: C
209:       ... for photon ...
210:       if ( IDTALY(5).eq.2 ) K = K - NGP1
211: C
212: C .. time
213: C
214:     else if ( KBIN.eq.3 ) then
215:       K = ITP(I)
216:       JTIME2 = 1
217: C
218: C .. angle
219: C
220:     else if ( KBIN.eq.4 ) then
221:       K = IBNK(IBP(I),KIBNK(10)+NTSRF+ISURF-1)
222: C
223: C .. generation
224: C
225:     else if ( KBIN.eq.5 ) then
226:       K = MIN(IBNK(IBP(I),KIBNK(4)),IDTALY(IDT+3))
227: C
228: C .. source set
229: C
230:     else if ( KBIN.eq.6 ) then
231:       K = IBNK(IBP(I),KIBNK(1))
232: C
233: C .. source region
234: C
235:     else if ( KBIN.eq.7 ) then
236:       K = IBNK(IBP(I),KIBNK(2))
237:     end if
238: C
239: C ... convert common bin to d-tally bin
240: C
241:     K = IDTALY(IDT+IDMOFF+K)
242:   end if
243: C
244: C ... IPP(I) = -1 : not in tally range !!!
245: C
246:   if ( K.eq.0 ) then
247:     IPP(I) = -1
248:   else
249:     IPP(I) = IPP(I)*IDTALY(IDT+2) + K - 1
250:   end if
251: end if
252: 170 continue
253: IDT = IDT + IDMOFF + IDTALY(IDT+3)
254: 180 continue
255: C
256: INUC = IDTALY(8) /1024
257: IMIC = IDTALY(8) - INUC*1024
258: C
259: C
260: C ... response ...

```

src/gmvp/staln3.f

```
261:      if ( IDTALY(7).lt.0 ) then
262:        do 190 I = 1, NN
263:          DNF(I) = FLX(I)*RESP(IEG(IBP0(I)),-IDTALY(7))
264:        190 continue
265: C
266: C      ... ???? ...
267: Ccccccc      else if( idtaly(7).lt.1000 ) then
268: Ccccccc      dnf(i) = dnf(i) *
269: C
270: C      ... smic ...
271: C      else if ( IDTALY(7).eq.1001 ) then
272: C        do 200 I = 1, NN
273: C          DNF(I) = FLX(I)*SMIC(IBP(I),INUC,LMIC(IMIC))*
274: C          &      DNZON(INUC,MZONE)
275: C        200 continue
276: C
277: C      ... smic ... (per atom)
278: C      else if ( IDTALY(7).eq.1002 ) then
279: C        do 210 I = 1, NN
280: C          DNF(I) = FLX(I)*SMIC(IBP(I),INUC,LMIC(IMIC))
281: C        210 continue
282: C
283: C      ... sgtal ...
284: C      else if ( IDTALY(7).eq.2000 ) then
285: C        do 220 I = 1, NN
286: C          DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
287: C        220 continue
288: C
289: C      ... smac ...
290: C      else if ( IDTALY(7).eq.3000 ) then
291: C        do 230 I = 1, NN
292: C          if ( IMB(IBP0(I)).gt.0 ) then
293: C            DNF(I) = FLX(I)*
294: C            &      SMAC(IBP(I),IMB(IBP0(I)),LMAC(IDTALY(8)))
295: C          else
296: C            DNF(I) = 0.0D0
297: C          end if
298: C        230 continue
299: C
300: C      ... simple current/flux ...
301: C      else
302: C        do 240 I = 1, NN
303: C          DNF(I) = FLX(I)
304: C        240 continue
305: C      end if
306: C
307: C
308: C      IP0 = IDTALY(3)
309: C      do 250 I = 1, NN
310: C        if ( IPP(I).ge.0 ) then
311: C          IP = IP0 + IPP(I)
312: C          DTALY(IP) = DTALY(IP) + DNF(I)
313: C        end if
314: C      250 continue
315: C
316: C      return
317: C      end
```

src/gmvp/staln7.f

```

1:      subroutine STALN7( IOW, IDTALY,
2:      I          FLX, IBP, NN,
3:      T          IEG, IRG, ITP,
4:      N          NGROUP,NGP1, NREG, NTIME, NBANK,
5:      R          DTALY,
6:      B          DBNK, KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
7:      W          DNF, IPP )
8: C=<MVP>=====
9: C PURPOSE: Take special tallies for noise analysis
10: C
11: C (can treat neutron & photon, but cannot treat both particles at once)
12: C
13: C CALLED IN: PANLGF, SEAONE
14: C=====
15: C
16: C arguments ( i = input, o=output, w=work )
17: C
18: C i IOW : message printout I/O unit
19: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
20: C routine
21: C i FLX(NN) : flux or contribution to tally
22: C i NN : NUMBER OF PARTICLES
23: C i IEG(NN) : GROUP #
24: C i IRG(NN) : REGION #
25: C i IBP(NN) : BANK POINTERS FOR EACH PARTICLE (NECESSARY ONLY WHEN
26: C JREAC=1 OR JEIGN=NE.0
27: C i ITP(NN) : current time bin of each particle
28: C o DTALY(*) : tally
29: C
30: C w DNF(NN) : WORK AREA
31: C w IPP(NN) : WORK AREA
32: C-----
33: C
34: C Structure of IDTAL(*)
35: C
36: C +----- direct tally loop ( NDTALY )
37: C |
38: C | (1) ID
39: C | (2) (dummy for debug: -999)
40: C | (3) pointer to DTALY(*)
41: C | (4) event #
42: C | (5) particle type
43: C | (6) number of dimensions
44: C | (7) multiplication factor type
45: C | (8) nuclide/atom # & reaction #
46: C | (9) direct tally #
47: C | (10) custom tally # (non-zero means user customized treatment)
48: C | (11) detector # (for surface event positive tally-surface #
49: C | means particle current tally, and negative one
50: C | means flux tally)
51: C | ( IDTOFF data so far )
52: C |
53: C | +----- tally dimension loop
54: C | | (1) dimension type
55: C | | (2) number of tallied bins
56: C | | (3) size of common bin -> tallied bin conversion table
57: C | | ( IDMOFF data so far )
58: C | | +----- common bin loop
59: C | | | tallied bin # for each common bin
60: C | | +-----
61: C | +-----
62: C +-----
63: C=====
64:      implicit real*8(A-H,O-Z)
65: C

```

```

66:      integer IDTALY(*)
67: C
68:      real*8 FLX(NN)
69:      integer IEG(NN), IBP(NN), IRG(NN), ITP(NN)
70: C
71:      real*8 DTALY(*)
72: C
73:      real*8 DBNK(NBANK,*)
74:      integer KDBNK(0:MDBNK)
75:      integer IBNK(NBANK,*)
76:      integer KIBNK(0:MIBNK)
77: C
78:      integer IPP(NN)
79:      real*8 DNF(NN)
80: C
81: C
82: C ... offset of direct-tally data in IDTAL(*)
83: C
84: CC      parameter( IDTOFF = 12 )
85: C
86: C ... offset of tally dimension dependent data in IDTAL(*)
87: C
88: CC      parameter( IDMOFF = 3 )
89: C
90:      include '../shared/INC/_IDXOFF'
91: C
92: C-----
93:      do 100 I = 1, NN
94:          IPP(I) = 0
95:      100 continue
96: C
97: C ... this flag shows that time dependent tally is really necessary
98: C
99: C ... calculate position in dtaly(*)
100: C
101:      IDT = IDTOFF
102:      NTT = 0
103:      do 130 IB = 1, IDTALY(6)
104:          KBIN = IDTALY(IDT+1)
105:          CCC      NTT = 0
106:          do 120 I = 1, NN
107:              if ( IPP(I).ge.0 ) then
108:                  K = 0
109: C
110: C ... only 1 dtaly bin
111: C
112: C if ( IDTALY(IDT+3).eq.0 ) then
113: C K = 1
114: C else
115: C
116: C .. region
117: C
118: C if ( KBIN.eq.1 ) then
119: C K = IRG(I)
120: C
121: C .. energy
122: C
123: C else if ( KBIN.eq.2 ) then
124: C K = IEG(I)
125: C ... for photon ...
126: C if ( IDTALY(5).eq.2 ) K = K - NGP1
127: C
128: C .. time
129: C
130: C else if ( KBIN.eq.3 ) then

```

src/gmvp/staln7.f

```
131:          K      = ITP(I)
132: C
133: Cccccccccc cc    else if( kbin.eq.4 ) then
134: Cccccccccc cc      k = ???
135: C
136: C
137: C      .. generation
138: C
139:          else if ( KBIN.eq.5 ) then
140:          K      = MIN( IBNK( IBP(I) ), KIBNK(4) ), IDTALY( IDT+3 ) )
141:          end if
142: C
143: C      ... convert common bin to d-tally bin
144: C
145:          K      = IDTALY( IDT+IDMOFF+K )
146:          end if
147: C
148: C      ... IPP(I) = -1 : not in tally range !!!
149: C
150:          if ( K.eq.0 ) then
151:          IPP(I) = -1
152:          else
153:          IPP(I) = IPP(I)*IDTALY( IDT+2 ) + K - 1
154:          NTT    = NTT + 1
155:          end if
156:          end if
157: 120      continue
158:          IDT    = IDT + IDMOFF + IDTALY( IDT+3 )
159: 130      continue
160: C
161: C
162:          if ( NTT.gt.0 ) then
163:          IP0     = IDTALY(3)
164:          do 140 I = 1, NN
165:          DNF(I) = FLX(I)
166:          if ( IPP(I).ge.0 ) then
167:          IP      = IP0 + IPP(I)
168:          DTALY(IP) = DTALY(IP) + DNF(I)
169:          end if
170: 140      continue
171:          end if
172: C
173:          return
174:          end
```


src/gmvp/staln9.f

```

1:      subroutine STALN9( IOW,  IDTALY, FLX  , IBP,   NN,
2:      T                    DTALY,
3:      C N                    IMPMAX, NGROUP, NGP1  , NREG  , NRESP , NSTAL ,
4:      N                    IMPMAX, NGROUP, NGP1  , NREG  , NRESP ,
5:      N                    NZONE,  NTIME,  NMKREG, MKREG,
6:      C T                    SG TAL,  RESP,
7:      T                    RESP,
8:      C X                    IMB,
9:      C N                    NUC,      NSMIC, NSMAC, MB,      NEMIC,
10:     C X                    DNZON, SMIC,  LMIC,  SMAC,  LMAC,
11:     B                    KDET P, IGGI,  ITTI,
12:     F                    DBNK,  KDBNK, MDENK, IBNK,  KIBNK, MIBNK,
13:     W                    DNF,    IPP,    IPMK )
14: C=<MVP>=====
15: C PURPOSE: Take special tallies (neutron/photon)
16: C
17: C (can treat neutron & photon, but cannot treat both particles at once)
18: C
19: C CALLED IN: NKTFLI
20: C UPDATE:
21: C=====
22: C
23: C arguments ( i = input, o=output, w=work )
24: C
25: C i IOW : message printout I/O unit
26: C i IDTALY(*) : part of IDTAL(*) for a direct tally calculated in this
27: C routine.
28: C i FLX(NN) : flux or contribution to tally
29: C i NN : NUMBER OF PARTICLES
30: C i IBP(NN) : BANK POINTERS FOR EACH PARTICLE
31: C i IMB(NN) : value of MMAC(IBP,1) for each particle
32: C o DTALY(*) : tally
33: C
34: C w DNF(NN) : working area
35: C w IPP(NN) : working area
36: C w IPMK(NN) : working area
37: C-----
38: C
39: C Structure of IDTAL(*)
40: C
41: C +----- direct tally loop ( NDTALY )
42: C |
43: C | (1) ID
44: C | (2) (dummy for debug: -999)
45: C | (3) pointer to DTALY(*)
46: C | (4) event #
47: C | (5) particle type
48: C | (6) number of dimensions
49: C | (7) multiplication factor type
50: C | (8) nuclide/atom # & reaction #
51: C | (9) direct tally #
52: C | (10) custom tally # (non-zero means user customized treatment)
53: C | (11) detector # (for surface event positive tally-surface #
54: C | means particle current tally, and negative one
55: C | means flux tally)
56: C | ( IDTOFF data so far )
57: C |
58: C | +----- tally dimension loop
59: C | | (1) dimension type
60: C | | (2) number of tallied bins
61: C | | (3) size of common bin -> tallied bin conversion table
62: C | | ( IDMOFF data so far )
63: C | | +----- common bin loop
64: C | | | tallied bin # for each common bin
65: C | | +-----

```

```

66: C | +-----
67: C | +-----
68: C=====
69:      implicit real*8(A-H,O-Z)
70: C
71:      integer IDTALY(*)
72: C
73:      real*8 FLX(NN)
74:      integer IBP(NN)
75: c      integer IMB(NN)
76: C
77:      real*8 DTALY(*)
78: C
79:      real RESP(NGROUP,NRESP)
80: c      real SG TAL(IMP MAX,NSTAL)
81: c      real DNZON(NUC,NZONE)
82: c      real SMIC(IMP MAX,NUC,NSMIC), SMAC(IMP MAX,MB,NSMAC)
83: c      integer LMAC(8), LMIC(8)
84: C
85: C ... bank ...
86:      integer KDET P(IMP MAX), IGGI(IMP MAX), ITTI(IMP MAX)
87:      real*8 DBNK(IMP MAX,*)
88:      integer KDBNK(0:MDBNK)
89:      integer IBNK(IMP MAX,*)
90:      integer KIBNK(0:MIBNK)
91: C
92:      integer MKREG(NREG,2)
93: C
94: C ... working array ...
95: C
96:      integer IPP(NN)
97:      integer IPMK(NN)
98:      real*8 DNF(NN)
99: C
100: C ... offset of direct-tally data in IDTAL(*)
101: C
102: CC      parameter( IDTOFF = 12 )
103: C
104: C ... offset of tally dimension dependent data in IDTAL(*)
105: C
106: CC      parameter( IDMOFF = 3 )
107: C
108:      include '../shared/INC/_IDXOFF'
109: C
110: C-----
111: C
112:      do 100 I = 1, NN
113:          IPP(I) = 0
114:      100 continue
115: C
116: C ... check marker region and marker region tally bin in IPMK(I) ...
117: C
118:      if ( NMKREG.gt.0 ) then
119:          IDT = IDTOFF
120:          do 110 IB = 1, IDTALY(6)
121:              KBIN = IDTALY(IDT+1)
122: C
123: C ... this tally has marker region
124:          if ( KBIN.eq.8 ) then
125:              IDTM = IDT
126:              go to 120
127:          end if
128:          IDT = IDT + IDMOFF + IDTALY(IDT+3)
129:      110 continue
130:      go to 160

```

src/gmvp/staln9.f

```

131: 120    continue
132: C
133: C      ... set non-marked as default
134:       do 130 I = 1, NN
135:         IPMK(I) = 2
136: 130    continue
137: C
138:       do 150 IM = 1, NMKREG
139:         KG      = IDTALY(IDTM+IDMOFF+MKREG(IM,2))
140: C
141: C      ... this marker region is used in current tally ...
142: C
143:       if ( KG.eq.1 ) then
144:         KI9      = KIBNK(9) + IM - 1
145:         do 140 I = 1, NN
146:           if ( IBNK(IBP(I),KI9).gt.0 ) IPMK(I)      = 1
147: 140        continue
148:       check %%%
149:       write(*,*) ' IM ',IM,' IPMK ',(IPMK(I),i=1,NN)
150:       c %%%%%%%%%
151:       end if
152: 150    continue
153: C
154: 160    continue
155: end if
156: C
157: C      ... calculate position in dtaly(*)
158: C
159:       IDT      = IDTOFF
160:       do 190 IB = 1, IDTALY(6)
161:         KBIN      = IDTALY(IDT+1)
162:         NTT      = 0
163:         do 180 I = 1, NN
164:           if ( IPP(I).ge.0 ) then
165: C
166: C      ... only 1 dtaly bin
167: C
168:           if ( IDTALY(IDT+3).eq.0 ) then
169:             K      = 1
170: C
171: C      .. marker region
172: C
173:           else if ( KBIN.eq.8 ) then
174:             K      = IPMK(I)
175: C
176:           else
177: C
178: C      .. region : no region for point detector
179: C
180:           if ( KBIN.eq.1 ) then
181:             K      = IRG(I)
182: C
183: C      .. energy
184: C
185:           if ( KBIN.eq.2 ) then
186:             K      = IGGI(IBP(I))
187: C      ... for photon ...
188:           if ( IDTALY(5).eq.2 ) K = K - NGP1
189: C
190: C      .. time
191: C
192:           else if ( KBIN.eq.3 ) then
193:             K      = ITTI(IBP(I))
194: C
195: Ccccccccccccccc cc      else if( kbin.eq.4 ) then

```

```

196: Ccccccccccccccc cc      k = ???
197: C
198: C
199: C      .. generation
200: C
201:           else if ( KBIN.eq.5 ) then
202:             K      =
203:             &      MIN(IBNK(IBP(I),KIBNK(4)),IDTALY(IDT+3))
204: C
205: C      .. source set
206: C
207:           else if ( KBIN.eq.6 ) then
208:             K      = IBNK(IBP(I),KIBNK(1))
209: C
210: C      .. source region
211: C
212:           else if ( KBIN.eq.7 ) then
213:             K      = IBNK(IBP(I),KIBNK(2))
214: C
215: C      .. spatial point
216: C
217:           else if ( KBIN.eq.9 ) then
218:             K      = KDETP(IBP(I))
219:           end if
220: C
221: C      ... convert common bin to d-tally bin
222: C
223:             K      = IDTALY(IDT+IDMOFF+K)
224:           end if
225: C
226: C      ... IPP(I) = -1 : not in tally range !!!
227: C
228:           if ( K.eq.0 ) then
229:             IPP(I) = -1
230:           else
231:             IPP(I) = IPP(I)*IDTALY(IDT+2) + K - 1
232:             NTT      = NTT + 1
233:           end if
234:           end if
235: 180    continue
236:       IDT      = IDT + IDMOFF + IDTALY(IDT+3)
237: 190 continue
238: C
239:       INUC      = IDTALY(8) /1024
240:       IMIC      = IDTALY(8) - INUC*1024
241: C
242: C      ... response ...
243:       if ( IDTALY(7).lt.0 ) then
244:         do 200 I = 1, NN
245:           DNF(I) = FLX(I)*RESP(IGGI(IBP(I)),-IDTALY(7))
246: 200    continue
247: C
248: C      ... ??? ...
249: C      else if( idtaly(7).lt.1000 ) then
250: C      dnf(i) = dnf(i) *
251: C
252: C      ... smic ...
253:       else if ( IDTALY(7).eq.1001 ) then
254:         do 210 I = 1, NN
255:           DNF(I) = FLX(I)*SMIC(IBP(I),INUC,LMIC(IMIC))*
256:           &      DNZON(INUC,MZONE)
257: 210    continue
258: C
259: C      ... smic ... (per atom)
260:       else if ( IDTALY(7).eq.1002 ) then

```

src/gmvp/staln9.f

```
261: c      do 220 I = 1, NN
262: c          DNF(I) = FLX(I)*SMIC(IBP(I),INUC,LMIC(IMIC))
263: c 220      continue
264: C
265: C      ... sgtal ...
266: c      else if ( IDTALY(7).eq.2000 ) then
267: c          do 230 I = 1, NN
268: c              DNF(I) = FLX(I)*SGTAL(IBP(I),IDTALY(8))
269: c 230      continue
270: C
271: C      ... smac ...
272: c      else if ( IDTALY(7).eq.3000 ) then
273: c          do 240 I = 1, NN
274: c              if ( IMB(I).gt.0 ) then
275: c                  DNF(I) = FLX(I)*
276: c                  & SMAC(IBP(I),IMB(I),LMAC(IDTALY(8)))
277: c              else
278: c                  DNF(I) = 0.0D0
279: c              end if
280: c 240      continue
281: C
282: C      ... flux ...
283: c      else
284: c          do 250 I = 1, NN
285: c              DNF(I) = FLX(I)
286: c 250      continue
287: c      end if
288: C
289: C      --- weight by time-bin occupation rate ---
290: C
291: c      IP0      = IDTALY(3)
292: c      do 260 I = 1, NN
293: c          if ( IPP(I).ge.0 ) then
294: c              IP      = IP0 + IPP(I)
295: c              DTALY(IP) = DTALY(IP) + DNF(I)
296: c          end if
297: c 260 continue
298: C
299: c      return
300: c      end
```

src/gmvp/store.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine STORE( IE,   IC,   SIGT,  NSIG )
3: C=====
4: C Purpose: PICKS UP CROSS SECTIONS FOR ELEMENT IE,COEF IC,AND *
5: C          STORES THEM IN TEMPORARY STORAGE
6: C IT STRIPS OFF ANY UNUSED PARTS OF THE INPUT * E.G., GAMMAS FROM A
7: C COUPLED SET WHEN ONLY NEUTRONS WILL BE USED * * * *
8: C
9: C=====
10:      common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
11: & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
12: & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
13: & NDS,   NGG,   NDSG,   INGP,   INDS,   NMED,   NELEM,
14: & NMIX,   NCOEF, NSCT,   NTS,   NTG,   NDSNGP, NDSNGG,
15: & IADJ,   NME,   LOC,   INGS,   INSG,   I1,   I0,
16: & KKK,   IXTAPE, IDEL,   ITEML, ITEMG, IRSG,   IRDSG,
17: & ISTR,   IPRIN, IEMU,   IMOM,   IDTF,   ISTAT, IPUN,
18: & NUS,   NGN,   IHT,   INUS,   INUSN, INGN,   INGNP,
19: & INNN,   IGGG,
20:      real SIGT(1)
21:      integer NSIG(1)
22: C      EQUIVALENCE(SIGT(1),NSIG(1))
23: C
24: C      INN      = INPBUF
25: C      IE1      = ITEML*(IE-1)
26: C      IF       = INFPOG + NTG*(IE-1)
27: C      ID       = ISIGOG + NTG*(IE-1)
28: C      NGP3     = INDS + IHT + NUS
29: C      NTGMT    = NTG + NME - 1
30: C
31: C FORWARD PROBLEM
32: C
33: C      if ( IADJ.le.0 ) then
34: C      if ( IC.le.1 ) then
35: C          J1      = ID - NME + 1
36: C          do 100 I = NME, NTGMT
37: C              K      = INPBUF + (I-1)*NGP3
38: C
39: C      STORE TOTAL FOR FIRST COEFF      ALL GROUPS
40: C
41: C          SIGT(I+J1) = SIGT(K+IHT)
42: C      100      continue
43: C      if ( NME-1.le.0 ) then
44: C          do 110 I = 1, NGP
45: C              K      = INPBUF + (I-1)*NGP3
46: C
47: C      STORE NU*FISS FOR FIRST COEFF      NEUTRON GROUPS
48: C
49: C          SIGT(I+IF) = SIGT(K+IHT-1)
50: C      110      continue
51: C      end if
52: C
53: C PO MATRIX
54: C
55: C      do 130 I = NME, NTGMT
56: C          NDSK      = I + NUS - NME + 1
57: C          if ( NDSK-NUS-NDS-NDSG.gt.0 ) then
58: C              NDSK      = NUS + NDS + NDSG
59: C          end if
60: C          IB      = ISPORT + (IE-1)*NTG*NTS
61: C          do 120 J = 1, NDSK
62: C              IA      = I - NME + 1 - J + NUS
63: C              if ( IA-NTG+1.le.0 ) then
64: C                  IJ      = IB + IA*NTS + J
65: C                  K      = INPBUF + (I-1)*NGP3 + J + IHT

```

```

66:      SIGT(IJ)      = SIGT(K)
67:      end if
68:      120      continue
69:      130      continue
70:      return
71:      end if
72: C
73: C      STORE PL COEFF
74: C
75: C      if ( NME-1.eq.0 ) then
76: C          do 150 I = 1, NGP
77: C              NDSK      = NSIG(INNN+I)
78: C              K      = INPBUF + (I-1-NUS)*NGP3 + IHT
79: C              IJ      = IABSOG + NSIG(I+INGS) + IE1 + (IC-2)*NDSNGP
80: C              do 140 J = 1, NDSK
81: C                  K      = K + 1
82: C                  if ( NUS+2-I-J.le.0 ) then
83: C                      SIGT(IJ+J) = SIGT(K)
84: C                  end if
85: C                  K      = K + NGP3
86: C              140      continue
87: C          150      continue
88: C          if ( NGG.gt.0 ) then
89: C              do 170 I = 1, NGG
90: C                  NDSK      = NSIG(IGGG+NGG-I+1)
91: C                  do 160 J = 1, NDSK
92: C                      K      = INPBUF + (I-1+NGP)*NGP3 + J + IHT
93: C                      IAB      = I - J + 1 + NUS
94: C                      if ( IAB-NGG.le.0 ) then
95: C                          IJ      = ITEMG + NSIG(IAB+INSNG) + IE1 + J
96: C                          &      + (IC-2)*NDSNGG
97: C                      SIGT(IJ)      = SIGT(K)
98: C                      end if
99: C              160      continue
100: C          170      continue
101: C          end if
102: C          return
103: C
104: C      GAMMA STRIP
105: C      else
106: C          do 190 I = NME, NTGMT
107: C              NDSK      = NSIG(INNN+NTGMT-I+1)
108: C              do 180 J = 1, NDSK
109: C                  IAB      = I - J - NME + 2 + NUS
110: C                  if ( IAB-NTG.le.0 ) then
111: C                      K      = INPBUF + (I-1)*NGP3 + J + IHT
112: C                      IJ      = IABSOG + NSIG(IAB+INGS) + IE1 + J +
113: C                          &      (IC-2)*NDSNGP
114: C                      SIGT(IJ)      = SIGT(K)
115: C                  end if
116: C              180      continue
117: C          190      continue
118: C          return
119: C      end if
120: C
121: C ADJOINT PROBLEM
122: C
123: C      else
124: C          if ( IC-1.le.0 ) then
125: C
126: C      PO MATRIX
127: C
128: C          do 200 I = NME, NTGMT
129: C              K      = INPBUF + (I-1)*NGP3
130: C              L      = NTGMT - I + 1

```

src/gmvp/store.f

```

131:          SIGT(L+ID) = SIGT(K+IHT)
132: 200      continue
133: C
134: C      IF NEUTRON ONLY, STORE NUSIGF
135: C
136:          if ( NTG-NGP.le.0 ) then
137:          if ( NME-1.le.0 ) then
138:              do 210 I = 1, NGP
139:                  K = INPBUF + (I-1)*NGP3
140:                  L = NGP - I + 1
141:                  SIGT(L+IF) = SIGT(K+IHT-1)
142: 210          continue
143:          end if
144: C
145: C      IF COUPLED, STORE NUSIGF
146: C
147:          else
148:              NGP1 = NGP + 1
149:              do 220 J = NGP1, NTG
150:                  K = INPBUF + (J-NGP1)*NGP3
151:                  L = NTG - (J-NGP1)
152:                  SIGT(L+IF) = SIGT(K+IHT-1)
153: 220          continue
154:          end if
155: C
156:          do 240 I = NME, NTGMT
157:              L = NTGMT - I + 1
158:              NDSK = I + NUS - NME + 1
159:              if ( NDSK-NUS-NDS-NDSG.gt.0 ) then
160:                  NDSK = NUS + NDS + NDSG
161:              end if
162: C
163: C      STORE PO MATRIX
164: C
165:          do 230 J = 1, NDSK
166:              K = INPBUF + (I-1)*NGP3 + J + IHT
167:              IJ = ISPORT + (IE-1)*NTG*NTS + (L-1)*NTS + J
168: C
169: C              if ( ISIGOG-IJ.gt.0 ) then
170: C7000          write(I0,7000) IJ, J, I, K
171: C              format(' ',4I5)
172: C              end if
173: C              SIGT(IJ) = SIGT(K)
174: 230          continue
175: 240          continue
176:          return
177: C
178: C      STORE PL COEFF
179: C
180:          else
181:          if ( NME-1.eq.0 ) then
182: C      STORE INPUT NEUTRONS AS GAMMAS
183: C
184:          do 260 I = 1, NGP
185:              NDSK = NSIG(INNN+I)
186:              IJ = IABSOG + NSIG(INGS+I) + IE1 + (IC-2)*NDSNGP
187:              K = INPBUF + (NTG-I)*NGP3 + IHT
188:              do 250 J = 1, NDSK
189:                  SIGT(IJ+J) = SIGT(K+J)
190: 250              continue
191: 260              continue
192: C
193: C      STORE INPUT GAMMAS AS NEUTRONS
194: C
195:          if ( NGG.gt.0 ) then

```

```

196:          NGP1 = NGP + 1
197:          do 280 I = NGP1, NTG
198:              NDSK = NSIG(INNN+I)
199:              IJ = ITEMG + NSIG(INGSG+I-NGP) + IE1 + (IC-2)*
200:              &      NDSNGG
201:              K = INPBUF + (NTG-I)*NGP3 + IHT
202:              do 270 J = 1, NDSK
203:                  SIGT(IJ+J) = SIGT(K+J)
204: 270              continue
205: 280              continue
206:          end if
207: C
208: C      GAMMA STRIP
209: C
210:          else
211:          do 300 I = NME, NTGMT
212:              L = NTGMT - I + 1
213:              NDSK = NSIG(INNN+NTGMT-I+1)
214:              IJ = IABSOG + NSIG(INGS+L) + IE1 + (IC-2)*NDSNGP
215:              do 290 J = 1, NDSK
216:                  K = INPBUF + (I-1)*NGP3 + J + IHT
217:                  SIGT(IJ+J) = SIGT(K)
218: 290              continue
219: 300              continue
220:          end if
221:          end if
222:          end if
223:          return
224:          end

```

src/gmvp/suncl.f

```

1:      subroutine SUNCL( IOW, A, H, NSORS, LSORS, KSRC, PSPAC,
2:      N      DUMAX, DUMIN, NPDET, NPLEN, NIMPT, NDIMPT,
3:      G      NDBNK, NIBNK, TIMEB, KZMAT, STOTX, VEL ,
4:      D      JPUSD, JSPDT, XPDET, IPDET, IPDT2,
5:      S      IMSFL, IMNFL, IMZFL,
6:      S      IMDED, IMSLT, IMNLT, IMZLT ,
7:      B      XXX , YYY , ZZZ , AAA , BBB , CCC ,
8:      B      WWW , IZZ , IGG , TTT , ITT ,
9:      B      LEVL , LZZ , LPOS , LCRS ,
10:     B      DBNK , KDBNK, MDBNK, IBNK , KIBNK, MIBNK,
11:     B      LZZI , LPOSI, LCRSI,
12:     F      OPTI , PATH, KDETP, KLSFI,
13:     W      X , Y , Z , A0 , B0 , C0 , AA , BB , CC , DI ,
14:     W      DW1, DW2, DW3, DW4, DW5, TI ,
15:     W      IGI, IRP, IT0, W )
16: C=====
17: C PURPOSE:  Calculation of uncollided contribution to next event
18: C            detectors.
19: C            <COMMENT>
20: C            Some contributions may not be calculated in
21: C            this routine, because the 'NXTEE' routine, that controls
22: C            tracking of imaginary test-particles, is called with
23: C            MODE=2 (cease processing when the number of particles
24: C            drops below a threshold value), and 'NXTEE' is not
25: C            called until the bank for imaginary test-particles is
26: C            full.
27: C
28: C CALLED IN: SOURCE
29: C CALLS:      NXTEE, LATUP2, LATDW2
30: C=====
31: C      NSORS : NUMBER OF SOURCE PARTICLES WHOSE UNCLLIDED CONTRIBUTIONS
32: C              ARE CALCULATED.
33: C      LSORS : BANK INDEX OF SOURCE  PARTICLES.
34: C      KSRC  : TYPE OF SOURCE
35: C              KSRC = 0 : UNIFORM SOURCE.
36: C              > 0 : OTHER NON UNIFORM SOURCE.
37: C=====
38: CCC      implicit real*8 (D)
39:      implicit real*8 (A-H,O-Z)
40: C
41:      real A(*)
42:      real H(*)
43: C
44: C**** BANK INDEX FOR SOURCE PARTICLES. *****
45: C
46:      integer LSORS(NSORS)
47: C
48: C**** SOURCE DATA *****
49: C
50:      real      PSPAC(10)
51: C
52: C**** DATA FOR NEXT EVENT (POINT) ESTIMATOR *****
53: C
54:      real*8  XPDET(NPLEN,NPDET)
55:      integer IPDET(NPDET,2), IPDT2(NEST,3,NPDET)
56:      integer JPUSD(NPDET), JSPDT(NPDET)
57: C
58: C**** TALLY BIN DATA & VELOCITY *****
59: C
60:      real      TIMEB(*), VEL(NGROUP)
61: C
62: C**** STACK FOR IMAGINARY PARTICLE *****
63: C
64:      integer IMSFL(IMPMAX), IMZFL(IMPMAX), IMNFL(NZONE+1),
65:      &      IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)

```

```

66: C
67: C**** PARTICLE BANK 1:REAL PARTICLE, 2:IMAGINARY PARTICLE *****
68: C
69:      real*8  XXX(NBANK,2), YYY(NBANK,2), ZZZ(NBANK,2)
70:      real*8  AAA(NBANK,2), BBB(NBANK,2), CCC(NBANK,2)
71: C
72:      real      WWW(NBANK,2)
73:      real*8  TTT(NBANK,2)
74: C
75:      integer IZZ(NBANK,2), IGG(NBANK,2), ITT(NBANK,2)
76:      integer LEVL(NBANK,2), LZZ(NBANK,NEST),
77:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), LZZI(IMPMAX,NEST),
78:      &      LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST)
79: C
80:      real*8  DBNK(NBANK,NDBNK,2)
81:      integer KDBNK(0:MDBNK)
82:      integer IBNK(NBANK,NIBNK,2)
83:      integer KIBNK(0:MIBNK)
84: C
85: C****
86: C
87:      real*8  OPTI(IMPMAX), PATH(IMPMAX)
88:      integer KDETP(IMPMAX), KLSFI(IMPMAX)
89: C
90: C**** total cross section and geometry data
91: C
92:      real      STOTX(NTGX,*)
93:      integer  KZMAT(NZONE)
94: C
95: C**** WORKING AREA *****
96: C
97:      integer IBP(NBANK), IT0(NBANK), IGI(NBANK)
98:      real*8  X(NBANK), Y(NBANK), Z(NBANK)
99:      real*8  AA(NBANK), BB(NBANK), CC(NBANK), DI(NBANK)
100:     real*8  A0(NBANK), B0(NBANK), C0(NBANK)
101:     real*8  DW1(NBANK), DW2(NBANK), DW3(NBANK), DW4(NBANK),
102:     &      DW5(NBANK), TI(NBANK)
103:     real      W(NBANK)
104: C****
105:     include ' ../shared/INC/_SIZES'
106:     include ' ../shared/INC/_PGEOM'
107:     include ' INC/_SIZES2'
108:     include ' INC/_FLAGS'
109: CCCC include ' INC/_WKNXU'
110:     include ' INC/_PXSEC'
111: C
112:     DPI2   = 2.0D0*ACOS(-1.0D0)
113:     DPI4I  = 1.0D0/(4*ACOS(-1.0D0))
114: C=====
115: C
116: C**** GATHER POSITION,ENERGY GROUP ETC *****
117: C
118: check
119: c      write(6,*) ' Nothing is done in SUNCL'
120: c      return
121: check
122:     if ( JLATT.eq.0.or.JEIGN.eq.0.or.(JEIGN.ne.0.and.NBATCH.le.0) )
123:     &      then
124: C
125: C ..... NO LATTICE
126: C
127:     do 100 I = 1, NSORS
128:         X(I)   = XXX(LSORS(I),1)
129:         Y(I)   = YYY(LSORS(I),1)
130:         Z(I)   = ZZZ(LSORS(I),1)

```

src/gmvp/suncl.f

```

131:      A0(I)      = AAA(LSORS(I),1)
132:      B0(I)      = BBB(LSORS(I),1)
133:      C0(I)      = CCC(LSORS(I),1)
134: 100      continue
135: C
136:      else
137: C
138: C ..... LATTICE
139: C ***** TRANSFORMATION OF CORDINATES AND DIRECTION
140: C FROM CELL-ORDINATE SYSTEM TO LABORATORY SYSTEM
141: C (XXX,YYY,ZZZ) --> (X,Y,Z)
142: C
143:      JDIR = 1
144:      JLS = 0
145:      call LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
146: & NZONE, NSORS, NBANK, JDIR, JLS, LSORS,
147: & X, Y, Z, A0, B0, C0,
148: & XXX, YYY, ZZZ,
149: & AAA, BBB, CCC,
150: & LEVL, LZZ, LEOS, LCRS,
151: & DBNK, KDBNK, MDBNK,
152: & A(LKZMAT), A(LKDALT), A(LDALT), A(LNVLAT), A(LSLZLAT),
153: & A(LCELSZ), A(LIPLAT), A(LKLAT), A(LKSLAT), A(LLTYP),
154: & A(LICTYP), A(LMLBZZ),
155: & IBP )
156: C
157:      end if
158: C
159: C
160: C**** LOOP OVER DETECTORS *****
161: C
162:      do 200 ND = 1, NPDET
163: C
164:      if ( JPUSD(ND).eq.0 ) go to 200
165: C
166: C**** OBTAIN THE DISTANCE TO THE DETECTOR
167:      DXDA = XPDET(3,ND)
168:      DYDA = XPDET(4,ND)
169:      DZDA = XPDET(5,ND)
170:      RMIN = XPDET(1,ND)
171:      RMIN2 = RMIN*RMIN
172:      RMAX = XPDET(2,ND)
173:      RMA2 = RMAX*RMAX
174: C
175:      IZD = IPDET(ND,1)
176:      MAT = KZMAT(IZD)
177:      call RANU2( IRAND, W, NSORS, ICON )
178: C
179: *VOCL LOOP, NOVREC
180:      do 210 I = 1, NSORS
181:      DAT = DXDA - X(I)
182:      DBT = DYDA - Y(I)
183:      DCT = DZDA - Z(I)
184:      DDI = DAT*DAT + DBT*DBT + DCT*DCT
185: C
186:      R = W(I)
187: C
188: C Calculate contriution with probability RMIN2/DDI
189: C when DDI > RMIN**2
190: C
191:      if( DDI*R.gt.RMIN2 ) then
192:      W(I) = 0.0
193:      elseif ( DDI.gt.RMIN2 ) then
194:      W(I) = WWW(LSORS(I),1) / RMIN2 * DPI4I
195: C

```

```

196: C      Bounded sphere approximation when DDI < RMAX**2
197: C
198:         elseif ( DDI.lt.RMA2.and.MAT.gt.0 ) then
199: CCCC      elseif ( DDI.lt.RMA2 ) then
200: CCCCC      IZ      = IZZ(LSORS(I),1)
201: CCCCC      RMD      = STOTX(IGI(I),KZMAT(IZ)) * RMAX
202:           IGIIII     = IGG(LSORS(I),1)
203:           RMD      = STOTX(IGIIII,MAT) * RMAX
204:           RMI      = 1. / RMD
205:           C2      = RMA2*(RMI*RMI*2.+(1.+2.*RMI)/(1.-EXP(RMD)))
206:           W(I)     = WWW(LSORS(I),1) / C2 * DPI4I
207:         else
208: C
209: C      WEIGHT / (DISTANCE)**2 / (4*PAI)
210: C
211:           W(I)      = WWW(LSORS(I),1) / DDI * DPI4I
212:         endif
213: C
214:           DI(I)     = SQRT(DDI)
215: C
216:           if ( DI(I).gt.0. ) then
217:             AA(I)    = -DAT/DI(I)
218:             BB(I)    = -DBT/DI(I)
219:             CC(I)    = -DCT/DI(I)
220:           else
221:             AA(I)    = -A0(I)
222:             BB(I)    = -B0(I)
223:             CC(I)    = -C0(I)
224:           end if
225: 210      continue
226: C
227: C      ..... APPLY ANGULAR DISTRIBUTION FUNCTION TO WEIGHT WW .....
228: C
229:           KSRC = JSPDT(ND)
230: C
231: C      *** KSRC = 0 : ISOTROPIC      FACTOR = 1.0 OR 2./(DUMAX-DUMIN)
232: C      *** KSRC > 0 : NON ISOTROPIC
233: C      *** KSRC < 0 : NO CONTRIBUTION
234: C
235:           if ( KSRC.eq.0 ) then
236:             DFACT    = 2./(DUMAX-DUMIN)
237: C
238:             do 220 I = 1, NSORS
239:               if ( -CC(I).le.DUMAX.and.-CC(I).ge.DUMIN ) then
240:                 W(I)  = W(I)*DFACT
241:               else
242:                 W(I)  = 0.0
243:               end if
244:             220      continue
245: C
246: C      CALCULATE FACTOR BY USING AA,BB,CC
247: C      else if ( KSRC.gt.0 ) then
248:           DELMU    = 1.D-1
249:           CONT     = 1.D0 - DELMU / DPI2
250:           DFACT    = 2.D0 * DPI2 / DELMU
251: C
252: C      do 230 I = 1, NSORS
253:           XMU      = A0(I)*AA(I)+B0(I)*BB(I)+C0(I)*CC(I)
254:           XMU      = -XMU
255:           if ( XMU.ge.CONT ) then
256:             W(I)    = W(I)*DFACT
257:           else
258:             W(I)    = 0.0
259:           end if
260: C

```

src/gmvp/suncl.f

```

261: 230      continue
262: C
263: C
264:      else
265:          do 240 I = 1, NSORS
266:              W(I) = 0.0
267:          240      continue
268:      end if
269: C
270: C      .... TIME CUT OFF .....
271: C
272:      if ( JTIME.ne.0 ) then
273:          do 250 I = 1, NSORS
274:              TI(I) = TTT(LSORS(I),1) + DI(I) / VEL(IGG(LSORS(I),1))
275:          250      continue
276:          if ( JPTIM.eq.0 ) then
277:              do 260 I = 1, NSORS
278:                  if ( TI(I).gt.TCUT ) then
279:                      W(I) = 0.0
280:                      TI(I) = TCUT
281:                  end if
282:              260      continue
283:          else
284:              do 270 I = 1, NSORS
285:                  if ( TI(I).ge.TCUT ) then
286:                      ITREP = TI(I) / TCUT
287:                      TI(I) = TI(I) - TCUT*ITREP
288:                  end if
289:              270      continue
290:          end if
291: C
292:      call BS0ISD( TIMEB, NTIME+1, TI, IT0, NSORS )
293: C
294:      do 280 I = 1, NSORS
295:          IT0(I) = MAX(1,MIN(NTIME,IT0(I)))
296:      280      continue
297:      end if
298: C
299: C
300: C **** SEND PARTICLES TO FREE-FLIGHT OR LATTICE STACK
301: C
302: C      FOR IMAGINARY PARTICLES ****
303: C
304: C
305: C      ..... CHECK NUMBER OF PARTICLES HAVING NON-ZERO ANGULAR FACTOR .....
306: C
307: C      IBP : BANK INDEX
308: C
309:      NN = 0
310: *VOCL LOOP,NOVREC
311:      do 290 I = 1, NSORS
312:          if ( W(I).ne.0.0 ) then
313:              NN = NN + 1
314:              IBP(NN) = LSORS(I)
315:              AA(NN) = AA(I)
316:              BB(NN) = BB(I)
317:              CC(NN) = CC(I)
318:              W (NN) = W (I)
319:              if ( JTIME.ne.0 ) IT0(NN) = IT0(I)
320:          end if
321:      290      continue
322: C
323: C
324: C=====
325: C

```

```

326: C      SEND PARTICLES TO STACK FOR IMAGINARY PARTICLES
327: C
328: C=====
329: C
330: C
331:      IZD = IPDET(ND,1)
332:      LVL = IPDET(ND,2)
333:      if ( JLATT.ne.0.and.LVL.gt.0 ) then
334:          if ( IPDT2(LVL,2,ND).lt.0 ) LVL = LVL - 1
335:          LP = 2 + 3*LVL
336:          DXDA = XPDET(LP+1,ND)
337:          DYDA = XPDET(LP+2,ND)
338:          DZDA = XPDET(LP+3,ND)
339:      end if
340: C
341: 300      continue
342: C
343:      if ( NN.gt.0 ) then
344: C
345:          NNN = MIN(NN,NDIMPT)
346: C
347:          if ( NNN.gt.0 ) then
348:              NDIMPT = NDIMPT - NNN
349: C
350: C **** SEND IMAGINARY PARTICLES TO FREE-FLIGHT STACK
351: C
352:          N = IMNFL(NZONE+1)
353:          N1 = NN - NNN
354: *VOCL LOOP,NOVREC
355:          do 310 J = 1, NNN
356:              J1 = N1 + J
357:              IMSFL(N+J) = IMDED(NDIMPT+J)
358:              IMZFL(N+J) = IZD
359:              XXX(IMSFL(N+J),2) = DXDA
360:              YYY(IMSFL(N+J),2) = DYDA
361:              ZZZ(IMSFL(N+J),2) = DZDA
362:              AAA(IMSFL(N+J),2) = AA(J1)
363:              BBB(IMSFL(N+J),2) = BB(J1)
364:              CCC(IMSFL(N+J),2) = CC(J1)
365:              WWW(IMSFL(N+J),2) = W(J1)
366:              IGG(IMSFL(N+J),2) = IGG(IBP(J1),1)
367:              TTT(IMSFL(N+J),2) = TTT(IBP(J1),1)
368:              PATH(IMSFL(N+J)) = DI(J1)
369:              OPTI(IMSFL(N+J)) = 0.0
370:              KDETP(IMSFL(N+J)) = ND
371:              KLSFI(IMSFL(N+J)) = 0
372:              if ( JTIME.ne.0 ) then
373:                  ITT(IMSFL(N+J),2) = IT0(J1)
374:              end if
375:              if ( KIBNK(5).ne.0 ) then
376:                  IBNK(IMSFL(N+J),KIBNK(5),2) = -1
377:              end if
378:          310      continue
379: C
380: C
381:          NN2 = NNN
382:          JSTG = 0
383: C
384: C=====
385: C
386: C      In LATTICE geometry, the following parameters should be set.
387: C      LEVL, LZZI, LPOSI, LCRSI
388: C      DBNK, IBNK (JFISX.ne.0)
389: C      JSTG : flag to indicate the detector is located in STG region.
390: C      NN2 : number of remaining particles when JFISX.ne.0.

```


src/gmvp/suncl.f

```

391: C
392: C=====
393: C
394:       if ( JLATT.ne.0 ) then
395:       call LATDW2(IOW, JDEBG, JHLAT, JFISX,
396:         N      IMPMAX,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
397:         N      DINF ,DEPS ,
398:         S      IMSFL(N+1),NNN ,JSTG ,NN2 ,
399:         I      IPDET(ND,2),IPDT2(1,1,ND),IPDT2(1,2,ND),
400:         I      IPDT2(1,3,ND), XPDET(3,ND),
401:         B      AAA(1,2),BBB(1,2),CCC(1,2) ,
402:         F      LEVL(1,2),LZZI ,LPOSI ,LCRSI ,
403:         B      DBNK(1,1,2),KDBNK ,MDBNK ,IBNK(1,1,2),KIBNK ,MIBNK,
404:         S      IMDED ,NDIMPT,
405:         G      A(LSDA) ,A(LKZDA) ,A(LKZAA) ,
406:         G      A(LKZMAT),A(LKCELL),A(LKZLBZ),A(LMLBZZ),A(LCELSZ),
407:         G      A(LSZLAT),A(LIDLAT),A(LIPLAT),A(LKLATT),A(LKSLAT),
408:         G      A(LNVLAT),A(LIPCEL),A(LLTYP) ,A(LICTYP),A(LKDALT),
409:         G      A(LDALT) ,A(LKZREG),
410:         W      DW1 ,DW2 ,DW3 ,AA(N1+1),BB(N1+1),CC(N1+1),
411:         W      DI(N1+1),DW4 ,DW5 ,W(N1+1),IGI )
412:       end if
413: C
414: C      .... ADJUST PARTICLE NUMBERS IN STACK ....
415: C      AND NUMBER OF IMAGINARY PARTICLE
416: C
417:       if ( JSTG.eq.0 ) then
418:       IMNFL(IZD) = IMNFL(IZD) + NN2
419:       IMNFL(NZONE+1) = IMNFL(NZONE+1) + NN2
420:       else
421: C
422: C
423:       NL      = IMNLT(NLBZ+1)
424: *VOCL LOOP,NOVREC
425:       do 320 J = 1, NN2
426:       IMSLT(NL+J) = IMSFL(N+J)
427:       IMZLT(NL+J) = IZD
428: 320      continue
429:       IMNLT(IZD) = IMNLT(IZD) + NN2
430:       IMNLT(NLBZ+1) = IMNLT(NLBZ+1) + NN2
431:       end if
432: C
433:       NN      = NN - NNN
434:       NIMPT   = NIMPT + NN2
435:       end if
436: check
437: c      write(6,*) '*** (SUNCL) nn,nnn,nl,nn2,jstg=',nn,nnn,nl,nn2,jstg
438: c      write(6,*) ' nimpt,ndimpt=',nimpt,ndimpt
439: c      write(6,*) ' imded=',(imded(j),j=1,ndimpt)
440: c      write(6,*) ' imslt=',(imslt(nl+j),j=1,nn2)
441: c      write(6,*) ' imzlt=',(imzlt(nl+j),j=1,nn2)
442: c      write(6,*) ' xxx =',(xxx(imslt(nl+j),2),j=1,nn2)
443: c      write(6,*) ' yyy =',(yyy(imslt(nl+j),2),j=1,nn2)
444: c      write(6,*) ' zzz =',(zzz(imslt(nl+j),2),j=1,nn2)
445: c      write(6,*) ' aaa =',(aaa(imslt(nl+j),2),j=1,nn2)
446: c      write(6,*) ' bbb =',(bbb(imslt(nl+j),2),j=1,nn2)
447: c      write(6,*) ' ccc =',(ccc(imslt(nl+j),2),j=1,nn2)
448: c      write(6,*) ' levl =',(levl(imslt(nl+j),2),j=1,nn2)
449: c      do 1000 l=1,levl(imslt(nl+1),2)
450: c      write(6,*) ' level =',l
451: c      write(6,*) ' lzz =',(lzzi(imslt(nl+j),1),j=1,nn2)
452: c      write(6,*) ' lpos =',(lposi(imslt(nl+j),1),j=1,nn2)
453: c      write(6,*) ' lcrsi =',(lcrsi(imslt(nl+j),1),j=1,nn2)
454: c1000 continue
455: C
456: C *** CALL TRACKING CONTROL ROUTINE FOR IMAGINARY PARTICLES
457: C      ( VALUES OF NIMPT,NDIMPT, IMNFL MAY BE CHANGED )
458: C
459:       if ( NDIMPT.eq.0 ) then
460:       call NXTEE( 2, A, H, NDIMPT, NIMPT )
461:       end if
462:       go to 300
463:
464:       end if
465: C
466: 200 continue
467:       return
468:       end

```

src/gmvp/tallyo.f

```

1:      subroutine TALLYO( TITLE, A,      CHA,      NTHIST,WSUM,  WLEK,
2:      &                  NPKIND,NGROUP,NREG,  NTREG,  NRESP, NBATCH,
3:      &                  NLENG, NGP1,   NGP2,   SFLTR, SFLCL, SRETR,
4:      &                  SRECL, ENGYB, RVOL,  TRVOL,  NSKIP, NMXLG,
5:      &                  NEITER,WCNTR, XSOC,  XSXV,  XKEF,  NPDET,
6:      &                  SFLPD, SREPD, TFLHS, ITRNM, TNAMS, NNAMES,
7:      &                  NSTALY,TIMEB, NTIME, XSOCB, XKB,   COVXK,
8:      &                  XKMLE, XKERR, CORR,  IWK1,  XKSD,  XKBB,  CA,
9:      &                  CR, UU )
10: C=<GMVP>=====
11: C  PURPOSE: CALCULATE FLUX, REACTION RATE AND THEIR ERRORS, AND
12: C            PRINT OUT THEM, AND OUTPUT TO FILE (FT30F001)
13: C  CALLED IN: CADENZ
14: C  CALL      : DALU, DLUIV ( SSL PREPARED BY FUJITSU) WHICH CALCULATE
15: C            : INVERSE MATRIX
16: C
17: C=====
18: Cccc  implicit real*8(D)
19: C      implicit real*8(A-H,O-Z)
20: C
21: C      real A(*)
22: C      character*4 CHA(*)
23: C
24: C      include 'INC/_FLAGS'
25: C      include '../shared/INC/_IOUNIT'
26: C      include '../shared/INC/_TASKDT'
27: C      include 'INC/_IOUNIT2'
28: C      parameter( NL      = 8 )
29: C
30: C      character*72 TITLE(2)
31: C/#IF INTEGER8
32: C      *      integer*8 NTHIST
33: C/#ELSE
34: C      integer      NTHIST
35: C/#ENDIF
36: C      real*8 WSUM, WLEK, WCNTR(*)
37: C      real*8 SFLTR(NGROUP,NTREG,2), SFLCL(NGROUP,NTREG,2),
38: C      &      SRETR(NTREG,NRESP,2), SRECL(NTREG,NRESP,2)
39: C      real*8 XSOC(NLENG), XSXV(NLENG,3), XKEF(5,NLENG,3)
40: C      real*8 SFLPD(NGROUP,NPDET,2), SREPD(NPDET,NRESP,2)
41: C      real*8 TFLHS(NPKIND,2)
42: C      real ENGYB(*), RVOL(NREG)
43: C      real TRVOL(NTREG)
44: C      real TIMEB(NTIME+1)
45: C
46: C      ... region name table ...
47: C      include '../shared/INC/_LNAM'
48: C
49: C      integer ITRNM(NTREG)
50: C      character*(LNAM) TNAMS(NNAMES)
51: C
52: C      ... working arrays
53: C
54: C      real*8 XSOCB(NBATCH), XKB(5,NBATCH), COVXK(4,4,NBATCH),
55: C      &      XKMLE(5,NBATCH), XKERR(5,NBATCH), CORR(4,4,3)
56: C      integer IWK1(*)
57: C
58: C      real*8 XKSD(4,NBATCH)
59: C      real*8 XKBB(NBATCH), CA(NMXLG), CR(NMXLG)
60: C
61: C      real*4 UU(NGROUP)
62: C
63: C      ... local variables
64: C
65: C      real*8 X1, X2, X3, X4

```

```

66: C      character*12 DAT
67: C      character*8 TIM
68: C
69: C      character*128 NAME
70: C
71: C      character*32 TAG
72: C
73: C      ... criterion to reject too mach correlated combination of
74: C      estimation in maximum likelihood estimation.
75: C
76: C      parameter( DECORR      = 1.0D-5 )
77: C      character*40 ESTCMB
78: C
79: C      integer JJCC(4)
80: C
81: C      real*8 FMIX(4,5)
82: C
83: C
84: C      real*8 AVG(12), SIGA(12), SIGR(12)
85: C      character*48 ESTKEF(12)
86: C
87: C -----
88: C
89: C      .... OUTPUT TO BINARY FILE (I) .....
90: C
91: C      WRITE(IORS)
92: C      &' OUTPUT FILE TYPE 1 (FROM 9/MARCH/1990)
93: C
94: C      WRITE(IORS)
95: C      &' OUTPUT FILE TYPE 2.0 (FROM 9/JUNE/1991)
96: C
97: C      write(IORS)
98: C      &' OUTPUT FILE TYPE 3.0 (FROM 01/MAY/1994)
99: C
100: C      write(IORS)
101: C      &' OUTPUT FILE TYPE 3.01 (FROM 14/MAY/1995)
102: C      write(IORS)
103: C      &' OUTPUT FILE TYPE 3.02 (FROM 8/MAR/1996)
104: C      write(IORS)
105: C      &'GMVP OUTPUT FILE TYPE 3.03 (FROM 3/AUG/1998)
106: C      write(IORS)
107: C      &'GMVP OUTPUT FILE TYPE 3.04 (FROM 27/OCT/1998)
108: C
109: C      call HIZUKE( DAT )
110: C      call JIKAN( TIM )
111: C
112: C      write(IORS) 'DATE ', DAT, ' TIME ', TIM
113: C
114: C      write(IORS) TITLE
115: C
116: C      0-----1-----2-----3--
117: C      TAG      = 'PROBLEM PARAMETERS
118: C      0-----1-----2-----3--
119: C
120: C      WRITE(IORS) TAG,
121: C      &      WSUM,NTHIST, NGROUP,NREG,NRESP,NBATCH,JRESP,JEIGN,NSKIP,
122: C      &      NPDET, NTREG
123: C
124: C      write(IORS) TAG, WSUM, NTHIST, NBATCH, NGROUP, NREG, NRESP, NSKIP,
125: C      &      NSTAL, NGP1, NGP2, NTREG, NPDET, NTIME, NSTALY,
126: C      &      JNEUT, JPHOT, JRESP,
127: C      &      JEIGN, JADJM
128: C
129: C      ... added JTIME,NTIME & NSTALY (ver 3.02) ...
130: C

```

src/gmvp/tallyo.f

```

131: C      write(IORS) TAG, WSUM, NTHIST, NBATCH, NGROUP, NREG, NRESP, NSKIP,
132: C      &          NSTAL, NGP1, NGP2, NTREG, NPDET,
133: C      &          JNEUT, JPHOT, JRESP,
134: C      &          JEIGN, JADJM, JTIME,
135: C      &          NTIME, NSTALY
136: C
137: C ... (Ver 3.03) ...
138: C
139: C      write(IORS) TAG, WSUM, I8TOI4(NTHIST), NBATCH, NGROUP, NREG,
140: C      &          NRESP, NSKIP,
141: C      &          NSTAL, NGP1, NGP2, NTREG, NPDET, NTIME, NSTALY, JNEUT,
142: C      &          JPHOT, JRESP, JEIGN, JADJM, JTIME
143: C
144: C      0-----1-----2-----3--
145: C      TAG      = 'ENERGY & REG.VOLUME'
146: C      -----
147: C
148: C      NGB      = NGROUP + 1
149: C      if ( JNEUT*JPHOT.ne.0 ) NGB = NGB + 1
150: C
151: C ... added TIMEB ( ver 3.02 )
152: C
153: C      write(IORS) TAG, (ENGYB(I),I=1,NGB), (TRVOL(I),I=1,NTREG),
154: C      &          (RVOL(I),I=1,NREG), (TIMEB(I),I=1,NTIME+1)
155: C
156: C ... change energy boundaries similar to MVP ( ver 3.03 )
157: C
158: C      IS      = NGP1 + 1
159: C      if ( JNEUT*JPHOT.eq.0 ) IS = 0
160: C      write(IORS) TAG, (ENGYB(I),I=1,NGP1+1), (ENGYB(I
161: C      &          +IS),I=1,NGP2+1), (TRVOL(I),I=1,NTREG),
162: C      &          (RVOL(I),I=1,NREG), (TIMEB(I),I=1,NTIME+1)
163: C
164: C      0-----1-----2-----3--
165: C      TAG      = 'TALLY REGION NAME'
166: C      -----
167: C
168: C      do 100 KR = 1, NTREG
169: C      if ( ITRNM(KR).lt.0 ) then
170: C      write(IORS) TAG, KR, LEN(TNAMS(1)), TNAMS(ABS(ITRNM(KR)))
171: C      else
172: C      call REGNM0( ITRNM(KR), '>', NAME, LNM, A, CHA )
173: C      write(IORS) TAG, KR, LNM, NAME(:LNM)
174: C      end if
175: C 100 continue
176: C
177: C
178: C .... CONVERT ENERGY BOUNDARY ARRAY TO LETHARGY WIDTH
179: C
180: C      if ( JADJM.eq.0 ) then
181: C      do 110 I = 1, NGP1
182: C      UU(I) = LOG(ENGYB(I)/ENGYB(I+1))
183: C 110 continue
184: C
185: C      if ( NGP2.ne.0 ) then
186: C      do 120 I = 1, NGP2
187: C      UU(I+NGP1) = LOG(ENGYB(NGP1+I+1)/ENGYB(NGP1+I+2))
188: C 120 continue
189: C      end if
190: C      else
191: C      do 130 I = 1, NGROUP
192: C      UU(I) = 1.0
193: C 130 continue
194: C      end if
195: C

```

```

196: C .... CALCULATE AVERAGE & ESTIMATED STANDARD DEVIATION (FLUX) ...
197: C
198: C      if ( (NBATCH-NSKIP).lt.2 ) then
199: C      write(IOT,'(1x,a,a)')
200: C      &          '!!! I CAN'T CALCULATE VARIANCE BECAUSE BATCH ',
201: C      &          'NUMBER (NBATCH-NSKIP) IS LESS THAN 2. '
202: C      DNB      = 0.0
203: C      else
204: C      if ( JEIGN.eq.0 ) then
205: C      DNB      = 1.0/DBLE(NBATCH-1-NSKIP)
206: C      else
207: C      DNB      = 1.0/DBLE(NTASK*(NBATCH-NSKIP)-1)
208: C      end if
209: C      end if
210: C
211: C      DN      = 1.0/WSUM
212: C
213: C      do 140 K = 1, NGROUP*NTREG
214: C      SFLTR(K,1,2) =
215: C      &          SQRT(ABS(SFLTR(K,1,2)-(SFLTR(K,1,1)**2)*DN)*DN*DNB)
216: C      SFLTR(K,1,1) = SFLTR(K,1,1)*DN
217: C      if ( SFLTR(K,1,1).ne.0.0 ) SFLTR(K,1,2) = SFLTR(K,1,2) /
218: C      &          SFLTR(K,1,1)*100.0
219: C      SFLCL(K,1,2) =
220: C      &          SQRT(ABS(SFLCL(K,1,2)-(SFLCL(K,1,1)**2)*DN)*DN*DNB)
221: C      SFLCL(K,1,1) = SFLCL(K,1,1)*DN
222: C      if ( SFLCL(K,1,1).ne.0.0 ) SFLCL(K,1,2) = SFLCL(K,1,2) /
223: C      &          SFLCL(K,1,1)*100.0
224: C 140 continue
225: C
226: C      do 160 I = 1, NTREG
227: C      do 150 N = 1, NGROUP
228: C      SFLTR(N,I,1) = SFLTR(N,I,1) /UU(N) /TRVOL(I)
229: C      SFLCL(N,I,1) = SFLCL(N,I,1) /UU(N) /TRVOL(I)
230: C 150 continue
231: C 160 continue
232: C
233: C      ..... NEXT EVENT (POINT) DETECTOR TALLIES .....
234: C
235: C      if ( NPDET.gt.0 ) then
236: C      do 180 N = 1, NPDET
237: C      do 170 K = 1, NGROUP
238: C      SFLPD(K,N,2) =
239: C      &          SQRT(ABS(SFLPD(K,N,2)-(SFLPD(K,N,1)**2)*DN)*DN*
240: C      &          DNB)
241: C      SFLPD(K,N,1) = SFLPD(K,N,1)*DN
242: C      if ( SFLPD(K,N,1).ne.0.0 ) SFLPD(K,N,2) = SFLPD(K,N,2) /
243: C      &          SFLPD(K,N,1)*100.0
244: C 170 continue
245: C 180 continue
246: C
247: C      do 200 N = 1, NPDET
248: C      do 190 K = 1, NGROUP
249: C      SFLPD(K,N,1) = SFLPD(K,N,1) /UU(K)
250: C 190 continue
251: C 200 continue
252: C
253: C      end if
254: C
255: C ..... OUTPUT TO BINARY FILE (II) FLUXES & THEIR ERROR
256: C
257: C
258: C      -----0-----1-----2-----3--
259: C      TAG      = 'FLUX & ERROR: TRACK LENGTH'
260: C      -----

```

src/gmvp/tallyo.f

```

261: C
262:     do 210 I = 1, NTREG
263:         write(IORS) TAG, (SFLTR(N,I,1),N=1,NGROUP),
264:         & (SFLTR(N,I,2),N=1,NGROUP)
265:     210 continue
266: C
267: C
268: C
269: C     -----0-----*-----1-----*-----2-----*-----3---
270:     TAG = 'FLUX & ERROR: COLLISION'
271: C
272: C
273:     do 220 I = 1, NTREG
274:         write(IORS) TAG, (SFLCL(N,I,1),N=1,NGROUP),
275:         & (SFLCL(N,I,2),N=1,NGROUP)
276:     220 continue
277: C
278: C     6/13/1991 .....
279: C
280:     if ( NPDET.ne.0 ) then
281: C
282: C     -----0-----*-----1-----*-----2-----*-----3---
283:     TAG = 'FLUX & ERROR: POINT DETECTOR'
284: C
285: C
286:     do 230 I = 1, NPDET
287:         write(IORS) TAG, (SFLPD(N,I,1),N=1,NGROUP),
288:         & (SFLPD(N,I,2),N=1,NGROUP)
289:     230 continue
290: C
291:     end if
292: C
293: C     .... REACTION RATES IF ANY .....
294: C
295:     if ( JRESP.ne.0 ) then
296: C
297:         do 240 K = 1, NTREG*NRESP
298:             SRETR(K,1,2) =
299:             & SQRT(ABS(SRETR(K,1,2)-(SRETR(K,1,1)**2)*DN)*DN*DNB)
300:             SRETR(K,1,1) = SRETR(K,1,1)*DN
301:             if ( SRETR(K,1,1).ne.0.0 ) SRETR(K,1,2) = SRETR(K,1,2) /
302:             & SRETR(K,1,1)*100.0
303:             SRECL(K,1,2) =
304:             & SQRT(ABS(SRECL(K,1,2)-(SRECL(K,1,1)**2)*DN)*DN*DNB)
305:             SRECL(K,1,1) = SRECL(K,1,1)*DN
306:             if ( SRECL(K,1,1).ne.0.0 ) SRECL(K,1,2) = SRECL(K,1,2) /
307:             & SRECL(K,1,1)*100.0
308:         240 continue
309: C
310:         do 260 I = 1, NRESP
311:             do 250 N = 1, NTREG
312:                 SRETR(N,I,1) = SRETR(N,I,1) /TRVOL(N)
313:                 SRECL(N,I,1) = SRECL(N,I,1) /TRVOL(N)
314:             250 continue
315:         260 continue
316: C
317:         if ( NPDET.ne.0 ) then
318:             do 270 K = 1, NPDET*NRESP
319:                 SREPD(K,1,2) =
320:                 & SQRT(ABS(SREPD(K,1,2)-(SREPD(K,1,1)**2)*DN)*DN*
321:                 & DNB)
322:                 SREPD(K,1,1) = SREPD(K,1,1)*DN
323:                 if ( SREPD(K,1,1).ne.0.0 ) SREPD(K,1,2) = SREPD(K,1,2) /
324:                 & SREPD(K,1,1)*100.0
325:             270 continue

```

```

326:         end if
327: C
328:     end if
329: C
330: C     .... OUTPUT TO BINARY FILE (III) REACTION RATES & THEIR ERROR
331: C
332:     if ( JRESP.ne.0 ) then
333: C
334: C     -----0-----*-----1-----*-----2-----*-----3---
335:     TAG = 'REACTION RATE: TRACK LENGTH'
336: C
337: C
338: C
339:     do 280 I = 1, NRESP
340:         write(IORS) TAG, (SRETR(N,I,1),N=1,NTREG),
341:         & (SRETR(N,I,2),N=1,NTREG)
342:     280 continue
343: C
344: C
345: C
346: C     -----0-----*-----1-----*-----2-----*-----3---
347:     TAG = 'REACTION RATE: COLLISION'
348: C
349: C
350:     do 290 I = 1, NRESP
351:         write(IORS) TAG, (SRECL(N,I,1),N=1,NTREG),
352:         & (SRECL(N,I,2),N=1,NTREG)
353:     290 continue
354: C
355:     if ( NPDET.ne.0 ) then
356: C
357: C     -----0-----*-----1-----*-----2-----*-----3---
358:     TAG = 'REACTION RATE: POINT DETECTOR'
359: C
360: C
361:     do 300 I = 1, NRESP
362:         write(IORS) TAG, (SREPD(N,I,1),N=1,NPDET),
363:         & (SREPD(N,I,2),N=1,NPDET)
364:     300 continue
365:     end if
366: C
367: C     .... weighted time of flight
368: C
369: C     ....
370: C
371:     if ( JTIME.ne.0 ) then
372:         do 310 K = 1, NPKIND
373: C
374:             TFLHS(K,2) =
375:             & SQRT(ABS(TFLHS(K,2)-(TFLHS(K,1)**2)*DN)*DN*DNB)
376:             TFLHS(K,1) = TFLHS(K,1)*DN
377:             if ( TFLHS(K,1).ne.0. ) TFLHS(K,2) = TFLHS(K,2) /TFLHS(K,1)
378:             & *100.0
379:         310 continue
380:     end if
381: C
382: C
383: C**** STATISTICAL ANALYSIS OF K-EFFECTIVE
384: C
385:     if ( JEIGN.ne.0 ) then
386: C
387: C
388: C
389: C     =====
390: C     call HEADER( IOT, 'RESULTS OF EIGENVALUE CALCULATION' )
391: C     =====

```

src/gmvp/tallyo.f

```

391: C
392: C *****
393: C      call LABEL( IOT, 'EIGEN-VALUE ( K-EFFECTIVE )' )
394: C *****
395: C
396: C..... SET WCNTR(7) = TOTAL LEAKAGE IN THIS RUN
397: C
398: C      WCNTR(7) = WLEK
399: C
400: C      XSOCB(1) = XSOC(1)
401: C      XKEF(3,1) = XKEF(3,1) + XKEF(5,1)
402: C      XKEF(4,1) = XKEF(4,1) + XKEF(5,1)
403: C      XKB(1,1) = XKEF(1,1) /XSOC(1)
404: C      XKB(2,1) = XKEF(2,1) /XSOC(1)
405: C      XKB(3,1) = XKEF(3,1) /XKEF(3,1)
406: C      XKB(4,1) = XKEF(4,1) /XKEF(4,1)
407: C      XKMLE(1,1) = XKB(1,1)
408: C      XKMLE(2,1) = XKB(2,1)
409: C      XKMLE(3,1) = XKB(3,1)
410: C      XKMLE(4,1) = XKB(4,1)
411: CC
412: CC XSOCB : SOURCE          (EACH BATCH)
413: CC XKB  : ESTIMATOR OF KEFF (EACH BATCH)
414: CC XKMLE : CUMULATIVE ESTIMATOR OF KEFF TO THIS BATCH
415: CC      1: TRACK LENGTH ESTIMATOR (PRODUCTION RATE)
416: CC      2: COLLISION ESTIMATOR   (PRODUCTION RATE)
417: CC      3: TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
418: CC      4: COLLISION ESTIMATOR   (NEUTRON BALANCE)
419: CC
420: C      do 330 I = 2, NBATCH
421: C      XKEF(3,I) = XKEF(3,I) + XKEF(5,I)
422: C      XKEF(4,I) = XKEF(4,I) + XKEF(5,I)
423: C      XKMLE(1,I) = XKEF(1,I) /XSOC(I)
424: C      XKMLE(2,I) = XKEF(2,I) /XSOC(I)
425: C980809 XKMLE(3,I) = XKEF(1,I) /XKEF(3,I)
426: C980809 XKMLE(4,I) = XKEF(2,I) /XKEF(4,I)
427: C      XSOCB(I) = XSOC(I) - XSOC(I-1)
428: C      XKB(1,I) = (XKEF(1,I)-XKEF(1,I-1)) /XSOCB(I)
429: C      XKB(2,I) = (XKEF(2,I)-XKEF(2,I-1)) /XSOCB(I)
430: C      XKB(3,I) = (XKEF(3,I)-XKEF(3,I-1)) /
431: C      &      (XKEF(3,I)-XKEF(3,I-1))
432: C      XKB(4,I) = (XKEF(4,I)-XKEF(4,I-1)) /
433: C      &      (XKEF(4,I)-XKEF(4,I-1))
434: C980809
435: C ... new: source weighted "balance" estimators
436: C
437: C      XKMLE(3,I) = 0.0D0
438: C      XKMLE(4,I) = 0.0D0
439: C      do 320 J = 1, I
440: C      XKMLE(3,I) = XKMLE(3,I) + XKB(3,J)*XSOCB(J)
441: C      XKMLE(4,I) = XKMLE(4,I) + XKB(4,J)*XSOCB(J)
442: C320      continue
443: C      XKMLE(3,I) = XKMLE(3,I) /XSOC(I)
444: C      XKMLE(4,I) = XKMLE(4,I) /XSOC(I)
445: C
446: C330      continue
447: C
448: C      write(IOT,7000)
449: C7000      format(/23X,'K-EFFECTIVE ( EACH BATCH )',30X,
450: C      &      'K-EFFECTIVE ( CUMULATIVE TO THIS BATCH )'/'/' ,
451: C      &      '      PRODUCTION          ',
452: C      &      '      NEUTRON BALANCE      ',
453: C      &      '      PRODUCTION          ',
454: C      &      '      NEUTRON BALANCE      '/'/' ,
455: C      &      ' BATCH TRACK EST.    COLLISION EST.',

```

```

456: C      &      ' TRACK EST.    COLLISION EST.',
457: C      &      ' TRACK EST.    COLLISION EST.',
458: C      &      ' TRACK EST.    COLLISION EST.')
459: C
460: C      do 340 I = 1, NBATCH
461: C      write(IOT,7020) I, (XKB(J,I),J=1,4), (XKMLE(J,I),J=1,4)
462: C340      continue
463: C7020      format(' ',I5,1P,4E15.5,5X,1P,4E15.5)
464: C
465: C
466: C
467: C ..... OUTPUT TO BINARY FILE (IV)      EIGEN VALUES
468: C
469: C
470: C      -----0-----*-----1-----*-----2-----*-----3-----
471: C      TAG      = 'K-EFF: BATCH & CUMULATIVE      '
472: C      -----
473: C
474: C      write(IORS) TAG, ((XKB(J,I),I=1,NBATCH),J=1,4),
475: C      &      ((XKMLE(J,I),I=1,NBATCH),J=1,4)
476: C
477: C      S1      = XSOC(NBATCH)
478: C      X1      = XKEF(1,NBATCH)
479: C      X2      = XKEF(2,NBATCH)
480: C980809 X3      = XKEF(3,NBATCH)
481: C980809 X4      = XKEF(4,NBATCH)
482: CCC
483: CCC XSOC(I) : SOURCE. CUMULATIVE TO I'TH BATCH
484: CCC      ---> SUM OF I'TH BATCH TO THE LAST BATCH.
485: CCC
486: CCC SIMILAR TO XKEF(4,I) KEFF (PRODUCTION & LOSS)
487: CCC
488: C      do 350 I = 2, NBATCH
489: C      J      = NBATCH - I + 2
490: C      XSOC(J) = S1 - XSOC(J-1)
491: C      XKEF(1,J) = (X1-XKEF(1,J-1)) /XSOC(J)
492: C      XKEF(2,J) = (X2-XKEF(2,J-1)) /XSOC(J)
493: C980809 XKEF(3,J) = (X1-XKEF(1,J-1)) / (X3-XKEF(3,J-1))
494: C980809 XKEF(4,J) = (X2-XKEF(2,J-1)) / (X4-XKEF(4,J-1))
495: C350      continue
496: C
497: C      XSOC(1) = S1
498: C      XKEF(1,1) = X1/S1
499: C      XKEF(2,1) = X2/S1
500: C980809 XKEF(3,1) = X1/X3
501: C980809 XKEF(4,1) = X2/X4
502: C980809
503: C ... new: source weighted "balance" estimators
504: C
505: C      do 370 I = 1, NBATCH
506: C      XKEF(3,I) = 0.0D0
507: C      XKEF(4,I) = 0.0D0
508: C      do 360 J = I, NBATCH
509: C      XKEF(3,I) = XKEF(3,I) + XKB(3,J)*XSOCB(J)
510: C      XKEF(4,I) = XKEF(4,I) + XKB(4,J)*XSOCB(J)
511: C360      continue
512: C      XKEF(3,I) = XKEF(3,I) /XSOC(I)
513: C      XKEF(4,I) = XKEF(4,I) /XSOC(I)
514: C370      continue
515: C
516: C..... COVXX : COVARIANCE MATRIX
517: C
518: C      do 380 K = 1, (NBATCH-1)*4*4
519: C      COVXX(K,1,1) = 0.0
520: C380      continue

```

src/gmvp/tallyo.f

```

521:
522:   do 420 I = 1, NBATCH - 1
523:     do 410 K = 1, 4
524:       do 400 L = 1, 4
525:         if ( L.ge.K ) then
526:           do 390 J = I, NBATCH
527:             X1 = XSOCB(J) / (XSOC(I)-XSOCB(J))
528:             COVXK(K,L,I) = COVXK(K,L,I)
529:             & + X1*(XKB(K,J)-XKEF(K,I))*
530:             & (XKB(L,J)-XKEF(L,I))
531:           390 continue
532:         else
533:           COVXK(K,L,I) = COVXK(L,K,I)
534:         end if
535:       400 continue
536:     410 continue
537:   420 continue
538:
539: C980808 ... do not destroy XSOCB
540: C..... XSOCB : TEMPORARY STORAGE FOR 1 / NO. OF BATCHES CONSIDERED
541: C
542: C   do 370 I = 1, NBATCH - 1
543: C     XSOCB(I) = 1./ (NBATCH-I+1)
544: C 370 continue
545:
546: C   do 450 I = 1, NBATCH - 1
547: C     XX = 1.D0/(NBATCH-I+1)
548: C     do 440 K = 1, 4
549: C       do 430 L = 1, 4
550: C         COVXK(K,L,I) = COVXK(K,L,I)*XX
551: C       430 continue
552: C     440 continue
553: C   450 continue
554:
555: C..... KEEP CORRELATION MATRIX FOR ESTIMATION AFTER (NSKIP+1)TH BATCH
556:
557:   NSKIP1 = NSKIP + 1
558:
559:   do 470 K = 1, 4
560:     do 460 L = 1, 4
561:       CORR(K,L,1) = COVXK(K,L,NSKIP1) /
562:       & SQRT(COVXK(K,K,NSKIP1)*COVXK(L,L,NSKIP1))
563:     460 continue
564:   470 continue
565: C
566: C..... ESTIMATION OF KEFF BY MAXIMUM LIKELYHOOD METHOD
567: C   XKMLE : KEFF
568: C   XKERR : STANDARD DEVIATION OF COMBINED ESTIMATION (%)
569: C     1 : PRODUCTION (TRK+COL)
570: C     2 : BALANCE (TRK+COL)
571: C     3 : TRACK LENGTH (PRODUCTION + BALANCE)
572: C     4 : COLLISION (PRODUCTION + BALANCE)
573: C     5 : ALL ESTIMATORS
574: C..... ESTIMATION OF KEFF BY EACH ESTIMATOR
575: C   XKEF : KEFF
576: C   XKB : STANDARD DEVIATION (%)
577: C     1 : TRACK LENGTH (PRODUCTION)
578: C     2 : COLLISION (PRODUCTION)
579: C     3 : TRACK LENGTH (BALANCE)
580: C     4 : COLLISION (BALANCE)
581: C
582:   do 490 J = 1, 4
583:     do 480 I = 1, NBATCH - 1
584:       XKSD(J,I) = SQRT(ABS(COVXK(J,J,I))) /XKEF(J,I)*100.
585:     480 continue

```

```

586:   490 continue
587: C
588: C *****
589: C   call LABEL( IOT, 'K-EFFECTIVE ( CUMULATIVE AFTER THIS BATCH )'
590: C & )
591: C *****
592: C
593: C
594: C   write(IOT,7040)
595: C 7040 format(/1X,
596: C & '
597: C & '
598: C & 1X,
599: C & ' BATCH TRACK LENGTH EST. COLLISION EST.
600: C & ' TRACK LENGTH EST. COLLISION EST.'
601: C & )
602:
603:   do 500 I = 1, NBATCH - 1
604:     write(IOT,7060) I, (XKEF(J,I),XKSD(J,I),J=1,4)
605:   500 continue
606:
607: C 7060 format(1X,I5,2(1P,E15.5,' (' ,0P,F6.4,'%')'),5X,2
608: C & (1P,E15.5,' (' ,0P,F6.4,'%')'))
609: C
610: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF 2 ESTIMATORS
611: C
612: C .... clear mixing factor of estimators (used in "real variance"
613: C calculation.
614: C
615:   do 520 J = 1, 5
616:     do 510 K = 1, 4
617:       FMIX(K,J) = 0.0D0
618:     510 continue
619:   520 continue
620: C
621: JCORR1 = 0
622: do 540 J = 1, 4
623:   if ( J.eq.1 ) then
624:     K = 1
625:     L = 2
626:     ESTCMB = 'production (track & coolision)'
627:   else if ( J.eq.2 ) then
628:     K = 3
629:     L = 4
630:     ESTCMB = 'balance (track & coolision)'
631:   else if ( J.eq.3 ) then
632:     K = 1
633:     L = 3
634:     ESTCMB = 'track length (production & balance)'
635:   else if ( J.eq.4 ) then
636:     K = 2
637:     L = 4
638:     ESTCMB = 'collision (production & balance)'
639:   end if
640: C
641:   do 530 I = 1, NBATCH - 2
642:     CORR1 = COVXK(K,L,I) /SQRT(COVXK(K,K,I)*COVXK(L,L,I))
643:     if ( ABS(CORR1-1.0D0).lt.DECORR ) then
644:       if ( JCORR1.eq.0 ) then
645:         JCORR1 = 1
646:         write(IOT,(''1''/1X,a,a/1X,a,E12.5/'))
647:         & '!!! Too strongly correlated estimator pairs ',
648:         & 'are found in M.L.E. calculation.',
649:         & ' (criterion : abs(correlation coeff.-1) < ',
650:         & DECORR

```

src/gmvp/tallyo.f

```

651:          write(IOT,'(/a/)')
652:      &          ' Dummy values of -1.0 will be output.'
653:      end if
654:      write(IOT,'(lx,a,i4,a,a,e14.7)')
655:      &          ' Cumulative after batch ', I, ' type : ',
656:      &          ESTCMB, ' : correlation =', CORR1
657:      XKMLE(J,I) = -1.0D0
658:      XKERR(J,I) = 0.0D0
659:      else
660:          X1 = COVXK(K,K,I) - COVXK(K,L,I)
661:          X2 = COVXK(L,L,I) - COVXK(K,L,I)
662:          XKMLE(J,I) = (X1*XKEF(L,I)+X2*XKEF(K,I)) / (X1+X2)
663:          XKERR(J,I) =
664:      &          (COVXK(K,K,I)*COVXK(L,L,I)-COVXK(K,L,I)*
665:      &          COVXK(K,L,I)) / (X1+X2)
666:          XKERR(J,I) = SQRT(ABS(XKERR(J,I))) / XKMLE(J,I)*100.
667:          if (I.eq.NSKIP1) then
668:              FMIX(L,J) = X1/(X1+X2)
669:              FMIX(K,J) = X2/(X1+X2)
670:          end if
671:      end if
672: 530      continue
673: 540      continue
674: C
675: C
676: C.... GET MAXIMUM LIKELYHOOD ESTIMATION OF ALL 4 ESTIMATORS
677: C      GET AN INVERSE MATRIX OF COVXK INTO COVXK
678: C
679: C
680: C      ... covariance matrix has (NEST+1)*NEST/2 degree of freedom.
681: C      (where NEST is number of combined estimators. here NEST=4)
682: C
683: C      To keep the degree of freedom, required number of
684: C      batches are calculated as;
685: C
686: C      NEST* NB - NEST >= (NEST+1)*NEST/2 --> NB >= (NEST+3)/2
687: C
688: C      ( In the equation above, "-NEST" is necessary because
689: C      expected values of each estimator are approximated by
690: C      ensemble means. )
691: C
692: C      NB00 = 2
693: C
694: C      ICON1 = 0
695: C
696: C      do 550 I = 1, NBATCH - 2
697: C          XKERR(5,I) = 0.0
698: C          XKMLE(5,I) = -1.0
699: 550      continue
700: C
701: C      do 640 I = 1, NBATCH - 2 - NB00
702: C
703: C      ... omit one estimator from too strongly correlated pair ...
704: C      ( non-diagonal elements of COVXK for omitted estimator is
705: C      cleared to 0.0 )
706: C
707: C      do 590 L = 4, 1, -1
708: C          JJCC(L) = 0
709: C          do 570 K = 1, L - 1
710: C              CORR1 = COVXK(K,L,I) /
711:      &          SQRT(COVXK(K,K,I)*COVXK(L,L,I))
712: C          if ( ABS(CORR1-1.0D0).lt.DECORR ) then
713: C              JJCC(L) = 1
714: C              COVXK(L,L,I) = 1.0D0
715: C              do 560 KK = 1, 4

```

```

716:          if ( KK.ne.L ) then
717:              COVXK(KK,L,I) = 0.0D0
718:              COVXK(L,KK,I) = 0.0D0
719:          end if
720: 560      continue
721:          go to 580
722:          end if
723: 570      continue
724: 580      continue
725: 590      continue
726: C
727: C      ... invert covariance matrix ...
728: C
729: C      call MTXINV( COVXK(1,1,I), 4, IWK1, CORR(1,1,2), 1.0D-16,
730:      &          ICON )
731: C
732: C      if ( ICON.ne.0 ) then
733: C          write(IOT,'(lx,a,i4,a)')
734: C          '!!! CANNOT INVERT COVARIANCE MATRIX (BATCH ', I,
735:      &          ' ) --> MAX.LIKELYHOOD ESTIMATOR "ALL" PRINTED AS -1'
736:      &
737: C          if ( I.eq.NSKIP1 ) NSKIP1 = NSKIP1 + 1
738: C
739: C          XKERR(5,I) = 0.0
740: C          XKMLE(5,I) = -1.0
741: C      else
742: C          XKERR(5,I) = 0.0
743: C          XKMLE(5,I) = 0.0
744: C      end if
745: C
746: C      if ( XKMLE(5,I).ne.-1.0 ) then
747: C          do 610 K = 1, 4
748: C              do 600 L = 1, 4
749: C                  if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
750: C                      XKERR(5,I) = XKERR(5,I) + COVXK(K,L,I)
751: C                      XKMLE(5,I) = XKMLE(5,I) + COVXK(K,L,I)*
752:      &                      XKEF(L,I)
753: C                  end if
754: C              continue
755: C          continue
756: C          if ( I.eq.NSKIP1 ) then
757: C              do 630 L = 1, 4
758: C                  FMIX(L,5) = 0.0D0
759: C                  do 620 K = 1, 4
760: C                      if ( JJCC(L).eq.0.and.JJCC(K).eq.0 ) then
761: C                          FMIX(L,5) = FMIX(L,5) + COVXK(K,L,I)
762: C                      end if
763: C                  continue
764: C                  FMIX(L,5) = FMIX(L,5) /XKERR(5,I)
765: C              continue
766: C          end if
767: C          end if
768: C
769: 640      continue
770: C
771: C      do 480 K = 1, 4
772: C          do 470 L = 1, 4
773: C              do 460 I = 1, NBATCH - 1 - NB00
774: C                  if ( XKMLE(5,I).ne.-1.0 ) then
775: C                      XKERR(5,I) = XKERR(5,I) + COVXK(K,L,I)
776: C                      XKMLE(5,I) = XKMLE(5,I) + COVXK(K,L,I)*XKEF(L,I)
777: C                  end if
778: C          continue
779: C          continue
780: C      continue

```

src/gmvp/tallyo.f

```

781:
782:     do 650 I = 1, NBATCH - 2 - NB00
783:         if ( XKMLE(5,I).ne.-1.0 ) then
784:             XKERR(5,I) = 1./XKERR(5,I)
785:             XKMLE(5,I) = XKMLE(5,I)*XKERR(5,I)
786:             XKERR(5,I) = SQRT(ABS(XKERR(5,I))) /XKMLE(5,I)*100.
787:         end if
788:     650 continue
789: C
790:     write(IOT,(''1''))
791: C
792: C *****
793: C *****
794: C *****
795: C *****
796: C *****
797: C *****
798:     write(IOT,7080)
799: 7080 format(/' < CUMULATIVE AFTER EACH BATCH >'//
800: & ' BATCH PRODUCTION NEUTRON BALANCE
801: & ' TRACK LENGTH EST. COLLISION EST.
802: & ' ALL ESTIMATORS')
803:
804:     do 660 I = 1, NBATCH - 2 - NB00
805:         write(IOT,7100) I, (XKMLE(J,I),XKERR(J,I),J=1,5)
806:     660 continue
807:
808:     do 670 I = NBATCH - 2 - NB00 + 1, NBATCH - 2
809:         write(IOT,7120) I, (XKMLE(J,I),XKERR(J,I),J=1,4)
810:     670 continue
811:
812: 7100 format(1X,I5,5(1P,E15.5,' (' ,0P,F6.4,'%)'))
813: 7120 format(1X,I5,4(1P,E15.5,' (' ,0P,F6.4,'%)'),5X,10(' '))
814: C
815: C
816:     write(IOT,(''1''))
817: C
818: C
819: C *****
820: C *****
821: C *****
822: C *****
823: C *****
824:     write(IOT,7140) NSKIP1, NBATCH
825: 7140 format(/' RESULT OF BATCHES FROM ',I4,' TO ',I4//
826: & ' 1 : TRACK LENGTH ESTIMATOR (PRODUCTION)
827: & ' 2 : COLLISION ESTIMATOR (PRODUCTION)
828: & ' 3 : TRACK LENGTH ESTIMATOR (NEUTRON BALANCE)
829: & ' /
830: & ' 4 : COLLISION ESTIMATOR (NEUTRON BALANCE)
831: & ' //)
832: C
833: C ... printout correlation table ....
834: C
835:     write(IOT,7160)
836: 7160 format(3X,8('-----'))
837:     write(IOT,('a'))
838: & ' KEFF ERROR(%) CORRELATION '
839:     write(IOT,7160)
840:     do 680 J = 1, 4
841:         write(IOT,7180) J, XKEF(J,NSKIP1), XKSD(J,NSKIP1),
842: & (CORR(K,J,1),K=1,J)
843:     680 continue
844: 7180 format(1X,I5,1P,E15.5,0P,F10.4,0P,4F9.3)
845:

```

```

846:     write(IOT,7160)
847: C
848: C ... printout mixing table for M.L.E. ....
849: C
850:     write(IOT,7200) ((FMIX(K,J),K=1,4),J=1,5)
851: C
852: 7200 format(/1X,4X,'MIXING RATIO FOR MAXIMUM LIKELIHOOD ESTIMATORS'
853: & //4X,'-----',4('-----')/4X,
854: & ' M.L.E. ',7X,' PRODUCTION ',7X,7X,
855: & ' BALANCE '/4X,' ',2
856: & (' TRACK LENGTH ', COLLISION ') /4X,
857: & '-----',4('-----')/4X,
858: & ' PRODUCTION ',4(F11.5,3X)/4X,' NEUTRON BALANCE',4
859: & (F11.5,3X)/4X,' TRACK LENGTH ',4(F11.5,3X)/4X,
860: & ' COLLISION ',4(F11.5,3X)/4X,' ALL ',4
861: & (F11.5,3X)/4X,'-----',4('-----'))
862: C
863: C
864:     write(IOT,*) ' '
865:     write(IOT,7220) (XKMLE(J,NSKIP1),XKERR(J,NSKIP1),
866: & XKMLE(J,NSKIP1),XKMLE(J,NSKIP1)*XKERR(J,NSKIP1)/100.0,
867: & J=1,5)
868: 7220 format(' RESULTS BY MAXIMUM LIKELIHOOD METHOD'//8X,
869: & 'PRODUCTION --> KEFF=',1P,E14.5,' (' ,0P,F7.4,
870: & %) ',F12.8,' +- ',F10.7/8X,
871: & 'NEUTRON BALANCE--> KEFF=',1P,E14.5,' (' ,0P,F7.4,
872: & %) ',F12.8,' +- ',F10.7/8X,
873: & 'TRACK LENGTH --> KEFF=',1P,E14.5,' (' ,0P,F7.4,
874: & %) ',F12.8,' +- ',F10.7/8X,
875: & 'COLLISION --> KEFF=',1P,E14.5,' (' ,0P,F7.4,
876: & %) ',F12.8,' +- ',F10.7/8X,
877: & 'ALL --> KEFF=',1P,E14.5,' (' ,0P,F7.4,
878: & %) ',F12.8,' +- ',F10.7)
879: C
880:     do 690 M = 1, 3
881:         write(IOT,7240) M, (XKMLE(J,NSKIP1)*
882: & (1.0-M*XKERR(J,NSKIP1)/100.0),XKMLE(J,NSKIP1)*
883: & (1.0+M*XKERR(J,NSKIP1)/100.0),J=1,5)
884:     690 continue
885: C
886: 7240 format(/1X,4X,' RANGE IN ',I1,' STANDARD DEVIATION'//8X,
887: & 'PRODUCTION --> ',F12.9,' < K < ',F12.9/8X,
888: & 'NEUTRON BALANCE--> ',F12.9,' < K < ',F12.9/8X,
889: & 'TRACK LENGTH --> ',F12.9,' < K < ',F12.9/8X,
890: & 'COLLISION --> ',F12.9,' < K < ',F12.9/8X,
891: & 'ALL --> ',F12.9,' < K < ',F12.9)
892: C
893: C
894: C -----
895: C Real variance estimation
896: C -----
897: C
898:     if ( NBATCH-NSKIP.gt.NMXLAG ) then
899:         KK = 0
900:         do 710 J = 1, 4
901:             KK = KK + 1
902: C
903:             if ( J.eq.1 ) then
904:                 ESTKEF(KK) = 'Track length(PRODUCTION) '
905:             else if ( J.eq.2 ) then
906:                 ESTKEF(KK) = 'Collision (PRODUCTION) '
907:             else if ( J.eq.3 ) then
908:                 ESTKEF(KK) = 'Track length(BALANCE) '
909:             else if ( J.eq.4 ) then
910:                 ESTKEF(KK) = 'Collision (BALANCE) '

```


src/gmvp/tallyo.f

```

911:          end if
912: C
913: C      ... calculate batchwise k-eff with weighting factor
914: C
915:          do 700 I = NSKIP + 1, NBATCH
916:              XKBB(I) = XKB(J,I)*XSOCB(I) /XSOC(NSKIP1)*
917:                  & (NBATCH-NSKIP)
918:          700      continue
919: C
920:          EPS      = 1.0D-10
921:          JPR      = -1
922:          IDEBG    = 1
923: C
924:          ESTCMB = 'K-eff by '//ESTKEF(KK)
925:          call REALVR( ESTCMB, XKBB, 1, 1, NSKIP+1, NBATCH,
926:              &          AVG(KK), SIGA(KK), SIGR(KK), CA, CR, NMXLAGE, EPS,
927:              &          NEITER, JPR, IDEBG, IERR )
928: C
929:          710      continue
930: C
931:          do 740 J = 1, 5
932:              KK      = KK + 1
933: C
934:              if ( J.eq.1 ) then
935:                  ESTKEF(KK) = 'M.L.E of PRODUCTION estimators'
936:              else if ( J.eq.2 ) then
937:                  ESTKEF(KK) = 'M.L.E of NEUTRON BALANCE estimators'
938:              else if ( J.eq.3 ) then
939:                  ESTKEF(KK) = 'M.L.E of TRACK LENGTH estimators'
940:              else if ( J.eq.4 ) then
941:                  ESTKEF(KK) = 'M.L.E of COLLISION estimators'
942:              else if ( J.eq.5 ) then
943:                  ESTKEF(KK) = 'M.L.E of ALL estimators'
944:              end if
945: C
946: C      ... calculate batchwise k-eff of combined estimates
947: C
948: C      (XSOC(I): source sum if batch I to NBATCH
949: C
950:          do 730 I = NSKIP + 1, NBATCH
951:              XKBB(I) = 0.0D0
952:              do 720 K = 1, 4
953:                  XKBB(I) = XKBB(I) + XKB(K,I)*FMIX(K,J)
954:          720      continue
955:              XKBB(I) = XKBB(I)*XSOCB(I) /XSOC(NSKIP1)*
956:                  & (NBATCH-NSKIP)
957:          730      continue
958: C
959:          EPS      = 1.0D-10
960:          JPR      = -1
961:          IDEBG    = 1
962: C
963:          ESTCMB = 'K-eff '//ESTKEF(KK)
964:          call REALVR( ESTCMB, XKBB, 1, 1, NSKIP+1, NBATCH,
965:              &          AVG(KK), SIGA(KK), SIGR(KK), CA, CR,
966:              &          NMXLAGE, EPS, NEITER, JPR,
967:              &          IDEBG, IERR )
968: C
969:          740      continue
970: C
971:          NWARN    = 0
972:          write(IOT,7250)
973:          7250      format(1X,'*****',7('*****'))
974:          write(IOT,*) ' * Real variance is estimated by the method',
975:              & ' of Nuc.Sci.Eng., 125,1(1997). *'

```

```

976: C          write(IOT,7260)
977:          7260      format(/1X,' K-eff Estimator',19X,2X,' K-eff ',2X,
978:              &          ' SigA(%) ',1X,' SigR(%) ',1X,'SigR/SigA')
979: C          write(IOT,7280)
980:          7280      format(1X,'-----',7('-----'))
981:          7300      format(1X,A35,2X,1P,E12.5,1X,0P,F9.4,1X,F9.4)
982:          do 750 K = 1, KK
983:              RRR      = SQRT( SIGR(K) / (SIGA(K)+1.0D-20) )
984:              if ( RRR.gt.1.5 ) then
985:                  NWARN = NWARN + 1
986:                  if ( NWARN.eq.1 ) then
987:                      write(IOT,7260)
988:                      write(IOT,7280)
989:                  end if
990:                  write(IOT,7300) ESTKEF(K), AVG(K),
991:                      &          SQRT(SIGA(K) / (AVG(K)+1.0D-20)*100D0,
992:                      &          SQRT(SIGR(K) / (AVG(K)+1.0D-20)*100D0,
993:                      &          SQRT( SIGR(K) / (SIGA(K)+1.0D-20) )
994:              end if
995:          750      continue
996:              if ( NWARN.gt.0 ) then
997:                  write(IOT,7280)
998:                  write(IOT,*) ' * Results for other estimators are ',
999:                      &          'reasonable.'
1000:              else
1001:                  write(IOT,*) ' * Standard deviation given above is ',
1002:                      &          'reasonable.'
1003:              end if
1004:              write(IOT,7250)
1005: C
1006:          end if
1007: C
1008: C      .... OUTPUT TO BINARY FILE (V)      EIGEN VALUES (AFTER EACH BATCH)
1009: C
1010: C
1011: C      -----0-----*-----1-----*-----2-----*-----3---
1012: C      TAG      = 'K-EFF: AFTER EACH BATCH'
1013: C
1014: C
1015:          write(IORS) TAG, (XKEF(1,I),I=1,NBATCH-1),
1016:              &          (XKSD(1,I),I=1,NBATCH-1)
1017:          write(IORS) TAG, (XKEF(2,I),I=1,NBATCH-1),
1018:              &          (XKSD(2,I),I=1,NBATCH-1)
1019:          write(IORS) TAG, (XKEF(3,I),I=1,NBATCH-1),
1020:              &          (XKSD(3,I),I=1,NBATCH-1)
1021:          write(IORS) TAG, (XKEF(4,I),I=1,NBATCH-1),
1022:              &          (XKSD(4,I),I=1,NBATCH-1)
1023: C
1024: C
1025: C      -----0-----*-----1-----*-----2-----*-----3---
1026: C      TAG      = 'K-EFF: MAXIMUM LIKELYHOOD'
1027: C
1028: C
1029:          write(IORS) TAG, (XKMLE(1,I),I=1,NBATCH-1),
1030:              &          (XKERR(1,I),I=1,NBATCH-1)
1031:          write(IORS) TAG, (XKMLE(2,I),I=1,NBATCH-1),
1032:              &          (XKERR(2,I),I=1,NBATCH-1)
1033:          write(IORS) TAG, (XKMLE(3,I),I=1,NBATCH-1),
1034:              &          (XKERR(3,I),I=1,NBATCH-1)
1035:          write(IORS) TAG, (XKMLE(4,I),I=1,NBATCH-1),
1036:              &          (XKERR(4,I),I=1,NBATCH-1)
1037:          write(IORS) TAG, (XKMLE(5,I),I=1,NBATCH-1),
1038:              &          (XKERR(5,I),I=1,NBATCH-1)
1039: C
1040:          end if

```

src/gmvp/tallyo.f

```

1041: C
1042: C
1043: C
1044: call HEADER( IOT, 'TALLY REGION NO. AND INPUT-NAME' )
1045: C
1046: C
1047: write(IOT,7320)
1048: 7320 format(' T-REG. # VOLUME(CC) NAME' /
1049: & ' ----- ',80('-','))
1050:
1051: do 760 KR = 1, NTREG
1052: if ( ITRNM(KR).lt.0 ) then
1053: write(IOT,'(4X,I5,3X,1P,E12.5,2X,A)') KR, TRVOL(KR),
1054: & TNAMS(ABS(ITRNM(KR)))
1055: else
1056: call REGNM0( ITRNM(KR), '>', NAME, LNM, A, CHA )
1057: write(IOT,'(4X,I5,3X,1P,E12.5,2X,A)') KR, TRVOL(KR),
1058: & NAME(:LNM)
1059: end if
1060: 760 continue
1061: C
1062: C-----
1063: C ..... PRINT weighted time of flight .....
1064: C-----
1065: C
1066: if ( JTIME.ne.0 ) then
1067: call LABEL( IOT, 'TIME OF FLIGHT PER HISTORY (SEC)' )
1068: KK = 1
1069: write(IOT,7340) TFLHS(KK,1), TFLHS(KK,2)
1070: 7340 format(/5X,'AVERAGE: ',1P,E13.5,' (STANDARD DEV. ',1P,E10.3,
1071: & ' % )')
1072: end if
1073: C
1074: C ..... PRINT OUT OF FLUX TALLIES (ON I/O UNIT IOF) .....
1075: C
1076: if ( JFPRT.ne.0 ) then
1077: C
1078: C=====
1079: if ( JADJM.eq.0 ) then
1080: call HEADER( IOF,
1081: & 'FLUX BY TRACK LENGTH ESTIMATOR (/SOURCE/LETHARGY/VOLUME)'
1082: & )
1083: else
1084: call HEADER( IOF,
1085: & 'FLUX BY TRACK LENGTH ESTIMATOR (/SOURCE/VOLUME)'
1086: end if
1087: C
1088: C=====
1089: C
1090: do 780 K = 1, NTREG, NL
1091: K1 = MIN(NTREG,K+NL-1)
1092: write(IOF,7360) ' GROUP ', ( ' T-REG.',L,L=K,K1)
1093: 7360 format(/1X,A10,8(A8,I4,1X)/)
1094: *VOCL LOOP,SCALAR
1095: do 770 I = 1, NGROUP
1096: write(IOF,7380) I, (SFLTR(I,L,1),L=K,K1)
1097: 7380 format(1X,I7,3X,1P,8E13.5)
1098: write(IOF,7400) (SFLTR(I,L,2),L=K,K1)
1099: 7400 format(1X,10X,8(' ',F7.3,'%') ':')
1100: 770 continue
1101: 780 continue
1102: C
1103: C
1104: C=====
1105: if ( JADJM.eq.0 ) then

```

```

1106: call HEADER( IOF,
1107: & 'FLUX BY COLLISION ESTIMATOR (/SOURCE/LETHARGY/VOLUME)'
1108: & )
1109: else
1110: call HEADER( IOF,
1111: & 'FLUX BY COLLISION ESTIMATOR (/SOURCE/VOLUME)' )
1112: end if
1113: C
1114: C=====
1115: C
1116: do 800 K = 1, NTREG, NL
1117: K1 = MIN(NTREG,K+NL-1)
1118: write(IOF,7360) ' GROUP ', ( ' T-REG.',L,L=K,K1)
1119: *VOCL LOOP,SCALAR
1120: do 790 I = 1, NGROUP
1121: write(IOF,7380) I, (SFLCL(I,L,1),L=K,K1)
1122: write(IOF,7400) (SFLCL(I,L,2),L=K,K1)
1123: 790 continue
1124: 800 continue
1125: end if
1126: C
1127: C FLUX BY POINT ESTIMATOR ..... 6/13/1991 .....
1128: C
1129: if ( JFPRT.ne.0.and.NPDET.gt.0 ) then
1130: C
1131: C
1132: C=====
1133: if ( JADJM.eq.0 ) then
1134: call HEADER( IOF,
1135: & 'FLUX BY POINT ESTIMATOR (/SOURCE/LETHARGY)' )
1136: else
1137: call HEADER( IOF, 'FLUX BY POINT ESTIMATOR (/SOURCE)' )
1138: end if
1139: C
1140: C=====
1141: C
1142: do 820 K = 1, NPDET, NL
1143: K1 = MIN(NPDET,K+NL-1)
1144: write(IOF,7360) ' GROUP ', ( 'DETECTOR',L,L=K,K1)
1145: *VOCL LOOP,SCALAR
1146: do 810 I = 1, NGROUP
1147: write(IOF,7380) I, (SFLPD(I,L,1),L=K,K1)
1148: write(IOF,7400) (SFLPD(I,L,2),L=K,K1)
1149: 810 continue
1150: 820 continue
1151: end if
1152: C
1153: C
1154: C
1155: if ( JRESP.ne.0 ) then
1156: C
1157: C
1158: C=====
1159: call HEADER( IOT, 'REACTION RATE BY TRACK LENGTH ESTIMATOR' )
1160: C
1161: C=====
1162: C
1163: do 840 K = 1, NRESP, NL
1164: K1 = MIN(NRESP,K+NL-1)
1165: write(IOT,7360) ' T-REGION ', ( ' REACT.',L,L=K,K1)
1166: *VOCL LOOP,SCALAR
1167: do 830 I = 1, NTREG
1168: write(IOT,7380) I, (SRETR(I,L,1),L=K,K1)
1169: write(IOT,7400) (SRETR(I,L,2),L=K,K1)
1170: 830 continue

```

src/gmvp/tallyo.f

```
1171:      840      continue
1172: C
1173: C
1174: C      =====
1175:      call HEADER( IOT, 'REACTION RATE BY COLLISION ESTIMATOR' )
1176: C      =====
1177: C
1178: C
1179:      do 860 K = 1, NRESP, NL
1180:          K1      = MIN(NRESP,K+NL-1)
1181:          write(IOT,7360) ' T-REGION ', ( ' REACT.',L,L=K,K1)
1182: *VOCL LOOP,SCALAR
1183:          do 850 I = 1, NTREG
1184:              write(IOT,7380) I, (SRECL(I,L,1),L=K,K1)
1185:              write(IOT,7400) (SRECL(I,L,2),L=K,K1)
1186:          850      continue
1187:      860      continue
1188: C
1189: C
1190: C
1191:      if ( NPDET.gt.0 ) then
1192: C
1193: C
1194: C      =====
1195:      call HEADER( IOT, 'REACTION RATE BY POINT ESTIMATOR' )
1196: C      =====
1197: C
1198: C
1199:      do 880 K = 1, NRESP, NL
1200:          K1      = MIN(NRESP,K+NL-1)
1201:          write(IOT,7360) 'DETECTOR', ( ' REACT.',L,L=K,K1)
1202: *VOCL LOOP,SCALAR
1203:          do 870 I = 1, NPDET
1204:              write(IOT,7380) I, (SREPD(I,L,1),L=K,K1)
1205:              write(IOT,7400) (SREPD(I,L,2),L=K,K1)
1206:          870      continue
1207:      880      continue
1208:          end if
1209:      end if
1210: C
1211:      return
1212:      end
```

src/gmvp/talsum.f

```

1:      subroutine TALSUM( IOW, A, H, CHA,
2:      &
3:      &
4:      &
5:      &
6:      &
7:      &
8:      &
9:      &
10:     &
11:     &
12:     &
13: C=====
14: C  PURPOSE: TAKE SUMMATION OF TALLY ARRAYS BETWEEN BATCHES.
15: C  CALLED IN: ACTION
16: C=====
17:      include 'INC/ FLAGS'
18:      include '../shared/INC/_TASKDT'
19: C
20: C  ... dynamic size memory array ...
21: C
22:      real A(*)
23:      real H(*)
24:      character*4 CHA(*)
25: C
26:      real*8 WSUMB, WLEK, WSUMB
27:      real RESP(NGROUP,NRESP)
28:      real*8 FLTR(NGROUP,NREG), SFLTR(NGROUP,NTREG,2),
29:      &
30:      &
31:      &
32:      real*8 XSOC(1), XKEF(5,1)
33:      real*8 TFLH(NPKIND), TFLHS(NPKIND,2)
34:      real*8 NCNTR(NEVENT), WCNTR(NEVENT)
35: C ..... SPECIAL TALLY .....
36: C
37:      integer IETAL(*)
38:      integer IDTAL(*)
39:      real*8 ETALY(NLETAL,2)
40:      real*8 DTALY(NLDTAL)
41: C
42: C ..... REGION / TALLY-REGION TABLE .....
43: C
44:      integer IPTRG(NTREG+1), LPTRG(*)
45: C
46: C .... special tallies requiring "real variance" calculation.
47: C
48:      integer IETRV(2,NETRV+1)
49:      real*8 ETRV(METRV,*)
50: C
51: C  ... working area ...
52: C
53:      real*8 FTR(NGROUP,NTREG), FCL(NGROUP,NTREG)
54:      real*8 DWRK(*)
55:      real*8 XKEF1(5,*)
56:      real*8 XSOCB(*)
57:      real*8 XKB(5,*)
58: C
59:      real*8 XKEF3, XKEF4, DFTR, DFCL
60: C
61: C  ... local array ....
62: C
63:      real*8 XKAV1(4), XKER1(4), XKAV2(4), XKER2(4)
64:      real*8 XKAV3(5), XKER3(5)
65: C

```

```

66: C-----
67: C
68:      NBATCH = NBATCH + 1
69: C-----
70: C  ... output monitor array for debug
71: C-----
72:      if ( mod(JDEBG(3)/2,2).eq.1 ) then
73:          write(IOW,7040) (K,NCNTR(K),WCNTR(K),K=1,NEVENT)
74:          format(1x,' *** Event monitor array (count and weight)'/
75:          &
76:          (1x,2(i4,2e16.8)))
77:      end if
78: C-----
79: C  Do something in customized version
80: C-----
81:      call ZZTALS( NGENEO, WSUMB, A, A, H, H, CHA )
82: C
83: C-----
84: C  CLEAR TALLY ARRAYS FOR REJECTED BATCHES
85: C-----
86:
87:      if ( NBATCH.eq.NSKIP ) then
88:          do 110 IR = 1, NREG
89:              do 100 IG = 1, NGROUP
90:                  FLTR(IG,IR) = 0.0
91:                  FLCL(IG,IR) = 0.0
92:              continue
93:          110 continue
94: C
95:          WSUMB = 0.0
96:          if ( JTIME.ne.0 ) then
97:              do 120 K = 1, NPKIND
98:                  TFLH(K) = 0.0D0
99:                  TFLHS(K,1) = 0.0D0
100:                  TFLHS(K,2) = 0.0D0
101:              continue
102:          120
103:          end if
104:          if ( NDTALY.gt.0 ) then
105:              do 130 I = 1, NLDTAL
106:                  DTALY(I) = 0.0D0
107:              continue
108:          130
109:          end if
110:          if ( NETALY.gt.0 ) then
111:              do 150 J = 1, 2
112:                  do 140 I = 1, NLETAL
113:                      ETALY(I,J) = 0.0D0
114:                  continue
115:              140
116:              continue
117:          150
118:          end if
119: C
120:      else if ( NBATCH.gt.NSKIP ) then
121:          C
122:          .... non universe dependent tally ....  NREG = NTREG
123:          C
124:          do 170 KR = 1, NTREG
125:              do 160 IG = 1, NGROUP
126:                  FTR(IG,KR) = 0.0D0
127:                  FCL(IG,KR) = 0.0D0
128:              continue
129:          160
130:          continue
131:          do 200 KR = 1, NTREG
132:              do 190 MR = IPTRG(KR), IPTRG(KR+1) - 1
133:                  IR = LPTRG(MR)
134:                  do 180 IG = 1, NGROUP
135:                      FTR(IG,KR) = FTR(IG,KR) + FLTR(IG,IR)
136:                  180
137:              continue
138:          190
139:          continue
140:      200

```

src/gmvp/talsum.f

```

131:          FCL(IG,KR) = FCL(IG,KR) + FLCL(IG,IR)
132: 180      continue
133: 190      continue
134: 200      continue
135:      do 220 KR = 1, NTREG
136:          do 210 IG = 1, NGROUP
137:              SFLTR(IG,KR,1) = SFLTR(IG,KR,1) + FTR(IG,KR)
138:              SFLCL(IG,KR,1) = SFLCL(IG,KR,1) + FCL(IG,KR)
139:              SFLTR(IG,KR,2) = SFLTR(IG,KR,2) + (FTR(IG,KR)**2) /
140:              &          WSUMB
141:              SFLCL(IG,KR,2) = SFLCL(IG,KR,2) + (FCL(IG,KR)**2) /
142:              &          WSUMB
143: 210      continue
144: 220      continue
145: C
146:      do 240 IR = 1, NREG
147:          do 230 IG = 1, NGROUP
148:              FLTR(IG,IR) = 0.0
149:              FLCL(IG,IR) = 0.0
150: 230      continue
151: 240      continue
152: CC
153:      if ( JRESP.ne.0 ) then
154:          do 270 N = 1, NRESP
155:              do 260 KR = 1, NTREG
156:                  DFTR = 0.0D0
157:                  DFCL = 0.0D0
158:                  do 250 IG = 1, NGROUP
159:                      DFTR = FTR(IG,KR)*RESP(IG,N) + DFTR
160:                      DFCL = FCL(IG,KR)*RESP(IG,N) + DFCL
161: 250              continue
162:                  SRETR(KR,N,1) = SRETR(KR,N,1) + DFTR
163:                  SRETR(KR,N,2) = SRETR(KR,N,2) + DFTR*DFTR/WSUMB
164:                  SRECL(KR,N,1) = SRECL(KR,N,1) + DFCL
165:                  SRECL(KR,N,2) = SRECL(KR,N,2) + DFCL*DFCL/WSUMB
166: 260              continue
167: 270              continue
168:          end if
169: C
170: C
171: C -----
172: C TAKE SPECIAL TALLIES
173: C -----
174: C
175:      if ( NETALY.gt.0 ) then
176:          IBSUM = NBATCH - NSKIP
177:          call STLSUM( IOW, IETAL, NETALY, IDTAL, ETALY, NLETAL,
178:          &          DTALY, NGENEO, WSUMB,
179:          &          IBSUM, NETRV, METRV, IETRV, ETRV,
180:          &          DWRK )
181:      end if
182: C
183:      if ( NDTALY.gt.0 ) then
184:          do 280 I = 1, NLDTAL
185:              DTALY(I) = 0.0D0
186: 280          continue
187:      end if
188: C
189: C -----
190: C weighted flight time
191: C -----
192: C
193:      if ( JTIME.ne.0 ) then
194:          do 290 KKK = 1, NPKIND
195:              TFLHS(KKK,1) = TFLHS(KKK,1) + TFLH(KKK)

```

```

196:          TFLHS(KKK,2) = TFLHS(KKK,2) + TFLH(KKK)**2/WSUMB
197:          TFLH(KKK) = 0.0D0
198: 290      continue
199:      end if
200:      end if
201: C
202:      if ( JEIGN.ne.0 ) then
203:          if ( JPRTS(7).ne.1 ) then
204: C/#IF PARA(SX* CRAY)
205: *          call MVPSYNC_LOCK(2)
206: C/#ENDIF
207:          if ( JBPRNT.ne.0 ) then
208:              write(IOW,7000) NCNTR(14), NCNTR(2), NFISB
209:
210:              XKEF3 = WCNTR(15) / (WCNTR(7)+WCNTR(20))
211:              XKEF4 = WCNTR(14) / (WCNTR(7)+WCNTR(19))
212:
213:              write(IOW,7020) WCNTR(15) /XSOC(NBATCH), WCNTR(14) /
214:              &          XSOC(NBATCH), XKEF3, XKEF4
215:          end if
216: C/#IF PARA(SX* CRAY)
217: *          call MVPSYNC_UNLOCK(2)
218: C/#ENDIF
219:      end if
220: C
221: 7000      format(1X,' FISSION POINT =',F11.0,' FISSION NEUTRONS =',F11.0,
222:      &          ' SELECTED FOR NEXT BATCH =',I7)
223: C
224: 7020      format(1X,'      KEFF(PRODUCTION) =',F6.3,'TRK ',F6.3,'COL',
225:      &          '      KEFF(BALANCE) =',F6.3,'TRK ',F6.3,'COL')
226: C
227:      if ( NBATCH.ge.2 ) then
228:          XKEF(1,NBATCH) = WCNTR(15) + XKEF(1,NBATCH-1)
229:          XKEF(2,NBATCH) = WCNTR(14) + XKEF(2,NBATCH-1)
230:          XKEF(3,NBATCH) = WCNTR(20) + XKEF(3,NBATCH-1)
231:          XKEF(4,NBATCH) = WCNTR(19) + XKEF(4,NBATCH-1)
232:          XKEF(5,NBATCH) = WCNTR(7) + XKEF(5,NBATCH-1)
233:          XSOC(NBATCH) = XSOC(NBATCH) + XSOC(NBATCH-1)
234:      else
235:          XKEF(1,1) = WCNTR(15)
236:          XKEF(2,1) = WCNTR(14)
237:          XKEF(3,1) = WCNTR(20)
238:          XKEF(4,1) = WCNTR(19)
239:          XKEF(5,1) = WCNTR(7)
240:      end if
241:      WLEK = WLEK + WCNTR(7)
242:      NCNTR(2) = 0
243:      NCNTR(14) = 0
244:      NCNTR(15) = 0
245:      NCNTR(19) = 0
246:      NCNTR(20) = 0
247:      WCNTR(2) = 0.0D0
248:      WCNTR(14) = 0.0D0
249:      WCNTR(15) = 0.0D0
250:      WCNTR(19) = 0.0D0
251:      WCNTR(20) = 0.0D0
252:      WCNTR(7) = 0.0D0
253: C
254: C ... fixed source
255: C
256:      else
257:          WLEK = WCNTR(7)
258:      end if
259: C
260: C ... tally monitoring output (k-eff)

```

src/gmvp/talsum.f

```
261: C
262:   if ( NTASK.eq.1.and.JEIGN.ne.0.and.mod(NBATCH,NTMINT).eq.0 ) then
263:     IPBT = NBATCH
264:     JPR = 1
265:     call KEFF0( NBATCH,IPBT, NSKIP, JPR, XSOC, XKEF,
266:       &          XKAV1(1),XKER1(1),XKAV2(1),XKER2(1),XKAV3(1),XKER3(1),
267:       &          XKEF1(1,1), XSOCB(1), XKB(1,1) )
268:   end if
269: C
270: C-----
271: C JBPRNT is set to zero if next batch # (-NSKIP) is not a multiple of
272: C NBPINT.
273: C If JBPRNT is equal to zero, batch monitor of the next batch is not
274: C printed.
275: C-----
276:   if ( NBATCH.ge.NSKIP.and.NBPINT.gt.1 ) then
277:     JBPRNT = 1
278:     if ( MOD(NBATCH+1-NSKIP,NBPINT).ne.1 ) JBPRNT = 0
279:   end if
280: C
281:   return
282: end
```

src/gmvp/verstr.f

```
1:      subroutine VERSTR
2: C=====
3: C Purpose: set version string of printout headers
4: C=====
5: C
6: C   For GMVP ;
7: C
8: C   "GMVP 94.1"   : version 94.1 release 0   ( Nov 1994 )
9: C   "GMVP 94.1a" : version 94.1a ( Apr 1995 )
10: C
11: C=====
12: Cccc CALL HVERSN( 'GMVP 94.1' )
13: Cccc CALL HVERSN( 'GMVP 94.1a' )
14: Cccc call HVERSN( 'GMVP 96.1beta1' )
15: Cccc
16: Cccc 96.1beta3 from 18 May 1997
17: Cccc call HVERSN( 'GMVP 96.1beta3' )
18: Cccc 96.1beta4 from 22 Oct 1997
19: Cccc call HVERSN( 'GMVP 96.1beta4.pre' )
20: Cccc 96.1beta4pre2 from 16 Nov 1997
21: Cccc call HVERSN( 'GMVP 96.1beta4.pre2' )
22: C
23: C   .... beta 4 patch 3 (Mar 3 1998)
24: C   .... beta 4 patch 4 (Mar 23 1998)
25: C   call HVERSN( 'GMVP 96.1beta4p4' )
26: C   .... version 2.0 preliminary (Apr 1998)
27: C   call HVERSN( 'GMVP v2.0pre' )
28: C   .... version 2.0 preliminary1 (14 may 1998)
29: C   call HVERSN( 'GMVP v2.0pre1' )
30: C   .... version 2.0 preliminary1 (9 June 1998)
31: C   call HVERSN( 'GMVP v2.0pre2' )
32: C
33: C   .... version 2.0 beta (22 June 1998)
34: C
35: CCC   call HVERSN( 'GMVP v2.0beta' )
36: C
37: C   .... version 2.0 beta2 ( ? 1998)
38: C
39: CCC   call HVERSN( 'GMVP v2.0beta2-pre' )
40: C
41: C   .... version 2.0 beta2 (Jul 1998)
42: C   call HVERSN( 'GMVP v2.0beta2' )
43: C   .... version 2.0 beta2.5 (25 Aug 1998) by Y.Nagaya
44: C   call HVERSN( 'GMVP v2.0beta2.5' )
45: C
46: C   call HVERSN( 'GMVP v2.0beta3-pre' )
47: C
48: C   ... 9 Sep 1998
49: CCC   call HVERSN( 'GMVP v2.0beta3-pre1' )
50: C
51: C   ... 5 Oct 1998 : beta 3.0
52: C
53: C   call HVERSN( 'GMVP v2.0beta3.0' )
54: C
55: C   ... 14 Oct 1998 : beta 3.01
56: C
57: C   call HVERSN( 'GMVP v2.0beta3.01' )
58: C
59: C   ... 6 Nov 1998 : beta 3.05
60: C
61: C   call HVERSN( 'GMVP v2.0beta3.05' )
62: C
63: C   ... 18 Nov 1998 : beta 3.07 (no-change for GMVP)
64: C
65: C   call HVERSN( 'GMVP v2.0beta3.07' )
66: C
67: C   ... 11 Dec 1998 : beta 3.08 (no-change for GMVP)
68: C   ... 16 Dec 1998 : beta 3.09 (no-change for GMVP)
69: C
70: C   call HVERSN( 'GMVP v2.0beta4.0pre' )
71: C
72: C   ... 11 Jan 1999 : v2.0beta4.0
73: C   call HVERSN( 'GMVP v2.0beta4.0' )
74: C   ... 20 Jan 1999 : v2.0beta4.01
75: C   call HVERSN( 'GMVP v2.0beta4.01' )
76: C   ... 1 Feb 1999 : v2.0beta4.02
77: C   call HVERSN( 'GMVP v2.0beta4.02', '1 February 1999' )
78: C
79: C   call HVERSN( 'GMVP v2.0beta4.03', '10 February 1999' )
80: C
81: C   call HVERSN( 'GMVP v2.0beta4.04', '17 April 1999' )
82: C
83: C   call HVERSN( 'GMVP v2.0beta4.05', '25 April 1999' )
84: C   call HVERSN( 'GMVP v2.0beta4.06', '10 May 1999' )
85: C   call HVERSN( 'GMVP v2.0beta4.07', '17 May 1999' )
86: C   call HVERSN( 'GMVP v2.0beta4.08', '25 May 1999' )
87: C   call HVERSN( 'GMVP v2.0beta4.09', '21 June 1999' )
88: C   call HVERSN( 'GMVP v2.0beta4.10', '7 July 1999' )
89: C   call HVERSN( 'GMVP v2.0beta4.11', '9 August 1999' )
90: C   call HVERSN( 'GMVP v2.0beta4.12', '7 October 1999' )
91: C   call HVERSN( 'GMVP v2.0beta4.13', '22 October 1999' )
92: C   call HVERSN( 'GMVP v2.0beta4.14pre', '22 October 1999' )
93: C   call HVERSN( 'GMVP v2.0beta4.14', '9 November 1999' )
94: C   call HVERSN( 'GMVP v2.0beta4.15pre', '9 November 1999' )
95: C   call HVERSN( 'GMVP v2.0beta4.15', '22 November 1999' )
96: C   call HVERSN( 'GMVP v2.0beta4.16', '24 December 1999' )
97: C   call HVERSN( 'GMVP v2.0beta4.17pre', '24 December 1999' )
98: C   call HVERSN( 'GMVP v2.0beta4.17pre1', '20 February 2000' )
99: C   call HVERSN( 'GMVP v2.0beta4.17', '3 April 2000' )
100: C   call HVERSN( 'GMVP v2.0beta4.18pre', '4 April 2000' )
101: C   call HVERSN( 'GMVP v2.0beta4.18', '30 April 2000' )
102: C   call HVERSN( 'GMVP v2.0beta5', '13 July 2000' )
103: C   call HVERSN( 'GMVP v2.0beta6.00', '18 September 2000' )
104: C   call HVERSN( 'GMVP v2.0beta6.01', '13 October 2000' )
105: C   call HVERSN( 'GMVP v2.0beta6.02', '24 Dec 2000' )
106: C   call HVERSN( 'GMVP v2.0beta6.03', '28 Dec 2000' )
107: C   call HVERSN( 'GMVP v2.0beta6.04', '30 Jan 2001' )
108: C   call HVERSN( 'GMVP v2.0beta6.05', '04 Jun 2001' )
109: C   call HVERSN( 'GMVP v2.0beta6.06', '05 Jul 2001' )
110: C   call HVERSN( 'GMVP v2.0beta6.07', '07 Sep 2001' )
111: C   call HVERSN( 'GMVP v2.0beta6.08', '11 Sep 2001' )
112: C   call HVERSN( 'GMVP v2.0beta6.09', '09 Dec 2001' )
113: C   call HVERSN( 'GMVP v2.0beta6.10', '07 Jan 2002' )
114: C   call HVERSN( 'GMVP v2.0beta6.11', '22 Jan 2002' )
115: C   call HVERSN( 'GMVP v2.0beta6.12', '05 Feb 2002' )
116: C   call HVERSN( 'GMVP v2.0beta6.13', '12 Feb 2002' )
117: C   call HVERSN( 'GMVP v2.0beta6.14', '28 Feb 2002' )
118: C   call HVERSN( 'GMVP v2.0beta6.15', '08 Mar 2002' )
119: C   call HVERSN( 'GMVP v2.0beta6.16', '10 Jun 2002' )
120: C   call HVERSN( 'GMVP v2.0beta6.17', '13 Aug 2002' )
121: C   call HVERSN( 'GMVP v2.0beta6.18', '21 Nov 2002' )
122: C   call HVERSN( 'GMVP v2.0beta6.19', '25 Jul 2003' )
123: C   call HVERSN( 'GMVP v2.0beta6.20', '19 Aug 2003' )
124: C   call HVERSN( 'GMVP v2.0beta6.21', '28 Sep 2003' )
125: C   call HVERSN( 'GMVP v2.0beta6.22', '26 Nov 2003' )
126: C   call HVERSN( 'GMVP v2.0beta6.23', '10 Feb 2004' )
127: C   call HVERSN( 'GMVP v2.0beta6.24', '07 Apr 2004' )
128: C   call HVERSN( 'GMVP v2.0beta6.25', '15 Nov 2004' )
129: C   call HVERSN( 'GMVP v2.0beta6.26', '09 Jan 2005' )
130: C   call HVERSN( 'GMVP v2.0beta6.27', '02 Mar 2005' )
```

src/gmvp/verstr.f

```
131: C      call HVERSN( 'GMVP v2.0beta6.28', '11 Jul 2005' )
132: C      call HVERSN( 'GMVP v2.0beta6.29', '07 Sep 2005' )
133: C      call HVERSN( 'GMVP 2.0', '10 Nov 2005' )
134: C      call HVERSN( 'GMVP 2.0.1', '06 Feb 2006' )
135: C      call HVERSN( 'GMVP 2.0.2', '27 Feb 2006' )
136: C      call HVERSN( 'GMVP 2.0.3', '06 Mar 2006' )
137: C      call HVERSN( 'GMVP 2.0.4', '07 Apr 2006' )
138: C      call HVERSN( 'GMVP 2.0.5', '13 Apr 2006' )
139: C      call HVERSN( 'GMVP 2.0.6', '28 Apr 2006' )
140: C      call HVERSN( 'GMVP 2.0.7', '02 May 2006' )
141: C      call HVERSN( 'GMVP 2.0.8', '15 May 2006' )
142: C      call HVERSN( 'GMVP 2.0.9', '09 Jun 2006' )
143: C      call HVERSN( 'GMVP 2.0.10', '19 Jun 2006' )
144: C      call HVERSN( 'GMVP 2.0.11', '15 Aug 2006' )
145: C      call HVERSN( 'GMVP 2.0.12', '05 Sep 2006' )
146: C      call HVERSN( 'GMVP 2.0.13', '05 Oct 2006' )
147: C      call HVERSN( 'GMVP 2.0.14', '06 Nov 2006' )
148: C      call HVERSN( 'GMVP 2.0.15', '02 Feb 2007' )
149: C      call HVERSN( 'GMVP 2.0.16', '05 Feb 2007' )
150: C      call HVERSN( 'GMVP 2.0.17', '09 Oct 2007' )
151: C      call HVERSN( 'GMVP 2.0.18', '11 Oct 2007' )
152: C      call HVERSN( 'GMVP 2.0.20', '10 Apr 2009' )
153: C      call HVERSN( 'GMVP 2.0.21', '17 Jul 2009' )
154: C      call HVERSN( 'GMVP 2.0.22', '06 Oct 2009' )
155: C      call HVERSN( 'GMVP 2.0.23', '19 Nov 2009' )
156: C      call HVERSN( 'GMVP 2.0.24', '03 Mar 2010' )
157: C      call HVERSN( 'GMVP 2.0.25', '09 Jul 2010' )
158: C      call HVERSN( 'GMVP 2.0.26', '25 Aug 2011' )
159: C      call HVERSN( 'GMVP 2.0.28', '28 Jan 2013' )
160: C      call HVERSN( 'GMVP 2.0.29', '12 Mar 2013' )
161: C      call HVERSN( 'GMVP 2.0.30', '20 Dec 2013' )
162: C      call HVERSN( 'GMVP 2.0.31', '11 Mar 2015' )
163: C      call HVERSN( 'GMVP 2.0.32', '24 Mar 2015' )
164: C      call HVERSN( 'GMVP 2.0.33', '07 Oct 2015' )
165: C      call HVERSN( 'GMVP 2.0.34', '29 Sep 2016' )
166: C      call HVERSN( 'GMVP 2.0.35', '04 Oct 2016' )
167: C      call HVERSN( 'GMVP 2.0.36', '14 Oct 2016' )
168: C      call HVERSN( 'GMVP 2.0.37', '24 Oct 2016' )
169: C      call HVERSN( 'GMVP 2.0.38', '09 Nov 2016' )
170: C      call HVERSN( 'GMVP 2.0.39', '20 Jan 2017' )
171: C      call HVERSN( 'GMVP 3.0', '31 Jan 2017' )
172: C
173:      return
174:      end
```


src/gmvp/vmntr.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine VMNTR0( NS,      IOW )
3: C=====
4: C  PURPOSE:   MONITORING OF CPU & VECTOR CALCULATION.
5: C  CALLED IN: SOURCE,SELECT,FLIGHT,SEARCH,COLISN,MIRROR
6: C  CALLS:     CLOCK
7: C
8: C=====
9:      parameter( NMON = 11 )
10:     character*6 SUBNM, SUBS(NMON)
11:     real CPU(NMON)
12:     integer NEX(NMON), NV(NMON)
13:     real VL(NMON)
14:     real VL2(NMON)
15:     integer MINVL(NMON), MAXVL(NMON)
16:     real SCPU(NMON), SCPU2(NMON)
17:
18: C/#IF SYSTEM(ACOS SX*)
19: *     real*8 T0(NMON), T1(NMON)
20: C/#ELSE
21:     real T0(NMON), T1(NMON)
22: C/#ENDIF
23: C/#IF SYSTEM(DECOSF* PARAGON*)
24: *     external ETIME
25: *     real TARRAY(2)
26: C/#ENDIF
27:
28: C
29: C CPU(NMON)  R4  CPU TIME OF SUBROUTINE
30: C NEX(NMON)  I4  NUMBER OF EXECUTION OF SUBROUTINE
31: C NV (NMON)  I4  TOTAL NUMBER OF VECTOR EVENT
32: C VL (NMON)  R4  SUM OR AVERAGE OF VECTOR LENGTH
33: C VL2(NMON)  R4  SQUARE SUM OR STANDARD DEVIATION OF VECTOR LENGTH
34: C MINVL(NMON) I4  MINIMUM VECTOR LENGTH
35: C MAXVL(NMON) I4  MAXIMUM VECTOR LENGTH
36: C
37: C SCPU(NMON) R4  CPU TIME FOR CPU-MONITOR CALLS      (SEC/CALL)
38: C              ( VMNTR0, VMNTR2 )
39: C SCPU2(NMON) R4  CPU TIME FOR VECTOR LENGTH MONITOR (SEC/CALL)
40: C              ( VMNTR1 )
41: C
42:     data SUBS /'SOURCE', 'SELECT', 'FLIGHT', 'SEARCH', 'COLISN',
43: &             'MIRROR', 'LATTICE', ' ', ' ', ' ', ' ', ' ', 'LFRAME'
44: &             /
45:     data CPU /NMON*0.0/
46:     data NEX /NMON*0/
47:     data NV /NMON*0/
48:     data VL /NMON*0.0/
49:     data VL2 /NMON*0.0/
50:     data MINVL /NMON*999999/
51:     data MAXVL /NMON* - 999/
52:     data T0(1) /0.0/
53:     data T1(1) /0.0/
54:     data SCPU /NMON*0.0/
55:     data SCPU2 /NMON*0.0/
56: C
57: C/#IF SYSTEM(FACOM*)
58: *     call CLOCK(T0(NS),0,1)
59: C/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
60: C     call cputime(t0(ns))
61: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
62: *     T0(NS) = ETIME( TARRAY )
63: C/#ELSEIF SYSTEM(ACOS SX*)
64: *     call CLOCK(T0(NS))
65: C/#ELSEIF SYSTEM(CRAY*)

```

```

66: *     call SECOND(T0(NS))
67: C/#ELSEIF SYSTEM(IBMRS* AIX*)
68: *     T0(NS) = MCLOCK()/100.0
69: C/#ELSE
70: C     T0(NS) = 0.0
71: *     call CPUTM(T0(NS))
72: C/#ENDIF
73:
74:     NEX(NS) = NEX(NS) + 1
75:     return
76: C
77: C ..... VECTOR LENGTH MONITORING .....
78: C
79:     entry VMNTR1(NS,NN)
80:     NV(NS) = NV(NS) + 1
81:     VL(NS) = VL(NS) + NN
82:     VL2(NS) = VL2(NS) + NN*NN
83:     MINVL(NS) = MIN(MINVL(NS),NN)
84:     MAXVL(NS) = MAX(MAXVL(NS),NN)
85:     return
86: C
87: C ..... CPU MONITORING .....
88: C
89:     entry VMNTR2(NS)
90: C/#IF SYSTEM(FACOM*)
91: *     call CLOCK(T1(NS),0,1)
92: C/#ELSEIF SYSTEM(HP* SUN* MIPS* NECEWS* SGI*)
93: C     call cputime(t1(ns))
94: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
95: *     T1(NS) = ETIME( TARRAY )
96: C/#ELSEIF SYSTEM(ACOS SX*)
97: *     call CLOCK(T1(NS))
98: C/#ELSEIF SYSTEM(CRAY*)
99: *     call SECOND(T1(NS))
100: C/#ELSEIF SYSTEM(IBMRS* AIX*)
101: *     T1(NS) = MCLOCK()/100.0
102: C/#ELSE
103: *     call CPUTM(T1(NS))
104: C/#ENDIF
105:     CPU(NS) = CPU(NS) + T1(NS) - T0(NS)
106:     return
107: C
108: C ..... PRINT OUT MONITORING INFORMATION .....
109: C
110:     entry VMNTRF(IOW)
111:     write(IOW,7000)
112: 7000 format('1',' CPU & VECTOR OPERATION MONITOR  '//
113: &           ' (CPU TIME FOR MONITOR ROUTINE CALLS IS REMOVED) '//1X,
114: &           7X,
115: &           ' CPU(SEC)      CALL CPU/CALL  VECTOR      AVE. VECTOR ',
116: &           ' STANDARD  MAX. VECTOR MIN. VECTOR  '//1X,7X,
117: &           ' (SEC)      EVENT      LENGTH      ',
118: &           ' DEVIATION  LENGTH      LENGTH      ' //)
119: C
120:     TCPU      = 0.0
121:     NMON2     = 7
122: *VOCL LOOP,SCALAR
123:     do 100 N = 1, NMON2
124:         CPU(N) = CPU(N) - SCPU(N)*NEX(N) - SCPU2(N)*NV(N)
125:         TCPU   = TCPU + CPU(N)
126:         if ( NV(N).gt.0 ) VL(N) = VL(N) /NV(N)
127:         if ( NV(N).gt.1 ) VL2(N) =
128: &         Sqrt(ABS((VL2(N)-NV(N)*VL(N)**2))/(NV(N)-1))
129:         if ( NV(N).eq.1 ) VL2(N) = 0.0
130:         CN      = 0.0

```

src/gmvp/vmntr.f

```

131:         if ( NV(N).eq.0 ) then
132:             MAXVL(N) = 0
133:             MINVL(N) = 0
134:         end if
135:         if ( NEX(N).ne.0 ) CN = CPU(N) /NEX(N)
136:         write(IOW,7020) SUBS(N), CPU(N), NEX(N), CN, NV(N), VL(N),
137:         &             VL2(N), MAXVL(N), MINVL(N)
138: 7020    format(1X,A6,1X,1PE11.4,I7,1X,1PE11.4,I8,3X,2E11.4,4X,I6,7X,I6)
139: 100 continue
140:     write(IOW,'(/1X,A,I3,A,E12.5,A/)' ) ' === CPU TOTAL FOR THESE ',
141:     &             NMON2, ' ROUTINES :', TCPU, ' SEC'
142: CCCCC
143: C
144: C
145:     TCPU2 = 0.0
146:     do 110 N = NMON2 + 1, NMON
147:         CPU(N) = CPU(N) - SCPU(N)*NEX(N) - SCPU2(N)*NV(N)
148:         TCPU2 = TCPU2 + CPU(N)
149:         if ( NV(N).gt.0 ) VL(N) = VL(N) /NV(N)
150:         if ( NV(N).gt.1 ) VL2(N) =
151:         &             Sqrt(ABS((VL2(N)-NV(N)*VL(N)**2))/(NV(N)-1))
152:         if ( NV(N).eq.1 ) VL2(N) = 0.0
153:         CN = 0.0
154:         if ( NV(N).eq.0 ) then
155:             MAXVL(N) = 0
156:             MINVL(N) = 0
157:         end if
158:         if ( NEX(N).ne.0 ) CN = CPU(N) /NEX(N)
159:         write(IOW,7020) SUBS(N), CPU(N), NEX(N), CN, NV(N), VL(N),
160:         &             VL2(N), MAXVL(N), MINVL(N)
161: 110 continue
162:     write(IOW,'(/1X,A,I3,A,E12.5,A/)' )
163:     &             ' === CPU TOTAL FOR THE LAST ', NMON - NMON2,
164:     &             ' ROUTINES :', TCPU2, ' SEC'
165:     return
166: C
167: C ***** CLEAR ALL MONITOR ARRAYS *****
168: C
169:     entry VMNTRC
170:     do 120 I = 1, NMON
171:         CPU(I) = 0.0
172:         NEX(I) = 0
173:         NV(I) = 0
174:         VL(I) = 0.0
175:         VL2(I) = 0.0
176:         MINVL(I) = 99999999
177:         MAXVL(I) = 0
178: 120 continue
179:     return
180: C
181: C ***** give SCPU a value *****
182: C
183:     entry VMNTRG(TT,TT2,NS)
184:     SCPU(NS) = TT
185:     SCPU2(NS) = TT2
186:     return
187: C
188: C ***** get value from SCPU *****
189: C
190:     entry VMNTRT(TT,NS)
191:     TT = CPU(NS)
192:     return
193: end

```

src/gmvp/wrkary.f

```

1:      subroutine WRKARY( A,      H,      LIMIT, LIMITL,MAXSF, MWVEC,
2:      &                  MVSTK, LLXYZ, LLPWRK,LLVSTK,LXYZ,  LPWRK,
3:      &                  LVSTK, MAXWK )
4: C=====
5: C  PURPOSE : CALCULATE WORKIG-ARRAY-POINTERS AND SIZE OF MEMORY
6: C          FOR EACH EVENT-PROCESSING ROUTINE.
7: C  CALLED IN : INTRO
8: C  CALLS   : KEEP
9: C=====
10:     real A(*)
11:     real H(*)
12: Ccccc integer ia(*)
13: C
14:     integer LXYZ(10), LPWRK(MWVEC), MVSTK(MVSTK)
15: C
16: CCCC include '../shared/INC/_ARRAY'
17: include 'INC/_FLAGS'
18: include '../shared/INC/_SIZES'
19: include 'INC/_SIZES2'
20: include '../shared/INC/_PTALY0'
21: include 'INC/_PTALY'
22: include '../shared/INC/_STALY'
23: C
24: include 'INC/_WKSOU'
25: include 'INC/_WKFSS'
26: include 'INC/_WKFL1'
27: include 'INC/_WKSE1'
28: include 'INC/_WKCOL'
29: include 'INC/_WKMIR'
30: include 'INC/_WKLAT'
31: include 'INC/_WKFLA'
32: include 'INC/_WKSEA'
33: include 'INC/_WKTLS'
34: include 'INC/_WKTLO'
35: C
36: include 'INC/_WKC2A'
37: C
38: include 'INC/_WKNXT'
39: include '../shared/INC/_IOUNIT'
40: C
41: -----
42: C
43: C ... keep a dummy data for memory violation check if necessary
44: C and for memory boudary adjustment.
45: C
46: call LKEPV(H(1), 'WDUMY', LWK, 1, 'R8', LAST, 1.234D-12, JDEBG )
47: C
48: call LGTLST( LWOR )
49: C
50: write(IPR,7000) LWOR
51: 7000 format(/1X,' === Working area starts at ',I12,' 'th element',
52: &          ' of task local data array. ===')
53: C
54: C .... WORKING ARRAY OVERFLOW COUNTER ....
55: C
56:     IOVFL   = 0
57:     MAXWK   = 0
58: C
59: C
60: C ===== FOR SOURCE =====
61: C
62: call LSTLST( 'SOURCE', LWOR, LASTP )
63: call LKEEP( 'R      ', P1(1), 6*NBANK, 'R4D', LASTP, JDEBG )
64: call LKEEP( 'IWK1  ', P1(2), NBANK, 'I4', LASTP, JDEBG )
65: call LKEEP( 'IWK2  ', P1(3), NBANK, 'I4', LASTP, JDEBG )

```

```

66: call LKEEP( 'RWK1  ', P1(4), NBANK, 'R4', LASTP, JDEBG )
67: call LKEEP( 'X      ', P1(5), NBANK, 'R8', LASTP, JDEBG )
68: call LKEEP( 'Y      ', P1(6), NBANK, 'R8', LASTP, JDEBG )
69: call LKEEP( 'Z      ', P1(7), NBANK, 'R8', LASTP, JDEBG )
70: call LKEEP( 'IFL    ', P1(8), NBANK, 'I4D', LASTP, JDEBG )
71: call LKEEP( 'IFI    ', P1(9), NBANK, 'I4', LASTP, JDEBG )
72: call LKEEP( 'NNSRC  ', P1(10), NSOUR, 'I4', LASTP, JDEBG )
73: call LKEEP( 'XSOUR  ', P1(11), NSOUR, 'R8', LASTP, JDEBG )
74: call LKEEP( 'RWF    ', P1(12), NHIST, 'R4', LASTP, JDEBG )
75: C
76: C
77: C ... allocation of pointer array itself might have failed ...
78: C
79:     NXYZ    = 10
80:     if ( LLXYZ.ne.0 ) then
81:         do 100 I = 1, NXYZ
82:             call LKEEP( 'XYZ ', LXYZ(I), NBANK, 'R8', LASTP, JDEBG )
83: 100    continue
84:     else
85:         do 110 I = 1, NXYZ
86:             call LKEEP( 'XYZ ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
87: 110    continue
88:     end if
89: C
90:     if ( LLPWRK.ne.0 ) then
91:         do 120 I = 1, MWVEC
92:             call LKEEP( 'PWRK ', LPWRK(I), NBANK, 'R8', LASTP, JDEBG )
93: 120    continue
94:     else
95:         do 130 I = 1, MWVEC
96:             call LKEEP( 'PWRK ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
97: 130    continue
98:     end if
99: C
100:    if ( LLVSTK.ne.0 ) then
101:        do 140 I = 1, MVSTK
102:            call LKEEP( 'VSTK ', LVSTK(I), NBANK, 'R8', LASTP, JDEBG )
103: 140    continue
104:    else
105:        do 150 I = 1, MVSTK
106:            call LKEEP( 'VSTK ', LLLLLL, NBANK, 'R8', LASTP, JDEBG )
107: 150    continue
108:    end if
109: C
110: C if ( JPTDT.ne.0 ) then
111: C call LKEEP( 'AA      ', P1(13), NBANK, 'R8', LASTP, JDEBG )
112: C call LKEEP( 'BB      ', P1(14), NBANK, 'R8', LASTP, JDEBG )
113: C call LKEEP( 'CC      ', P1(15), NBANK, 'R8', LASTP, JDEBG )
114: C call LKEEP( 'DI      ', P1(16), NBANK, 'R8', LASTP, JDEBG )
115: C
116: C .... FOR NXTEE ....
117: C
118: call LSTLST( 'SOURCE', LASTP, LASTPS )
119: C
120: C
121: C if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
122: C MAXWK = MAX(MAXWK,LASTP)
123: C write(IPR,7020) 'SOURCE', LASTP - LWOR, LASTP
124: C
125: C ===== For FISSET =====
126: C
127: C if ( JEIGN.ne.0 ) then
128: C call LSTLST( 'FISSET', LWOR, LASTP )
129: C
130: call LKEEP( 'RWF    ', P10(1), NHIST, 'R4', LASTP, JDEBG )

```

src/gmvp/wrkary.f

```

131:
132:     if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
133:     MAXWK = MAX(MAXWK,LASTP)
134:     write(IPR,7020) 'FISSET', LASTP - LWORK, LASTP
135:     end if
136: C
137: C ===== FOR FLIONE =====
138: C
139:     if ( JONEZ.eq.1 ) then
140: C
141:         call LSTLST( 'FLIONE', LWORK, LASTP )
142:         call LKEEP( 'X', P0(1), NBANK, 'R8', LASTP, JDEBG )
143:         call LKEEP( 'Y', P0(2), NBANK, 'R8', LASTP, JDEBG )
144:         call LKEEP( 'Z', P0(3), NBANK, 'R8', LASTP, JDEBG )
145:         call LKEEP( 'AI', P0(4), NBANK, 'R8', LASTP, JDEBG )
146:         call LKEEP( 'BI', P0(5), NBANK, 'R8', LASTP, JDEBG )
147:         call LKEEP( 'CI', P0(6), NBANK, 'R8', LASTP, JDEBG )
148:         call LKEEP( 'W', P0(7), NBANK, 'R8', LASTP, JDEBG )
149:         call LKEEP( 'DI', P0(8), NBANK, 'R8', LASTP, JDEBG )
150:         call LKEEP( 'IGI', P0(9), NBANK, 'I4', LASTP, JDEBG )
151:         call LKEEP( 'IBP', P0(10), NBANK, 'I4', LASTP, JDEBG )
152:         call LKEEP( 'IFC', P0(11), NBANK, 'I4', LASTP, JDEBG )
153:         call LKEEP( 'R', P0(12), 3*NBANK, 'R4', LASTP, JDEBG )
154:         call LKEEP( 'DLOC1', P0(13), NBANK, 'R8', LASTP, JDEBG )
155:         call LKEEP( 'DLOC2', P0(14), NBANK, 'R8', LASTP, JDEBG )
156:         call LKEEP( 'IKL', P0(15), NBANK, 'I4', LASTP, JDEBG )
157:         call LKEEP( 'IKL2', P0(16), NBANK, 'I4', LASTP, JDEBG )
158:         if ( JHLAT.ne.0 .or. JFISX.ne.0 .or. JTSRF.ne.0 ) then
159:             call LKEEP( 'XUP', P0(17), NBANK, 'R8', LASTP, JDEBG )
160:             call LKEEP( 'YUP', P0(18), NBANK, 'R8', LASTP, JDEBG )
161:             call LKEEP( 'ZUP', P0(19), NBANK, 'R8', LASTP, JDEBG )
162:             call LKEEP( 'AUP', P0(20), NBANK, 'R8', LASTP, JDEBG )
163:             call LKEEP( 'BUP', P0(21), NBANK, 'R8', LASTP, JDEBG )
164:             call LKEEP( 'CUP', P0(22), NBANK, 'R8', LASTP, JDEBG )
165:         end if
166:         call LKEEP( 'IRG', P0(23), NBANK, 'I4', LASTP, JDEBG )
167: CCCC
168:         if ( JTIME.ne.0 ) then
169:             call LKEEP( 'TI', P0(24), NBANK, 'R8', LASTP, JDEBG )
170:         end if
171:         call LKEEP( 'ISTG', P0(25), NBANK, 'I4', LASTP, JDEBG )
172:         call LKEEP( 'ILUP', P0(26), NBANK, 'I4', LASTP, JDEBG )
173:         call LKEEP( 'ILST', P0(27), NBANK, 'I4', LASTP, JDEBG )
174:         if ( JFISX.ne.0 .or. JTSRF.ne.0 ) then
175:             call LKEEP( 'XYZ1', P0(28), 3*NBANK, 'R8', LASTP, JDEBG )
176:             call LKEEP( 'ABC1', P0(29), 3*NBANK, 'R8', LASTP, JDEBG )
177:             call LKEEP( 'XYZ2', P0(30), 3*NBANK, 'R8', LASTP, JDEBG )
178:             call LKEEP( 'ABC2', P0(31), 3*NBANK, 'R8', LASTP, JDEBG )
179:             call LKEEP( 'DDI', P0(32), NBANK, 'R8', LASTP, JDEBG )
180:             call LKEEP( 'IBP0', P0(33), NBANK, 'I4', LASTP, JDEBG )
181:             call LKEEP( 'IBP1', P0(34), NBANK, 'I4', LASTP, JDEBG )
182:             call LKEEP( 'IBP2', P0(35), NBANK, 'I4', LASTP, JDEBG )
183:             call LKEEP( 'KPLT', P0(36), NLBZ+1, 'I4', LASTP, JDEBG )
184:             call LKEEP( 'JKSF', P0(37), NLBZ, 'I4', LASTP, JDEBG )
185:         end if
186:         if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
187:         MAXWK = MAX(MAXWK,LASTP)
188:         write(IPR,7020) 'FLIONE', LASTP - LWORK, LASTP
189:     end if
190: C
191: C ===== FOR SEAONE =====
192: C
193:     if ( JONEZ.eq.1 ) then
194: C
195:         call LSTLST( 'SEAONE', LWORK, LASTP )

```

```

196:         call LKEEP( 'X', P2(1), NBANK, 'R8', LASTP, JDEBG )
197:         call LKEEP( 'Y', P2(2), NBANK, 'R8', LASTP, JDEBG )
198:         call LKEEP( 'Z', P2(3), NBANK, 'R8', LASTP, JDEBG )
199:         call LKEEP( 'W', P2(4), NBANK, 'R8', LASTP, JDEBG )
200:         call LKEEP( 'IBP', P2(5), NBANK, 'I4', LASTP, JDEBG )
201:         call LKEEP( 'IFI', P2(6), NBANK, 'I4', LASTP, JDEBG )
202:         call LKEEP( 'IZI', P2(7), NBANK, 'I4', LASTP, JDEBG )
203:         call LKEEP( 'IWK', P2(8), NBANK, 'I4', LASTP, JDEBG )
204:         call LKEEP( 'IFL', P2(9), NBANK, 'I4', LASTP, JDEBG )
205:         call LKEEP( 'R', P2(10), NBANK, 'R4', LASTP, JDEBG )
206:         call LKEEP( 'ISRF', P2(11), NBANK, 'I4', LASTP, JDEBG )
207:         call LKEEP( 'FXYZ', P2(12), NBANK, 'R4', LASTP, JDEBG )
208:         if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
209:         MAXWK = MAX(MAXWK,LASTP)
210:         write(IPR,7020) 'SEAONE', LASTP - LWORK, LASTP
211: C
212:     end if
213: C
214: C ===== FOR COLISN =====
215: C
216:         call LSTLST( 'COLISN', LWORK, LASTP )
217:         call LKEEP( 'IWK1', P3(1), NBANK, 'I4', LASTP, JDEBG )
218:         call LKEEP( 'IWK2', P3(2), NBANK, 'I4', LASTP, JDEBG )
219:         call LKEEP( 'IWK3', P3(3), NBANK, 'I4', LASTP, JDEBG )
220:         call LKEEP( 'IWK4', P3(4), NBANK, 'I4', LASTP, JDEBG )
221:         call LKEEP( 'IWK5', P3(5), NBANK, 'I4', LASTP, JDEBG )
222:         call LKEEP( 'IWK6', P3(6), NBANK, 'I4', LASTP, JDEBG )
223:         call LKEEP( 'IWK7', P3(7), NBANK, 'I4', LASTP, JDEBG )
224:         call LKEEP( 'IWK8', P3(8), NBANK, 'I4', LASTP, JDEBG )
225:         call LKEEP( 'IWK9', P3(9), NBANK, 'I4', LASTP, JDEBG )
226:         call LKEEP( 'RP', P3(10), NBANK, 'R4', LASTP, JDEBG )
227:         call LKEEP( 'R', P3(11), 4*NBANK, 'R4', LASTP, JDEBG )
228: C
229:         'IWK'
230:         P3(12) = P3(11)
231:         call LKEEP( 'DWK1', P3(13), NBANK, 'R8', LASTP, JDEBG )
232:         call LKEEP( 'DWK2', P3(14), NBANK, 'R8', LASTP, JDEBG )
233: C
234:         if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
235:         MAXWK = MAX(MAXWK,LASTP)
236:         write(IPR,7020) 'COLISN', LASTP - LWORK, LASTP
237: C
238: C ===== FOR MIRROR =====
239: C
240:         if ( JREFL.ne.0 ) then
241:             call LSTLST( 'MIRROR', LWORK, LASTP )
242:             call LKEEP( 'KKSFl', P4(1), NREFS+1, 'I4', LASTP, JDEBG )
243:             call LKEEP( 'KKSf2', P4(2), NREFS+1, 'I4', LASTP, JDEBG )
244:             call LKEEP( 'JKSF', P4(3), NREFS+1, 'I4', LASTP, JDEBG )
245:             call LKEEP( 'IPSURF', P4(4), NREFS+1, 'I4', LASTP, JDEBG )
246:             call LKEEP( 'X', P4(5), NBANK, 'R8', LASTP, JDEBG )
247:             call LKEEP( 'Y', P4(6), NBANK, 'R8', LASTP, JDEBG )
248:             call LKEEP( 'Z', P4(7), NBANK, 'R8', LASTP, JDEBG )
249:             call LKEEP( 'AI', P4(8), NBANK, 'R8', LASTP, JDEBG )
250:             call LKEEP( 'BI', P4(9), NBANK, 'R8', LASTP, JDEBG )
251:             call LKEEP( 'CI', P4(10), NBANK, 'R8', LASTP, JDEBG )
252:             call LKEEP( 'R', P4(11), 2*NBANK, 'R4', LASTP, JDEBG )
253:             P4(12) = P4(11)
254:             call LKEEP( 'IBP', P4(13), NBANK, 'I4', LASTP, JDEBG )
255:             call LKEEP( 'IZT', P4(14), NBANK, 'I4', LASTP, JDEBG )
256:             if ( MAXSF.ge.1 )
257:                 & call LKEEP( 'DSDA0', P4(15), NBANK, 'R8', LASTP, JDEBG )
258:             if ( MAXSF.ge.2 )
259:                 & call LKEEP( 'DSDA1', P4(16), NBANK, 'R8', LASTP, JDEBG )
260:             if ( MAXSF.ge.3 )
261:                 & call LKEEP( 'DSDA2', P4(17), NBANK, 'R8', LASTP, JDEBG )

```

src/gmvp/wrkary.f

```

261:      if ( MAXSF.ge.4 )
262:      &    call LKEEP( 'DSDA3 ', P4(18), NBANK, 'R8', LASTP, JDEBG )
263:      if ( MAXSF.ge.5 )
264:      &    call LKEEP( 'DSDA4 ', P4(19), NBANK, 'R8', LASTP, JDEBG )
265:      if ( MAXSF.ge.6 )
266:      &    call LKEEP( 'DSDA5 ', P4(20), NBANK, 'R8', LASTP, JDEBG )
267:      if ( MAXSF.ge.7 )
268:      &    call LKEEP( 'DSDA6 ', P4(21), NBANK, 'R8', LASTP, JDEBG )
269:      if ( MAXSF.ge.8 )
270:      &    call LKEEP( 'DSDA7 ', P4(22), NBANK, 'R8', LASTP, JDEBG )
271:      if ( MAXSF.ge.9 )
272:      &    call LKEEP( 'DSDA8 ', P4(23), NBANK, 'R8', LASTP, JDEBG )
273:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
274:      MAXWK = MAX(MAXWK,LASTP)
275:      write(IPR,7020) 'MIRROR', LASTP - LWORK, LASTP
276:      end if
277: C
278: C ===== FOR LATTICE =====
279: C
280:      if ( JLAT.ne.0 ) then
281:      call LSTLST( 'LATTICE', LWORK, LASTP )
282:      call LKEEP( 'X ', P5(1), NBANK, 'R8', LASTP, JDEBG )
283:      call LKEEP( 'Y ', P5(2), NBANK, 'R8', LASTP, JDEBG )
284:      call LKEEP( 'Z ', P5(3), NBANK, 'R8', LASTP, JDEBG )
285:      call LKEEP( 'AI ', P5(4), NBANK, 'R8', LASTP, JDEBG )
286:      call LKEEP( 'BI ', P5(5), NBANK, 'R8', LASTP, JDEBG )
287:      call LKEEP( 'CI ', P5(6), NBANK, 'R8', LASTP, JDEBG )
288:      call LKEEP( 'IWK ', P5(7), NBANK, 'I4', LASTP, JDEBG )
289:      call LKEEP( 'JKSF ', P5(8), NZONE+1, 'I4', LASTP, JDEBG )
290:      call LKEEP( 'IPZONE', P5(9), NZONE+1, 'I4', LASTP, JDEBG )
291:      call LKEEP( 'MEM ', P5(10), NZONE+1, 'I4', LASTP, JDEBG )
292:      if ( JFISX.ne.0 .or. JSTGR.ne.0 ) then
293:      call LKEEP( 'DI ', P5(11), NBANK, 'R8', LASTP, JDEBG )
294:      call LKEEP( 'DLOC1 ', P5(12), NBANK, 'R8', LASTP, JDEBG )
295:      call LKEEP( 'DLOC2 ', P5(13), NBANK, 'R8', LASTP, JDEBG )
296:      call LKEEP( 'IKL ', P5(14), NBANK, 'I4', LASTP, JDEBG )
297:      call LKEEP( 'IKL2 ', P5(15), NBANK, 'I4', LASTP, JDEBG )
298:      call LKEEP( 'R ', P5(16), 3*NBANK, 'I4', LASTP, JDEBG )
299:      end if
300:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
301:      MAXWK = MAX(MAXWK,LASTP)
302:      write(IPR,7020) 'LATTICE', LASTP - LWORK, LASTP
303:      end if
304: C
305: C
306:      if ( JALLZ.eq.1 ) then
307: C
308: C ===== FOR FLIGHT =====
309: C
310:      call LSTLST( 'FLIGHT', LWORK, LASTP )
311:      call LKEEP( 'KKSF ', P6(1), NZONE+1, 'I4', LASTP, JDEBG )
312:      call LKEEP( 'JKSF ', P6(2), NZONE+1, 'I4', LASTP, JDEBG )
313:      call LKEEP( 'IPZONE', P6(3), NZONE+1, 'I4', LASTP, JDEBG )
314:      call LKEEP( 'X ', P6(4), NBANK, 'R8', LASTP, JDEBG )
315:      call LKEEP( 'Y ', P6(5), NBANK, 'R8', LASTP, JDEBG )
316:      call LKEEP( 'Z ', P6(6), NBANK, 'R8', LASTP, JDEBG )
317:      call LKEEP( 'AI ', P6(7), NBANK, 'R8', LASTP, JDEBG )
318:      call LKEEP( 'BI ', P6(8), NBANK, 'R8', LASTP, JDEBG )
319:      call LKEEP( 'CI ', P6(9), NBANK, 'R8', LASTP, JDEBG )
320:      call LKEEP( 'W ', P6(10), NBANK, 'R8', LASTP, JDEBG )
321:      call LKEEP( 'DI ', P6(11), NBANK, 'R8', LASTP, JDEBG )
322:      call LKEEP( 'IGI ', P6(12), NBANK, 'I4', LASTP, JDEBG )
323:      call LKEEP( 'IBP ', P6(13), NBANK, 'I4', LASTP, JDEBG )
324:      call LKEEP( 'R ', P6(14), NBANK, 'R4', LASTP, JDEBG )
325:      call LKEEP( 'D11 ', P6(15), NBANK, 'R8', LASTP, JDEBG )

```

```

326:      call LKEEP( 'D22 ', P6(16), NBANK, 'R8', LASTP, JDEBG )
327:      call LKEEP( 'DLOC1 ', P6(17), NBANK, 'R8', LASTP, JDEBG )
328:      call LKEEP( 'DLOC2 ', P6(18), NBANK, 'R8', LASTP, JDEBG )
329:      call LKEEP( 'LLS ', P6(19), NBANK, 'I4', LASTP, JDEBG )
330:      call LKEEP( 'IZT ', P6(20), NBANK, 'I4', LASTP, JDEBG )
331:      call LKEEP( 'DSDA0 ', P6(21), NBANK, 'R8', LASTP, JDEBG )
332:      call LKEEP( 'DSDA1 ', P6(22), NBANK, 'R8', LASTP, JDEBG )
333:      call LKEEP( 'DSDA2 ', P6(23), NBANK, 'R8', LASTP, JDEBG )
334:      call LKEEP( 'DSDA3 ', P6(24), NBANK, 'R8', LASTP, JDEBG )
335:      call LKEEP( 'DSDA4 ', P6(25), NBANK, 'R8', LASTP, JDEBG )
336:      call LKEEP( 'DSDA5 ', P6(26), NBANK, 'R8', LASTP, JDEBG )
337:      if ( MAXSF.ge.7 )
338:      &    call LKEEP( 'DSDA6 ', P6(27), NBANK, 'R8', LASTP, JDEBG )
339:      if ( MAXSF.ge.8 )
340:      &    call LKEEP( 'DSDA7 ', P6(28), NBANK, 'R8', LASTP, JDEBG )
341:      if ( MAXSF.ge.9 )
342:      &    call LKEEP( 'DSDA8 ', P6(29), NBANK, 'R8', LASTP, JDEBG )
343:      if ( MAXSF.ge.10 )
344:      &    call LKEEP( 'DSDA9 ', P6(30), NBANK, 'R8', LASTP, JDEBG )
345:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
346:      MAXWK = MAX(MAXWK,LASTP)
347:      write(IPR,7020) 'FLIGHT', LASTP - LWORK, LASTP
348: C
349: C ===== FOR SEARCH =====
350: C
351:      call LSTLST( 'SEARCH', LWORK, LASTP )
352:      call LKEEP( 'IZT ', P7(1), NZONE+1, 'I4', LASTP, JDEBG )
353:      call LKEEP( 'IPZONE', P7(2), NZONE+1, 'I4', LASTP, JDEBG )
354:      call LKEEP( 'MEM ', P7(3), NZONE+1, 'I4', LASTP, JDEBG )
355:      call LKEEP( 'KKSF ', P7(4), NZONE+1, 'I4', LASTP, JDEBG )
356:      call LKEEP( 'IWK2 ', P7(5), NZONE+1, 'I4', LASTP, JDEBG )
357:      call LKEEP( 'IWK3 ', P7(6), NZONE+1, 'I4', LASTP, JDEBG )
358:      call LKEEP( 'IWK4 ', P7(7), NZONE+1, 'I4', LASTP, JDEBG )
359:      call LKEEP( 'X ', P7(8), NBANK, 'R8', LASTP, JDEBG )
360:      call LKEEP( 'Y ', P7(9), NBANK, 'R8', LASTP, JDEBG )
361:      call LKEEP( 'Z ', P7(10), NBANK, 'R8', LASTP, JDEBG )
362:      call LKEEP( 'W ', P7(11), NBANK, 'R8', LASTP, JDEBG )
363:      call LKEEP( 'IWK ', P7(12), NBANK, 'I4', LASTP, JDEBG )
364:      call LKEEP( 'IFG ', P7(13), NBANK, 'I4', LASTP, JDEBG )
365:      call LKEEP( 'LLS ', P7(14), NBANK, 'I4', LASTP, JDEBG )
366:      call LKEEP( 'IWK0 ', P7(15), NBANK, 'I4', LASTP, JDEBG )
367:      call LKEEP( 'IFI ', P7(16), NBANK, 'I4', LASTP, JDEBG )
368:      call LKEEP( 'IFL ', P7(17), NBANK, 'I4', LASTP, JDEBG )
369:      call LKEEP( 'ISRF ', P7(18), NBANK, 'I4', LASTP, JDEBG )
370:      call LKEEP( 'FXYZ ', P7(19), NBANK, 'R4', LASTP, JDEBG )
371:      call LKEEP( 'R ', P7(20), NBANK, 'R4', LASTP, JDEBG )
372:      call LKEEP( 'DSDA0 ', P7(21), NBANK, 'R8', LASTP, JDEBG )
373:      call LKEEP( 'DSDA1 ', P7(22), NBANK, 'R8', LASTP, JDEBG )
374:      call LKEEP( 'DSDA2 ', P7(23), NBANK, 'R8', LASTP, JDEBG )
375:      call LKEEP( 'DSDA3 ', P7(24), NBANK, 'R8', LASTP, JDEBG )
376:      call LKEEP( 'DSDA4 ', P7(25), NBANK, 'R8', LASTP, JDEBG )
377:      call LKEEP( 'DSDA5 ', P7(26), NBANK, 'R8', LASTP, JDEBG )
378:      if ( MAXSF.ge.7 )
379:      &    call LKEEP( 'DSDA6 ', P7(27), NBANK, 'R8', LASTP, JDEBG )
380:      if ( MAXSF.ge.8 )
381:      &    call LKEEP( 'DSDA7 ', P7(28), NBANK, 'R8', LASTP, JDEBG )
382:      if ( MAXSF.ge.9 )
383:      &    call LKEEP( 'DSDA8 ', P7(29), NBANK, 'R8', LASTP, JDEBG )
384:      if ( MAXSF.ge.10 )
385:      &    call LKEEP( 'DSDA9 ', P7(30), NBANK, 'R8', LASTP, JDEBG )
386:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
387:      MAXWK = MAX(MAXWK,LASTP)
388:      write(IPR,7020) 'SEARCH', LASTP - LWORK, LASTP
389: C
390:      end if

```

src/gmvp/wrkary.f

```

391: C
392: C ===== FOR TALSUM routine =====
393: C
394:       call LSTLST( 'TALSUM', LWORK, LASTP )
395: C
396:       call LKEEP( 'FTR', P8(1), NGROUP*NTREG, 'R8', LASTP, JDEBG )
397:       call LKEEP( 'FCL', P8(2), NGROUP*NTREG, 'R8', LASTP, JDEBG )
398: CCCC call LKEEP( 'DWRK', P8(3), NEDTMX, 'R8', LASTP, JDEBG )
399:       call LKEEP( 'DWRK', P8(3), NEDTMX+NDTWMX, 'R8', LASTP, JDEBG )
400:       if ( JEIGN.ne.0 ) then
401:           call LKEEP( 'XKEF1', P8(4), 5*NLENG, 'R8', LASTP, JDEBG )
402:           call LKEEP( 'XSOCB', P8(5), NLENG, 'R8', LASTP, JDEBG )
403:           call LKEEP( 'XKB', P8(6), 5*NLENG, 'R8', LASTP, JDEBG )
404:       end if
405:       if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
406:       MAXWK = MAX(MAXWK,LASTP)
407:       write(IPR,7020) 'TALSUM', LASTP - LWORK, LASTP
408: C
409: C ===== FOR CEL2AB used for FISSION source file output =====
410: C
411:       if ( JEIGN.ne.0.and.JLATT.ne.0.and.JOSRC.ne.0 ) then
412:           call LSTLST( 'CEL2AB', LWORK, LASTP )
413: C
414:           call LKEEP( 'X', PCA(1), NFBNK0, 'R8', LASTP, JDEBG )
415:           call LKEEP( 'Y', PCA(2), NFBNK0, 'R8', LASTP, JDEBG )
416:           call LKEEP( 'Z', PCA(3), NFBNK0, 'R8', LASTP, JDEBG )
417: C
418:           if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
419:           MAXWK = MAX(MAXWK,LASTP)
420:           write(IPR,7020) 'CEL2AB', LASTP - LWORK, LASTP
421:       end if
422: C
423: C ===== FOR TALLYO routine =====
424: C
425:       call LSTLST( 'TALLYO', LWORK, LASTP )
426: C
427:       if ( JEIGN.ne.0 ) then
428: C
429:           call LKEEP( 'XSOCB', P9(1), NLENG, 'R8', LASTP, JDEBG )
430:           call LKEEP( 'XKB', P9(2), 5*NLENG, 'R8', LASTP, JDEBG )
431:           call LKEEP( 'COVKK', P9(3), 4*4*NLENG, 'R8', LASTP, JDEBG )
432:           call LKEEP( 'XKMLE', P9(4), 5*NLENG, 'R8', LASTP, JDEBG )
433:           call LKEEP( 'XKERR', P9(5), 5*NLENG, 'R8', LASTP, JDEBG )
434:           call LKEEP( 'CORR', P9(6), 4*4*3, 'R8', LASTP, JDEBG )
435:           call LKEEP( 'IWK1', P9(7), 4*4, 'I4', LASTP, JDEBG )
436: C
437:       end if
438: C
439:       call LKEEP( 'BIN', P9(11), 2*8*NBINMX, 'R8', LASTP, JDEBG )
440: CCCC call LKEEP( 'RNM', P9(12), 128*NBINMX, 'R4', LASTP, JDEBG )
441: C
442:       if ( JEIGN.ne.0 ) then
443:           call LKEEP( 'XKSD', P9(13), 4*NLENG, 'R8', LASTP, JDEBG )
444:           call LKEEP( 'XKBB', P9(14), NLENG, 'R8', LASTP, JDEBG )
445:       end if
446:       if ( JEIGN.ne.0 .or. NETRV.gt.0 ) then
447:           call LKEEP( 'CA', P9(15), NMXLG, 'R8', LASTP, JDEBG )
448:           call LKEEP( 'CR', P9(16), NMXLG, 'R8', LASTP, JDEBG )
449:       end if
450:       call LKEEP( 'UU', P9(17), NGROUP, 'R4', LASTP, JDEBG )
451: C
452:       if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
453:       MAXWK = MAX(MAXWK,LASTP)
454:       write(IPR,7020) 'TALLYO', LASTP - LWORK, LASTP
455: C

```

```

456: C
457: C ===== FOR NXTEC ===== JUNE 13 1991
458: C
459:       if ( JPTDT.ne.0 ) then
460: C
461:           call LSTLST( 'NXTEC', LWORK, LASTP )
462:           call LKEEP( 'X', PC(1), NBANK, 'R8', LASTP, JDEBG )
463:           call LKEEP( 'Y', PC(2), NBANK, 'R8', LASTP, JDEBG )
464:           call LKEEP( 'Z', PC(3), NBANK, 'R8', LASTP, JDEBG )
465:           call LKEEP( 'A', PC(4), NBANK, 'R8', LASTP, JDEBG )
466:           call LKEEP( 'B', PC(5), NBANK, 'R8', LASTP, JDEBG )
467:           call LKEEP( 'C', PC(6), NBANK, 'R8', LASTP, JDEBG )
468:           call LKEEP( 'AA', PC(7), NBANK, 'R8', LASTP, JDEBG )
469:           call LKEEP( 'BB', PC(8), NBANK, 'R8', LASTP, JDEBG )
470:           call LKEEP( 'CC', PC(9), NBANK, 'R8', LASTP, JDEBG )
471:           call LKEEP( 'DI', PC(10), NBANK, 'R8', LASTP, JDEBG )
472:           call LKEEP( 'WW', PC(11), NBANK, 'R4', LASTP, JDEBG )
473:           call LKEEP( 'MATWK', PC(12), NBANK, 'I4', LASTP, JDEBG )
474:           call LKEEP( 'IGIN', PC(13), NBANK, 'I4', LASTP, JDEBG )
475:           call LKEEP( 'ISEL', PC(14), NBANK, 'I4', LASTP, JDEBG )
476:           call LKEEP( 'IGO', PC(15), NBANK, 'I4', LASTP, JDEBG )
477:           call LKEEP( 'IWK1', PC(16), NBANK, 'I4D', LASTP, JDEBG )
478:           call LKEEP( 'IWK5', PC(17), NBANK, 'I4D', LASTP, JDEBG )
479:           call LKEEP( 'IWK6', PC(18), NBANK, 'I4D', LASTP, JDEBG )
480:           call LKEEP( 'IWK7', PC(19), NBANK, 'I4D', LASTP, JDEBG )
481:           call LKEEP( 'IWK9', PC(20), NBANK, 'I4D', LASTP, JDEBG )
482:           call LKEEP( 'IWK10', PC(21), NBANK, 'I4D', LASTP, JDEBG )
483:           call LKEEP( 'COSL', PC(22), NBANK, 'R4D', LASTP, JDEBG )
484:           call LKEEP( 'RP', PC(23), NBANK, 'R4D', LASTP, JDEBG )
485:           call LKEEP( 'R', PC(24), 2*NBANK, 'R4D', LASTP, JDEBG )
486:           call LKEEP( 'W', PC(25), NBANK, 'R4', LASTP, JDEBG )
487:           call LKEEP( 'ITO', PC(26), NBANK, 'I4', LASTP, JDEBG )
488:           call LKEEP( 'TI', PC(27), NBANK, 'R8', LASTP, JDEBG )
489: C
490:           if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
491:           MAXWK = MAX(MAXWK,LASTP)
492:           write(IPR,7020) 'NXTEC', LASTP - LWORK, LASTP
493: C
494: C .... FOR NXTEE ....
495: C
496:           call LSTLST( 'NXTEC', LASTP, LASTPC )
497:           LASTPN = MAX(LASTPS,LASTPC)
498: C
499: C .... FOR NXTFLLI CALLED IN NXTEE
500: C
501:           call LSTLST( 'NXTFLLI', LASTPN, LASTPF )
502:           call LKEEP( 'X', PF(1), IMPMAX, 'R8', LASTPF, JDEBG )
503:           call LKEEP( 'Y', PF(2), IMPMAX, 'R8', LASTPF, JDEBG )
504:           call LKEEP( 'Z', PF(3), IMPMAX, 'R8', LASTPF, JDEBG )
505:           call LKEEP( 'AI', PF(4), IMPMAX, 'R8', LASTPF, JDEBG )
506:           call LKEEP( 'BI', PF(5), IMPMAX, 'R8', LASTPF, JDEBG )
507:           call LKEEP( 'CI', PF(6), IMPMAX, 'R8', LASTPF, JDEBG )
508:           call LKEEP( 'OPT', PF(7), IMPMAX, 'R8', LASTPF, JDEBG )
509:           call LKEEP( 'DI', PF(8), IMPMAX, 'R8', LASTPF, JDEBG )
510:           call LKEEP( 'IGI', PF(9), IMPMAX, 'I4', LASTPF, JDEBG )
511:           call LKEEP( 'IBP', PF(10), IMPMAX, 'I4', LASTPF, JDEBG )
512:           call LKEEP( 'IFC', PF(11), IMPMAX, 'I4', LASTPF, JDEBG )
513:           call LKEEP( 'R', PF(12), 3*IMPMAX, 'R4', LASTPF, JDEBG )
514:           call LKEEP( 'DLOC1', PF(13), IMPMAX, 'R8', LASTPF, JDEBG )
515:           call LKEEP( 'DLOC2', PF(14), IMPMAX, 'R8', LASTPF, JDEBG )
516:           call LKEEP( 'IKL', PF(15), IMPMAX, 'I4', LASTPF, JDEBG )
517:           call LKEEP( 'IKL2', PF(16), IMPMAX, 'I4', LASTPF, JDEBG )
518:           if ( JHLAT.ne.0.or.JFISX.ne.0 ) then
519:               call LKEEP( 'XUP', PF(17), IMPMAX, 'R8', LASTPF, JDEBG )
520:               call LKEEP( 'YUP', PF(18), IMPMAX, 'R8', LASTPF, JDEBG )

```

src/gmvp/wrkary.f

```

521:      call LKEEP( 'ZUP  ', PF(19), IMPMAX, 'R8', LASTPF, JDEBG )
522:      call LKEEP( 'AUP  ', PF(20), IMPMAX, 'R8', LASTPF, JDEBG )
523:      call LKEEP( 'BUP  ', PF(21), IMPMAX, 'R8', LASTPF, JDEBG )
524:      call LKEEP( 'CUP  ', PF(22), IMPMAX, 'R8', LASTPF, JDEBG )
525:      end if
526:      call LKEEP( 'SIGT  ', PF(23), IMPMAX, 'R4', LASTPF, JDEBG )
527:      call LKEEP( 'ISTG  ', PF(24), IMPMAX, 'I4', LASTPF, JDEBG )
528:      call LKEEP( 'ILUP  ', PF(25), IMPMAX, 'I4', LASTPF, JDEBG )
529:      call LKEEP( 'ILST  ', PF(26), IMPMAX, 'I4', LASTPF, JDEBG )
530: C      if ( JFISX.ne.0 ) then
531: C          call LKEEP( 'XYZ1 ', PF(27), 3*IMPMAX, 'R8',LASTPF, JDEBG )
532: C          call LKEEP( 'ABC1 ', PF(28), 3*IMPMAX, 'R8',LASTPF, JDEBG )
533: C          call LKEEP( 'XYZ2 ', PF(29), 3*IMPMAX, 'R8',LASTPF, JDEBG )
534: C          call LKEEP( 'ABC2 ', PF(30), 3*IMPMAX, 'R8',LASTPF, JDEBG )
535: C          call LKEEP( 'DD1 ', PF(31), IMPMAX, 'R8',LASTPF, JDEBG )
536: C          call LKEEP( 'IBP0 ', PF(32), IMPMAX, 'I4',LASTPF, JDEBG )
537: C          call LKEEP( 'IBP1 ', PF(33), IMPMAX, 'I4',LASTPF, JDEBG )
538: C          call LKEEP( 'IBP2 ', PF(34), IMPMAX, 'I4',LASTPF, JDEBG )
539: C          call LKEEP( 'KPLT ', PF(35), NLBZ+1, 'I4',LASTPF, JDEBG )
540: C          call LKEEP( 'JKSF ', PF(36), NLBZ, 'I4',LASTPF, JDEBG )
541: C      end if
542: C
543: C      .... FOR NEESEA CALLED IN NXTEE
544: C
545: C      call LSTLST( 'NEESEA', LASTPN, LASTPS )
546: C      call LKEEP( 'X ', PS(1), IMPMAX, 'R8', LASTPS, JDEBG )
547: C      call LKEEP( 'Y ', PS(2), IMPMAX, 'R8', LASTPS, JDEBG )
548: C      call LKEEP( 'Z ', PS(3), IMPMAX, 'R8', LASTPS, JDEBG )
549: C      call LKEEP( 'W ', PS(4), IMPMAX, 'R8', LASTPS, JDEBG )
550: C      call LKEEP( 'IBP ', PS(5), IMPMAX, 'I4', LASTPS, JDEBG )
551: C      call LKEEP( 'IFI ', PS(6), IMPMAX, 'I4', LASTPS, JDEBG )
552: C      call LKEEP( 'FXYZ ', PS(7), IMPMAX, 'R4', LASTPS, JDEBG )
553: C      call LKEEP( 'ISRF ', PS(8), IMPMAX, 'I4', LASTPS, JDEBG )
554: C      call LKEEP( 'IZI ', PS(9), IMPMAX, 'I4', LASTPS, JDEBG )
555: C      call LKEEP( 'IFL ', PS(10), IMPMAX, 'I4', LASTPS, JDEBG )
556: C
557: C      .... FOR LATTICE CALLED IN NXTEE
558: C
559: C      call LSTLST( 'LATTICE', LASTPN, LASTPL )
560: C      if ( JLATT.ne.0 ) then
561: C          call LKEEP( 'X ', PL(1), IMPMAX, 'R8',
562: C                      LASTPL, JDEBG )
563: C          call LKEEP( 'Y ', PL(2), IMPMAX, 'R8',
564: C                      LASTPL, JDEBG )
565: C          call LKEEP( 'Z ', PL(3), IMPMAX, 'R8',
566: C                      LASTPL, JDEBG )
567: C          call LKEEP( 'AI ', PL(4), IMPMAX, 'R8',
568: C                      LASTPL, JDEBG )
569: C          call LKEEP( 'BI ', PL(5), IMPMAX, 'R8',
570: C                      LASTPL, JDEBG )
571: C          call LKEEP( 'CI ', PL(6), IMPMAX, 'R8',
572: C                      LASTPL, JDEBG )
573: C          call LKEEP( 'IWK ', PL(7), IMPMAX, 'I4',
574: C                      LASTPL, JDEBG )
575: C          call LKEEP( 'JKSF ', PL(8), NZONE+1, 'I4',
576: C                      LASTPL, JDEBG )
577: C          call LKEEP( 'IPZONE', PL(9), NZONE+1, 'I4',
578: C                      LASTPL, JDEBG )
579: C          call LKEEP( 'MEM ', PL(10), NZONE+1, 'I4',
580: C                      LASTPL, JDEBG )
581: C          if ( JFISX.ne.0 .or. JSTGR.ne.0 ) then
582: C              call LKEEP( 'DI ', PL(11), IMPMAX, 'R8',
583: C                          LASTPL, JDEBG )
584: C              call LKEEP( 'DLOC1 ', PL(12), IMPMAX, 'R8',
585: C                          LASTPL, JDEBG )

```

```

586:      call LKEEP( 'DLOC2 ', PL(13), IMPMAX, 'R8',
587:                  LASTPL, JDEBG )
588:      call LKEEP( 'IKL ', PL(14), IMPMAX, 'I4',
589:                  LASTPL, JDEBG )
590:      call LKEEP( 'IKL2 ', PL(15), IMPMAX, 'I4',
591:                  LASTPL, JDEBG )
592:      call LKEEP( 'R ', PL(16), 3*IMPMAX, 'I4',
593:                  LASTPL, JDEBG )
594:      end if
595:      end if
596: C
597: C      LASTP = MAX(LASTPF,LASTPS,LASTPL)
598: C      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
599: C
600: C      MAXWK = MAX(MAXWK,LASTP)
601: C      write(IPR,7020) 'NXTEE ', LASTP - LWORK, LASTP
602: C
603: C      end if
604: C
605: C      ===== FOR "underground" modules =====
606: C      Working area for "underground" modules must be put after any
607: C      ordinary working area (
608: C
609: C      if ( JUNDG.ne.0 ) then
610: C          LWORK2 = MAXWK
611: C          call WKAA2Z( 'GMVP', A, H, LIMIT, LIMITL, JLATT, JHLAT, JTLLT,
612: C                      JDEBG, IOVFL, LWORK2, MAXWK )
613: C      end if
614: C
615: C      C=====
616: C
617: C      7020 format(' === SUBROUTINE <','A6,> NEEDS WORKING AREA OF ',I12,
618: C                & ' WORDS. ==== ( LAST POSITION ',I12,')')
619: C
620: C
621: C      Cccc LAST = MAXWK
622: C      Cccc call lgtlst( last )
623: C      if ( LSIZL(MAXWK).gt.LIMITL ) then
624: C          write(IMG,7040) LSIZL(MAXWK), LIMITL
625: C          7040 format(' XXX INSUFFICIENT TASK LOCAL MEMORY TO KEEP',
626: C                      & ' WORKING AREA.'/' ( NECESSARY ',I12,
627: C                      & ' WORDS. LIMIT ',I12,'WORDS)')
628: C          call CNTERR( 'FATAL' )
629: C      end if
630: C
631: C      return
632: C      end

```

src/gmvp/xseccn.f

```

1:      subroutine XSECCN( IOW, JDDX, JFISS, JEIGN, JSTATX,JADJM,
2:      &                  NGROUP,NGP1, NGP2, NGPX, NGGX, NTGX,
3:      &                  NDSGX, NDSGX, NDSTX, NUSX, NSCTX, NCOEFX,
4:      &                  NMEDX, NGSX, NNNX, STOTX, SSCX, SNUFX,
5:      &                  SNAPX, SGPX, SNFPX, SNUSX, SGPBX, IGPBX,
6:      &                  SSCBX, ISCBX, SDBBX, IDDBX, SANGX, IANGX,
7:      &                  SGPPX, SSCPX, SPORT, S2DPX, S2PPX, FKAI, KKAI
8:      &                  )
9: C=====
10: C PURPOSE: CONDENSE OF MULTI-GROUP CROSS SECTION WHEN NGROUP<NTG IN
11: C           A NEUTRON-PHOTON MIXED PROBLEM
12: C CALLED IN: INTRO
13: C CALLS:
14: C=====
15:      implicit real*8(A-H,O-Z)
16: C
17: C
18: C**** CROSS SECTION DATA
19:      real STOTX(NTGX,1), SSCX(NTGX,1), SNUFX(NTGX,1), SNAPX(NTGX,1),
20:      & SGPX(NGPX,1), SNFPX(NTGX,1), SNUSX(NTGX,1),
21:      & S2DPX(2,NTGX,1), S2PPX(3,NTGX,1), SGPBX(NGGX,3,NGPX,1),
22:      & SSCBX(1), SDBBX(1), SGPPX(1), SSCPX(1), SPORT(1),
23:      & SANGX(NSCTX,3,NTGX,NMEDX), FKAI(NTGX,2)
24:      integer NGSX(1), NNNX(1), IGPBX(NGGX,3,NGPX,1), ISCBX(1),
25:      & IDDBX(1), KKAI(2,NTGX,1), IANGX(NSCTX,3,NTGX,NMEDX)
26: C
27: C ..... ISCBX & IDDBX HAVE THE SAME ADDRESSES AS SSCBX & SDBBX .....
28: C IANGX HAVE THE SAME ADDRESSES AS SANGX .....
29: C
30: C
31: C
32:      NSCT3 = NSCTX*3
33:      NSCT3N = NSCT3*NDSTX
34:      NDST3 = NDSTX*3
35: C
36: C
37:      if ( NGP1.lt.NGPX ) then
38: C****
39: *VOCL LOOP,NOVREC
40:      do 100 I = 1, NGP2
41:          JN = NGP1 + I
42:          JO = NGPX + I
43:          NGSX(JN) = NGSX(JO)
44:          NNNX(JN) = NNNX(JO)
45:      100 continue
46: C
47: C**** ONE-DIMENSIONAL CROSS SECTION
48: C
49: *VOCL LOOP,NOVREC
50:      do 120 K = 1, NMEDX
51: *VOCL LOOP,NOVREC
52:      do 110 I = 1, NGP2
53:          JN = NGP1 + I
54:          JO = NGPX + I
55:          STOTX(JN,K) = STOTX(JO,K)
56:          SSCX(JN,K) = SSCX(JO,K)
57:          SNUFX(JN,K) = SNUFX(JO,K)
58:          SNAPX(JN,K) = SNAPX(JO,K)
59:          SNFPX(JN,K) = SNFPX(JO,K)
60:          SNUSX(JN,K) = SNUSX(JO,K)
61:          S2DPX(1,JN,K) = S2DPX(1,JO,K)
62:          S2DPX(2,JN,K) = S2DPX(2,JO,K)
63:          S2PPX(1,JN,K) = S2PPX(1,JO,K)
64:          S2PPX(2,JN,K) = S2PPX(2,JO,K)
65:          S2PPX(3,JN,K) = S2PPX(3,JO,K)

```

```

66:      110 continue
67:      120 continue
68: C
69: C***** ENERGY-ANGLE DISTRIBUTION
70: C
71:      if ( JDDX.ne.0 ) then
72: C
73:          if ( JSTATX.eq.0 ) then
74:              IGDM = (NGROUP+1+NUSX)*NSCTX
75: *VOCL LOOP,NOVREC
76:          do 150 KMED = 1, NMEDX
77:              do 140 IG = 1, NGP1
78: C ***** DDX DATA FOR BMC SAMPLING
79: C THIS PROCEDURE IS USED FOR ENERGY CUT-OFF
80: C ..... L0: HEAD POSITION OF MATRIX
81:              L0 = (KMED-1)*NSCT3N + NGSX(IG)*NSCT3
82:              N1 = NNNX(IG)*NSCTX
83:              L2 = N1 + L0
84:              IGDM = (NGP1-IG+1+NUSX)*NSCTX
85:              if ( IGDM.lt.N1 ) then
86:                  do 130 I = 1, N1*2
87:                      if ( IDDBX(L2+I).gt.IGDM ) IDDBX(L2+I) =
88:                          & IGDM
89:                  130 continue
90:              end if
91: C
92:          140 continue
93:          150 continue
94: C ..... JSTATX > 0
95:          else
96:              IGDM = NGROUP + 1 + NUSX
97: *VOCL LOOP,NOVREC
98:              do 220 KMED = 1, NMEDX
99: C
100: C ***** ENERGY DISTRIBUTION FOR BMC SAMPLING
101: C THIS PROCEDURE IS USED FOR ENERGY CUT-OFF
102:              do 180 IG = 1, NGP1
103:                  NNNX1 = NNNX(IG)
104:                  IGDM = NGP1 - IG + 1 + NUSX
105:                  if ( NNNX1.gt.IGDM ) then
106:                      L0 = (KMED-1)*NSCT3N + NGSX(IG)*NSCT3
107:                      do 170 MU = 1, NSCTX
108:                          L1 = L0 + (MU-1)*NNNX1*3 + NNNX1
109:                          do 160 I = 1, NNNX1*2
110:                              if ( IDDBX(L1+I).gt.IGDM ) IDDBX(L1
111:                                  & +I) = IGDM
112:                          160 continue
113:                      170 continue
114:                  end if
115:              180 continue
116: C
117: C ..... ANGULAR DISTRIBUTION
118:              do 200 IG = 1, NGP2
119:                  JN = NGP1 + IG
120:                  JO = NGPX + IG
121: *VOCL LOOP,NOVREC
122:                  do 190 I = 1, 3*NSCTX
123:                      SANGX(I,1,JN,KMED) = SANGX(I,1,JO,KMED)
124:                  190 continue
125:                  200 continue
126:                  JO = NSCTX*NTGX*(KMED-1)
127:                  JN = JO + NGP1*NSCTX
128:                  JO = JO + NGPX*NSCTX
129: *VOCL LOOP,NOVREC
130:                  do 210 I = 1, NGP2*NSCTX

```


src/gmvp/xseccn.f

```
131:          SPORT(JN+I) = SPORT(JO+I)
132:          continue
133: C
134: 210          continue
135:          end if
136: C
137:          else
138: C
139: C**** PL
140: C
141:          IGDMM = NGROUP + 1 + NUSX
142: *VOCL LOOP,NOVREC
143:          do 250 KMED = 1, NMEDX
144:          do 240 IG = 1, NGP1
145:              L0 = (KMED-1)*NDST3 + NGSX(IG)*3
146:              NNNX1 = NNNX(IG)
147:              IGDM = NGP1 - IG + 1 + NUSX
148:              if ( NNNX1.gt.IGDM ) then
149:                  L1 = L0 + NNNX1
150:                  do 230 I = 1, NNNX1*2
151:                      if ( ISCBX(L1+I).gt.IGDM ) ISCBX(L1+I) = IGDM
152:                  continue
153:              end if
154:          continue
155: 250          continue
156: C
157:          end if
158: C
159:          end if
160: C
161: C **** PHOTON PRODUCTION
162:          do 280 KMED = 1, NMEDX
163:          do 270 IG = 1, NGP1
164:              do 260 I = 1, NGGX*2
165:                  IGPBX(I,2,IG,KMED) = IGPBX(I,2,IG,KMED) + NGP1
166:              continue
167:          continue
168: 280          continue
169: C
170: C **** FISSION SPECTRUM
171:          if ( JFISS.ne.0 ) then
172:              do 290 I = 1, NMEDX*NTGX*2
173:                  if ( KKAI(I,1,1).gt.NGP1.and.KKAI(I,1,1).le.NGPX ) then
174:                      KKAI(I,1,1) = NGROUP + 1
175:                  else if ( KKAI(I,1,1).gt.NGPX ) then
176:                      KKAI(I,1,1) = KKAI(I,1,1) - NGPX + NGP1
177:                  end if
178:              continue
179:          end if
180: C
181:          return
182:          end
```

src/gmvp/xsecin.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine XSECIN( ICALL, SIGT, NSIG, LAST, LIMIT, IO, IN )
3: C=====
4: C PURPOSE : INPUT CROSS-SECTION RELATED DATA
5: C
6: C << Comment >>
7: C Most of cross section modules of GMVP are borrowed from MORSE-CG
8: C and MORSE-DD (JAERI's hack enabling DDX type cross section),
9: C and their coding style is old-fashioned and confusing.
10: C Check for array pointer overflow may be imperfect at some points.
11: C Variables in a common area like /LOCSIG/ may have different names
12: C in other routines under XSECIN.
13: C They should be re-written into a modern coding style, I hope.
14: C
15: C SIGT(*) & NSIG(*) have the same memory address and equivalent to
16: C the array A(*) in common /ARRAY/, and pointed as SIGT/NSIG
17: C because of a historical reason commented above.
18: C
19: C Array index pointers are used differently from those in other parts
20: C of GMVP.
21: C For example, actual variable data area pointed by IRSG start from
22: C SIGT(IRSG+1) and not from SIGT(IRSG). The index pointers are
23: C corrected for use in other part of the program by incrementing their
24: C value.
25: C
26: C -----
27: C
28: C CALLED IN : INTRO
29: C CALLS : GTNDSK,PRSTOP,JNPUT,XSTAPE
30: C=====
31: C
32: C THIS ROUTINE IS THE PRIMARY INTERFACE BETWEEN CROSS SECTION MODUL
33: C AND THE REST * * * * *
34: C IN THIS CROSS SECTION MODULE, VARIABLES IN LOCSIG SHOW THE
35: C ( STARTING LOCATION -1 ).
36: C
37: C include '../shared/INC/_IOUNIT'
38: C
39: C include '../shared/INC/_LISTOFF'
40: C include 'INC/_PXSEC'
41: C include 'INC/_FLAGS'
42: C include '../shared/INC/_LISTON'
43: C/#IF PARA(PVM)
44: C * include '../shared/INC/_PVMPARA'
45: C/#ELSEIF PARA(MPI)
46: C * include 'mpif.h'
47: C/#ENDIF
48: C include '../shared/INC/_TASKDT'
49: C
50: C ---- PRINT OUT OF COMMON /PXSEC/ & /FLAGS/ ARE SUPPRESSED ----
51: C
52: C common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IFNGP,
53: C & IFSPOG, IDSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
54: C & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
55: C & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
56: C & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
57: C & IADJ, NME, LOC, INGS, INSG, I1, IO,
58: C & KKK, IXTAPE, IDEL, ITEM1, ITEMG, IRSG, IRDSG,
59: C & ISTR, IPRIN, IFMU, IMOM, IDTF, ISTAT, IPUN,
60: C & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
61: C & INNN, IGGG, IPNGP, IPSPOG, IPNGOG, IANG, INUFIS,
62: C & IS2DP, IS2PP
63: C common /GTSC1/ NPT(16), NXSECT(17), NDSGP, ID(2),
64: C & IO6RT, IGQPT, NEGPS, ESPD(100), NNIC, NXPM,
65: C & NINC

```

```

66: C
67: C common /DDX/ IDDX, MAXU, MAXSD, IMAX, NLASTD, MAXDDX,
68: C & ISTTRT
69: C
70: C character*4 TITLE(20)
71: C
72: C .... DATA AREA ....
73: C
74: C integer NSIG(*)
75: C real SIGT(*)
76: C
77: C *** EQUIVALENCE (SIGT(1),NSIG(1)) **** ASSUMED ****
78: C
79: C/#IF PARA(PVM MPI)
80: C
81: C ... working array for data passing to other tasks ...
82: C * parameter( MXSEND = 50 )
83: C * integer ISEND(MXSEND)
84: C/#ENDIF
85: C
86: C ---- STATEMENT FUNCTION TO INDICATE 'POINTER' CALCULATION ! ----
87: C
88: C IPOINT(IP,NSIZE) = IP + NSIZE
89: C
90: C -----
91: C
92: C ICALL = 1
93: C I1 = IN
94: C IO = IO
95: C NGN = 0
96: C IHT = 3
97: C NMKSEC = 0
98: C LSTRT = LAST
99: C NLAST = LAST - 1
100: C
101: C ... LIMIT is size of usable area in array SIGT/NSIG.
102: C ILIMIT is max index of array SIGT/NSIG.
103: C
104: C ILIMIT = ILSIZ(LIMIT)
105: C
106: C ICHK = 0
107: C
108: C .... INPUT A COMMENT FOR CROSS SECTION DATA ....
109: C
110: C read(I1,7000) TITLE
111: C 7000 format(20A4)
112: C
113: C
114: C =====
115: C CALL HEADER( IO, 'CROSS SECTION PROCESSING' )
116: C =====
117: C
118: C
119: C *****
120: C call LABEL( IO, 'TITLE FOR CROSS SECTION DATA' )
121: C *****
122: C
123: C
124: C write(IO,7020) TITLE
125: C 7020 format(' TITLE : ',20A4)
126: C
127: C
128: C IADJ = JADJM
129: C
130: C ..... READ DATA X-1 .....

```

src/gmvp/xsecin.f

```

131: C
132: C   read(I1,*) NGP, NDS, NGG, NDSG, INGP, ITBL, ISGG, NMED, NELEM,
133: C   &      NMIX, NCOEF, NSCT, ISTAT, IDDX
134: C   read(I1,*) NGP, NDS, NGG, NDSG, INGP, ITBL, ISGG, NMED, NELEM,
135: C   &      NMIX, NCOEF, NSCT, ISTAT, IDDX, IHT
136: C
137: C   ==== DDX MODE ====
138: C
139: C   if ( IDDX.ne.0 ) then
140: C       if ( JNEUT.eq.0.and.NGP.ne.0 ) NGP = 0
141: C       if ( JPHOT.eq.0.and.NGG.ne.0 ) NGG = 0
142: C       if ( INGP.ne.(NGG+NGP) ) then
143: C           write(IMG,*)
144: C           &      '!!!(XSECIN) INGP IS NOT EQUAL TO NGP+NGG IN DDX MODE,',
145: C           &      '      ' INGP IS CHANGED TO NGP + NGG !!!'
146: C           call CNTERR( 'WARNING' )
147: C           INGP = NGG + NGP
148: C       end if
149: C   end if
150: C
151: C   ICHK = 0
152: C   if ( JNEUT.ne.0.and.NGP.le.0 ) then
153: C       write(IMG,*) 'XXX(XSECIN) NGP=0 FOR NEUTRON CALCULATION !'
154: C       call CNTERR( 'FATAL' )
155: C       ICHK = ICHK + 1
156: C   end if
157: C   if ( JPHOT.ne.0.and.NGG.le.0.and.IDDX.ne.0 ) then
158: C       write(IMG,*) 'XXX(XSECIN) NGG=0 FOR PHOTON CALCULATION !'
159: C       call CNTERR( 'FATAL' )
160: C       ICHK = ICHK + 1
161: C   end if
162: C
163: C   *****
164: C   call LABEL( I0, 'CONTROL PARAMETERS' )
165: C   *****
166: C
167: C   write(I0,7060) NGP, NDS, NGG, NDSG, INGP, ITBL, IHT, ISGG,
168: C   &      NMED, NELEM, NMIX, NCOEF, NSCT, ISTAT, IADJ, IDDX, IHT
169: C   7060 format('      NUMBER OF PRIMARY GROUPS (NGP)      ',I8/
170: C   &      '      NUMBER OF PRIMARY DOWNSCATTERS (NDS)   ',I8/
171: C   &      '      NUMBER OF SECONDARY GROUPS (NGG)        ',I8/
172: C   &      '      NUMBER OF SECONDARY DOWNSCATTERS (NDSG)  ',I8/
173: C   &      '      NUMBER OF PRIM+SEC GROUPS (INGP)         ',I8/
174: C   &      '      TABLE LENGTH (ITBL)                    ',I8/
175: C   &      '      LOC OF TOTAL X-SEC (SIG T) (IHT)         ',I8/
176: C   &      '      LOC OF WITHIN GROUP (SIG GG) (ISGG)      ',I8/
177: C   &      '      NUMBER OF MEDIA (NMED)                   ',I8/
178: C   &      '      NUMBER OF INPUT ELEMENTS (NELEM)         ',I8/
179: C   &      '      NUMBER OF MIXING ENTRIES (NMIX)          ',I8/
180: C   &      '      NUMBER OF COEFFICIENTS (NCOEF)           ',I8/
181: C   &      '      NUMBER OF ANGLES (NSCT)                  ',I8/
182: C   &      '      RESTORE COEFF (ISTAT)                    ',I8/
183: C   &      '      ADJOINT SWITCH (JADJM)                   ',I8/
184: C   &      '      DUBLE DIFF. C.S. SWITCH (IDDX)          ',I8/
185: C   &      '      POSITION OF SIGMA TOTAL (IHT)             ',I8)
186: C
187: C   .... SET ISTAT = 1 IF POINT DETECTOR OPITION IS ACTIVE ...
188: C
189: C   if ( JPTDT.ne.0.and.ISTAT.eq.0 ) then
190: C       write(IMG,/'(lx,/'!!! ISTAT IS SET TO 1 BECAUSE THE POINT-',
191: C       &      'DETECTOR OPTION IS ACTIVE.(')') )
192: C       call CNTERR( 'WARNING' )
193: C       ISTAT = JPTDT
194: C   end if
195: C

```

```

196: C..... READ DATA X-2 .....
197: C
198: C   read(I1,*) IRDSG, ISTR, IFMU, IMOM, IPRIN, IPUN, IDTF, IXTAPE,
199: C   &      JXTAPE
200: C
201: C
202: C   *****
203: C   call LABEL( I0, 'INPUT/OUTPUT OPTIONS' )
204: C   *****
205: C
206: C
207: C   write(I0,7080) IRDSG, ISTR, IFMU, IMOM, IPRIN, IPUN, IDTF, IXTAPE,
208: C   &      JXTAPE
209: C
210: C   7080 format(7X,'IRDSG (PRINT AS READ)      ',I5/7X,
211: C   &      'ISTR (PRINT AS STORE)              ',I5/7X,
212: C   &      'IFMU (PRINT MUS)                    ',I5/7X,
213: C   &      'IMOM (PRINT MOMENTS)                ',I5/7X,
214: C   &      'IPRIN (PRINT ANGLES,PROB)            ',I5/7X,
215: C   &      'IPUN (PRINT IMPOSSIBLE COEF)        ',I5/7X,
216: C   &      'PL DATA FORMAT (IDTF)              ',I5/7X,
217: C   &      'INPUT FILE I/O UNIT(IXTAPE)         ',I5/7X,
218: C   &      'OUTPUT FILE I/O UNIT(JXTAPE)       ',I5)
219: C
220: C
221: C   ... abs(IXTAPE) > 1000 : ANISN type x-sec from MGCL-MAIL etc.
222: C   (Each x-sec record contain only one group.)
223: C
224: C
225: C   if ( ABS(IXTAPE).gt.1000 ) then
226: C       JSEP = 1
227: C       IXTAPE = SIGN(ABS(IXTAPE)-1000,IXTAPE)
228: C   else
229: C       JSEP = 0
230: C   end if
231: C
232: C   if ( ICHK.ne.0 ) then
233: C       call PRSTOP( ICHK, 'ERROR IN NUMBER OF ENERGY GROUP ETC.' )
234: C       stop 888
235: C   end if
236: C
237: C   -----
238: C   NGG =0 IF NOT DOING GAMMA TRANSPORT
239: C   ISTAT GT 0 SAVES LEGENDRE COEFF FOR STATISTICAL ESTIMATION
240: C   -----
241: C
242: C   IXTAP = IXTAPE
243: C   NTG = NGP + NGG
244: C   ....( EASY PATH. READ ALREADY PROCESSED CROSS SECTIONS )...
245: C   if ( IXTAP.lt.0 ) go to 140
246: C
247: C
248: C
249: C   ===== PROCESSING OF CROSS SECTION DATA =====
250: C
251: C
252: C
253: C   NME = 1
254: C   INDS = ITBL - ISGG + 1
255: C   NUS = ISGG - IHT - 1
256: C   if ( NGP.le.0 ) then
257: C       NGP = NGG
258: C       NDS = NDSG
259: C       NME = INGP - NGG + 1
260: C       NGG = 0

```

src/gmvp/xsecin.f

```

261:      NDSG      = 0
262:    end if
263:      NTS        = NDS + NDSG
264:      if ( IADJ.gt.0.and.NGG.gt.0 ) then
265:        NTE      = NGG
266:        NGG      = NGP
267:        NGP      = NTE
268:        NTE      = NDS
269:        NDS      = NDSG
270:        NDSG     = NTE
271:      end if
272: C
273:      IRSG      = IPOINT(NLAST,0)
274:      IANG      = IPOINT(IRSG,3*NMIX)
275: C
276:      if ( IDDX.ne.0 ) then
277: C
278: C .... READ COSINE BOUNDARIES FOR DDX DATA .....
279: C
280: C
281: C *****
282: C      call LABEL( IO, 'ANGLE STRUCTURE OF DDX ( COSINE BOUNDARIES '
283: C &
284: C *****
285: C
286: C
287: C      read(I1,*) (SIGT(IANG+I),I=1,NSCT+1)
288: C      write(IO,7100) (SIGT(IANG+I),I=1,NSCT+1)
289: 7100 C      format(10X,10E12.3)
290: C
291: C      INGS     = IPOINT(IANG,NSCT+1)
292: C      NCOEF    = 1
293: C      else
294: C      INGS     = IPOINT(IANG,0)
295: C      end if
296: C
297: C      INSG     = IPOINT(ING,NGP)
298: C      IDEL     = IPOINT(ING,NGG)
299: C      NEC      = 0
300: C      NTS      = NTS + NUS
301: C      if ( IXTAP.gt.0 ) then
302: C      call RWIND( IXTAP )
303: C
304: C ..... READ DATA X-3 .....
305: C
306: C      NEC      = NELEM*NCOEF
307: C      IDM      = IPOINT(IDEL,NEC)
308: C      IDE      = IDEL + 1
309: C      read(I1,*) (NSIG(M),M=IDE,IDM)
310: C
311: C
312: C *****
313: C      call LABEL( IO, 'ELEMENTS FROM LIBRARY FILE (IDENTIFIERS)' )
314: C *****
315: C
316: C
317: C      write(IO,7120) (NSIG(M),M=IDE,IDM)
318: 7120 C      format(10X,10I8)
319: C      end if
320: C
321: C      INNN     = IPOINT(IDEL,NEC)
322: C      IGGG     = IPOINT(INNN,NGP)
323: C      ISTART   = IPOINT(IGGG,NGG)
324: C      NDSNGG   = 0
325: C      NDSNGP   = 0

326:      N1        = NUS + NDS
327: C
328: C      do 100 I = 1, NGP
329: C      NSIG(ING+I) = NDSNGP
330: C      NDSK      = NGP - I + 1 + NUS
331: C
332: C      NNNNN    = NDSK
333: C      NDSK     = MIN(MAX(NUS+1,NNNNN),N1)
334: C      NSIG(I+INNN) = NDSK
335: C      NDSNGP    = NDSNGP + NDSK
336: C      100 continue
337: C
338: C      N1        = NUS + NDSG
339: C      NDSNGP    = 0
340: C      if ( NGG.gt.0 ) then
341: C      do 110 I = 1, NGG
342: C      NSIG(ING+I) = NDSNGP
343: C      NDSK      = NGG - I + 1 + NUS
344: C
345: C      NNNNN    = NDSK
346: C      NDSK     = MIN(MAX(NUS+1,NNNNN),N1)
347: C      NSIG(I+IGGG) = NDSK
348: C      NDSNGP    = NDSNGP + NDSK
349: C      110 continue
350: C      end if
351: C
352: C
353: C      if ( IXTAP.gt.0 ) then
354: C      NLASTD = ILIMIT
355: C
356: C ..... GET MAXIMUM NUMBER OF DOWNSCATTERS .....
357: C
358: C      call GTNDSK( SIGT, NSIG, ILIMIT, JNEUT, JPHOT, JSEP )
359: C      if ( IADJ.le.0 ) then
360: C      N1      = 1
361: C      N2      = NGP
362: C      else if ( NGG.le.0 ) then
363: C      N1      = 1
364: C      N2      = NGP
365: C      else
366: C      N1      = NGP + 1
367: C      N2      = NTG
368: C      end if
369: C      N3      = N1 + 1
370: C      NSIG(ING+N1) = 0
371: C      do 120 I = N3, N2
372: C      NSIG(ING+I) = NSIG(ING+I-1) + NSIG(INNN+I-1)
373: C      120 continue
374: C      end if
375: C
376: C      NDSNGP = NSIG(ING+NGP) + NSIG(INNN+NGP)
377: C      if ( NGG.gt.0 ) then
378: C      NDSNGG = NSIG(ING+NGG) + NSIG(IGGG+NGG)
379: C      end if
380: C
381: C      ISCCOG = IPOINT(ISTART,NTG*NMED)
382: C      INUFIS = IPOINT(ISCCOG,NTG*NMED)
383: C      INABOG = IPOINT(INUFIS,NTG*NMED)
384: C      IGABOG = IPOINT(INABOG,NTG*NMED)
385: C      IFPORG = IPOINT(IGABOG,NGP*NMED)
386: C      INUS   = IPOINT(IFPORG,NTG*NMED)
387: C      INGN   = IPOINT(INUS,NTG*NMED)
388: C      INGNP  = IPOINT(INGN,0)
389: C * * INGNP=INGN+NGG * NMED
390: C      IPNGP=INGNP

```

src/gmvp/xsecin.f

```

391: C * * IFNGP=INGNP+NGG*NGP * NMED
392:   IS2DP = IPOINT(INGNP,0)
393:   IS2PP = IPOINT(IS2DP,2*NTG*NMED)
394:   IPNGP = IPOINT(IS2PP,3*NTG*NMED)
395:   IPSPOG = IPOINT(IPNGP,NGP*NGG*3*NMED)
396:   IPRBNG = IPOINT(IPSPOG,(NDSNGP+NDSNGG)*3*NMED)
397:   if ( IDDX.ne.0 ) IPRBNG = IPOINT(IPSPOG,0)
398: C
399: C * * THE FOLLOWING AREA IS RESERVED FOR DDX BMC SAMPLING(DDX MODE)
400: C * * IN PL MODE, SCATTERING ANGLE SAMPLING IS CARRIED OUT BY BMC SAMPLI
401: C
402:   ISCANG = IPOINT(IPRBNG,(NDSNGP+NDSNGG)*NSCT*3*NMED)
403:   NTEMP = 0
404:   if ( IDDX.ne.0.and.ISTAT.ne.0 ) NTEMP = INGP*NSCT*3*NMED
405:   ISPORG = IPOINT(ISCANG,NTEMP)
406: C
407:   if ( IDDX.ne.0 ) then
408:     if ( ISTAT.ne.0 ) then
409:       IFNGP = IPOINT(ISPORG,INGP*NSCT*NMED)
410:     else
411:       IFNGP = IPOINT(ISPORG,0)
412:     end if
413:     IFSPOG = IPOINT(IFNGP,0)
414:     IDSGOG = IPOINT(IFSPOG,0)
415:   else
416:     IFNGP = IPOINT(ISPORG,0)
417:     NMEDT = NMED
418:     IFSPOG = IPOINT(IFNGP,NGP*NGG*NMEDT)
419:     IDSGOG = IPOINT(IFSPOG,(NDSNGP+NDSNGG)*NMEDT)
420:   end if
421: C
422: C
423:   ISPM = IDSGOG - ISTART
424:   NGP3 = INGP + IHT + NUS
425:   IMEND = IPOINT(ISTART,MAX(ISPM,INGP*NGP3))
426: C
427: C
428: C BUFFER REGION FOR INPUT CROSS SECTIONS STARTS AT ISTART
429: C
430: C
431:   INPBUF = ISTART
432: C
433:   if ( IDDX.ne.0 ) then
434: C----- USED IN APMATX
435:     IBUF = NSCT*INGP + INGP + 6 + NSCT + NGG
436:     if ( IBUF.lt.(NSCT*INGP*3) ) IBUF = NSCT*INGP*3
437:   else
438:     IBUF = NELEM*(NTG*(NTS+2)+(NDSNGP+NDSNGG)*(NCOEF-1))
439:     if ( NCOEF.eq.1 ) IBUF = IBUF + NTG
440:   end if
441: C
442: C -----
443: C Memory mapping ....
444: C
445: C SIGT 1 ... IRDSG : unusable under XSECIN.
446: C SIGT IRDSG + 1 ... ISTART(=INPBUF) : number density information etc.
447: C SIGT ISTART+1 ... IMEND : processed xsec data or buffer area.
448: C SIGT IMEND+1 ... ISPORT : not used (or should not be)
449: C SIGT ISPORT+1 ... ILIMIT : working area
450: C -----
451: C
452:   ISPORT = ILIMIT - IBUF
453:   NTEMP = IMEND - ISPORT
454: C
455:   if ( NTEMP.gt.0 ) then

```

```

456:   write(IMSG,7140) NTEMP, LIMIT
457: 7140 format('// XXX(XSECIN) TEMPORARY CROSS-SECTION STORAGE EXCEEDS',
458: & ' WORKING AREA BY',I12,' WORDS '/'
459: & ' MEMORY LIMIT FOR THIS LOAD MODULE IS',I12,
460: & ' WORDS !! '/' )
461:   call CNTERR( 'FATAL' )
462:   call PRSTOP( 1, 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.'
463: & )
464:   stop 999
465: end if
466: C
467: C
468:   ISPORT = IPOINT(ISPORT,0)
469:   ISIGOG = IPOINT(ISPORT,NTG*NTS*NELEM)
470:   INFPOG = IPOINT(ISIGOG,NTG*NELEM)
471:   IABSOG = IPOINT(INFPOG,NTG*NELEM)
472:   ITEMG = IPOINT(IABSOG,NDSNGP*(NCOEF-1))
473:   ITOTSG = IPOINT(ITEMG,NDSNGG*(NCOEF-1))
474:   ITEML = ITOTSG - IABSOG
475: C
476: C
477:   NMXSEC = IABSOG+ITEML*NELEM
478: C
479:   NMXSEC = ILIMIT
480:   if ( ISTAT.le.0 ) then
481:     NLAST = IPOINT(ISPORG,0)
482:     IBUF = ISPORT - NLAST
483:   else
484:     if ( IDDX.eq.0 ) then
485:       ISPM = NMED*ITEML
486:       NLAST = IPOINT(ISPORG,ISPM+(NDSNGP+NDSNGG+NGP*NGG)*NMED)
487:       IBUF = IABSOG - NLAST
488:     else
489:       ISPM = (NDSNGP+NDSNGG) * NSCT * NMED
490:       ISPM = INGP*NSCT*NMED
491:       NLAST = IPOINT(ISPORG,ISPM)
492:       IBUF = ISPORT - NLAST
493:     end if
494:   end if
495: C
496: C *****
497: call LABEL( I0, 'STORAGE ALLOCATIONS' )
498: C *****
499: C
500: C
501:   write(I0,7160) IRSG + 1, NLAST, ISPORT + 1, NMXSEC, IBUF, ILIMIT
502: 7160 format(1x,' CROSS SECTIONS START AT ',I12/
503: & 1x,' LAST LOCATION USED (PERM) ',I12/
504: & 1x,' TEMP LOCATIONS USED ',I12,' TO ',I12/
505: & 1x,' EXCESS STORAGE (TEMP) ',I12/
506: & 1x,' MAX. LOCATION (ILIMIT) ',I12)
507: C
508:   if ( IBUF.lt.0 ) then
509:     write(IMSG,7180) abs(IBUF)
510: 7180 format(1x,'XXX NOT ENOUGH MEMORY FOR CROSS SECTION TREATMENT.'/
511: & 1x,' MORE ',I12,' WORDS ARE NECESSARY.')
512:     call PRSTOP( 1, 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.'
513: & )
514:     stop 999
515:   end if
516: C
517: C
518:   ISTTRT = ISTART
519:   NLASTD = ILIMIT
520: C

```

src/gmvp/xsecin.f

```

521: C
522: C.... WORKING AREA USED FOR MACROSCOPIC CROSS SECTION PREPARATION
523: C      IN DDX MODE
524: C
525:       if ( IDDX.ne.0 ) then
526:         NTEMP = NSCT*MAXSD*INGP + 5*INGP
527:         if ( NGG*NGP.ne.0 ) NTEMP = NTEMP + INGP + NGP*NGG
528:         NNLAST = IPOINT(ISTART,2*NTEMP)
529:         NTEMP2 = INGP*MAXSD
530:         if ( NUS.gt.0 ) NTEMP2 = MAX(NTEMP2,(MAXSD+NUS)*NSCT)
531:         NTEMP2 = NTEMP2 + INGP*NSCT
532:         if ( NTEMP.lt.NTEMP2 ) then
533:           NNLAST = IPOINT(NNLAST,INGP*NSCT+NSCT*(MAXSD+NUS)*3)
534:         else
535:           NTEMP2 = NTEMP2 + NSCT*(MAXSD+NUS)*3
536:           if ( NTEMP.lt.NTEMP2 ) then
537:             NNLAST = NNLAST + NSCT*(MAXSD+NUS)*3
538:           end if
539:         end if
540:         NTEMP = NNLAST - ILIMIT
541: C
542: C
543:       if ( NTEMP.gt.0 ) then
544:         write(IMG,7200) NTEMP
545: 7200       format('XXX NOT ENOUGH MEMORY FOR ',
546: &              ' DDX-CROSS SECTION TREATMENT'//10X,
547: &              ' SHORTAGE BY ',I10,' WORDS !'//)
548:
549:         call PRSTOP( 1,
550: &              'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.' )
551:         stop 999
552:       end if
553:     end if
554: C
555: C
556: C
557:       if ( ISTAT.gt.0 ) then
558:         IBUF = ISPOR + 1
559:         write(I0,7220) IBUF
560: 7220       format('      STARTING POSITION OF LEGENDRE COEFFICIENTS',
561: &              ' OR ENERGY-ANGLE DISTRIBUTION  :',I12,
562: &              ' (SAVED FOR NEXT EVENT ESTIMATOR ETC.)'//)
563:       end if
564: C
565: C
566:       NTEMP = NLAST - ILIMIT
567:       if ( NTEMP.gt.0 ) then
568:         write(IMG,7240) NTEMP
569: 7240       format(1X,'XXX MEMORY INSUFICIENT FOR PERMANENT CROSS-',
570: &              ' SECTION STORAGE BY ',I10,' WORDS !!' )
571:         call PRSTOP( 1, 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.'
572: &              )
573:         stop 999
574:       end if
575: C
576: C      WORKING MODULE OF CROSS SECTION PROCESSING SECTION !!!
577: C
578:       call JNPUP( SIGT, NSIG, IMEND, JEIGN, JFISS, JNEUT, JPHOT, JSEP )
579: C
580: C
581: C *** INTERFACE TO PARTS OF THE PROGRAM OTHER THAN CROSS SECTION
582: C      PROCESSING MODULES. *****
583: C
584: C
585: C

```

```

586:       if ( NGG.gt.0 ) then
587:         do 130 I = 1, NGG
588:           NSIG(INSIG+I) = NSIG(INSIG+I) + NDSNGP
589: 130       continue
590:         end if
591: C
592:       JDDX = IDDX
593:       JSTATX = ISTAT
594:       NGPX = NGP
595:       NGGX = NGG
596:       NTGX = NTG
597:       NDSPIX = NDSNGP
598:       NDSGX = NDSNGG
599:       NDSTX = NDSNGP + NDSNGG
600:       NUSX = NUS
601:       NSCTX = NSCT
602:       NCOEFX = NCOEF
603:       NMEDX = NMED
604:       LRSGX = IRSG + 1
605:       LANGX = IANG + 1
606:       LNGSX = INGS + 1
607:       LNSGX = INSG + 1
608:       LDELX = IDEL + 1
609:       LNNNX = INNN + 1
610:       LNNGX = IGGG + 1
611:       LSTOTX = ISTART + 1
612:       LSXCX = ISCCOG + 1
613:       LSNUFX = INUFIS + 1
614:       LSNAPX = INABOG + 1
615:       LSGPX = IGABOG + 1
616:       LSNFPX = IFPOR + 1
617:       LSNUSX = INUS + 1
618:       LS2DPX = IS2DP + 1
619:       LS2PPX = IS2PP + 1
620:       LSGPBX = IPNGP + 1
621:       LSSCBX = IPSPOG + 1
622:       LSDDBX = IPRBNG + 1
623:       LSANGX = ISCANG + 1
624:       LSGPPX = IFNGP + 1
625:       LSSCPX = IFSPOG + 1
626:       LSPORT = ISPORT + 1
627: C
628: C
629:       if ( JXTAPE.gt.0 ) then
630:         call XSTAPE( 2, SIGT, NSIG, JXTAPE, TITLE, LSTRT, ILIMIT, JADJM,
631: &              JDDX, JSTATX, NGPX, NGGX, NTGX, NDSPIX, NDSGX, NDSTX,
632: &              NUSX, NSCTX, NCOEFX, NMEDX, LRSGX, LANGX, LNSGX, LNSGX,
633: &              LDELX, LNNNX, LNNGX, LSTOTX, LSXCX, LSNUFX, LSNAPX,
634: &              LSGPX, LSNFPX, LSNUSX, LS2DPX, LS2PPX, LSGPBX, LSSCBX,
635: &              LSDDBX, LSANGX, LSGPPX, LSSCPX, LSPORT, NLAST )
636:       end if
637:       ITAPEX = JXTAPE
638:       go to 150
639: C
640: C
641: C
642: C      ..... READ IN ALREADY PROCESSED CROSS SECTION DATA (IXTAPE<0) .....
643: C
644: C
645: C
646: 140 IXTAP = IABS(IXTAPE)
647:       call XSTAPE( 1, SIGT, NSIG, IXTAP, TITLE, LSTRT, ILIMIT, JADJM,
648: &              JDDX, JSTATX, NGPX, NGGX, NTGX, NDSPIX, NDSGX, NDSTX, NUSX,
649: &              NSCTX, NCOEFX, NMEDX, LRSGX, LANGX, LNSGX, LNSGX, LDELX,
650: &              LNNNX, LNNGX, LSTOTX, LSXCX, LSNUFX, LSNAPX, LSGPX,

```

src/gmvp/xsecin.f

```

651:      &      LSNFPX, LSNUSX, LS2DPX, LS2PPX, LSGPBX, LSSCBX, LSDDBX,
652:      &      LSGX, LSGPPX, LSSCPX, LSPORT, NLAST )
653:      ITAPEX = IXTAP
654: C
655: C
656: C
657:      if ( JDDX.ne.IDDX ) then
658:      write(IMG,7260)
659: 7260      format('1',
660:      &      ' CROSS SECTION TYPE IDDX IN IXTAPE IS NOT EQUAL TO',
661:      &      ' INPUT IDDX')
662:      call CNTERR( 'FATAL ' )
663:      end if
664: C
665: C
666: C
667:      if ( IADJ.eq.1.and.NGG.ne.0 ) then
668:      if ( NMEDX-NMED+NGPX-NGG+NTGX-NTG.eq.0 ) then
669:      go to 150
670:      else
671:      write(I0,7280) NMED, NMEDX, NGG, NGPX, NTG, NTGX
672:      end if
673:      else
674:      if ( NMEDX-NMED+NGPX-NGP+NTGX-NTG.eq.0 ) then
675:      go to 150
676:      else
677:      write(I0,7300) NMED, NMEDX, NGG, NGGX, NTG, NTGX
678:      end if
679:      end if
680: C
681:      call PRSTOP( 1, 'CROSS SECTION ERROR.' )
682:      stop 888
683: C
684: 7280 format('      IN A COMBINED ADJOINT PROBLEM USING '
685:      &      'A PROCESSED CROSS SECTION FILE'//
686:      &      '      NMED IN INPUT = ',I8/
687:      &      '      NMEDX IN XSEC = ',I8/
688:      &      '      NGG      = ',I8,' NGPX      = ',I8/
689:      &      '      NTG      = ',I8,' NTGX      = ',I8/)
690: 7300 format('      NMED IN INPUT = ',I8,' NMED IN XSEC = ',I8/
691:      &      '      NGP      = ',I8,' NGP      = ',I8/
692:      &      '      NTG      = ',I8,' NTG      = ',I8/)
693: C
694: C
695: C
696: 150 LAST      = NLAST + 1
697: C
698: C      WRITE(I0,1180)
699: C1180 FORMAT(' ***** XSEC MODULE END *****')
700: C
701: C/#IF PARA( PVM MPI )
702: C
703: CCC      if ( ITASK0.lt.0.and.NTASK.gt.1 ) then
704: *      if ( IDTASK.eq.1.and.NTASK.gt.1 ) then
705: C
706: C      .... send sizes & pointers to subtask in PVM mode.
707: C
708: *      call TOKEI( T000, 0 )
709: *      ISEND(1)      = JDDX
710: *      ISEND(2)      = JSTATX
711: *      ISEND(3)      = NGPX
712: *      ISEND(4)      = NGGX
713: *      ISEND(5)      = NTGX
714: *      ISEND(6)      = NDSPX
715: *      ISEND(7)      = NDSGX

```

```

716: *      ISEND(8)      = NDSTX
717: *      ISEND(9)      = NUSX
718: *      ISEND(10)     = NSCTX
719: *      ISEND(11)     = NCOEFX
720: *      ISEND(12)     = NMEDX
721: *      ISEND(13)     = LRSGX
722: *      ISEND(14)     = LANGX
723: *      ISEND(15)     = LNSGX
724: *      ISEND(16)     = LNSGX
725: *      ISEND(17)     = LDELX
726: *      ISEND(18)     = LNNNX
727: *      ISEND(19)     = LNNGX
728: *      ISEND(20)     = LSTOTX
729: *      ISEND(21)     = LSSCX
730: *      ISEND(22)     = LSNUFX
731: *      ISEND(23)     = LSNAPX
732: *      ISEND(24)     = LSGPX
733: *      ISEND(25)     = LSNFPX
734: *      ISEND(26)     = LSNUSX
735: *      ISEND(27)     = LS2DPX
736: *      ISEND(28)     = LS2PPX
737: *      ISEND(29)     = LSGPBX
738: *      ISEND(30)     = LSSCBX
739: *      ISEND(31)     = LSDDBX
740: *      ISEND(32)     = LSGX
741: *      ISEND(33)     = LSGPPX
742: *      ISEND(34)     = LSSCPX
743: *      ISEND(35)     = LSPORT
744:
745: *      LSTRT      = LRSGX
746: *      NXDATA     = LAST - LSTRT
747: *      ISEND(36)   = LSTRT
748: *      ISEND(37)   = NXDATA
749:
750: *      NSEND      = 37
751:
752: *      it0 = 1
753: *      call MVPCOM_BCAST_I4( IT0, 1, ISEND, NSEND, INFO )
754:
755: *      MSGNUM      = 123
756: *      NBLOCK      = 2*15
757: C
758: C      .... send cross sections to subtask in PVM mode.
759: C
760: C
761: *      do 160 K = 1, NXDATA, NBLOCK
762: *      L1      = LSTRT + K - 1
763: *      K2      = MIN(NXDATA,K+NBLOCK-1)
764: *      NN      = K2 - K + 1
765: *      MSGNUM   = MSGNUM + 1
766: *      it0 = 1
767: *      call mvpcom_bcast_i4( it0, MSGNUM, NN, 1, INFO )
768: *      call mvpcom_bcast_i4( it0, MSGNUM, NSIG(L1), NN, INFO )
769: * 160      continue
770:
771: *      call TOKEI( T111, 0 )
772:
773: *      write(I0,'(1x,a,e12.5/)' )
774: *      &      '=== Elapsed time to send cross section to subtasks : ',
775: *      &      T111 - T000
776: *      end if
777: C/#ENDIF
778:      return
779:      end

```

src/gmvp/xsgmvp.f

```

1:      subroutine XSGMVP( A,      IA,      DA,      LAST,      LIMIT,      LFKAI,
2:      &                  LIDMAT,NMAT,      NGP1,      NGP2 )
3: C=====
4: C purpose: To translate GMVP specific cross section & material input
5: C           to those of old-style ( MORSE-DD like ).
6: C=====
7:      real      A(*)
8:      integer IA(*)
9:      real*8 DA(*)
10: C
11:      include '../shared/INC/_IOUNIT'
12:      include 'INC/_FLAGS'
13: C
14: C-----
15: C Input description : < Default >
16: C-----
17: C
18: C $CROSS SECTION      .... header
19: C
20: C TITLE( title_string )      <blank>
21: C
22: C Title string up to 72 characters.
23: C
24: C TYPE( PL or DDX )      <PL>
25: C
26: C cross section type ( Legendre expansion or DDX )
27: C
28: C PLFORM( ANISN or ANISN-MAIL or KENO-IV or DTF or FIDO ... )
29: C      <ANISN>
30: C
31: C format of Legendre expansion xsec.
32: C
33: C NGP1( ngp1 [nds1] )      <NGP1 in /SIZES/ if defined else error>
34: C
35: C number of energy group for primary particle
36: C (and down scatter group if necessary )
37: C
38: C NGP2( ngp2 [nds2] )      <NGP2 in /SIZES/ if defined else error>
39: C
40: C number of energy group for secondary particle
41: C (and down scatter group if necessary )
42: C
43: C ITBL( itbl )      < NGP1 + NGP2 + 3 >
44: C
45: C Cross section table length ( default = ngroup + 3 )
46: C
47: C IHT( iht )      < 3 >
48: C
49: C position of total scatter in xsec table.
50: C
51: C ISGG( isgg )      < itbl - nds1 + 1 >
52: C
53: C position of in group scatter in xsec table.
54: C
55: C INGP( ingp )      <NGP1+NGP2>
56: C
57: C Number of energy groups of cross sections stored in file.
58: C Necessary to specify if the number of group is not equal to
59: C NGP1 + NGP2.
60: C
61: C NPL ( npl )      < 0 >
62: C
63: C Order of Legendre expansion
64: C
65: C INCELM( incelm )      < 0 >

```

```

66: C
67: C Element ID's of Legendre coefficients are incremented
68: C by INCELM from 0'th order.
69: C
70: C NSCT( nsct )      < (NPL + 1)/2 for PL, 20 for DDX >
71: C
72: C Number of scattering angles in Legendre expansion mode.
73: C
74: C PRFLAG( irdsg istr ifmu imom iprin ipun )      < all 0 >
75: C
76: C Printout flags.
77: C
78: C INPUT( ixtape )      < 1 for PL mode , none for DDX >
79: C
80: C Cross section input I/O unit for PL mode.
81: C Negative value means input of GMVP processed data
82: C from I/O unit -ixtape..
83: C
84: C OUTPUT( jxtape )      < 0 >
85: C
86: C Output unit of processed cross section.
87: C
88: C MUS( mus )      < -1 to 1 divided into NSCT intervals >
89: C
90: C Cosine boundary of DDX cross section.
91: C
92: C XLAST( id sigt sigpr siga sigf nu-sigf
93: C        id2 sigt2 ..... )      <none>
94: C
95: C Cross sections of nuclide with cross section-id of 'id'.
96: C These are cross sections of the last group for DDX
97: C mode.
98: C
99: C
100: C Input data above must be placed before the following data.
101: C
102: C
103: C & IDMAT( matid ) COMPOSITION( id density id density ... )
104: C [ FKAI( fission spectrum data ) ]
105: C
106: C Composition of material with material id 'matid'.
107: C 'Id' is nuclide or material id in cross section input.
108: C
109: C Fission spectrum data can be input for fission problem.
110: C
111: C Input these data for each material.
112: C
113: C #
114: C any input data copied to I/O unit IUGL as input
115: C ( mainly for DTF or FIDO type crosssection input )
116: C
117: C $END CROSS SECTION
118: C
119: C-----
120: C
121: C ... local data ....
122: C
123: C character*72 XTITLE, LINE
124: C equivalence(XTITLE,LINE)
125: C
126: C character*64 STR
127: C character*16 PLFORM
128: C character*4 TYPE
129: C
130: C integer ITEMP(6)

```


src/gmvp/xsgmvp.f

```

131: C
132: C .....
133: C
134:       include '../shared/INC/_WORDL'
135: C
136: C ... statement function to translate integer array index to real*8
137: C
138:       LDBLE(L)      = L/MWORD + MWORD - 1
139: C
140: C-----
141:       LASTO = LAST
142:       call RESET
143: C
144:       call RWIND( IUG1 )
145: C
146:       XTITLE = ' '
147:       IDBX   = 0
148:       PLFORM = 'ANISN'
149:       NGP10  = 0
150:       NDS10  = 0
151:       NGP20  = 0
152:       NDS20  = 0
153:       INGP   = 0
154:       ITBL   = 0
155:       IHT    = 0
156:       ISGG   = 0
157:       NPL    = 0
158:       NSCT   = 0
159:       INCELM = 0
160: C
161:       IRDSG  = 0
162:       ISTR   = 0
163:       IFMU   = 0
164:       IMOM   = 0
165:       IPRIN  = 0
166:       IPUN   = 0
167: C
168:       IXTAPE = -999
169:       JXTAPE = 0
170: C
171:       LMUS   = 0
172:       NMUS   = 0
173:       LXLAST = 0
174:       NXLAST = 0
175: C
176:       NMED   = 0
177:       IDMAT  = 0
178:       NCOMP  = 0
179:       NMIX   = 0
180: C
181:       JLDATA = 0
182:       JJFKAI = 0
183: C
184:       call LABEL( IPR, 'DATA INPUT MONITORING' )
185: C
186: C
187: 100 call FREADS( STR, NL, IEND )
188:       if ( IEND.ne.0 ) go to 999
189: C
190: C
191: C
192:       if ( STR(:NL).eq.'TITLE' ) then
193: 110       call CHREAD( STR(:NL), XTITLE, ' ', NTLL, ITERM, IEND )
194:       if ( IEND.ne.0 ) go to 999
195: C

```

```

196:       if ( ITERM.eq.0 ) go to 110
197: C       write(ipr,*) 'XXX Cross section title is invalid.'
198: C       call cnterr('FATAL')
199: C       goto 9999
200: C     endif
201: C
202: C
203:       write(IPR,7000) 'TITLE', XTITLE
204: 7000 format(5X,A,' <',A,'>')
205: 7020 format(5X,'DATA : ',A/(10X,8I8))
206: 7040 format(5X,'DATA : ',A/(10X,5E13.5))
207: 7060 format(5X,'DATA : ',A/(10X,5D13.5))
208: C
209:       else if ( STR(:NL).eq.'TYPE' ) then
210: C
211: 120       call CHREAD( 'TYPE', STR, ' ', NLL, ITERM, IEND )
212:       if ( IEND.ne.0 ) go to 999
213:       if ( NLL.gt.0 ) then
214:         if ( STR(:NLL).eq.'PL' ) then
215:           IDDX = 0
216:         else if ( STR(:NLL).eq.'DDX' ) then
217:           IDDX = 1
218:         else
219:           write(IMG,*) 'XXX(XSGMVP) INVALID CROSS SECTION TYPE. <',
220: & STR(:NLL), '>'
221:           call CNTERR( 'FATAL' )
222:         end if
223:         write(IPR,7000) 'TYPE', STR(:NLL)
224:       end if
225:       if ( ITERM.eq.0 ) go to 120
226: C
227: C
228: C
229:       else if ( STR(:NL).eq.'PLFORM' ) then
230: 130       PLFORM = ' '
231:       call CHREAD( 'PLFORM', PLFORM, ' ', NLL, ITERM, IEND )
232: C
233:       if ( IEND.ne.0 ) go to 999
234:       write(IPR,7000) 'PLFORM', PLFORM
235: C
236:       if ( ITERM.eq.0 ) then
237:         write(IMG,*) 'XXX(XSGMVP) EXTRA DATA FOR "PLFORM"'
238:         call CNTERR( 'FATAL' )
239:         go to 9000
240:       end if
241: C
242: C
243: C
244:       else if ( STR(:NL).eq.'NGP1' ) then
245:         call I4READ( 'NGP1', ITEMP, NA, -2, IEND )
246:         if ( IEND.ne.0 ) go to 999
247: C
248:         if ( NA.ge.1 ) NGP10 = ITEMP(1)
249:         if ( NA.ge.2 ) NDS10 = ITEMP(2)
250:         write(IPR,7020) 'NGP1', (ITEMP(I),I=1,NA)
251: C
252: C
253: C
254:       else if ( STR(:NL).eq.'NGP2' ) then
255:         call I4READ( 'NGP2', ITEMP, NA, -2, IEND )
256:         if ( IEND.ne.0 ) go to 999
257: C
258:         if ( NA.ge.1 ) NGP20 = ITEMP(1)
259:         if ( NA.ge.2 ) NDS20 = ITEMP(2)
260:         write(IPR,7020) 'NGP2', (ITEMP(I),I=1,NA)

```

src/gmvp/xsgmvp.f

```

261: C
262: C
263:     else if ( STR(:NL).eq.'ITBL' ) then
264:         call I4READ( 'ITBL', ITBL, NA, 1, IEND )
265:         if ( IEND.ne.0 ) go to 999
266: C
267: C
268:     else if ( STR(:NL).eq.'IHT' ) then
269: C
270:         call I4READ( 'IHT', IHT, NA, 1, IEND )
271:         if ( IEND.ne.0 ) go to 999
272: C
273: C
274:     else if ( STR(:NL).eq.'ISGG' ) then
275: C
276:         call I4READ( 'ISGG', ISGG, NA, 1, IEND )
277:         if ( IEND.ne.0 ) go to 999
278: C
279: C
280:     else if ( STR(:NL).eq.'INGP' ) then
281:         call I4READ( 'INGP', INGP, NA, 1, IEND )
282:         if ( IEND.ne.0 ) go to 999
283: C
284: C
285:     else if ( STR(:NL).eq.'NPL' ) then
286: C
287:         call I4READ( 'NPL', NPL, NA, 1, IEND )
288:         if ( IEND.ne.0 ) go to 999
289: C
290: C
291:     else if ( STR(:NL).eq.'INCELM' ) then
292: C
293:         call I4READ( 'INCELM', INCELM, NA, 1, IEND )
294:         if ( IEND.ne.0 ) go to 999
295: C
296: C
297:     else if ( STR(:NL).eq.'NSCT' ) then
298: C
299:         call I4READ( 'NSCT', NSCT, NA, 1, IEND )
300:         if ( IEND.ne.0 ) go to 999
301: C
302: C
303:     else if ( STR(:NL).eq.'PRFLAG' ) then
304: C
305:         call I4READ( 'PRFLAG', ITEMP, NA, 6, IEND )
306:         if ( IEND.ne.0 ) go to 999
307: C
308:         if ( NA.ge.1 ) IRDSG = ITEMP(1)
309:         if ( NA.ge.2 ) ISTR = ITEMP(2)
310:         if ( NA.ge.3 ) IFMU = ITEMP(3)
311:         if ( NA.ge.4 ) IMOM = ITEMP(4)
312:         if ( NA.ge.5 ) IPRIN = ITEMP(5)
313:         if ( NA.ge.6 ) IPUN = ITEMP(6)
314: C
315:         write(IPR,7020) 'PRFLAG', (ITEMP(I),I=1,NA)
316: C
317:     else if ( STR(:NL).eq.'INPUT' ) then
318: C
319:         call I4READ( 'INPUT', IXTAPE, NA, 1, IEND )
320:         if ( IEND.ne.0 ) go to 999
321: C
322: C
323:     else if ( STR(:NL).eq.'OUTPUT' ) then
324: C
325:         call I4READ( 'OUTPUT', JXTAPE, NA, 1, IEND )

```

```

326:         if ( IEND.ne.0 ) go to 999
327: C
328: C
329:     else if ( STR(:NL).eq.'MUS' ) then
330: C
331:         if ( NSCT.ne.0 ) then
332:             NMUS = NSCT + 1
333:         else
334:             NMUS = 0
335:         end if
336:         call ARAYIN( 'MUS', A, ICALL, 'R4', LMUS, ICK, JDEBG, LAST, NMUS,
337: &                 ' ' )
338:         write(IPR,7040) 'MUS', (A(LMUS+I),I=0,NMUS-1)
339: C
340: C
341:     else if ( STR(:NL).eq.'XLAST' ) then
342: C
343:         call ARAYIN( 'XLAST', A, ICALL, 'R4', LXLAST, ICK, JDEBG, LAST,
344: &                 NXLAST, ' ' )
345: C
346:         write(IPR,7040) 'XLAST', (A(LXLAST+I),I=0,NXLAST-1)
347: C
348:         if ( MOD(NXLAST,6).ne.0 ) then
349:             write(IMG,*)
350:             &         'XXX(XSGMVP) number of the last-group cross section',
351:             &         'data "XLAST" is not valid. ',
352:             &         '(must be set of "ID ST SPR SC SF NSF")'
353:             call CNTERR( 'FATAL' )
354:             go to 9000
355:         end if
356: C
357: C
358: C
359:     else if ( STR(:NL).eq.'&' .or. STR(:NL).eq.'#'
360: &         .or. STR(:NL).eq.'$END' ) then
361: C
362:         if ( NMED.eq.0 ) then
363: C
364:             ... keep working area & data packet container for material
365:             composition input.
366: C
367: C
368:             ... save "last" pointer
369: C
370:             call GTLAST( LASTSV )
371: C
372:             NWK = 128
373:             call KEEP( 'work', LWK, NWK, 'R8', LAST, JDEBG )
374:             NWK2 = 1024
375:             call KEEP( 'work2', LWK2, NWK2, 'R4', LAST, JDEBG )
376: C
377:             LDWK = LDBLE(LWK)
378: C
379:             call REMAIN( 'MATERIAL COMPOSITION', NLTEMP, 'I4D', LAST )
380:             call KEEP( 'MATERIAL', LMATSP, NLTEMP, 'I4D', LAST, JDEBG )
381: C
382:             call PACK00( A(LMATSP), 'MATERIAL COMPOSITION', NLTEMP, IRET
383: &                 )
384:             if ( IRET.ne.0 ) go to 9000
385: C
386: C
387:             ... termination process for each material input ...
388: C
389:         else if ( NMED.gt.0 ) then
390:             if ( IDMAT.eq.0 ) then
391:                 IDMAT = NMED

```

src/gmvp/xsgmvp.f

```

391:         write(IMG,*) '!!!(XSGMVP) NO MATERIAL ID INPUT.',
392:         &         ' TREATED AS MATERIAL <', IDMAT, '> '
393:         call CNTERR( 'WARNING' )
394:     end if
395:     if ( NCOMP.eq.0 ) then
396:         write(IMG,*)
397:         &         'XXX(XSGMVP) NO MATERIAL COMPOSITION DATA FOR ',
398:         &         ' MATERIAL <', IDMAT, '> '
399:         call CNTERR( 'FATAL' )
400:     end if
401: C
402: C ... pack material ID , ncomp & nfkai
403: C
404:         ITEMP(1) = IDMAT
405:         ITEMP(2) = NCOMP
406:         ITEMP(3) = NFKAI
407:         call PACKND( A(LMATSP), 'MAT-ID', 'I4', ITEMP, 3, IRET )
408:         if ( IRET.ne.0 ) go to 9000
409: C
410: C ... pack material composition
411: C
412:         call PACKND( A(LMATSP), 'COMPOSITION', 'R8', DA(LDWK),
413:         &         NCOMP, IRET )
414:         if ( IRET.ne.0 ) go to 9000
415: C
416: C ... pack fission spectrum ...
417: C
418:         if ( NFKAI.gt.0 ) then
419:             call PACKND( A(LMATSP), 'FKAI', 'R4', A(LWK2), NFKAI,
420:             &             IRET )
421:             if ( IRET.ne.0 ) go to 9000
422:         end if
423:     end if
424:
425:     if ( STR(:NL).eq.'&' ) then
426:         NMED = NMED + 1
427:         NCOMP = 0
428:         NFKAI = 0
429:     else
430:         if ( STR(:NL).eq. '#' ) JLDATA = 1
431:         go to 140
432:     end if
433: C
434: C
435:     else if ( STR(:NL).eq.'IDMAT' ) then
436:         if ( NMED.eq.0 ) then
437:             write(IMG,*)
438:             &             'XXX(XSGMVP) position of "IDMAT" data is invalid'
439:             call CNTERR( 'FATAL' )
440:             call DMREAD( 'IDMAT', NA, 1, IRET )
441:         else
442:             call I4READ( 'IDMAT', IDMAT, NA, 1, IEND )
443:         end if
444: C
445: C
446:     else if ( STR(:6).eq.'COMPOS' ) then
447:         if ( NMED.eq.0 ) then
448:             write(IMG,*)
449:             &             'XXX(XSGMVP) position of "COMPOSITION" data is invalid'
450:             call CNTERR( 'FATAL' )
451:             call DMREAD( 'COMPOSITION', NA, -NWK, IRET )
452:         else
453:             call R8READ( 'COMPOSITION', DA(LDWK), NA, -NWK, IEND )
454:             if ( IEND.ne.0 ) go to 999
455:             if ( NA.gt.NWK ) then

```

```

456:         write(IMG,*)
457:         &         'XXX(XSGMVP) Too many composition data (max=', NWK, ')'
458:         call CNTERR( 'FATAL' )
459:     else
460:         NCOMP = NA
461:         if ( MOD(NCOMP,2).ne.0 ) then
462:             write(IMG,*) 'XXX(XSGMVP) material composition data ',
463:             &             'is not pair of "XSEC-ID" & "DENSITY".'
464:             call CNTERR( 'FATAL' )
465:             DA(LDWK+NCOMP) = 0.0
466:             NCOMP = NCOMP + 1
467:         end if
468:         NMIX = NMIX + NCOMP/2
469:     end if
470:     write(IPR,7080) 'COMPOSITION',
471:     &         (NINT(DA(LDWK+I)),DA(LDWK+I+1),I=0,NCOMP-1,2)
472: 7080 format(5X,A,': ID DENSITY .../(10X,5(I8,E12.4)))
473:     end if
474: C
475: C
476:     else if ( STR(:NL).eq.'FKAI' ) then
477:         if ( NMED.eq.0 ) then
478:             write(IMG,*)
479:             &             'XXX(XSGMVP) position of "FKAI" data is invalid'
480:             call CNTERR( 'FATAL' )
481:             call DMREAD( 'FKAI', NA, -NWK, IRET )
482:         else
483:             call R4READ( 'FKAI', A(LWK2), NFKAI, -NWK2, IEND )
484:             if ( IEND.ne.0 ) go to 999
485:             if ( NFKAI.gt.NWK2 ) then
486:                 write(IMG,*)
487:                 &                 'XXX(XSGMVP) Too many "FKAI" data (max=', NWK2, ')'
488:                 call CNTERR( 'FATAL' )
489:             end if
490:             if ( JFISS.eq.0.and.NFKAI.gt.0 ) then
491:                 write(IMG,*) '!!!(XSGMVP) FISSION SPECTRUM DATA FOR',
492:                 &                 ' NON-FISSION PROBLEM ARE MEANINGLESS.'
493:             end if
494:             if ( NFKAI.gt.0 ) JJFKAI = 1
495:         end if
496: C
497: C
498:     else
499:         write(IMG,*)
500:         &         'XXX(XSGMVP) UNRECOGNIZABLE DATA <', STR(:NL), '> FOUND.'
501:         call CNTERR( 'FATAL' )
502:         go to 9000
503:     end if
504: C
505:     go to 100
506: C
507: C -----
508: C ... make MORSE-DD type input data ...
509: C -----
510: C
511: 140 continue
512: C
513: C ... release unused memory ...
514: C
515:     call PCTCLS( A(LMATSP), 'MAT', NLTEMP, NMATSP, IRET )
516:     call RESIZE( 'MATERIAL COMPOSITION', LMATSP, NMATSP, 'I4D', LAST )
517: C
518: C
519:     if ( NMAT.eq.0 ) then
520:         NMAT = NMED

```

src/gmvp/xsgmvp.f

```

521:     else if ( NMAT.ne.NMED ) then
522:       write(IMG,*)
523:       & 'XXX(XSGMVP) YOUR INPUT FOR NUMBER OF MATERIALS (NMAT)',
524:       & ' CONFLICTS WITH THAT OF IN $CROSS SECTION BLOCK (NMED=',
525:       & NMED, ').' ,
526:       & ' REMOVE "NMAT" INPUT AND MODIFY YOUR INPUT IF NECESSARY.'
527:       call CNTERR( 'FATAL' )
528:     end if
529: C
530: C
531: C
532: C
533: IDTF = 0
534: if ( DDX.eq.0 ) then
535:   if ( PLFORM.ne.'ANISN'.and.PLFORM.ne.'ANISN-MAIL'
536:   & .and.PLFORM.ne.'DTF'.and.PLFORM.ne.'KENO-IV'
537:   & .and.PLFORM.ne.'FIDO' ) then
538:     write(IMG,*)
539:     & 'XXX(XSGMVP) invalid format for PL type cross section <',
540:     & PLFORM, '>'
541:     call CNTERR( 'FATAL' )
542:   end if
543: C
544: C
545: C
546:   ... default I/O unit of PL cross section = 1 ...
547:   if ( IXTAPE.eq.-999 ) then
548:     if ( PLFORM.ne.'DTF' ) then
549:       IXTAPE = 1
550:     else
551:       IXTAPE = 0
552:     end if
553:     if ( PLFORM.eq.'DTF' ) then
554:       IDTF = 1
555:       IXTAPE = 0
556:     else
557:       IDTF = 0
558:       IXTAPE = 1
559:     end if
560:   end if
561: C
562:   if ( PLFORM.eq.'ANISN-MAIL' ) then
563:     IXTAPE = SIGN(1000+ABS(IXTAPE),IXTAPE)
564:   else if ( PLFORM.eq.'DTF' ) then
565:     IDTF = 1
566:     IXTAPE = 0
567:   else if ( PLFORM.eq.'FIDO' ) then
568:     IDTF = 0
569:     IXTAPE = 0
570:   end if
571: C
572:   if ( PLFORM.eq.'KENO-IV' ) then
573:     if ( NGP20.gt.0 ) then
574:       write(IMG,*)
575:       & 'XXX(XSGMVP) SECONDARY PARTICLE TREATMENT ',
576:       & 'IMPOSSIBLE FOR KENO-IV TYPE LIBRARY'
577:       call CNTERR( 'FATAL' )
578:     end if
579:     if ( ITBL.ne.0 .or. ISGG.ne.0 ) then
580:       write(IMG,*)
581:       & '!!!XSGMVP) IGNORE INPUT FOR CROSS SECTION TABLE LENGTH & ',
582:       & 'SELF SCATTERING POSITION FOR KENO-IV TYPE LIBRARY.'
583:       call CNTERR( 'WARNING' )
584:     end if
585:   end if

```

```

586: C
587: C ... DDX ..
588: C
589:   else
590: C
591:   ... default I/O unit of DDX cross section = 40 ...
592: C
593:   if ( IXTAPE.eq.-999 ) IXTAPE = 40
594:   end if
595: C
596: C
597: C
598: C ... For photon-only problems, set NGP1, NGP2 to NGP2, 0, respectively.
599: C
600: C comment: The same process is performed in CHKNUM but it must be
601: C done here for phton-only problems.
602: C
603:   if ( JNEUT.eq.0.and.JPHOT.ne.0 ) then
604:     NGP1 = NGP2
605:     NGP2 = 0
606:     write(IMG, '(/a,a,i3,a,a)') '!!! NGP1, NGP2 is set to NGP2,',
607:     & ' 0, respectively. (NGP1=',NGP1,') because you are solving',
608:     & ' photon-only problem !!!'
609:     call CNTERR( 'WARNING' )
610:   end if
611: C
612: C ... check consistency between NGP1 in control data and NGP1 in CROSS
613: C SECTION block.
614: C
615:   if ( NGP1.ne.0.and.NGP10.ne.0 ) then
616:     if ( NGP1.ne.NGP10 ) then
617:       write(IMG,*)
618:       & 'XXX(XSGMVP) "NGP1" value input in Cross section block',
619:       & ' conflicts with already input "NGP1".'
620:       call CNTERR( 'FATAL' )
621:     end if
622:   else if ( NGP1.eq.0.and.NGP10.ne.0 ) then
623:     NGP1 = NGP10
624:   else if ( NGP1.ne.0.and.NGP10.eq.0 ) then
625:     NGP10 = NGP1
626:   else
627:     write(IMG,*) 'XXX(XSGMVP) no input for "NGP1" ',
628:     & ' (number of energy group for primary particle)'
629:     call CNTERR( 'FATAL' )
630:   end if
631: C
632: C ... For neutron-photon coupled problems, check consistency between
633: C NGP2 in control data and NGP2 in CROSS SECTION block.
634: C
635:   if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
636: C
637:   if ( NGP2.ne.0.and.NGP20.ne.0 ) then
638:     if ( NGP2.ne.NGP20 ) then
639:       write(IMG,*)
640:       & 'XXX(XSGMVP) "NGP2" value input in Cross section block',
641:       & ' conflicts with already input "NGP2".'
642:       call CNTERR( 'FATAL' )
643:     end if
644:   else if ( NGP2.eq.0.and.NGP20.ne.0 ) then
645:     NGP2 = NGP20
646:   else if ( NGP2.ne.0.and.NGP20.eq.0 ) then
647:     NGP20 = NGP2
648:   else
649:     write(IMG,*) 'XXX(XSGMVP) No input for "NGP2"',
650:     & ' (number of energy group for secondary particle)'

```

src/gmvp/xsgmvp.f

```

651:      call CNTERR( 'FATAL' )
652:      end if
653:
654:  end if
655:
656:      if ( NDS10.eq.0 ) NDS10 = NGP1
657:      if ( NDS20.eq.0 ) NDS20 = NGP2
658:      NTG      = NGP1 + NGP2
659:      if ( INGP.eq.0 ) INGP  = NTG
660: C
661: C
662: C
663:      if ( IHT.eq.0 ) IHT      = 3
664:      if ( ITBL.eq.0 ) ITBL     = NGP1 + NGP2 + IHT
665:      if ( ISGG.eq.0 ) ISGG     = ITBL - NDS10 + 1
666:
667:      NCOEF      = NPL + 1
668:      if ( IDDX.ne.0 ) NCOEF     = 1
669:
670:      if ( IDDX.eq.0.and.NSCT.eq.0 ) NSCT = NCOEF/2
671:
672:      if ( IDDX.ne.0 ) then
673:      if ( NMUS.eq.0.and.NSCT.eq.0 ) then
674:          NSCT      = 20
675:          NMUS      = NSCT + 1
676:      else if ( NSCT.ne.0.and.NMUS.eq.0 ) then
677:          NMUS      = NSCT + 1
678:      else if ( NSCT.eq.0.and.NMUS.ne.0 ) then
679:          NSCT      = NMUS - 1
680:      else if ( NMUS.ne.NSCT+1 ) then
681:          write(IMG,*)
682:      &      'XXX(XSGMVP) NUMBER OF DDX COSINE INTERVALS AND ',
683:      &      ' THAT OF COSINE BOUNDARY INPUT VALUES CONFLICTS ',
684:      &      NSCT + 1, ' REQUIRED BUT ', NMUS, ' INPUT.'
685:      call CNTERR( 'FATAL' )
686:      NMUS      = NSCT + 1
687:      end if
688:  end if
689:
690:  ISTAT      = 0
691:  if ( JPTDT.ne.0 ) ISTAT = 1
692:
693: C
694: C
695: C
696: C
697:  call KEEP( 'IDMAT', LIDMAT, NMED, 'I4', LAST, JDBG )
698:  if ( JJFKAI.ne.0 ) then
699:      call KEVP(A(1),'FKAI', LFKAI, NMED*NTG, 'R4', LAST, 0.0, JDBG )
700:  end if
701: C
702:  if ( JFISS.ne.0.and.JJFKAI.eq.0.and.PLFORM.ne.'KENO-IV' ) then
703:      write(IMG,*)
704:      &      '!!!XSGMVP NO FISSION SPECTRUM INPUT IN $CROSS SECTION',
705:      &      ' BLOCK FOR FISSION PROBLEM.'
706:      write(IMG,*)
707:      &      ' FISSION SPECTRUM INPUT EXPECTED IN AOTHER PLACE IN ',
708:      &      ' INPUT STREAM.'
709:      call CNTERR('WARNING')
710:  end if
711: C
712:  call REMAIN( 'ELEM', NLTEMP, 'I4', LAST )
713:  call KEEP( 'IELEM', LIELEM, NLTEMP, 'I4', LAST, JDBG )
714: C
715:  NELEM      = 0

```

```

716:      LDMAT      = LDBLE(LMATSP) - 1
717: C
718:  call PCTRW( A(LMATSP), 'MAT', IRET )
719:  do 180 I = 1, NMED
720:      call UNPKND( A(LMATSP), 'IDMAT', 'I4', ITEMP, 3, N2, IRET )
721:      if ( IRET.ne.0 ) call CNTERR( 'FATAL' )
722:      IDMAT      = ITEMP(1)
723:      NCOMP      = ITEMP(2)
724:      NFKAI      = ITEMP(3)
725: C
726:      IA(LIDMAT+I-1) = IDMAT
727:      call UNPKPT( A(LMATSP), 'COMPOSITION', 'R8', LCMP, ND, IRET )
728:      if ( IRET.ne.0 ) call CNTERR( 'FATAL' )
729: C
730:  ... make cross section element ID table
731: C
732:  do 160 K = 0, ND/2 - 1
733:      do 150 J = 1, NELEM
734:          if ( IA(LIELEM+J-1).eq.NINT(DA(LDMAT+LCMP+K*2)) ) go to
735:      &      160
736:      continue
737: C
738:      NELEM      = NELEM + 1
739:      if ( NELEM.gt.NLTEMP ) then
740:          write(IMG,'(1x,a/a,a,i10,a)')
741:      &      'XXX(XSGMVP) Too many cross section elements.',
742:      &      ' PLEASE INCREASE MEMORY SIZE.',
743:      &      ' (CURRENT SIZE ', LIMIT, ')')
744:      call PRSTOP( 1, 'MEMORY OVER' )
745:      stop 999
746:      end if
747:      IA(LIELEM+NELEM-1) = NINT(DA(LDMAT+LCMP+K*2))
748: 160  continue
749: C
750:  ... save fission spectrum if necessary
751: C
752:  if ( NFKAI.gt.0 ) then
753:      call UNPKPT( A(LMATSP), 'FKAI', 'R4', LLLKAI, ND2, IRET )
754:      if ( IRET.ne.0 ) call CNTERR( 'FATAL' )
755:      LLLKAI      = LMATSP - 1 + LLLKAI
756:      LL          = (I-1)*NTG
757:      do 170 J = 0, MIN(NTG,NFKAI) - 1
758:          A(LFKAI+LL+J) = A(LLLKAI+J)
759: 170  continue
760:      end if
761: 180  continue
762: C
763:  do 200 I = 1, NMED - 1
764:      do 190 J = I + 1, NMED
765:          if ( IA(LIDMAT+I-1).eq.IA(LIDMAT+J-1) ) then
766:              write(IMG,*)
767:      &      'XXX(XSGMVP) DUPLICATE MATERIAL ID !! :', J,
768:      &      ' ''TH ID & ', I, ' ''TH ID = ', IA(LIDMAT+I-1)
769:              call CNTERR( 'FATAL' )
770:          end if
771: 190  continue
772: 200  continue
773: C
774: C
775:      call RESIZE( 'IELEM', LIELEM, NELEM, 'I4', LAST )
776: C
777: C
778: C ... check elements whose ID s must be output as negative in data X-3
779: C
780: C

```

src/gmvp/xsgmvp.f

```

781:      if ( IDDX.ne.0.and.LXLAST.ne.0 ) then
782:        do 220 N = 1, NXLAST, 6
783:          L      = LXLAST + N - 1
784: C
785:          ID      = NINT(A(L))
786:          do 210 I = 0, NELEM - 1
787:            if ( ID.eq.ABS(IA(LIELEM+I)) ) then
788:              IA(LIELEM+I) = -ABS(IA(LIELEM+I))
789:            end if
790:          210 continue
791:        220 continue
792:      end if
793: C
794: C ... output DATA X-0 , X-1 & X-2
795: C
796: C
797:      write(IUG1,'(a)') XTITLE
798:      write(IUG1,'(8i8)') NGP10, NDS10, NGP20, NDS20, INGP, ITBL, ISCG,
799:      &      NMED, NELEM, NMIX, NCOEF, NSCT, ISTAT, IDDX, IHT
800: C
801:      write(IUG1,'(8i8)') IRDSG, ISTR, IFMU, IMOM, IPRIN, IPUN, IDTF,
802:      &      IXTAPE, JXTAPE
803: C
804:      if ( JXTAPE.ge.0 ) then
805: C
806: C
807: C .... DATA X-2.2
808: C
809:      if ( IDDX.ne.0 ) then
810:        if ( LMUS.ne.0 ) then
811:          write(IUG1,'(lp,5e14.5)') (A(LMUS+I),I=0,NSCT)
812:        else
813:          write(IUG1,'(lp,5e14.5)')
814:      &      (1.0-2.0*(I/REAL(NSCT)),I=0,NSCT)
815:        end if
816:      end if
817: C
818: C
819: C .... DATA X-3
820: C
821: C980823 if ( IDTF.eq.0 ) then
822:      if ( IXTAPE.gt.0 ) then
823:        do 230 I = 0, NELEM - 1
824:          KE      = IA(LIELEM+I)
825:          write(IUG1,'(8i8)') (KE+(N-1)*INCELM,N=1,NCOEF)
826:        230 continue
827:      end if
828: C
829: C .... DATA X-4
830: C
831:      if ( IDDX.ne.0.and.NXLAST.ne.0 ) then
832:        do 240 N = 1, NXLAST, 6
833:          write(IUG1,'(i7,lp,5e13.5)') ABS(NINT(A(LXLAST+N-1))),
834:      &      (A(LXLAST+N+I),I=0,4)
835:        240 continue
836:      end if
837: C
838: C .... DATA X-5
839: C
840:      LDMAT      = LDBLE(LMATSP) - 1
841: C
842:      call PCTRW( A(LMATSP), 'MAT', IRET )
843:      do 280 I = 1, NMED
844:        call UNPKND( A(LMATSP), 'IDMAT', 'I4', ITEMP, 3, N2, IRET )
845:        if ( IRET.ne.0 ) call CNTERR( 'FATAL' )

```

```

846:      IDMAT      = ITEMP(1)
847:      NCOMP      = ITEMP(2)
848:      NFKAI      = ITEMP(3)
849: C
850:      call UNPKPT( A(LMATSP), 'COMPOSITION', 'R8', LCMP, ND, IRET
851:      &
852:      if ( IRET.ne.0 ) call CNTERR( 'FATAL' )
853: C
854:      do 270 K = 0, ND/2 - 1
855:        IDE      = NINT(DA(LDMAT+LCMP+K*2))
856:        do 250 KE = 1, NELEM
857:          if ( IDE.eq.IA(LIELEM+KE-1) ) go to 260
858:        250 continue
859:        260 if ( K.eq.ND/2-1 ) KE = -KE
860:        RHO      = DA(LDMAT+LCMP+K*2+1)
861: Ccccccccccccc write(IUG1,'(i8,1x,i8,1x,e13.5)') IDMAT, KE, RHO
862: C
863: C ... output sequential media # ...
864: C
865:      write(IUG1,'(i8,1x,i8,1x,e13.5)') I, KE, RHO
866:      270 continue
867:      if ( NFKAI.gt.0 ) then
868:        call UNPKPT( A(LMATSP), 'FKAI', 'R4', LTMP, ND, IRET )
869:        if ( IRET.ne.0 ) call CNTERR( 'FATAL' )
870:      end if
871:      280 continue
872:    end if
873: C
874: C
875:      call RESET
876: C
877: C ... copy data lines between # and $END if any.
878: C
879:      if ( JLDATA.ne.0 ) then
880:      290 call GETLIN( IOIN, IPR, LINE, NCHAR, IEND )
881:        if ( IEND.ne.0 ) go to 999
882:        if ( LINE(1:4).ne.'$END' ) then
883:          write(IUG1,'(a)') LINE
884:          go to 290
885:        end if
886:        call RESET
887:      end if
888: C
889:      call RWIND( IUG1 )
890:      return
891: C
892: 999 write(IMG,*) 'XXX(XSGMVP) UNEXPECTED END OF INPUT DATA'
893:      go to 9000
894: C
895: C
896: 9000 call PRSTOP( 0, 'INVALID "$CROSS SECTION" INPUT' )
897:      stop 888
898:      end

```

src/gmvp/xstape.f

```

1: *VOCL TOTAL,SCALAR
2: subroutine XSTAPE( ISIG, SIGT, NSIG, ITAPE, TITLE, LSTRT,
3: & ILIMIT, JADJM, JDDX, JSTATX,NGPX, NGGX, NTGX,
4: & NDSPX, NDSGX, NDSTX, NUSX, NSCTX, NCOEFX,
5: & NMEDX, LRSGX, LANGX, LNGSX, LNSGX, LDELX,
6: & LNNNX, LNNGX, LSTOTX,LSSCX, LSNUFX,LSNAPX,
7: & LSGPX, LSNFPX,LSNUSX,LS2DPX,LS2PPX,LSGPBX,
8: & LSSCBX,LSDDBX,LSANGX,LSGPPX,LSSCPX,LSPORT,NLAST
9: & )
10: C=====
11: C PURPOSE : EITHER READS OR WRITES A PROCESSED XSEC FILE
12: C CALLED IN : XSECIN
13: C CALLS : XSCHLP
14: C=====
15: C
16: include '../shared/INC/_IOUNIT'
17: C
18: character*4 TITLE(20)
19: common /LOCSIG/ ISTART, ISCCOG, INABOG, IGABOG, IFPORG, IENGP,
20: & IPSPOG, IPSGOG, IPRBNG, IPRBGG, ISCANG, ISCAGG, ISPORG,
21: & ISPORT, INPBUF, ISIGOG, INFPOG, IABSOG, ITOTSG, NGP,
22: & NDS, NGG, NDSG, INGP, INDS, NMED, NELEM,
23: & NMIX, NCOEF, NSCT, NTS, NTG, NDSNGP, NDSNGG,
24: & IADJ, NME, LOC, INGS, INSG, I1, I0,
25: & KKK, IXTAPE, IDEL, ITEM1, ITEMG, IRSG, IRDSG,
26: & ISTR, IPRIM, IFMU, IMOM, IDNF, ISTAT, IPUN,
27: & NUS, NGN, IHT, INUS, INUSN, INGN, INGNP,
28: & INNN, IGGG, IPNGP, IPSPOG, IPSGOG, IANG, INUFIS,
29: & IS2DP, IS2PP
30: real SIGT(*)
31: integer NSIG(*)
32: C
33: C**** EQUIVALENCE (SIGT(1),NSIG(1)) ***** ASSUMED *****
34: C
35: if ( ISIG.eq.2 ) go to 120
36: C
37: C ==== READ ALREADY PROCESSED X-SEC FILE ==== ( UPTO LABEL 40 )
38: C
39: C
40: read(ITAPE) (TITLE(M),M=1,20)
41: read(ITAPE) JADJM, JDDX, JSTATX, NGPX, NGGX, NTGX, NDSPX, NDSGX,
42: & NDSTX, NUSX, NSCTX, NCOEFX, NMEDX, LRSGX, LANGX, LNGSX,
43: & LNSGX, LDELX, LNNNX, LNNGX, LSTOTX, LSSCX, LSNUFX, LSNAPX,
44: & LSGPX, LSNFPX, LSNUSX, LS2DPX, LS2PPX, LSGPBX, LSSCBX,
45: & LSDDBX, LSANGX, LSGPPX, LSSCPX, LSPORT, NLAST
46: C
47: C
48: C ..... ADJUST POINTERS TO ACCOMODATE CURRENT MEMORY ALLOCATION .....
49: C
50: C
51: NL = LSTRT - LRSGX
52: LRSGX = LRSGX + NL
53: LANGX = LANGX + NL
54: LNGSX = LNGSX + NL
55: LNSGX = LNSGX + NL
56: LDELX = LDELX + NL
57: LNNNX = LNNNX + NL
58: LNNGX = LNNGX + NL
59: LSTOTX = LSTOTX + NL
60: LSSCX = LSSCX + NL
61: LSNUFX = LSNUFX + NL
62: LSNAPX = LSNAPX + NL
63: LSGPX = LSGPX + NL
64: LSNFPX = LSNFPX + NL
65: Ccccc LNUSX = LNUSX + NL

```

```

66: LS2DPX = LS2DPX + NL
67: LS2PPX = LS2PPX + NL
68: LSGPBX = LSGPBX + NL
69: LSSCBX = LSSCBX + NL
70: LSDDBX = LSDDBX + NL
71: LSANGX = LSANGX + NL
72: LSGPPX = LSGPPX + NL
73: LSSCPX = LSSCPX + NL
74: LSPORT = LSPORT + NL
75: C
76: NLAST = NLAST + NL
77: NTEMP = NLAST - ILIMIT
78: C
79: 100 write(I0,7000) LRSGX, NLAST
80: 7000 format(/' CROSS SECTIONS START AT LOCATION ',I12/,
81: & ' LAST LOCATION USED ',I12/)
82: C
83: if ( NLAST.gt.ILIMIT ) then
84: write(IMG,7020) NTEMP
85: 7020 format(/'XXX(XSTAPE) CROSS-SECTION STORAGE EXCEEDS',
86: & ' AVAILABLE MEMORY BY ',I12,' WORDS !!')
87: call PRSTOP( 1, 'MEMORY SHORTAGE IN CROSS SECTION PROCESSING.'
88: & )
89: stop 999
90: end if
91: C
92: read(ITAPE) (SIGT(M),M=LRSGX,NLAST)
93: C
94: C
95: C *****
96: call LABEL( I0,
97: & 'THE FOLLOWING VALUES ARE FROM PROCESSED CROSS SECTION FILE'
98: & )
99: C *****
100: C
101: C
102: write(I0,7040) (TITLE(M),M=1,20)
103: 7040 format(1X,20A4)
104: C
105: C ISGG=NUS+IHT+1
106: C ITBL=INDS+ISGG-1
107: C
108: write(I0,7060) NGPX, NGGX, NTGX, NMEDX, NCOEFX, NSCTX, JSTATX,
109: & JADJM, JDDX
110: C
111: 7060 format(///' NUMBER OF PRIMARY GROUPS (NGPX) ',I6/1X,
112: & ' NUMBER OF SECONDARY GROUPS (NGGX) ',I6/1X,
113: & ' NUMBER OF PRIM+SEC GROUPS (NTGX) ',I6/1X,
114: & ' NUMBER OF MEDIA (NMEDX) ',I6/1X,
115: & ' NUMBER OF COEFFICIENTS (NCOEFX) ',I6/1X,
116: & ' NUMBER OF ANGLES (NSCTX) ',I6/1X,
117: & ' RESTORE COEFF (JSTATX) ',I6/1X,
118: & ' ADJOINT SWITCH (JADJM) ',I6/1X,
119: & ' CROSS SECTION TYPE (JDDX) ',I6/)
120: C
121: C
122: if ( JDDX.ne.0 ) then
123: write(I0,7080)
124: write(I0,7100) (SIGT(LANGX-1+I),I=1,NSCTX+1)
125: end if
126: 7080 format(/' ANGLE STRUCTURE'/)
127: 7100 format(7X,10E11.3)
128: C
129: C
130: write(I0,7120)

```

src/gmvp/xstape.f

```

131: 7120 format('/' MIXING TABLE'/')
132: C
133:      NMIX      = (LANGX-LRSGX) /3
134:      do 110 I = 1, NMIX
135:          J1      = LRSGX + I - 1
136:          J2      = LRSGX + NMIX + I - 1
137:          J3      = LRSGX + 2*NMIX + I - 1
138:          write(I0,7140) I, NSIG(J1), NSIG(J2), SIGT(J3)
139:      110 continue
140: 7140 format(10X,3I5,1PE15.5)
141: C
142:      write(I0,7160) (NSIG(I),I=LDELX,LNNNX-1)
143: 7160 format('/' ELEMENT ID'S'/(/5X,20I6))
144: C
145:      call RWIND( ITAPE )
146:      if ( ISTR.gt.0 ) go to 140
147: C
148:      return
149: C
150: C
151: C ===== WRITE PROCESSD DATA ON FILE =====
152: C
153: C
154: 120 continue
155:      write(ITAPE) (TITLE(M),M=1,20)
156:      write(ITAPE) JADJM, JDDX, JSTATX, NGPX, NGGX, NTGX, NDSPX, MDSGX,
157:      &      NDSTX, NUSX, NSCTX, NCOEFX, NMEDX, LRSGX, LANGX, LNSGX,
158:      &      LNSGX, LDELX, LNNNX, LNNGX, LSTOTX, LSSCX, LSNFX, LSNAPX,
159:      &      LSGPX, LSNFPX, LSNUSX, LS2DPX, LS2PPX, LSGPBX, LSSCBX,
160:      &      LSDDBX, LSANGX, LSGPPX, LSSCPX, LSPORT, NLAST
161:      IA      = LRSGX
162:      IB      = NLAST
163: C
164: C
165: C... 45 WRITE(ITAPE) ( SIGT(M),M=IA,IB)
166: C
167: C March 1994:
168: C
169: C The system library routine for binary output in DEC/OSF1 pre 1.2
170: C version has a bug (does not convert real data to big-endian even if
171: C "-convert big_endian" compiler option is specified).
172: C So output to scratch file is done through a integer variable IDUM(*)
173: C equivalent to SIGT(*)..
174: C
175: 130 write(ITAPE) (NSIG(M),M=IA,IB)
176: C
177: 140 continue
178: C
179: C
180: C
181:      if ( ISTR.gt.0 ) then
182: C
183:          write(I0,7180) (TITLE(M),M=1,20)
184:          write(I0,7200) JADJM, JDDX, JSTATX, NGPX, NGGX, NTGX, NDSPX,
185:          &      NDSGX, NDSTX, NUSX, NSCTX, NCOEFX, NMEDX, LRSGX, LANGX,
186:          &      LNSGX, LNSGX, LDELX, LNNNX, LNNGX, LSTOTX, LSSCX,
187:          &      LSNFX, LSNAPX, LSGPX, LSNFPX, LSGPBX, LSSCBX, LSDDBX,
188:          &      LSANGX, LSGPPX, LSSCPX, LSPORT, NLAST
189:          write(I0,7220) (SIGT(M),M=LANGX,LNSGX-1)
190:          write(I0,7240) (NSIG(M),M=LNSGX,LNSGX-1)
191:          write(I0,7260) (NSIG(M),M=LNSGX,LDELX-1)
192:          write(I0,7280) (NSIG(M),M=LDELX,LNNNX-1)
193:          write(I0,7300) (NSIG(M),M=LNNNX,LNNGX-1)
194:          write(I0,7320) (NSIG(M),M=LNNGX,LSTOTX-1)
195:          write(I0,7340) (SIGT(M),M=LSTOTX,LSGPBX-1)
196: C
197: C
198:      if ( LSSCBX.ne.LSGPBX ) then
199:          IC      = NGGX
200:          IA      = LSGPBX
201:          IB      = LSGPBX + IC - 1
202:          do 160 J = 1, NMEDX
203:              do 150 J1 = 1, NGPX
204:                  write(I0,7360) (SIGT(M),M=IA,IB)
205:                  IA      = IB + 1
206:                  IB      = IB + IC*2
207:                  write(I0,7380) (NSIG(M),M=IA,IB)
208:                  IA      = IB + 1
209:                  IB      = IB + IC
210:              continue
211:          continue
212:          if ( IC*NMEDX*3*NGPX.ne.LSSCBX-LSGPBX ) stop
213:      end if
214: C
215: C
216:      if ( JDDX.eq.0 ) then
217:          IB      = LSSCBX - 1
218:          do 180 J = 1, NMEDX
219:              do 170 J1 = 1, NGPX + NGGX
220:                  IC      = NSIG(LNNNX+J1-1)
221:                  IA      = IB + 1
222:                  IB      = IB + IC
223:                  write(I0,7400) (SIGT(M),M=IA,IB)
224:                  IA      = IB + 1
225:                  IB      = IB + IC*2
226:                  write(I0,7420) (NSIG(M),M=IA,IB)
227:              continue
228:          continue
229:          if ( IB.ne.LSDDBX-1 ) stop
230: C
231: C
232:      if ( LSDDBX.ne.LSANGX ) then
233: C
234:          IB      = LSDDBX - 1
235:          do 210 J = 1, NMEDX
236:              do 200 J1 = 1, NGGX + NGPX
237:                  IC      = NSIG(LNNNX+J1-1)
238:                  do 190 J2 = 1, IC
239:                      IA      = IB + 1
240:                      IB      = IB + NSCTX
241:                      write(I0,7440) (SIGT(M),M=IA,IB)
242:                      IA      = IB + 1
243:                      IB      = IB + NSCTX*2
244:                      write(I0,7480) (SIGT(M),M=IA,IB)
245:                  continue
246:              continue
247:          continue
248: C
249: C
250:          if ( IB.ne.LSANGX-1 ) stop
251:      end if
252: C
253: C ..... IDDX .NE. 0 ....
254:      else
255:          IB      = LSDDBX - 1
256:          do 240 J = 1, NMEDX
257:              do 230 J1 = 1, NGPX + NGGX
258:                  if ( JSTATX.eq.0 ) then
259:                      IC      = NSIG(LNNNX+J1-1)*NSCTX
260:                      IA      = IB + 1

```


src/gmvp/xstape.f

```
261:          IB      = IB + IC
262:          write(I0,7440) (SIGT(M),M=IA,IB)
263:          IA      = IB + 1
264:          IB      = IB + IC*2
265:          write(I0,7460) (NSIG(M),M=IA,IB)
266:          else
267:            IC      = NSCTX
268:            ID      = NSIG(LNNX+J1-1)
269:            do 220 J2 = 1, IC
270:              IA    = IB + 1
271:              IB    = IB + ID
272:              write(I0,7440) (SIGT(M),M=IA,IB)
273:              IA    = IB + 1
274:              IB    = IB + ID*2
275:              write(I0,7460) (NSIG(M),M=IA,IB)
276: 220      continue
277:            end if
278: 230      continue
279: 240      continue
280:            if ( IB.ne.LSANGX-1 ) stop
281:          end if
282: C
283: C
284:          if ( JSTATX.ne.0 ) then
285:            if ( JDDX.eq.0 ) then
286:              write(I0,7500) (SIGT(M),M=LSGPPX,LSSCPX-1)
287:              write(I0,7520) (SIGT(M),M=LSSCPX,LSPORT-1)
288:            else
289:              IB    = LSANGX - 1
290:              IC    = NSCTX
291:              do 260 J = 1, NMEDX
292:                do 250 J1 = 1, NGPX + NGGX
293:                  IA = IB + 1
294:                  IB = IB + IC
295:                  write(I0,7560) (SIGT(M),M=IA,IB)
296:                  IA = IB + 1
297:                  IB = IB + IC*2
298:                  write(I0,7580) (NSIG(M),M=IA,IB)
299:                continue
300:              260      continue
301:                if ( IB.ne.LSPORT-1 ) stop
302:              end if
303:              write(I0,7540) (SIGT(M),M=LSPORT,NLAST)
304:            end if
305: C
306:          end if
307: C
308:          7180 format('1',20A4)
309:          7200 format(20I6)
310:          7220 format(1P10E12.4)
311:          7240 format(' INGS',20I6)
312:          7260 format(' INSG',20I6)
313:          7280 format(' IDEL',20I6)
314:          7300 format(' INNN',20I6)
315:          7320 format(' IGGG',20I6)
316:          7340 format(' ISTART',1P10E12.4)
317:          7360 format(' IPNGP ',1P10E12.4)
318:          7380 format(' IPNGP ',20I6)
319:          7400 format(' IPSPOG',1P10E12.4)
320:          7420 format(' IPSPOG',20I6)
321:          7440 format(' IPRBNG',1P10E12.4)
322:          7460 format(' IPRBNG',20I6)
323:          7480 format(' IPRBNG',20F6.3)
324:          7500 format(' IFNGP ',1P10E12.4)
325:          7520 format(' IFSPOG',1P10E12.4)
326:          7540 format(' ISPORT',1P10E12.4)
327:          7560 format(' ISCANG',1P10E12.4)
328:          7580 format(' ISCANG',20I6)
329: C
330:          if ( ISIG.eq.2 ) call RWIND( ITAPE )
331: C
332:          return
333:          end
```

src/gmvp/zzbank.f

```
1:      subroutine ZZBANK( KDBNK, MDBNK, KIBNK, MIBNK,  
2:      &                 KDFBK, MDFBK, KIFBK, MIFBK )  
3: C=====
```

4: C PURPOSE: Optional particle bank reservation for custom use.
5: C CALLED IN: INTRO2
6: C CALLS:
7: C-----

8: C About optional data bank arrays:
9: C
10: C Particle attributes are stored in arrays XXX(1:NBANK) (X-coordinates)
11: C EEE(1:NBANK) (energy) etc. in MVP/GMVP and they are used for fixed
12: C purpose.
13: C MVP/GMVP version2.x and newer has arrays called "optinal bank data"
14: C used for any purpose and their data size may vary depending on
15: C problems to be solved or used in custom version by user.
16: C
17: C DBNK(1:NBANK,*) : double word bank array.
18: C IBNK(1:NBANK,*) : double word bank array.
19: C DFBK(1:NFBNK0,*) : double word bank array for fission neutron in
20: C eigenvalue calculation mode.
21: C IFBK(1:NFBNK0,*) : double word bank array for fission neutron in
22: C eigenvalue calculation mode.
23: C
24: C Array KDBNK(*) is used to point specified optinal attributes in
25: C array DBNK like DBNK(*,KDBNK(1)). Some elements of KDBNK(*) have
26: C meanings defined by developer as:
27: C
28: C DBNK(*,KDBNK(1)) is particle weight on birth,
29: C DBNK(*,KDBNK(2)) is time of birth,
30: C ...
31: C
32: C and elements KDBNK(11) to KDBNK(16) are for custom use and users can
33: C give any attributes in their custom version.
34: C
35: C KIBNK, KDFBK and KIFBK works in similar way for arrays IBNK, DFBK and
36: C IFBK respectively.
37: C
38: C-----

39: C
40: C In this routine, users must set size of blocks of NBANK/NFBNK0 words
41: C used from optional bank arrays. Conversion to second array index for
42: C DBNK etc. is performed automatically in subroutine INTRO2.
43: C
44: C For example, if a user want to use attribute 11 of DBNK with size of
45: C 3*NBANK words (double) and attribute 13 of IFBK with size of
46: C NBANK* (NEST + 1) words (single), necessary coding in this routine is
47: C as follows:
48: C
49: C KDBNK(11) = 3
50: C KIFBK(13) = NEST + 1
51: C
52: C Index 11 to 16 of KDBNK, KIBNK, KDFBK and KIFBK are provided for
53: C custom use.
54: C
55: C=====

```
56: C include '../shared/INC/_IUNIT'  
57: C include 'INC/_SIZES'  
58: C include 'INC/_FLAGS'  
59: C  
60: C integer KDBNK(0:MDBNK)  
61: C integer KIBNK(0:MIBNK)  
62: C  
63: C integer KDFBK(0:MDFBK)  
64: C integer KIFBK(0:MIFBK)  
65: C
```

```
66: C-----  
67: C KDBNK(11) = 3  
68: C KIFBK(13) = NEST + 1  
69: C  
70: C return  
71: C end
```

src/gmvp/zzcustom.f

```

1:      subroutine CUSTOM( IOIN1, IPR1,  A, DA, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: input data for a custom version.
4: C       Users should add procedure to input/process data
5: C       related to customization.
6: C
7: C   Data block for custom version is assumed as follows.
8: C
9: C       $CUSTOM
10: C
11: C       custom data
12: C
13: C   $END CUSTOM
14: C
15: C   ( The string "$CUSTOM" is already input in "INTRO" routine.
16: C     Data are ignored in non-custom version )
17: C
18: C CALLED IN: INTRO
19: C----< comments for custom version >-----
20: C
21: C HISTORY:
22: C UPDATE:
23: C
24: C-----
25: C
26: C   arguments ( i = input, o = output, c = constant, w = work )
27: C
28: C c   IOIN1 : current input I/O-unit
29: C c   IPR1  : printout I/O-unit
30: C io  A(*)  : dynamic memory array area (real)
31: C io  DA(*) : dynamic memory array area (double precision)
32: C io  CHA(*) : dynamic memory array area (4 byte character strings)
33: C
34: C*io last : last position of used area in array A(*) of common /ARRAY/
35: C
36: C-----
37: C
38: C   include '../shared/INC/_ARRAY'
39: C   include 'INC/_FLAGS'
40: C   include '../shared/INC/_SIZES'
41: C   include 'INC/_SIZES2'
42: C   include 'INC/_CXSEC'
43: C   include 'INC/_PXSEC'
44: C   include 'INC/_XBANK'
45: C   include 'INC/_SBANK'
46: C   include 'INC/_STACK'
47: C   include 'INC/_XWORK'
48: C   include '../shared/INC/_PGEOM'
49: C   include '../shared/INC/_PVRED'
50: C   include 'INC/_PSOUR'
51: C   include '../shared/INC/_PTALY0'
52: C   include 'INC/_PTALY'
53: C   include 'INC/_PTALY2'
54: C   include '../shared/INC/_PTLSP'
55: C   include '../shared/INC/_LISTON'
56: C   include '../shared/INC/_WORDL'
57: C
58: C-----
59: C
60: C   By default skip input data until "$END CUSTOM"
61: C
62: C   To customize , remove the following lines and add your own procedure.
63: C
64: C Hints for customization:
65: C

```

```

66: C   You can use some utility routines for MVP/GMVP type free form inputs.
67: C   * FREADS routine to get data-name.
68: C   * I4READ, R4READ, R8READ & CHREAD to read data elements of
69: C     integer, real, double precision and string respectively.
70: C
71: C-----
72: C Description of utility routines:
73: C   ( arguments : i = input, o = output, c =constant, w = work )
74: C
75: C-----
76: C   SUBROUTINE FREADS( VNAME, NLEN, IEND )
77: C-----
78: C
79: C       Get next data-name from current input file.
80: C
81: C       o VNAME (character*(*)): data-name found.
82: C       o NLEN (integer): length of data-name found.
83: C       o IEND (integer): end of file encountered if not zero.
84: C
85: C-----
86: C   subroutine I4READ( VNAME,IA, NA,NB,IEND )
87: C   subroutine R4READ( VNAME,RA, NA,NB,IEND )
88: C   subroutine R8READ( VNAME,RA8,NA,NB, IEND )
89: C-----
90: C
91: C       Get numerical data of integer, real and double real type.
92: C       This routine must be called after "FREADS" succeeded in getting
93: C       a data-name.
94: C
95: C       i VNAME (character*(*)): data-name.
96: C       o IA(*),RA(*),RA8(*) : arrays to put data
97: C                             (integer, real , duple real)
98: C       o NA (integer) : number of data read in.
99: C       i NB (integer) : if NB > 0, NB is the number of data
100: C                       expected to be input.
101: C                       if NB < 0, no expected number of data
102: C                       but data over abs(NB)'th data
103: C                       are skipped.
104: C                       NB = 0 also means no expected number of
105: C                       data but not recommended to use.
106: C       o IEND (integer) : end of file encountered if not zero.
107: C
108: C-----
109: C   subroutine CHREAD( VNAME, CGET, CHAR, NLEN, ITERM, IEND )
110: C-----
111: C
112: C       Get a string data from current input file. This routine gets
113: C       only one non-blank string per call, so you may need to call
114: C       this routine repeatedly until a termination point is reached.
115: C
116: C       i VNAME (character*(*)): data-name.
117: C       o CGET (character*(*)) : a string input.
118: C       i CHAR (character*(*)) : list of termination characters
119: C                               other than blank. The most popular one is
120: C                               ' ' which terminates data sequence such as;
121: C                               data-name( str1 str2 ... strn )
122: C       o NLEN (integer) : length of string read in.
123: C       o ITERM (integer) : If non-zero, encountered a termination
124: C                           character CHAR(ITERM:ITERM).
125: C       o IEND (integer) : end of file encountered if not zero.
126: C
127: C-----
128: C comment: GETLIN routine requires printout I/O unit as an argument
129: C          after MVP/GMVP version 2.0
130: C-----

```

src/gmvp/zzcustom.f

```
131: C
132:     real A(*)
133:     real*8 DA(*)
134:     character*4 CHA(*)
135: C
136: C ... local data ...
137: C
138:     character VNAME*32
139:     character LINE*72
140: C
141:     call HEADER( IPRI, 'INPUT FOR CUSTOMIZED VERSION' )
142:     write(IPRI, '/lx, ' !!! This version ignores "$CUSTOM" block. ' )
143:     write(IPRI, '/lx, ' ( parameter definition is effective ) ' )
144:     call CNTERR( 'WARNING' )
145: C
146: C ... "GETLIN" gets a non-comment line from input data.
147: C (parameter line ( % name=value, name=value, ...) is processed.
148: C If you want to ignore parameter definition also, use "CPLIN"
149: C routine which has the same arguments as "GETLIN" )
150: C
151: 100 call GETLIN( IOIN1, IPRI, LINE, NCHAR, IEND )
152: C
153:     if ( IEND.ne.0 ) then
154:         write(IPRI,*) 'XXX Unexpected end of input-data ',
155: & 'in custom data block.'
156:         write(IPRI,*) ' Probably missing "$END CUSTOM" line.'
157:         call CNTERR( 'FATAL' )
158:         call PRSTOP( 0, 'custom data input error.' )
159:         stop 888
160:     end if
161: C
162:     if ( LINE(1:11).ne.'$END CUSTOM' ) go to 100
163: C
164: C
165: C===== END OF DEFAULT PROCEDURE =====
166: C
167: C
168:     return
169: end
```

src/gmvp/zzinit.f

```
1:      subroutine ZZINIT
2: C=====
3: C  PURPOSE: initialization for custom version.
4: C  CALLED IN: MAIN
5: C  CALLS:
6: C-----
7: C This routine is called in main routine before any data input or
8: C command argument processing. It is provided for initialization for
9: C custom version by user.
10: C
11: C Users can set default value of user-added global data or
12: C printout message etc..
13: C
14: C-----
15: C      include '../shared/INC/_IUNIT'
16: C
17:      return
18:      end
```

src/gmvp/zztalo.f

```
1:      subroutine ZZTALO( WSUM1, IPRI, A, DA, H, DH, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: tally output etc. after random walk in custom version
4: C CALLED IN: CADENZ
5: C-----< comments for custom version >-----
6: C
7: C HISTORY:
8: C UPDATE:
9: C
10: C-----
11: C
12: C   arguments ( i = input, o = output, c = constant, w = work )
13: C
14: C i WSUM1 (REAL*8) : sum of particle weights. May have the same address
15: C                   with WSUM in COMMON area.
16: C io A(*) : dynamic memory array (task shared,real)
17: C io DA(*) : dynamic memory array (task shared,double precision)
18: C io H(*) : dynamic memory array (task local,real)
19: C io DH(*) : dynamic memory array (task local,double precision)
20: C io CHA(*) : dynamic memory array area (4 byte character strings)
21: C
22: C-----
23: C
24: C   include '../shared/INC/_ARRAY'
25: C   include 'INC/_FLAGS'
26: C   include '../shared/INC/_SIZES'
27: C   include 'INC/_SIZES2'
28: C   include 'INC/_CXSEC'
29: C   include 'INC/_PXSEC'
30: C   include 'INC/_XBANK'
31: C   include 'INC/_SBANK'
32: C   include 'INC/_STACK'
33: C   include 'INC/_XWORK'
34: C   include '../shared/INC/_PGEOM'
35: C   include '../shared/INC/_PVRED'
36: C   include 'INC/_PSOUR'
37: C   include '../shared/INC/_PTALY0'
38: C   include 'INC/_PTALY'
39: C   include 'INC/_PTALY2'
40: C   include '../shared/INC/_PTLSP'
41: C   include '../shared/INC/_WORDL'
42: C-----
43:      real*8 WSUM1
44:      real A(*)
45:      real*8 DA(*)
46:      real H(*)
47:      real*8 DH(*)
48:      character*4 CHA(*)
49: C
50:      return
51: end
```

src/gmvp/zztals.f

```
1:      subroutine ZZTALS(NGENE0, WSUMB,  A, DA, H, DH, CHA )
2: C=<MVP/GMVP>=====
3: C PURPOSE: take tally sum etc. after each batch in custom version
4: C CALLED IN: TALSUM
5: C-----< comments for custom version >-----
6: C
7: C HISTORY:
8: C UPDATE:
9: C
10: C-----
11: C
12: C  arguments ( i = input, o = output, c = constant, w = work )
13: C
14: C i  NGENE0 : number of particles generated in current batch
15: C i  WSUMB  : total weight of particles generated in current batch
16: C io A(*) : dynamic memory array (task shared,real)
17: C io DA(*) : dynamic memory array (task shared,double precision)
18: C io H(*) : dynamic memory array (task local,real)
19: C io DH(*) : dynamic memory array (task local,double precision)
20: C io CHA(*) : dynamic memory array area (4 byte character strings)
21: C
22: C-----
23: C
24: C      include 'INC/_FLAGS'
25: C      include '../shared/INC/_SIZES'
26: C      include 'INC/_SIZES2'
27: C      include 'INC/_CXSEC'
28: C      include 'INC/_PXSEC'
29: C      include 'INC/_XBANK'
30: C      include 'INC/_SBANK'
31: C      include 'INC/_STACK'
32: C      include 'INC/_XWORK'
33: C      include '../shared/INC/_PGEOM'
34: C      include '../shared/INC/_PVRED'
35: C      include 'INC/_PSOUR'
36: C      include '../shared/INC/_PTALY0'
37: C      include 'INC/_PTALY'
38: C      include 'INC/_PTALY2'
39: C      include '../shared/INC/_PTLSP'
40: C      include '../shared/INC/_WORDL'
41: C-----
42:      real A(*)
43:      real*8 DA(*)
44:      real H(*)
45:      real*8 DH(*)
46:      real*8 WSUMB
47:      character*4 CHA(*)
48: C
49:      return
50:      end
```

src/gmvp/INC/_FBANK2

```
1: C*
2: C*.... Secondary fission bank (copy of fission bank for calculation
3: C*   which needs more than one sub-batch for a batch)
4: C*
5: C*   See _XBANK for description of variables (XXXF2 corresponds to
6: C*   XXXF and array size is NHIST etc.)
7: C*
8:   parameter ( MDFBK2 =20 , MIFBK2 = 20 )
9:   common /XBNKF2/
10:  & LXXXF2,LYYF2, LZZZF2, LIZZF2, LKLSFF2,
11:  & LLEVLF2, LLZZF2, LLPOSF2, LLCRSF2,
12:  & LIBSFF2, LIBRGF2,
13:  & LDFBK2, KDFBK2(0:MDFBK2), LIFBK2, RIFBK2(0:MIFBK2)
```


src/gmvp/INC/_FLAGS

```

1: C*
2: C* .... OPTIOTN FLAGS (GMVP) .....
3: C*
4:     parameter (MAXJDB = 16, MXJVP = 8 )
5:     parameter (MKPAR = 64)
6:     COMMON /FLAGS /
7:     & JREST, JARST,
8:     & JNEUT, JPHOT, JIMAG, JTIME, JDELT, JFISS, JEIGN, JFIXD,
9:     & JADJM, JDDX , JIMPT, JWWND, JPSTR, JWTIM, JFCOL, JBIAS, JRWVR,
10:    & JRESP, JMNTR, JVMNT, JSTPRN(3), JDEBG(MAXJDB),
11:    & JALLZ, JONEZ, JSIMP, JPICT, JREFL, JLATT, JHLAT, JFISX, JSTGR,
12:    & JFPRT, JRRLT, JSTOP, JPRTS(20), JSTATX,
13:    & JPTDT, JMCCHK, JTLLT, JREPR, JRUNM,
14:    & JNCOL, JOSRC, JRSTF, JSCTR, JUNDG, JVPPS(MXJVP),
15:    & JANLF, JPTIM, JTSRF, JKPAR(MKPAR),
16:    & JNRUN
17: C*
18: C*.... DEFAULT VALUES OF OPTION FLAGS ARE GIVEN IN 'FLAGIN'
19: C*
20: C*      INPUT-SYMBOL      DEFAULT      INCOMPATIBLE OPTION
21: C JREST I4      RESTART      : NO
22: C JARST I4      AUTO-RESTART  : NO
23: C JNEUT I4      NEUTRON      : YES
24: C JPHOT I4      PHOTON       : NO
25: C JIMAG I4      IMAGINARY-PARTICLE : NO      JNEUT JPHOT
26: C JTIME I4      TIME-DEPENDENT : NO
27: C JDELT I4      DELTA-TRACKING : NO
28: C JFISS I4      FISSION       : NO
29: C JEIGN I4      EIGEN-VALUE   : NO
30: C JFIXD I4      FIXED-SOURCE  : YES
31: C JDDX I4       DDX           : YES
32: C JSTATX I4     KEEP PL MOMENT OR      (to be set internally)
33: C JADJM I4      ADJOINT        : NO
34: C JRRLT I4      RUSSIAN-ROULETTE : YES      JIMPT
35: C JIMPT I4      IMPORTANCE      : NO      JWWND
36: C JWWND I4      WEIGHT-WINDOW   : NO      JIMPT
37: C JPSTR I4      PATH-STRETCHING : NO
38: C** JWTIM added July 1998
39: C JWTIM I4      apply time weight : NO      (to be set internally)
40: C JFCOL I4      FORCED-COLLISION : NO
41: C JBIAS I4      SOURCE-BIAS     : NO
42: C** JRWVR added July 1998
43: C JRWVR I4      RELATIVE-WEIGHT  : NO
44: C      Apply weight reduction and secondary partical generation$
45: C      $ by weight relative to that at particle generation.$
46: C      $ (source or secondary/tertiary/... particle generation)
47: C JRESP I4      RESPONSE        : NO
48: C JMNTR I4      MONITOR         : NO
49: C JVMNT I4      VP-MONITOR      : YES
50: C*JDEBG I4      DEBUG-PRINT     : NO
51: C JDEBG(1) I4   DEBUG-PRINT     : NO
52: C      Debug printout in data input phase.
53: C JDEBG(2) I4   DEBUG-PRINT     : NO
54: C      Unused.
55: C JDEBG(3) I4   DEBUG-PRINT     : NO
56: C      Printout particle source attributes.
57: C JDEBG(4) I4   DEBUG-PRINT     : NO
58: C      Check in event selection (SELONE)
59: C      .eq.2 : printout selecteed event # and stack sizes
60: C      after every call of SELONE.
61: C JDEBG(5) I4   DEBUG-PRINT     : NO
62: C      Check stack size consistency in event selection (SELONE)
63: C JDEBG(6) I4   DEBUG-PRINT     : NO
64: C      Save last event # for last JDEBG(6) events for each
65: C      particle.

```

```

66: C JALLZ I4      ALL-ZONE        : YES
67: C JONEZ I4      ONE-ZONE        : NO
68: C JSIMP I4      GEOMETRY IS SIMPLE (NO (-) SIGNED BODY HAVING MORE THAN ONE
69: C      SURFACE. SET BY CODE IN SUBROUTINE 'ZONEIN'.
70: C JPICT I4      PICTURE         : NO
71: C JREFL I4      (REFLECTIVE SURFACE) : NO      (TO BE SET INTERNALLY)
72: C JLATT I4      LATTICE GEOMETRY  : NO
73: C JHLAT I4      PROBLEM WITH HEXAGONAL LATTICE
74: C JFPRT I4      FLUX-RPINT       : NO
75: C JSTOP I4      JOB TERMINATION (TO BE SET INTERNALLY)
76: C JNRUN I4      NO RANDOM WALK   (TO BE SET INTERNALLY)
77: C      If JNRUN != 0, no random walk is performed but a restart
78: C      file is output.
79: C
80: C JREPR I4      REPRODUCIBLE-RUN  : YES
81: C      $ Run multitasking mode so as to get reproducible $
82: C      $results. Non-reproducible run (if possible) may be $
83: C      $more effective in multitasking mode.
84: C*
85: C***** PRINT SUPPRESS FLAGS *****
86: C JPRTS(20) I4  If JPRTS(I)=1, printout of I'th print item is suppressed.
87: C
88: C      1 : group-wise flux          2 : angular-dependent flux
89: C      3 : time-dependent flux      4 : response
90: C      5 : time-dependent response  6 : source information #1
91: C      7 : source information #2
92: C      8-20 : not used
93: C*
94: C JSTATX I4     Save angular distribution for scattering : NO
95: C JPTDT I4      POINT-DETECTOR      : NO
96: C*
97: C*JMCHK I4      'MEMORY-CHECK' OPTION FOR DEBUGGING.
98: C*      *PRINT MEMORY ASSIGNMENT IN 'KEEP' ROUTINE.
99: C*      *PROGRAMMERS CAN CHECK
100: C*      INVALID VALUE ASSIGNMENTS FOR MEMORIES ON /ARRAY/ COMMON
101: C*      BY CALLING 'MMCHK' ROUTINE.
102: C JTLLT I4      TALLY-LATTICE      : NO
103: C
104: C      $ Independent region assignments to each nested zones in$
105: C      $ lattice geometry
106: C JRUNM I4      'RUN-MODE' option to stop processing before random-walk
107: C      for input data check etc.
108: C
109: C      Run mode is controled by each decimal digits as follows;
110: C
111: C      all digit is 0 : ORDINARY RUN.
112: C
113: C      n1*1          : n1>0 stop before randomwalk.
114: C      + n2*10       : n2=1 skip cross section input (and no randomwalk).
115: C                      (n2: not implimented currenty, Sep 1994)
116: C      + n3*100      : < no use >
117: C      + n4*1000     : < no use >
118: C      + n5*10000    : < no use >
119: C      + n6*100000   : < no use >
120: C      + n7*1000000  : < no use >
121: C      + n8*10000000 : < no use >
122: C*
123: C*** JNCOL : Added Oct 1996
124: C JNCOL I4      COLLISION-LESS : particles have no collision.
125: C      Only collisionless contribution by source particles
126: C      is evaluated as tracklength estimators.
127: C      Particle weights are reduced with absorption probabilitiy.
128: C      Virtual materials such as perfect absorber(-1000),
129: C      perfect/white reflectors are treated as such.
130: C

```

src/gmvp/INC/_FLAGS

```
131: C*** JOSRC : Added May 1997
132: C JOSRC I4  SOURCE-OUTPUT : output particle source data on a file.
133: C*** JRSFT : Added May 1998
134: C JRSTF I4  RESTART-FILE : output restart file.
135: C*** JSCRT : Added Jun 1997
136: C JSCRT I4  OPEN-SCRATCH : Open scratch I/O units
137: C           if they are not specified to open in command line.
138: C           (This is not to make ugly fort.** etc in sub-task of
139: C           PVM/MPI multi-task mode.)
140: C JTSKR I4  Spawn "tasker" process which assigns jobs to workhorse
141: C           tasks in multi-tasking mode if necessary.
142: C*** added 25 Jul 1997
143: C JUNDG I4  "Underground" level particle event processing is necessary.
144: C           Currently "underground" level processings are particle
145: C           tracking for next event estimator, and zone-based sampled
146: C           source rejection.
147: C           This flag is set internally and cannot be given as input.
148: C*** added 3 Aug 1997
149: C
150: C JVPSP(*) I4 Option flags specific to VPP-Fortran multi-task mode.
151: C           JVPSP(1) = 1 : Printouts of "ACTION" phase of each task are
152: C           output on the same I/O unit.
153: C
154: C*** added Nov 1998
155: C JFISX I4  Lattice frames or cells may intersect each other.
156: C           (to be set internally)
157: C JSTGR I4  Problem with STG region (to be set internally)
158: C*** added 9 Apr 1999
159: C JANLF I4  ANALOG-FISSION : NO
160: C           if JANLF=1, automatically set
161: C           JFIXD=1,JEIGN=0,JFISS=1,
162: C           currently not effective in GMVP.
163: C*** added 7 Jul 1999
164: C JPTIM I4  PERIODIC-TIME : NO
165: C           effective only when if JTIME=1.
166: C*** added 3 Feb 2000
167: C JSTPRN I4  SUBTASK-PRINT
168: C           printout from subtask in multi-task mode.
169: C
170: C           JSTPRN(1) : for printout in input/preparation phase.
171: C
172: C           0: throw away all printout.
173: C           1: print warning/error message (I/O unit IMMSG) but throw away
174: C           other printout. (default in MPI/PVM)
175: C           2: print all printout. (default in multi-thread)
176: C           -1: same as 1 but append to printout of main task
177: C           (currently unsupported)
178: C           -2: same as 2 but append to printout of main task
179: C           (currently unsupported)
180: C
181: C           JSTPRN(2) : for printout in random walk phase.
182: C           (currently no control is effective)
183: C
184: C           Option value has the same meaning with SUBTASK-PRINT(1).
185: C           Default is SUBTASK-PRINT(2)=2 (print all)
186: C*** added Mar 2000
187: C JTSRF I4  Use surface crossing tally
188: C*** added Apr 2000
189: C JKPAP(MKPAR) I4 specified particles are treated in current
190: C           calculation if non-zero.
191: C*
```

src/gmvp/INC/_IOUNIT2

```
1: C*
2: C*  I/O UNIT PARAMETERS SPECIFIC TO GMVP:
3: C*
4: C*  IODDX : INPUT UNIT OF DDX CROSS SECTION LIBRARY.
5: C*  IOWDX1 : WORKING UNIT OF DDX CROSS SECTION HANDLING (1).
6: C*  IOWDX2 : WORKING UNIT OF DDX CROSS SECTION HANDLING (2).
7: C*  IOMDX : UNIT OF PROCESSED MACRO DDX CROSS SECTION.
8: C*
9:      PARAMETER ( IODDX=99, IOWDX1 = 97, IOWDX2 = 98, IOMDX=96)
```

SAFE

src/gmvp/INC/_KPIDS

```
1: C
2: C-----
3: C Particle ID's and particle species number limit for GMVP
4: C
5: C Please modify particle symbol information in file _KPSYMS
6: C and symbol initialization routine (KPINIT) when particle species
7: C are added or removed from this file.
8: C
9: C ... neutron and photon
10: C parameter ( KPNEUT=1 )
11: C parameter ( KPPHOT=2 )
12: C ... electron and proton
13: C parameter ( KPELEC=3 )
14: C parameter ( KPPROT=4 )
15: C ... pi mesons (pi-0 pi+ pi- )
16: C parameter ( KPPI0=5 )
17: C parameter ( KPPIP=6 )
18: C parameter ( KPPIM=7 )
19: C ... mu mesons (mu+ mu- )
20: C parameter ( KPMUP =8 )
21: C parameter ( KPMUM =9 )
22: C
23: C-----
24: C ... number of particles possible to treat in this code
25: C
26: C * this should be maximum of possible KP*
27: C * flag array JKPAR must have element size >= KPLIM.
28: C
29: C parameter ( KPLIM = 2 )
30: C
```

src/gmvp/INC/_KPSYMS

```
1: C
2: C Particle ID's and symbol matching table used in symbol/ID conversion
3: C routine and its initialization routine of GMVP.
4: C
5: C -----
6: C
7: C << "Must-be" 's of this file !!! >>
8: C
9: C * Particle ID definition file _KPIDS *must be* included before
10: C this file
11: C
12: C * When contents of particle ID definition file _KPIDS are changed,
13: C this file *must be* modified to follow the change.
14: C
15: C -----
16: C
17: C ... number of particles possible to treat in this code
18: C (this should be maximum of possible KP* )
19: C
20: C parameter ( KPLIM = ? )
21: C
22: C
23: C ... particle symbol string and their alias (or short form)
24: C
25: C character*32 KPSYM
26: C common /CKPSYM/ KPSYM(2,KPLIM)
27: C
28: C === possible initialization
29: C
30: C ... neutron and photon
31: C KPSYM(1,KPNEUT) = 'NEUTRON'
32: C KPSYM(2,KPNEUT) = 'N'
33: C KPSYM(1,KPPHOT) = 'PHOTON'
34: C KPSYM(2,KPPHOT) = 'P'
35: C ... electron and proton
36: C KPSYM(1,KPELEC) = 'ELECTRON'
37: C KPSYM(2,KPELEC) = 'EL'
38: C KPSYM(1,KPPROT) = 'PROTON'
39: C KPSYM(2,KPPROT) = 'PR'
40: C ... pi mesons (pi-0 pi+ pi- )
41: C KPSYM(1,KPPIO) = 'PIO'
42: C KPSYM(2,KPPIO) = 'PI0'
43: C KPSYM(1,KPPIP) = 'PI-PLUS'
44: C KPSYM(2,KPPIP) = 'PIP'
45: C KPSYM(1,KPPIM) = 'PI-MINUS'
46: C KPSYM(2,KPPIM) = 'PIM'
47: C ... mu mesons (mu+ mu- )
48: C KPSYM(1,KPMUP) = 'MU-PLUS'
49: C KPSYM(2,KPMUP) = 'MUP'
50: C KPSYM(1,KPMUM) = 'MU-MINUS'
51: C KPSYM(2,KPMUM) = 'MUM'
52: C
```

src/gmvp/INC/_NGPS

```
1: C
2: C   ... INC/_KPIDS must be included before this file to refer KPLIM
3: C
4: C   COMMON /NGPCOM/  NGP(KPLIM), KNGP(KPLIM+1), KENGP(KPLIM+1)
5: C
6: C NGP(KPLIM)   I4   number of (common) energy groups for each particle
7: C               species.
8: C KNGP(KPLIM+1) I4   KNGP(J) is < Sum of NGP(i),i=1,J-1> + 1.
9: C               It is also used to calcualte energy group # for each
10: C              particle species (from 1 to NGP(particleID)) from
11: C              "unified" energy group # (from 1 to NGROUP).
12: C KENGP(KPLIM+1) I4   KNGP(J) is < Sum of (NGP(i)+1),i=1,J-1> + 1.
13: C              It is used to point arrays on which data for each particle
14: C              species having length of <NGP(*)+1> (such as energy group
15: C              boundaries) are packed in the order of particle species ID.
16: C
```

src/gmvp/INC/_PMNTR

```
1: C*... CPU & VECTOR LENGTH MONITOR .....
2: cc*      COMMON /PMNTR/ CPUSOR,CPUSEL,CPUFLY,CPU SCH,CPUCOL,
3: cc*      &   NEXSOR,NVSOR,VLSOR,VL2SOR,MIVSOR,MAVSOR,
4: cc*      &   NEXSEL,NVSEL,VLSEL,VL2SEL,MIVSEL,MAVSEL,
5: cc*      &   NEXFLY,NVFLY,VLFLY,VL2FLY,MIVFLY,MAVFLY,
6: cc*      &   NEXSCH,NVSCH,VLSCH,VL2SCH,MIVSCH,MAVSCH,
7: cc*      &   NEXCOL,NVCOL,VLCOL,VL2COL,MIVCOL,MAVCOL
8: C*
9: C* CPU??? R4   CPU TIME OF SUBROUTINE
10: C* NEX??? I4   NUMBER OF EXECUTION OF SUBROUTINE
11: C* NV??? I4   TOTAL NUMBER OF VECTOR EVENT
12: C* VL??? R4   SUM OF VECTOR LENGTH
13: C* VL2??? I4  SQUARE SUM OF VECTOR LENGTH
14: C* MI??? I4   MINIMUM VECTOR LENGTH
15: C* MA??? I4   MAXIMUM VECTOR LENGTH
16: C*
17: C* '???' IS THE SYMBOL FOR EACH SUBROUTINE.
18: C* SOR = SOURCE, SEL = SELECT, FLY = FLIGHT, SCH = SEARCH, COL = COLLISION
19: C*
```

src/gmvp/INC/_PROGV

```
1: C*..... DESCRIPTION OF PROGRAM VERSION  ETC. (GMVP).....
2: C*
3:      common /cprog/ prog
4:      CHARACTER*60 PROG(40)
```

SAFE

src/gmvp/INC/_PSOUR

```
1: C*
2: C*.... SOURCE DATA & ARRAY POINTERS
3: C*
4:      COMMON /PSOUR/
5:      & LKSOUR, LISZON, LSOUR , LIDSRC,
6:      & LPSPAC, LPENRG, LKENRG, LNSTIM, LSTIM ,
7:      & LPSTIM, LISTIM, LNSANG, LSANG , LSAXIS, LPSANG, LISANG,
8:      & LIFISM,
9:      & LSRCSPP, NSRCSP, MWVEC, MVSTK, MXREJ,
10: C ... local data in multi processing
11:      & LMEMZN, llxyz, llprk, llvst
12: C*
13: C KSOUR(NSOUR)      I4  Type of particle source (old type)
14: C ISZON(2,NSOUR)    I4  Input zone # & zone # of particle source
15: C                    (old type)
16: C***** 5Nov 1996 make SOUR(NSOUR) real*8
17: C SOUR(NSOUR)      R8  Relative source intensity of each source.
18: C IDSRC(NSOUR)     I4  source set ID of each source. (this may not be$
19: C                    $ a uniq number.)
20: C PSPAC(10,NSOUR)   R4  Spatial distribution of particle source $
21: C                    $(old type). Meanings of each element$
22: C                    $ depend on the type of each source.
23: C PENRG(NGROUP,NSOUR) R4  energy distributions of each source.
24: C                    (old type)
25: C*KENRG(2,NGROUP,NSOUR) I4  ENERGY BIN # (FOR DISCRETE SAMPLING OF$
26: C                    $ SOURCE)
27: C*NSTIM(NSOUR)      I4  TIME BIN NUMBER OF EACH SOURCE. <JTIME>
28: C*STIM(NSTIM(NSOUR),NSOUR) R4  TIME BINS OF EACH SOURCE. <JTIME>
29: C*PSTIM(NSTIM(NSOUR),NSOUR) R4  TIME DISTRIBUTION. <JTIME>
30: C*ISTIM(NSOUR)      I4  POINTER TO STIM & PSTIM OF EACH SOURCE.$
31: C                    $ <JTIME>
32: C*NSANG(NSOUR)      I4  ANGLE NUMBER OF EACH SOURCE.
33: C*SANG(NSANG(NSOUR),NSOUR) R4  ANGLES (COSINE)
34: C*SAXIS(3,NSOUR)    R4  DIRECTION OF SOURCE GENERATION AXES.
35: C*PSANG(1+2*NSANG(NSOUR),NSOUR) R4  ANGLE DISTRIBUTION.
36: C*ISANG(NSOUR)      I4  POINTER TO SANG & PSANG OF EACH SOURCE.
37: C
38: C IFISM(NSOUR)      I4  Material # whose fission spectrum is used$
39: C                    $ to sample intial source in eigenvalue problem.
40: C MEMZN(NMEMS,NSOUR) I4  Generated zone # memory for particle sources.
41: C*
42: C* .... new type source sampling routines ....
43: C*
44: C SRCSP(*) I4 Data container of source sampling information.
45: C NSRCSP I4 Length of the data container of source sampling information.
46: C MWVEC I4 Maximum number of working vector necessary in sampling
47: C MVSTK I4 Maximum vector stack depth for vector calculator used
48: C          in 'SYSSRC' routine.
49: C LXYZ(10) I4 Pointers to save sampled data temporary in 'SOURCE'
50: C          routine.
51: C LPWRK(MWVEC) I4 Pointers to working vector necessary in
52: C          source sampling
53: C LVSTK(MVSTK) I4 Pointers to vector stack for vector calculator used
54: C          in 'SYSSRC' routine.
55: C MXREJ I4 Maximum of number of rejections in "ACCEPT" block of
56: C          source generation.
57: C
```

src/gmvp/INC/_PTALY2

1: C
2: C



src/gmvp/INC/_PTALY

```
1: C*
2: C* ... POINT DETECTOR SPECIFIC DATA ...
3: C*
4: C      COMMON /PTDET/ NPDET, NPLEN, LXPDET,
5: C      &          LIPDET,LIPDT2
6: C      (moved to shared/INC/_PTALY0, 18 Nov,1999)
7: C*
8: C**
9: C ... local data pointer in multi processing
10:      COMMON /PTALYD/ LFLPD, LSFLPD, LVFLPD, LREPD, LSREPD, LVREPD
11: C*
12: C FLPD(NGROUP,NPDET)  R8  Flux -point estimator (batch) <JPTDT>
13: C SFLLPD(NGROUP,NPDET) R8      Average or sum of 'FLPD' <JPTDT>
14: C VFLLPD(NGROUP,NPDET) R8      Variance or square sum of 'FLPD' <JPTDT>
15: C REPD(NPDET,NRESP)   R8  Reaction rate -point estimator <JRESP&JPTDT>
16: C SREPD(NPDET,NRESP)  R8      Average or sum of 'REPD' <JRESP&JPTDT>
17: C VREPD(NPDET,NRESP)  R8      Variance of 'REPD' <JRESP&JPTDT>
18: C*
```

src/gmvp/INC/_PXSEC

```
1: C*
2: C*.... CROSS SECTION ARRAY POINTERS (GMVP)
3: C*
4: C* June 1995: add LVEL
5: C* July 1996: add LIDMAT
6: C*
7: COMMON /PXSEC/
8: & NGPX , NGGX , NTGX , NDSPX , NDSGX , NDSTX , NUSX , NSCTX ,
9: & NCOEFX, NMEDX , ITAPEX,
10: & LRSGX , LANGX , LNGSX , LNSGX , LDELX , LNNX , LNXG , LSTOTX,
11: & LSSCX , LNUFX , LSNAPX , LSGPX , LSNFPX , LSNUSX , LSGPBX , LSSCBX ,
12: & LSDBX , LSANGX , LSGPPX , LSCPX , LSPORT, LWGTF , LWGTFI , LFKAI ,
13: & LKAI , LWGTG , LWGTGI , LS2DPX , LS2PPX,
14: & LVEL , LIDMAT
15: C*
16: C LRSGX I4 Starting location of cross section data$
17: C $ list of mixing table. ( 3*NMIX )
18: C ANGX(NSCT+1) R4 DDX angle boundaries,ddx only
19: C NGSX(NGPX) I4 Location of group to group transfer matrix$
20: C $ of each energy group (primary particle)
21: C NSGX(NGGX) I4 Location of group to group transfer matrix$
22: C $ of each energy group (secondary particle)
23: C DELX(NCOEFF,NELEM) I4 List of element ID.
24: C NNNX(NGPX) I4 Number of outgoing group. (primary particle)
25: C NNGX(NGGX) I4 Number of outgoing group. (secondary particle)
26: C STOTX(NTGX,NMEDX) R4 Total cross section.
27: C SSCX(NTGX,NMEDX) R4 Scattering cross section.
28: C SNUFX(NTGX,NMED) R4 Nu*fission cross section.
29: C SNAPX(NTGX,NMED) R4 Non absorption probability.
30: C SGPX(NGPX,NMEDX) R4 Gamma production probability.
31: C SNFPX(NTGX,NMEDX) R4 nu*sigf/sigt.
32: C SNUSX(NTGX,NMEDX) R4 Sum of upscattering.
33: C S2DPX(NTGX,NMEDX) R4 (SIGS+NU*SIGF+SIG(G-PRODUCTION))/SIGT
34: C S2PPX(2,NTGX,NMEDX) R4
35: C
36: C 1: SIGS/(S2DPX*SIGT), 2:(SIGS+NU*SIGF)/(S2DPX*SIGT)
37: C
38: C SGPBX(NGGX,3,NGPX,NMEDX) R4 N-gamma matrix ( BMC ) ( NGG*3*NGP*NMED )
39: C SSCBX(NDS*3,NTGX,NMEDX) R4 Group to group matrix ( P1 mode only )
40: C ( BMC ) ((NDSNGP+NDSNGG)*3*NMED )
41: C*****
42: C SDBX(*) R4 Energy-angle transfer probability table in ddx-mode.
43: C (Length: (DNSNGP+NDSNGG)*NSCT*3*NMED)
44: C***** ; P1 mode ((NDSNGP+NDSNGG)*NSCT*3*NMED)
45: C SANGX(*) R4 Angular distribution (BMC) DDX only (NSCT*3*NTGX*NMED)
46: C <JSTATX>
47: C SGPPX(*) R4 Gamma production matrix ( NGP*NGG ) P1 mode only
48: C SSCPX(*) R4 Scattering matrix ( NDSNGP+NDSNGG ) P1 mode only
49: C SPORT(*) R4 Anisotropic scattering data <JSTATX>
50: C
51: C WGTF(NREG) R4 Fission particle weight of each region <JFISS>
52: C WGTFI(NREG) R4 1.0/WGTF <JFISS>
53: C FKAI(NGROUP,NMAT) R4 Fission spectrum of each material <JFISS>
54: C KKAI(2,NGROUP,NMAT) I4 Energy bin# for B.M.C. sampling of FKAI <JFISS>
55: C WGTG(NREG) R4 Secondary particle weight of each region
56: C WGTFG(NREG) R4 1.0/WGTG <JNEUT*JPOT>
57: C
58: C VEL(NGROUP) R4 Energy group averaged particle velocity (cm/s).
59: C Gamma ray velocity is set to the light speed internally.
60: C IDAMT(NMAT) I4 Material ID
61: C
```

src/gmvp/INC/_SIZES2

```
1: C* ... GMVP specific parameters ...
2: C
3:     COMMON /SIZES2/  NSCT
4: C
5: C NSCT   I4   Scattering angle number. (DDX)
6: C
```

SAE

src/gmvp/INC/_STACK

```
1: C*
2: C*... PARTICLE STACK POINTERS & LENGTHS
3: C*
4:      COMMON /STACK/
5:      & LLSFFL, LNFFL, LIZFFL, LLSCOL,          LLSRC, LNNXT, LIZNXT,
6:      & LLSDED,          LLSREF, LIZREF, LISREF, LNBREF,
7:      & LLSLAT, LIZLAT, LNLXT,
8:      & LIMSFL, LIMNFL, LIMZFL, LIMSFX, LIMNFX, LIMZNX,
9:      & LIMSLT, LIMNLT, LIMZLT, LIMDED
10: C
11: C ... task local scalar variables ...
12: C
13: C/#IF PARA(XX*)
14: *      LOCAL COMMON /LSTACK/
15: C/#ELSEIF PARA(CRAY)
16: *      TASK COMMON /LSTACK/
17: C/#ELSE
18:      COMMON /LSTACK/
19: C/#ENDIF
20:      & NCOLS, IDEFER, NDEAD, NFISB,
21:      & NIMPT, NDIMPT, IMDEFR
22: C*
23: C*      **** FREE FLIGHT STACK ****
24: C LSFFL(NBANK) I4 Bank index of free-flight particle
25: C NFFL(NZONE+1) I4 Number of particles for each zone
26: C IZFFL(NBANK) I4 Zone #
27: C*      **** COLLISION STACK ****
28: C LSCOL(NBANK) I4 Bank index of collision particle
29: C NCOLS I4 Length of collision stack
30: C*      **** SEARCH (NEXT ZONE SEARCH) STACK ****
31: C LSSRC(NBANK) I4 Bank index of particles searching next zone
32: C NNXT(NZONE+1) I4 Length of search stack
33: C IZNXT(NBANK) I4 Number of particles for each zone
34: C IDEFER I4 Number of deferred particles
35: C***** MORGUE (DEAD PARTICLE STACK) ****
36: C LDED(NBANK) I4 Bank index of dead particles
37: C NDEAD I4 Number of dead particles
38: C***** FISSION BANK ****
39: C NFISB I4 Length of fission bank
40: C***** REFLECTION STACK ****
41: C LSREF(NBANK) I4 Bank index of particles to be reflected
42: C IZREF(NBANK) I4 Zone # for reflected particles to enter
43: C ISREF(NBANK) I4 Reflection surface # for each particle
44: C NBREF(NREFS+1) I4 Number of particles for each reflection surface
45: C*
46: C***** LATTICE STACK ****
47: C LSLAT(NBANK) I4 Bank index of particles waiting for lattice-search
48: C IZLAT(NBANK) I4 Lattice-buffer zone #
49: C NLBT(NLBZ+1) I4 Number of particles in each lattice-buffer-zone
50: C*
51: C NIMPT I4 Number of imaginary particles
52: C NDIMPT I4 Number of non-active imaginary particles
53: C
54: C IMSFL(IMPMAX) I4 Flight stack for imaginary particles.
55: C IMNFL(NZONE+1) I4 Number of imaginary particles in flight stack.
56: C IMZFL(IMPMAX) I4 Zone #'s of imaginary particles in flight stack.
57: C
58: C IMSNX(IMPMAX) I4 Next-zone-search stack
59: C IMNFX(NZONE+1) I4 Number of imaginary particles in search stack.
60: C IMZNX(IMPMAX) I4 Zone #'s of imaginary particles in search stack.
61: C IMDEFR I4 Number of deferred particles in search stack$
62: C $ for imaginary particles.
63: C IMSLT(IMPMAX) I4 Lattice-search stack
64: C IMNLT(NLBZ+1) I4 Number of imaginary particles in lattice search$
65: C $ stack.
```

```
66: C IMZLT(IMPMAX) I4 Zone #'s of imaginary particles in lattice-search$
67: C $ stack.
68: C*
69: C IMDED(IMPMAX) I4 Stack for non-active imaginary particles
```

src/gmvp/INC/_WKC2A

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR CEL2AB  (July 1997)
3: C*
4:      COMMON /WKC2A/  PCA(10)
5:      INTEGER      PCA
6: C*
7: C PCA(10) I4  Pointers to working arrays for 'CEL2AB' routine.
```

SAFE

src/gmvp/INC/_WKCOL

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : COLISN .....
3: C*
4:      COMMON /WKCOL/ P3(20)
5:      INTEGER      P3
6: C*
7: C P3(20) I4 Pointers to working arrays for 'colisn' routine.
```

SAFE

src/gmvp/INC/_WKFL1

```
1: C*
2: C* ..... POINTER TO WORKING ARRAYS : FLIONE (FLION0)
3: C*
4:      COMMON /WKFL1/ P0(50)
5:      INTEGER   P0
6: C* array size fro 30 to 50 (Nov 1998)
7: C P0(50) I4   Pointers to working arrays for 'FLIONE' routine.
```

SAFE

src/gmvp/INC/_WKFLA

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : FLIGHT
3: C*
4:      COMMON /WKFLA/ P6(30)
5:      INTEGER   P6
6: C*
7: C P6(30) I4  Pointers to working arrays for 'FLIGHT' routine.
```

WFLA

src/gmvp/INC/_WKFSS

```
1: C*<GMVP>
2: C*  .... Pointers to working arrays for FISSET (Feb 2000)
3: C*
4:      COMMON /WKSOU/  P10(10)
5:      INTEGER      P10
6: C*
7: C P10(10) I4 Pointers to working arrays for 'FISSET' routine.
```

SAFE

src/gmvp/INC/_WKLAT

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : LATIC
3: C*
4:      COMMON /WKLAT/ P5(20)
5:      INTEGER    P5
6: C*
7: C P5(20) I4   Pointers to working arrays for 'LATIC' routine.
```

WKLAT

src/gmvp/INC/_WKMIR

```
1: C*
2: C*.... POINTERS TO WORKING ARRAYS : MIRROR
3: C*
4:      COMMON /WKMIR/  P4(30)
5:      INTEGER      P4
6: C*
7: C P4(30) I4   Pointers to working arrays for 'MIRROR' routine.
```

SAFE

src/gmvp/INC/_WKNXT

```
1: C*
2: C* .... POINTERS TO WORKING ARRAYS : NXTEE CALLED IN ACTION
3: C*
4:      COMMON /WKNXT/ PC(30), PF(50), PS(20), PL(20)
5:      INTEGER      PC, PF, PS, PL
6: C*
7: C PC(30) I4   Pointers to working arrays : NXTEC  called in ACTION
8: C PF(50) I4   Pointers to working arrays : NEEFLI called in NXTEE
9: C PS(20) I4   Pointers to working arrays : NEESEA called in NXTEE
10: C PL(20) I4   Pointers to working arrays : LATICE called in NXTEE
11: C*
```

SAFE

src/gmvp/INC/_WKSE1

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEAOE .....
3: C*
4:      COMMON /WKSE1/ P2(30)
5:      INTEGER      P2
6: C*
7: C P2(30) I4  Pointers to working arrays for 'SEAOE' routine.
```

SEA

src/gmvp/INC/_WKSEA

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEARCH
3: C*
4:      COMMON /WKSEA/ P7(30)
5:      INTEGER      P7
6: C*
7: C P7(30) I4 Pointers to working arrays for 'SEARCH' routine.
```

SEARCH

src/gmvp/INC/_WKSOU

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS FOR SOURCE
3: C*
4:      COMMON /WKSOU/  P1(30)
5:      INTEGER      P1
6: C*
7: C P1(30) I4  Pointers to working arrays for 'SOURCE' routine.
```

SAFE

src/gmvp/INC/_WKTLO

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR TALLYO   NOV/14/91
3: C*
4:      COMMON /WKTLO/  P9(30)
5:      INTEGER      P9
6: C*
7: C P9(30) I4  Pointers to working arrays for 'TALLYO' routine.
```

SAFE

src/gmvp/INC/_WKTLS

```
1: C*<MVP>
2: C*  .... POINTERS TO WORKING ARRAYS FOR TALSUM   NOV/14/91
3: C*
4:      COMMON /WKTLS/  P8(30)
5:      INTEGER      P8
6: C*
7: C P8(30) I4  Pointers to working arrays for 'TALSUM' routine.
```

SAFE

```

1: C*
2: C*... PARTICLE BANK POINTERS
3: C*
4:      integer MDBNK, MIBNK
5:      integer MDFBK, MIFBK
6:
7: C      parameter ( MDBNK =6 , MIBNK = 8 )
8: C      parameter ( MDFBK =6 , MIFBK = 8 )
9: C ... May 1999 ... extended
10: C      index 11 to 16 is assinged to custom use ...
11:
12:      parameter ( MDBNK =20 , MIBNK = 20 )
13:      parameter ( MDFBK =20 , MIFBK = 20 )
14:
15:      COMMON /XBANK/
16:      & LXXX, LYXX, LZZZ, LAAA, LBBB, LCCC, LWWW, LIZZ, LIGG, LTTT, LITT,
17:      & LLSSF, LLEVLL, LLZZ, LLPOS, LXIM, LLCRS,
18:      & LIBSPC, LIBREG,
19:      & LDBNK, KDBNK(0:MDBNK), LIBNK, KIBNK(0:MIBNK),
20:      & LIFISB
21:
22: C*
23: C XXX(NBANK)      R8      X-position      (cm)
24: C YYY(NBANK)      R8      Y-position      (cm)
25: C ZZZ(NBANK)      R8      Z-position      (cm)
26: C AAA(NBANK)      R8      X-direction
27: C BBB(NBANK)      R8      Y-direction
28: C CCC(NBANK)      R8      Z-direction
29: C WWW(NBANK)      R4      Weight
30: C IZZ(NBANK)      I4      Zone number
31: C IGG(NBANK)      I4      Energy group
32: C TTT(NBANK)      R8      Time              (msec)          <JTIME>
33: C ITT(NBANK)      R4      Time bin          <JTIME>
34: C XIM(NBANK)      R4      Old importance    <JIMP>
35: C***** LATTICE PARAMETER BANK < JLATT > *****
36: C LEVL(NBANK)      I4      Current lattice level number
37: C LZZ (NBANK,NEST) I4      Zone number to return
38: C LPOS(NBANK,NEST) I4      Position number in lattice
39: C LCRS(NBANK,NEST) I4      Flag to particles whose flight track may cross$
40: C                      $ lattice-frame. (hexagonal lattice)
41: C KLSF(NBANK)      I4      Surface data pointer if particle is on-surface
42: C*
43: C**** Other optional bank parameters and indices *****
44: C* added July 1998
45: C*
46: C DBNK(NBANK,*) R8 optional bank parameters in floating point $
47: C                  $(double precision) form.
48: C MDBNK I4 (parameter constant) number of optional bank parameters $
49: C                  $(DBNK) in floating point$
50: C                  $(double precision) form. Currently this is 10.
51: C KDBNK(0:MDBNK) I4 entry #'s of bank parameters in DBNK array.
52: C When LDBNK(i) is zero, no bank data are stored for i'th $
53: C $ property.
54: C
55: C $ KDBNK(0) : total number of DBNK entries
56: C $ DBNK(*,KDBNK(1)) : particle weight on birth
57: C $ DBNK(*,KDBNK(2)) : time of birth
58: C $ DBNK(*,KDBNK(3)) : energy on birth
59: C $ DBNK(*,KDBNK(4):KDBNK(4)+NEST) : distance to lattice frames
60: C (required when lattice frames or cells may
61: C intersect each other. JFISX .ne. 0 )
62: C DBNK(*,KDBNK(4)+i-1) is distance to lattice
63: C frame of i'th level. i=NEST is reserved for
64: C some future extension.
65: C $ DBNK(*,KDBNK(5):KDBNK(5)+3*NEST-1) :
```

```

66: C Position of STG cell in coordinate system
67: C of STG region frame.
68: C $ DBNK(*,KDBNK(6)+6*NEST-1) :
69: C Position(1-3) on frame boundary to which particles
70: C should reach (not the current position) and
71: C flight direction(4-6) in frame.
72: C of each lattice level. Those of level 1 are
73: C identical to absolute corrodinates and direction.
74: C $ DBNK(*,KDBNK(7)+2*NTSRF) :
75: C distance and crossing angle cosine to tally-surface
76: C $ DBNK(*,KDBNK(7)+IS-1) : distance to tally-surface IS
77: C $ DBNK(*,KDBNK(7)+NTSRF+IS-1) : cosine of crossing angle.
78: C for normal vector to tally surface IS.
79: C Positive for inside to outside crossing.
80: C Negative for outside to inside crossing.
81: C
82: C $ Elements KDBNK(11) to KDBNK(16) is reserved for custom use by
83: C users. MVP/GMVP developers doesn't assign any attributes
84: C to these indices.
85: C*
86: C IBNK(NBANK,*) I4 optional bank parameters in integer form.
87: C MIBNK I4 (parameter constant) number of optional bank parameters
88: C (IBNK) in integer form.
89: C KIBNK(0:MIBNK) I4 entry #'s of bank parameters in IBNK array.
90: C When KIBNK(i) is zero, no bank data are stored for i'th $
91: C $property.
92: C*
93: C $ KIBNK(0) : total number of IBNK entries
94: C $ IBNK(*,KIBNK(1)) : source set # on birth
95: C $ IBNK(*,KIBNK(2)) : region # on birth
96: C $ IBNK(*,KIBNK(3)) : zone # on birth
97: C $ IBNK(*,KIBNK(4)) : generation of particle (primary,secondary,$
98: C $tertiary,...)
99: C $ IBNK(*,KIBNK(5)) : flight count in a flight path.
100: C Required when lattice frames or cells may
101: C intersect each other. JFISX.ne.0.
102: C Set -1 when particles are born from sources.
103: C $ IBNK(*,KIBNK(6):KIBNK(6)+NEST-1) : lattice level giving
104: C the smallest distance until the level
105: C $ IBNK(*,KIBNK(7)) : type of Nearest Neighbour Distribution (NND)
106: C if STG particle position sampling is
107: C required.
108: C $ 1 = on the surface of STG particle.
109: C $ 2 = from base material (matrix) region.
110: C $ 3 = on outer surface if STG region.
111: C $ 0 = no NND sampling is required.
112: C $ IBNK(*,KIBNK(8)) : save event # of last JDEBG(6) events for
113: C debug.
114: C**** defined in Sep 1999 ****
115: C $ IBNK(*,KIBNK(9)+NMKREG) : flag for "marker region" passing
116: C*
117: C $ IBNK(*,KIBNK(10)+2*NTSRF) : tally-surface side flag
118: C and angle bin
119: C $ IBNK(*,KIBNK(10)+IS-1) : tally-surface (IS) side flag
120: C $ 0 : flight path do not cross the surface
121: C $ +1 : inside of surface
122: C $ and distance is calcaulted.
123: C $ -1 : outside of surface
124: C $ and distance is calcaulted.
125: C $ +2 : inside on surface but distance is
126: C $ not calcaulted yet.
127: C $ -2 : outside of surface but distance is
128: C $ not calcaulted yet.
129: C $ 3 : inside/outside is unknown
130: C $ and distance is not calcaulted.

```

src/gmvp/INC/_XBANK

```

131: C      $      IBNK(*,KIBNK(10)+NTSRF+IS-1) : angle bin #
132: C      $      for ANGLB(1:NANGLE+1)
133: C
134: C      $ Elements KIBNK(11) to KIBNK(16) are reserved for custom use by
135: C      users. MVP/GMVP developers doesn't assign any attributes
136: C      to these indices.
137: C* ... added 8 Feb 2000 ...
138: C IFISB(NHIST) I4 store index pointers to fission bank arrays for
139: C      fission sources selected for a batch.
140: C*
141: C*=====
142: C*
143: C      COMMON /XBANK/
144: C      & LXXXF,LYYYF, LZZZF, LIZZF, LKLSFF,
145: C      & LLEVL, LLZZF, LLPOSF, LLCRSF
146: C      & LIBSPF, LIBRGF,
147: C      & LDFBK, KDFBK(0:MDFBK), LIFBK, KIFBK(0:MIFBK)
148: C***** FISSON PARTICLE BANK *****
149: C XXXF(NFBANK) R8 X-position (cm)
150: C YYYF(NFBANK) R8 Y-position (cm)
151: C ZZZF(NFBANK) R8 Z-position (cm)
152: C IZZF(NFBANK) I4 Zone number
153: C KLSFF(NFBANK) I4 Surface data pointer if particle is on-surface
154: C* **** FISSON BANK **** <JLAMI> & <JFISS>
155: C LEVLF(NFBANK) I4 Current lattice level number$
156: C      $ for banked fission particles.
157: C LZZF (NFBANK,NEST) I4 Zone number to return
158: C LPOSF(NFBANK,NEST) I4 Position number in lattice$
159: C      $ for banked fission particles.
160: C LCRSF(NFBANK,NEST) I4 Flag to particles whose flight-track might$
161: C      $ cross lattice-frame. (hexagonal lattice)$
162: C      $ for banked fission particles.
163: C*
164: C***** Nov 1998 : DFBK & IFBK is added ***
165: C*
166: C DFBK(NFBANK,*) R8 optional bank parameters in floating point $
167: C      $(double precision) form for banked fission particles.
168: C      See KDFBK,
169: C MDFBK I4 (parameter constant) number of optional bank parameters $
170: C      $(DFBK) in floating point$
171: C      $(double precision) form for banked fission particles.
172: C      Currently this is 5.
173: C KDFBK(0:MDFBK) I4 entry #'s of bank parameters in DFBK array.
174: C      When KDFBK(i) is zero, no bank data are stored for i'th $
175: C      $ property. See KDBNK(*) for description of parameters.
176: C IFBK(NFBANK,*) R8 optional bank parameters in integer $
177: C      $(double precision) form for banked fission particles.
178: C      See KIFBK,
179: C MIFBK I4 (parameter constant) number of optional bank parameters $
180: C      $(IFBK) in integer form for banked fission particles.
181: C      Currently this is 5.
182: C KIFBK(0:MIFBK) I4 entry #'s of bank parameters in IFBK array.
183: C      When KIFBK(i) is zero, no bank data are stored for i'th $
184: C      $ property. See KIBNK(*) for description of parameters.
185: C*=====
186: C*
187: C*..... IMAGINARY PARTICLES ...
188: C*
189: C      COMMON /XIBANK/
190: C      & LXXXI,LYYYI, LZZZI, LAAAI, LBBBI, LCCCI,
191: C      & LWWWI, LIZZI, LIGGI, LTTTI, LITTI,
192: C      & LLEVLI, LLZZI, LLPOSI,
193: C      & LLCRSI, LKLSFI, LKDETP, LOPTI, LPATH ,
194: C      & LDBNKI, LIBNKI
195: C*

```

```

196: C*5/27/91 === BANK DATA FOR IMGINARY PATICLES FOR NEXT EVENT ESTIMATOR
197: C KDETP(IMPMAX) I4 Detector numbers for each imaginary particles.
198: C OPTI(IMPMAX) R8 Optical path for imaginary particles.
199: C PATH(IMPMAX) R8 Path length for imaginary particles.
200: C*

```

src/gmvp/INC/_XWORK

```
1: C*
2: C*..... VECTOR WORKING AREA POINTER .....
3: C*
4:      COMMON /XWORK/ NWORK,LWORK,NWORKB,LWORKB
5: C*
6: C*      .... PARAMETER2 OTHER THAN LWORK ARE MEANINGLESS (3/12/1991)
7: C*      SEE SUBROUTINE WRKARY.
8: C*
9: C* NWORK      I4  NUMBER OF WORK AREA WHOSE LENGTH IS NBANK*4 BYTE.
10: C* WORK(NBANK,NWORK)      R4/I4/R8  WORK AREA
11: C* NWORKB      I4  NUMBER OF WORK AREA WHOSE LENGTH IS (NZONE+1)*4
12: C*                                BYTE.
13: C* WORKB(NZONE+1,NWORKB)  R4/I4/R8  WORK AREA
14: C*
15: C LWORK I4 Pointer of the beginning point of working area in variable$
16: C      $-length data area in COMMON /ARRAY/.
```

src/shared/abs2zn.f

```

1:      subroutine ABS2ZN(IOW, A, H, JDEBG, JLATT,JTLT,JHLAT, NNLOST,
2:      J JMODE, IERR, NN, X1, Y1, Z1, A1, B1, C1,
3:      & MEMZN, NMEMZN, JMEMZN,
4:      B IZZ1, LEVL1, LZZ1, LPOS1, LCRS1, IBREG1, IBSPC1)
5: C=====
6: C  PURPOSE: find zone/region from positions of particles given in
7: C  absolute coordinates.
8: C  To be used in zone/region/material dependent rejection sampling
9: C  in source generation phase and/or calculation of uncollided
10: C  contribution to next event type detectors from source particles.
11: C
12: C  Obtained particle attribute data may not be used in the
13: C  processes after this routine.
14: C
15: C  CALLED IN: anywhere
16: C  CALLS: ACTA2Z
17: C=====
18: C arguments (i=input, o=output, w=work)
19: C i IOW : message printout I/O unit
20: C i A(*) : dynamic memory array (task shared)
21: C i H(*) : dynamic memory array (task local)
22: C i JDEBG : debug option flag
23: C i JLATT : lattice geometry used if non zero
24: C i JTLT : frame dependent tally mode if non zero
25: C i JHLAT : hexagonal lattice geometry used if non zero
26: C o NNLOST : number of lost particles under this routine.
27: C
28: C i JMODE : running mode.
29: C bits (1/0=on/off)
30: C 1 : returns region numbers IBREG1.
31: C 2 : returns lattice geometry parameters also.
32: C 3 : returns translated in-cell coordinates on X1,Y1,Z1.
33: C 4 : direction is given on A1,B1,C1 and returns translated
34: C in-cell directions on A1,B1,C1.
35: C
36: C o IERR : error codes. non-zero means some errors have occurred.
37: C
38: C i NN : number of particles whose zones/attributes are determined
39: C i(o) X1(NN),Y1(NN),Z1(NN) : absolute coordinates of the particles.
40: C i(o) A1(NN),B1(NN),C1(NN) : direction cosines of the particles.
41: C
42: C i MEMZN(NMENZN) : "MEMZN" type of zone memory array.
43: C i JMEMZN : use MEMZN if non-zero.
44: C
45: C o IZZ1(NN), LEVL1(NN),LZZ1(NN,NEST),LPOS1(NN,NEST),LCRS1(NN,NEST)
46: C IBREG1(NN), IBSPC1(NN,0:NEST) : determined attributes.
47: C arrays other than IZZ1 are optional ( they may have dummy
48: C addresses ...).
49: C IZZ1 is set to negative when a zone is not found for
50: C the particle.
51: C set IZZ1=0 and LPOS1(LEVL1)=-1 when a zone is STG region.
52: C
53: C-----
54: C important common variables:
55: C NBNK2 : size of banks, stacks, working arrays for "underground" level
56: C modules.
57: C=====
58: C
59: C .... particle attributes for which zones etc. are determined.....
60: C
61:      real*8 X1(NN), Y1(NN), Z1(NN)
62:      real*8 A1(NN), B1(NN), C1(NN)
63: C
64:      integer MEMZN(NMEMZN)
65: C

```

```

66: C .... returned attributes. Data other than IZZ1 is optional .....
67: C
68:      integer IZZ1(NN)
69:      integer LEVL1(NN), LZZ1(NN,NEST),
70:      & LPOS1(NN,NEST), LCRS1(NN,NEST)
71:      integer IBREG1(NN), IBSPC1(NN,0:NEST)
72: C
73:      integer JDEBG(*)
74: C
75: CCC include 'INC/_ARRAY'
76:      real A(*)
77:      real H(*)
78: C
79:      include 'INC/_SIZES'
80:      include 'INC/_PGEOM'
81:      include 'INC/_PTLSP'
82: C
83: C ... pointers to temporary bank/stacks and working arrays (PAZ(*))
84: C under this routine. Must synchronize with initialization in WKAA2Z
85: C routine.
86: C
87: C ... temporary particle banks ...
88: C
89: C XXX,YYY,ZZZ : h(paz(1)),h(paz(2)),h(paz(3))
90: C AAA,BBB,CCC : h(paz(4)),h(paz(5)),h(paz(6)) (maybe unused)
91: C IZZ,LEVL,LZZ,LPOS : h(paz(7)),h(paz(8)),h(paz(9)),h(paz(10))
92: C LCRS,IBREG, IBSPC : h(paz(11)),h(paz(12)),h(paz(13))
93: C
94: C ... temporary particle stack
95: C
96: C LSEND : h(paz(14))
97: C LSSRC(NBNK2), IZ NXT(NBNK2), NNXT(NZONE+1):
98: C h(paz(15)),h(paz(16)),h(paz(17))
99: C LSLAT(NBNK2), IZLAT(NBNK2), NXLT(NLBZ+1)
100: C h(paz(18)),h(paz(19)),h(paz(20))
101: C
102: C
103:      include 'INC/_WKAB2Z'
104: C
105: C-----
106: C
107:      NNLOST = 0
108:      do 100 II=1,NN,NBNK2
109:          II2 = min(NN,II+NBNK2-1)
110: C
111: C ---- ACTA2Z is a control routine like the ACTION routine ----
112: C
113:      call ACTA2Z(IOW, A, H, JMODE,JDEBG,JLATT,JTLT,JHLAT,ILOST,
114:      & X1, Y1,Z1, A1, B1, C1,
115:      & MEMZN, NMEMZN, JMEMZN,
116:      & IZZ1, LEVL1, LZZ1,
117:      & LPOS1, LCRS1, IBREG1, IBSPC1,
118:      & NN, II, II2, IERR,
119:      & A(LIPCEL),A(LSDA), A(LKZDA), A(LKZAA), A(LKZMAT),A(LMLBZZ),
120:      & A(LKZREG),A(LKCELL),A(LISUSP),
121:      & H(PAZ(1)),H(PAZ(2)),H(PAZ(3)),
122:      & H(PAZ(4)),H(PAZ(5)),H(PAZ(6)),
123:      & H(PAZ(7)),H(PAZ(8)),H(PAZ(9)),H(PAZ(10)),
124:      & H(PAZ(11)),H(PAZ(12)),H(PAZ(13)),
125:      & H(PAZ(14)),
126:      & H(PAZ(15)),H(PAZ(16)),H(PAZ(17)),
127:      & H(PAZ(18)),H(PAZ(19)),H(PAZ(20)),
128:      & H(PAZ(21)),H(PAZ(22)),H(PAZ(23)),
129:      & H(PAZ(24)),H(PAZ(25)),H(PAZ(26)))
130: C

```

src/shared/abs2zn.f

```
131:          NNLOST = NNLOST + ILOST
132: 100 continue
133: C
134:      return
135:      end
```

SAFE

src/shared/acta2z.f

```

1:      subroutine ACTA2Z( IOW,A,H,JMODE,JDEBG, JLATT,JTLT,JHLAT, ILOST,
2:      I      X1, Y1,Z1, A1,B1,C1,
3:      &      MEMZN, NMEMZN, JMEMZN,
4:      I      IZZ1, LEVL1, LZZ1,
5:      I      LPOS1, LCRS1, IBREG1, IBSPC1,
6:      &      NN, II, I12, IERR,
7:      G      IPCEL, SDA, KZDA, KZAA, KZMAT,MLBZZ, KZREG, KCELL, ISUSP,
8:      B      XXX,YYY,ZZZ,
9:      B      AAA,BBB,CCC,
10:     B      IZZ,LEVL,LZZ,LPOS,
11:     B      LCRS,IBREG, IBSPC,
12:     S      LSEND,
13:     S      LSSRC,IZNXT,NNXT,
14:     S      LSLAT, IZLAT, NXLT,
15:     W      X,Y,Z,IFI,IFL, INK1)
16: C=====
17: C PURPOSE: control routine or procedures to find zone/region from
18: C particle positions given in absolute coordinates.
19: C This is a miniature version of ACTION routine and should be used
20: C in zone/region/material dependent rejection sampling
21: C in source generation phase, etc.
22: C Obtained particle attribute data may not be used in the
23: C processes after this routine.
24: C
25: C << This routine cannot handle STG-region (lattice) >>
26: C return with IZZ1=0 and LPOS1(LEVL1)=-1
27: C
28: C CALLED IN: ABS2ZN
29: C CALLS      : JUDGE, LATDWN
30: C=====
31: C arguments (i=input, o=output, w=work)
32: C
33: C i IOW : message printout I/O unit
34: C io A(*) : dynamic memory area(task shared)
35: C io H(*) : dynamic memory area(task local)
36: C i JMODE : running mode.
37: C      0 : only returns zone numbers IZZ1 for each particle.
38: C
39: C Optional information is returned if specified bit is non-zero.
40: C
41: C bit 1 : returns region numbers in IBREG1.
42: C bit 2 : returns lattice geometry parameters on LEVL1, LZZ1,
43: C LPOS1, LCRS1, IBREG1 & IBSPC1.
44: C bit 3 : returns translated in-cell coordinates on X1,Y1,Z1.
45: C bit 4 : direction is given on A1,B1,C1 and returns translated
46: C in-cell directions on A1,B1,C1.
47: C
48: C i JDEBG : debug option flag
49: C i JLATT : lattice geometry used if non zero
50: C i JTLT : frame dependent tally mode if non zero
51: C i JHLAT : hexagonal lattice geometry used if non zero
52: C o ILOST : number of lost particles
53: C      (no message of lost particle in this routine)
54: C
55: C i(o) X1(NN), Y1(NN),Z1(NN) : absolute coordinates of the particles.
56: C i(o) A1(NN), B1(NN),C1(NN) : absolute coordinates of the particles.
57: C
58: C i MEMZN(NMEMZN) : "MEMZN" type of zone memory array.
59: C i JMEMZN : use MEMZN if non-zero.
60: C
61: C o IZZ1(NN), LEVL1(NN),LZZ1(NN,NEST),LPOS1(NN,NEST),LCRS1(NN,NEST)
62: C IBREG1(NN), IBSPC1(NN,0:NEST) : determined attributes.
63: C
64: C Arrays other than IZZ1 may not have actual memory area.
65: C They must have real memory area when return values are

```

```

66: C      required through JMODE option.
67: C
68: C      IZZ1 is set to negative when a zone is not found for
69: C the particle. set to zero when a zone is STG region.
70: C
71: C i NN : number of particles whose zones/attributes are determined
72: C i I1,I12 : X1(I1:I12),Y1(I1:I12) ... is processed in this routine.
73: C      I12-I1-1 must not exceed NBNK2. It is responsibility of routines
74: C calling this routine not to violate this rule.
75: C
76: C o IERR : error code
77: C
78: C i IPCEL(*), SDA(NSDA) ... : geometry data for JUDGE routine etc.
79: C
80: C w XXX,YYY,ZZZ,AAA,BBB,CCC,IZZ,LEVL,LZZ,LPOS,LCRS,IBREG,IBSPC :
81: C Temporary particle bank used within this routine.
82: C w LSEND(NBNK2) : temporary stack of "finished" particles
83: C w LSSRC,IZNXT,NNXT : temporary "NEXT ZONE SEARCH" stack.
84: C w LSLAT,IZLAT,NXLT : temporary lattice search stack.
85: C
86: C w X,Y,Z,IFI,IFL: working array of size NBNK2
87: C
88: C=====
89: C important common variables:
90: C NBNK2 : size of banks, stacks, working arrays for "underground" level
91: C modules.
92: C=====
93: CCCC include 'INC/_ARRAY'
94: C      real A(*)
95: C      real H(*)
96: C      include 'INC/_SIZES'
97: C      include 'INC/_PGEOM'
98: C      include 'INC/_PTLSP'
99: C ... working array pointers PAZ & PAZL
100: C include 'INC/_WKAB2Z'
101: C
102: C .... particle attributes for which zones etc. are determined.....
103: C
104: C      real*8 X1(NN), Y1(NN), Z1(NN)
105: C      real*8 A1(NN), B1(NN), C1(NN)
106: C
107: C      integer MEMZN(NMEMZN)
108: C      integer IPCEL(*)
109: C      real*8 SDA(NSDA)
110: C      integer KZDA(NZDA,2), KZAA(NZONE+1), KZMAT(NZONE), MLBZZ(NZONE)
111: C      integer KZREG(NZONE), KCELL(NZONE)
112: C      integer ISUSP(NSUZON,NSPACE)
113: C
114: C .... returned attributes. Data other than IZZ1 is optional .....
115: C
116: C      integer IZZ1(NN)
117: C      integer LEVL1(NN), LZZ1(NN,NEST), LPOS1(NN,NEST), LCRS1(NN,NEST)
118: C      integer IBREG1(NN), IBSPC1(NN,0:NEST)
119: C
120: C ... temporary bank/stacks and working arrays
121: C under this routine. Must synchronize with initialization in WKAA2Z
122: C routine.
123: C
124: C ... temporary particle banks ...
125: C
126: C      real*8 XXX(NBNK2), YYY(NBNK2), ZZZ(NBNK2)
127: C      real*8 AAA(NBNK2), BBB(NBNK2), CCC(NBNK2)
128: C      integer IZZ(NBNK2), LEVL(NBNK2), LZZ(NBNK2,NEST), LPOS(NBNK2,NEST)
129: C      integer LCRS(NBNK2,NEST)
130: C      integer IBREG(NBNK2), IBSPC(NBNK2,0:NEST)

```

src/shared/acta2z.f

```

131: C
132: C ... temporary stacks ....
133: C
134:     integer LSEND(NBNK2)
135:     integer LSSRC(NBNK2), IZNXT(NBNK2), NNX(NZONE+1)
136:     integer LSLAT(NBNK2), IZLAT(NBNK2), NXLT(NLBZ+1)
137: C
138:     integer JDEBG(*)
139: C
140: C .... working area ....
141: C
142:     real*8 X(NBNK2), Y(NBNK2), Z(NBNK2)
143:     logical IFI(NBNK2), IFL(NBNK2)
144:     integer IWK1(NBNK2)
145: C
146:     include 'INC/_PMLATT'
147:     include 'INC/_SLATT'
148: C
149: C-----
150: C
151:     IERR = 0
152:     ILOST = 0
153: C
154: C ... copy coordinate data in "temporary" bank arrays.
155: C
156:     NNP = II2 - II + 1
157: C
158:     if ( NNP.gt.NBNK2 ) then
159:         write(IOW,*) 'XXX(routine ACTA2Z) the number of data to be'
160:         & ' processed (NNP=', NNP, ' for X1(', II, ':', II2,
161:         & ') etc.)',
162:         & ' exceeds the size of working arrays (NBNK2=', NBNK2,
163:         & ')',
164:         stop 666
165:     end if
166:
167: *VOCL LOOP,NOVREC
168:     do 100 I = 1, NNP
169:         XXX(I) = X1(II+I-1)
170:         YYY(I) = Y1(II+I-1)
171:         ZZZ(I) = Z1(II+I-1)
172:         IZZ(I) = 0
173: C ... sent to "search" stack at first
174:         LSSRC(I) = I
175: C ... IWK1 is index in bank
176:         IWK1(I) = I
177:         X(I) = XXX(I)
178:         Y(I) = YYY(I)
179:         Z(I) = ZZZ(I)
180:     100 continue
181: C
182:         JTRDIR = 0
183:         JTRPOS = 0
184:         if ( JLATT.ne.0 ) then
185:             do 110 I = 1, NNP
186:                 LEVL(I) = 0
187:             110 continue
188:             do 130 K = 1, NEST
189: *VOCL LOOP,NOVREC
190:                 do 120 I = 1, NNP
191:                     LZZ(I,K) = 0
192:                     LPOS(I,K) = 0
193:                     if ( JHLAT.ne.0 ) LCRS(I,K) = 0
194:                 120 continue
195:             130 continue

```

```

196: C
197: C ... clear lattice serach array
198:     do 140 K = 1, NLBZ
199:         NXLT(K) = 0
200:     140 continue
201:         NXLT(NLBZ+1) = 0
202: C
203: C ... translated directions are returned
204: C
205:         JTRDIR = MOD(JMODE,2**4) /2**3
206:         if ( JTRDIR.ne.0 ) then
207: *VOCL LOOP,NOVREC
208:             do 150 I = 1, NNP
209:                 AAA(I) = A1(II+I-1)
210:                 BBB(I) = B1(II+I-1)
211:                 CCC(I) = C1(II+I-1)
212:             150 continue
213:             end if
214:             JTRDIR = MOD(JMODE,2**4) /2**3
215:             JTRPOS = MOD(JMODE,2**3) /2**2
216:         end if
217: C
218:         NEND = 0
219: C
220: C-----
221: C first step : determine zone in root universe
222: C-----
223: C
224:         MMZ = NZONE
225:         if ( JLATT.ne.0 ) MMZ = IPCEL(1) - 1
226:         KKZ = 1
227:         MEM = 1
228:         INN = NNP
229: C
230:         do 220 M = 1, MMZ
231:             IZ = M
232:             if ( JMEMZN.ne.0 ) then
233: C
234: C ... do not update MEMZN in this routine...
235: C
236:                 if ( MEM.le.NMEMZN.and.MEMZN(MEM).ne.0 ) then
237:                     IZ = MEMZN(MEM)
238:                     MEM = MEM + 1
239:                 else
240:                     do 170 KZ = KKZ, MMZ
241:                         do 160 K = 1, NMEMZN
242:                             if ( MEMZN(K).ne.0.and.KZ.eq.MEMZN(K) ) go to 170
243:                         160 continue
244:                         go to 180
245:                     170 continue
246:                     KZ = MMZ
247:                     180 IZ = KZ
248:                     KKZ = IZ + 1
249:                 end if
250:             end if
251: C
252: C =====
253:         call JUDGE( 'ACTA2Z', IZ, INN, X, Y, Z, IFI, SDA, KZDA, KZAA,
254:         & NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, .false., DUMMY,
255:         & DUMMY, DUMMY )
256: C =====
257: C
258: C .... lattice search stack ...
259: C
260: CCCCC if ( JLATT.ne.0.and.KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998 )

```

src/shared/acta2z.f

```

261: CCCCC&      then
262:      if ( JLATT.ne.0.and.ISLATT(KZMAT(IZ)) ) then
263: C
264:      IN      = 0
265:      ILAT    = NXLT(NLBZ+1)
266: *VOCL LOOP,NOVREC
267:      do 190 I = 1, INN
268:      if ( IFI(I) ) then
269:      IZZ(IWK1(I)) = IZ
270:      IN      = IN + 1
271:      LSLAT(ILAT+IN) = IWK1(I)
272:      IZLAT(ILAT+IN) = MLBZZ(IZ)
273: C      ... always set IBREG (from Jun 2000)
274:      IBREG(IWK1(I)) = KZREG(IZ)
275:      if ( JTLLT.ne.0 ) then
276:      IBSPQ(IWK1(I),0) = 0
277:      end if
278:      end if
279: 190      continue
280:      NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + IN
281:      NXLT(NLBZ+1) = NXLT(NLBZ+1) + IN
282: C
283: C      .... zone found in root universe ...
284: C      ( material may be -1000 or reflection material.)
285: C
286:      else
287: *VOCL LOOP,NOVREC
288:      do 200 I = 1, INN
289:      if ( IFI(I) ) then
290:      IZZ1(II+IWK1(I)-1) = IZ
291:      NEND = NEND + 1
292:      LSEND(NEND) = IWK1(I)
293:      end if
294: 200      continue
295:      end if
296: C
297: C      ... condense working array ...
298: C
299:      IN      = 0
300: *VOCL LOOP,NOVREC
301:      do 210 I = 1, INN
302:      if ( .not.IFI(I) ) then
303:      IN      = IN + 1
304:      X(IN)   = X(I)
305:      Y(IN)   = Y(I)
306:      Z(IN)   = Z(I)
307:      IWK1(IN) = IWK1(I)
308:      end if
309: 210      continue
310:      INN      = IN
311:      if ( INN.eq.0 ) go to 230
312: 220      continue
313: C
314: C      ... lost particles in root universe ...
315: C
316: 230      continue
317:      if ( INN.ne.0 ) then
318:      IERR      = 1
319:      ILOST     = ILOST + INN
320: *VOCL LOOP,NOVREC
321:      do 240 I = 1, INN
322:      IZZ1(II+IWK1(I)-1) = -1
323: 240      continue
324:      end if
325: C

```

```

326: C ... set regions and/or lattice parameters ...
327: C
328:      if ( MOD(JMODE,2**1).ne.0 ) then
329: *VOCL LOOP,NOVREC
330:      do 250 I = 1, NEND
331:      IBREG1(II+LSEND(I)-1) = KZREG(IZZ1(II+LSEND(I)-1))
332: 250      continue
333:      end if
334: C
335:      if ( MOD(JMODE,2**2)/2**1.ne.0 ) then
336: *VOCL LOOP,NOVREC
337:      do 260 I = 1, NEND
338:      LEVL1(II+LSEND(I)-1) = 0
339: Ccccccc      LPOS1(II+lsend(i)-1,0) = 0
340: Ccccccc      if( JHLAT.ne.0 ) LCRS1(II+lsend(i)-1,0) = 0
341: 260      continue
342:      end if
343: C
344: C
345: C      ... all particles are in root universe ....
346: C
347:      if ( NEND+ILOST.eq.NNP ) return
348: C
349: C
350: C=====
351: C      start zone search of particles in lattice region
352: C=====
353: C
354:      NEND0 = NEND
355: C
356:      do 270 IZ = 1, NZONE
357:      NNXT(IZ) = 0
358: 270      continue
359:      NNXT(NZONE+1) = 0
360: C
361: C      .....
362: C
363: 280      continue
364: C
365:      if ( MOD(JDEBG(3),2).ne.0 ) then
366:      write(IOW,*) '(ACTA2Z) NNP ', NNP, ' NEND ', NEND, ' ilost ',
367:      &      ILOST
368:      write(IOW,*) '(ACTA2Z) LSEND ', (LSEND(I),I=1,NEND)
369:      write(IOW,*) '(ACTA2Z) IZZ1(ended) ',
370:      &      (IZZ1(II+LSEND(I)-1),I=1,NEND)
371:      end if
372: C
373: C      ... MACT=2 : down lattice level, MACT=1 : search zone
374: C
375:      if ( NEND+ILOST.lt.NNP ) then
376:      if ( NXLT(NLBZ+1).gt.NNXT(NZONE+1) ) then
377:      MACT = 2
378:      else
379:
380:      MAXNZ = 0
381:      MZONE = 0
382:      do 290 IZ = 1, NZONE
383:      if ( NNXT(IZ).gt.MAXNZ ) then
384:      MZONE = IZ
385:      MAXNZ = NNXT(IZ)
386:      end if
387: 290      continue
388:      if ( MAXNZ.gt.NXLT(NLBZ+1) ) then
389:      MACT = 1
390:      else

```

src/shared/acta2z.f

```

391:          MACT      = 2
392:        end if
393:      end if
394: C
395:      if ( MOD(JDEBG(3),2).ne.0 ) then
396:        write(IOW,*) '(ACTA2Z) mact ', MACT, ' mzone ', MZONE
397:      end if
398: C
399: C-----
400: C ... reached actual zone or enter a lattice ...
401: C-----
402: C
403:      if ( MACT.eq.1 ) then
404: C
405:        IN      = 0
406:        do 300 I = 1, NNXT(MZONE+1)
407:          if ( IZNXT(I).eq.MZONE ) then
408:            IN      = IN + 1
409:            IWKL(IN) = LSSRC(I)
410:          end if
411: 300      continue
412: C
413:      if ( MOD(JDEBG(3),2).ne.0 ) then
414:        write(IOW,*) '(ACTA2Z) IN ', IN, ' NNXT(MZONE) ',
415:      &      NNXT(MZONE)
416:      end if
417: C
418:      do 310 I = 1, NNXT(MZONE)
419:        X(I) = XXX(IWKL(I))
420:        Y(I) = YYY(IWKL(I))
421:        Z(I) = ZZZ(IWKL(I))
422: 310      continue
423: C
424:        ICEL    = KCELL(MZONE)
425:        MMZ     = IPCEL(ICEL+1) - 1
426:        KKZ     = 1
427:        if ( ICEL.gt.0 ) KKZ = IPCEL(ICEL)
428: C
429:        INX     = NNXT(MZONE)
430:        do 350 IZ = KKZ, MMZ
431:          if ( IZ.eq.MZONE ) go to 350
432: C=====
433:          call JUDGE( 'ACTA2Z', IZ, INX, X, Y, Z, IFI, SDA, KZDA,
434:      &      KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
435:      &      .false., DUMMY, DUMMY, DUMMY )
436: C=====
437: C
438: C ... sent to lattice search ...
439: C
440: CCCCC      if ( KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998 ) then
441:      if ( ISLATT(KZMAT(IZ)) ) then
442:        IN      = 0
443:        ILAT    = NXLT(NLBZ+1)
444: *VOCL LOOP,NOVREC
445:        do 320 I = 1, INX
446:          if ( IFI(I) ) then
447:            IZZ(IWKL(I)) = IZ
448:            IN      = IN + 1
449:            LSLAT(ILAT+IN) = IWKL(I)
450:            IZLAT(ILAT+IN) = MLBZZ(IZ)
451:          end if
452: 320      continue
453:        NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + IN
454:        NXLT(NLBZ+1)    = NXLT(NLBZ+1) + IN
455: C

```

```

456: C ... actual zone is determined.
457: C
458:      else
459: *VOCL LOOP,NOVREC
460:        do 330 I = 1, INX
461:          if ( IFI(I) ) then
462:            IZZ1(II+IWKL(I)-1) = IZ
463:            NEND = NEND + 1
464:            LSEND(NEND) = IWKL(I)
465:          end if
466: 330      continue
467:        end if
468: C
469: C ... condense array ...
470: C
471:        IN      = 0
472: *VOCL LOOP,NOVREC
473:        do 340 I = 1, INX
474:          if ( .not.IFI(I) ) then
475:            IN      = IN + 1
476:            X(IN)    = X(I)
477:            Y(IN)    = Y(I)
478:            Z(IN)    = Z(I)
479:            IWKL(IN) = IWKL(I)
480:          end if
481: 340      continue
482:        INX = IN
483:        if ( INX.eq.0 ) go to 360
484: C
485: 350      continue
486: C
487: 360      continue
488: C
489:        if ( INX.ne.0 ) then
490:          IERR = 1
491:          ILOST = ILOST + INX
492: *VOCL LOOP,NOVREC
493:          do 370 I = 1, INX
494:            IZZ1(II+IWKL(I)-1) = -(1+ICEL)
495: 370      continue
496:          end if
497: C
498:          IIN      = 0
499: *VOCL LOOP,NOVREC
500:          do 380 I = 1, NNXT(NZONE+1)
501:            if ( IZNXT(I).ne.MZONE ) then
502:              IIN      = IIN + 1
503:              LSSRC(IIN) = LSSRC(I)
504:              IZNXT(IIN) = IZNXT(I)
505:            end if
506: 380      continue
507: C
508:        NNXT(NZONE+1) = NNXT(NZONE+1) - NNXT(MZONE)
509:        NNXT(MZONE) = 0
510: C
511: C-----
512: C ... enter a lattice and determine cell ...
513: C-----
514: C
515:      else if ( MACT.eq.2 ) then
516: C
517:        NEND1 = NEND
518:        call LATDWN( IOW, JDEBG, JVMNT, JTLT, JHLAT, NBNK2, NZONE,
519:      &      NLATT, NCELL, NLBZ, NEST, NSPACE, NUNV, DEPS, XXX,
520:      &      YYY, ZZZ, JTRDIR, AAA, BBB, CCC, IZZ, LEVL, LZZ,

```

src/shared/acta2z.f

```

521:      &          LPOS, LCRS, IBREG, IBSPC, LSSRC, IZNXT, NNXT, LSLAT,
522:      &          IZLAT, NXLT,
523:      &          LSEND, NEND,
524:      &          A(LKZMAT), A(LKCELL), A(LKZLBZ),
525:      &          A(LMLBZZ), A(LCELSZ), A(LSZLAT), A(LIDLAT),
526:      &          A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT),
527:      &          A(LIPCEL), A(LLTYP), A(LICTYP), A(LKDALT), A(LDALT),
528:      &          A(LKZREG), A(LKSPSU), A(LKTCSP), NKTCS, A(LKHLAT),
529:      &          X, Y, Z, H(PAZL(1)), H(PAZL(2)), H(PAZL(3)), IWK1,
530:      &          H(PAZL(4)), H(PAZL(5)) )
531: C
532: C ... in LATDWN, particles in STG region are treated as end particle.
533: C
534:       if ( NEND.gt.NEND1 ) then
535:         do 390 I = NEND1, NEND
536:           IZZ1(II+LSEND(I)-1) = 0
537: 390       continue
538:         end if
539: C
540:       end if
541: C
542:       go to 280
543:     end if
544: C
545: C-----
546: C
547:       if ( JTRPOS.ne.0 ) then
548: *VOCL LOOP,NOVREC
549:         do 400 I = NEND0 + 1, NEND
550:           X1(II+LSEND(I)-1) = XXX(LSEND(I))
551:           Y1(II+LSEND(I)-1) = YYY(LSEND(I))
552:           Z1(II+LSEND(I)-1) = ZZZ(LSEND(I))
553: 400       continue
554:         end if
555:       if ( JTRDIR.ne.0 ) then
556: *VOCL LOOP,NOVREC
557:         do 410 I = NEND0 + 1, NEND
558:           A1(II+LSEND(I)-1) = AAA(LSEND(I))
559:           B1(II+LSEND(I)-1) = BBB(LSEND(I))
560:           C1(II+LSEND(I)-1) = CCC(LSEND(I))
561: 410       continue
562:         end if
563: C
564: C ... set regions and/or lattice parameters ...
565: C
566:       if ( MOD(JMODE,2*1).ne.0 ) then
567: *VOCL LOOP,NOVREC
568:         do 420 I = NEND0 + 1, NEND
569:           ILV = LEVL(LSEND(I))
570:           KZ = IZZ1(II+LSEND(I)-1)
571:           if ( ILV.gt.0 ) then
572:             IBREG1(II+LSEND(I)-1) =
573:       &             ISUSP(KZREG(KZ),IBSPC(LSEND(I),ILV))
574:           else
575:             IBREG1(II+LSEND(I)-1) = KZREG(KZ)
576:           end if
577: 420       continue
578:         end if
579: C
580:       if ( MOD(JMODE,2*2)/2*1.ne.0 ) then
581: *VOCL LOOP,NOVREC
582:         do 430 I = NEND0 + 1, NEND
583:           LEVL1(II+LSEND(I)-1) = LEVL(LSEND(I))
584: 430       continue
585:         do 450 K = 1, NEST

```

```

586: *VOCL LOOP,NOVREC
587: C       do 440 I = 1, NNP
588:       do 440 I = 1, NEND
589:         LZZ1(II+LSEND(I)-1,K) = LZZ(LSEND(I),K)
590:         LPOS1(II+LSEND(I)-1,K) = LPOS(LSEND(I),K)
591:         if ( JHLAT.ne.0 ) then
592:           LCRS1(II+LSEND(I)-1,K) = LCRS(LSEND(I),K)
593:         end if
594: 440       continue
595: 450     continue
596:       end if
597: C
598:       return
599:     end

```

src/shared/angset.f

```

1:      subroutine ANGSET( MODE,  ANGLB, NANGLE, IPR, IMSG )
2: C=====
3: C Purpose: set/check angle bin data and convert from angle to cosine
4: C         if necessary.
5: C-----
6: C arguments ( i=input, o=output, w=work)
7: C
8: C i  MODE    operation mode
9: C
10: C      -2 : set uniform width angle bin from zero to 180 degree.
11: C      -1 : set uniform width cosine bin from 1 to -1
12: C      1 : ANGLB is input as cosine bin, check it.
13: C      2 : ANGLB is input as angle bin, convert it to cosine bin
14: C         and check it.
15: C
16: C      Other MODE value does only bin order and range check.
17: C
18: C io ANGLB(NANGLE+1) : angle bin
19: C i  NANGLE  : number of angle bins
20: C i  IMSG : message output I/O unit
21: C-----
22:      real*8 ANGLB(NANGLE+1)
23: C
24: C
25:      real*8 CDIFF
26:      parameter( CDIFF = 0.00001D0 )
27:      real*8 PAI
28: C
29: C-----
30: C
31:      call LABEL( IPR, 'ANGLE BIN' )
32: C
33: C ... set as uniform cosine bin
34: C
35:      if ( MODE.eq.-1 ) then
36:        do 100 I = 0, NANGLE
37:          ANGLB(I+1) = 1.0D0 - 2.0D0*(DBLE(I)/DBLE(NANGLE))
38:        100 continue
39:        return
40: C
41: C ... set as uniform angle bin between zero and 180
42: C
43:      else if ( MODE.eq.-2 ) then
44:        do 110 I = 0, NANGLE
45:          ANGLB(I+1) = 180.0D0/NANGLE*I
46:        110 continue
47:        return
48: C
49: C ... input as angle(degree). convert to cosine
50: C
51:      else if ( MODE.eq.2 ) then
52:        call R8PRNT('DEGREE',ANGLB,NANGLE+1,1,'ANGLE',' ',IPR)
53:        call CKODR8( ANGLB, NANGLE+1, 2, NERR )
54:        if ( NERR.gt.0 ) then
55: c##<2007/03/14:PN3:
56: c##      write(IMG,*)
57: c## &      'XXX ANGLE(...) is not given in increasing order.'
58: c##      write(IMG,'(1x,3x,10e12.5)') (ANGLB(I),I=1,NANGLE+1)
59:      write(IMG,901) (ANGLB(I),I=1,NANGLE+1)
60:      901 format(/' XXX(angset) ANGLE(...) is not given in increasing',
61:        & ' order.'
62:        &/1P,(4X,10E12.5))
63: c##>
64:      call CNTERR( 'FATAL' )
65:      end if

```

```

66:      PAI = ACOS(-1.0D0)
67:      do 120 I = 1, NANGLE + 1
68:        ANGLB(I) = COS(ANGLB(I)/180.0D0*PAI)
69:      120 continue
70: C
71: C ... input as cosine. (mode=1)
72: C
73:      else
74:        call CKODR8( ANGLB, NANGLE+1, 1, NERR )
75:        if ( NERR.gt.0 ) then
76: c##<2007/03/14:PN3:
77: c##      write(IMG,*)
78: c## &      'XXX COSINE(...) is not given in decreasing order.'
79: c##      write(IMG,'(1x,3x,10e12.5)') (ANGLB(I),I=1,NANGLE+1)
80:      write(IMG,902) (ANGLB(I),I=1,NANGLE+1)
81:      902 format(/' XXX(angset) COSINE(...) is not given in decreasing',
82:        & ' order.'
83:        &/1P,(4X,10E12.5))
84: c##>
85:      call CNTERR( 'FATAL' )
86:      end if
87:      end if
88: C
89: C
90: C ... check : angle must be from 0 to 180 degree
91: C
92: C
93:      if ( ABS(ANGLB(1)-1.0D0).gt.CDIFF ) then
94: c##<2007/03/14:PN3:
95: c##      write(IMG,*)
96: c## &      'XXX First data of COSINE(...) is not cosine(0): ',
97: c## &      ANGLB(1)
98:      write(IMG,903) ANGLB(1)
99:      903 format(/' XXX(angset) First data of COSINE(...) is not',
100:        & ' cosine(0): ',1P,E12.5)
101: c##>
102:      call CNTERR( 'FATAL' )
103:      else
104:        ANGLB(1) = 1.0D0
105:      end if
106: C
107:      if ( ABS(ANGLB(NANGLE+1)+1.0D0).gt.CDIFF ) then
108: c##<2007/03/14:PN3:
109: c##      write(IMG,*)
110: c## &      'XXX Last data of COSINE(...) is not cosine(PAI): ',
111: c## &      ANGLB(NANGLE+1)
112:      write(IMG,904) ANGLB(NANGLE+1)
113:      904 format(/' XXX(angset) Last data of COSINE(...) is not',
114:        & ' cosine(PAI): ',1P,E12.5)
115: c##>
116:      call CNTERR( 'FATAL' )
117:      else
118:        ANGLB(NANGLE+1) = -1.0D0
119:      end if
120: C
121:      call R8PRNT('COSINE',ANGLB,NANGLE+1,1,'ANGLE',' ',IPR)
122: C
123:      return
124:      end

```

src/shared/arayin.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ARAYIN( VNAME, A,      ICALL, TYP,  LX,    ICK,
3:     &                JDEBG, LAST,  NX,     NXNAME )
4: C
5:   real A(*)
6:   character VNAME*(*), TYP*2, NXNAME*(*)
7:   integer JDEBG(*)
8: C
9:   NOFFST = 0
10:  call ARYIN0( VNAME, A, ICALL, TYP, LX, ICK, JDEBG, LAST, NX,
11:    &          NOFFST, NXNAME )
12:  return
13: end
14: C
15: C-----
16: C
17:   subroutine ARYIN0( VNAME, A,      ICALL, TYP,  LX,    ICK,
18:     &                JDEBG, LAST,  NX,     NOFFST,NXNAME )
19: C
20: C   JAERI MONTE CARLO CODE UTILITY
21: C
22: C=====
23: C   PURPOSE: Input array values on "task shared" memory area
24: C             (common /ARRAY/)
25: C   CALLED IN: INTRO and many place (as ARAYIN or ARYIN0)
26: C   CALLS: I4READ,R4READ,R8READ
27: C
28: C=====
29: C   arguments  (i=input, o=output, w= work)
30: C
31: C i VNAME: variable name input
32: C o A: head position of task share dynamic array on which data
33: C   are input
34: C i TYP : data type string ( 'I4'/'R4'/'R8'/'I4D'/'R4D' )
35: C o LX: Location in array A(*)
36: C i o ICK: error counter (not cleared in this routine)
37: C i JDEBG(*) : debug flags
38: C o LAST: last available memory position as index for A(*)
39: C i o NX: Length of data to be input. If=0, length is determined
40: C   by the number of data read in.
41: C i NOFFST : expected data number is NX+ NOFFST when NXNAME != ' '
42: C   Value of NX is set to <number of data>-NOFFST.
43: C i NXNAME: VARIABLE NAME OF 'NX'. FOR MESSAGE.
44: C
45: C=====
46:   real A(*)
47:   CCCC include 'INC/_ARRAY'
48:   include 'INC/_IOUNIT'
49:   character VNAME*(*), TYP*2, NXNAME*(*)
50:   integer JDEBG(*)
51: C
52: C --- WORD LENGTH PARAMETER ( MWORD = 2 / 1 ... VP,SX/CRAY )
53: C   (WORD LENGTH OF 'REAL*8' OR 'CHARACTER*8' VARIABLES)
54: C   SEE subroutine WDBL
55: C
56:   include 'INC/_WORDL'
57: C
58: C-----
59: C
60: C ... SAVE ORIGINAL POINTER VALUE & LAST POSITION POINTER >>>
61: C
62:   LX00 = LX
63:   call GTLAST( LST00 )
64: C
65: C .... ICALL : FLAG INDICATING THAT THIS ROUTINE IS CALLED. ....

```

```

66: C
67:   ICALL = 1
68: C
69:   NB = NX + NOFFST
70:   NBB = -NB
71:   if ( NX.ne.0 ) then
72:     call KEEP( VNAME, LX, NB, TYP, LAST, JDBG )
73:   else
74:     call REMAIN( 'ARYIN0', NLTEMP, TYP, LAST )
75:     call KEEP( VNAME, LX, NLTEMP, TYP, LAST, JDEBG )
76:     NBB = -NLTEMP
77:   end if
78:   M = 1
79:   LVN = INDEX(VNAME,' ') - 1
80:   if ( LVN.lt.0 ) LVN = LEN(VNAME)
81: C
82:   if ( TYP.eq.'I4' ) then
83:     call I4READ( VNAME, A(LX), NA, NBB, IERR )
84:   else if ( TYP.eq.'R4' ) then
85:     call R4READ( VNAME, A(LX), NA, NBB, IERR )
86:   else if ( TYP.eq.'R8' ) then
87:     M = MWORD
88: C
89: C .... A(LX:) MUST BEGIN ON A DOUBLE-WORD BOUNDARY
90: C   FOR 4-BYTE MACHINES
91: C
92:   IF(MOD(LX,2).EQ.0) LX = LX + 1
93: C
94:   call R8READ( VNAME, A(LX), NA, NBB, IERR )
95:   else
96:     write(IPR,*) 'XXX(ARAYIN) Invalid data type <', TYP, '>'
97:     write(IPR,*) '      NXNAME : ', NXNAME, ' NX ', NX, ' NOFFST ',
98:       &          NOFFST
99:     stop 666
100:   end if
101: C
102: C ===== NA is number of data read in
103: C
104:   if ( NA.ne.0 ) then
105: C
106: C ..... Determine NX from NA, if the value of 'NX' is not defined.
107: C
108:   if ( NX.eq.0.and.NXNAME.ne.' ' ) then
109:     NX = NA - NOFFST
110:     write(IPR,7000) NXNAME, NX, VNAME(1:LVN)
111: 7000   format(1X,' *** The value of <',A6,> is set to ',I8,' from ',
112:     &      ' the length of input data <',A,> *** ')
113: C
114: C ..... Number of data NX or NA is invalid !!
115: C
116:   else if ( NX.ne.0.and.NX.ne.NA.and.NXNAME.ne.' ' ) then
117:     write(IMG,7020) VNAME, NA, NXNAME, NB
118: 7020   format(1X,'XXX(ARAYIN) The number of data given as <',A,
119:     &      '> (',I8,') is not equal to expected number (',A,
120:     &      ' ',I8,') ')
121:     call CNTERR( 'FATAL' )
122:     ICK = ICK + 1
123: C
124: C ..... Number of data is invalid !!
125: C
126:   else if ( NX.eq.0.and.NXNAME.eq.' ' ) then
127:     write(IMG,7040) VNAME(1:LVN), NA
128: 7040   format(1X,'XXX(ARAYIN) The expected number of data <',A,
129:     &      '> in input is zero (',I8,' data are input). '/'
130:     &      ' Probably some sizes have not been defined yet.')

```

src/shared/arayin.f

```
131:      call CNTERR( 'WARNING' )
132:      ICK      = ICK + 1
133: C
134: C ..... Number of data  NA < NX+NOFFST !!
135: C
136:      else if ( NA.lt.NB.and.NXNAME.eq.' ' ) then
137:      write(IMG,7060) VNAME(1:LVN), NA, NB
138: 7060      format(1X,'XXX(ARAYIN) The length of <','A,> ('',I8,
139:      &      ' ') in input is less than expected length ('',I8,')'.')
140:      call CNTERR( 'FATAL' )
141:      ICK      = ICK + 1
142:      end if
143: C
144: C ..... Adjust data size
145: C
146:      call RESIZE( VNAME, LX, NX+NOFFST, TYP, LAST )
147: C
148:      if ( LSIZ(LAST).gt.LIMGT() ) then
149:      write(IMG,*) 'XXX(ARAYIN) Memory over in array input ('',
150:      &      VNAME(1:LVN), ' )  MEMORY TAKEN = ', LSIZ(LAST),
151:      &      ' ', LIMIT = ' ', LIMGT()
152:      call PRSTOP( 1, 'MEMORY OVER.' )
153:      stop 999
154:      end if
155: C
156: C .... DEBUG PRINT .....
157: C
158:      if ( JDEBG(1).ne.0 ) then
159:      write(IPR,*) VNAME(1:LVN), ' : FROM ', LX, ' TO ', LAST - 1
160:      end if
161: C
162: C .... Empty input data ( empty parentheses ) .....
163: C
164:      else
165:      LX      = LX00
166:      call STLAST( 'ARAYIN', LST00, LAST )
167: C
168: C .... DEBUG PRINT .....
169: C
170:      if ( JDEBG(1).ne.0 ) then
171:      write(IPR,*) ' INPUT DATA < ', VNAME(1:LVN),
172:      &      '> IS EMPTY. SO IGNORED '
173:      end if
174:      end if
175: C
176:      return
177:      end
```


src/shared/aryii4.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ARYII4( VNAME, IA,   ICALL, ICK,   JDEBG, NI,   NJ,
3:     &               IS,   NX,   NXNAME )
4: C
5: C   JAERI MONTE CARLO CODE UTILITY
6: C
7: C=====
8: C   (obsolete. called from nowhere)
9: C
10: C PURPOSE: INPUT INTEGER VALUES AND PUT THEM IN SPECIFIED PLACES IN
11: C           AN ARRAY.
12: C
13: C CALLED IN: INTRO
14: C CALLS: I4READ,R4READ,R8READ
15: C=====
16: C arguments (i=input, o=output, w=work)
17: C
18: C i VNAME      : name of input data
19: C o IA(NI,NJ) :   AN INTGER ARRAY.
20: C io ICALL     : set to 1 when called
21: C io ICK       : add 1 when error occurred
22: C i JDEBG      : flag for debug printout
23: C i IS, NX     : STORE INPUT DATA FROM IA(IS,J) TO IA(IS+NX-1,J)
24: C               FOR EACH SECOND INDEX J.
25: C i NXNAME     : name of data size variable if any
26: C=====
27:
28:   write(*,*) 'XXX(ARYII4) This routine is obsolete! do not call.'
29:   stop 666
30:
31: *   include 'INC/_ARRAY'
32: *   include 'INC/_IOUNIT'
33:
34: C/#IF DYNAMIC
35: C/# IF NOPOINTER
36: C*   integer IMEM(1)
37: C*   equivalence (IMEM(1),A(1))
38: C/# ELSE
39: C*   integer IMEM
40: C*   pointer(LPI4,IMEM(1))
41: C/# ENDIF
42: C/#ELSE
43: C*   integer IMEM(1)
44: C*   equivalence (IMEM(1),A(1))
45: C/#ENDIF
46: C
47: *   character VNAME*(*), NXNAME*(*)
48: C
49: *   integer JDEBG(*)
50: C
51: *   include 'INC/_WORDL'
52: C
53: *   integer IA(NI,NJ)
54: C
55: *   ICALL   = 1
56: C
57: C/#IF DYNAMIC
58: C*   LPI4    = LPA
59: C/#ENDIF
60: *   LVN      = INDEX(VNAME,' ') - 1
61: *   if ( LVN.lt.0 ) LVN = LEN(VNAME)
62: C-----
63: C   CHECK NUMBER OF INPUT VALUES
64: C-----
65: *   NB       = NX*NJ

```

```

66: *   if ( NB.eq.0 ) then
67: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
68: *     write(IPR,*) 'XXX LACKS SIZE-INFORMATION TO INPUT VALUES!!',
69: *     &           ' (AT ARYII4/R4/R8 ) '
70: *     write(IPR,*) 'XXX VARIABLE: <', VNAME(1:LVN), '> '
71: *     write(IPR,*) 'XXX TRIED TO INPUT VALUES ON ', VNAME(1:LVN),
72: *     &           '(', NI, ',', NJ, ') FROM I=', IS, ' TO ', IS + NX - 1
73: *     write(IPR,*) ' PLEASE SPECIFY SIZE PARAMETERS AND TRY AGAIN!'
74: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
75: *     call PRSTOP( 1, 'MAYBE SOME PROBLEMS IN ORDER OF INPUT DATA.' )
76: *     stop 888
77: *   end if
78: C-----
79: C   FOR ONE DIMENSIONAL ARRAY (NJ = 1)
80: C-----
81:
82: *   if ( NJ.le.1 ) then
83: *     call I4READ( VNAME(1:LVN), IA(IS,1), NA, NB, IERR )
84: C-----
85: C   FOR TWO DIMENSIONAL ARRAY (NJ > 1)
86: C-----
87:
88: *   else
89: *     call GTLAST( LAST )
90: *     LAST0 = LAST
91: *     call KEEP( 'temp', LX, NB, 'I4', LAST, JDEBG )
92: C
93: *     if ( LSIZ(LAST).gt.LIMIT ) then
94: *       call MEMERR( 'ARYII4/R4/R8',
95: *       &           'TRY TO KEEP A TEMPORARY AREA FOR INPUT.',
96: *       &           LSIZ(LAST), LIMIT )
97: *       call PRSTOP( 0, 'MEMORY OVER.' )
98: *       stop 999
99: *     end if
100: C
101: *     call I4READ( VNAME(1:LVN), A(LX), NA, NB, IERR )
102: *   end if
103: C
104: C ..... DATA NUMBER 'NA' < 'NB' !!
105: C
106: *   if ( NA.lt.NB ) then
107: *     write(IPR,9000) VNAME(1:LVN), NA, NB
108: *9000   format(/IX,'!!! NUMBER OF DATA FOR <',A,'> (',I8,') IS ',
109: *     &           'LESS THAN REQUIRED LENGTH (',I8,').')
110: *     ICK = ICK + 1
111: *   end if
112:
113: C-----
114: C   DATA TRANSFER TO ARRAY FOR TWO DIMENSIONAL CASE
115: C-----
116:
117: *   if ( NJ.gt.1 ) then
118: *     NNI = IS
119: *     NNJ = 1
120: *     do 100 I = LX, LAST - 1
121: *       IA(NNI,NNJ) = IMEM(I)
122: *       A(I) = 0.0
123: *       NNI = NNI + 1
124: *       if ( NNI.eq.IS+NX ) then
125: *         NNI = IS
126: *         NNJ = NNJ + 1
127: *       end if
128: * 100   continue
129: C
130: C ... free memory of temporary working area ...

```

src/shared/aryii4.f

```
131: C
132: *      call STLAST( 'ARYII4', LAST0, LAST )
133: *      end if
134: C
135: C .... DEBUG PRINT .....
136: C
137: *      return
138:      end
```

SAFE

src/shared/aryinv.f

```
1:      subroutine ARYINV(A,B,N)
2: C=====
3: C  PURPOSE: store 1/A(i) on 1/B(i)
4: C  CALLED IN: evrywhere
5: C=====
6:      real A(N), B(N)
7:      do 100 I = 1, N
8:          if( A(I).ne.0 ) then
9:              B(I) = 1.0/A(I)
10:         else
11:             B(I) = 0.0
12:         end if
13: 100 continue
14:      return
15:      end
```

src/shared/aryir4.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ARYIR4( VNAME, RA,      ICALL, ICK,      JDEBG, NI,      NJ,
3:     &               IS,      NX,      NXNAME )
4: C
5: C   JAERI MONTE CARLO CODE UTILITY
6: C
7: C-----
8: C   (obsolete. called from nowhere)
9: C
10: C  PURPOSE: INPUT REAL VALUES AND PUT THEM IN SPECIFIED PLACES IN
11: C           AN ARRAY.
12: C  CALLED IN: INTRO
13: C  CALLS: I4READ,R4READ,R8READ
14: C-----
15: C  arguments (i=input, o=output, w=work )
16: C
17: C i VNAME      : name of input data
18: C o RA(NI,NJ) : A REAL ARRAY.
19: C io ICALL     : set to 1 when called
20: C io ICK       : add 1 when error occurred
21: C i JDEBG      : flag for debug printout
22: C i IS, NX     : STORE INPUT DATA FROM RA(IS,J) TO RA(IS+NX-1,J)
23: C               FOR EACH SECOND INDEX J.
24: C i NXNAME     : name of data size variable if any
25: C-----
26:
27:   write(*,*) 'XXX(ARYIR4) This routine is obsolete! do not call.'
28:   stop 666
29:
30: *   include 'INC/_ARRAY'
31: *   include 'INC/_IOUNIT'
32: C
33: C
34: *   character VNAME*(*), NXNAME*(*)
35: *   integer JDEBG(*)
36: C
37: *   include 'INC/_WORDL'
38: C
39: *   real RA(NI,NJ)
40: C
41: *   ICALL = 1
42: C
43: *   LVN = INDEX(VNAME,' ') - 1
44: *   if ( LVN.lt.0 ) LVN = LEN(VNAME)
45: C-----
46: C   CHECK NUMBER OF INPUT VALUES
47: C-----
48: *   NB = NX*NJ
49: *   if ( NB.eq.0 ) then
50: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
51: *     write(IPR,*) 'XXX LACKS SIZE-INFORMATION TO INPUT VALUES!!',
52: *     &           ' (AT ARYII4/R4/R8 ) '
53: *     write(IPR,*) 'XXX VARIABLE: <', VNAME(1:LVN), '> '
54: *     write(IPR,*) 'XXX TRIED TO INPUT VALUES ON ', VNAME(1:LVN),
55: *     &           '(', NI, ',', NJ, ') FROM I=', IS, ' TO ', IS + NX - 1
56: *     write(IPR,*) ' PLEASE SPECIFY SIZE PARAMETERS AND TRY AGAIN!'
57: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
58: *     call PRSTOP( 1, 'MAYBE SOME PROBLEMS IN ORDER OF INPUT DATA.' )
59: *     stop 888
60: *   end if
61: C-----
62: C   FOR ONE DIMENSIONAL ARRAY (NJ = 1)
63: C-----
64:
65: *   if ( NJ.le.1 ) then

```

```

66: *       call R4READ( VNAME(1:LVN), RA(IS,1), NA, NB, IERR )
67: C-----
68: C   FOR TWO DIMENSIONAL ARRAY (NJ > 1)
69: C-----
70: *   else
71: *     call GTLAST( LAST )
72: *     LAST0 = LAST
73: *     call KEEP( 'temp', LX, NB, 'R4', LAST, JDEBG )
74: C
75: *     if ( LSIZ(LAST).gt.LIMIT ) then
76: *       call MEMERR( 'ARYII4/R4/R8',
77: *     &           'TRY TO KEEP A TEMPORARY AREA FOR INPUT.',
78: *     &           LSIZ(LAST), LIMIT )
79: *       call PRSTOP( 0, 'MEMORY OVER.' )
80: *       stop 999
81: *     end if
82: C
83: *     call R4READ( VNAME(1:LVN), A(LX), NA, NB, IERR )
84: *   end if
85: C
86: C ..... DATA NUMBER 'NA' < 'NB' !!
87: C
88: *   if ( NA.lt.NB ) then
89: *     write(IPR,9000) VNAME(1:LVN), NA, NB
90: *9000   format(/IX,'!!! NUMBER OF DATA FOR <',A,'> (',I8,') IS ',
91: *     &           'LESS THAN REQUIRED LENGTH (',I8,')'./)
92: *     ICK = ICK + 1
93: *   end if
94: C-----
95: C   DATA TRANSFER TO ARRAY FOR TWO DIMENSIONAL CASE
96: C-----
97: C
98: C
99: *   if ( NJ.gt.1 ) then
100: *     NNI = IS
101: *     NNJ = 1
102: *     do 100 I = LX, LAST - 1
103: *       RA(NNI,NNJ) = A(I)
104: *       A(I) = 0.0
105: *       NNI = NNI + 1
106: *       if ( NNI.eq.IS+NX ) then
107: *         NNI = IS
108: *         NNJ = NNJ + 1
109: *       end if
110: *     100 continue
111: C
112: C ... free memory of temporary working area ...
113: C
114: *     call STLAST( 'ARYIR4', LAST0, LAST )
115: *   end if
116: C
117: C .... DEBUG PRINT .....
118: C
119: C IF(JDEBG.NE.0) THEN
120: C   WRITE(IPR,*) VNAME,' : FROM ',LX,' TO ',LSTART-1
121: C ENDIF
122: C
123: *   return
124: * end

```

src/shared/aryir8.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ARYIR8( VNAME, R8A,   ICALL, ICK,   JDEBG, NI,   NJ,
3:     &               IS,   NX,   NXNAME )
4: C
5: C   JAERI MONTE CARLO CODE UTILITY
6: C
7: C=====
8: C   (obsolete. called from nowhere)
9: C
10: C PURPOSE: INPUT REAL*8 VALUES AND PUT THEM IN SPECIFIED PLACES IN
11: C           AN ARRAY.
12: C CALLED IN: INTRO
13: C CALLS: IREAD,R4READ,R8READ
14: C=====
15: C arguments (i=input, o=output, w=work)
16: C
17: C i VNAME      : name of input data
18: C o R8A(NI,NJ) : A REAL*8 ARRAY.
19: C io ICALL     : set to 1 when called
20: C io ICK       : add 1 when error occurred
21: C i JDEBG      : flag for debug printout
22: C i IS, NX     : STORE INPUT DATA FROM R8A(IS,J) TO R8A(IS+NX-1,J)
23: C               FOR EACH SECOND INDEX J.
24: C i NXNAME     : name of data size variable if any
25: C=====
26: C
27: C   write(*,*) 'XXX(ARYIR8) This routine is obsolete! do not call.'
28: C   stop 666
29: C
30: *   include 'INC/_ARRAY'
31: *   include 'INC/_IOUNIT'
32: C
33: C/#IF DYNAMIC
34: C/# IF NOPOINTER
35: C*   real*8   R8MEM(1)
36: C*   equivalence (R8MEM(1),A(1))
37: C/# ELSE
38: C*   real*8 R8MEM
39: C*   pointer(LPA8,R8MEM(1))
40: C/# ENDIF
41: C/#ELSE
42: C*   real*8   R8MEM(1)
43: C*   equivalence (R8MEM(1),A(1))
44: C/#ENDIF
45: C
46: *   character VNAME*(*), NXNAME*(*), TYPE*2
47: *   integer JDEBG(*)
48: C
49: *   include 'INC/_WORDL'
50: C
51: *   real*8 R8A(NI,NJ)
52: C
53: *   ICALL   = 1
54: C
55: C .... CALL as 'R4' when 64 bit/word ...
56: C
57: *   if ( MWORD.eq.1 ) then
58: *     call ARYIR4( VNAME, R8A, ICALL, ICK, JDEBG, NI, NJ, IS, NX,
59: *       &         NXNAME )
60: *     return
61: *   end if
62: C
63: C/#IF DYNAMIC
64: C*   LPA8   = LPA
65: C/#ENDIF

```

```

66: *   LVN     = INDEX(VNAME,' ') - 1
67: *   if ( LVN.lt.0 ) LVN = LEN(VNAME)
68: C-----
69: C   CHECK NUMBER OF INPUT VALUES
70: C-----
71: *   NB      = NX*NJ
72: *   if ( NB.eq.0 ) then
73: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
74: *     write(IPR,*) 'XXX LACKS SIZE-INFORMATION TO INPUT VALUES!!',
75: *       &         ' (AT ARYII4/R4/R8 ) '
76: *     write(IPR,*) 'XXX VARIABLE: <', VNAME(1:LVN), '> '
77: *     write(IPR,*) 'XXX TRIED TO INPUT VALUES ON ', VNAME(1:LVN),
78: *       &         '(', NI, ',', NJ, ') FROM I=', IS, ' TO ', IS + NX - 1
79: *     write(IPR,*) ' PLEASE SPECIFY SIZE PARAMETERS AND TRY AGAIN!'
80: *     write(IPR,*) 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
81: *     call PRSTOP( 1, 'MAYBE SOME PROBLEMS IN ORDER OF INPUT DATA.' )
82: *     stop 888
83: *   end if
84: C-----
85: C   FOR ONE DIMENSIONAL ARRAY (NJ = 1)
86: C-----
87: C
88: *   if ( NJ.le.1 ) then
89: *     call R8READ( VNAME(1:LVN), R8A(IS,1), NA, NB, IERR )
90: C-----
91: C
92: C   FOR TWO DIMENSIONAL ARRAY (NJ > 1)
93: C-----
94: C
95: *   else
96: *     call GTLAST( LAST )
97: *     LAST0 = LAST
98: *     call KEEP( 'temp', LX, NB, 'R8', LAST, JDEBG )
99: C
100: *   if ( LSIZ(LAST).gt.LIMIT ) then
101: *     call MEMERR( 'ARYII4/R4/R8',
102: *       &         'TRY TO KEEP A TEMPORARY AREA FOR INPUT.',
103: *       &         LSIZ(LAST), LIMIT )
104: *     call PRSTOP( 0, 'MEMORY OVER.' )
105: *     stop 999
106: *   end if
107: C
108: *   call R8READ( VNAME(1:LVN), A(LX), NA, NB, IERR )
109: *   end if
110: C
111: C ..... DATA NUMBER 'NA' < 'NB' !!
112: C
113: *   if ( NA.lt.NB ) then
114: *     write(IPR,9000) VNAME(1:LVN), NA, NB
115: *9000   format(1X,' !!! NUMBER OF DATA FOR <',A,'> (',I8,') IS ',
116: *     &         'LESS THAN REQUIRED LENGTH (',I8,').')
117: *     ICK   = ICK + 1
118: *   end if
119: C
120: C-----
121: C   DATA TRANSFER TO ARRAY FOR TWO DIMENSIONAL CASE
122: C-----
123: C
124: *   if ( NJ.gt.1 ) then
125: *     NNI   = IS
126: *     NNJ   = 1
127: *     do 100 I = LX, LAST - 1, 2
128: *       II   = (I+1) /2
129: *       R8A(NNI,NNJ) = R8MEM(II)
130: *       R8MEM(II)   = 0.0

```

src/shared/aryir8.f

```
131: *          NNI      = NNI + 1
132: *          if ( NNI.eq.IS+NX ) then
133: *              NNI      = IS
134: *              NNJ      = NNJ + 1
135: *          end if
136: * 100      continue
137: C
138: C ... free memory of temporary working area ...
139: C
140: *          call STLAST( 'ARYIR8', LAST0, LAST )
141: *      end if
142: C
143: C ... DEBUG PRINT .....
144: C
145: C IF(JDEEG(1).NE.0) THEN
146: C     WRITE(IPR,*) VNAME,' : FROM ',LX,' TO ',LSTART-1
147: C ENDIF
148: C
149: *      return
150:      end
```

src/shared/arysum.f

```
1:      subroutine ARYSUM( IA,      N,      NSUM )
2: C=====
3: C  PURPOSE: TO TAKE SUM OF ARRAY IA.
4: C  CALLED IN: INTRO
5: C=====
6:      integer IA(N)
7:      NSUM      = 0
8:      do 100 I = 1, N
9:          NSUM      = NSUM + IA(I)
10: 100 continue
11:      return
12:      end
```

WAVE

src/shared/ascti4.f

```

1:      subroutine ASCTI4( VNAME, IA,      IB,      NI,      NJ,      MI,      IS,
2:      &                  NX )
3:      C
4:      C  JAERI MONTE CARLO CODE UTILITY
5:      C
6:      C=====
7:      C  PURPOSE: scatter data in array B(MI,NJ) to array A(NI,NJ)
8:      C
9:      C      ascti4 (FOR INTEGER), asctr4 ( FOR SINGLE PRECISION REAL)
10:     C      asctr8 (FOR DOUBLE PRECISION DATA)
11:     C
12:     C  CALLED IN: everywhere
13:     C  CALLS: none
14:     C-----
15:     C  arguments (i=input, o=output, w=work)
16:     C
17:     C  o IA(NI,NJ) : an integer array
18:     C  i IB(MI,NJ) : an integer array
19:     C
20:     C  o RA(NI,NJ) : a real array
21:     C  i RB(MI,NJ) : a real array
22:     C
23:     C  o R8A(NI,NJ) : a double precision array
24:     C  i R8B(MI,NJ): a double precision array
25:     C      (EQUIVALENT TO RA ON 64BIT MACHINE)
26:     C
27:     CM IS, NX      :      STORE INPUT DATA FROM B(IS,J) TO A(IS+NX-1,J)
28:     CM              FOR EACH SECOND INDEX J.
29:     C  i IS, NX      :      STORE INPUT DATA FROM B(I ,J) TO A(IS+I-1,J),I=1,NX
30:     C              FOR EACH SECOND INDEX J.
31:     C=====
32:     C
33:     C      character VNAME*(*)
34:     C
35:     C
36:     C      integer IA(NI,NJ), IB(MI,NJ)
37:     C
38:     C      do 100 i=is,is+nx-1
39:     C          ia(i,j) = ib(i,j)
40:     C
41:     C      do 110 J = 1, NJ
42:     C          do 100 I = 1, NX
43:     C              IA(IS+I-1,J)  = IB(I,J)
44:     C          100 continue
45:     C      110 continue
46:     C      return
47:     C      end
48:     C
49:     C=====
50:     C
51:     C      subroutine ASCTR4( VNAME, RA,      RB,      NI,      NJ,      MI,      IS,
52:     C      &                  NX )
53:     C
54:     C      character VNAME*(*)
55:     C
56:     C      real RA(NI,NJ), RB(MI,NJ)
57:     C
58:     C      do 110 J = 1, NJ
59:     C          do 100 I = 1, NX
60:     C              RA(IS+I-1,J)  = RB(I,J)
61:     C          100 continue
62:     C      110 continue
63:     C      return
64:     C      end
65:     C

```

```

66: C=====
67: C
68:     C      subroutine ASCTR8( VNAME, R8A,      R8B,      NI,      NJ,      MI,      IS,
69:     C      &                  NX )
70:     C
71:     C      character VNAME*(*)
72:     C
73:     C      real*8 R8A(NI,NJ), R8B(MI,NJ)
74:     C
75:     C      do 110 J = 1, NJ
76:     C          do 100 I = 1, NX
77:     C              R8A(IS+I-1,J)  = R8B(I,J)
78:     C          100 continue
79:     C      110 continue
80:     C      return
81:     C      end

```


src/shared/atrans.f

```
1:      subroutine ATRANS( A,      B,      N )
2: C
3: C      MVP/GMVP UTILITY
4: C
5: C=====
6: C  PURPOSE: transfer data of single-word type from array 'A' to 'B'.
7: C          A & B should not overlap in memory.
8: C=====
9: Cccc  REAL      A(N),B(N)
10:      integer A(N), B(N)
11:      do 100 I = 1, N
12:          B(I) = A(I)
13:      100 continue
14:      return
15:      end
```

src/shared/atrn2.f

```
1:      subroutine ATRNS2( IA,      LST,      N,      LDST )
2: C=====
3: C purpose: copy 'N' array elements staring at 'IA(LST)' to elements
4: C           starting from 'IA(LDST)'
5: C=====
6:      real IA(*)
7: C
8: C .... no overlap ...
9: C
10:     if ( LST+N-1.lt.LDST .or. LDST+N-1.lt.LST ) then
11: *VOCL LOOP,NOVREC
12:       do 100 I = 0, N - 1
13:         IA(LDST+I) = IA(LST+I)
14:       100 continue
15: C
16: C ... overlap ...
17: C
18:       else if ( LST.gt.LDST ) then
19: C === unvectorizable !! ===
20:       do 110 I = 0, N - 1
21:         IA(LDST+I) = IA(LST+I)
22:       110 continue
23:       else
24:       do 120 I = N - 1, 0, -1
25:         IA(LDST+I) = IA(LST+I)
26:       120 continue
27:       end if
28: C
29:       return
30:     end
```

src/shared/bmcmk0.f

```

1:      subroutine BMCMK0( F,      X,      Q,      ANS,      N1 )
2: C=====
3: C  PURPOSE: PREPARATION FOR DISCRETE SAMPLING
4: C      Make translated tables to sample an real value directly
5: C      by discrete sampling
6: C  CALLED IN: JNPUT
7: C-----
8: C  arguments (i=input,o=output,w=work)
9: C
10: C io F : Probability density.
11: C      Original values are changed to meaningless ones.
12: C i X : values selected with probability density table F.
13: C o Q : branch probability to ANS(1,i)
14: C o ANS(2,N1) : values selected
15: C i N1 : length of table
16: C=====
17:      real F(N1), X(N1), Q(N1), ANS(2,N1)
18: C
19:      S      = 0.0
20:      do 100 K = 1, N1
21:          S      = S + F(K)
22:      100 continue
23:      do 110 K = 1, N1
24:          F(K)    = F(K) /S
25:      110 continue
26:      FN1      = FLOAT(N1)
27:      do 160 K = 1, N1
28:          FMAX    = -0.5
29:          FMIN    = 2.0
30:          I      = 1
31:          J      = 1
32:
33: C/#IF  SYSTEM( SX* )
34:
35: *      do 120 K1 = 1, N1
36: *          if ( F(K1).gt.FMAX ) then
37: *              I      = K1
38: *              FMAX    = F(K1)
39: *          end if
40: *          if ( F(K1).ge.0. ) FMIN = MIN(FMIN,F(K1))
41: *      120 continue
42: *      do 130 K1 = 1, N1
43: *          if ( F(K1).eq.FMIN ) go to 140
44: *      130 continue
45: *      140      J      = K1
46:
47: C/#ELSE
48:
49:      do 150 K1 = 1, N1
50:          if ( F(K1).gt.FMAX ) then
51:              I      = K1
52:              FMAX    = F(K1)
53:          end if
54:          if ( F(K1).ge.0. ) then
55:              if ( F(K1).lt.FMIN ) then
56:                  J      = K1
57:                  FMIN    = F(K1)
58:              end if
59:          end if
60:      150 continue
61:
62: C/#ENDIF
63:      Q(J)      = 1.0 - FN1*F(J)
64:      F(J)      = -1.0
65:      F(I)      = F(I) - Q(J) /FN1

```

```

66:      ANS(1,J)  = X(I)
67:      ANS(2,J)  = X(J)
68:      160 continue
69: C
70:      do 170 K = 1, N1
71:          if ( Q(K).lt.0.0 ) Q(K) = 0.0
72:      170 continue
73: C
74:      return
75:      end

```

src/shared/bmcmk1.f

```
1:      subroutine BMCMK1( F,      Q,      IANS, N1 )
2: C=====
3: C  PURPOSE: PREPARATION FOR DISCRETE SAMPLING
4: C      Integers from 1 to N1 is selected with probability density F.
5: C  CALLED IN: many place
6: C-----
7: C arguments ( i=input, o=output, w=work)
8: C
9: C io F : Probability density.
10: C      Original values are changed to meaningless ones.
11: C o Q : branch probability to ANS(1,i)
12: C o ANS(2,N1) : integer values selected
13: C i N1 : length of table
14: C=====
15:      real F(N1), Q(N1)
16:      integer IANS(2,N1)
17: C
18:      S      = 0.0
19:      do 100 K = 1, N1
20:          S      = S + F(K)
21:      100 continue
22:      do 110 K = 1, N1
23:          F(K)    = F(K) /S
24:      110 continue
25:      FN1      = FLOAT(N1)
26:      do 160 K = 1, N1
27:          FMAX    = -0.5
28:          FMIN    = 2.0
29:          I      = 1
30:          J      = 1
31: C/#IF SYSTEM( SX* )
32: *      do 120 K1 = 1, N1
33: *          if ( F(K1).gt.FMAX ) then
34: *              I      = K1
35: *              FMAX    = F(K1)
36: *          end if
37: *          if ( F(K1).ge.0. ) FMIN = MIN(FMIN,F(K1))
38: * 120 continue
39: *      do 130 K1 = 1, N1
40: *          if ( FMIN.eq.F(K1) ) go to 140
41: * 130 continue
42: * 140      J      = K1
43: C/#ELSE
44:      do 150 K1 = 1, N1
45:          if ( F(K1).gt.FMAX ) then
46:              I      = K1
47:              FMAX    = F(K1)
48:          end if
49:          if ( F(K1).ge.0. ) then
50:              if ( F(K1).lt.FMIN ) then
51:                  J      = K1
52:                  FMIN    = F(K1)
53:              end if
54:          end if
55:      150 continue
56: C/#ENDIF
57:      Q(J)      = 1.0 - FN1*F(J)
58:      F(J)      = -1.0
59:      F(I)      = F(I) - Q(J) /FN1
60:      IANS(1,J) = I
61:      IANS(2,J) = J
62:      160 continue
63: C
64:      do 170 K = 1, N1
65:          if ( Q(K).lt.0.0 ) Q(K) = 0.0
66:      170 continue
67: C
68:      return
69:      end
```

src/shared/bmcmk.f

```
1:      subroutine BMCMK( F,      X,      Q,      ANS,      N1 )
2: C
3: C      MVP/GMVP UTILITY
4: C
5: C=====
6: C PURPOSE: PREPROCESSING FOR B.M.C. SAMPLING.
7: C      (NEVER CALLED)
8: C=====
9:      real F(N1), X(N1), Q(N1), ANS(2,N1)
10:      S      = 0.0
11:      do 100 K = 1, N1
12:          S      = S + F(K)
13: 100 continue
14:      do 110 K = 1, N1
15:          F(K)    = F(K) /S
16:          ANS(2,K) = -1.0
17: 110 continue
18:      FN1      = FLOAT(N1)
19:      do 130 K = 1, N1
20:          FMAX    = -0.5
21:          FMIN     = 2.0
22:          I      = 1
23:          J      = 1
24:          do 120 K1 = 1, N1
25:              if ( ANS(2,K1).eq.-1.0 ) then
26:                  if ( F(K1).gt.FMAX ) then
27:                      I      = K1
28:                      FMAX    = F(K1)
29:                  end if
30:                  if ( F(K1).lt.FMIN ) then
31:                      J      = K1
32:                      FMIN    = F(K1)
33:                  end if
34:              end if
35: 120 continue
36:          Q(J)      = 1.0 - FN1*F(J)
37: C          F(J)      = -1.0
38:          F(I)      = F(I) - Q(J) /FN1
39:          ANS(1,J)   = X(I)
40:          ANS(2,J)   = X(J)
41: 130 continue
42: C
43:      do 140 K = 1, N1
44:          if ( Q(K).lt.0.0 ) Q(K) = 0.0
45: 140 continue
46: C
47:      return
48:      end
```

src/shared/bodarb.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODARB( ICALL, SDA, LSDA, BD, IBODN, IBSDA, NS,
3:     & LSDAT, DINF, GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'ARB' (ARBITRARY POLYHEDRON)
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTRERR
12: C
13: C-----
14: C arguments : see description in BODBOX routine.
15: C=====
16:   implicit real*8(D)
17:   real*8 SDA(*), BD(*), DINF
18:   real*8 GRANGE(6)
19:   include 'INC/_WORDL'
20:   include 'INC/_IUNIT'
21: C
22: C ... local data
23: C
24:   integer IFLAG(8), IC(4)
25: C-----
26:   ICODE = 0
27: C ..... ICALL: NUMBER OF BODY DATA + 1
28:   ICALL = 31
29:   call R8READ( ' ', BD, NA, -ICALL, IERR )
30: C
31:   if ( NA.ne.ICALL ) then
32:     write(IMG,*) 'XXX Number of data for body "ARB" ', IBODN,
33:     & ' is not ',ICALL
34:     call CNTRERR( 'FATAL' )
35:   end if
36: C
37:   IBODN = INT(BD(1))
38:   IBSDA = (LSDAT-LSDA) /MWORD + 1
39: C
40: C-----
41: C BD2,BD3,BD4 ... BD23,BD24,BD25 : Vertices
42: C BD26,BD27,BD28,BD29,BD30,BD31 : Surface descriptor
43: C-----
44: C
45: C A surface descriptor is an integer whose digits indicate
46: C vertices composing a surface.
47: C
48: C ex.
49: C   123 --> surface composed by vertices 1, 2 and 3
50: C   3541 --> surface composed by vertices 3, 5, 4 and 1
51: C
52: C
53: C ..... NS : NUMBER OF SURFACE .....
54: C
55:   do 100 I = 1, 8
56:     IFLAG(I) = 0
57: 100 continue
58: C
59:   NERR = 0
60: C
61:   NS = 0
62:   do 140 K = 1, 6
63: C
64: C ..... GET VERTEX NUMBERS AND COUNT THE NUMBER OF VERTICES TO BE USED.
65:   L = 0

```

```

66:   M = INT(BD(25+K))
67:   do 130 I = 1, 4
68:     LL = MOD(M,10)
69:     if ( LL.ne.0 ) then
70:       do 110 II = 1, L
71:         if ( LL.eq.IC(II) ) go to 120
72: 110       continue
73:       L = L + 1
74:       IC(L) = LL
75:       IFLAG(LL) = 1
76:     end if
77: 120     M = M/10
78: 130   continue
79: C
80: C ..... IF THE NUMBER OF VERTEXES IS LESS THAN THREE, THE SURFACE
81: C DESCRIPTOR SHOULD BE IGNORED.
82: C
83:   if ( L.lt.3 ) then
84: C
85: C ... (fixed Jan 2000) ... call CNTRERR and NERR++ should be
86: C in if block !!!
87:   if ( L.gt.1 ) then
88:     write(IMG, '(lx,a,i2,a,i6,a)')
89:     & ' XXX Surface descriptor # ', K, ' of body "ARB" ',
90:     & IBODN, ' is invalid !!!'
91:     call CNTRERR( 'FATAL' )
92:     NERR = NERR + 1
93:   end if
94:   go to 140
95: end if
96: C
97: C ..... CALCULATE A NORMAL VECTOR
98: C
99:   NS = NS + 1
100:   M1 = (IC(1)-1)*3 + 2
101:   M2 = (IC(2)-1)*3 + 2
102:   M3 = (IC(3)-1)*3 + 2
103:   DX1 = BD(M1) - BD(M2)
104:   DY1 = BD(M1+1) - BD(M2+1)
105:   DZ1 = BD(M1+2) - BD(M2+2)
106:   DX2 = BD(M3) - BD(M2)
107:   DY2 = BD(M3+1) - BD(M2+1)
108:   DZ2 = BD(M3+2) - BD(M2+2)
109:   D1 = DY1*DZ2 - DZ1*DY2
110:   D2 = DZ1*DX2 - DX1*DZ2
111:   D3 = DX1*DY2 - DY1*DX2
112:   DD = SQRT(D1**2+D2**2+D3**2)
113:   LL = (NS-1)*5
114:   if ( DD.eq.0.0D0 ) then
115:     write(IMG, '(lx,a,i2,a,i6,a)') ' XXX CANNOT COMPOSE ', K,
116:     & ' ' 'TH NORMAL VECTOR OF BODY "ARB" ', IBODN, ' !!!'
117:     call CNTRERR( 'FATAL' )
118:     NS = NS - 1
119:     NERR = NERR + 1
120:   else
121:     SDA(LL+1) = 5.0 + 10000*5
122:     SDA(LL+2) = D1/DD
123:     SDA(LL+3) = D2/DD
124:     SDA(LL+4) = D3/DD
125:     SDA(LL+5) = SDA(LL+2)*BD(M2) + SDA(LL+3)*BD(M2+1)
126:     & + SDA(LL+4)*BD(M2+2)
127:     if ( L.eq.4 ) then
128:       M4 = (IC(4)-1)*3 + 2
129:       DD = SDA(LL+2)*BD(M4) + SDA(LL+3)*BD(M4+1)
130:       & + SDA(LL+4)*BD(M4+2)

```

src/shared/bodarb.f

```
131:         if ( ABS(DD-SDA(LL+5)).gt.1.0D-6 ) then
132:             write(IMG,'(lx,a,i2,a,i6,a,3i3,a)') ' XXX VERTEX ',
133:             &             IC(4), ' OF "ARB" ', IBODN,
134:             &             ' IS NOT ON THE PLANE MADE BY VERTICES ',
135:             &             (IC(I),I=1,3), '.'
136:             write(IMG,'(lx,a,i4,a)') ' SURFACE ',
137:             &             INT(BD(25+K)), ' IS IGNORED !!!'
138:             call CNTERR( 'FATAL' )
139:             NERR = NERR + 1
140:             NS = NS - 1
141:         end if
142:     end if
143: end if
144: 140 continue
145: C
146: C ..... GRAVITY CENTER
147:     L = 0
148:     D1 = 0
149:     D2 = 0
150:     D3 = 0
151:     do 150 I = 1, 8
152:         if ( IFLAG(I).eq.1 ) then
153:             L = L + 1
154:             D1 = D1 + BD(3*I-1)
155:             D2 = D2 + BD(3*I)
156:             D3 = D3 + BD(3*I+1)
157:         end if
158:     150 continue
159:     D1 = D1/L
160:     D2 = D2/L
161:     D3 = D3/L
162: C
163: C ..... ADJUST DIRECTIONS OF THE NORMAL VECTORS SO THAT THE GRAVITY
164: C CENTER LIES IN THE INSIDE OF THE HALF SPACES.
165:     do 160 K = 1, NS
166:         M1 = 5*(K-1) + 2
167:         DD = SDA(M1)*D1 + SDA(M1+1)*D2 + SDA(M1+2)*D3 - SDA(M1+3)
168:         if ( DD.gt.0.0 ) then
169:             SDA(M1) = -SDA(M1)
170:             SDA(M1+1) = -SDA(M1+1)
171:             SDA(M1+2) = -SDA(M1+2)
172:             SDA(M1+3) = -SDA(M1+3)
173:         end if
174:     160 continue
175: C
176:     do 170 I = 1, 8
177:         if ( IFLAG(I).eq.1 ) then
178:             D1 = BD(3*I-1)
179:             D2 = BD(3*I)
180:             D3 = BD(3*I+1)
181:             GRANGE(1) = MIN( GRANGE(1), D1 )
182:             GRANGE(2) = MAX( GRANGE(2), D1 )
183:             GRANGE(3) = MIN( GRANGE(3), D2 )
184:             GRANGE(4) = MAX( GRANGE(4), D2 )
185:             GRANGE(5) = MIN( GRANGE(5), D3 )
186:             GRANGE(6) = MAX( GRANGE(6), D3 )
187:         end if
188:     170 continue
189: C
190:     LSDAT = LSDAT + 5*NS*MWORD
191:     if ( NERR.gt.0 ) ICODE = 1
192:     return
193: end
```

src/shared/bodbbc.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine BODBBC( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &               LSDAT, DINF,   GRANGE, ICODE )
4: C=====
5: C   Purpose: Input body 'BBC' (Body by Body Combination)
6: C
7: C       No relationships to the British Broadcasting Corporation.
8: C
9: C       Return code;
10: C       ICODE = 0 : no-problem
11: C       ICODE = -1 : suspicious but acceptable.
12: C       ICODE = 1 : fatal error.
13: C   Called in: GEOMIN
14: C   Calls: R8READ, CNTERR
15: C-----
16: C arguments : see description in BODBOX routine.
17: C
18: C --- GRANGE is not set here for this body !! ---
19: C x GRANGE(6) : guess of X,Y,Z range of geometry(to be used in picture?)
20: C       GRANGE(1:2) : min-max of X range
21: C       GRANGE(3:4) : min-max of Y range
22: C       GRANGE(5:6) : min-max of Z range
23: C=====
24:   real*8 SDA(*), BD(*), DINF
25:   real*8 GRANGE(6)
26:   include 'INC/_WORDL'
27:   include 'INC/_IOUNIT'
28: C
29:   ICODE = 0
30: C
31:   ICALL = 30
32:   call R8READ( ' ', BD, NA, -ICALL, IERR )
33:   if ( NA.le.1 ) then
34:     write(IMG,*) 'XXX Number of data for body "BBC" ', IBODN,
35:     & ' is too small.'
36:     call CNTERR( 'FATAL' )
37:   else if ( NA.gt.ICALL ) then
38:     write(IMG,*) 'XXX Number of data for body "BBC" ', IBODN,
39:     & ' exceeds program limit (=',ICALL,')
40:     call CNTERR( 'FATAL' )
41:   end if
42: C
43:   NBBC = min(NA,ICALL)
44:   ICALL = NBBC
45: C
46:   IBODN = INT(BD(1))
47:   IBSDA = (LSDAT-LSDA) /MWORD + 1
48: C
49: C   ... no suraface# count up for this body
50:   NS = 0
51: C
52:   do 100 I=2,NBBC
53:     IB = nint(BD(I))
54:     if( IB.eq.0 ) then
55:       write(IMG,*) '!!! ',I-1,'th' body ID in "BBC" body',
56:       & ' is zero.'
57:       call CNTERR( 'WARNING' )
58:     else if( IB.lt.0 ) then
59:       write(IMG,7000) -IB
60: 7000   format(3x,'XXX(BODBBC) Using negative body ID ',I5,
61:     & ' in "BBC" body, /5x,
62:     & ' but only "HAF" (half space) type of body is possible',
63:     & ' for such a use.')
64:       call CNTERR( 'FATAL' )
65:     end if
66:   100 continue
67: C
68:   LSDAT = LSDAT + 0
69: C
70:   return
71:   end
```


src/shared/bodbox.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODBOX( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C   PURPOSE: TRANSLATE BODY 'BOX' TO SURFACES
6: C     Return code;
7: C     ICODE = 0 : no-problem
8: C     ICODE =-1 : suspicious but acceptable.
9: C     ICODE = 1 : fatal error.
10: C   CALLED IN: GEOMIN
11: C   CALLS: R8READ, CNTERR
12: C-----
13: C
14: C   arguments (i=input, o=output, w=work)
15: C   (generic over BOD*** routines )
16: C
17: C   o ICALL : number of data read as data of a body.
18: C   ( currently used only to check whether BOD*** routine is
19: C     called or not )
20: C   o SDA(*): an array on which surface data are stored.
21: C   i LSDA : starting index of SDA(1) in an single precision array
22: C     on which variable length data including SDA are stored.
23: C   o BD   : array to store body data as input.
24: C   o IBODN: body ID #
25: C   o IBSDA: SDA(*) array index of the first surface of the body.
26: C   o NS   : number of surfaces composing this body.
27: C   io LSDAT: index in an single precision array on which variable
28: C     length data including SDA are stored.
29: C     This indicates the starting position of data stored
30: C     in SDA data array for the body to be input in this routine
31: C     at the beginning, and for the next body before RETURN.
32: C   ( Currently LSDA & LSDAT are used to check memory area
33: C     overflow in the routine called this routine.)
34: C   i DINF : a large distance taken to be infinite.
35: C   o GRANGE(6) : guess of X,Y,Z range of geometry
36: C     (to be used in CGVIEW currently)
37: C     GRANGE(1:2) : min-max of X range
38: C     GRANGE(3:4) : min-max of Y range
39: C     GRANGE(5:6) : min-max of Z range
40: C   o ICODE : return code.
41: C=====
42: C     implicit real*8(D)
43: C     real*8 SDA(*), BD(*), DINF
44: C     real*8 GRANGE(6)
45: C     include 'INC/_WORDL'
46: C     include 'INC/_IOUNIT'
47: C
48: C     .... local variable
49: C     real*8 BR
50: C-----
51: C     ICODE = 0
52: C   ..... ICALL: NUMBER OF BODY DATA + 1
53: C     ICALL = 13
54: C     call R8READ( ' ', BD, NA, -ICALL, IERR )
55: C     if ( NA.ne.ICALL ) then
56: C       write(IMG,*) 'XXX Number of data for body "BOX" ', IBODN,
57: C     & ' is not ', ICALL
58: C       call CNTERR( 'FATAL' )
59: C     end if
60: C
61: C     IBODN = INT(BD(1))
62: C     IBSDA = (LSDAT-LSDA) /MWORD + 1
63: C   ..... NS : NUMBER OF SURFACE .....
64: C     NS = 0
65: C     NERR = 0

```

```

66: C
67: C   ..... SDA(1) IS DATA TO RECOGNIZE THE KIND OF SURFACE .....
68: C     KK + 10000 * NN
69: C     KK: SURFACE IDENTIFIER ( 1/2/3 = SLAB/SPHERE/CYLINDER ETC.
70: C     NN: NUMBER OF NEEDED SDA DATA FOR A SURFACE. ONLY FOR 'ZONEIN'
71: C     ROUTINE.
72: C
73: C     SDA(1) = 1.0 + 10000*6
74: C     NS = NS + 1
75: C
76: C     D1 = BD(9)*BD(13) - BD(10)*BD(12)
77: C     D2 = BD(10)*BD(11) - BD(8)*BD(13)
78: C     D3 = BD(8)*BD(12) - BD(9)*BD(11)
79: C     BR = SQRT(D1**2+D2**2+D3**2)
80: C     if ( BR.eq.0.0D0 ) then
81: C       write(IMG,*) ' XXX Vertex #2 & #3 of "BOX" ', IBODN,
82: C     & ' are parallel or cannot compose a plane. '
83: C       call CNTERR( 'FATAL' )
84: C       NERR = NERR + 1
85: C     else
86: C       SDA(2) = D1/BR
87: C       SDA(3) = D2/BR
88: C       SDA(4) = D3/BR
89: C       SDA(5) = SDA(2)*BD(2) + SDA(3)*BD(3) + SDA(4)*BD(4)
90: C       SDA(6) = SDA(5) + SDA(2)*BD(5) + SDA(3)*BD(6) + SDA(4)*BD(7)
91: C     end if
92: C
93: C   ..... SDA(7) IS DATA TO RECOGNIZE THE KIND OF SURFACE .....
94: C     SDA(7) = 1.0 + 10000*6
95: C     NS = NS + 1
96: C
97: C     D1 = BD(12)*BD(7) - BD(13)*BD(6)
98: C     D2 = BD(13)*BD(5) - BD(11)*BD(7)
99: C     D3 = BD(11)*BD(6) - BD(12)*BD(5)
100: C     BR = SQRT(D1**2+D2**2+D3**2)
101: C     if ( BR.eq.0.0D0 ) then
102: C       write(IMG,*) ' XXX Vertex #3 & #1 of "BOX" ', IBODN,
103: C     & ' are parallel or cannot compose a plane. '
104: C       call CNTERR( 'FATAL' )
105: C       NERR = NERR + 1
106: C     else
107: C       SDA(8) = D1/BR
108: C       SDA(9) = D2/BR
109: C       SDA(10) = D3/BR
110: C       SDA(11) = SDA(8)*BD(2) + SDA(9)*BD(3) + SDA(10)*BD(4)
111: C       SDA(12) = SDA(11) + SDA(8)*BD(5) + SDA(9)*BD(6) + SDA(10)*
112: C     & BD(10)
113: C     end if
114: C
115: C   ..... SDA(13) IS DATA TO RECOGNIZE THE KIND OF SURFACE .....
116: C
117: C     SDA(13) = 1.0 + 10000*6
118: C     NS = NS + 1
119: C
120: C     D1 = BD(6)*BD(10) - BD(7)*BD(9)
121: C     D2 = BD(7)*BD(8) - BD(5)*BD(10)
122: C     D3 = BD(5)*BD(9) - BD(6)*BD(8)
123: C     BR = SQRT(D1**2+D2**2+D3**2)
124: C     if ( BR.eq.0.0D0 ) then
125: C       write(IMG,*) ' XXX Vertex #1 & #2 of "BOX" ', IBODN,
126: C     & ' are parallel or cannot compose a plane. '
127: C       call CNTERR( 'FATAL' )
128: C       NERR = NERR + 1
129: C     else
130: C       SDA(14) = D1/BR

```

src/shared/bodbox.f

```
131:      SDA(15) = D2/BR
132:      SDA(16) = D3/BR
133:      SDA(17) = SDA(14)*BD(2) + SDA(15)*BD(3) + SDA(16)*BD(4)
134:      SDA(18) = SDA(17) + SDA(14)*BD(11) + SDA(15)*BD(12) + SDA(16)*
135:      &      BD(13)
136:      end if
137: C
138:      LSDAT = LSDAT + 18*MWORD
139: C
140:      do 120 M=1,8
141:      I = mod(M,2)
142:      J = mod(M/2,2)
143:      K = mod(M/4,2)
144: C
145:      DOX = BD(2) + I*BD(5) + J*BD(8) + K*BD(11)
146:      DOY = BD(3) + I*BD(6) + J*BD(9) + K*BD(12)
147:      DOZ = BD(4) + I*BD(7) + J*BD(10) + K*BD(13)
148: C
149:      GRANGE(1) = MIN(GRANGE(1),DOX)
150:      GRANGE(2) = MAX(GRANGE(2),DOX)
151:      GRANGE(3) = MIN(GRANGE(3),DOY)
152:      GRANGE(4) = MAX(GRANGE(4),DOY)
153:      GRANGE(5) = MIN(GRANGE(5),DOZ)
154:      GRANGE(6) = MAX(GRANGE(6),DOZ)
155: 120 continue
156: C
157:      if ( NERR.gt.0 ) ICODE = 1
158:      return
159:      end
```

src/shared/bodcyl.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODCYL( ICALL, SDA,  LSDA, BD,  IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'CYL' TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:   real*8 SDA(*), BD(*), DINF
16:   real*8 GRANGE(6)
17:   include 'INC/WORDL'
18:   include 'INC/IUNIT'
19: C-----
20:   ICODE = 0
21: C
22:   ICALL = 6
23:   call R8READ( ' ', BD, NA, -ICALL, IERR )
24:   if ( NA.ne.ICALL ) then
25:     write(IMG,*) 'XXX Number of data for body "CYL" ', IBODN
26:     & ' is not ', ICALL
27:     call CNTERR( 'FATAL' )
28:   end if
29: C
30:   IBODN = INT(BD(1))
31:   IBSDA = (LSDAT-LSDA) /MWORD + 1
32: C
33:   NERR = 0
34: C
35: C .... IF HEIGHT OF CYLINDER (BD(5) = HZ) >= DINF , OR = 0.0,
36: C   CYLINDER IS INFINITE AND TOP & BOTTOM SIDE ARE UNNECESSARY
37: C
38:   if ( ABS(BD(5)).lt.DINF.and.BD(5).ne.0.0 ) then
39: C
40: C .... SLAB : SDA2 *X + SDA3 *Y + SDA4 *Z - SDA5 (OR SDA6 ) = 0
41:   SDA(1) = 1.0 + 10000*6
42: C .. SDA(1) IS A DATA FOR SURFACE RECOGNITION.(SEE COMMENT IN 'BODBOX')
43:   SDA(2) = 0.0
44:   SDA(3) = 0.0
45:   SDA(4) = SIGN(1.0D0,BD(5))
46:   SDA(5) = BD(4)
47:   SDA(6) = BD(4) + BD(5)
48:   LSDAT = LSDAT + 14*MWORD
49:   NS = 2
50:   GRANGE(5) = MIN(GRANGE(5),MIN(SDA(5),SDA(6)))
51:   GRANGE(6) = MAX(GRANGE(6),MAX(SDA(5),SDA(6)))
52:   else
53:     LSDAT = LSDAT + 8*MWORD
54:     NS = 1
55:     write(IPR,*) '<<MESSAGE>> "CYL" ', IBODN,
56:   & ' HAS INFINITE LENGTH.'
57:     call CNTERR( 'MESSAGE' )
58:   end if
59: C
60: C .... SDA2,SDA3,SDA4 : CENTER OF BOTTOM SDA5,SDA6,SDA7 : DIRECTION
61: C   SDA8 : SQUARE OF RADIUS
62:   K = (NS-1)*6
63:   SDA(K+1) = 3.0 + 10000*8
64: C .. SDA(1) IS A DATA FOR SURFACE RECOGNITION.(SEE COMMENT IN 'BODBOX')
65:   SDA(K+2) = BD(2)

```

```

66:   SDA(K+3) = BD(3)
67:   SDA(K+4) = BD(4)
68:   SDA(K+5) = 0.0
69:   SDA(K+6) = 0.0
70:   SDA(K+7) = 1.0
71:   SDA(K+8) = BD(6)**2
72:   if ( BD(6).lt.0.0D0 ) then
73:     write(IMG,*) ' XXX A NEGATIVE RADIUS ', BD(6),
74:   & ' IS GIVEN TO "CYL" ', IBODN, '.'
75:     call CNTERR( 'FATAL' )
76:     ICODE = 1
77:   else if ( BD(6).eq.0.0D0 ) then
78:     write(IMG,*) ' !!! "CYL" ', IBODN,
79:   & ' SHRINKS TO ZERO-RADIUS CYLINDER.'
80:     call CNTERR( 'WARNING' )
81:     ICODE = -1
82:   end if
83: C
84:   GRANGE(1) = MIN(GRANGE(1),BD(2)-BD(6))
85:   GRANGE(2) = MAX(GRANGE(2),BD(2)+BD(6))
86:   GRANGE(3) = MIN(GRANGE(3),BD(3)-BD(6))
87:   GRANGE(4) = MAX(GRANGE(4),BD(3)+BD(6))
88: C
89:   return
90: end

```

src/shared/bodell.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODELL( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,   GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'ELL' (ELLIPSOID BY REVOLUTION) TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:   implicit real*8(D)
16:   real*8 SDA(*), BD(*), DINF
17:   real*8 GRANGE(6)
18:   include 'INC/_WORDL'
19:   include 'INC/_IOUNIT'
20: C-----
21: c
22:   ICODE = 0
23: c
24:   ICALL = 8
25:   call R8READ( ' ', BD, NA, -ICALL, IERR )
26:   if ( NA.ne.ICALL ) then
27:     write(IMG,*) 'XXX Number of data for body "ELL" ', IBODN,
28:     & ' is not ', ICALL
29:     call CNTERR( 'FATAL' )
30:   end if
31: c
32:   IBODN = INT(BD(1))
33:   IBSDA = (LSDAT-LSDA) /MWORD + 1
34:   NS = 1
35:   NERR = 0
36: c .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
37:   SDA(1) = 4.0 + 10000*11
38: c
39: c ..... BD(2),BD(3),BD(4) : FOCUS 1  BD(5),BD(6),BD(7) : FOCUS 2
40: c    BD(8) : LENGTH OF MAJOR AXIS
41: c
42: c >>>> CENTER.
43:   DCX = (BD(2)+BD(5))*0.5
44:   DCY = (BD(3)+BD(6))*0.5
45:   DCZ = (BD(4)+BD(7))*0.5
46: c >>>> VECTOR FROM FOCUS 1 TO FOCUS 2.
47:   DNX = BD(5) - BD(2)
48:   DNY = BD(6) - BD(3)
49:   DNZ = BD(7) - BD(4)
50: c >>>> SQUARE OF LENGTH OF MAJOR & MINOR AXES.
51:   if ( BD(8).lt.0.0D0 ) then
52:     write(IMG,*) ' XXX A NEGATIVE MAJOR AXIS LENGTH ', BD(8),
53:     & ' IS GIVEN TO "ELL" ', IBODN, ' . '
54:     call CNTERR( 'FATAL' )
55:     NERR = NERR + 1
56:   else if ( BD(8).eq.0.0D0 ) then
57:     write(IMG,*) '!!! "ELL" ', IBODN,
58:     & ' SHRINKS TO ZERO-VOLUME ELLIPSOID.'
59:     call CNTERR( 'WARNING' )
60:     ICODE = -1
61:   end if
62:
63:   DL = BD(8)**2
64:   DR = DL - DNX**2 - DNY**2 - DNZ**2
65:

```

```

66:   if ( DR.le.0.0D0 ) then
67:     write(IMG,*) '!!! "ELL" ', IBODN,
68:     & ' SHRINKS TO ZERO-VOLUME ELLIPSOID.'
69:     call CNTERR( 'WARNING' )
70:     ICODE = -1
71:   end if
72: c
73: c ..... EQUATION .....
74: c
75: c  $\{(X-C)*N\}^2/(L/2)^2 + ((X-C)^2 - \{(X-C)*N\}^2)/(R/2)^2 - 1 = 0$ 
76: c WHERE
77: c
78: c  $\bar{X}$  : COORDINATES VECTOR.  $\bar{C}$  : CENTER.  $\bar{N}$  : UNIT VECTOR ON THE
79: c    DIRECTION FROM FOCUS 1 TO FOCUS 2
80: c L : LENGTH OF MAJOR AXIS. R : LENGTH OF MINOR AXIS
81: c
82: c  $(R/2)^2 + (K/2)^2 = (L/2)^2$  --->  $R^2 = L^2 - K^2$ 
83: c (K: DISTANCE BETWEEN FOCI)
84: c
85: c ..... EXPANDED FORM .....
86: c
87: c  $DL*\{(X-DCX)^2 + (Y-DCY)^2 + (Z-DCZ)^2\}$ 
88: c  $- \{(X-DCX)*DNX + (Y-DCY)*DNY + (Z-DCZ)*DNZ\}^2 - DL*DR/4 = 0$ 
89: c
90:   SDA(2) = DL - DNX**2
91:   SDA(3) = DL - DNY**2
92:   SDA(4) = DL - DNZ**2
93:   SDA(5) = -2.0*DNX*DNY
94:   SDA(6) = -2.0*DNY*DNZ
95:   SDA(7) = -2.0*DNZ*DNX
96:   DD = DNX*DCX + DNY*DCY + DNZ*DCZ
97:   SDA(8) = 2.0*(DNX*DD-DL*DCX)
98:   SDA(9) = 2.0*(DNY*DD-DL*DCY)
99:   SDA(10) = 2.0*(DNZ*DD-DL*DCZ)
100:   SDA(11) = DL*(DCX**2+DCY**2+DCZ**2) - DD**2 - DL*DR/4.0
101:   LSDAT = LSDAT + 11*MWORD
102: c
103: c GRANGE(1) = -SQRT(ABS(DL))/2
104: c GRANGE(2) = -GRANGE(1)
105: c GRANGE(3) = -SQRT(ABS(DR))/2
106: c GRANGE(4) = -GRANGE(3)
107: c GRANGE(5) = GRANGE(3)
108: c GRANGE(6) = GRANGE(4)
109:   call GRCONV( GRANGE, DCX, DCY, DCZ, DNX, DNY, DNZ, 0.0d0, 0.0d0, 0.0d0,
110:   & DINF )
111: c
112:   if ( NERR.gt.0 ) ICODE = 1
113:   return
114: end

```

src/shared/bodelt.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODELT( ICALL, SDA, LSDA, BD, IBODN, IBSDA, NS,
3:     & LSDAT, DINF, GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'ELT' (ELLIPTIC TORUS) TO SURFACE DATA
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS:    R8READ,CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15: C    implicit real*8(A-H,O-Z)
16: C    real*8 SDA(*), BD(*), DINF
17: C    real*8 GRANGE(6)
18: C    include 'INC/_WORDL'
19: C    include 'INC/_IUNIT'
20: C
21: C-----
22: C    ICODE = 0
23: C
24: C    ICALL = 11
25: C    call R8READ( ' ', BD, NA, ICALL, IERR )
26: C    if ( NA.ne.ICALL ) then
27: C      write(IMG,*) 'XXX Number of data for body "ELT" ', IBODN,
28: C      & ' is not ',ICALL
29: C      call CNTERR( 'FATAL' )
30: C    end if
31: C
32: C    IBODN = INT(BD(1))
33: C    IBSDA = (LSDAT-LSDA) /MWORD + 1
34: C    NS = 1
35: C    NERR = 0
36: C
37: C .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
38: C
39: C    SDA(1) = 6.0 + 10000*11
40: C
41: C
42: C    V = BD(2),BD(3),BD(4) : Center of a torus
43: C    N = BD(5),BD(6),BD(7) : Rotation axis vector
44: C    A = BD(8) : Mean radius
45: C    R1 = BD(9) : Short/long radius of cross-section oval
46: C    R2 = BD(10) : Short/long radius of cross-section oval
47: C    TH = BD(11) : Tilting angle of oval (rotation plane
48: C                to raduis-2, not radius-1 !! as described
49: C                in JAERI Data/Code 94-007 page 22)
50: C
51: C..... EQUATION .....
52: C
53: C      2      2
54: C      B T + (R-A) + C T(R-A) - D = 0
55: C
56: C  Where
57: C
58: C    T = NX*(X-VX) + NY*(Y-VY)+NZ*(Z-VZ)
59: C    (VX,VY,VZ) : Center of the torus
60: C    (NX,NY,NZ) : Rotation axis vector of the torus (unit vector)
61: C
62: C    R : Distance from rotation axis
63: C      2      2      2
64: C      R = (X-VX) + (Y-VY) + (Z-VZ) - T
65: C

```

```

66: C    A : Rotation radius
67: C
68: C    B =
69: C      (R1*SIN(TH))**2+(R2*COS(TH))**2)/(R2*SIN(TH))**2+(R1*COS(TH))**2)
70: C    C =
71: C      2*(R1**2-R2**2)*SIN(TH)*COS(TH)/((R2*SIN(TH))**2+(R1*COS(TH))**2)
72: C    D = (R1*R2)**2/((R2*SIN(TH))**2+(R1*COS(TH))**2)
73: C
74: C..... SDA ARRAY .....
75: C
76: C    SDA(2:4) = (VX,VY,VZ)
77: C    SDA(5:7) = (NX,NY,NZ) (NORMALIZED)
78: C    SDA(8) = A
79: C    SDA(9) = B
80: C    SDA(10) = C
81: C    SDA(11) = D
82: C.....
83: C
84: C    SDA(2) = BD(2)
85: C    SDA(3) = BD(3)
86: C    SDA(4) = BD(4)
87: C
88: C    DL = SQRT(BD(5)**2+BD(6)**2+BD(7)**2)
89: C    if ( DL.eq.0.0D0 ) then
90: C      write(IMG,*) ' XXX ZERO LENGTH AXIS VECTOR FOR BODY "ELT" ',
91: C      & IBODN, ' '
92: C      call CNTERR( 'FATAL' )
93: C      NERR = NERR + 1
94: C    else
95: C      SDA(5) = BD(5) /DL
96: C      SDA(6) = BD(6) /DL
97: C      SDA(7) = BD(7) /DL
98: C    end if
99: C
100: C    SDA(8) = BD(8)
101: C
102: C    R1 = BD(9)
103: C    R2 = BD(10)
104: C    if ( R1.eq.0.0D0 .or. R2.eq.0.0D0 ) then
105: C      write(IMG,*) ' !!! ZERO VOLUME "ELT" ', IBODN, ' R1 =', R1,
106: C      & ' R2 =', R2
107: C      call CNTERR( 'WARNING' )
108: C      ICODE = -1
109: C    end if
110: C    TH = BD(11) /180.0*3.141592653589793D0
111: C    SDA(9) = (R2*COS(TH))**2 + (R1*SIN(TH))**2
112: C    SDA(11) = 1.0/((R2*SIN(TH))**2+(R1*COS(TH))**2)
113: C    SDA(9) = SDA(9)*SDA(11)
114: C    SDA(10) = (R1**2-R2**2)*SIN(2.0*TH)*SDA(11)
115: C    SDA(11) = (R1*R2)**2*SDA(11)
116: C
117: C
118: C    LSDAT = LSDAT + 11*MWORD
119: C
120: C    GRANGE(1) = -max(BD(9),BD(10))
121: C    GRANGE(2) = max(BD(9),BD(10))
122: C    GRANGE(3) = -(BD(8) + max(BD(9),BD(10)))
123: C    GRANGE(4) = BD(8) + max(BD(9),BD(10))
124: C    GRANGE(5) = GRANGE(3)
125: C    GRANGE(6) = GRANGE(4)
126: C    call GRCONV( GRANGE, BD(2),BD(3),BD(4),BD(5),BD(6),BD(7),
127: C    & 0.0D0, 0.0D0, 0.0D0, DINF )
128: C
129: C    if ( NERR.ne.0 ) ICODE = 1
130: C    return

```

src/shared/bodelt.f

131: end



src/shared/bodgel.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODGEL( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'GEL' (GENERAL ELLIPSOID) TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see descriptpion in BODBOX routine.
14: C=====
15: C    implicit real*8(B-D)
16: C    real*8 SDA(*), BD(*), DINF
17: C    real*8 GRANGE(6)
18: C    include 'INC/_WORDL'
19: C    include 'INC/_IOUNIT'
20: C
21: C    real*8 PAI
22: C    parameter( PAI = 3.1415926535897932D0 )
23: C-----
24: C    ICODE = 0
25: C
26: C    ICALL = 13
27: C    call R8READ( ' ', BD, NA, -ICALL, IERR )
28: C    if ( NA.ne.ICALL ) then
29: C      write(IMG,*) 'XXX Number of data for body "GEL" ', IBODN,
30: C      & ' is not ',ICALL
31: C      call CNTERR( 'FATAL' )
32: C    end if
33: C
34: C    IBODN = INT(BD(1))
35: C    IBSDA = (LSDAT-LSDA) /MWORD + 1
36: C    NS = 1
37: C    NERR = 0
38: C
39: C .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
40: C
41: C ..... BD(2),BD(3),BD(4) : CENTER   BD(5),BD(6),BD(7) : AXIS 1
42: C    BD(8),BD(9),BD(10) : AXIS 2   BD(11),BD(12),BD(13) : AXIS LENGTH
43: C
44: C    SDA(1) = 4.0 + 10000*11
45: C
46: C >>>> NORMALIZE AXIS VECTORS
47: C
48: C    DD = SQRT(BD(5)**2+BD(6)**2+BD(7)**2)
49: C
50: C    if ( DD.eq.0.0D0 ) then
51: C      write(IMG, '(1x,a,i6,a)')
52: C      & ' XXX PRIMARY AXIS LENGTH LENGTH OF "GEL" ', IBODN, ' IS ZERO.'
53: C      call CNTERR( 'FATAL' )
54: C      NERR = NERR + 1
55: C      D1X = 1
56: C      D1Y = 0
57: C      D1Z = 0
58: C    else
59: C      D1X = BD(5) /DD
60: C      D1Y = BD(6) /DD
61: C      D1Z = BD(7) /DD
62: C    end if
63: C
64: C    DD = SQRT(BD(8)**2+BD(9)**2+BD(10)**2)
65:

```

```

66:   if ( DD.eq.0.0D0 ) then
67:     write(IMG, '(1x,a,i6,a)')
68:     & ' XXX SECONDARY AXIS LENGTH LENGTH OF "GEL" ', IBODN,
69:     & ' IS ZERO.'
70:     call CNTERR( 'FATAL' )
71:     NERR = NERR + 1
72:     D2X = 0
73:     D2Y = 1
74:     D2Z = 0
75:   else
76:     D2X = BD(8) /DD
77:     D2Y = BD(9) /DD
78:     D2Z = BD(10) /DD
79:   end if
80: C
81: C .... check angle between axis1 & axis2 ...
82: C
83: C    DC = D1X*D2X + D1Y*D2Y + D1Z*D2Z
84: C    DD = ACOS(DC)
85: C    if ( ABS(DD-PAI*0.5D0).gt.PAI*1.0D-7 ) then
86: C      write(IMG, '(1x,a,i6,a, 1p,e12.5/5x,a,a)')
87: C      & ' !!! TWO AXES OF "GEL" ', IBODN,
88: C      & ' DOES NOT MAKE A RIGHT ANGLE. DEGREE = ', DD/PAI*180D0,
89: C      & ' Second axis is modified to be perpendicular to the',
90: C      & ' first one.'
91: C      call CNTERR( 'WARNING' )
92: C      ICODE = -1
93: C
94: C      D2X = D2X - D1X*DC
95: C      D2Y = D2Y - D1Y*DC
96: C      D2Z = D2Z - D1Z*DC
97: C    end if
98: C
99: C >>>> THIRD AXIS
100: C
101: C    D3X = BD(6)*BD(10) - BD(9)*BD(7)
102: C    D3Y = BD(7)*BD(8) - BD(10)*BD(5)
103: C    D3Z = BD(5)*BD(9) - BD(8)*BD(6)
104: C    DD = SQRT(D3X**2+D3Y**2+D3Z**2)
105: C    if ( DD.eq.0.0D0 ) then
106: C      write(IMG, '(1x,a,i6,a)')
107: C      & ' XXX CANNOT CALCULATE THIRD AXIS VECTOR OF "GEL" ',
108: C      & IBODN, ' .'
109: C      call CNTERR( 'FATAL' )
110: C      NERR = NERR + 1
111: C      D3X = 0
112: C      D3Y = 0
113: C      D3Z = 1
114: C    else
115: C      D3X = D3X/DD
116: C      D3Y = D3Y/DD
117: C      D3Z = D3Z/DD
118: C    end if
119: C
120: C ..... EQUATION .....
121: C
122: C    { (X-BD2)*D1X + (Y-BD3)*D1Y + (Z-BD4)*D1Z }**2/BD11**2
123: C    + { (X-BD2)*D2X + (Y-BD3)*D2Y + (Z-BD4)*D2Z }**2/BD12**2
124: C    + { (X-BD2)*D3X + (Y-BD3)*D3Y + (Z-BD4)*D3Z }**2/BD13**2 - 1 < 0
125: C
126: C
127: C    if ( BD(11).le.0.0 .or. BD(12).le.0.0 .or. BD(13).le.0.0 ) then
128: C      write(IMG, '(1x,a,i6,a/5x,3(a,1p,e14.7))') ' XXX "GEL" ', IBODN,
129: C      & ' SHRINKS TO ZERO-VOLUME ELLIPSOID.', ' AXIS1=',
130: C      & BD(11), ' AXIS2=', BD(12), ' AXIS3=', BD(13)

```

src/shared/bodgel.f

```
131:      call CNTERR( 'FATAL' )
132:      NERR      = NERR + 1
133:      else
134:      DA        = 1.0/BD(11)**2
135:      DB        = 1.0/BD(12)**2
136:      DC        = 1.0/BD(13)**2
137:  C
138:      DV1       = BD(2)*D1X + BD(3)*D1Y + BD(4)*D1Z
139:      DV2       = BD(2)*D2X + BD(3)*D2Y + BD(4)*D2Z
140:      DV3       = BD(2)*D3X + BD(3)*D3Y + BD(4)*D3Z
141:      SDA(2)    = DA*D1X**2 + DB*D2X**2 + DC*D3X**2
142:      SDA(3)    = DA*D1Y**2 + DB*D2Y**2 + DC*D3Y**2
143:      SDA(4)    = DA*D1Z**2 + DB*D2Z**2 + DC*D3Z**2
144:      SDA(5)    = 2.0*(DA*D1X*D1Y+DB*D2X*D2Y+DC*D3X*D3Y)
145:      SDA(6)    = 2.0*(DA*D1Y*D1Z+DB*D2Y*D2Z+DC*D3Y*D3Z)
146:      SDA(7)    = 2.0*(DA*D1Z*D1X+DB*D2Z*D2X+DC*D3Z*D3X)
147:      SDA(8)    = -2.0*(DA*D1X*DV1+DB*D2X*DV2+DC*D3X*DV3)
148:      SDA(9)    = -2.0*(DA*D1Y*DV1+DB*D2Y*DV2+DC*D3Y*DV3)
149:      SDA(10)   = -2.0*(DA*D1Z*DV1+DB*D2Z*DV2+DC*D3Z*DV3)
150:      SDA(11)   = DA*DV1**2 + DB*DV2**2 + DC*DV3**2 - 1
151:  end if
152:  C
153:      GRANGE(1) = -BD(11)
154:      GRANGE(2) =  BD(11)
155:      GRANGE(3) = -BD(12)
156:      GRANGE(4) =  BD(12)
157:      GRANGE(5) = -BD(13)
158:      GRANGE(6) =  BD(13)
159:      call GRCONV( GRANGE, BD(2),BD(3),BD(4), BD(5),BD(6),BD(7),
160:      &            BD(8),BD(9),BD(10),
161:      &            DINF )
162:  C
163:  C
164:      LSDAT      = LSDAT + 11*MWORD
165:  C
166:      if ( NERR.gt.0 ) ICODE = 1
167:      return
168:  end
```


src/shared/bodgqs.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODGQS( ICALL, SDA,  LSDA, BD,  IBODN, IBSDA, NS,
3:     &                LSDAT, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'GQS' (GENERAL QUADRATIC SURFACE) TO SURFACE
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C.....
11: C  << input data definition >>
12: C    SDA(1) = 4 + 10000*11
13: C
14: C    SDA(2)*X**2 + SDA(3)*Y**2 + SDA(4)*Z**2 +
15: C    SDA(5)*X*Y + SDA(6)*Y*Z + SDA(7)*Z*X +
16: C    SDA(8)*X + SDA(9)*Y + SDA(10)*Z + SDA(11) < 0
17: C
18: C    or
19: C
20: C    / SDA(2) SDA(5) SDA(7) SDA(8) \ X
21: C    / SDA(3) SDA(6) SDA(9) \ Y
22: C    ( X Y Z 1 ) / 0 0 SDA(4) SDA(10) \ Z < 0
23: C    / 0 0 0 SDA(11) \ 1
24: C    \ \ \ \ \ /
25: C.....
26: C
27: C  CALLED IN: GEOMIN
28: C  CALLS: R8READ,CNTERR
29: C-----
30: C arguments : see description in BODBOX routine.
31: C=====
32:   implicit real*8(B-D)
33:   real*8 SDA(*), BD(*)
34:   include 'INC/_WORDL'
35:   include 'INC/_IOUNIT'
36: C
37:   real*8 PAI
38:   parameter( PAI = 3.1415926535897932D0 )
39: C-----
40:   ICODE = 0
41: C
42:   ICALL = 11
43:   call R8READ( ' ', BD, NA, -ICALL, IERR )
44:   if ( NA.ne.ICALL ) then
45:     write(IMG,*) 'XXX Number of data for body "GQS" ', IBODN,
46:     & ' is not ',ICALL
47:     call CNTERR( 'FATAL' )
48:   end if
49: C
50:   IBODN = INT(BD(1))
51:   IBSDA = (LSDAT-LSDA) /MWORD + 1
52:   NS = 1
53:   NERR = 0
54: C
55: C .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
56: C
57:   SDA(1) = 4.0 + 10000*11
58: C
59:   SDA(2) = BD(2)
60:   SDA(3) = BD(3)
61:   SDA(4) = BD(4)
62:   SDA(5) = BD(5)
63:   SDA(6) = BD(6)
64:   SDA(7) = BD(7)
65:   SDA(8) = BD(8)
66:   SDA(9) = BD(9)
67:   SDA(10) = BD(10)
68:   SDA(11) = BD(11)
69: C
70:   LSDAT = LSDAT + 11*MWORD
71: C
72:   if ( NERR.gt.0 ) ICODE = 1
73:   return
74: end

```

src/shared/bodhaf.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine BODHAF( ICALL, SDA,  LSDA,  BD,   IBODN, IBSDA, NS,
3:     &               LSDAT, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'HAF' (HALF SPACE) TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:   real*8 SDA(*), BD(*), BR
16:   include 'INC/_WORDL'
17:   include 'INC/_IOWNIT'
18: c
19:   ICODE = 0
20: C ..... ICALL: NUMBER OF BODY DATA + 1
21:   ICALL = 5
22:   call R8READ( ' ', BD, NA, -ICALL, IERR )
23:   if ( NA.ne.ICALL ) then
24:     write(IMG,*) 'XXX Number of data for body "HAF" ', IBODN,
25:     & ' is not ', ICALL
26:     call CNTERR( 'FATAL' )
27:   end if
28: C
29:   IBODN = INT(BD(1))
30:   IBSDA = (LSDAT-LSDA) /MWORD + 1
31: C ..... NS : NUMBER OF SURFACE .....
32:   NS = 1
33: C
34: C
35: C (BD(2)*X + BD(3)*Y + BD(4)*Z)/BR + BD(5) < 0 (INSIDE)
36: C (ADOPT THE DIFINITION IN 'SAM-CE')
37: C BD(2),BD(3),BD(4) IS AN OUTWARD NORMAL VECTOR.
38: C IN 'MVP' CODE THIS EQUATION IS SLIGHTLY CHANGED
39: C
40: C SDA(2)*X + SDA(3)*Y + SDA(4)*Z - SDA(5) < 0 (INSIDE)
41: C
42: C ..... SDA(1) IS DATA TO RECOGNIZE TYPE OF THE SURFACE .....
43:   SDA(1) = 5.0 + 10000*5
44:   BR = SQRT(BD(2)*BD(2)+BD(3)*BD(3)+BD(4)*BD(4))
45:   if ( BR.eq.0.0D0 ) then
46:     write(IMG,*) ' XXX Zero length normal vector for body "HAF" ',
47:     & IBODN, ' '
48:     call CNTERR( 'FATAL' )
49:     ICODE = 1
50:   else
51:     SDA(2) = BD(2) /BR
52:     SDA(3) = BD(3) /BR
53:     SDA(4) = BD(4) /BR
54:     SDA(5) = -BD(5)
55:   end if
56:   LSDAT = LSDAT + 5*MWORD
57:   return
58: end
```

src/shared/bodhex.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODHEX( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE:  TRANSLATE BODY 'HEX' (HEXAGONAL PRISM) TO SURFACES
6: C            (ADDED 1989 JANUARY)
7: C            Return code:
8: C            ICODE = 0 : no-problem
9: C            ICODE =-1 : suspicious but acceptable.
10: C            ICODE = 1 : fatal error.
11: C  CALLED IN: GEOMIN
12: C  CALLS:      R8READ, CNTERR
13: C-----
14: C arguments : see description in BODBOX routine.
15: C=====
16:   implicit real*8(D)
17:   real*8 SDA(*), BD(*), DINF
18:   real*8 GRANGE(6)
19: C
20:   include 'INC/_WORDL'
21:   include 'INC/_IUNIT'
22: C
23:   real*8 PAI
24:   parameter( PAI = 3.1415926535897932D0 )
25: C
26: C-----
27:   ICODE = 0
28: C
29: C ..... ICALL: NUMBER OF BODY DATA + 1
30:   ICALL = 11
31:   call R8READ( ' ', BD, NA, -ICALL, IERR )
32: C
33:   if ( NA.ne.ICALL ) then
34:     write(IMG,*) 'XXX Number of data for body "HEX" ', IBODN,
35:   &      ' is not ', ICALL
36:     call CNTERR( 'FATAL' )
37:   end if
38: C
39:   IBODN = INT(BD(1))
40:   IBSDA = (LSDAT-LSDA) /MWORD + 1
41: C
42:   NERR = 0
43: C
44: C ..... BD2,BD3,BD4 : CENTER OF BASE   BD5,BD6,BD7 : AXIS VECTOR
45: C            BD8      : WIDTH
46: C            BD9,BD10,BD11: NORMAL VECTOR OF LATERAL SIDES.
47: C
48: C
49: C
50: C
51: C
52: C
53: C
54: C
55: C
56: C
57:   DL = SQRT(BD(5)**2+BD(6)**2+BD(7)**2)
58:   if ( DL.eq.0.0d0 ) then
59:     write(IMG,*) 'XXX Axis vector length of "HEX" ', IBODN,
60:   &      ' is zero.'
61:     call CNTERR( 'FATAL' )
62:     BD(7) = 1.0
63:     NERR = NERR + 1
64:   end if
65: C

```

```

66:   DH = SQRT(BD(9)**2+BD(10)**2+BD(11)**2)
67:   if ( DH.eq.0.0d0 ) then
68:     write(IMG,*) 'XXX Side vector length of "HEX" ', IBODN,
69:   &      ' is zero.'
70:     call CNTERR( 'FATAL' )
71:     NERR = NERR + 1
72:     DY1 = 1.0D0
73:     DY2 = 0.0D0
74:     DY3 = 0.0D0
75:   else
76:     BD(9) = BD(9) /DH
77:     BD(10) = BD(10) /DH
78:     BD(11) = BD(11) /DH
79: C
80:     DC = BD(5)*BD(9) + BD(6)*BD(10) + BD(7)*BD(11)
81:     DD = ACOS(DC/DL)
82:     if ( ABS(DD-PAI*0.5D0).gt.PAI*1.0D-7 ) then
83:       write(IMG, ' (1X,A,I6,A, 1P,E12.5/5X,A,A)')
84:     &      '!!! Two vectors of "HEX" ', IBODN,
85:     &      ' does not make right angle. Degree = ', DD/PAI*180D0,
86:     &      'Side vector is modified to be perpendicular to the',
87:     &      ' axis vector.'
88:     call CNTERR( 'WARNING' )
89:     ICODE = -1
90:     BD(9) = BD(9) - BD(5)*DC
91:     BD(10) = BD(10) - BD(6)*DC
92:     BD(11) = BD(11) - BD(7)*DC
93:     DH = SQRT(BD(9)**2+BD(10)**2+BD(11)**2)
94:     BD(9) = BD(9) /DH
95:     BD(10) = BD(10) /DH
96:     BD(11) = BD(11) /DH
97:   end if
98: C
99:   DY1 = BD(6)*BD(11) - BD(7)*BD(10)
100:   DY2 = BD(7)*BD(9) - BD(5)*BD(11)
101:   DY3 = BD(5)*BD(10) - BD(6)*BD(9)
102:   DDY = SQRT(DY1**2+DY2**2+DY3**2)
103:   if ( DDY.eq.0.0D0 ) then
104:     write(IMG,*)
105:   &      'XXX Cannot generate 3rd axis vector of "HEX" ', IBODN, '.'
106:     call CNTERR( 'FATAL' )
107:     NERR = NERR + 1
108:   else
109:     DY1 = DY1/DDY
110:     DY2 = DY2/DDY
111:     DY3 = DY3/DDY
112:   end if
113: end if
114: C
115: C ..... HEXAGONAL SIDE ARE MADE OF THREE SLABS
116: C
117: C >>>>>>> SIDE 1 ("LATERAL" SIDES)
118: C
119:   SDA(1) = 1.0 + 10000*6
120:   SDA(2) = BD(9)
121:   SDA(3) = BD(10)
122:   SDA(4) = BD(11)
123:   DD = BD(2)*SDA(2) + BD(3)*SDA(3) + BD(4)*SDA(4)
124:   SDA(5) = DD - BD(8)*0.5
125:   SDA(6) = DD + BD(8)*0.5
126: C
127: C >>>>>>> SIDE 2 ( 60 DEGREE ROTATION OF SIDE 1)
128: C
129:   SDA(7) = 1.0 + 10000*6
130:   DAL = 0.5

```

src/shared/bodhex.f

```
131:      DA2      = SQRT(3.0D0)*0.5
132:      SDA(8)   = DA1*BD(9) + DA2*DY1
133:      SDA(9)   = DA1*BD(10) + DA2*DY2
134:      SDA(10)  = DA1*BD(11) + DA2*DY3
135:      DD       = SQRT(SDA(8)**2+SDA(9)**2+SDA(10)**2)
136:      if ( DD.eq.0.0D0 ) then
137:        write(IMG,*)
138:        &      'XXX CANNOT FIX NORMAL VECTOR TO "HEX" ',IBODN,'.'
139:        call CNTERR( 'FATAL' )
140:        NERR    = NERR + 1
141:      else
142:        SDA(8)  = SDA(8) /DD
143:        SDA(9)  = SDA(9) /DD
144:        SDA(10) = SDA(10) /DD
145:      end if
146:      SDA(11) = BD(2)*SDA(8) + BD(3)*SDA(9) + BD(4)*SDA(10) - BD(8)*0.5
147:      SDA(12) = SDA(11) + BD(8)
148: C
149: C >>>>>>> SIDE 3 (120 DEGREE ROTATION OF SIDE 1)
150: C
151:      SDA(13) = 1.0 + 10000*6
152:      SDA(14) = -DA1*BD(9) + DA2*DY1
153:      SDA(15) = -DA1*BD(10) + DA2*DY2
154:      SDA(16) = -DA1*BD(11) + DA2*DY3
155:      DD      = SQRT(SDA(14)**2+SDA(15)**2+SDA(16)**2)
156:      if ( DD.eq.0.0D0 ) then
157:        write(IMG,*)'XXX CANNOT FIX NORMAL VECTOR TO "HEX" ',IBODN,'.'
158:        call CNTERR( 'FATAL' )
159:        NERR    = NERR + 1
160:      else
161:        SDA(14) = SDA(14) /DD
162:        SDA(15) = SDA(15) /DD
163:        SDA(16) = SDA(16) /DD
164:      end if
165:      SDA(17) = BD(2)*SDA(14) + BD(3)*SDA(15) + BD(4)*SDA(16)
166:      &      - BD(8)*0.5
167:      SDA(18) = SDA(17) + BD(8)
168: C
169: C ..... BOTTOM & TOP SURFACE .....
170:      DZ1     = BD(5) /DL
171:      DZ2     = BD(6) /DL
172:      DZ3     = BD(7) /DL
173:      SDA(19) = 1.0 + 10000*6
174:      SDA(20) = DZ1
175:      SDA(21) = DZ2
176:      SDA(22) = DZ3
177:      SDA(23) = DZ1*BD(2) + DZ2*BD(3) + DZ3*BD(4)
178:      SDA(24) = SDA(23) + DL
179: C
180:      LSDAT   = LSDAT + 24*MWORD
181:      NS      = 4
182: C
183: C
184:      GRANGE(1) = 0.0D0
185:      GRANGE(2) = DL
186:      GRANGE(3) = -BD(8)/2.0D0
187:      GRANGE(4) = +BD(8)/2.0D0
188:      GRANGE(5) = -BD(8)/SQRT(3.0)
189:      GRANGE(6) = +BD(8)/SQRT(3.0)
190:      call GRCONV( GRANGE, BD(2),BD(3),BD(4),BD(5),BD(6),BD(7),
191:      &            BD(9),BD(10),BD(11),
192:      &            DINF )
193: C
194:      if ( NERR.gt.0 ) ICODE = 1
195:      return
```

196: end

src/shared/bodrrc.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODRCC( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'RCC' TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTRERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:   real*8 SDA(*), BD(*), DINF
16:   real*8 GRANGE(6)
17:   include 'INC/WORDL'
18:   include 'INC/_IOUNIT'
19: C
20:   real*8 HH
21: C
22: C-----
23: C
24:   ICODE = 0
25:   NERR = 0
26: C
27:   ICALL = 8
28:   call R8READ( ' ', BD, NA, ICALL, IERR )
29:   if ( NA.ne.ICALL ) then
30:     write(IMG,*) 'XXX Number of data for body "RCC" ', IBODN,
31:     & ' is not ', ICALL
32:     call CNTRERR( 'FATAL' )
33:   end if
34: C
35:   IBODN = INT(BD(1))
36:   IBSDA = (LSDAT-LSDA) /MWORD + 1
37:   HH = SQRT(BD(5)**2+BD(6)**2+BD(7)**2)
38: C
39: C .... IF HEIGHT OF CYLINDER (HH) >= DINF OR = 0.0,
40: C   CYLINDER IS INFINITE AND TOP & BOTTOM SIDE ARE UNNECESSARY
41: C
42:   JINF = 0
43:   if ( ABS(HH).lt.DINF.and.HH.ne.0.0 ) then
44: C
45: C .... SLAB : SDA2 *X + SDA3 *Y + SDA4 *Z - SDA5 (OR SDA6 ) = 0
46:   SDA(1) = 1.0 + 10000*6
47: C .. SDA(9) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
48:   SDA(2) = BD(5) /HH
49:   SDA(3) = BD(6) /HH
50:   SDA(4) = BD(7) /HH
51:   SDA(5) = BD(2)*SDA(2) + BD(3)*SDA(3) + BD(4)*SDA(4)
52:   SDA(6) = SDA(5) + HH
53:   NS = 2
54:   LSDAT = LSDAT + 14*MWORD
55:   else
56:     NS = 1
57:     LSDAT = LSDAT + 8*MWORD
58:     write(IPR,*) '<<MESSAGE>> "RCC" ', IBODN,
59:     & ' HAS INFINITE LENGTH.'
60:     call CNTRERR( 'MESSAGE' )
61:     JINF = 1
62:   end if
63: C
64: C .... SDA2,SDA3,SDA4 : CENTER OF BOTTOM SDA5,SDA6,SDA7 : DIRECTION
65: C   SDA8 : SQUARE OF RADIUS

```

```

66: C
67:   K = (NS-1)*6
68:   SDA(K+1) = 3.0 + 10000*8
69: C .. SDA(1) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
70:   SDA(K+2) = BD(2)
71:   SDA(K+3) = BD(3)
72:   SDA(K+4) = BD(4)
73:   SDA(K+5) = BD(5) /HH
74:   SDA(K+6) = BD(6) /HH
75:   SDA(K+7) = BD(7) /HH
76:   SDA(K+8) = BD(8)**2
77:   if ( BD(8).lt.0.0 ) then
78:     write(IMG, '(/lx,a,i5,a,e12.5,a/)') ' XXX "RCC" ', IBODN,
79:     & ' HAS A NEGATIVE RADIUS ', BD(8), '.'
80:     call CNTRERR( 'FATAL' )
81:     NERR = NERR + 1
82:   else if ( BD(8).eq.0.0 ) then
83:     write(IMG, '(/lx,a,i5,a/)') ' !!! "RCC" ', IBODN,
84:     & ' SHRINKS TO ZERO-VOLUME CYLINDER.'
85:     call CNTRERR( 'WARNING' )
86:     ICODE = -1
87:   end if
88:
89: C
90: C ... set GRANGE only for special cases ...
91: C
92: C ... axis in X-direction ...
93:   if( BD(6).eq.0.0d0 .and. BD(7).eq.0.0d0 ) then
94:     if( JINF.eq.0 ) then
95:       GRANGE(1) = MIN(GRANGE(1), MIN(BD(2), BD(2)+BD(5)) )
96:       GRANGE(2) = MAX(GRANGE(2), MAX(BD(2), BD(2)+BD(5)) )
97:     endif
98:     GRANGE(3) = MIN(GRANGE(3), BD(3) - BD(8) )
99:     GRANGE(4) = MAX(GRANGE(4), BD(3) + BD(8) )
100:    GRANGE(5) = MIN(GRANGE(5), BD(4) - BD(8) )
101:    GRANGE(6) = MAX(GRANGE(6), BD(4) + BD(8) )
102: C ... axis in Y-direction ...
103:   else if( BD(5).eq.0.0d0 .and. BD(7).eq.0.0d0 ) then
104:     if( JINF.eq.0 ) then
105:       GRANGE(3) = MIN(GRANGE(3), MIN(BD(3), BD(3)+BD(6)) )
106:       GRANGE(4) = MAX(GRANGE(4), MAX(BD(3), BD(3)+BD(6)) )
107:     endif
108:     GRANGE(5) = MIN(GRANGE(5), BD(4) - BD(8) )
109:     GRANGE(6) = MAX(GRANGE(6), BD(4) + BD(8) )
110:     GRANGE(1) = MIN(GRANGE(1), BD(2) - BD(8) )
111:     GRANGE(2) = MAX(GRANGE(2), BD(2) + BD(8) )
112: C ... axis in Z-direction ...
113:   else if( BD(5).eq.0.0d0 .and. BD(6).eq.0.0d0 ) then
114:     if( JINF.eq.0 ) then
115:       GRANGE(5) = MIN(GRANGE(5), MIN(BD(4), BD(4)+BD(7)) )
116:       GRANGE(6) = MAX(GRANGE(6), MAX(BD(4), BD(4)+BD(7)) )
117:     endif
118:     GRANGE(1) = MIN(GRANGE(1), BD(2) - BD(8) )
119:     GRANGE(2) = MAX(GRANGE(2), BD(2) + BD(8) )
120:     GRANGE(3) = MIN(GRANGE(3), BD(3) - BD(8) )
121:     GRANGE(4) = MAX(GRANGE(4), BD(3) + BD(8) )
122:   else
123:     if ( JINF.eq.0 ) then
124:       GRANGE(1) = 0.0d0
125:       GRANGE(2) = HH
126:       GRANGE(3) = -BD(8)
127:       GRANGE(4) = BD(8)
128:       GRANGE(5) = -BD(8)
129:       GRANGE(6) = BD(8)
130:       call GRCONV( GRANGE, BD(2),BD(3),BD(4), BD(5),BD(6),BD(7),

```

src/shared/bodrrcc.f

```
131:      &          0.0D0, 0.0D0, 0.0D0,  
132:      &          DINF  )  
133:      end if  
134:      endif  
135:  
136:      if ( NERR.gt.0 ) ICODE = 1  
137:  
138:      return  
139:      end
```

SAFE

src/shared/bodrc1.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODRCL( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &                LSDAT, DINF,  GRANGE,ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'RCL' TO SURFACES
6: C  (FOR USE OF A LATTICE OF TYPE 3 (HEXAGONAL LATTICE WITH CYLINDRICAL
7: C  ENCLOSING-FRAME))
8: C    Return code;
9: C    ICODE = 0 : no-problem
10: C    ICODE =-1 : suspicious but acceptable.
11: C    ICODE = 1 : fatal error.
12: C  CALLED IN: GEOMIN
13: C  CALLS: R8READ,CNTERR
14: C-----
15: C arguments : see description in BODBOX routine.
16: C=====
17:   real*8 SDA(*), BD(*)
18:   real*8 DINF
19:   real*8 GRANGE(6)
20:   include 'INC/_WORDL'
21:   include 'INC/_IOUNIT'
22: C
23: C ... local variable
24:   real*8 D1, DX, DY, DZ
25: C
26: C-----
27: C
28:   ICODE = 0
29:   NERR = 0
30: C
31: C
32: C  BD(2) ... BD(8) : SIMILAR TO 'RCC' BODY.
33: C  BD(9),BD(10),BD(11) : 'X-AXIS' VECTOR OF LOCAL COORDINATE SYSTEM
34: C  OF A TYPE-3 LATTICE. (MUST BE PERPENDICULAR TO
35: C  THE AXIS-VECTOR OF THE CYLINDER)
36: C
37:   ICALL = 11
38:   call R8READ( ' ', BD, NA, -ICALL, IERR )
39:   if ( NA.ne.ICALL ) then
40:     write(IMG,*) 'XXX Number of data for body "RCL" ', IBODN,
41:     & ' is not ',ICALL
42:     call CNTERR( 'FATAL' )
43:   end if
44: C
45:   IBODN = INT(BD(1))
46:   IBSDA = (LSDAT-LSDA) /MWORD + 1
47:   D1 = SQRT(BD(5)**2+BD(6)**2+BD(7)**2)
48: C
49: C .... SLAB : SDA2 *X + SDA3 *Y + SDA4 *Z - SDA5 (OR SDA6 ) = 0
50:   SDA(1) = 1.0 + 10000*6
51: C
52: C
53:   JINF = 0
54:   if ( D1.eq.0.0D0 ) then
55:     write(IMG,*) ' XXX Axis length of body "RCL" ', IBODN,
56:     & ' must not be zero.'
57:     call CNTERR( 'FATAL' )
58:     NERR = NERR + 1
59:     JINF = 1
60:   else if ( D1.ge.DINF ) then
61:     write(IMG,*) ' XXX Axis length of body "RCL" ', IBODN,
62:     & ' in infinite: ', D1
63:     NERR = NERR + 1
64:     call CNTERR( 'FATAL' )
65:     JINF = 1

```

```

66:   else
67:     SDA(2) = BD(5) /D1
68:     SDA(3) = BD(6) /D1
69:     SDA(4) = BD(7) /D1
70:     SDA(5) = BD(2)*SDA(2) + BD(3)*SDA(3) + BD(4)*SDA(4)
71:     SDA(6) = SDA(5) + D1
72:   end if
73: C
74: C .... SDA2,SDA3,SDA4 : CENTER OF BOTTOM SDA5,SDA6,SDA7 : DIRECTION
75: C    SDA8 : SQUARE OF RADIUS
76: C
77:   SDA(7) = 3.0 + 10000*8
78: C
79: C .. SDA(7) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
80: C
81:   SDA(8) = BD(2)
82:   SDA(9) = BD(3)
83:   SDA(10) = BD(4)
84:   if ( D1.ne.0.0 ) then
85:     SDA(11) = BD(5) /D1
86:     SDA(12) = BD(6) /D1
87:     SDA(13) = BD(7) /D1
88:   end if
89: C
90:   SDA(14) = BD(8)**2
91:   if ( BD(8).lt.0.0 ) then
92:     write(IMG,*) ' XXX Body "RCL" ', IBODN,
93:     & ' has a negative radius ', BD(8), ' .'
94:     call CNTERR( 'FATAL' )
95:     NERR = NERR + 1
96:   else if ( BD(8).eq.0.0 ) then
97:     write(IMG,*) ' !!! Body "RCL" ', IBODN,
98:     & ' shrinks to zero-volume cylinder.'
99:     call CNTERR( 'WARNING' )
100:     ICODE = -1
101:   end if
102: C
103: C ... set GRANGE only for special cases ...
104: C
105: C ... axis in X-direction ...
106:   if ( BD(6).eq.0.0D0.and.BD(7).eq.0.0D0 ) then
107:     if ( JINF.eq.0 ) then
108:       GRANGE(1) = MIN(GRANGE(1),MIN(BD(2),BD(2)+BD(5)))
109:       GRANGE(2) = MAX(GRANGE(2),MAX(BD(2),BD(2)+BD(5)))
110:     end if
111:     GRANGE(3) = MIN(GRANGE(3),BD(3)-BD(8))
112:     GRANGE(4) = MAX(GRANGE(4),BD(3)+BD(8))
113:     GRANGE(5) = MIN(GRANGE(5),BD(4)-BD(8))
114:     GRANGE(6) = MAX(GRANGE(6),BD(4)+BD(8))
115: C ... axis in Y-direction ...
116:   else if ( BD(5).eq.0.0D0.and.BD(7).eq.0.0D0 ) then
117:     if ( JINF.eq.0 ) then
118:       GRANGE(3) = MIN(GRANGE(3),MIN(BD(3),BD(3)+BD(6)))
119:       GRANGE(4) = MAX(GRANGE(4),MAX(BD(3),BD(3)+BD(6)))
120:     end if
121:     GRANGE(5) = MIN(GRANGE(5),BD(4)-BD(8))
122:     GRANGE(6) = MAX(GRANGE(6),BD(4)+BD(8))
123:     GRANGE(1) = MIN(GRANGE(1),BD(2)-BD(8))
124:     GRANGE(2) = MAX(GRANGE(2),BD(2)+BD(8))
125: C ... axis in Z-direction ...
126:   else if ( BD(5).eq.0.0D0.and.BD(6).eq.0.0D0 ) then
127:     if ( JINF.eq.0 ) then
128:       GRANGE(5) = MIN(GRANGE(5),MIN(BD(4),BD(4)+BD(7)))
129:       GRANGE(6) = MAX(GRANGE(6),MAX(BD(4),BD(4)+BD(7)))
130:     end if

```

src/shared/bodrc1.f

```
131:      GRANGE(1) = MIN(GRANGE(1),BD(2)-BD(8))
132:      GRANGE(2) = MAX(GRANGE(2),BD(2)+BD(8))
133:      GRANGE(3) = MIN(GRANGE(3),BD(3)-BD(8))
134:      GRANGE(4) = MAX(GRANGE(4),BD(3)+BD(8))
135:    else
136:      if ( JINF.eq.0 ) then
137:        GRANGE(1) = 0.0d0
138:        GRANGE(2) = D1
139:        GRANGE(3) = -BD(8)
140:        GRANGE(4) = BD(8)
141:        GRANGE(5) = -BD(8)
142:        GRANGE(6) = BD(8)
143:        call GRCONV( GRANGE, BD(2),BD(3),BD(4), BD(5),BD(6),BD(7),
144: & 0.0D0, 0.0D0, 0.0D0,
145: & DINF )
146:      end if
147:    end if
148: C
149: C   SDA15... SDA19 : A half space, but only its normal vector has
150: C   meanings in lattice geometry.
151: C   ('X-AXIS' vector in a local coordinate system
152: C   for the lattice).
153: C
154:   SDA(15) = 5.0 + 10000*5
155:   DX      = BD(9)
156:   DY      = BD(10)
157:   DZ      = BD(11)
158:   D1      = SQRT(DX**2+DY**2+DZ**2)
159:
160:   if ( D1.eq.0.0D0 ) then
161:     write(IMG,*) ' XXX Length of normal vector of body "RCL" ,
162: & IBODN, ' is zero.'
163:     call CNTERR( 'FATAL' )
164:     NERR      = NERR + 1
165:   else
166:     SDA(16) = DX/D1
167:     SDA(17) = DY/D1
168:     SDA(18) = DZ/D1
169:   end if
170:
171:   SDA(19) = SDA(16)*(BD(2)+2.0*SDA(16)*BD(8)) + SDA(17)*
172: & (BD(3)+2.0*SDA(17)*BD(8)) + SDA(18)*
173: & (BD(4)+2.0*SDA(18)*BD(8))
174:   LSDAT   = LSDAT + 19*MWORD
175:   NS      = 3
176:
177:   if ( NERR.gt.0 ) ICODE = 1
178:
179:   return
180: end
```


src/shared/bodrhpf.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine BODRHP( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:      &                  LSDAT, DINP,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE:  TRANSLATE BODY 'RHP' (RIGHT HEXAGONAL PRISM) TO SURFACES
6: C      Return code;
7: C      ICODE = 0 : no-problem
8: C      ICODE =-1 : suspicious but acceptable.
9: C      ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS:    R8READ,CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:      implicit real*8(D)
16:      real*8 GRANGE(6)
17:      real*8 SDA(*), BD(*)
18:      include 'INC/_WORDL'
19:      include 'INC/_IUNIT'
20: C-----
21:      ICODE = 0
22:      NERR = 0
23: C
24: C ..... ICALL: NUMBER OF BODY DATA + 1
25:      ICALL = 6
26:      call R8READ( ' ', BD, NA, -ICALL, IERR )
27:      if ( NA.ne.ICALL ) then
28:        write(IMG,*) 'XXX Number of data for body "RHP" ', IBODN,
29:        &          ' is not ', ICALL
30:        call CNTERR( 'FATAL' )
31:      end if
32: C
33:      IBODN = INT(BD(1))
34:      IBSDA = (LSDAT-LSDA) /MWORD + 1
35: C
36: C ..... BD2,BD3,BD4 : CENTER OF BASE   BD5: HEIGHT   BD6: WIDTH
37: C
38:      if ( BD(6).lt.0.0 ) then
39:        write(IMG,*) ' XXX "RHP" ', IBODN, ' HAS A NEGATIVE WIDTH ',
40:        &          BD(6), '.'
41:        call CNTERR( 'FATAL' )
42:        NERR = NERR + 1
43:      else if ( BD(6).eq.0.0 ) then
44:        write(IMG,*) ' !!! "RHP" ', IBODN,
45:        &          ' SHRINKS TO ZERO-VOLUME HEXAGONAL PRISM.'
46:        call CNTERR( 'WARNING' )
47:        ICODE = -1
48:      end if
49: C
50: C ..... HEXAGONAL SIDE ARE MADE OF THREE SLABS
51: C
52: C >>>>>>> SIDE 1 (PERPENDICULAR TO X-AXIS)
53:      SDA(1) = 1.0 + 10000*6
54:      SDA(2) = 1.0
55:      SDA(3) = 0.0
56:      SDA(4) = 0.0
57:      SDA(5) = BD(2) - BD(6)*0.5
58:      SDA(6) = BD(2) + BD(6)*0.5
59: C >>>>>>> SIDE 2 ( 60 DEGREE ROTATION OF SIDE 1)
60:      SDA(7) = 1.0 + 10000*6
61:      SDA(8) = 0.5
62:      SDA(9) = SQRT(3.0D0)*0.5
63:      SDA(10) = 0.0
64:      SDA(11) = BD(2)*SDA(8) + BD(3)*SDA(9) - BD(6)*0.5
65:      SDA(12) = SDA(11) + BD(6)

```

```

66: C >>>>>>> SIDE 3 (120 DEGREE ROTATION OF SIDE 1)
67:      SDA(13) = 1.0 + 10000*6
68:      SDA(14) = -SDA(8)
69:      SDA(15) = SDA(9)
70:      SDA(16) = 0.0
71:      SDA(17) = BD(2)*SDA(14) + BD(3)*SDA(15) - BD(6)*0.5
72:      SDA(18) = SDA(17) + BD(6)
73:      LSDAT = LSDAT + 18*MWORD
74:      NS = 3
75:
76:      GRANGE(1) = MIN(GRANGE(1), BD(2) - 0.5*BD(6) )
77:      GRANGE(2) = MAX(GRANGE(2), BD(2) + 0.5*BD(6) )
78:      GRANGE(3) = MIN(GRANGE(3), BD(2) - 1D0/SQRT(3.0D0)* BD(6) )
79:      GRANGE(4) = MAX(GRANGE(4), BD(2) + 1D0/SQRT(3.0D0)* BD(6) )
80: C
81: C ..... IF HEIGHT IS 0.0 OR GREATER THAN DINP , HEIGHT IS INFINITE
82: C
83:      if ( ABS(BD(5)).lt.DINF.and.BD(5).ne.0.0 ) then
84:        SDA(19) = 1.0 + 10000*6
85:        SDA(20) = 0.0
86:        SDA(21) = 0.0
87:        SDA(22) = 1.0
88:        SDA(23) = BD(4)
89:        SDA(24) = BD(4) + BD(5)
90:        LSDAT = LSDAT + 6*MWORD
91:        NS = 4
92:
93:        GRANGE(5) = MIN(GRANGE(5),MIN(SDA(23),SDA(24)))
94:        GRANGE(6) = MAX(GRANGE(6),MAX(SDA(23),SDA(24)))
95:
96:      else
97:        write(IPR,*) '<<MESSAGE>> "RHP" ', IBODN,
98:        &          ' HAS INFINITE LENGTH.'
99:        ICODE = -1
100:        call CNTERR( 'MESSAGE' )
101:      end if
102:
103:      if ( NERR.ne.0 ) ICODE = 1
104:      return
105: end

```

src/shared/bodrpp.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODRPP( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:     &               LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C   PURPOSE: TRANSLATE BODY 'RPP' TO SURFACES
6: C     Return code;
7: C     ICODE = 0 : no-problem
8: C     ICODE =-1 : suspicious but acceptable.
9: C     ICODE = 1 : fatal error.
10: C   CALLED IN: GEOMIN
11: C   CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C
15: C o GRANGE(6) : guess of X,Y,Z range of geometry(to be used in picture?)
16: C     GRANGE(1:2) : min-max of X range
17: C     GRANGE(3:4) : min-max of Y range
18: C     GRANGE(5:6) : min-max of Z range
19: C-----
20:   real*8 SDA(*), BD(*), DINF
21:   real*8 GRANGE(6)
22:   include 'INC/_WORDL'
23:   include 'INC/_IUNIT'
24: c
25:   ICODE = 0
26: c
27:   ICALL = 7
28:   call R8READ( ' ', BD, NA, ICALL, IERR )
29:   if ( NA.ne.ICALL ) then
30:     write(IMG,*) 'XXX Number of data for body "RPP" ', IBODN,
31:   &   ' is not ', ICALL
32:     call CNTERR( 'FATAL' )
33:   end if
34: C
35:   IBODN = INT(BD(1))
36:   IBSDA = (LSDAT-LSDA) /MWORD + 1
37:   NS = 0
38: C .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
39: C   IF RANGE OF A CERTAIN COORDINATE IS ZERO WIDTH,
40:   IB = 0
41:   if ( BD(2).ne.BD(3).and.ABS(BD(3)-BD(2)).lt.DINF ) then
42:     SDA(IB+1) = 1.0 + 10000*6
43:     SDA(IB+2) = 1.0
44:     SDA(IB+3) = 0.0
45:     SDA(IB+4) = 0.0
46:     SDA(IB+5) = MIN(BD(2),BD(3))
47:     SDA(IB+6) = MAX(BD(2),BD(3))
48:
49:     GRANGE(1) = MIN( GRANGE(1), SDA(IB+5) )
50:     GRANGE(2) = MAX( GRANGE(2), SDA(IB+6) )
51:
52:     IB = IB + 6
53:     NS = NS + 1
54:   else
55:     write(IPR,*) '<<MESSAGE>> "RPP" ', IBODN,
56:   &   ' IS BOUNDLESS IN X-DIRECTION.'
57:     call CNTERR( 'MESSAGE' )
58:     ICODE = -1
59:   end if
60: C .. SDA(7) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
61:   if ( BD(4).ne.BD(5).and.ABS(BD(5)-BD(4)).lt.DINF ) then
62:     SDA(IB+1) = 1.0 + 10000*6
63:     SDA(IB+2) = 0.0
64:     SDA(IB+3) = 1.0
65:     SDA(IB+4) = 0.0

```

```

66:     SDA(IB+5) = MIN(BD(4),BD(5))
67:     SDA(IB+6) = MAX(BD(4),BD(5))
68:
69:     GRANGE(3) = MIN( GRANGE(3), SDA(IB+5) )
70:     GRANGE(4) = MAX( GRANGE(4), SDA(IB+6) )
71:
72:     IB = IB + 6
73:     NS = NS + 1
74:   else
75:     write(IPR,*) '<<MESSAGE>> "RPP" ', IBODN,
76:   &   ' IS BOUNDLESS IN Y-DIRECTION.'
77:     call CNTERR( 'MESSAGE' )
78:     ICODE = -1
79:   end if
80: C .. SDA(13) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
81:   if ( BD(6).ne.BD(7).and.ABS(BD(7)-BD(6)).lt.DINF ) then
82:     SDA(IB+1) = 1.0 + 10000*6
83:     SDA(IB+2) = 0.0
84:     SDA(IB+3) = 0.0
85:     SDA(IB+4) = 1.0
86:     SDA(IB+5) = MIN(BD(6),BD(7))
87:     SDA(IB+6) = MAX(BD(6),BD(7))
88:
89:     GRANGE(5) = MIN( GRANGE(5), SDA(IB+5) )
90:     GRANGE(6) = MAX( GRANGE(6), SDA(IB+6) )
91:
92:     IB = IB + 6
93:     NS = NS + 1
94:   else
95:     write(IPR,*) '<<MESSAGE>> "RPP" ', IBODN,
96:   &   ' IS BOUNDLESS IN Z-DIRECTION.'
97:     call CNTERR( 'MESSAGE' )
98:     ICODE = -1
99:   end if
100:
101:   LSDAT = LSDAT + IB*MWORD
102:   return
103: end

```

src/shared/bodsph.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine BODSPH( ICALL, SDA,  LSDA, BD,   IBODN, IBSDA, NS,
3:     &               LSDAT, GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'SPH' TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C CALLED IN: GEOMIN
11: C CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15:   real*8 SDA(*), BD(*)
16:   real*8 GRANGE(6)
17:   include 'INC/WORDL'
18:   include 'INC/_IUNIT'
19: c
20:   ICODE = 0
21:   NERR = 0
22: c
23:   ICALL = 5
24:   call R8READ( ' ', BD, NA, -ICALL, IERR )
25:   if ( NA.ne.ICALL ) then
26:     write(IMG,*) 'XXX: Number of data for body "SPH" ', IBODN,
27: & ' is not ', ICALL
28:     call CNTERR( 'FATAL' )
29:   end if
30: C
31:   IBODN = INT(BD(1))
32:   IBSDA = (LSDAT-LSDA) /MWORD + 1
33:   NS = 1
34: C .. SDA(1) IS DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
35:   SDA(1) = 2.0 + 10000*5
36:   SDA(2) = BD(2)
37:   SDA(3) = BD(3)
38:   SDA(4) = BD(4)
39:   SDA(5) = BD(5)**2
40:   if ( BD(5).lt.0.0 ) then
41:     write(IMG,*) ' XXX "SPH" ', IBODN, ' HAS A NEGATIVE RADIUS. ',
42: & BD(5), '.'
43:     call CNTERR( 'FATAL' )
44:     NERR = NERR + 1
45:   else if ( BD(5).eq.0.0 ) then
46:     write(IMG,*) ' !!! "SPH" ', IBODN,
47: & ' SHRINKS TO ZERO-RADIUS SPHERE.'
48:     call CNTERR( 'WARNING' )
49:     ICODE = -1
50:   end if
51:   LSDAT = LSDAT + 5*MWORD
52:
53:   GRANGE(1) = min( GRANGE(1), BD(2) - BD(5) )
54:   GRANGE(2) = max( GRANGE(2), BD(2) + BD(5) )
55:   GRANGE(3) = min( GRANGE(3), BD(3) - BD(5) )
56:   GRANGE(4) = max( GRANGE(4), BD(3) + BD(5) )
57:   GRANGE(5) = min( GRANGE(5), BD(4) - BD(5) )
58:   GRANGE(6) = max( GRANGE(6), BD(4) + BD(5) )
59:
60:   if ( NERR.gt.0 ) ICODE = 1
61:
62:   return
63: end
```

src/shared/bodtec.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine BODTEC( ICALL, SDA,   LSDA, BD,   IBODN, IBSDA, NS,
3:      &                  LSDAT, DINF,  GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'TEC' (TRUNCATED ELLIPTIC CONE)TO SURFACES
6: C      Return code;
7: C      ICODE = 0 : no-problem
8: C      ICODE =-1 : suspicious but acceptable.
9: C      ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15: C      implicit real*8(D)
16: C      real*8 SDA(*), BD(*), DINF
17: C      real*8 GRANGE(6)
18: C      include 'INC/_IOUNIT'
19: C      include 'INC/_WORDL'
20: C-----
21: C      ICODE = 0
22: C      NERR = 0
23: C
24: C      ICALL = 14
25: C      call R8READ( ' ', BD, NA, ICALL, IERR )
26: C      if ( NA.ne.ICALL ) then
27: C         write(MSG,*) 'XXX Number of data for body "TEC" is', IBODN,
28: C         & ' is not ',ICALL
29: C         call CNTERR( 'FATAL' )
30: C      end if
31: C      IBODN = INT(BD(1))
32: C      IBSDA = (LSDAT-LSDA) /MWORD + 1
33: C
34: C .... BD2,BD3,BD4 : CENTER OF BOTTOM BD5,BD6,BD7 : HEIGHT VECTOR
35: C      BD8,BD9,BD10: FIRST SEMI-AXIS  BD11,BD12,BD13: SECOND SEMI-AXIS
36: C      FIRST AND SECOND SEMI-AXES MUST BE PERPENDICULAR TO EACH OTHER
37: C      BOD14 : RATIO P OF THE BOTTOM TO TOP ELLIPSE
38: C
39: C .... SLAB : SDA2 *X + SDA3 *Y + SDA4 *Z - SDA5 (OR SDA6) = 0
40: C      SDA(1) = 1.0 + 10000*6
41: C .. SDA(1) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
42: C      (D1X,D1Y,D1Z):NORMAL VECTOR TO BOTTOM SURFACE
43: C      D1X = BD(9)*BD(13) - BD(10)*BD(12)
44: C      D1Y = BD(10)*BD(11) - BD(8)*BD(13)
45: C      D1Z = BD(8)*BD(12) - BD(9)*BD(11)
46: C      DD = SQRT(D1X**2+D1Y**2+D1Z**2)
47: C      SDA(2) = D1X/DD
48: C      SDA(3) = D1Y/DD
49: C      SDA(4) = D1Z/DD
50: C      DHN = BD(5)*SDA(2) + BD(6)*SDA(3) + BD(7)*SDA(4)
51: C      SDA(5) = BD(2)*SDA(2) + BD(3)*SDA(3) + BD(4)*SDA(4)
52: C      SDA(6) = SDA(5) + DHN
53: C      NS = 1
54: C
55: C >>>> CONE : GENERAL QUADRATIC SURFACE
56: C      SDA(7) = 4.0 + 10000*11
57: C .. SDA(7) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
58: C ..... DA2 : LENGTH OF SEMI-AXIS 1(A) **2
59: C ..... DB2 : LENGTH OF SEMI-AXIS 2(B) **2
60: C ..... DH2 : LENGTH OF HEIGHT VECTOR(H)**2
61: C      DA2 = BD(8)**2 + BD(9)**2 + BD(10)**2
62: C      DB2 = BD(11)**2 + BD(12)**2 + BD(13)**2
63: C      DH2 = BD(5)**2 + BD(6)**2 + BD(7)**2
64: C      DH = SQRT(DH2)
65: C ..... (DHX,DHY,DHZ) : UNIT VECTOR OF HEIGHT DIRECTION

```

```

66:      DHX = BD(5)
67:      DHY = BD(6)
68:      DHZ = BD(7)
69:      DNX = SDA(2) /DHN
70:      DNY = SDA(3) /DHN
71:      DNZ = SDA(4) /DHN
72: C
73:      DAX = (DNX*DHX-1.D0)*BD(8) + DNX*DHY*BD(9) + DHZ*DNX*BD(10)
74:      DBX = (DNX*DHX-1.D0)*BD(11) + DNX*DHY*BD(12) + DHZ*DNX*BD(13)
75:      DAY = DHX*DNY*BD(8) + (DNY*DHY-1.D0)*BD(9) + DNY*DHZ*BD(10)
76:      DBY = DHX*DNY*BD(11) + (DNY*DHY-1.D0)*BD(12) + DNY*DHZ*BD(13)
77:      DAZ = DNZ*DHX*BD(8) + DHY*DNZ*BD(9) + (DNZ*DHZ-1.D0)*BD(10)
78:      DBZ = DNZ*DHX*BD(11) + DHY*DNZ*BD(12) + (DNZ*DHZ-1.D0)*BD(13)
79:      DAD = DAX*BD(2) + DAY*BD(3) + DAZ*BD(4)
80:      DBD = DBX*BD(2) + DBY*BD(3) + DBZ*BD(4)
81:      DVD = DNX*BD(2) + DNY*BD(3) + DNZ*BD(4)
82:      DT = (BD(14)-1.D0)
83: C
84: C << EQUATION >>
85: C
86:      SDA(8) = (DAX/DA2)**2 + (DBX/DB2)**2 - (DT*DNX)**2
87:      SDA(9) = (DAY/DA2)**2 + (DBY/DB2)**2 - (DT*DNY)**2
88:      SDA(10) = (DAZ/DA2)**2 + (DBZ/DB2)**2 - (DT*DNZ)**2
89:      SDA(11) = 2.0*(DAX*DAY/DA2/DA2+DBX*DBY/DB2/DB2-DT*DT*DNX*DNY)
90:      SDA(12) = 2.0*(DAY*DAZ/DA2/DA2+DBY*DBZ/DB2/DB2-DT*DT*DNY*DNZ)
91:      SDA(13) = 2.0*(DAZ*DAX/DA2/DA2+DBZ*DBX/DB2/DB2-DT*DT*DNZ*DNX)
92:      DTV1 = DT*DVD - 1.D0
93:      SDA(14) = 2.0*(DT*DTV1*DNX-DAD*DAX/DA2/DA2-DBD*DBX/DB2/DB2)
94:      SDA(15) = 2.0*(DT*DTV1*DNY-DAD*DAY/DA2/DA2-DBD*DBY/DB2/DB2)
95:      SDA(16) = 2.0*(DT*DTV1*DNZ-DAD*DAZ/DA2/DA2-DBD*DBZ/DB2/DB2)
96:      SDA(17) = (DAD/DA2)**2 + (DBD/DB2)**2 - DTV1**2
97: C
98:      NS = 2
99:      LSDAT = LSDAT + 17*MWORD
100: C
101: C .... BD2,BD3,BD4 : CENTER OF BOTTOM BD5,BD6,BD7 : HEIGHT VECTOR
102: C      BD8,BD9,BD10: FIRST SEMI-AXIS  BD11,BD12,BD13: SECOND SEMI-AXIS
103: C      FIRST AND SECOND SEMI-AXES MUST BE PERPENDICULAR TO EACH OTHER
104: C      BOD14 : RATIO P OF THE BOTTOM TO TOP ELLIPSE
105: C
106: C      GRANGE(1) = 0.0d0
107: C      GRANGE(2) = DHN
108: C      D2X = BD(5) - SDA(2)*DHN
109: C      D2Y = BD(6) - SDA(3)*DHN
110: C      D2Z = BD(7) - SDA(4)*DHN
111: C      DDD = SQRT(D2X*D2X+D2Y*D2Y+D2Z*D2Z)
112: C      DM = SQRT(MAX(BD(8)**2+BD(9)**2+BD(10)**2,
113: C      & BD(11)**2+BD(12)**2+BD(13)**2))
114: C      GRANGE(3) = -DM
115: C      GRANGE(4) = DDD+DM
116: C      GRANGE(5) = -DM
117: C      GRANGE(6) = DM
118: C      call GRCONV(GRANGE, BD(2),BD(3),BD(4), SDA(2),SDA(3),SDA(4),
119: C      & D2X,D2Y,D2Z,
120: C      & DINF )
121: C
122:      return
123:      end

```

src/shared/bodtrc.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODTRC( ICALL, SDA, LSDA, BD, IBODN, IBSDA, NS,
3:     & LSDAT, DINF, GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'TRC' (TRUNCATED RIGHT ANGLE CONE) TO SURFACES
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE = -1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTRR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C=====
15: C    implicit real*8(D)
16: C    real*8 SDA(*), BD(*), DINF
17: C    real*8 GRANGE(6)
18: C    include 'INC/_WORDL'
19: C    include 'INC/_IOUNIT'
20: C-----
21: C    ICODE = 0
22: C    NERR = 0
23: C
24: C    ICALL = 9
25: C    call R8READ( ' ', BD, NA, ICALL, IERR )
26: C    if ( NA.ne.ICALL ) then
27: C      write(IMG,*) 'XXX Number of data for body "TRC" ', IBODN,
28: C      & ' is not ', ICALL
29: C      call CNTRR( 'FATAL' )
30: C    end if
31: C
32: C    IBODN = INT(BD(1))
33: C    IBSDA = (LSDAT-LSDA) /MWORD + 1
34: C
35: C    ... BD2,BD3,BD4 : CENTER OF BOTTOM BD5,BD6,BD7 : DIRECTION
36: C    BD8 : LOWER BASE RADIUS BD9: UPPER BASE RADIUS
37: C
38: C    ... SLAB : SDA2 *X + SDA3 *Y + SDA4 *Z - SDA5 (OR SDA6) = 0
39: C
40: C    SDA(1) = 1.0 + 10000*6
41: C
42: C .. SDA(1) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
43: C
44: C    DH2 = BD(5)**2 + BD(6)**2 + BD(7)**2
45: C    DH = SQRT(DH2)
46: C    if ( DH.eq.0.0D0 ) then
47: C      write(IMG,*) ' XXX LENGTH OF DIRECTION VECTOR OF "TRC" ',
48: C      & IBODN, ' IS ZERO.'
49: C      call CNTRR( 'FATAL' )
50: C      NERR = NERR + 1
51: C      DH = 1.0D0
52: C    else
53: C      SDA(2) = BD(5) /DH
54: C      SDA(3) = BD(6) /DH
55: C      SDA(4) = BD(7) /DH
56: C      SDA(5) = BD(2)*SDA(2) + BD(3)*SDA(3) + BD(4)*SDA(4)
57: C      SDA(6) = SDA(5) + DH
58: C    end if
59: C    NS = 1
60: C
61: C    if ( BD(8).ne.BD(9) ) then
62: C
63: C >>>> CONE : GENERAL QUADRATIC SURFACE
64: C    SDA(7) = 4.0 + 10000*11
65: C .. SDA(7) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')

```

```

66: C    .... DRR : COTANGENT OF TAPERED ANGLE
67: C    DRR = BD(8) / (BD(8)-BD(9))
68: C    .... (DPX,DPY,DPZ) : HEAD POINT OF CONE
69: C    DPX = BD(2) + DRR*BD(5)
70: C    DPY = BD(3) + DRR*BD(6)
71: C    DPZ = BD(4) + DRR*BD(7)
72: C
73: C << EQUATION >>
74: C (X-DPX)**2 + (Y-DPY)**2 + (Z-DPZ)**2 -
75: C (1+(1/DRR)**2)*{(X-DPX)*SDA(2)+(Y-DPY)*SDA(3)+(Z-DPZ)*SDA(4)}**2 = 0
76: C
77: C    DLM = 1.0D0 + (BD(8)-BD(9))**2 / (BD(5)**2+BD(6)**2+BD(7)**2)
78: C    DABC = SDA(2)*DPX + SDA(3)*DPY + SDA(4)*DPZ
79: C    SDA(8) = 1.0D0 - DLM*SDA(2)**2
80: C    SDA(9) = 1.0D0 - DLM*SDA(3)**2
81: C    SDA(10) = 1.0D0 - DLM*SDA(4)**2
82: C    SDA(11) = -2.0D0*DLM*SDA(2)*SDA(3)
83: C    SDA(12) = -2.0D0*DLM*SDA(3)*SDA(4)
84: C    SDA(13) = -2.0D0*DLM*SDA(4)*SDA(2)
85: C    SDA(14) = 2.0D0*(-DPX+DLM*SDA(2)*DABC)
86: C    SDA(15) = 2.0D0*(-DPY+DLM*SDA(3)*DABC)
87: C    SDA(16) = 2.0D0*(-DPZ+DLM*SDA(4)*DABC)
88: C    SDA(17) = DPX*DPX + DPY*DPY + DPZ*DPZ - DLM*DABC*DABC
89: C    INN = 17
90: C
91: C ... Both of the bases have the same radius.
92: C
93: C    else
94: C      write(IMG,*) '!!! TOP & BOTTOM OF "TRC" ', IBODN,
95: C      & ' HAVE THE SAME RADIUS.',
96: C      & ' DEGENERATE INTO A CYLINDER.'
97: C      call CNTRR( 'WARNING' )
98: C      ICODE = -1
99: C
100: C    SDA(7) = 3.0 + 10000*8
101: C .. SDA(7) IS A DATA FOR SURFACE RECOGNITION. (SEE COMMENT IN 'BODBOX')
102: C    SDA(8) = BD(2)
103: C    SDA(9) = BD(3)
104: C    SDA(10) = BD(4)
105: C    SDA(11) = BD(5) /DH
106: C    SDA(12) = BD(6) /DH
107: C    SDA(13) = BD(7) /DH
108: C    SDA(14) = BD(8)**2
109: C    INN = 14
110: C    if ( BD(8).lt.0.0D0 ) then
111: C      write(IMG,*) ' XXX "TRC" ', IBODN,
112: C      & ' HAS A NEGATIVE RADIUS ', BD(8), '.'
113: C      call CNTRR( 'FATAL' )
114: C      NERR = NERR + 1
115: C    else if ( BD(8).eq.0.0D0 ) then
116: C      write(IMG,*) ' !!! "TRC" ', IBODN,
117: C      & ' SHRINKS TO ZERO-VOLUME CYLINDER.'
118: C      call CNTRR( 'WARNING' )
119: C      ICODE = -1
120: C    end if
121: C  end if
122: C
123: C    NS = 2
124: C    LSDAT = LSDAT + INN*MWORD
125: C
126: C    GRANGE(1) = 0
127: C    GRANGE(2) = DH
128: C    GRANGE(3) = -max(BD(8),BD(9))
129: C    GRANGE(4) = -GRANGE(3)
130: C    GRANGE(5) = GRANGE(3)

```

src/shared/bodtrc.f

```
131:      GRANGE(6) = GRANGE(4)
132:      call GRCONV(GRANGE,BD(2),BD(3),BD(4), BD(5),BD(6),BD(7),
133:      &          0.0D0, 0.0D0, 0.0D0,
134:      &          DINF  )
135: c
136:      if ( NERR.gt.0 ) ICODE = 1
137:      return
138:      end
```

SAE

src/shared/bodwed.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine BODWED( ICALL, SDA, LSDA, BD, IBODN, IBSDA, NS,
3:     & LSDAT, DINF, GRANGE, ICODE )
4: C=====
5: C  PURPOSE: TRANSLATE BODY 'WED' (RIGHT OR NON-RIGHT ANGLE WEDGE)
6: C    Return code;
7: C    ICODE = 0 : no-problem
8: C    ICODE =-1 : suspicious but acceptable.
9: C    ICODE = 1 : fatal error.
10: C  CALLED IN: GEOMIN
11: C  CALLS: R8READ, CNTERR
12: C-----
13: C arguments : see description in BODBOX routine.
14: C-----
15: C    implicit real*8(D)
16: C    real*8 SDA(*), BD(*), DINF
17: C    real*8 GRANGE(6)
18: C    include 'INC/_WORDL'
19: C    include 'INC/_IOUNIT'
20: C-----
21: C    ICODE = 0
22: C    NERR = 0
23: C
24: C ..... ICALL: NUMBER OF BODY DATA + 1
25: C    ICALL = 13
26: C    call R8READ( ' ', BD, NA, -ICALL, IERR )
27: C    if ( NA.ne.ICALL ) then
28: C      write(IMG,*) 'XXX Number of data for body "WED" ', IBODN,
29: C      & ' is not ', ICALL
30: C      call CNTERR( 'FATAL' )
31: C    end if
32: C
33: C    IBODN = INT(BD(1))
34: C    IBSDA = (LSDAT-LSDA) /MWORD + 1
35: C
36: C  BD2,BD3,BD4 : CORNER POINT      BD5,BD6,BD7 : EDGE 1
37: C  BD8,BD9,BD10 : EDGE 2          BD11,BD12,BD13 : EDGE 3
38: C
39: C ..... NS : NUMBER OF SURFACE .....
40: C    NS = 4
41: C
42: C >>>>  BOTTOM & TOP SURFACE      (SLAB)
43: C
44: C    SDA(1) = 1.0 + 10000*6
45: C    D1 = BD(6)*BD(10) - BD(7)*BD(9)
46: C    D2 = BD(7)*BD(8) - BD(5)*BD(10)
47: C    D3 = BD(5)*BD(9) - BD(6)*BD(8)
48: C    DD = SQRT(D1*D1+D2*D2+D3*D3)
49: C
50: C    if ( DD.eq.0.0D0 ) then
51: C      write(IMG,*) ' XXX Edge 1 & 2 of "WED" ', IBODN,
52: C      & ' cannot compose a surface.'
53: C      call CNTERR( 'FATAL' )
54: C      NERR = NERR + 1
55: C    else
56: C      if ( D1*BD(11)+D2*BD(12)+D3*BD(13).lt.0.0 ) then
57: C        D1 = -D1
58: C        D2 = -D2
59: C        D3 = -D3
60: C      end if
61: C      SDA(2) = D1/DD
62: C      SDA(3) = D2/DD
63: C      SDA(4) = D3/DD
64: C      SDA(5) = SDA(2)*BD(2) + SDA(3)*BD(3) + SDA(4)*BD(4)
65: C      SDA(6) = SDA(2)*BD(11) + SDA(3)*BD(12) + SDA(4)*BD(13) +

```

```

66: C      & SDA(5)
67: C    end if
68: C
69: C >>>  SURFACE BY  EDGE 1  AND  EDGE 3
70: C
71: C    SDA(7) = 5.0 + 10000*5
72: C    D1 = BD(6)*BD(13) - BD(7)*BD(12)
73: C    D2 = BD(7)*BD(11) - BD(5)*BD(13)
74: C    D3 = BD(5)*BD(12) - BD(6)*BD(11)
75: C    DD = SQRT(D1*D1+D2*D2+D3*D3)
76: C
77: C    if ( DD.eq.0.0D0 ) then
78: C      write(IMG,*) ' XXX Edge 1 & 3 of "WED" ', IBODN,
79: C      & ' cannot compose a surface.'
80: C      call CNTERR( 'FATAL' )
81: C      NERR = NERR + 1
82: C    else
83: C      if ( D1*BD(8)+D2*BD(9)+D3*BD(10).gt.0.0 ) then
84: C        D1 = -D1
85: C        D2 = -D2
86: C        D3 = -D3
87: C      end if
88: C      SDA(8) = D1/DD
89: C      SDA(9) = D2/DD
90: C      SDA(10) = D3/DD
91: C      SDA(11) = SDA(8)*BD(2) + SDA(9)*BD(3) + SDA(10)*BD(4)
92: C    end if
93: C
94: C >>>  SURFACE BY  EDGE 2  AND  EDGE 3
95: C
96: C    SDA(12) = 5.0 + 10000*5
97: C    D1 = BD(9)*BD(13) - BD(10)*BD(12)
98: C    D2 = BD(10)*BD(11) - BD(8)*BD(13)
99: C    D3 = BD(8)*BD(12) - BD(9)*BD(11)
100: C    DD = SQRT(D1*D1+D2*D2+D3*D3)
101: C
102: C    if ( DD.eq.0.0D0 ) then
103: C      write(IMG,*) ' XXX Edge 2 & 3 of "WED" ', IBODN,
104: C      & ' cannot compose a surface.'
105: C      call CNTERR( 'FATAL' )
106: C      NERR = NERR + 1
107: C    else
108: C      if ( D1*BD(5)+D2*BD(6)+D3*BD(7).gt.0.0 ) then
109: C        D1 = -D1
110: C        D2 = -D2
111: C        D3 = -D3
112: C      end if
113: C      SDA(13) = D1/DD
114: C      SDA(14) = D2/DD
115: C      SDA(15) = D3/DD
116: C      SDA(16) = SDA(13)*BD(2) + SDA(14)*BD(3) + SDA(15)*BD(4)
117: C    end if
118: C
119: C >>>  LAST SURFACE
120: C
121: C    SDA(17) = 5.0 + 10000*5
122: C    DA = BD(5) - BD(8)
123: C    DB = BD(6) - BD(9)
124: C    DC = BD(7) - BD(10)
125: C    D1 = DB*BD(13) - DC*BD(12)
126: C    D2 = DC*BD(11) - DA*BD(13)
127: C    D3 = DA*BD(12) - DB*BD(11)
128: C    DD = SQRT(D1*D1+D2*D2+D3*D3)
129: C
130: C    if ( DD.eq.0.0D0 ) then

```

src/shared/bodwed.f

```
131:      write(IMG,*) ' XXX "WED" ', IBODN,
132:      &      ' cannot compose a surface parallel to the edge 3.'
133:      call CNTERR( 'FATAL' )
134:      NERR      = NERR + 1
135:    else
136:      if ( D1*(BD(5)+BD(8))+D2*(BD(6)+BD(9))+D3*(BD(7)+BD(10)).lt.0.0
137:      &      ) then
138:        D1      = -D1
139:        D2      = -D2
140:        D3      = -D3
141:      end if
142:      SDA(18) = D1/DD
143:      SDA(19) = D2/DD
144:      SDA(20) = D3/DD
145:      SDA(21) = SDA(18)*(BD(2)+BD(5)) + SDA(19)*(BD(3)+BD(6))
146:      &      + SDA(20)*(BD(4)+BD(7))
147:    end if
148: CCC
149:    LSDAT      = LSDAT + 21*MWORD
150:  C
151:    do 120 M=0,7
152:      if ( M.eq.3 .or. M.eq.7 ) goto 120
153:      I = mod(M,2)
154:      J = mod(M/2,2)
155:      K = mod(M/4,2)
156:  C
157:      DOX = BD(2) + I*BD(5) + J*BD(8) + K*BD(11)
158:      DOY = BD(3) + I*BD(6) + J*BD(9) + K*BD(12)
159:      DOZ = BD(4) + I*BD(7) + J*BD(10) + K*BD(13)
160:  C
161:      GRANGE(1) = MIN(GRANGE(1),DOX)
162:      GRANGE(2) = MAX(GRANGE(2),DOX)
163:      GRANGE(3) = MIN(GRANGE(3),DOY)
164:      GRANGE(4) = MAX(GRANGE(4),DOY)
165:      GRANGE(5) = MIN(GRANGE(5),DOZ)
166:      GRANGE(6) = MAX(GRANGE(6),DOZ)
167: 120 continue
168:  C
169:      if ( NERR.gt.0 ) ICODE = 1
170:      return
171:    end
```


src/shared/bsdec1.f

```

1: C
2: C   JAERI MONTE CARLO CODE UTILITY
3: C
4: C=====
5: C   BSDEC1, BSDEC2, BSDEC3, BSDEC4
6: C=====
7: C PURPOSE: BINARY SEARCH EACH PARTICLE USES DIFFERENT TABLES (BUT OF
8: C   THE SAME LENGTH) OF DECREASING ORDER.
9: C CALLED IN: (VAREIOUS ROUTINES)
10: C
11: C -----
12: C   BSDEC1 & BSDEC2 : COMMON TABLE LENGTH
13: C -----
14: C
15: C -----
16: C arguments (i=input, o=output, w=work)
17: C
18: C i T(*) : ORDERED TABLE VECTOR, DECREASING, ARBITRARY LENGTH.
19: C i N : COMMON TABLE LENGTH FOR EACH PARTICLE.
20: C i LS: STARTING ADDRESS OF TABLES FOR EACH PARTICLE.
21: C   ( T(LS(I)) TO T(LS(I)+N-1) --> TABLE FOR EACH PARTICLE )
22: C i LLS: LS IS INDEXED BY LLS(I) FOR BSDEC2
23: C i X(N1) : INPUT VECTOR. LENGTH N1
24: C o IT(N1): OUTPUT INDEX VECTOR, SUCH THAT
25: C
26: C   T(LS+IT(I)-1) >= X(I) > T(LS+IT(I))
27: C   IT(I) = 0      IF X(I) > T(LS) WHEN N:EVEN
28: C   IT(I) = 1      IF X(I) > T(LS) WHEN N:ODD
29: C   IT(I) = N-1    IF X(I) <= T(LS+N-1)
30: C
31: C=====
32: C
33: C   subroutine BSDEC1( T,      N,      LS,      X,      IT,      N1 )
34: C
35: C   real T(*), X(N1)
36: C   integer LS(*), LLS(N1), IT(N1)
37: C
38: C   do 100 I = 1, N1
39: C     IT(I) = LS(I) + (N-1)/2 + 1
40: C     IT(I) = LS(I) + (N-1) /2
41: C 100 continue
42: C   IBS = 1
43: C   go to 120
44: C
45: C ===== HAVING LIST VECTOR LLS(I) FOR LS =====
46: C
47: C   entry BSDEC2(T,N,LS,LLS,X,IT,N1)
48: C
49: C   do 110 I = 1, N1
50: C     IT(I) = LS(LLS(I)) + (N-1)/2 + 1
51: C     IT(I) = LS(LLS(I)) + (N-1) /2
52: C 110 continue
53: C   IBS = 2
54: C   go to 120
55: C
56: C=====
57: C
58: C 120 M = (N-2) /2
59: C 130 continue
60: C   if ( M.gt.0 ) then
61: C     MP = (M+1) /2
62: C     M = M/2
63: C     do 140 I = 1, N1
64: C       II = IT(I)
65: C       IF( X(I).GT.T(II) ) THEN

```

```

66: C           IT(I) = IT(I) - MP
67: C       ELSE
68: C           IT(I) = IT(I) + MP
69: C       ENDIF
70: C       MM = MP
71: C       if ( X(I).gt.T(IT(I)) ) MM = -MM
72: C       IT(I) = IT(I) + MM
73: C 140 continue
74: C   go to 130
75: C end if
76: C do 150 I = 1, N1
77: C   II = IT(I)
78: C   IF( X(I).GT.T(II) ) IT(I) = IT(I) - 1
79: C   if ( X(I).gt.T(IT(I)) ) IT(I) = IT(I) - 1
80: C 150 continue
81: C -----
82: C   if ( IBS.eq.1 ) then
83: C     do 160 I = 1, N1
84: C       IT(I) = IT(I) - LS(I) + 1
85: C 160 continue
86: C   else
87: C     do 170 I = 1, N1
88: C       IT(I) = IT(I) - LS(LLS(I)) + 1
89: C 170 continue
90: C   end if
91: C   return
92: C end

```

src/shared/bsdec3.f

```

1: C-----
2: C   BSDEC3, BSDEC4 : EACH PARTICLES HAVE THEIR OWN TABLE LENGTHES.
3: C-----
4: C
5: C arguments (i=input, o=output, w=work)
6: C
7: C i T : ORDERED TABLE VECTOR, DECREASING, ARBITRARY LENGTH.
8: C i NT: TABLE LENGTH FOR EACH PARTICLE.
9: C w MW: WORKING AREA TO STORE INITIAL TABLE LENGTH AND USED AS
10: C     POSITION MODIFIER.
11: C i LS: STARTING ADDRESS OF TABLES FOR EACH PARTICLE.
12: C     ( T(LS(I)) TO T(LS(I)+N-1) --> TABLE FOR EACH PARTICLE )
13: C i LLS: LS IS INDEXED BY LLS(I) FOR BSDEC4
14: C i X : INPUT VECTOR. LENGTH N1
15: C o IT : OUTPUT INDEX VECTOR, SUCH THAT
16: C
17: C     T(LS+IT(J)-1) >= X(J) > T(LS+IT(J))
18: C     IF X(J) > T(LS) WHEN N:EVEN
19: C     IT(J) = 1 IF X(J) > T(LS) WHEN N:ODD
20: C     IT(J) = NT-1 IF X(J) <= T(LS+NT-1)
21: C-----
22: C
23: C   subroutine BSDEC3( T, NT, MW, LS, X, IT, N1 )
24: C
25: C   real T(*), X(N1)
26: C   integer LS(*), LLS(N1), NT(N1), MW(N1), IT(N1)
27: C
28: C   NMAX = 0
29: C   do 100 I = 1, N1
30: C     IT(I) = LS(I) + (NT(I)-1)/2 + 1
31: C     IT(I) = LS(I) + (NT(I)-1) /2
32: C     MW(I) = (NT(I)-2) /2
33: C     NMAX = MAX(NT(I),NMAX)
34: C   100 continue
35: C   IBS = 3
36: C   go to 120
37: C
38: C ===== HAVING LIST VECTOR LLS(I) FOR LS =====
39: C
40: C   entry BSDEC4(T,NT,MW,LS,LLS,X,IT,N1)
41: C
42: C   NMAX = 0
43: C   do 110 I = 1, N1
44: C     IT(I) = LS(LLS(I)) + (NT(I)-1)/2 + 1
45: C     IT(I) = LS(LLS(I)) + (NT(I)-1) /2
46: C     MW(I) = (NT(I)-2) /2
47: C     NMAX = MAX(NT(I),NMAX)
48: C   110 continue
49: C   IBS = 4
50: C   go to 120
51: C
52: C-----
53: C
54: C   120 M = (NMAX-2) /2
55: C   130 continue
56: C
57: C
58: C   if ( M.gt.0 ) then
59: C     CCCCC MP = (M+1)/2
60: C     M = M/2
61: C     do 140 I = 1, N1
62: C       MP = (MW(I)+1) /2
63: C       II = IT(I)
64: C       IF( X(I).GT.T(II) ) THEN
65: C         IT(I) = IT(I) - MP

```

```

66: C     ELSE
67: C       IT(I) = IT(I) + MP
68: C     ENDIF
69: C     if ( X(I).gt.T(IT(I)) ) MP = -MP
70: C     IT(I) = IT(I) + MP
71: C     MW(I) = MW(I) /2
72: C   140 continue
73: C   go to 130
74: C   end if
75: C   do 150 I = 1, N1
76: C     II = IT(I)
77: C     IF( X(I).GT.T(II) ) IT(I) = IT(I) - 1
78: C     if ( X(I).gt.T(IT(I)) ) IT(I) = IT(I) - 1
79: C   150 continue
80: C-----
81: C   if ( IBS.eq.3 ) then
82: C     do 160 I = 1, N1
83: C       IT(I) = IT(I) - LS(I) + 1
84: C   160 continue
85: C   else
86: C     do 170 I = 1, N1
87: C       IT(I) = IT(I) - LS(LLS(I)) + 1
88: C   170 continue
89: C   end if
90: C   return
91: C   end

```

src/shared/bsincl.f

```

1: C=====
2: C      BSINC1, BSINC2
3: C=====
4: C PURPOSE: BINARY SEARCH EACH PARTICLE USES DIFFERENT TABLES (BUT OF
5: C      THE SAME LENGTH).
6: C CALLED IN: (VAREIOUS ROUTINES)
7: C
8: C arguments (i=input, o=output, w=work)
9: C
10: C i T : ORDERED TABLE VECTOR, INCREASING, ARBITRARY LENGTH.
11: C i N : COMMON TABLE LENGTH FOR EACH PARTICLE.
12: C i LS: STARTING ADDRESS OF TABLES FOR EACH PARTICLE.
13: C      ( T(LS(I)) TO T(LS(I)+N-1) --> TABLE FOR EACH PARTICLE )
14: C i X : INPUT VECTOR. LENGTH N1
15: C o IT: OUTPUT INDEX VECTOR, SUCH THAT
16: C
17: C      T(IT(I)) <= X(I) < T(IT(I)+1)
18: C      IT(I)= 0      IF X(I)<T(LS)  WHEN N:EVEN
19: C      IT(I)= 1      IF X(I)<T(LS)  WHEN N:ODD
20: C      IT(I)= NT(I)-1 IF X(I)>= T(LS+N-1)
21: C=====
22: C
23: C      subroutine BSINC1( T,      N,      LS,      X,      IT,      N1 )
24: C
25: C      real T(*), X(N1)
26: C      integer LS(*), LLS(N1), IT(N1)
27: C
28: C      do 100 I = 1, N1
29: C      IT(I) = LS(I) + (N-1)/2 + 1
30: C      IT(I) = LS(I) + (N-1) /2
31: C 100 continue
32: C      IBS      = 1
33: C      go to 120
34: C
35: C ===== HAVING LIST VECTOR LLS(I) FOR LS =====
36: C
37: C      entry BSINC2(T,N,LS,LLS,X,IT,N1)
38: C
39: C      do 110 I = 1, N1
40: C      IT(I) = LS(LLS(I)) + (N-1)/2 + 1
41: C      IT(I) = LS(LLS(I)) + (N-1) /2
42: C 110 continue
43: C      IBS      = 2
44: C      go to 120
45: C
46: C=====
47: C
48: C      120 M      = (N-2) /2
49: C      130 continue
50: C      if ( M.gt.0 ) then
51: C      MP      = (M+1) /2
52: C      M      = M/2
53: C      do 140 I = 1, N1
54: C      II = IT(I)
55: C      IF( X(I).LT.T(IT) ) THEN
56: C      IT(I) = IT(I) - MP
57: C      ELSE
58: C      IT(I) = IT(I) + MP
59: C      ENDIF
60: C      MM      = MP
61: C      if ( X(I).lt.T(IT(I)) ) MM = -MM
62: C      IT(I) = IT(I) + MM
63: C 140 continue
64: C      go to 130
65: C      end if

```

```

66:      do 150 I = 1, N1
67: C      II = IT(I)
68: C      IF( X(I).LT.T(IT) ) IT(I) = IT(I) - 1
69: C      if ( X(I).lt.T(IT(I)) ) IT(I) = IT(I) - 1
70: C 150 continue
71: C-----
72: C      if ( IBS.eq.1 ) then
73: C      do 160 I = 1, N1
74: C      IT(I) = IT(I) - LS(I) + 1
75: C 160 continue
76: C      else
77: C      do 170 I = 1, N1
78: C      IT(I) = IT(I) - LS(LLS(I)) + 1
79: C 170 continue
80: C      end if
81: C      return
82: C      end

```

src/shared/bsinc3.f

```

1: C=====
2: C      BSINC3, BSINC4
3: C=====
4: C PURPOSE: BINARY SEARCH EACH PARTICLE USES DIFFERENT TABLES AND THEIR
5: C      ALSO MAY BE DIFFERENT.
6: C CALLED IN: (VARIOUS ROUTINES)
7: C
8: C arguments (i=input, o=output, w=work)
9: C
10: C i T : ORDERED TABLE VECTOR, INCREASING, ARBITRARY LENGTH.
11: C i NT: TABLE LENGTH FOR EACH PARTICLE.
12: C w MW: WORK AREA TO STORE INITIAL TABLE LENGTH AND USED AS
13: C      POSITION MODIFIER.
14: C i LS: STARTING ADDRESS OF TABLES FOR EACH PARTICLE.
15: C      T(LS(I)) TO T(LS(I)+N-1) --> TABLE FOR EACH PARTICLE )
16: C i X : INPUT VECTOR. LENGTH N1
17: C o IT : OUTPUT INDEX VECTOR, SUCH THAT
18: C
19: C      T(IT(J)) <= X(J) < T(IT(J)+1))
20: C      IT(J)=0      IF X(J) < T(LS)  WHEN N:EVEN
21: C      IT(J)=1      IF X(J) < T(LS)  WHEN N:ODD
22: C      IT(J)=N-1    IF X(J) >= T(LS+NT-1)
23: C=====
24: C
25: C      subroutine BSINC3( T,      NT,      MW,      LS,      X,      IT,      N1 )
26: C
27: C      real T(*), X(N1)
28: C      integer LS(*), LLS(N1), NT(N1), MW(N1), IT(N1)
29: C
30: C      NMAX      = 0
31: C      do 100 I = 1, N1
32: C          IT(I)      = LS(I) + (NT(I)-1)/2 + 1
33: C          IT(I)      = LS(I) + (NT(I)-1) /2
34: C          MW(I)      = (NT(I)-2) /2
35: C          NMAX      = MAX(NT(I),NMAX)
36: C      100 continue
37: C      IBS      = 3
38: C      go to 120
39: C
40: C ===== HAVING LIST VECTOR LLS(I) FOR LS =====
41: C
42: C      entry BSINC4(T,NT,MW,LS,LLS,X,IT,N1)
43: C
44: C      NMAX      = 0
45: C      do 110 I = 1, N1
46: C          IT(I)      = LS(LLS(I)) + (NT(I)-1)/2 + 1
47: C          IT(I)      = LS(LLS(I)) + (NT(I)-1) /2
48: C          MW(I)      = (NT(I)-2) /2
49: C          NMAX      = MAX(NT(I),NMAX)
50: C      110 continue
51: C      IBS      = 4
52: C      go to 120
53: C
54: C=====
55: C
56: C      120 M      = (NMAX-2) /2
57: C      130 continue
58: C
59: C
60: C      if ( M.gt.0 ) then
61: C          CCCCC MP      = (M+1)/2
62: C          M      = M/2
63: C          do 140 I = 1, N1
64: C              MP      = (MW(I)+1) /2
65: C              II = IT(I)

```

```

66: C          IF( X(I).LT.T(II) ) THEN
67: C              IT(I) = IT(I) - MP
68: C          ELSE
69: C              IT(I) = IT(I) + MP
70: C          ENDIF
71: C          if ( X(I).lt.T(IT(I)) ) MP      = -MP
72: C          IT(I)      = IT(I) + MP
73: C          MW(I)      = MW(I) /2
74: C      140 continue
75: C      go to 130
76: C      end if
77: C      do 150 I = 1, N1
78: C          II = IT(I)
79: C          IF( X(I).LT.T(II) ) IT(I) = IT(I) - 1
80: C          if ( X(I).lt.T(IT(I)) ) IT(I)      = IT(I) - 1
81: C      150 continue
82: C-----
83: C      if ( IBS.eq.3 ) then
84: C          do 160 I = 1, N1
85: C              IT(I)      = IT(I) - LS(I) + 1
86: C      160 continue
87: C      else
88: C          do 170 I = 1, N1
89: C              IT(I)      = IT(I) - LS(LLS(I)) + 1
90: C      170 continue
91: C      end if
92: C      return
93: C      end
94: C
95: C-----
96: C
97: C      subroutine BS3ISD( T,      NT,      MW,      LS,      X,      IT,      N1 )
98: C
99: C      real T(*)
100: C      real*8 X(N1)
101: C      integer LS(*), LLS(N1), NT(N1), MW(N1), IT(N1)
102: C
103: C      NMAX      = 0
104: C      do 100 I = 1, N1
105: C          IT(I)      = LS(I) + (NT(I)-1)/2 + 1
106: C          IT(I)      = LS(I) + (NT(I)-1) /2
107: C          MW(I)      = (NT(I)-2) /2
108: C          NMAX      = MAX(NT(I),NMAX)
109: C      100 continue
110: C      IBS      = 3
111: C      go to 120
112: C
113: C ===== HAVING LIST VECTOR LLS(I) FOR LS =====
114: C
115: C      entry BS4ISD(T,NT,MW,LS,LLS,X,IT,N1)
116: C
117: C      NMAX      = 0
118: C      do 110 I = 1, N1
119: C          IT(I)      = LS(LLS(I)) + (NT(I)-1)/2 + 1
120: C          IT(I)      = LS(LLS(I)) + (NT(I)-1) /2
121: C          MW(I)      = (NT(I)-2) /2
122: C          NMAX      = MAX(NT(I),NMAX)
123: C      110 continue
124: C      IBS      = 4
125: C      go to 120
126: C
127: C=====
128: C
129: C      120 M      = (NMAX-2) /2
130: C      130 continue

```

src/shared/bsinc3.f

```

131: C
132: C
133:   if ( M.gt.0 ) then
134: CCCCC   MP   = (M+1)/2
135:       M     = M/2
136:       do 140 I = 1, N1
137:           MP   = (MW(I)+1) /2
138: C       II = IT(I)
139: C       IF( X(I).LT.T(IT(I)) ) THEN
140: C           IT(I) = IT(I) - MP
141: C       ELSE
142: C           IT(I) = IT(I) + MP
143: C       ENDIF
144: C       if ( X(I).lt.T(IT(I)) ) MP = -MP
145: C       IT(I) = IT(I) + MP
146: C       MW(I) = MW(I) /2
147: 140 continue
148:       go to 130
149:   end if
150:   do 150 I = 1, N1
151: C       II = IT(I)
152: C       IF( X(I).LT.T(IT(I)) ) IT(I) = IT(I) - 1
153: C       if ( X(I).lt.T(IT(I)) ) IT(I) = IT(I) - 1
154: 150 continue
155: C-----
156:   if ( IBS.eq.3 ) then
157:       do 160 I = 1, N1
158:           IT(I) = IT(I) - LS(I) + 1
159: 160 continue
160:   else
161:       do 170 I = 1, N1
162:           IT(I) = IT(I) - LS(LLS(I)) + 1
163: 170 continue
164:   end if
165:   return
166: end
167: C
168: C-----
169: C
170:   subroutine BS3IDD( T,      NT,      MW,      LS,      X,      IT,      N1 )
171: C
172:   real*8 T(*)
173:   real*8 X(N1)
174:   integer LS(*), LLS(N1), NT(N1), MW(N1), IT(N1)
175: C
176:   NMAX   = 0
177:   do 100 I = 1, N1
178: C       IT(I)   = LS(I) + (NT(I)-1)/2 + 1
179: C       IT(I)   = LS(I) + (NT(I)-1) /2
180: C       MW(I)   = (NT(I)-2) /2
181: C       NMAX    = MAX(NT(I),NMAX)
182: 100 continue
183:   IBS    = 3
184:   go to 120
185: C
186: C===== HAVING LIST VECTOR LLS(I) FOR LS =====
187: C
188:   entry BS4IDD(T,NT,MW,LS,LLS,X,IT,N1)
189: C
190:   NMAX   = 0
191:   do 110 I = 1, N1
192: C       IT(I)   = LS(LLS(I)) + (NT(I)-1)/2 + 1
193: C       IT(I)   = LS(LLS(I)) + (NT(I)-1) /2
194: C       MW(I)   = (NT(I)-2) /2
195: C       NMAX    = MAX(NT(I),NMAX)

```

```

196:   110 continue
197:       IBS   = 4
198:       go to 120
199: C
200: C=====
201: C
202: 120 M     = (NMAX-2) /2
203: 130 continue
204: C
205: C
206:   if ( M.gt.0 ) then
207: CCCCC   MP   = (M+1)/2
208:       M     = M/2
209:       do 140 I = 1, N1
210:           MP   = (MW(I)+1) /2
211: C       II = IT(I)
212: C       IF( X(I).LT.T(IT(I)) ) THEN
213: C           IT(I) = IT(I) - MP
214: C       ELSE
215: C           IT(I) = IT(I) + MP
216: C       ENDIF
217: C       if ( X(I).lt.T(IT(I)) ) MP = -MP
218: C       IT(I) = IT(I) + MP
219: C       MW(I) = MW(I) /2
220: 140 continue
221:       go to 130
222:   end if
223:   do 150 I = 1, N1
224: C       II = IT(I)
225: C       IF( X(I).LT.T(IT(I)) ) IT(I) = IT(I) - 1
226: C       if ( X(I).lt.T(IT(I)) ) IT(I) = IT(I) - 1
227: 150 continue
228: C-----
229:   if ( IBS.eq.3 ) then
230:       do 160 I = 1, N1
231:           IT(I) = IT(I) - LS(I) + 1
232: 160 continue
233:   else
234:       do 170 I = 1, N1
235:           IT(I) = IT(I) - LS(LLS(I)) + 1
236: 170 continue
237:   end if
238:   return
239: end

```

src/shared/bsvdec.f

```

1: C=====
2: C  PURPOSE:  BINARY SEARCH IN A VALUE TABLE IN DECREASING ORDER.
3: C  CALLED IN:  many routines.
4: C
5: C arguments (i=input, o=output, w=work)
6: C
7: C i  T : ORDERED TABLE VECTOR, DECREASING (REAL).  LENGTH = N
8: C i  X : INPUT VECTOR.  LENGTH N1
9: C o  I : OUTPUT INDEX VECTOR, SUCH THAT
10: C      T(I(J)) >= X(J) > T(I(J)+1)
11: C      I(J)=0      IF X(J) > T(1)  WHEN N:EVEN
12: C      I(J)=1      IF X(J) > T(1)  WHEN N:ODD
13: C      I(J)=N-1    IF X(J) <= T(N)
14: C=====
15: C      subroutine BSVDEC( T,      N,      X,      I,      N1 )
16: C
17: C      ... both T & X are in single precision ...
18: C
19: C      real T(N), X(N1)
20: C      integer I(N1)
21: C      M      = (N-2) /2
22: C      do 100 J = 1, N1
23: C          I(J)      = (N-1) /2 + 1
24: C      100 continue
25: C
26: C      110 continue
27: C          if ( M.gt.0 ) then
28: C              MP      = (M+1) /2
29: C              M      = M/2
30: C              do 120 J = 1, N1
31: C                  CCCCCC      II = I(J)
32: C                  CCCCCCCCCCCC IF( X(J).GT.T(II) ) THEN
33: C                      I(J) = I(J) - MP
34: C                  ELSE
35: C                      I(J) = I(J) + MP
36: C                  CCCCCCCCCCCC ENDIF
37: C                      MM      = MP
38: C                      if ( X(J).gt.T(I(J)) ) MM      = -MM
39: C                      I(J)      = I(J) + MM
40: C      120      continue
41: C      else
42: C          do 130 J = 1, N1
43: C              II = I(J)
44: C              IF( X(J).GT.T(II) ) THEN
45: C                  if ( X(J).gt.T(I(J)) ) then
46: C                      I(J)      = I(J) - 1
47: C                  end if
48: C      130      continue
49: C          return
50: C      end if
51: C      go to 110
52: C      end
53: C
54: C
55: C
56: C      subroutine BS0DSD( T,      N,      X,      I,      N1 )
57: C
58: C      ... X is double precision ...
59: C
60: C      real      T(N)
61: C      real*8 X(N1)
62: C      integer I(N1)
63: C      M      = (N-2) /2
64: C      do 100 J = 1, N1
65: C          I(J)      = (N-1) /2 + 1

```

```

66:      100 continue
67: C
68: C      110 continue
69: C          if ( M.gt.0 ) then
70: C              MP      = (M+1) /2
71: C              M      = M/2
72: C              do 120 J = 1, N1
73: C                  CCCCCC      II = I(J)
74: C                  CCCCCCCCCCCC IF( X(J).GT.T(II) ) THEN
75: C                      I(J) = I(J) - MP
76: C                  ELSE
77: C                      I(J) = I(J) + MP
78: C                  CCCCCCCCCCCC ENDIF
79: C                      MM      = MP
80: C                      if ( X(J).gt.T(I(J)) ) MM      = -MM
81: C                      I(J)      = I(J) + MM
82: C      120      continue
83: C      else
84: C          do 130 J = 1, N1
85: C              II = I(J)
86: C              IF( X(J).GT.T(II) ) THEN
87: C                  if ( X(J).gt.T(I(J)) ) then
88: C                      I(J)      = I(J) - 1
89: C                  end if
90: C      130      continue
91: C          return
92: C      end if
93: C      go to 110
94: C      end
95: C
96: C
97: C
98: C      subroutine BS0DDD( T,      N,      X,      I,      N1 )
99: C
100: C      ... both T & X are double precision ...
101: C
102: C      real*8 T(N)
103: C      real*8 X(N1)
104: C      integer I(N1)
105: C      M      = (N-2) /2
106: C      do 100 J = 1, N1
107: C          I(J)      = (N-1) /2 + 1
108: C      100 continue
109: C
110: C      110 continue
111: C          if ( M.gt.0 ) then
112: C              MP      = (M+1) /2
113: C              M      = M/2
114: C              do 120 J = 1, N1
115: C                  CCCCCC      II = I(J)
116: C                  CCCCCCCCCCCC IF( X(J).GT.T(II) ) THEN
117: C                      I(J) = I(J) - MP
118: C                  ELSE
119: C                      I(J) = I(J) + MP
120: C                  CCCCCCCCCCCC ENDIF
121: C                      MM      = MP
122: C                      if ( X(J).gt.T(I(J)) ) MM      = -MM
123: C                      I(J)      = I(J) + MM
124: C      120      continue
125: C      else
126: C          do 130 J = 1, N1
127: C              II = I(J)
128: C              IF( X(J).GT.T(II) ) THEN
129: C                  if ( X(J).gt.T(I(J)) ) then
130: C                      I(J)      = I(J) - 1

```

src/shared/bsvdec.f

```
131:         end if
132: 130      continue
133:         return
134:     end if
135:     go to 110
136: end
```

SAFE

src/shared/bsvinc.f

```

1: C=====
2: C  PURPOSE:  BINARY SEARCH
3: C  CALLED IN: (NOWHERE)
4: C
5: C arguments (i=input, o=output, w=work)
6: C
7: C i  T : ORDERED TABLE VECTOR, INCREASING.  LENGTH = N
8: C i  X : INPUT VECTOR.  LENGTH N1
9: C o  I : OUTPUT INDEX VECTOR, SUCH THAT
10: C      T(I(J)) <= X(J) < T(I(J)+1))
11: C      I(J)=0  IF X(J)<T(1)  WHEN N:EVEN
12: C      I(J)=1  IF X(J)<T(1)  WHEN N:ODD
13: C      I(J)=N-1 IF X(J)>= T(N)
14: C=====
15: C      subroutine BSVINC( T,      N,      X,      I,      N1 )
16: C      real T(N), X(N1)
17: C      integer I(N1)
18: C
19: C      M      = (N-2) /2
20: C      do 100 J = 1, N1
21: C          I(J) = (N-1) /2 + 1
22: C      100 continue
23: C
24: C
25: C      110 continue
26: C      if ( M.gt.0 ) then
27: C          MP      = (M+1) /2
28: C          M      = M/2
29: C          do 120 J = 1, N1
30: C              II = I(J)
31: C              IF( X(J).LT.T(II) ) THEN
32: C                  I(J) = I(J) - MP
33: C              ELSE
34: C                  I(J) = I(J) + MP
35: C              ENDIF
36: C              MM      = MP
37: C              if ( X(J).lt.T(I(J)) ) MM      = -MM
38: C              I(J)      = I(J) + MM
39: C      120  continue
40: C      else
41: C          do 130 J = 1, N1
42: C              II = I(J)
43: C              IF( X(J).LT.T(II) ) THEN
44: C                  if ( X(J).lt.T(I(J)) ) then
45: C                      I(J)      = I(J) - 1
46: C                  end if
47: C      130  continue
48: C          return
49: C      end if
50: C      go to 110
51: C
52: C      end
53: C
54: C-----
55: C
56: C      subroutine BS0ISD( T,      N,      X,      I,      N1 )
57: C      real T(N)
58: C      real*8 X(N1)
59: C      integer I(N1)
60: C
61: C      M      = (N-2) /2
62: C      do 100 J = 1, N1
63: C          I(J) = (N-1) /2 + 1
64: C      100 continue
65: C

```

```

66: C
67: C      110 continue
68: C      if ( M.gt.0 ) then
69: C          MP      = (M+1) /2
70: C          M      = M/2
71: C          do 120 J = 1, N1
72: C              II = I(J)
73: C              IF( X(J).LT.T(II) ) THEN
74: C                  I(J) = I(J) - MP
75: C              ELSE
76: C                  I(J) = I(J) + MP
77: C              ENDIF
78: C              MM      = MP
79: C              if ( X(J).lt.T(I(J)) ) MM      = -MM
80: C              I(J)      = I(J) + MM
81: C      120  continue
82: C      else
83: C          do 130 J = 1, N1
84: C              II = I(J)
85: C              IF( X(J).LT.T(II) ) THEN
86: C                  if ( X(J).lt.T(I(J)) ) then
87: C                      I(J)      = I(J) - 1
88: C                  end if
89: C      130  continue
90: C          return
91: C      end if
92: C      go to 110
93: C
94: C      end
95: C
96: C-----
97: C
98: C      subroutine BS0IDD( T,      N,      X,      I,      N1 )
99: C      real*8 T(N)
100: C      real*8 X(N1)
101: C      integer I(N1)
102: C
103: C      M      = (N-2) /2
104: C      do 100 J = 1, N1
105: C          I(J) = (N-1) /2 + 1
106: C      100 continue
107: C
108: C
109: C      110 continue
110: C      if ( M.gt.0 ) then
111: C          MP      = (M+1) /2
112: C          M      = M/2
113: C          do 120 J = 1, N1
114: C              II = I(J)
115: C              IF( X(J).LT.T(II) ) THEN
116: C                  I(J) = I(J) - MP
117: C              ELSE
118: C                  I(J) = I(J) + MP
119: C              ENDIF
120: C              MM      = MP
121: C              if ( X(J).lt.T(I(J)) ) MM      = -MM
122: C              I(J)      = I(J) + MM
123: C      120  continue
124: C      else
125: C          do 130 J = 1, N1
126: C              II = I(J)
127: C              IF( X(J).LT.T(II) ) THEN
128: C                  if ( X(J).lt.T(I(J)) ) then
129: C                      I(J)      = I(J) - 1
130: C                  end if

```


src/shared/bsvinc.f

```
131: 130    continue
132:      return
133:    end if
134:    go to 110
135: C
136:    end
```

SAFE

src/shared/catstr.f

```
1:      subroutine CATSTR( STR,  LSTR,  STR2,  IERR )
2: C==<MVP/GMVP>=====
3: C purpose: concatenate an character string STR2 to a character string
4: C          STR after STR(:LSTR) and return the length of the resulting
5: C          string in LSTR.
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C
9: C io STR : string before/after concatenation
10: C io LSTR : string length before/after concatenation
11: C i STR2 : string to append
12: C o IERR : returns non zero when length of concatenated string exceeds
13: C          length of STR.
14: C=====
15:      character*(*) STR, STR2
16: C
17:      LLS      = LSTR + LEN(STR2)
18:      if ( LLS.gt.LEN(STR) ) then
19:          IERR  = 1
20:          return
21:      end if
22: C
23:      IERR      = 0
24:      STR(LSTR+1:LLS) = STR2
25:      LSTR      = LLS
26:      return
27:      end
28:
29:      subroutine CATST2( STR,  LSTR,  STR1,  STR2,  IERR )
30: C==<MVP/GMVP>=====
31: C purpose: concatenate character strings STR1 & STR2 to a character
32: C          string STR after STR(:LSTR) and return the length of the
33: C          resulting string in LSTR.
34: C history: programmed by M.Sasaki  (25 Jun 1994)
35: C-----
36: C arguments (i=input, o=output, w=work)
37: C
38: C io STR : string before/after concatenation
39: C io LSTR : string length before/after concatenation
40: C i STR1 : string to append
41: C i STR2 : string to append
42: C o IERR : returns non zero when length of concatenated string exceeds
43: C i STR2 : string to be concatenated
44: C=====
45:      character*(*) STR, STR1, STR2
46: C
47:      LLS1      = LSTR + LEN(STR1)
48:      LLS2      = LLS1 + LEN(STR2)
49:      if ( LLS2.gt.LEN(STR) ) then
50:          IERR    = 1
51:          return
52:      end if
53: C
54:      IERR      = 0
55:      STR(LSTR+1:LLS1) = STR1
56:      STR(LLS1+1:LLS2) = STR2
57:      LSTR      = LLS2
58:      return
59:      end
```

src/shared/cbsinc.f

```

1:      subroutine CBSINC( TNAMS, NNames,NAME,  IPoS )
2: C
3: C      MVP/GMVP UTILITY
4: C
5: C=====
6: C  PURPOSE:  FIND POSITION OF 'NAME' IN NAME TABLE 'TNAMS(NNames)'
7: C            AND RETURN POSITION IN IPoS.
8: C            IPoS = 0 IF NAME WAS NOT FOUND IN TNAMS
9: C
10: C      *Names IN TNAMS MUST BE SORTED IN ASCENDING ORDER.
11: C      *LENGTH OF NAMES ARE GIVEN BY PARAMETER LNAME
12: C  CALLS :  ISTRCM ( FUNCTIONS TO DETERMINE CHARACTER STRING ORDER )
13: C=====
14: C
15: C      CHARACTER*12 TNAMS(5)
16: C      DATA TNAMS /
17: C      @      'A'
18: C      @      'KB'
19: C      @      '12'
20: C      @      '23CDE'
21: C      @      '1000000' /
22: C      CALL CBSINC( TNAMS, 5, '12', IPoS )
23: C      WRITE( 6, * ) IPoS
24: C      CALL CBSINC( TNAMS, 5, '1000000', IPoS )
25: C      WRITE( 6, * ) IPoS
26: C      CALL CBSINC( TNAMS, 5, 'A', IPoS )
27: C      WRITE( 6, * ) IPoS
28: C      CALL CBSINC( TNAMS, 5, '89', IPoS )
29: C      WRITE( 6, * ) IPoS
30: C      END
31: C
32: C      include 'INC/_LNAME'
33: C      character*(LNAME) TNAMS(NNames)
34: C      character*(*) NAME
35: C
36: C      LL      = INDEX(NAME,' ') - 1
37: C      if ( LL.lt.0 ) LL      = LEN(NAME)
38: C
39: C      if ( NNames.lt.128 ) then
40: C
41: C      ... LINEAR SEARCH ...
42: C
43: C      do 100 I = 1, NNames
44: C      if ( TNAMS(I).eq.NAME ) then
45: C      IPoS      = I
46: C      return
47: C      end if
48: C 100  continue
49: C      IPoS      = 0
50: C      return
51: C      else
52: C
53: C      ... BINARY SEARCH ...
54: C
55: C      I0      = 1
56: C      I1      = NNames
57: C
58: C 110  I2      = (I0+I1) /2
59: C      LTN      = INDEX(TNAMS(I2),' ') - 1
60: C      if ( LTN.lt.0 ) LTN = LEN(TNAMS(I2))
61: C      KK      = ISTRCM(TNAMS(I2)(:LTN),NAME(:LL))
62: C
63: C      if ( KK.eq.0 ) then
64: C      IPoS      = I2
65: C      return

```

```

66: C      else if ( I0.eq.I1 ) then
67: C      IPoS      = 0
68: C      return
69: C      end if
70: C
71: C      if ( KK.gt.0 ) then
72: C      I1      = I2
73: C      else
74: C      I0      = I2
75: C      end if
76: C      if ( I1.gt.I0+1 ) go to 110
77: C
78: C
79: C      LTN      = INDEX(TNAMS(I1),' ') - 1
80: C      if ( LTN.lt.0 ) LTN = LEN(TNAMS(I1))
81: C      if ( ISTRCM(TNAMS(I1)(:LTN),NAME(:LL)).eq.0 ) then
82: C      IPoS      = I1
83: C      return
84: C      end if
85: C      LTN      = INDEX(TNAMS(I0),' ') - 1
86: C      if ( LTN.lt.0 ) LTN = LEN(TNAMS(I0))
87: C      if ( ISTRCM(TNAMS(I0)(:LTN),NAME(:LL)).eq.0 ) then
88: C      IPoS      = I0
89: C      return
90: C      end if
91: C
92: C      IPoS      = 0
93: C      return
94: C      end if
95: C      end

```

src/shared/ccomp.f

```
1:      subroutine CCOMP( OUTCH, LN,      INCH, IFL )
2:      C
3:      C      GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C  PURPOSE:  'Compress' character string INCH by removing blanks
7:      C      and output result on string 'OUTCH' upto LN characters.
8:      C      ('OUTCH' & 'INCH' can overlap in memory but starting address of
9:      C      'OUTCH' must precede or be identical to that of 'INCH'. )
10:     C
11:     C      IFL : flag returned
12:     C          0 = normal end
13:     C          1 = number of non-blank characters in INCH exceeds LN.
14:     C
15:     C-----
16:     C arguments (i=input, o=output, w=work)
17:     C
18:     C o  OUTCH : string after compression
19:     C i  LN : do not save characters after OUTCH(LN:LN).
20:     C i  INCH : input character string
21:     C o  IFLAG : non zero if number of non-blank characters exceeds LN.
22:     C=====
23:     C      character*(*) OUTCH, INCH
24:     C      ... local data
25:     C      character*1 CH1
26:     C-----
27:     C
28:     C      IFL      = 0
29:     C      LL       = LEN(INCH)
30:     C      NN       = 0
31:     C      do 100 I = 1, LL
32:     C          if ( INCH(I:I).ne.' ' ) then
33:     C              NN      = NN + 1
34:     C              if ( NN.le.LN ) then
35:     C                  CH1   = INCH(I:I)
36:     C                  OUTCH(NN:NN) = CH1
37:     C              end if
38:     C          end if
39:     C      100 continue
40:     C      if ( NN.lt.LN ) OUTCH(NN+1:LN) = ' '
41:     C      if ( NN.gt.LN ) IFL = 1
42:     C      return
43:     C      end
```

src/shared/cel2ab.f

```

1:      subroutine CEL2AB( IOW,  NN,   NN0,
2:      &                  A,    JDEBG, XXX0,  YYY0,  ZZZ0,
3:      &                  LEVL0, LZZ0,  LPOS0, LCRS0,
4:      &                  DBNK0,KDBNK0,MDBNK0,
5:      &                  XX,    YY,    ZZ )
6: C=====
7: C purpose: translate in-cell coordinates to absolute coordinates.
8: C called in: actsgl,actsmp,actmpp,actvpp
9: C calls : LATUPW
10: C-----
11: C arguments ( i=input o=output w=work )
12: C
13: C i IOW : printout I/O unit
14: C i NN : number of particles to be processed
15: C i NN0 : size of the first dimension of arrays
16: C io a : task shared dynamic data area used as "REAL" type.
17: C i JDEBG : debug option flag
18: C io XXX0(NN0),YYY0(NN0),ZZZ0(NN0) : in cell coordinates.
19: C i LEVL0(NN0),LZZ0(NN0,NEST),LPOS0(NN0,NEST),LCRS0(NN0,NEST) :
20: C      lattice data.
21: C i DBNK0 : optional bank data. Currently, STG cell position
22: C      is the only effective data when NSTGR.ne.0.
23: C w XX(NN0),YY(NN0),ZZ(NN0) : work array to store translated coordinates
24: C=====
25:      real A(*)
26: C
27: C .... particle attributes in lattice .....
28: C
29:      real*8 XXX0(NN0), YYY0(NN0), ZZZ0(NN0)
30:      integer LEVL0(NN0), LZZ0(NN0,NEST), LPOS0(NN0,NEST),
31:      &      LCRS0(NN0,NEST)
32: C
33:      real*8 DBNK0(NN0,*)
34:      integer KDBNK0(0:MDBNK0)
35: C
36:      real*8 XX(NN0), YY(NN0), ZZ(NN0)
37: C
38:      integer JDEBG(*)
39: C
40: CCCC include 'INC/_ARRAY'
41:      include 'INC/_PGEOM'
42:      include 'INC/_SIZES'
43: C
44: C ... local
45: C
46:      real*8 DUMMY
47: C
48: C-----
49: C
50: C ... no need to get direction in LATUPW
51:      JDIR = 0
52: C
53:      call LATUPW( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ, NZONE, NN, NN0,
54:      &      JDIR,
55:      &      XX, YY, ZZ, DUMMY, DUMMY, DUMMY,
56:      B      XXX0, YYY0, ZZZ0,
57:      B      DUMMY, DUMMY, DUMMY, LEVL0, LZZ0, LPOS0, LCRS0,
58:      B      DBNK0,KDBNK0,MDBNK0,
59:      G      A(LKZMAT),
60:      G      A(LKDALT), A(LDALT), A(LNVLAT), A(LSZLAT), A(LCELSZ),
61:      G      A(LIPLAT), A(LKLAT), A(LKSLAT), A(LLTYP), A(LICTYP),
62:      G      A(LMLBZZ) )
63: C
64:      do 100 I = 1, NN
65:          XXX0(I) = XX(I)
66:          YYY0(I) = YY(I)
67:          ZZZ0(I) = ZZ(I)
68:      100 continue
69: C
70:      return
71:      end

```

src/shared/celtr.f

```

1:      subroutine CELTR( JTLLT, NLATT, NCELL, KCELL, IDCEL, CELSZ, KCELI,
2:      &                  KINPZ, NINPZ, KZMAT, KZREG,
3:      &                  IPCEL, IDLAT, SZLAT, IPLAT, KLATT,
4:      &                  NZDA, NZONE, KZDA, KZAA, SDA, ICTYP, LTYP,
5:      &                  DEPS )
6: C=====
7: C PURPOSE: TO CHECK THE DATA FOR LATTICE-CELL
8: C CALLED IN: GEOMIN
9: C-----
10: C arguments (i=input, o=output, w=work )
11: C
12: C*i IOG: message printout I/O unit ( removed Jan 2000 )
13: C i NLATT, NCELL : number of lattice geometry and lattice cells
14: C o KCELL(NZONE) : cell numbers to which each zone belongs
15: C i IDCEL(NCELL) : ID of cells
16: C o CELSZ(NCELL) : size parameters of cells
17: C o KCELI(NINPZ) : cell numbers to which each input-zone belongs
18: C i KINPZ, NINPZ, KZMAT, KZREG, IDLAT, SZLAT, IPLAT, KLATT, NZDA,
19: C NZONE, KZDA, KZAA, SDA : geometry data (see INC/_PGEOM)
20: C i ICTYP(NCELL) : type of cells
21: C i LTYP(NLATT) : type of lattice geometry
22: C i DEPS : small distance.
23: C=====
24:
25:      implicit real*8(D)
26:      integer KCELL(NZONE), IDCEL(NCELL), IPCEL(NCELL+1), IDLAT(NLATT),
27:      &        IPLAT(NLATT+1), KLATT(*), KZDA(2,NZDA), KZAA(*),
28:      &        ICTYP(NCELL), LTYP(NLATT)
29:      integer KCELI(NINPZ), KINPZ(NZONE)
30:      integer KZMAT(NZONE,2), KZREG(NZONE)
31:      real*8 DEPS, CELSZ(6,NCELL), SZLAT(8,NLATT), SDA(*)
32:      character*12 TYPE
33: C
34:      include 'INC/_IOUNIT'
35: C
36:      include 'INC/_PMLATT'
37:      include 'INC/_SFLATT'
38: C
39: C-----
40: C
41: C *****
42:
43:      call LABEL( IOG, 'CELL DATA' )
44:
45: C *****
46: C
47: C ..... KCELL ARRAY (CELL NUMBERS TO WHICH EACH ZONE BELONGS. )
48: C
49:      do 100 I = 1, IPCEL(1) - 1
50:          KCELL(I) = 0
51:      100 continue
52:      do 120 N = 1, NCELL
53:          do 110 I = IPCEL(N), IPCEL(N+1) - 1
54:              KCELL(I) = N
55:          110 continue
56:      120 continue
57:      do 130 I = 1, NZONE
58:          KCELI(KINPZ(I)) = KCELL(I)
59:      130 continue
60: C
61: C ..... CELL ID NUMBERS IN KLATT --> CELL SEQUENTIAL NUMBERS
62: C
63:      IFF = 0
64:      do 240 N = 1, NLATT
65: C

```

```

66: C ... non STGM region
67: C
68:      if ( LTYP(N).ne.10 ) then
69:          do 140 K = IPLAT(N), IPLAT(N+1) - 1
70:              KLATT(K) = -KLATT(K)
71:          140 continue
72:          do 160 I = 1, NCELL
73:              ID = -IDCEL(I)
74:              do 150 K = IPLAT(N), IPLAT(N+1) - 1
75:                  if ( KLATT(K).lt.0.and.KLATT(K).eq.ID ) KLATT(K) = I
76:              150 continue
77:          160 continue
78:          do 170 K = IPLAT(N), IPLAT(N+1) - 1
79:              if ( KLATT(K).lt.0 ) then
80:                  IFF = 1
81:                  write(IMSG,7000) - KLATT(K), IDLAT(N)
82:                  format(' XXX CELL ',I5,' IN LATTICE ',I5,
83:                      ' IS NOT DEFINED !!!')
84:                  & call CNTERR( 'FATAL' )
85:              end if
86:          170 continue
87: C
88: C ... STGM region
89: C
90:      else if ( LTYP(N).eq.10 ) then
91:          K = IPLAT(N)
92:          NSTG = KLATT(K+9)
93:          do 180 I = 1, NCELL
94:              if ( KLATT(K).eq.IDCEL(I).and.ICTYP(I).eq.5 ) then
95:                  KLATT(K) = I
96:                  go to 190
97:              end if
98:          180 continue
99:          IFF = 1
100:          write(IMSG,7020) KLATT(K), IDLAT(N)
101:          format(' XXX matrix material cell ',I5,' in STGM lattice ',
102:              & I5,' has no corresponding "MBASE" type cell!!')
103:          call CNTERR( 'FATAL' )
104:          190 continue
105:          do 210 KK = 1, NSTG
106:              do 200 I = 1, NCELL
107:                  if ( KLATT(K+KK).eq.IDCEL(I).and.ICTYP(I).eq.3 ) then
108:                      KLATT(K+KK) = I
109:                      go to 210
110:                  end if
111:              200 continue
112:              IFF = 1
113:              write(IMSG,7040) KLATT(K+KK), IDLAT(N)
114:              format(' XXX STG cell ',I5,' in lattice ',I5,
115:                  & ' has no corresponding STG cell!!')
116:              call CNTERR( 'FATAL' )
117:          210 continue
118: C
119:      if ( IFF.ne.0 ) go to 230
120: C
121: C ... convert KZMAT(*,2) and KZREG of STGM region to those
122: C of the second zone of "MBASE" cell.
123: C (KZREG here is not the real one for TALLY-LATTICE mode)
124: C
125:      IKZ = IPCEL(KLATT(K)) + 1
126:      do 220 IZ = 1, NZONE
127:          if ( ISLATT(KZMAT(IZ,1)).and.LATTNM(KZMAT(IZ,1)).eq.N )
128:              & then
129:                  KZMAT(IZ,2) = KZMAT(IKZ,1)
130:                  if ( JTLLT.eq.0 ) KZREG(IZ) = KZREG(IKZ)

```

src/shared/celtr.f

```

131:          end if
132: 220      continue
133: C
134: 230      continue
135:          end if
136: 240      continue
137: C
138:          if ( IFF.ne.0 ) then
139:              write(IMG,7060)
140:              call PRSTOP( 1, 'ERROR IN LATTICE GEOMETRY DATA.' )
141:              stop 888
142:          end if
143: 7060      format(/1X,'XXX Fatal error in lattice geometry input'/1X,
144:          &      'XXX STOP ')
145: C
146: C .... CELL SIZE ....
147: C
148:          do 250 N = 1, NCELL
149:
150:              call CELTYP( TYPE, '<', ICTYP(N) )
151:
152:              write(IOG,7080) N, IDCEL(N), ICTYP(N), TYPE
153: 7080      format(/3X,60('#')/5X,'CELL NO:',I4,' (CELL-ID : ',I5,
154:          &      ' TYPE =',I2,' <=',A,'> ')/3X,60('#'))
155:
156:              KK      = KZAA(IPCEL(N))
157:              IFF3     = 0
158: C
159: C ..... RECTANGULAR CELL (ICTYP = 1) .....
160: C ..... CHECK WHETHER THE FIRST BODY OF THE CELL IS RPP OR NOT...
161: C
162:          if ( ICTYP(N).eq.1 ) then
163:              LSX      = ABS(KZDA(1,KK))
164:              LSY      = ABS(KZDA(1,KK+1))
165:              LSZ      = ABS(KZDA(1,KK+2))
166:              ISX      = INT(SDA(LSX))
167:              ISY      = INT(SDA(LSY))
168:              ISZ      = INT(SDA(LSZ))
169:
170:              if ( ISX.ne.1 .or. ISY.ne.1 .or. ISZ.ne.1 ) IFF3 = 1
171:              if ( SDA(LSX+1).ne.1 .or. SDA(LSY+2).ne.1
172:          &      .or. SDA(LSZ+3).ne.1 ) IFF3 = 1
173:
174:              if ( IFF3.eq.1 ) then
175:                  write(IMG,7100) IDCEL(N)
176: 7100      format(/1X,'XXX FIRST BODY OF THE CELL ',I5,
177:          &      ' IS NOT "RPP" !!')
178:                  call CNTERR( 'FATAL' )
179:                  IFF      = 1
180:              else
181: C
182: C CELSZ(1 - 3) : CELL PITCH IN X,Y & Z DIRECTION
183: C
184:              CELSZ(1,N) = SDA(LSX+5) - SDA(LSX+4)
185:              CELSZ(2,N) = SDA(LSY+5) - SDA(LSY+4)
186:              CELSZ(3,N) = SDA(LSZ+5) - SDA(LSZ+4)
187: C
188: C CELSZ(4 - 6) : COORDINATE OF THE CELL CENTER
189: C
190:              CELSZ(4,N) = (SDA(LSX+5)+SDA(LSX+4))*0.5D0
191:              CELSZ(5,N) = (SDA(LSY+5)+SDA(LSY+4))*0.5D0
192:              CELSZ(6,N) = (SDA(LSZ+5)+SDA(LSZ+4))*0.5D0
193:              write(IOG,7140) (CELSZ(K,N),K=1,6), IPCEL(N), IPCEL(N+1)
194:          &      - 1
195:          end if

```

```

196: C
197: C ..... HEXAGONAL CELL .....
198: C (ASSUMING THAT THE FIRST ZONE OF THE CELL BE CONSTRUCTED OF
199: C AN 'RHP' BODY AND NOT 'HEX' BODY)
200: C
201:          else if ( ICTYP(N).eq.2 ) then
202:              LSX      = ABS(KZDA(1,KK))
203:              LS1      = ABS(KZDA(1,KK+1))
204:              LSZ      = ABS(KZDA(1,KK+3))
205: C
206: C ..... CELL PITCH ....
207: C
208:              CELSZ(1,N) = ABS(SDA(LSX+5)-SDA(LSX+4))
209: C
210: C ..... CELL HEIGHT ....
211: C
212:              CELSZ(2,N) = ABS(SDA(LSZ+5)-SDA(LSZ+4))
213: C
214: C ..... CELL CENTER (X,Y,Z) .....
215: C
216:              CELSZ(3,N) = 0.5*(SDA(LSX+5)+SDA(LSX+4))
217:              CELSZ(4,N) = (SDA(LS1+5)+SDA(LS1+4)-CELSZ(3,N)) /SQRT(3.D0)
218:              CELSZ(5,N) = 0.5D0*(SDA(LSZ+5)+SDA(LSZ+4))
219:              write(IOG,7160) (CELSZ(K,N),K=1,5), IPCEL(N), IPCEL(N+1) - 1
220: C
221: C ..... STG CELL (sphere) .....
222: C
223: C CELSZ(1 - 3) : CELL CENTER in cell coordinate
224: C CELSZ(4) : CELL radius
225: C
226:          else if ( ICTYP(N).eq.3 ) then
227:              LSX      = ABS(KZDA(1,KK))
228:              ISX      = INT(SDA(LSX))
229:              if ( ISX.ne.2 ) then
230:                  write(IMG,7120) IDCEL(N)
231: 7120      format(/1X,'XXX First body of the cell ',I5,
232:          &      ' of type "STG" is not a sphere !! ')
233:                  call CNTERR( 'FATAL' )
234:                  IFF      = 1
235:              end if
236:              CELSZ(1,N) = SDA(LSX+1)
237:              CELSZ(2,N) = SDA(LSX+2)
238:              CELSZ(3,N) = SDA(LSX+3)
239:              CELSZ(4,N) = SQRT(ABS(SDA(LSX+4)))
240:              write(IOG,7180) (CELSZ(K,N),K=1,4), IPCEL(N), IPCEL(N+1) - 1
241:          end if
242: 250      continue
243: 7140      format(' ** CELL PARAMETERS **/' X,Y,Z PITCHES ',
244:          &      1P,3E12.5/' CELL CENTER ',3E12.5/' ZONE ',
245:          &      I6,' TO ',I6)
246: 7160      format(' ** CELL PARAMETERS **/' PITCH ',1P,E12.5,
247:          &      ' HEIGHT ',E12.5/' CELL CENTER ',3E12.5/
248:          &      ' ZONE ',I6,' TO ',I6/)
249: 7180      format(' ** CELL PARAMETERS **/' CELL CENTER ',
250:          &      1P,3E12.5/' RADIUS ',E12.5/' ZONE ',I6,
251:          &      ' TO ',I6/)
252: C
253: C ..... CHECK WHETHER THE CELL SIZES MATCH THE SIZE OF LATTICES
254: C WHICH THE CELLS BELONG TO.
255: C
256:          DEPS1      = -DEPS*1.D-5
257:          do 280 N = 1, NLATT
258: C
259: C ... other than STG region ...
260: C

```

src/shared/celtr.f

```
261:         if ( LTYP(N).ne.10 ) then
262:             do 260 K = IPLAT(N), IPLAT(N+1) - 1
263:                 KK
264:                 = KLATT(K)
265:                 if ( KK.ne.0 ) then
266:                     if ( LTYP(N).eq.1 ) then
267:                         DSX
268:                         = CELSZ(1,KK)
269:                         DSY
270:                         = CELSZ(2,KK)
271:                         DSZ
272:                         = CELSZ(3,KK)
273:                         if ( ABS(DSX-SZLAT(1,N)).gt.DEPS
274:                             & .or. ABS(DSY-SZLAT(2,N)).gt.DEPS
275:                             & .or. ABS(DSZ-SZLAT(3,N)).gt.DEPS
276:                             & .or. ABS(DSX-SZLAT(1,N)).lt.DEPS1
277:                             & .or. ABS(DSY-SZLAT(2,N)).lt.DEPS1
278:                             & .or. ABS(DSZ-SZLAT(3,N)).lt.DEPS1 ) then
279:                             write(IMG,7200)IDCEL(KK), IDLAT(N), SZLAT(1,N),
280:                                 & SZLAT(2,N), SZLAT(3,N)
281:                             call CNTERR( 'FATAL' )
282:                             IFF
283:                             = 1
284:                             end if
285:                         else if ( LTYP(N).ge.2.and.LTYP(N).le.4 ) then
286:                             if ( ABS(CELSZ(1,KK)-SZLAT(1,N)).gt.DEPS
287:                                 & .or. CELSZ(1,KK)-SZLAT(1,N).lt.DEPS1
288:                                 & .or. ABS(CELSZ(2,KK)-SZLAT(2,N)).gt.DEPS
289:                                 & .or. ABS(CELSZ(3,KK)-SZLAT(3,N)).gt.DEPS
290:                                 & .or. CELSZ(2,KK)-SZLAT(2,N).lt.DEPS1
291:                                 & .or. CELSZ(3,KK)-SZLAT(3,N).lt.DEPS1 ) then
292:                                 write(IMG,7220)IDCEL(KK), IDLAT(N), SZLAT(1,N),
293:                                     & SZLAT(2,N), SZLAT(3,N)
294:                                 call CNTERR( 'FATAL' )
295:                                 IFF
296:                                 = 1
297:                                 end if
298:                             end if
299:                         end if
300:                     continue
301:                 end if
302:             end if
303:             continue
304:         C
305:         ... STG region ...
306:         C
307:         else if ( LTYP(N).eq.10 ) then
308:             if ( SZLAT(2,N).eq.0.0D0 ) then
309:                 KK
310:                 = KLATT(IPLAT(N)+1)
311:                 SZLAT(2,N) = CELSZ(4,KK)
312:             end if
313:             do 270 K = 1, KLATT(IPLAT(N)+9)
314:                 KKK
315:                 = KLATT(IPLAT(N)+K)
316:                 if ( ABS(CELSZ(4,KKK)-SZLAT(2,N)).gt.DEPS ) then
317:                     write(IMG,7240) IDCEL(KKK), IDLAT(N), SZLAT(2,N)
318:                     call CNTERR( 'FATAL' )
319:                     IFF
320:                     = 1
321:                 end if
322:             end if
323:             continue
324:         end if
325:         280 continue
326:     312:
327:     7200 format(/' XXX CELL SIZE OF CELL ',I5,' DOES NOT MATCH THE ',
328:         & 'SIZE OF LATTICE ',I5,' !!! ',1P,3E12.5)
329:     7220 format(/' XXX CELL SIZE OF CELL ',I5,' DOES NOT MATCH THE ',
330:         & 'SIZE OF LATTICE ',I5,' !!! ',1P,2E12.5)
331:     7240 format(/' XXX CELL SIZE OF CELL ',I5,' (STG) DOES NOT MATCH THE ',
332:         & 'SIZE OF LATTICE ',I5,' !!! ',1P,E12.5)
333:     return
334: end
```


src/shared/celtyp.f

```
1:      subroutine CELTYP( CTYPE, DIREC, ICTYP )
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C   PURPOSE :   Cell type symbol <==>   cell type number conversion
7: C               (   DIREC   '>' : SYMBOL -> NUMBER )
8: C               '<' : SYMBOL <- NUMBER )
9: C=====
10:      character CTYPE*(*), DIREC*1
11: C
12:      parameter( NTYPE      = 7 )
13:      character*8 CTYPES(NTYPE)
14:      integer      ITYPES(NTYPE)
15:      data CTYPES /'BOX', 'HEXA', 'STGP', 'FREE',
16:      & 'MBASE', 'STG', 'DUMMY' /
17:      data ITYPES / 1, 2, 3, 4, 5, 3, 5 /
18: C
19: C-----
20: C
21:      if ( DIREC.eq.'>' ) then
22:        do 100 I = 1, NTYPE
23:          if ( CTYPE.eq.CTYPES(I) ) then
24:            ICTYP = ITYPES(I)
25:            return
26:          end if
27: 100    continue
28:        ICTYP = 0
29:        return
30:      else if ( DIREC.eq.'<' ) then
31:        do 110 I=1,NTYPE
32:          if ( ICTYP.eq.ITYPES(I) ) then
33:            CTYPE = CTYPES(I)
34:            return
35:          end if
36: 110    continue
37:        CTYPE = ' '
38:      end if
39:      return
40: C
41:      end
```

src/shared/chkdup.f

```
1:      subroutine CHKDUP( IDATA, NCHECK,NSTEP, NERR,  IOUT,  OBJECT )
2: C=<MVP/GMVP>=====
3: C purpose: Check duplicate data in an integer type array.
4: C
5: C calls : (none)
6: C called in: anywhere
7: C-----
8: C arguments (i=input, o=output, w=work )
9: C
10: C i idata(*) : integer array checked.
11: C i ncheck   : number of elements checked.
12: C i nstep    : check every nstepth elements.
13: C o nerr     : = 0    no duplicate data.
14: C             > 0    number of duplicated elements.
15: C i iout     : printout i/o unit of discovered errors if non negative.
16: C i object   : character string to indicate name,attribute etc.
17: C             of 'idata'.
18: C-----
19: C
20: C=====
21:      integer IDATA(NSTEP,NCHECK)
22:      character*(*) OBJECT
23: C
24: C
25:      NERR      = 0
26: C
27:      if ( NSTEP.eq.0 ) then
28:        if ( IOUT.ge.0 ) write(IOUT,*)
29:        &      'XXX ZERO STEP REFERENCE IN "CHKDUP" ROUTINE. ',
30:        &      '( CHECKING <', OBJECT, '> )'
31:        stop 666
32:      end if
33: C
34:      do 110 J = 2, NCHECK
35: C
36:        do 100 I = 1, J - 1
37:          if ( IDATA(1,I).eq.IDATA(1,J) ) then
38:            NERR      = NERR + 1
39:            if ( IOUT.ge.0 ) then
40:              write(IOUT,7000) J, OBJECT, IDATA(1,J), I, OBJECT
41: 7000      format(1X,' XXX ',I5,'''TH ',A,' ( =',I8,
42:  &      ' ) IS IDENTICAL TO ',I5,'''TH ',A,' !!!')
43:            end if
44:          go to 110
45:        end if
46:      100    continue
47: C
48:      110 continue
49:      return
50:      end
```

src/shared/chkngp.f

```

1:      subroutine CHKNGP( CODE, JNEUT, NGP1, JPHOT, NGP2, NGROUP )
2: C=====
3: C Purpose: check (or set if necessary) number of energy groups
4: C         (or energy bins)
5: C         This should be called before procedure that may require
6: C         definition of energy bin numbers.
7: C called in: INTRO, INTRO2
8: C-----
9: C i CODE : program name (MVP/GMVP)
10: C i JNEUT : flag for neutron problem
11: C i NGP1  : number of neutron energy bins
12: C i JPHOT : flag for photon problem
13: C i NGP2  : number of photon energy bins
14: C i NGROUP: total number of energy bins over all particles
15: C=====
16:      include 'INC/_IOUNIT'
17: C
18:      character*4 CODE
19: C-----
20: C
21: C ... check bin(group) numbers of each particles
22: C
23:      JSET = 0
24: 7000 format(/lx,'!!! (CHKNGP) Unspecified ',a,' energy bin (group) ',
25: &         'number is set to ',i4)
26:
27:      if ( JNEUT.ne.0.and.NGP1.eq.0 ) then
28:          NGP1 = 1
29:          write(IMG,7000) 'neutron',NGP1
30:          call CNTERR('WARNING')
31:          JSET = 1
32:      end if
33: C
34:      if ( JPHOT.ne.0.and.NGP2.eq.0 ) then
35:          NGP2 = 1
36:          write(IMG,7000) 'photon',NGP2
37:          call CNTERR('WARNING')
38:          JSET = 1
39:      end if
40: C
41: C ... check total number of enrgy bins (gorups) NGROUP
42: C
43: 7020 format(/lx,'!!! (CHKNGP) Total number of energy bin(group) ',
44: &         '"NGROUP" is set to ',I4)
45:
46: 7040 format(/lx,'XXX (CHKNGP) Total number of energy bin(group) ',
47: &         '"NGROUP" (= ',I4,' ) is different from sum of number of ',
48: &         ' energy bin(group) of particles (= ',i4,' ).')
49:
50: 7060 format(/lx,'!!! (CHKNGP) Total number of energy bin(group) ',
51: &         '"NGROUP" (= ',I4,' ) is different from sum of number of ',
52: &         ' energy bin(group) of particles (= ',i4,' )'/lx,
53: &         ' NGROUP is reset to the summation value.')
54:
55:      NN      = NGP1 + NGP2
56:      if ( NGROUP.eq.0 ) then
57:          NGROUP = NN
58:          write(IMG,7020) NN
59:          call CNTERR('WARNING')
60:      else if ( NGROUP.ne.NN ) then
61:          if ( JSET.eq.0 ) then
62:              write(IMG,7040) NGROUP, NN
63:              call CNTERR('FATAL')
64:          else
65:              write(IMG,7060) NGROUP, NN
66:
67:          NGROUP = NN
68:          call CNTERR('WARNING')
69:          end if
70:      C
71:      return
72:      end

```

src/shared/ciszon.f

```
1:      subroutine CISZON( ISZON, NSOUR, KINPZ, NZONE )
2: C=====
3: C PURPOSE:  CHANGE THE COMBINATION OF ISZON(1,NSOUR) (INPUT-ZONE)
4: C           AND ISZON(2,NSOUR) (ZONE NUMBER IN INPUT-ZONE)  TO SEQUENTIAL
5: C           ZONE NUMBER.
6: C
7: C << obsolescent: this routine should never be called >>
8: C
9: C CALLED IN:  INTRO
10: C CALLS:    (NONE)
11: C=====
12:      integer ISZON(2,NSOUR), KINPZ(NZONE)
13:      include 'INC/_IOUNIT'
14: C
15:      do 120 N = 1, NSOUR
16:      do 100 I = 1, NZONE
17:      if ( ISZON(1,N).eq.KINPZ(I) ) go to 110
18:      100 continue
19:      write(IMG,'(1x,a,i4,a,a,i3,a)') 'XXX (CISZON) INPUT-ZONE ',
20:      & ISZON(1,N), ' DOES ', 'NOT EXIST !' (SOURCE NUMBER = '
21:      & N, ' ) '
22:      call PRSTOP( 1, 'INVALID "ISZON" DATA.' )
23:      stop 888
24:      110 ISZON(1,N) = I + ISZON(2,N) - 1
25:      120 continue
26: C
27:      return
28:      end
```

src/shared/ckalp.f

```
1:      subroutine CKALP( PSALP, NREG )
2: C=====
3: C PURPOSE : Check consistency of path stretching factor PSALP.
4: C
5: C Called in : INTRO2
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C i PSALP(NREG) : path stretching factor
9: C i NREG: number of regions
10: C=====
11:      real PSALP(NREG)
12: C
13:      include 'INC/_IOUNIT'
14: C
15: C-----
16: C
17:      NERR = 0
18:      do 100 IR = 1, NREG
19:          if ( ABS(PSALP(IR)) .ge. 1.0 ) then
20:              write(IMG,7000) IR, PSALP(IR)
21:              NERR = NERR + 1
22:              call CNTERR( 'FATAL' )
23:          end if
24:      100 continue
25: C
26:      return
27: C
28: 7000 format(1X,'XXX ABS(PSALP) is greater than or equal to 1 for ',
29:           & ' REGION ',I4,' (PSALP=',E12.5,')')
30: C
31:      end
```

src/shared/ckbats.f

```
1:      subroutine CKBATS( NHSUB, NHIST, NBANK )
2: C=====
3: C purpose: check batch history size (NHIST) and
4: C          sub-batch history size (NHSUB) and set it if necessary.
5: C called in: INTRO2
6: C=====
7:      include 'INC/_IUNIT'
8: C
9: C-----
10: C
11:      if ( NHIST.le.0 ) then
12:         write(IMG,*) 'XXX Histories per batch (NHIST) is not given',
13:         & ' or non-positive : ',NHIST
14:         call CNTERR('FATAL')
15: C      ... set tempoaray value
16:         NHSUB = 1000
17:      end if
18: C
19: C ... check/set sub-batch size
20: C
21:      if ( NHSUB.le.0 ) then
22: C
23:         write(IPR,*) '<<MESSAGE>> Sub-batch history size (NHSUB)',
24:         & ' is not given.'
25:         call CNTERR('MESSAGE')
26:         if ( NBANK.le.0 ) then
27:            NHSUB = NHIST
28:            write(IPR,*) ' NHSUB is set to batch size(NHIST) : ',NHSUB
29:         else
30:            NHSUB = min(NHIST,NBANK)
31:            write(IPR,*) ' NHSUB is set to min(NHIST,NBANK) : ',NHSUB
32:         end if
33:
34:      else if ( NHSUB.gt.NHIST ) then
35:
36:         write(IMG,*) '!!! Sub-batch history size (NHSUB=',NHSUB,')',
37:         & ' is larger than that of batch (NHIST=',NHIST,')'
38:         call CNTERR('WARNING')
39:         if ( NBANK.le.0 ) then
40:            NHSUB = NHIST
41:            write(IMG,*) ' NHSUB is set to batch size(NHIST) : ',NHSUB
42:         else
43:            NHSUB = min(NBANK,NHIST)
44:            write(IMG,*) ' NHSUB is set to min(NHIST,NBANK) : ',NHSUB
45:         end if
46:
47:      end if
48: C
49:      return
50: end
```

src/shared/cknul0.f

```
1:      subroutine CKNUL0( A, LAST,  LSTART )
2:      C
3:      C   MVP/GMVP UTILITY
4:      C
5:      C=====
6:      C PURPOSE: CHECK ASSIGNMENT TO UNDEFINED ADDRESS IN ARRAY MEMORY AREA
7:      C HISTORY: PROGRAMMED BY M.SASAKI   MAY 1992
8:      C CALLED IN:  CKNUL0 --- INTRO AS INITIALIZATION
9:      C              CKNULL --- ANYWHERE NECESSARY FOR DEBUGGING )
10:     C=====
11:     CCCCC include 'INC/_ARRAY'
12:     real A(*)
13:     C
14:     include 'INC/_IOUNIT'
15:     C
16:     real M2, M3
17:     parameter( M1 = 12345, M2 = 999., M3 = -999. )
18:     C
19:     C ... data statement to save L1,L2 after call ...
20:     data L1 /1/, L2 /2/
21:     C
22:     CCCC  LSTART = IAINF(1)
23:     LSTART = ILSIZ(1)
24:     L1 = LSTART
25:     L2 = LSTART + 1
26:     L3 = LSTART + 2
27:     cccc  IDUM = M1
28:     A(L1) = M2
29:     A(L2) = M3
30:     call STLAST( 'CKNUL0', L3, LAST )
31:     return
32:     C
33:     C
34:     entry CKNULL( A, IFLAG )
35:     IFLAG = 0
36:     cccc  IF( IDUM.NE.M1 .OR. A(L1).NE.M2 .OR. A(L2).NE.M3 ) THEN
37:     if ( A(L1).ne.M2 .or. A(L2).ne.M3 ) then
38:       write(IMG,*) 'XXX INVALID VALUE ASSIGNMENT OCCURRED !! ', M2,
39:       &      M3, ' -> ', A(L1), A(L2)
40:       IFLAG = 1
41:     end if
42:     return
43:     end
```

src/shared/ckodr4.f

```

1:      subroutine CKODR4( A,      N,      J,      ICK )
2: C
3: C      MVP/GMVP UTILITY
4: C
5: C=====
6: C Purpose: Check order of single precision array A(1:n)
7: C-----
8: C arguments (i=input, o=output, w=work)
9: C
10: C i  A(N) : array to be checked
11: C i  J : operation mode
12: C      J=1, DECREASING ORDER OR NOT
13: C      J=2, INCREASING ORDER OR NOT
14: C      ( If J < 0. equality is allowed. )
15: C o  ICK : number of elements not in order
16: C-----
17: C
18:      real A(N)
19: C
20:      ICK      = 0
21: C
22:      if ( J.eq.1 ) then
23:      do 100 I = 2, N
24:          if ( A(I).ge.A(I-1) ) then
25:              ICK      = ICK + 1
26:          end if
27:      100  continue
28: C
29:      else if ( J.eq.2 ) then
30:      do 110 I = 2, N
31:          if ( A(I).le.A(I-1) ) then
32:              ICK      = ICK + 1
33:          end if
34:      110  continue
35: C
36:      else if ( J.eq.-1 ) then
37:      do 200 I = 2, N
38:          if ( A(I).gt.A(I-1) ) then
39:              ICK      = ICK + 1
40:          end if
41:      200  continue
42: C
43:      else if ( J.eq.-2 ) then
44:      do 210 I = 2, N
45:          if ( A(I).lt.A(I-1) ) then
46:              ICK      = ICK + 1
47:          end if
48:      210  continue
49:      end if
50: C
51:      return
52:      end
53:
54:      subroutine CKODR8( A,      N,      J,      ICK )
55: C
56: C      MVP/GMVP UTILITY
57: C
58: C=====
59: C      CHECK ORDER OF DOUBLE PRECISION ARRAY A(1:n)
60: C-----
61: C arguments (i=input, o=output, w=work)
62: C
63: C i  A(N) : array to be checked
64: C i  J : operation mode
65: C      J=1, DECREASING ORDER OR NOT

```

```

66: C      J=2, INCREASING ORDER OR NOT
67: C      ( If J < 0. equality is allowed. )
68: C o  ICK : number of elements not in order
69: C=====
70: C
71:      real*8 A(N)
72: C
73:      ICK      = 0
74: C
75:      if ( J.eq.1 ) then
76:      do 100 I = 2, N
77:          if ( A(I).ge.A(I-1) ) then
78:              ICK      = ICK + 1
79:          end if
80:      100  continue
81: C
82:      else if ( J.eq.2 ) then
83:      do 110 I = 2, N
84:          if ( A(I).le.A(I-1) ) then
85:              ICK      = ICK + 1
86:          end if
87:      110  continue
88: C
89:      else if ( J.eq.-1 ) then
90:      do 200 I = 2, N
91:          if ( A(I).gt.A(I-1) ) then
92:              ICK      = ICK + 1
93:          end if
94:      200  continue
95: C
96:      else if ( J.eq.-2 ) then
97:      do 210 I = 2, N
98:          if ( A(I).lt.A(I-1) ) then
99:              ICK      = ICK + 1
100:          end if
101:      210  continue
102:      end if
103: C
104:      return
105:      end

```


src/shared/ckval4.f

```

1:      subroutine CKVAL4( VNAME, A,      N,      IFL,      IOP )
2:      C
3:      C   JAERI MONTE CARLO CODE UTILITY
4:      C
5:      C=====
6:      C PURPOSE :   CHECK SIGNS OF DATA IN ARRAY A (REAL )
7:      C   Option code:
8:      C       IOP = 0 : FIND NEGATIVE VALUES
9:      C       IOP = 1 : FIND NEGATIVE AND ZERO VALUES
10:     C   Return code:
11:     C       IFL = 0 : Not found
12:     C       IFL = 1 : found
13:     C
14:     C CALLED IN:  VARIOUS ROUTINES
15:     C-----
16:     C arguments (i=input, o=output, w=work)
17:     C i VNAME : data name if any
18:     C i A(N)  : array to be checked
19:     C i N     : size of array
20:     C o IFL   : return code (see above)
21:     C o IOP   : operation mode (see above)
22:     C=====
23:     real A(N)
24:     character*(*) VNAME
25:     include 'INC/_IOUNIT'
26: CC
27:     LVN      = INDEX(VNAME, ' ') - 1
28:     if ( LVN.lt.0 ) LVN = LEN(VNAME)
29: CC
30:     IFL      = 0
31:     if ( IOP.eq.0 ) then
32:       do 100 I = 1, N
33:         if ( A(I).lt.0.0 ) then
34:           IFL      = 1
35:           write(IMSG,7000) I, VNAME(1:LVN), A(I)
36:         end if
37:       100 continue
38:     else if ( IOP.eq.1 ) then
39:       do 110 I = 1, N
40:         if ( A(I).le.0.0 ) then
41:           IFL      = 1
42:           write(IMSG,7020) I, VNAME(1:LVN), A(I)
43:         end if
44:       110 continue
45:     end if
46: C
47: 7000 format(//
48:   &' *****
49:   &' ****      SIGN CHECK FOR INPUT DATA !!!!!!!!!!! *****
50:   &' *****
51:   &' **
52:   &'      ' ** ',I6,'''TH ELEMENT OF ARRAY <',A,'> IS NEGATIVE.',
53:   &'      T61,'** '/' **      VALUE = ',E12.5,T61,'** '/'
54:   &' **
55:   &' *****
56: 7020 format(//
57:   &' *****
58:   &' ****      SIGN CHECK FOR INPUT DATA !!!!!!!!!!! *****
59:   &' *****
60:   &' **
61:   &'      ' ** ',I6,'''TH ELEMENT OF ARRAY <',A,'> IS <= 0.0 ',
62:   &'      T61,'** '/' **      VALUE = ',E12.5,T61,'** '/'
63:   &' **
64:   &' *****
65:   return

```

66: end

src/shared/ckval8.f

```

1:      subroutine CKVAL8( VNAME, D,      N,      IFL,      IOP )
2: C=====
3: C PURPOSE : CHECK SIGNS OF DATA IN ARRAY A (REAL*8)
4: C   Option code:
5: C       IOP = 0 : FIND NEGATIVE VALUES
6: C       IOP = 1 : FIND NEGATIVE AND ZERO VALUES
7: C   Return code:
8: C       IFL = 0 : Not found
9: C       IFL = 1 : found
10: C
11: C CALLED IN : VARIOUS ROUTINES
12: C-----
13: C arguments (i=input, o=output, w=work)
14: C i VNAME : data name if any
15: C i D(N) : array to be checked
16: C i N      : size of array
17: C o IFL : return code (see above)
18: C o IOP : operation mode (see above)
19: C-----
20:      real*8 D(N)
21:      character*(*) VNAME
22:      include 'INC/_IOUNIT'
23: CC
24:      LVN      = INDEX(VNAME,' ') - 1
25:      if ( LVN.lt.0 ) LVN = LEN(VNAME)
26: CC
27: CC
28:      IFL      = 0
29:      if ( IOP.eq.0 ) then
30:        do 100 I = 1, N
31:          if ( D(I).lt.0.0 ) then
32:            IFL      = 1
33:            write(IMSG,7000) I, VNAME(1:LVN), D(I)
34:          end if
35:        continue
36:      else if ( IOP.eq.1 ) then
37:        do 110 I = 1, N
38:          if ( D(I).lt.0.0 ) then
39:            IFL      = 1
40:            write(IMSG,7020) I, VNAME(1:LVN), D(I)
41:          end if
42:        continue
43:      end if
44: C
45: 7000 format(//
46: &' *****
47: &' ****      SIGN CHECK FOR INPUT DATA !!!!!!!!!!! *****
48: &' *****
49: &' **
50: &'      ' ** ',I5,' 'TH ELEMENT OF ARRAY <','A,'> IS NEGATIVE.',
51: &'      T69,'** ' /
52: &' **
53: &'      ' **      VALUE  =  ',E12.5,T69,'** ' /
54: &' **
55: &' *****
56: 7020 format(//
57: &' *****
58: &' ****      SIGN CHECK FOR INPUT DATA !!!!!!!!!!! *****
59: &' *****
60: &' **
61: &'      ' ** ',I5,' 'TH ELEMENT OF ARRAY <','A,'> IS <= 0.0  ',
62: &'      T69,'** ' /
63: &' **
64: &'      ' **      VALUE  =  ',E12.5,T69,'** ' /
65: &' **

```

```

66:      &' *****
67:      return
68:      end

```

src/shared/ckwght.f

```
1:      subroutine CKWGHT( WSRV, WKIL, NGROUP,NREG )
2: C=====
3: C PURPOSE : Check consistency of WSRV and WKIL
4: C
5: C Called in : INTRO2
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C
9: C i WSRV(NGROUP,NREG) : survival weight
10: C i WKIL(NGROUP,NREG) : russian roulette threshold weight
11: C i NGROUP: number of energy groups
12: C i NREG: number of regions
13: C=====
14:      real WSRV(NGROUP,NREG)
15:      real WKIL(NGROUP,NREG)
16: C
17:      include 'INC/ IOWNIT'
18: C
19: C-----
20: C
21:      call CKVAL4( 'WSRV', WSRV, NGROUP*NREG, IFL, 1 )
22:      if ( IFL.gt.0 ) then
23:        write(IMG,*) 'XXX WSRV contains ', IFL, ' non-positive data.'
24:        call CNTERR( 'FATAL' )
25:      end if
26: C
27:      call CKVAL4( 'WKIL', WKIL, NGROUP*NREG, IFL, 1 )
28:      if ( IFL.gt.0 ) then
29:        write(IMG,*) 'XXX WKIL contains ', IFL, ' non-positive data.'
30:        call CNTERR( 'FATAL' )
31:      end if
32: C
33:      NERR      = 0
34:      do 110 IR = 1, NREG
35:        do 100 IG = 1, NGROUP
36:          if ( WSRV(IG,IR).lt.WKIL(IG,IR) ) then
37:            write(IMG,7000) IR, IG, WSRV(IG,IR), WKIL(IG,IR)
38:            NERR      = NERR + 1
39:            call CNTERR( 'FATAL' )
40:          end if
41:        100 continue
42:      110 continue
43: C
44:      return
45: C
46: 7000 format(1X,'XXX WSRV is smaller then WKIL in region ',I5,
47:    &      ', energy-group ',I4,' (WSRV=',E12.5,' WKIL=',E12.5,' )')
48: C
49:      end
```

src/shared/cmpchi.f

```
1: C/#IF MS_VISUAL
2: * subroutine CMPCHI( N,      IM,      CCC,  III )
3: C/#ELSE
4:   subroutine CMPCHI( N,      IM,      CHAR, INT )
5: C/#ENDIF
6: C==<MVP/GMVP>=====
7: C purpose: byte-wise comparison of character string and integer.
8: C called in: data packet handling routines
9: C-----
10: C arguments (i=input, o=output, w=work )
11: C i  N : number of character (bytes) compared
12: C o  IM : returns non zero when chracters does not match.
13: C i  CHAR : character string
14: C i  INT : character string (probably declared as integer in caller
15: C          size)
16: C=====
17: C/#IF MS_VISUAL
18: C
19: C ... The type of real & dummy arguments must match each other
20: C    in MS-Visual Fortran. ...
21: C
22: *   character*1 CCC(*)
23: *   integer*1   III(*)
24: *   do 100 I = 1, N
25: *     if ( char(III(I)).ne.CCC(I) ) then
26: *       IM      = 0
27: *       return
28: *     end if
29: * 100 continue
30: *     IM      = 1
31: C/#ELSE
32:   character*1 CHAR(*)
33:   character*1 INT(*)
34:   do 100 I = 1, N
35:     if ( INT(I).ne.CHAR(I) ) then
36:       IM      = 0
37:       return
38:     end if
39:   100 continue
40:   IM      = 1
41: C/#ENDIF
42:   return
43:   end
```

[illegible][illegible]

[illegible][illegible]

src/shared/cnvtox.f

```
1: c=<MVP/GMVP>=====
2: c Purpose: data type conversion functions (or "disguise" data type)
3: c called in: VARREG etc.
4: c=====
5:     function CNVTOI(X)
6:     integer CNVTOI, X
7:     CNVTOI = X
8:     return
9:     end
10: c
11:     function CNVTOR(X)
12:     real CNVTOR, X
13:     CNVTOR = X
14:     return
15:     end
16: c
17:     function CNVTOD(X)
18:     real*8 CNVTOD, X
19:     CNVTOD = X
20:     return
21:     end
```

src/shared/crsinp.f

```

1:      subroutine CRSINP( VNAME, A,      IA,      LICRES,NICRES,NSTAL, JDEBG
2:      &
3: C=<MVP/GMVP>=====
4: C  purpose: input point-wise response information.
5: C  called in: intro routine of mvp and talinp routine.
6: C-----
7: C  arguments ( i=input, o=output, w=work )
8: C io  A : task shared dynamic data area used as "REAL" type.
9: C io  IA : task shared dynamic data area used as "INTEGER" type.
10: C
11: C (a,ia must have the same address as A(*) of
12: C common /ARRAY/
13: C-----
14: C
15: C include 'INC/_IOUNIT'
16: C
17: C real A(*)
18: C integer IA(*)
19: C character(*) VNAME
20: C integer JDEBG(*)
21: C
22: C ===== local variables =====
23: C
24: C character ID*72
25: C character MT*72
26: C
27: C character*6 TAG
28: C character*12 THEAD
29: C
30: C
31: C-----
32: C ... pointer to a data packet area.
33: C
34: C LICRES = 0
35: C NICRES = 0
36: C NERR = 0
37: C
38: C if ( NSTAL.eq.0 ) then
39: C   write(IMG, '(1X,A)') '!!!(CRSINP) Number of point-wise ',
40: C   & 'response (NSTAL) is not specified.'
41: C   write(IMG,*) ' Input data are skipped.'
42: C   call CNTERR( 'WARNING' )
43: C   call DMREAD( 'STAL', NA, NB, IEND )
44: C   if ( IEND.ne.0 ) go to 140
45: C   return
46: C end if
47: C
48: C ... create data packet region.
49: C
50: C call REMAIN( 'packet', NLTEMP, 'R4D', LAST )
51: C if ( NLTEMP.le.0 ) then
52: C##<2007/03/14:PN3:
53: C## write(IMG,*)
54: C## & 'XXX no-more memory to store point-wise response information'
55: C## write(IMG,902)
56: C 902 format(/' XXX(crsinp) no-more memory to store point-wise',
57: C & ' response information.')
58: C##>
59: C call PRSTOP( 1, 'MEMORY OVER IN CRSINP.' )
60: C stop 999
61: C end if
62: C
63: C call KEEP( 'PACKET', LICRES, NLTEMP, 'R4D', LAST, JDEBG )
64: C
65: C call PACK00( A(LICRES), '&ICRES', NLTEMP, IRET )

```

```

66: C
67: C if ( IRET.ne.0 ) then
68: C##<2007/03/14:PN3:
69: C## write(IMG,*)
70: C## & 'XXX insufficient memory to make data packet area ',
71: C## & 'to process point-wise response information !!!'
72: C## write(IMG,903)
73: C 903 format(/' XXX(crsinp) insufficient memory to make data packet',
74: C & ' area to process point-wise response information !!!')
75: C##>
76: C call PRSTOP( 1, 'MEMORY SHORTAGE IN CRSINP.' )
77: C stop 999
78: C end if
79: C
80: C.....
81: C LOOP 100 : READ NUCLIDE ID and REACTION TYPE
82: C.....
83: C
84: C IRES = 0
85: C
86: C 100 call CHREAD( 'IDNAME', ID, '()', NLEN, ITERM, IEND )
87: C if ( IEND.ne.0 ) go to 130
88: C if ( ITERM.eq.1.and.NLEN.eq.0 ) go to 100
89: C if ( ITERM.eq.2 ) then
90: C   if ( NLEN.eq.0 ) then
91: C     go to 120
92: C   else
93: C.... error message
94: C##<2007/03/14:PN3:
95: C## write(IMG,*) ' XXX Unexpected "(" or ")" during reading',
96: C## & ' STAL or CRESP data.'
97: C## write(IMG,904)
98: C 904 format(/' XXX(crsinp) Unexpected "(" or ")" during reading STAL',
99: C & ' or CRESP data.')
100: C##>
101: C call PRSTOP( 1, 'UNEXPECTED ( OR ) in STAL or CRESP data.' )
102: C stop
103: C end if
104: C end if
105: C
106: C II = 0
107: C NL = 1
108: C 110 call CHREAD( ID(:NLEN), MT, '()', NLT, ITERM, IEND )
109: C if ( IEND.ne.0 ) go to 130
110: C if ( ITERM.eq.0 ) then
111: C   II = II + 1
112: C   NL = NLT
113: C   go to 110
114: C else if ( II.eq.0 ) then
115: C   II = II + 1
116: C   NL = NLT
117: C else if ( NLT.gt.0 ) then
118: C   II = II + 1
119: C end if
120: C if ( II.gt.1 ) then
121: C.... error message
122: C##<2007/03/14:PN3:
123: C## write(IMG,*) ' XXX Unexpected "(" or ")" during reading',
124: C## & ' STAL or CRESP data.'
125: C## write(IMG,904)
126: C##>
127: C call PRSTOP( 1, 'UNEXPECTED ( OR ) in STAL or CRESP data.' )
128: C stop
129: C end if
130: C

```


src/shared/crsinp.f

```
131: C
132: C=====
133: C ===== store in data packet ====
134: C=====
135: C
136:       IRES      = IRES + 1
137:       TAG       = ' '
138:       write(TAG,'(I5,','')') IRES
139:       call CCOMP( TAG, LEN(TAG), TAG, IIF )
140:       LTAG      = ICLEN(TAG)
141:       THEAD     = TAG(:LTAG) //'&ICRES'
142:       call PACKLB( A(LICRES), '&ICRES', THEAD(:ICLEN2(THEAD)), IRET )
143:       call PACKCS( A(LICRES), 'RID', ID(:NLN), IRET )
144:       call PACKCS( A(LICRES), 'MT', MT(:NL), IRET )
145: C
146:       go to 100
147: C
148: C
149: C      .... END OF INPUT
150: C
151:       120 continue
152: c##<2007/03/14:PN3:
153: c##   write(IPR,'(1x,a,a,i10)') ' <', VNAME, '> NUMBER OF DATA = ',
154: c##   &      IRES
155:       write(IPR,7010) VNAME, IRES
156:       7010 format(' <',A,'>',t25,'NUMBER OF DATA =',i13)
157: c##>
158: C
159:       call PCTCLS( A(LICRES), 'CLOSE $LICRES', NPSIZE, NICRES, IRET )
160: C
161:       if ( JDEBG(1).ne.0 ) then
162:         call PCTDMP( IPR, A(LICRES), A(LICRES), A(LICRES) )
163:       end if
164: C
165:       call PCTERR( A(LICRES), ' ', NPKERR, IRET )
166: C
167: C      ... free memory ...
168:       call RESIZE( 'packet', LICRES, NICRES, 'R4D', LAST )
169: C
170:       if ( NERR.gt.0 ) then
171: c##<2007/03/14:PN3:
172: c##   write(IMG,*) 'XXX STAL or CRESP data contains error.'
173:       write(IMG,905)
174:       905 format('/' XXX(crsinp) STAL or CRESP data contains error.')
175: c##>
176:       call CNTERR( 'FATAL' )
177:       NICRES = 0
178:       end if
179: C
180:       if ( NPKERR.gt.0 ) then
181: c##<2007/03/14:PN3:
182: c##   write(IMG,*) 'XXX Could not store all STAL or CRESP data ',
183: c##   &      'correctly in memory.'
184:       write(IMG,906)
185:       906 format('/' XXX(crsinp) Could not store all STAL or CRESP data',
186:       &      ' correctly in memory.')
187: c##>
188:       call CNTERR( 'FATAL' )
189:       NICRES = 0
190:       end if
191: C
192:       if ( IRES.ne.NSTAL ) then
193: c##<2007/03/14:PN3:
194: c##   write(IMG,*) 'XXX The number of point-wise response data (',
195: c##   &      IRES, ') is not equal to expected value (', NSTAL, ')'
```

```
196:       write(IMG,907) IRES, NSTAL
197:       907 format('/' XXX(crsinp) The number of point-wise response data (',
198:       &      I4,') is not equal to expected value (',I4,').')
199: c##>
200:       NSTAL      = IRES
201:       if ( IRES.eq.0 ) NICRES = 0
202:       call CNTERR( 'FATAL' )
203:       end if
204: C
205:       return
206: C
207: C ===== error messages ===
208: C
209:       130 continue
210: c##<2007/03/14:PN3:
211: c##   write(IMG,7000)
212: c7000 format(/1X,'XXX Unexpected end of input data during reading',
213: c##   &      ' STAL or CRESP data.')
214:       write(IMG,908)
215:       908 format('/' XXX(crsinp) Unexpected end of input data during',
216:       &      ' reading STAL or CRESP data.')
217: c##>
218:       call PRSTOP( 1, 'End of input data in STAL or CRESP data.' )
219:       stop 888
220: C
221: C
222: c##<2007/03/14:PN3:
223: c#140 write(IMG,7020)
224: c7020 format(1X,'XXX Unexpected end of data or data read error ',
225: c##   &      'during skipping unnecessary data (CRSINP)')
226:       140 write(IMG,909)
227:       909 format('/' XXX(crsinp) Unexpected end of data or data read error',
228:       &      ' during skipping unnecessary data.')
229: c##>
230:       call PRSTOP( 1, 'ERROR DURING SKIPPING STAL OR CRESP DATA.' )
231:       stop 888
232: C
233:       end
```

src/shared/crvol1.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine CRVOL1( RVOL, NREG, VOL, KREG, NINPZ, KMAT )
3: C=====
4: C PURPOSE: Calculate volume of input-zones from region volume
5: C   ( when input-zone volumes are not input ... )
6: C CALLED IN: INTRO
7: C-----
8: C arguments (i=input, o=output, w=work )
9: C i RVOL(NREG) : volume of regions
10: C o VOL(NINPZ) : volume of input-zones
11: C i KREG(NINPZ) : region # of input-zones
12: C i KMAT(NINPZ) : material # of input-zones
13: C=====
14:   real RVOL(NREG), VOL(NINPZ)
15:   integer KREG(NINPZ), KMAT(NINPZ)
16: C
17: C ..... COUNT NUMBER OF ZONES OF REAL MATERIAL FOR EACH INPUT-ZONE
18: C
19:   do 120 N = 1, NREG
20:     NZ = 0
21:     do 100 I = 1, NINPZ
22:       if ( KREG(I).eq.N.and.KMAT(I).ge.0 )then
23:         NZ = NZ + 1
24:       end if
25:     100 continue
26:     if ( NZ.gt.0 ) then
27:       do 110 I = 1, NINPZ
28:         if ( KREG(I).eq.N ) then
29:           if ( KMAT(I).ge.0 ) then
30:             VOL(I) = RVOL(N) /REAL(NZ)
31:           else
32:             VOL(I) = 0.0
33:           end if
34:         end if
35:       110 continue
36:     end if
37:   120 continue
38:   return
39: end
```

src/shared/crvol.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine CRVOL( RVOL, NREG, VOL, KREG, NINPZ )
3: C=====
4: C  PURPOSE: CALCULATE REGION VOLUMES FROM INPUT-ZONE VOLUMES
5: C  CALLED IN: INTRO
6: C
7: C  arguments (i=input, o=output, w=work )
8: C  o RVOL(NREG) : volume of regions
9: C  i NREG : number of regions
10: C  i VOL : volume of input zones
11: C  i KREG : region # of input-zones
12: C  i NINPZ : number of input-zones
13: C=====
14:   real RVOL(NREG), VOL(NINPZ)
15:   integer KREG(NINPZ)
16: C
17:   do 100 I = 1, NINPZ
18:     if ( KREG(I).gt.0.and.KREG(I).le.NREG )
19:       & RVOL(KREG(I)) = RVOL(KREG(I)) + VOL(I)
20:   100 continue
21:   return
22:   end
```

src/shared/crvolt.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine CRVOLT( TRVOL, NTREG, RVOL, NREG, IPTRG, LPTRG )
3: C=====
4: C PURPOSE: CALCULATE TALLY-REGION VOLUMES FROM REGION VOLUMES
5: C HISTORY: PROGRAMMED BY M.SASAKI (31 JUL 1992)
6: C CALLED IN: INTRO2
7: C-----
8: C arguments (i=input, o=output, w=work )
9: C o TRVOL(NTREG) : volume of tally regions
10: C i RVOL(NREG) : volume of regions
11: C i IPTRG(NTREG+1) : index to LPTRG for each tally region
12: C i LPTRG(*) : list of regions for tally regions
13: C=====
14:   real TRVOL(NTREG), RVOL(NREG)
15:   integer IPTRG(NTREG+1), LPTRG(*)
16: C
17:   do 110 KR = 1, NTREG
18:     TRVOL(KR) = 0.0
19:     do 100 MR = IPTRG(KR), IPTRG(KR+1) - 1
20:       IR = LPTRG(MR)
21:       TRVOL(KR) = TRVOL(KR) + RVOL(IR)
22:   100 continue
23:   110 continue
24:   return
25: end
```

src/shared/dframe.f

```

1:      subroutine DFRAME( IOW, MZONE, NN,JDEBG,NBANK,NEST,NLATT,NLBZ,
2:      N      NZONE, DINP, DEPS,
3:      I      X,Y,Z,AI,BI,CI,IBP,
4:      O      DI,IFC,XUP,YUP,ZUP,AUP,BUP,CUP,ILUP,ILST,
5:      B      LEVL, LZZ, LPOS, LCRS,
6:      B      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
7:      G      KZMAT, KZDA, KZAA, SDA, KCELL,
8:      G      KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT,
9:      G      KLATT, KSLAT, LTYP, MLBZZ, KZLBZ,
10:     W      KPLT, JKSF, IBP0, IBP1, IBP2,
11:     W      XX, YY, ZZ, AA, BB, CC, XX2, YY2, ZZ2, AA2, BB2, CC2,
12:     W      DDI, IKL, IKL2, DLOC1, DLOC2 )
13: C=====
14: C PURPOSE: Calculate distances to frame of lattice, from current
15: C lattice cell level to the 1st lattice level.
16: C Necessary to treat arbitrary shape of lattices and
17: C their overlap.
18: C
19: C Compare them with those to zone boundary,
20: C And calculate coordinates & directions in upper level.
21: C IFC = N/0 = CROSS THE FRAME/ NO CROSSING
22: C (N = ZONE NUMBER IN UPPER LEVEL)
23: C <<caution>>
24: C This routine cannot handle surface # of torus (IKL & IKL2)
25: C for SQSOL1 routine called in SPEAR1 routine.
26: C Torus should not be a surface to compose lattice frame.
27: C
28: C CALLED IN: FLIONE
29: C CALLS: SPEAR1
30: C-----
31: C Arguments ( i=input, o=output, w=work ) :
32: C (see description on COMMON undescribed ones)
33: C i IOW: message printout I/O unit.
34: C i MZONE: particles are in zone MZONE in current lattice level
35: C i NN : number of particles
36: C i JDEBG,NBANK,NEST,NLATT,NLBZ,NZONE,DINP,DEPS:
37: C
38: C i X(NN),Y(NN),Z(NN),AI(NN),BI(NN),CI(NN): position and directions
39: C i IBP(NN): bank pointers for X,Y,Z,AI,BI,CI,ILUP,ILST and DI.
40: C
41: C i o DI(NN) : distance to be calculated (minimum of distances to frame
42: C and incoming value to this routine.)
43: C o IFC : frame zone # if particle may reach frame boundary or zero.
44: C o XUP(NN),YUP(NN),ZUP(NN),AUP(NN),BUP(NN),CUP(NN):
45: C positions and directions in upper level.
46: C o ILUP(NN) : lattice frame level giving smallest distance.
47: C o ILST(NN) : set flag when particle is lost
48: C
49: C i LEVL(NBANK), LZZ(NBANK), LPOS(NBANK), LCRS(NBANK) :
50: C lattice-cell bank
51: C
52: C i o DBNK(NBANK,*),KDBNK(0:MDBNK),IBNK(NBANK,*),KIBNK(0:MIBNK):
53: C optional bank parameters.
54: C
55: C <<Parameters related to this routine>>
56: C
57: C IBNK(*,KIBNK(5)) : flight path count. Particles having
58: C zero value are processed in this routine.
59: C IBNK(*,KIBNK(6):KIBNK(6)+NEST-1) : lattice level giving
60: C the smallest distance each the level
61: C DBNK(*,KDBNK(4):KDBNK(4)+NEST+i-1) : distance to lattice frames
62: C of i'th level. i=NEST is reserved for
63: C some future extension.
64: C
65: C i KZMAT,KDALT,DALT,NVLAT,SZLAT,CELSZ,IPLAT,KLATT,KSLAT,LTYP,MLBZZ:

```

```

66: C geometry data.
67: C
68: C w KPLT, JKSF, IBP0, IBP1, IBP2
69: C w XX, YY, ZZ, AA, BB, CC
70: C w XX2, YY2, ZZ2, AA2, BB2, CC2
71: C w DDI, IKL, IKL2, DLOC1, DLOC2
72: C=====
73: C
74: C implicit real*8(D)
75: C
76: C integer JDEBG(*)
77: C
78: C .... DISTANCE, COORDINATES & FLAG .....
79: C
80: C real*8 X(NN), Y(NN), Z(NN), AI(NN), BI(NN), CI(NN)
81: C integer IBP(NN)
82: C
83: C real*8 DI(NN), XUP(NN), YUP(NN), ZUP(NN)
84: C real*8 AUP(NN), BUP(NN), CUP(NN)
85: C integer IFC(NN)
86: C integer ILUP(NN)
87: C integer ILST(NN)
88: C
89: C .... PARTICLE BANK .....
90: C
91: C integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST),
92: C & LCRS(NBANK,NEST)
93: C
94: C ... optional bank parameters
95: C
96: C real*8 DBNK(NBANK,*)
97: C integer KDBNK(0:MDBNK)
98: C integer IBNK(NBANK,*)
99: C integer KIBNK(0:MIBNK)
100: C
101: C .... GEOMETRY .....
102: C
103: C real*8 SDA(*)
104: C integer KZDA(2,*), KZAA(*)
105: C
106: C real*8 DALT(*), SZLAT(8,NLATT), CELSZ(6,1)
107: C integer KCELL(NZONE)
108: C integer KZMAT(NZONE,2), KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT),
109: C & NVLAT(4,NLATT), IPLAT(NLATT+1), KSLAT(*), KLATT(*)
110: C integer KZLBZ(NLBZ)
111: C
112: C .... WORK AREA .....
113: C
114: C integer KPLT(NLBZ+1),JKSF(NLBZ)
115: C integer IBP0(NBANK),IBP1(NBANK),IBP2(NBANK)
116: C real*8 XX(NBANK), YY(NBANK), ZZ(NBANK)
117: C real*8 XX2(NBANK), YY2(NBANK), ZZ2(NBANK)
118: C real*8 AA(NBANK), BB(NBANK), CC(NBANK)
119: C real*8 AA2(NBANK), BB2(NBANK), CC2(NBANK)
120: C real*8 DDI(NBANK)
121: C real*8 DLOC1(NBANK), DLOC2(NBANK)
122: C integer IKL(NBANK), IKL2(NBANK)
123: C
124: C .....
125: C
126: C common /HEXROT/ DROT
127: C real*8 DROT(0:5,2)
128: C
129: C parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
130: C

```

src/shared/dframe.f

```

131:      include 'INC/_PMLATT'
132:      include 'INC/_SFLATT'
133: C
134: C-----
135: C
136: C ... compress IBP array into IBP1 ...
137: C     IBP0 is used to point X(I) etc., in compressed context (i=1,NN2)
138: C
139: C *VOCL LOOP,NOVREC
140: C     NN2 = 0
141: C     do 10 J = 1, NN
142: C       if ( IBNK(IBP(J),KIBNK(5)).eq.0 ) then
143: C         NN2 = NN2 + 1
144: C         IBP1(NN2) = IBP(J)
145: C         IBP0(NN2) = J
146: C       end if
147: C     if ( JFC(J) = 0
148: C 10 continue
149: C     if ( NN2.eq.0 ) return
150: C
151: C     MXLV = 0
152: C     do 100 I = 1, NN2
153: C       MXLV = MAX(LEVEL(1BP1(I)),MXLV)
154: C       XX2(I) = X(1BP0(I))
155: C       YY2(I) = Y(1BP0(I))
156: C       ZZ2(I) = Z(1BP0(I))
157: C       AA2(I) = AI(1BP0(I))
158: C       BB2(I) = BI(1BP0(I))
159: C       CC2(I) = CI(1BP0(I))
160: C 100 continue
161: C
162: C     if ( MXLV.eq.0 ) return
163: C
164: C ... Promote lattice level upward .....
165: C
166: C     do 210 LV = MXLV, 1, -1
167: C
168: C ..... Sort by lattice # (lattice frame) .....
169: C
170: C     NLZ = 0
171: C     NLL = 0
172: C     KPLT(1) = 1
173: C     do 120 NL = 1, NLBZ
174: C       if( KZMAT(KZLBZ(NL),1).ne.-999) then
175: C         NLL0 = NLL
176: C *VOCL LOOP,NOVREC
177: C       do 110 I = 1, NN2
178: C         if ( LEVEL(1BP1(I)).ge.LV ) then
179: C           LZ = LZZ(1BP1(I),LV)
180: C           if ( LZ.eq.KZLBZ(NL) ) then
181: C             NLL = NLL + 1
182: C             1BP2(NLL) = I
183: C           end if
184: C         end if
185: C       continue
186: C       if ( NLL.gt.NLL0 ) then
187: C         NLZ = NLZ + 1
188: C         JKSF(NLZ) = NL
189: C         KPLT(NLZ+1) = NLL + 1
190: C       end if
191: C     end if
192: C 120 continue
193: C
194: C *VOCL LOOP,NOVREC
195: C     do 130 I = 1, NLL

```

```

196: C     XX(I) = XX2(1BP2(I))
197: C     YY(I) = YY2(1BP2(I))
198: C     ZZ(I) = ZZ2(1BP2(I))
199: C     AA(I) = AA2(1BP2(I))
200: C     BB(I) = BB2(1BP2(I))
201: C     CC(I) = CC2(1BP2(I))
202: C     DDI(I) = DINF
203: C 130 continue
204: C
205: C ..... LOOP BY LATTICE frame zone .....
206: C
207: C     do 190 NL = 1, NLZ
208: C
209: C       LZ = KZLBZ(JKSF(NL))
210: C       MLT = LATNM(KZMAT(LZ,1))
211: C       M = KDALT(JKSF(NL))
212: C
213: C ..... COORDINATE TRANSFORMATION TO IN-LATTICE COORDINATES
214: C
215: C ..... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
216: C
217: C SZLAT * TYPE 1 * TYPE 2 * TYPE 3 * TYPE 4
218: C -----
219: C 1 * CELL-SIZE(X) * CELL PITCH
220: C 2 * (Y) * ANGLE OF CELL ARRAY
221: C 3 * (Z) * CELL HEIGHT
222: C -----
223: C 4 * ... * ORIGIN OF CELL ARRAY (X)
224: C 5 * ... * ORIGIN OF CELL ARRAY (Y)
225: C -----
226: C (LATTICE FRAME SIZE)
227: C -----
228: C 6 * ... * LATTICE- * CYLINDER- * WIDTH (X)
229: C * * * WIDTH * RADIUS *
230: C 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
231: C 8 * ... * * * * WIDTH (Z)
232: C -----
233: C
234: C ===== rectangular cell lattice =====
235: C
236: C     if ( LTPY(MLT).eq.1 ) then
237: C *VOCL LOOP,NOVREC
238: C     do 140 I = KPLT(NL), KPLT(NL+1) - 1
239: C       LPS = LPOS(1BP1(1BP2(I)),LV)
240: C
241: C       IZ = LPS/NVLAT(4,MLT)
242: C       IY = (LPS-IZ*NVLAT(4,MLT)) / NVLAT(1,MLT)
243: C       IX = MOD(LPS,NVLAT(1,MLT))
244: C       KSL = KSLAT(IPLAT(MLT)+LPS)
245: C       ICEL = KLATT(IPLAT(MLT)+LPS)
246: C       IDRX = KSL/100
247: C       IDRY = (KSL-IDRX*100) / 10
248: C       IDRZ = MOD(KSL,10)
249: C
250: C ... transformation to cell array coordinates ...
251: C
252: C     DSX = SZLAT(1,MLT)
253: C     DSY = SZLAT(2,MLT)
254: C     DSZ = SZLAT(3,MLT)
255: C     ISGX = 1 - 2*IDRX
256: C     ISGY = 1 - 2*IDRY
257: C     ISGZ = 1 - 2*IDRZ
258: C     DX = ISGX*(XX(I)-CELSZ(4,ICEL)) + (IX+0.5D0)*DSX
259: C     DYY = ISGY*(YY(I)-CELSZ(5,ICEL)) + (IY+0.5D0)*DSY
260: C     DZZ = ISGZ*(ZZ(I)-CELSZ(6,ICEL)) + (IZ+0.5D0)*DSZ

```

src/shared/dframe.f

```

261:      DAA = ISGX*AA(I)
262:      DBB = ISGY*BB(I)
263:      DCC = ISGZ*CC(I)
264: C
265: C ... transformation to upper level coordinates ...
266: C
267:      DXX = DXX + DALT(M+9)
268:      DYY = DYY + DALT(M+10)
269:      DZZ = DZZ + DALT(M+11)
270:      XX(I) = DALT(M)*DXX + DALT(M+3)*DYY + DALT(M+6)*DZZ
271:      YY(I) = DALT(M+1)*DXX + DALT(M+4)*DYY + DALT(M+7)*
272:      &      DZZ
273:      ZZ(I) = DALT(M+2)*DXX + DALT(M+5)*DYY + DALT(M+8)*
274:      &      DZZ
275: C
276:      AA(I) = DALT(M)*DAA + DALT(M+3)*DBB + DALT(M+6)*DCC
277:      BB(I) = DALT(M+1)*DAA + DALT(M+4)*DBB + DALT(M+7)*
278:      &      DCC
279:      CC(I) = DALT(M+2)*DAA + DALT(M+5)*DBB + DALT(M+8)*
280:      &      DCC
281:      140 continue
282: C
283: C ===== hexagonal cell lattice =====
284: C
285:      else if ( LTYP(MLT).eq.2.and.LTYP(MLT).le.4 ) then
286: *VOCL LOOP,NOVREC
287:      do 150 I = KPLT(NL), KPLT(NL+1) - 1
288:      LPS = LPOS(1BP1(1BP2(I)),LV)
289: C
290:      IZ = LPS/NVLAT(4,MLT)
291:      IY = (LPS-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
292:      IX = MOD(LPS,NVLAT(1,MLT))
293:      KSL = KSLAT(IPLAT(MLT)+LPS)
294:      ICEL = KLATT(IPLAT(MLT)+LPS)
295:      IDRX = KSL/100
296:      IDRY = (KSL-IDRX*100) /10
297:      IDRZ = MOD(KSL,10)
298: C
299: C ... transformation to cell array coordinates ...
300: C
301:      DSX = SZLAT(1,MLT)
302:      DSZ = SZLAT(3,MLT)
303:      DXX = XX(I) - CELSZ(3,ICEL)
304:      DYY = YY(I) - CELSZ(4,ICEL)
305:      DZZ = ZZ(I) - CELSZ(5,ICEL)
306:      DX = DROT(IDRY,1)*DXX + DROT(IDRY,2)*DYY
307:      DY = -DROT(IDRY,2)*DXX + DROT(IDRY,1)*DYY
308:      DXX = (1-2*IDRX)*DX + (IX+0.5D0*IY)*DSX
309:      DYY = DY + DHRT3*IY*DSX
310:      DZZ = (1-2*IDRZ)*DZZ + (IZ+0.5D0)*DSZ
311:      DA = AA(I)
312:      DB = BB(I)
313:      DAA = (1-2*IDRX)*(DROT(IDRY,1)*DA+DROT(IDRY,2)*DB)
314:      DBB = -DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
315:      DCC = (1-2*IDRZ)*CC(I)
316: C
317: C ... transformation to upper level coordinates ...
318: C
319:      DXX = DXX + DALT(M+9)
320:      DYY = DYY + DALT(M+10)
321:      DZZ = DZZ + DALT(M+11)
322:      XX(I) = DALT(M)*DXX + DALT(M+3)*DYY + DALT(M+6)*DZZ
323:      YY(I) = DALT(M+1)*DXX + DALT(M+4)*DYY + DALT(M+7)*
324:      &      DZZ
325:      ZZ(I) = DALT(M+2)*DXX + DALT(M+5)*DYY + DALT(M+8)*
326:      &      DZZ
327: C
328:      AA(I) = DALT(M)*DAA + DALT(M+3)*DBB + DALT(M+6)*DCC
329:      BB(I) = DALT(M+1)*DAA + DALT(M+4)*DBB + DALT(M+7)*
330:      &      DCC
331:      CC(I) = DALT(M+2)*DAA + DALT(M+5)*DBB + DALT(M+8)*
332:      &      DCC
333:      150 continue
334: C
335: C ===== STG region as lattice =====
336: C
337:      else if ( LTYP(MLT).eq.10 ) then
338:      K5X = KDBNK(5) + (LV-1)*3
339:      K5Y = K5X + 1
340:      K5Z = K5X + 2
341: *VOCL LOOP,NOVREC
342:      do 160 I = KPLT(NL), KPLT(NL+1) - 1
343: C
344: C ... no transformation is required if not in STG cell zones
345: C
346:      LPS = LPOS(1BP1(1BP2(I)),LV)
347:      if ( LPS.gt.0 ) then
348:      ICEL = KLATT(IPLAT(MLT)+LPS)
349:      XX(I) = XX(I) - CELSZ(1,ICEL)
350:      &      + DBNK(1BP1(1BP2(I)),K5X)
351:      YY(I) = YY(I) - CELSZ(2,ICEL)
352:      &      + DBNK(1BP1(1BP2(I)),K5Y)
353:      ZZ(I) = ZZ(I) - CELSZ(3,ICEL)
354:      &      + DBNK(1BP1(1BP2(I)),K5Z)
355:      end if
356:      160 continue
357:      end if
358: C
359: C .... distance to current zone boundary ....
360: C
361:      K1 = KPLT(NL)
362:      N1 = KPLT(NL+1) - KPLT(NL)
363: C
364:      JSS = 0
365:      call SPEAR1( LZ, KZDA, KZAA, SDA, N1, NBANK, JSS,
366:      &      XX(K1), YY(K1), ZZ(K1), AA(K1), BB(K1), CC(K1),
367:      &      DDI(K1), DLOC1(K1),
368:      &      DLOC2(K1), IKL(K1), IKL2(K1), DUMMY1,
369:      &      DINF, DEPS )
370: C
371:      KD4 = KDBNK(4) + LV - 1
372:      KD6 = KDBNK(6) + 6*(LV-1)
373:      NLST = 0
374: *VOCL LOOP,NOVREC
375:      do 170 I = KPLT(NL), KPLT(NL+1) - 1
376: C
377: C ... store distance to frame in level LV ...
378: C
379:      IK = 1BP1(1BP2(I))
380:      DBNK(IK,KD4) = DDI(I)
381:      DBNK(IK,KD6) = XX(I) + AA(I)*DDI(I)
382:      DBNK(IK,KD6+1) = YY(I) + BB(I)*DDI(I)
383:      DBNK(IK,KD6+2) = ZZ(I) + CC(I)*DDI(I)
384:      DBNK(IK,KD6+3) = AA(I)
385:      DBNK(IK,KD6+4) = BB(I)
386:      DBNK(IK,KD6+5) = CC(I)
387: C
388: C .... check lost particle
389: C
390:      if ( DDI(I).eq.DINF ) then

```

src/shared/dframe.f

```

391:          NLST      = NLST + 1
392:          end if
393: C
394: C      ... check distance and set possible "upper level"
395: C      corrodinates and directions.
396: C
397:          II = IBP0(IBP2(I))
398:          D2  = DI(II)
399:          if ( DDI(I).lt.D2 ) then
400:              IFC(II) = LZ
401:              ILUP(II) = LV - 1
402:              DI(II) = DDI(I)
403:              XUP(II) = XX(I)
404:              YUP(II) = YY(I)
405:              ZUP(II) = ZZ(I)
406:              AUP(II) = AA(I)
407:              BUP(II) = BB(I)
408:              CUP(II) = CC(I)
409:          end if
410: 170      continue
411: C
412: C      ... lost ... add 16 to ILST
413: C
414:          if ( NLST.gt.0 ) then
415:              MLT = LATNM(KZMAT(LZ,1))
416:              write(IOW,7000) NLST, MZONE, LZ, LV, MLT, LTPY(MLT)
417:              do 180 I = KPLN(NL), KPLT(NL+1) - 1
418:                  if ( DDI(I).eq.DINF ) then
419:                      II = IBP0(IBP2(I))
420:                      write(IOW,7020) IBP(II), XX(I), YY(I), ZZ(I),
421:                          & AA(I), BB(I), CC(I), X(II), Y(II), Z(II),
422:                          & AI(II), BI(II), CI(II)
423:                      if ( ILST(II).eq.0 .or. MOD(ILST(II)/16,2).eq.0 )
424:                          & then
425:                              ILST(II) = ILST(II) + 16
426:                          end if
427:                      end if
428: 180      continue
429:          end if
430: 7000      format(1X,'XXX(DFRAME) ',I5,' particles are lost in',
431:          & ' calculation of distance to frame. Zone:',I5,
432:          & ', Frame zone:',I5,
433:          & ', Level:',I3
434:          & '/15X,' Lattice #:',I5,' Type: ',I3/
435:          & 1X,2X,' No. ', 'XX',12X,'YY',12X,'ZZ',
436:          & 12X,'AA',12X,'BB',12X,'CC'/1X,2X,5X,'X',13X,'Y',13X,
437:          & 'Z',13X,'AI',12X,'BI',12X,'CI')
438: 7020      format(1X,I5,1P,6E14.6/1X,5X,1P,6E14.6)
439: C
440: 190      continue
441: C
442:          if ( LV.gt.1 ) then
443: *VOCL LOOP,NOVREC
444:              do 200 I = 1, NLL
445:                  XX2(IBP2(I)) = XX(I)
446:                  YY2(IBP2(I)) = YY(I)
447:                  ZZ2(IBP2(I)) = ZZ(I)
448:                  AA2(IBP2(I)) = AA(I)
449:                  BB2(IBP2(I)) = BB(I)
450:                  CC2(IBP2(I)) = CC(I)
451: 200      continue
452:          end if
453: C
454: 210 continue
455: C

```

```

456: C
457: C      ... set level # giving smallest frame distance in each level ...
458: C
459:          KI6 = KIBNK(6)
460: *VOCL LOOP,NOVREC
461:          do 220 I = 1, NN2
462:              if ( LEVL(IBP1(I)).gt.0 ) then
463:                  IBNK(IBP1(I),KI6) = 1
464:              end if
465: 220      continue
466: C
467:          do 240 LV = 2, MXLV
468:              KD4 = KDBNK(4) + LV - 1
469:              KI6 = KIBNK(6) + LV - 1
470:              KD41 = KDBNK(4) - 1
471: *VOCL LOOP,NOVREC
472:              do 230 I = 1, NN2
473:                  if ( LEVL(IBP1(I)).ge.LV )
474:                      & then
475:                          IBNK(IBP1(I),KI6) = LV
476:                          if ( DBNK(IBP1(I),KD4).ge.
477:                              & DBNK(IBP1(I),KD41+IBNK(IBP1(I),KI6-1))
478:                              & ) then
479:                              IBNK(IBP1(I),KI6) = IBNK(IBP1(I),KI6-1)
480:                          end if
481:                      end if
482: 230      continue
483: 240      continue
484: C
485:          return
486:          end

```


src/shared/dotmax.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine DOTMAX( NDMAX, NPICT, PAPER )
3: C=====
4: C PURPOSE:  FIND MAXIMUM NUMBER OF DOTS IN PICTURES.
5: C CALLED IN: PABRO
6: C CALLS:    (NONE)
7: C=====
8:   real PAPER(12,NPICT)
9:   NDMAX   = 0
10:  do 100 N = 1, NPICT
11:     NDMAX = MAX(NDMAX,INT(PAPER(10,N)*PAPER(11,N)))
12: 100 continue
13:  return
14:  end
```

src/shared/feynmx.f

```

1:      subroutine FEYNM0( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2 )
2: C=====
3: C what to do: edit Feynman-Y tally having 0 dimensions. no meaning
4: C called in : STLSUM
5: C=====
6:      implicit real*8(A-H,O-Z)
7: C
8:      real*8 DWRK(1)
9:      real*8 DWRK2(1)
10: C
11:      real*8 ETALY(1)
12:      real*8 SETALY(1)
13:      real*8 DTALY(1)
14: C
15:      include 'INC/_IOUNIT'
16: C
17: C-----
18: C
19: C
20:      DNG = 1.0D0/DBLE(NGENE)
21:      ETALY(1) = ETALY(1) + DTALY(1)
22:      SETALY(1) = SETALY(1) + DTALY(1)**2*DNG
23: C
24:      return
25:      end
26: C
27: C-----
28: C
29:      subroutine FEYNM1( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2,
30:      &      NE1, ND1, IEDT1 )
31: C=====
32: C what to do: edit Feynman-Y tally having 1 dimensions.
33: C called in : stlsum
34: C=====
35:      implicit real*8(A-H,O-Z)
36: C
37:      real*8 DWRK(ND1)
38:      real*8 DWRK2(NE1)
39: C
40:      real*8 ETALY(NE1)
41:      real*8 SETALY(NE1)
42:      real*8 DTALY(ND1)
43: C
44:      integer IEDT1(*)
45: C
46:      include 'INC/_IOUNIT'
47: C-----
48: C      do 100 N1 = 1, ND1
49: C          DWRK(N1) = 0.0D0
50: C      100 continue
51: C      do 110 N1 = 1, NE1
52: C          DWRK2(N1) = 0.0D0
53: C      110 continue
54: C
55: C
56:      L1 = 1
57:      M1 = IEDT1(L1+1)
58: C      if ( M1.lt.0 ) then
59: C          do 120 N1 = 1, ND1
60: C              DWRK(N1) = DTALY(N1)
61: C          120 continue
62: C
63: C
64: C      check
65: C      write(6,*) ' ND1,NE1,ngene,sum=',ND1,ne1,ngene,sum

```

```

66: C      write(6,*) ' dwrk=',(dwrk(i),i=1,10)
67: C      IWAR = 0
68: C      do 1000 N1 = 1, NE1
69: C          NN = ND1 / N1
70: C          NDE = NN * N1
71: C
72: C          X = 0.0D0
73: C          X2 = 0.0D0
74: C          do 1010 I = 1, NDE
75: C              X = X + DWRK(I)
76: C          1010 continue
77: C          if ( N1.eq.1 ) then
78: C              do 1020 I = 1, NDE
79: C                  X2 = X2 + DWRK(I)**2
80: C              1020 continue
81: C          else if ( N1.eq.2 ) then
82: C              do 1030 I = 1, NDE, 2
83: C                  X2 = X2 + (DWRK(I)+DWRK(I+1))**2
84: C              1030 continue
85: C          else if ( N1.eq.3 ) then
86: C              do 1040 I = 1, NDE, 3
87: C                  X2 = X2 + (DWRK(I)+DWRK(I+1)+DWRK(I+2))**2
88: C              1040 continue
89: C          else
90: C              do 1050 I = 1, NDE, N1
91: C                  T = 0
92: C                  do 1060 J = 0, N1-1
93: C                      T = T + DWRK(I+J)
94: C                  1060 continue
95: C                  X2 = X2 + T**2
96: C              1050 continue
97: C          end if
98: C      check
99: C      write(6,*) ' n1,nde,nn,X,X2=',n1,nde,nn,x,x2
100: C
101: C          X = X / DBLE(NN)
102: C          X2 = X2 / DBLE(NN)
103: C          VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
104: C          if( X.gt.0.d0 ) then
105: C              DWRK2(N1) = VAR / X - 1.D0
106: C          else
107: C              DWRK2(N1) = 0.d0
108: C              IWAR = 1
109: C          end if
110: C      check
111: C      write(6,*) ' X,X2,VARI,Y',x,x2,var,dwrk2(n1)
112: C      1000 continue
113: C
114: C          if( IWAR.eq.1 ) then
115: C              write(IOW,'(a,a,a)') ' !!! Warning !!! ',
116: C              &      ' No counts for Feynman-A bins.',
117: C              &      ' Feynman-Y value has set to 0.'
118: C          end if
119: C
120: C
121: C      DNG = DBLE(NGENE)
122: C      do 2000 N1 = 1, NE1
123: C          ETALY(N1) = ETALY(N1) + DWRK2(N1)*DNG
124: C          SETALY(N1) = SETALY(N1) + DWRK2(N1)**2*DNG
125: C      2000 continue
126: C
127: C      return
128: C      end
129: C
130: C-----

```

src/shared/feynm.f

```

131: C
132:      subroutine FEYNM2( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2,
133:      &      NE1, ND1, IEDT1,
134:      &      NE2, ND2, IEDT2 )
135: C=====
136: C what to do: edit Feynman-Y tally having 2 dimensions.
137: C called in : stlsum
138: C=====
139:      implicit real*8(A-H,O-Z)
140: C
141:      real*8 DWRK(ND2,ND1)
142:      real*8 DWRK2(NE2,NE1)
143: C
144:      real*8 ETALY(NE2,NE1)
145:      real*8 SETALY(NE2,NE1)
146:      real*8 DTALY(ND2,ND1)
147: C
148:      integer IEDT1(*)
149:      integer IEDT2(*)
150: C
151:      include 'INC/_IUNIT'
152: C-----
153:      do 100 N1 = 1, ND1
154:      do 110 N2 = 1, ND2
155:          DWRK(N2,N1) = 0.0D0
156:      110 continue
157:      100 continue
158: C      do 120 N1 = 1, NE1
159: C      do 130 N2 = 1, NE2
160: C          DWRK2(N2,N1) = 0.0D0
161: C      130 continue
162: C      120 continue
163: C
164:      ITBIN = 0
165:      if( IEDT1(2).lt.0 ) then
166:          ITBIN = 1
167:      else if( IEDT2(2).lt.0 ) then
168:          ITBIN = 2
169:      end if
170: C
171: C
172:      IWAR = 0
173:      if ( ITBIN.eq.1 ) then
174:          do 200 N1 = 1, ND1
175:              L2 = 1
176:              do 210 N2 = 1, NE2
177:                  do 220 I2 = 1, IEDT2(L2)
178:                      M2 = IEDT2(L2+I2)
179: C-----
180:                      DWRK(N2,N1) = DWRK(N2,N1) + DTALY(M2,N1)
181: C-----
182: C      220 continue
183: C          L2 = L2 + IEDT2(L2) + 1
184: C      210 continue
185: C      200 continue
186: C
187:      do 1100 N2 = 1, NE2
188:      do 1000 N1 = 1, NE1
189:          NN = ND1 / N1
190:          NDE = NN * N1
191: C
192:          X = 0.0D0
193:          X2 = 0.0D0
194:          do 1010 I = 1, NDE
195:              X = X + DWRK(N2,I)

```

```

196:      1010 continue
197:      if ( N1.eq.1 ) then
198:          do 1020 I = 1, NDE
199:              X2 = X2 + DWRK(N2,I)**2
200:      1020 continue
201:      else if ( N1.eq.2 ) then
202:          do 1030 I = 1, NDE, 2
203:              X2 = X2 + (DWRK(N2,I)+DWRK(N2,I+1))**2
204:      1030 continue
205:      else if ( N1.eq.3 ) then
206:          do 1040 I = 1, NDE, 3
207:              X2 = X2 +
208:      &      (DWRK(N2,I)+DWRK(N2,I+1)+DWRK(N2,I+2))**2
209:      1040 continue
210:      else
211:          do 1050 I = 1, NDE, N1
212:              T = 0
213:              do 1060 J = 0, N1-1
214:                  T = T + DWRK(N2,I+J)
215:      1060 continue
216:                  X2 = X2 + T**2
217:      1050 continue
218:      end if
219: C
220:          X = X / DBLE(NN)
221:          X2 = X2 / DBLE(NN)
222:          VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
223:          if( X.gt.0.d0 ) then
224:              DWRK2(N2,N1) = VAR / X - 1.D0
225:          else
226:              DWRK2(N2,N1) = 0.d0
227:              IWAR = 1
228:          end if
229:      1000 continue
230:      1100 continue
231: C
232:      else if ( ITBIN.eq.2 ) then
233:          L1 = 1
234:          do 230 N1 = 1, NE1
235:              do 240 I1 = 1, IEDT1(L1)
236:                  M1 = IEDT1(L1+I1)
237:                  do 250 N2 = 1, ND2
238: C-----
239:                      DWRK(N2,N1) = DWRK(N2,N1) + DTALY(N2,M1)
240: C-----
241: C      250 continue
242: C      240 continue
243: C          L1 = L1 + IEDT1(L1) + 1
244: C      230 continue
245: C
246:      do 1300 N1 = 1, NE1
247:      do 1200 N2 = 1, NE2
248:          NN = ND2 / N2
249:          NDE = NN * N2
250: C
251:          X = 0.0D0
252:          X2 = 0.0D0
253:          do 1210 I = 1, NDE
254:              X = X + DWRK(I,N1)
255:      1210 continue
256:          if ( N2.eq.1 ) then
257:              do 1220 I = 1, NDE
258:                  X2 = X2 + DWRK(I,N1)**2
259:      1220 continue
260:          else if ( N2.eq.2 ) then

```

src/shared/feynmx.f

```

261:      do 1230 I = 1, NDE, 2
262:        X2 = X2 + (DWRK(I,N1)+DWRK(I+1,N1))*2
263:      1230      continue
264:      else if ( N2.eq.3 ) then
265:        do 1240 I = 1, NDE, 3
266:          X2 = X2 +
267:            &      (DWRK(I,N1)+DWRK(I+1,N1)+DWRK(I+2,N1))*2
268:      1240      continue
269:      else
270:        do 1250 I = 1, NDE, N2
271:          T = 0
272:          do 1260 J = 0, N2-1
273:            T = T + DWRK(I+J,N1)
274:      1260      continue
275:          X2 = X2 + T*2
276:      1250      continue
277:      end if
278: C
279:      X      = X / DBLE(NN)
280:      X2     = X2 / DBLE(NN)
281:      VAR    = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
282:      if( X.gt.0.d0 ) then
283:        DWRK2(N2,N1) = VAR / X - 1.D0
284:      else
285:        DWRK2(N2,N1) = 0.d0
286:        IWAR = 1
287:      end if
288: 1200      continue
289: 1300      continue
290: C
291:      end if
292: C
293:      if( IWAR.eq.1 ) then
294:        write(IOW,'(a,a,a)' ) ' !!! Warning !!! ',
295:        &      ' No counts for Feynman-A bins.',
296:        &      ' Feynman-Y value has set to 0.'
297:      end if
298: C
299: C
300:      DNG    = DBLE(NGENE)
301:      do 2000 N1 = 1, NE1
302:        do 2010 N2 = 1, NE2
303:          ETALY(N2,N1) = ETALY(N2,N1) + DWRK2(N2,N1)*DNG
304:          SETALY(N2,N1) = SETALY(N2,N1) + DWRK2(N2,N1)**2*DNG
305: 2010      continue
306: 2000      continue
307: C
308:      return
309:      end
310: C
311: C-----
312: C
313:      subroutine FEYNM3( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2,
314:      &      NE1, ND1, IEDT1,
315:      &      NE2, ND2, IEDT2,
316:      &      NE3, ND3, IEDT3
317:      &      )
318: C=====
319: C what to do: edit Feynman-Y tally having 3 dimensions.
320: C called in : stlsum
321: C=====
322:      implicit real*8(A-H,O-Z)
323: C
324:      real*8 DWRK(ND3,ND2,ND1)
325:      real*8 DWRK2(NE3,NE2,NE1)

```

```

326: C
327:      real*8 ETALY(NE3,NE2,NE1)
328:      real*8 SETALY(NE3,NE2,NE1)
329:      real*8 DTALY(ND3,ND2,ND1)
330: C
331:      integer IEDT1(*)
332:      integer IEDT2(*)
333:      integer IEDT3(*)
334: C
335:      include 'INC/_IOUNIT'
336: C-----
337:      do 100 N1 = 1, ND1
338:        do 110 N2 = 1, ND2
339:          do 120 N3 = 1, ND3
340:            DWRK(N3,N2,N1) = 0.0D0
341:      120      continue
342:      110      continue
343:      100      continue
344: C
345:      do 130 N1 = 1, NE1
346:        do 140 N2 = 1, NE2
347:          do 150 N3 = 1, NE3
348:            DWRK2(N3,N2,N1) = 0.0D0
349:          150      continue
350:        140      continue
351:      130      continue
352: C
353:      ITBIN = 0
354:      if( IEDT1(2).lt.0 ) then
355:        ITBIN = 1
356:      else if( IEDT2(2).lt.0 ) then
357:        ITBIN = 2
358:      else if( IEDT3(2).lt.0 ) then
359:        ITBIN = 3
360:      end if
361: C
362:      IWAR = 0
363:      if ( ITBIN.eq.1 ) then
364:        do 200 N1 = 1, ND1
365:          C
366:          L2      = 1
367:          do 210 N2 = 1, NE2
368:            do 220 I2 = 1, IEDT2(L2)
369:              M2      = IEDT2(L2+I2)
370:          C
371:          L3      = 1
372:          do 230 N3 = 1, NE3
373:            do 240 I3 = 1, IEDT3(L3)
374:              M3      = IEDT3(L3+I3)
375:          C-----
376:              DWRK(N3,N2,N1) = DWRK(N3,N2,N1)
377:              &      + DTALY(M3,M2,N1)
378:          C-----
379:          240      continue
380:          L3      = L3 + IEDT3(L3) + 1
381:          230      continue
382:          C
383:          220      continue
384:          L2      = L2 + IEDT2(L2) + 1
385:          210      continue
386:          C
387:          200      continue
388:          C
389:          do 1100 N3 = 1, NE3
390:            do 1110 N2 = 1, NE2

```

src/shared/feynm.f

```

391:      do 1000 N1 = 1, NE1
392:      NN = ND1 / N1
393:      NDE = NN * N1
394: C
395:      X = 0.0D0
396:      X2 = 0.0D0
397:      do 1010 I = 1, NDE
398:      X = X + DWRK(N3,N2,I)
399: 1010 continue
400:      if ( N1.eq.1 ) then
401:      do 1020 I = 1, NDE
402:      X2 = X2 + DWRK(N3,N2,I)**2
403: 1020 continue
404:      else if ( N1.eq.2 ) then
405:      do 1030 I = 1, NDE, 2
406:      X2 = X2 + (DWRK(N3,N2,I)+DWRK(N3,N2,I+1))**2
407: 1030 continue
408:      else if ( N1.eq.3 ) then
409:      do 1040 I = 1, NDE, 3
410:      X2 = X2 +
411:      & (DWRK(N3,N2,I)+DWRK(N3,N2,I+1)+DWRK(N3,N2,I+2))**2
412: 1040 continue
413:      else
414:      do 1050 I = 1, NDE, N1
415:      T = 0
416:      do 1060 J = 0, N1-1
417:      T = T + DWRK(N3,N2,I+J)
418: 1060 continue
419:      X2 = X2 + T**2
420: 1050 continue
421:      end if
422: C
423:      X = X / DBLE(NN)
424:      X2 = X2 / DBLE(NN)
425:      VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
426:      if( X.gt.0.d0 ) then
427:      DWRK2(N3,N2,N1) = VAR / X - 1.D0
428:      else
429:      DWRK2(N3,N2,N1) = 0.d0
430:      IWAR = 1
431:      end if
432: 1000 continue
433: 1110 continue
434: 1100 continue
435: C
436:      else if ( ITBIN.eq.2 ) then
437:      L1 = 1
438:      do 250 N1 = 1, NE1
439:      do 260 I1 = 1, IEDT1(L1)
440:      M1 = IEDT1(L1+I1)
441: C
442:      do 270 N2 = 1, ND2
443: C
444:      L3 = 1
445:      do 280 N3 = 1, NE3
446:      do 290 I3 = 1, IEDT3(L3)
447:      M3 = IEDT3(L3+I3)
448: C-----
449:      DWRK(N3,N2,N1) = DWRK(N3,N2,N1)
450:      & + DTALY(M3,N2,M1)
451: C-----
452: 290 continue
453:      L3 = L3 + IEDT3(L3) + 1
454: 280 continue
455: C
456: 270 continue
457: C
458: 260 continue
459:      L1 = L1 + IEDT1(L1) + 1
460: 250 continue
461: C
462:      do 1300 N3 = 1, NE3
463:      do 1310 N1 = 1, NE1
464:      do 1200 N2 = 1, NE2
465:      NN = ND2 / N2
466:      NDE = NN * N2
467: C
468:      X = 0.0D0
469:      X2 = 0.0D0
470:      do 1210 I = 1, NDE
471:      X = X + DWRK(N3,I,N1)
472: 1210 continue
473:      if ( N2.eq.1 ) then
474:      do 1220 I = 1, NDE
475:      X2 = X2 + DWRK(N3,I,N1)**2
476: 1220 continue
477:      else if ( N2.eq.2 ) then
478:      do 1230 I = 1, NDE, 2
479:      X2 = X2 + (DWRK(N3,I,N1)+DWRK(N3,I+1,N1))**2
480: 1230 continue
481:      else if ( N2.eq.3 ) then
482:      do 1240 I = 1, NDE, 3
483:      X2 = X2 +
484:      & (DWRK(N3,I,N1)+DWRK(N3,I+1,N1)+DWRK(N3,I+2,N1))**2
485: 1240 continue
486:      else
487:      do 1250 I = 1, NDE, N2
488:      T = 0
489:      do 1260 J = 0, N2-1
490:      T = T + DWRK(N3,I+J,N1)
491: 1260 continue
492:      X2 = X2 + T**2
493: 1250 continue
494:      end if
495: C
496:      X = X / DBLE(NN)
497:      X2 = X2 / DBLE(NN)
498:      VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
499:      if( X.gt.0.d0 ) then
500:      DWRK2(N3,N2,N1) = VAR / X - 1.D0
501:      else
502:      DWRK2(N3,N2,N1) = 0.d0
503:      IWAR = 1
504:      end if
505: 1200 continue
506: 1310 continue
507: 1300 continue
508: C
509:      else if ( ITBIN.eq.3 ) then
510:      L1 = 1
511:      do 300 N1 = 1, NE1
512:      do 310 I1 = 1, IEDT1(L1)
513:      M1 = IEDT1(L1+I1)
514: C
515:      L2 = 1
516:      do 320 N2 = 1, NE2
517:      do 330 I2 = 1, IEDT2(L2)
518:      M2 = IEDT2(L2+I2)
519:      do 340 N3 = 1, ND3
520: C-----

```

src/shared/feynmx.f

```

521:                DWRK(N3,N2,N1) = DWRK(N3,N2,N1)
522:                &                + DTALY(N3,M2,M1)
523: C-----
524: 340                continue
525: C
526: 330                continue
527:                L2      = L2 + IEDT2(L2) + 1
528: 320                continue
529: C
530: 310                continue
531:                L1      = L1 + IEDT1(L1) + 1
532: 300                continue
533: C
534:                do 1500 N1 = 1, NE1
535:                do 1510 N2 = 1, NE2
536:                do 1400 N3 = 1, NE3
537:                    NN = N1 * N2 / N3
538:                    NDE = NN * N3
539: C
540:                    X = 0.0D0
541:                    X2 = 0.0D0
542:                    do 1410 I = 1, NDE
543:                        X = X + DWRK(I,N2,N1)
544:                    continue
545:                    if ( N3.eq.1 ) then
546:                        do 1420 I = 1, NDE
547:                            X2 = X2 + DWRK(I,N2,N1)**2
548:                        continue
549:                    else if ( N3.eq.2 ) then
550:                        do 1430 I = 1, NDE, 2
551:                            X2 = X2 + (DWRK(I,N2,N1)+DWRK(I+1,N2,N1))**2
552:                        continue
553:                    else if ( N3.eq.3 ) then
554:                        do 1440 I = 1, NDE, 3
555:                            X2 = X2 +
556:                                (DWRK(I,N2,N1)+DWRK(I+1,N2,N1)+DWRK(I+2,N2,N1))**2
557:                        continue
558:                    else
559:                        do 1450 I = 1, NDE, N3
560:                            T = 0
561:                            do 1460 J = 0, N3-1
562:                                T = T + DWRK(I+J,N2,N1)
563:                            continue
564:                            X2 = X2 + T**2
565:                        continue
566:                    end if
567: C
568:                    X      = X / DBLE(NN)
569:                    X2     = X2 / DBLE(NN)
570:                    VAR    = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
571:                    if( X.gt.0.d0 ) then
572:                        DWRK2(N3,N2,N1) = VAR / X - 1.D0
573:                    else
574:                        DWRK2(N3,N2,N1) = 0.d0
575:                    IWAR = 1
576:                    end if
577:                1400        continue
578:                1510        continue
579:                1500        continue
580: C
581:                end if
582: C
583:                if( IWAR.eq.1 ) then
584:                    write(IOW,'(a,a,a)' ) ' !!! Warning !!! ',
585:                    &                ' No counts for Feynman-A bins.',

```

```

586:                &                ' Feynman-Y value has set to 0.'
587:                end if
588: C
589: C
590: C
591:                DNG      = DBLE(NGENE)
592:                do 2000 N1 = 1, NE1
593:                    do 2010 N2 = 1, NE2
594:                        do 2020 N3 = 1, NE3
595:                            ETALY(N3,N2,N1) = ETALY(N3,N2,N1) + DWRK2(N3,N2,N1)*DNG
596:                            SETALY(N3,N2,N1) = SETALY(N3,N2,N1) + DWRK2(N3,N2,N1)**
597:                                &                2*DNG
598:                2020        continue
599:                2010        continue
600:                2000        continue
601: C
602:                return
603:                end
604: C
605: C-----
606: C
607:                subroutine FEYNM4( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2,
608:                &                NE1, ND1, IEDT1,
609:                &                NE2, ND2, IEDT2,
610:                &                NE3, ND3, IEDT3,
611:                &                NE4, ND4, IEDT4 )
612: C=====
613: C what to do: edit Feynman-Y tally having 4 dimensions.
614: C called in : stlsum
615: C=====
616:                implicit real*8(A-H,O-Z)
617: C
618:                real*8 DWRK(ND4,ND3,ND2,ND1)
619:                real*8 DWRK2(NE4,NE3,NE2,NE1)
620: C
621:                real*8 ETALY(NE4,NE3,NE2,NE1)
622:                real*8 SETALY(NE4,NE3,NE2,NE1)
623:                real*8 DTALY(ND4,ND3,ND2,ND1)
624: C
625:                integer IEDT1(*)
626:                integer IEDT2(*)
627:                integer IEDT3(*)
628:                integer IEDT4(*)
629: C
630:                include 'INC/_IOUNIT'
631: C-----
632:                do 100 N1 = 1, ND1
633:                    do 110 N2 = 1, ND2
634:                        do 120 N3 = 1, ND3
635:                            do 130 N4 = 1, ND4
636:                                DWRK(N4,N3,N2,N1) = 0.0D0
637:                            130                continue
638:                        120                continue
639:                    110                continue
640:                100                continue
641: C
642:                ITBIN = 0
643:                if( IEDT1(2).lt.0 ) then
644:                    ITBIN = 1
645:                else if( IEDT2(2).lt.0 ) then
646:                    ITBIN = 2
647:                else if( IEDT3(2).lt.0 ) then
648:                    ITBIN = 3
649:                else if( IEDT4(2).lt.0 ) then
650:                    ITBIN = 4

```

src/shared/feynmx.f

```

651:      end if
652: C
653: C
654:      IWAR = 0
655:      if ( ITBIN.eq.1 ) then
656:        do 200 N1 = 1, ND1
657: C
658:          L2      = 1
659:          do 210 N2 = 1, NE2
660:            do 220 I2 = 1, IEDT2(L2)
661:              M2      = IEDT2(L2+I2)
662: C
663:              L3      = 1
664:              do 230 N3 = 1, NE3
665:                do 240 I3 = 1, IEDT3(L3)
666:                  M3      = IEDT3(L3+I3)
667: C
668:                  L4      = 1
669:                  do 250 N4 = 1, NE4
670:                    do 260 I4 = 1, IEDT4(L4)
671:                      M4      = IEDT4(L4+I4)
672: C-----
673:                      DWRK(N4,N3,N2,N1) = DWRK(N4,N3,N2,N1)
674:                      &
675: C-----
676:                      continue
677:                      L4 = L4 + IEDT4(L4) + 1
678: 250      continue
679: C
680:          240      continue
681:          L3      = L3 + IEDT3(L3) + 1
682: 230      continue
683: C
684:          220      continue
685:          L2      = L2 + IEDT2(L2) + 1
686: 210      continue
687: C
688:          200      continue
689: C
690:          do 1100 N4 = 1, NE4
691:            do 1110 N3 = 1, NE3
692:              do 1120 N2 = 1, NE2
693:                do 1000 N1 = 1, NE1
694:                  NN = ND1 / N1
695:                  NDE = NN * N1
696: C
697:                  X = 0.0D0
698:                  X2 = 0.0D0
699:                  do 1010 I = 1, NDE
700:                    X = X + DWRK(N4,N3,N2,I)
701: 1010      continue
702:                  if ( N1.eq.1 ) then
703:                    do 1020 I = 1, NDE
704:                      X2 = X2 + DWRK(N4,N3,N2,I)**2
705: 1020      continue
706:                  else if ( N1.eq.2 ) then
707:                    do 1030 I = 1, NDE, 2
708:                      X2 = X2 +
709:                      &
710:                      (DWRK(N4,N3,N2,I)+DWRK(N4,N3,N2,I+1))**2
711: 1030      continue
712:                  else if ( N1.eq.3 ) then
713:                    do 1040 I = 1, NDE, 3
714:                      X2 = X2 +
715:                      &

```

```

716: 1040      continue
717:      else
718:        do 1050 I = 1, NDE, N1
719:          T = 0
720:          do 1060 J = 0, N1-1
721:            T = T + DWRK(N4,N3,N2,I+J)
722: 1060      continue
723:            X2 = X2 + T**2
724: 1050      continue
725:          end if
726: C
727:          X      = X / DBLE(NN)
728:          X2      = X2 / DBLE(NN)
729:          VAR      = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
730:          if( X.gt.0.d0 ) then
731:            DWRK2(N4,N3,N2,N1) = VAR / X - 1.D0
732:          else
733:            DWRK2(N4,N3,N2,N1) = 0.d0
734:            IWAR = 1
735:          end if
736: 1000      continue
737: 1120      continue
738: 1110      continue
739: 1100      continue
740: C
741:      else if ( ITBIN.eq.2 ) then
742:        L1      = 1
743:        do 270 N1 = 1, NE1
744:          do 280 I1 = 1, IEDT1(L1)
745:            M1      = IEDT1(L1+I1)
746: C
747:            do 290 N2 = 1, ND2
748: C
749:              L3      = 1
750:              do 300 N3 = 1, NE3
751:                do 310 I3 = 1, IEDT3(L3)
752:                  M3      = IEDT3(L3+I3)
753: C
754:                  L4      = 1
755:                  do 320 N4 = 1, NE4
756:                    do 330 I4 = 1, IEDT4(L4)
757:                      M4      = IEDT4(L4+I4)
758: C-----
759:                      DWRK(N4,N3,N2,N1) = DWRK(N4,N3,N2,N1)
760:                      &
761: C-----
762:                      continue
763:                      L4 = L4 + IEDT4(L4) + 1
764: 320      continue
765: C
766:          310      continue
767:          L3      = L3 + IEDT3(L3) + 1
768: 300      continue
769: C
770:          290      continue
771: C
772:          280      continue
773:          L1      = L1 + IEDT1(L1) + 1
774: 270      continue
775: C
776:          do 1300 N4 = 1, NE4
777:            do 1310 N3 = 1, NE3
778:              do 1320 N1 = 1, NE1
779:                do 1200 N2 = 1, NE2
780:                  NN = ND2 / N2

```

src/shared/feynmx.f

```

781:      NDE = NN * N2
782: C
783:      X = 0.0D0
784:      X2 = 0.0D0
785:      do 1210 I = 1, NDE
786:        X = X + DWRK(N4,N3,I,N1)
787: 1210      continue
788:      if ( N2.eq.1 ) then
789:        do 1220 I = 1, NDE
790:          X2 = X2 + DWRK(N4,N3,I,N1)**2
791: 1220      continue
792:      else if ( N2.eq.2 ) then
793:        do 1230 I = 1, NDE, 2
794:          X2 = X2 +
795:            & (DWRK(N4,N3,I,N1)+DWRK(N4,N3,I+1,N1))**2
796: 1230      continue
797:      else if ( N2.eq.3 ) then
798:        do 1240 I = 1, NDE, 3
799:          X2 = X2 +
800:            & (DWRK(N4,N3,I,N1)+DWRK(N4,N3,I+1,N1)+
801:            & DWRK(N4,N3,I+2,N1))**2
802: 1240      continue
803:      else
804:        do 1250 I = 1, NDE, N2
805:          T = 0
806:          do 1260 J = 0, N2-1
807:            T = T + DWRK(N4,N3,I+J,N1)
808: 1260          continue
809:          X2 = X2 + T**2
810: 1250          continue
811:        end if
812: C
813:        X = X / DBLE(NN)
814:        X2 = X2 / DBLE(NN)
815:        VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
816:        if( X.gt.0.d0 ) then
817:          DWRK2(N4,N3,N2,N1) = VAR / X - 1.D0
818:        else
819:          DWRK2(N4,N3,N2,N1) = 0.d0
820:          IWAR = 1
821:        end if
822: 1200      continue
823: 1320      continue
824: 1310      continue
825: 1300      continue
826: C
827:      else if ( ITBIN.eq.3 ) then
828:        L1 = 1
829:        do 340 N1 = 1, NE1
830:          do 350 I1 = 1, IEDT1(L1)
831:            M1 = IEDT1(L1+I1)
832: C
833:            L2 = 1
834:            do 360 N2 = 1, NE2
835:              do 370 I2 = 1, IEDT2(L2)
836:                M2 = IEDT2(L2+I2)
837: C
838:                do 380 N3 = 1, ND3
839: C
840:                  L4 = 1
841:                  do 390 N4 = 1, NE4
842:                    do 400 I4 = 1, IEDT4(L4)
843:                      M4 = IEDT4(L4+I4)
844: C-----
845:                      DWRK(N4,N3,N2,N1) = DWRK(N4,N3,N2,N1)

```

```

846:      & + DTALY(M4,N3,M2,M1)
847: C-----
848:      400      continue
849:      L4 = L4 + IEDT4(L4) + 1
850: 390      continue
851: C
852: 380      continue
853: C
854: 370      continue
855:      L2 = L2 + IEDT2(L2) + 1
856: 360      continue
857: C
858: 350      continue
859:      L1 = L1 + IEDT1(L1) + 1
860: 340      continue
861: C
862:      do 1500 N4 = 1, NE4
863:      do 1510 N2 = 1, NE2
864:      do 1520 N1 = 1, NE1
865:        do 1400 N3 = 1, NE3
866:          NN = ND3 / N3
867:          NDE = NN * N3
868: C
869:          X = 0.0D0
870:          X2 = 0.0D0
871:          do 1410 I = 1, NDE
872:            X = X + DWRK(N4,I,N2,N1)
873: 1410          continue
874:          if ( N3.eq.1 ) then
875:            do 1420 I = 1, NDE
876:              X2 = X2 + DWRK(N4,I,N2,N1)**2
877: 1420            continue
878:          else if ( N3.eq.2 ) then
879:            do 1430 I = 1, NDE, 2
880:              X2 = X2 +
881:                & (DWRK(N4,I,N2,N1)+DWRK(N4,I+1,N2,N1))**2
882: 1430            continue
883:          else if ( N3.eq.3 ) then
884:            do 1440 I = 1, NDE, 3
885:              X2 = X2 +
886:                & (DWRK(N4,I,N2,N1)+DWRK(N4,I+1,N2,N1)+
887:                & DWRK(N4,I+2,N2,N1))**2
888: 1440            continue
889:          else
890:            do 1450 I = 1, NDE, N3
891:              T = 0
892:              do 1460 J = 0, N3-1
893:                T = T + DWRK(N4,I+J,N2,N1)
894: 1460              continue
895:              X2 = X2 + T**2
896: 1450              continue
897:            end if
898: C
899:            X = X / DBLE(NN)
900:            X2 = X2 / DBLE(NN)
901:            VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
902:            if( X.gt.0.d0 ) then
903:              DWRK2(N4,N3,N2,N1) = VAR / X - 1.D0
904:            else
905:              DWRK2(N4,N3,N2,N1) = 0.d0
906:              IWAR = 1
907:            end if
908: 1400          continue
909: 1520          continue
910: 1510          continue

```


src/shared/feynmx.f

```

911: 1500 continue
912: C
913:     else if ( ITBIN.eq.4 ) then
914:         L1 = 1
915:         do 410 N1 = 1, NE1
916:             do 420 I1 = 1, IEDT1(L1)
917:                 M1 = IEDT1(L1+I1)
918: C
919:                 L2 = 1
920:                 do 430 N2 = 1, NE2
921:                     do 440 I2 = 1, IEDT2(L2)
922:                         M2 = IEDT2(L2+I2)
923: C
924:                         L3 = 1
925:                         do 450 N3 = 1, NE3
926:                             do 460 I3 = 1, IEDT3(L3)
927:                                 M3 = IEDT3(L3+I3)
928: C
929:                                 do 470 N4 = 1, ND4
930: C-----
931:                                     DWRK(N4,N3,N2,N1) = DWRK(N4,N3,N2,N1)
932:                                     &
933:                                     + DTALY(N4,M3,M2,M1)
934: C-----
935:                                     continue
936:                                     470
937:                                     continue
938:                                     L3 = L3 + IEDT3(L3) + 1
939: C
940:                                     continue
941:                                     L2 = L2 + IEDT2(L2) + 1
942: C
943:                                     continue
944:                                     420
945:                                     continue
946:                                     L1 = L1 + IEDT1(L1) + 1
947: C
948:                                     do 1700 N1 = 1, NE1
949:                                     do 1710 N2 = 1, NE2
950:                                     do 1720 N3 = 1, NE3
951:                                     do 1600 N4 = 1, NE4
952:                                         NN = ND4 / N4
953:                                         NDE = NN * N4
954: C
955:                                         X = 0.0D0
956:                                         X2 = 0.0D0
957:                                         do 1610 I = 1, NDE
958:                                             X = X + DWRK(I,N3,N2,N1)
959:                                         continue
960:                                         if ( N4.eq.1 ) then
961:                                             do 1620 I = 1, NDE
962:                                                 X2 = X2 + DWRK(I,N3,N2,N1)**2
963:                                             continue
964:                                         else if ( N4.eq.2 ) then
965:                                             do 1630 I = 1, NDE, 2
966:                                                 X2 = X2 +
967:                                                 (DWRK(I,N3,N2,N1)+DWRK(I+1,N3,N2,N1))**2
968:                                             continue
969:                                         else if ( N4.eq.3 ) then
970:                                             do 1640 I = 1, NDE, 3
971:                                                 X2 = X2 +
972:                                                 (DWRK(I,N3,N2,N1)+DWRK(I+1,N3,N2,N1)+
973:                                                 &
974:                                                 DWRK(I+2,N3,N2,N1))**2
975:                                             continue
976:                                         else

```

```

976:                                     do 1650 I = 1, NDE, N4
977:                                         T = 0
978:                                         do 1660 J = 0, N4-1
979:                                             T = T + DWRK(I+J,N3,N2,N1)
980:                                         continue
981:                                         X2 = X2 + T**2
982:                                     1650
983:                                     continue
984:                                     end if
985: C
986:                                     X = X / DBLE(NN)
987:                                     X2 = X2 / DBLE(NN)
988:                                     VAR = (X2-X*X)*DBLE(NN)/DBLE(NN-1)
989:                                     if( X.gt.0.d0 ) then
990:                                         DWRK2(N4,N3,N2,N1) = VAR / X - 1.D0
991:                                     else
992:                                         DWRK2(N4,N3,N2,N1) = 0.d0
993:                                         IWAR = 1
994:                                     end if
995:                                     1600
996:                                     continue
997:                                     1720
998:                                     continue
999:                                     1710
1000:                                     continue
1001:                                     end if
1002:                                     if( IWAR.eq.1 ) then
1003:                                         write(IOW,'(a,a,a)') ' !!! Warning !!! ',
1004:                                         &
1005:                                         ' No counts for Feynman-A bins.',
1006:                                         &
1007:                                         ' Feynman-Y value has set to 0.'
1008:                                     end if
1009: C
1010: C
1011: DNG = DBLE(NGENE)
1012: do 2000 N1 = 1, NE1
1013:     do 2010 N2 = 1, NE2
1014:         do 2020 N3 = 1, NE3
1015:             do 2030 N4 = 1, NE4
1016:                 ETALY(N4,N3,N2,N1) = ETALY(N4,N3,N2,N1)
1017:                 &
1018:                 + DWRK2(N4,N3,N2,N1)*DNG
1019:                 SETALY(N4,N3,N2,N1) = SETALY(N4,N3,N2,N1)
1020:                 &
1021:                 + DWRK2(N4,N3,N2,N1)**2*DNG
1022:             2030
1023:             continue
1024:         2020
1025:         continue
1026:     2010
1027:     continue
1028: 2000
1029: continue
1030: C
1031: return
1032: end
1033: C-----
1034: C
1035: subroutine FEYNM5( ETALY, SETALY,DTALY, NGENE, DWRK, DWRK2,
1036: &
1037: &
1038: &
1039: &
1040: &
1041: &
1042: &
1043: &
1044: &
1045: )
1046: C=====
1047: C what to do: edit Feynman-Y tally having 5 dimensions.
1048: C called in : stlsum
1049: C=====
1050: implicit real*8(A-H,O-Z)
1051: C
1052: real*8 DWRK(ND5,ND4,ND3,ND2,ND1)
1053: real*8 DWRK2(NE5,NE4,NE3,NE2,NE1)

```

src/shared/feynm.f

```

1041: C
1042:   real*8 ETALY(NE5,NE4,NE3,NE2,NE1)
1043:   real*8 SETALY(NE5,NE4,NE3,NE2,NE1)
1044:   real*8 DTALY(ND5,ND4,ND3,ND2,ND1)
1045: C
1046:   integer IEDT1(*)
1047:   integer IEDT2(*)
1048:   integer IEDT3(*)
1049:   integer IEDT4(*)
1050:   integer IEDT5(*)
1051: C
1052:   include 'INC/_IOUNIT'
1053: C-----
1054:   write(IOW,*) ' XXX FEYNM5 is not prepared. STOP '
1055:   stop
1056: C
1057:   do 140 N1 = 1, NE1
1058:     do 130 N2 = 1, NE2
1059:       do 120 N3 = 1, NE3
1060:         do 110 N4 = 1, NE4
1061:           do 100 N5 = 1, NE5
1062:             DWRK(N5,N4,N3,N2,N1) = 0.0D0
1063:             100 continue
1064:             110 continue
1065:             120 continue
1066:             130 continue
1067:             140 continue
1068: C
1069: C
1070: C
1071:   L1 = 1
1072:   do 240 N1 = 1, NE1
1073:     do 230 I1 = 1, IEDT1(L1)
1074:       M1 = IEDT1(L1+I1)
1075: C
1076:   L2 = 1
1077:   do 220 N2 = 1, NE2
1078:     do 210 I2 = 1, IEDT2(L2)
1079:       M2 = IEDT2(L2+I2)
1080: C
1081:   L3 = 1
1082:   do 200 N3 = 1, NE3
1083:     do 190 I3 = 1, IEDT3(L3)
1084:       M3 = IEDT3(L3+I3)
1085: C
1086:   L4 = 1
1087:   do 180 N4 = 1, NE4
1088:     do 170 I4 = 1, IEDT4(L4)
1089:       M4 = IEDT4(L4+I4)
1090: C
1091:   L5 = 1
1092:   do 160 N5 = 1, NE5
1093:     do 150 I5 = 1, IEDT5(L5)
1094:       M5 = IEDT5(L5+I5)
1095: C-----
1096:   DWRK(N5,N4,N3,N2,N1) =
1097:   & DWRK(N5,N4,N3,N2,N1)
1098:   & + DTALY(M5,M4,M3,M2,M1)
1099: C-----
1100:   150 continue
1101:   L5 = L5 + IEDT5(L5) + 1
1102:   160 continue
1103: C
1104:   170 continue
1105:   L4 = L4 + IEDT4(L4) + 1

```

```

1106:   180 continue
1107: C
1108:   190 continue
1109:   L3 = L3 + IEDT3(L3) + 1
1110:   200 continue
1111: C
1112:   210 continue
1113:   L2 = L2 + IEDT2(L2) + 1
1114:   220 continue
1115: C
1116:   230 continue
1117:   L1 = L1 + IEDT1(L1) + 1
1118:   240 continue
1119: C
1120: C
1121:   DNG = 1.0D0/DBLE(NGENE)
1122:   do 290 N1 = 1, NE1
1123:     do 280 N2 = 1, NE2
1124:       do 270 N3 = 1, NE3
1125:         do 260 N4 = 1, NE4
1126:           do 250 N5 = 1, NE5
1127:             ETALY(N5,N4,N3,N2,N1) = ETALY(N5,N4,N3,N2,N1)
1128:             & + DWRK(N5,N4,N3,N2,N1)
1129:             SETALY(N5,N4,N3,N2,N1) = SETALY(N5,N4,N3,N2,N1)
1130:             & + DWRK(N5,N4,N3,N2,N1)**2*DNG
1131:           250 continue
1132:           260 continue
1133:           270 continue
1134:           280 continue
1135:           290 continue
1136: C
1137:   return
1138: end

```

src/shared/flsurf.f

```

1:      subroutine FLSURF( ISURF, NTSRF, KTSRF, ITSURF, IDSRF, NN,      IBP,
2:      &                    NBANK, XX, YY, ZZ, AAI, BBI, CCI,
3:      &                    SDA, NSDA, DINF, DEPS, ANGLB, NANGLE, DBNK,
4:      &                    KDBNK, MDBNK, IBNK, KIBNK, MIBNK,
5:      W                    X, Y, Z, AI, BI, CI, DI,
6:      W                    X2, Y2, Z2, AI2, BI2, CI2,
7:      W                    IFI, IKL, IWK1, IWK2 )
8:      C=====
9:      C PURPOSE: Calculation of distances to a tally-surface boundary.
10:     C
11:     C Accept NN particles whose coordinate & direction data are stored
12:     C in arrays XX,YY,ZZ, AAI,BBI and CCI sequentially.
13:     C Calculate distances and angles(cosine) to a tally-surface.
14:     C
15:     C CALLED IN: FLIONE
16:     C CALLS: SOSOL1
17:     C-----
18:     C Arguments (i=input, o=output, w=work)
19:     C i ISURF : surface # for which crossing distance & angle is calculated
20:     C i NTSRF : total number of tally-surface
21:     C i KTSRF, ITSURF, IDSRF : tally-surface data (see INC/_PTABY0 )
22:     C
23:     C i NN : number of particles to be processed.
24:     C i IBP(*) : bank pointer of processed particles
25:     C i XX(NN), YY(NN), ZZ(NN) : coordinates of particles
26:     C      This is current position (before flight with newly
27:     C      calculated distance to zone boundary or collision
28:     C      point)
29:     C i AAI(NN), BBI(NN), CCI(NN) : direction vector of particles
30:     C
31:     C XX,YY,ZZ and AAI,BBI,CCI must be that of absolute coordinates.
32:     C
33:     C i SDA(NSDA): geometry data (see description attached to
34:     C      common /PGEOM/ ety.)
35:     C i DINF : "infinite" distance.
36:     C i DEPS : "minimum" distance.
37:     C i ANGLB(NANGLE+1) : common angle bin in cosine (from 1 to -1)
38:     C i o DBNK(NBANK,*),KDBNK(0:MDBNK),IBNK(NBANK,*),KIBNK(0:MIBNK):
39:     C      optimal bank parameters
40:     C
41:     C <<Parameters set in this routine>>
42:     C
43:     C IBNK(*,KIBNK(10)+ISURF-1) : tally-surface side flag
44:     C      +1 : inside on surface
45:     C      -1 : outside of surface
46:     C      +2 : inside on surface but distance is
47:     C      not calculated yet
48:     C      -2 : outside of surface but distance is
49:     C      not calculated yet
50:     C      3 : inside/outside is unknown
51:     C      (and inside/outside is determined
52:     C      in this routine)
53:     C      This routine calculates distance and angle for
54:     C      particles IBNK(*,KIBNK(10)+ISURF-1) = +2, -2 or 3.
55:     C
56:     C IBNK(*,KIBNK(10)+NTSRF+ISURF-1) : angle bin #
57:     C DBNK(*,KDBNK(7)+ISURF-1) : distance to tally-surface
58:     C DBNK(*,KDBNK(7)+NTSRF+ISURF-1) : cosine of crossing angle.
59:     C      positive for inside to outside crossing.
60:     C
61:     C w X,Y,Z, AI, BI, CI, DI, X2, Y2, Z2, AI2, BI2, CI2:
62:     C      working array (real*8)
63:     C w IFI : working array (logical)
64:     C w IKL, IWK1, IWK2 : working array (integer)
65:     C

```

```

66:     C-----
67:     C UPDATE:
68:     C=====
69:     C      implicit real*8(D)
70:     C
71:     C      integer IBP(NN)
72:     C
73:     C
74:     C      real*8 XX(NN), YY(NN), ZZ(NN)
75:     C      real*8 AAI(NN), BBI(NN), CCI(NN)
76:     C
77:     C ... Tally surface .....
78:     C
79:     C      integer KTSRF(2,*), ITSURF(NTSRF+1,2), IDSRF(NTSRF)
80:     C
81:     C .... GEOMETRY (surface shape data).....
82:     C
83:     C      real*8 SDA(NSDA)
84:     C
85:     C .... angle bin (cosine) ....
86:     C
87:     C      real*8 ANGLB(NANGLE+1)
88:     C
89:     C ... optional bank parameters
90:     C
91:     C      real*8 DBNK(NBANK,*)
92:     C      integer KDBNK(0:MDBNK)
93:     C      integer IBNK(NBANK,*)
94:     C      integer KIBNK(0:MIBNK)
95:     C
96:     C .... WORKING AREA .....
97:     C
98:     C      real*8 X(NN), Y(NN), Z(NN), AI(NN), BI(NN), CI(NN)
99:     C      real*8 DI(NN)
100:     C      real*8 X2(NN), Y2(NN), Z2(NN), AI2(NN), BI2(NN), CI2(NN)
101:     C      logical IFI(NN)
102:     C      integer IKL(NN)
103:     C      integer IWK1(NN)
104:     C      integer IWK2(NN)
105:     C
106:     C .... local data
107:     C
108:     C      logical JRR
109:     C      integer KZA(2)
110:     C
111:     C      real*8 DSMALL
112:     C      parameter( DSMALL = 1.0D-7 )
113:     C
114:     C-----
115:     C
116:     C      DEPS1 = DEPS*1.00001D0
117:     C      DEPS2 = (DEPS1/2.D0)**2
118:     C
119:     C      NNS1 = ITSURF(ISURF,1)
120:     C      NNS2 = ABS(ITSURF(ISURF,2))
121:     C      NNS3 = ITSURF(ISURF+1,1) - 1
122:     C
123:     C ... check inside/outside for crossing surface
124:     C
125:     C      KI10 = KIBNK(10) + ISURF - 1
126:     C      ICC = 0
127:     C *VOCL LOOP,NOVREC
128:     C      do 100 I = 1, NN
129:     C          if ( IBNK(IBP(I),KI10).eq.3 ) then
130:     C              ICC = ICC + 1

```

src/shared/flsurf.f

```

131:      IWK1(ICC) = IBP(I)
132:      X(ICC) = XX(I)
133:      Y(ICC) = YY(I)
134:      Z(ICC) = ZZ(I)
135:    end if
136:  100 continue
137: C
138:    if ( ICC.gt.0 ) then
139:      KZA(1) = 1
140:      KZA(2) = NNS2 - NNS1 + 1 + 1
141:      JRR = .false.
142:      call JUDGE( 'FLSURF', 1, ICC, X, Y, Z, IFI, SDA, KTSRF(1,NNS1),
143: &      KZA, NSDA, KZA(2), 2, JYMNT, IWK2, .true., JRR, IDUMMY,
144: &      IDUMMY, RDUMMY )
145: C
146: *VOCL LOOP,NOVREC
147:    do 110 I = 1, ICC
148:      IS = -2
149:      if ( IFI(I) ) IS = +2
150:      IBNK(IWK1(I),KI10) = IS
151:    110 continue
152:    end if
153: C
154: C-----
155: C calculate distances to actual tally-surface until the position
156: C that gives the distance is within "clipping" pseudo-zone.
157: C-----
158: C <<attention>> (March 2000)
159: C Current logic for distance calculation is assuming that the surface
160: C be composed as inside or outside of single body.
161: C To support a tally-surface (actual, not for clipping) with multiple
162: C body, we may need a different logic (more complicated or simpler?).
163: C-----
164: C
165:      KD71 = KDBNK(7) + ISURF - 1
166:      NN2 = 0
167: *VOCL LOOP,NOVREC
168:    do 120 I = 1, NN
169:      IJ = IBNK(IBP(I),KI10)
170:      if ( ABS(IJ).eq.2 ) then
171:        NN2 = NN2 + 1
172:        IWK1(NN2) = IBP(I)
173:        X(NN2) = XX(I)
174:        Y(NN2) = YY(I)
175:        Z(NN2) = ZZ(I)
176:        AI(NN2) = AAI(I)
177:        BI(NN2) = BBI(I)
178:        CI(NN2) = CCI(I)
179: C
180: C ... set flag +1/-1 here ...
181: C
182:        IBNK(IBP(I),KI10) = SIGN(1,IJ)
183:        DBNK(IBP(I),KD71) = 0.0D0
184:      end if
185:    120 continue
186: C
187:    if ( NN2.eq.0 ) go to 290
188: C
189: C == here is a returning point of implicit loop when
190: C we have more than one possible surface crossing points and
191: C the first one is rejected by clipping psuedo-zone.
192: C
193:  130 continue
194: C
195: C

```

```

196: C .....
197: C
198: C
199: C
200: C ... IBNK(*,KIBNK(10)+ITSURF-1) :
201: C
202: C +1 : particle is in inside of pseudo-zone
203: C -1 : particle is in outside of pseudo-zone
204: C
205: C -----
206: C LLG      inside/outside of particle      effective LLG?      sign
207: C -----
208: C 0          inside (+1)                    0                    *(+1)
209: C 0          outside (-1)                   1(2)                  *(-1)
210: C 1(2)       inside (+1)                    1(2)                  *(+1)
211: C 1(2)       outside (-1)                   0                    *(-1)
212: C -----
213: C
214: C "effective LLG" is LLG value compatible with that in SPEAR1 routine
215: C "effective LLG" = 0 : calculate minimum of larger distance
216: C "effective LLG" != 0 : calculate maximum of minimum distance
217: C
218: C
219: C LLG2 = SIGN(1,ITSRF(ISURF,2))
220: C
221: C IKNEG = 0
222: *VOCL LOOP,NOVREC
223: do 140 I = 1, NN2
224:   IK = LLG2*IBNK(IWK1(I),KI10)
225:   if ( IK.gt.0 ) then
226:     DI(I) = DINF
227:   else
228:     DI(I) = -DINF
229:     IKNEG = IKNEG + 1
230:   end if
231:   IWK2(I) = IK
232:   X2(I) = DINF
233: 140 continue
234: C
235: C
236: do 200 N = NNS1, NNS2
237:   JS = SIGN(1,KTSRF(1,N))
238:   LS1 = ABS(KTSRF(1,N))
239:   KSF = INT(SDA(LS1))
240: C
241: C ..... JS : + OR - SIGN INDICATES ON WHICH SIDE OF THE SURFACE
242: C THE PARTICLES EXIST. ( +/- = INSIDE/OUTSIDE )
243: C LS : HEAD POSITION OF SURFACE DATA IN SDA ARRAY
244: C KSF: KIND OF SURFACE (1/2/3 = SLAB/SPHERE/CYLINDER)
245: C
246: C LS = LS1 + 1
247: C
248: C ..... SLAB .....
249: C
250:   if ( KSF.eq.1 ) then
251: *VOCL LOOP,NOVREC
252:   do 150 I = 1, NN2
253:     DUP = -(SDA(LS)*X(I)+SDA(LS+1)*Y(I)+SDA(LS+2)*Z(I))
254:     DUV = SDA(LS)*AI(I) + SDA(LS+1)*BI(I) + SDA(LS+2)*
255: &      CI(I)
256:     IK = IWK2(I)
257: *VOCL STMT,IF(100)
258:     if ( DUV.ne.0.0D0 ) then
259:       D1 = (SDA(LS+3)+DUP) /DUV
260:       D2 = (SDA(LS+4)+DUP) /DUV

```

src/shared/flsurf.f

```

261:         if ( IK.gt.0 ) then
262:             D22 = MAX(D1,D2)
263:             if ( D22.gt.0.0.and.D22.lt.DI(I) ) then
264:                 DI(I) = D22
265:                 IKL(I) = LS1
266:             end if
267:         else
268:             D22 = MAX(D1,D2)
269:             D11 = MIN(D1,D2)
270:             if ( D11.gt.0.0.and.D11.gt.DI(I) ) then
271:                 DI(I) = D11
272:                 IKL(I) = LS1
273:             end if
274:             X2(I) = MIN(X2(I),D22)
275:         end if
276: C         ... particle does not cross and outside of the slab
277: C         ----> never cross
278: C
279: C
280: *VOCL STMT,IF(0)
281:         else if ( (SDA(LS+3)+DUP)*(SDA(LS+4)+DUP).ge.0.0D0 ) then
282:             if ( IK.lt.0 ) then
283:                 DI(I) = DINF
284:                 X2(I) = -DINF
285:             end if
286:         end if
287: C
288: 150         continue
289: C
290: C         ..... SPHERE .....
291: C
292:         else if ( KSF.eq.2 ) then
293: *VOCL LOOP,NOVREC
294:         do 160 I = 1, NN2
295:             DPCX = X(I) - SDA(LS)
296:             DPCY = Y(I) - SDA(LS+1)
297:             DPCZ = Z(I) - SDA(LS+2)
298:             DVPC = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
299: C         .... if ( LLG.ne.0 ) then
300:             DH = SDA(LS+3) - DPCX*DPCX - DPCY*DPCY - DPCZ*DPCZ
301:             DD = DVPC*DVPC + DH
302: C
303:             IK = IWK2(I)
304:             if ( DD.gt.0.0D0 ) then
305:                 DDD = SQRT(DD)
306:                 D11 = -DVPC - DDD
307:                 D22 = -DVPC + DDD
308:                 if ( IK.gt.0 ) then
309:                     if ( D22.gt.0.0.and.D22.lt.DI(I) ) then
310:                         DI(I) = D22
311:                         IKL(I) = LS1
312:                     end if
313:                 else
314:                     if ( D11.gt.0.0.and.D11.gt.DI(I) ) then
315:                         DI(I) = D11
316:                         IKL(I) = LS1
317:                     end if
318:                     X2(I) = MIN(X2(I),D22)
319:                 end if
320: C
321: C         ... particle does not cross current surface and
322: C         on outside of body ----> never cross
323: *VOCL STMT,IF(0)
324:         else
325:             if ( IK.lt.0 ) then

```

```

326:                 DI(I) = DINF
327:                 X2(I) = -DINF
328:             end if
329:         end if
330: 160         continue
331: C
332: C         ..... CYLINDER .....
333: C
334:         else if ( KSF.eq.3 ) then
335: *VOCL LOOP,NOVREC
336:         do 170 I = 1, NN2
337:             DPCX = X(I) - SDA(LS)
338:             DPCY = Y(I) - SDA(LS+1)
339:             DPCZ = Z(I) - SDA(LS+2)
340:             DUV = SDA(LS+3)*AI(I) + SDA(LS+4)*BI(I) + SDA(LS+5)*
341:             & CI(I)
342:             DA = 1.0D0 - DUV*DUV
343:             DUPC = SDA(LS+3)*DPCX + SDA(LS+4)*DPCY + SDA(LS+5)*
344:             & DPCZ
345:             DB = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ - DUV*DUPC
346: C
347: C         ..... if ( LLG.ne.0 ) then
348:             DH = DPCX**2 + DPCY**2 + DPCZ**2 - SDA(LS+6)
349:             & - DUPC**2
350:             DD = DB*DB - DA*DH
351: C
352:             IK = IWK2(I)
353:             if ( DA.gt.0.0.and.DD.gt.0.0 ) then
354:                 DD = SQRT(DD)
355:                 D11 = (-DB-DD)/DA
356:                 D22 = (-DB+DD)/DA
357:                 if ( IK.gt.0 ) then
358:                     if ( D22.gt.0.0.and.D22.lt.DI(I) ) then
359:                         DI(I) = D22
360:                         IKL(I) = LS1
361:                     end if
362:                 else
363:                     if ( D11.gt.0.0.and.D11.gt.DI(I) ) then
364:                         DI(I) = D11
365:                         IKL(I) = LS1
366:                     end if
367:                     X2(I) = MIN(X2(I),D22)
368:                 end if
369: C
370: C         ... particle does not cross current surface and
371: C         on outside of body ----> never cross
372: *VOCL STMT,IF(0)
373:         else if ( DH.ge.0.0 ) then
374:             if ( IK.lt.0 ) then
375:                 DI(I) = DINF
376:                 X2(I) = -DINF
377:             end if
378:         end if
379: 170         continue
380: C
381: C         ..... QUADRATIC EXPRESSION .....
382: C
383: C         S1*X**2 + S2*Y**2 + S3*Z**2 + S4*X*Y + S5*Y*Z + S6*Z*X
384: C         S7*X + S8*Y + S9*Z + S10 = 0
385: C
386:         else if ( KSF.eq.4 ) then
387: *VOCL LOOP,NOVREC
388:         do 180 I = 1, NN2
389: C
390:             DRV1 = SDA(LS)*AI(I) + SDA(LS+3)*BI(I) + SDA(LS+5)*

```

src/shared/flsurf.f

```

391:      &          CI(I)
392:      DRV2      = SDA(LS+1)*BI(I) + SDA(LS+4)*CI(I)
393:      DRV3      = SDA(LS+2)*CI(I)
394:      DRW1      = SDA(LS)*X(I) + SDA(LS+3)*Y(I) + SDA(LS+5)*Z(I)
395:      &          + SDA(LS+6)
396:      DRW2      = SDA(LS+1)*Y(I) + SDA(LS+4)*Z(I) + SDA(LS+7)
397:      DRW3      = SDA(LS+2)*Z(I) + SDA(LS+8)
398:      DA        = AI(I)*DRV1 + BI(I)*DRV2 + CI(I)*DRV3
399:      DB        =
400:      &          (AI(I)*DRW1+BI(I)*DRW2+CI(I)*DRW3+X(I)*DRV1+Y(I)*
401:      &          DRV2+Z(I)*DRV3)*0.5D0
402:      DC        = X(I)*DRW1 + Y(I)*DRW2 + Z(I)*DRW3 + SDA(LS+9)
403: C
404: C
405: C      (x,y,z) = (X(I),Y(I),Z(I)) + t (AI(I),BI(I),CI(I))
406: C
407: C      ==>
408: C
409: C      DA*t**2 + 2*DB*t + DC < 0 : inside
410: C      DA*t**2 + 2*DB*t + DC = 0 : on surface
411: C      DA*t**2 + 2*DB*t + DC > 0 : outside
412: C
413:      D11       = -DINF
414:      D22       = DINF
415: C
416:      IK        = IWK2(I)
417:      if ( DA.gt.DSMALL ) then
418:      DD        = DB**2 - DC*DA
419:      DEPSDA    = DEPS2*DA**2
420:      if ( DD.gt.DEPSDA .or. (DD.ge.0.0.and.IK.gt.0) ) then
421:      DH        = SQRT(DD)
422:      D11       = (-DB-DH) /DA
423:      D22       = (-DB+DH) /DA
424:      else if ( IK.lt.0 ) then
425:      D11       = DINF
426:      D22       = -DINF
427:      end if
428: C
429: C      ..... CONCAVE SURFACE      (DA < 0, D22 = D11 )
430: C
431:      else if ( DA.lt.-DSMALL ) then
432:      DD        = DB**2 - DC*DA
433:      if ( DD.gt.0.0 ) then
434:      DH        = SQRT(DD)
435:      D11       = (-DB-DH) /DA
436:      D22       = (-DB+DH) /DA
437:      if ( D22.lt.0.0 ) then
438:      D22       = DINF
439:      else if ( IK.gt.0 ) then
440:      D11       = -DINF
441:      else
442: C
443: C      >>> is the following procs. correct ??? (Mar 2000 mako)
444: C
445:      D111      = D11
446:      if ( DI(I).le.D22 ) D11 = -DINF
447:      if ( DI(I).ge.D111 ) D22 = DINF
448:      end if
449:      end if
450: C      ....
451: C      ..... ONLY ONE REAL ROOT IS POSSIBLE (DA = 0) .....
452:      else
453:      if ( DB.gt.0.0 ) then
454:      D22       = -DC/(DB*2.0D0)
455:      else if ( DB.lt.0.0 ) then

```

```

456:      D11       = -DC/(DB*2.0D0)
457: CCCCCCCCCCCCCC else if ( LLG.gt.0.and.DC.ge.0.0 ) then
458:      else if ( IK.lt.0.and.DC.ge.0.0 ) then
459:      D11       = DINF
460:      D22       = -DINF
461:      end if
462:      end if
463:
464:      if ( IK.gt.0 ) then
465:      if ( D22.gt.0.0.and.D22.lt.DI(I) ) then
466:      DI(I)     = D22
467:      IKL(I)    = LS1
468:      end if
469:      else
470:      if ( D11.gt.0.0.and.D11.gt.DI(I) ) then
471:      DI(I)     = D11
472:      IKL(I)    = LS1
473:      end if
474:      X2(I)     = MIN(X2(I),D22)
475:      end if
476:
477:      180      continue
478: C
479: C      ..... PLANE      (HALF SPACE) .....
480: C
481:      else if ( KSF.eq.5 ) then
482:      DSDA0     = SDA(LS)
483:      DSDA1     = SDA(LS+1)
484:      DSDA2     = SDA(LS+2)
485:      DSDA3     = SDA(LS+3)
486: C
487: C      ... if this half space is used in BBC body with minus
488: C      sign, and tally-surface uses outside of th BBC body.
489:      if ( NNS2.gt.NNS1 .and. LLG2.lt.0.and.JS.gt.0 ) then
490:      DSDA0     = -DSDA0
491:      DSDA1     = -DSDA1
492:      DSDA2     = -DSDA2
493:      DSDA3     = -DSDA3
494:      end if
495:      *VOCL LOOP,NOVREC
496:      do 190 I = 1, NN2
497:      DA        = DSDA0*AI(I)+DSDA1*BI(I)+DSDA2*CI(I)
498:      DB        = DSDA3 - (DSDA0*X(I)+DSDA1*Y(I)+DSDA2*Z(I))
499:
500:      IK        = IWK2(I)
501:
502:      D11       = -DINF
503:      D22       = DINF
504:      if ( DA.gt.0.0 ) then
505:      D22       = DB/DA
506:      else if ( DA.lt.0.0 ) then
507:      D11       = DB/DA
508:      else if ( IK.lt.0.and.DB.le.0.0 ) then
509:      D11       = DINF
510:      end if
511:
512:      if ( IK.gt.0 ) then
513:      if ( D22.gt.0.0.and.D22.lt.DI(I) ) then
514:      DI(I)     = D22
515:      IKL(I)    = LS1
516:      end if
517:      else
518:      if ( D11.gt.0.0.and.D11.gt.DI(I) ) then
519:      DI(I)     = D11
520:      IKL(I)    = LS1

```

src/shared/flsurf.f

```

521:         end if
522:         X2(I) = MIN(X2(I),D22)
523:         end if
524: 190      continue
525: C
526: C ..... ELLIPTIC TORUS .....
527: C ..... torus as tally-surface ... is not supported currently .....
528: C
529: CC      else if ( KSF.eq.6 ) then
530: CC      call SQSOL1( X, Y, Z, IKL, AI, BI, CI, SDA, LS-1, NN2, DINF,
531: CC      &          DEPS, DI, IKL2 )
532: C      end if
533: C
534: C 200 continue
535: C
536: C ... check validity of DI(I) for pathes from outside of the body
537: C
538: C if ( IKNEG.gt.0 ) then
539: C do 210 I = 1, NN2
540: C   IK = IKW2(I)
541: C   if ( IK.lt.0.and.DI(I)+DEPS1.gt.X2(I) ) then
542: C     DI(I) = -DINF
543: C   end if
544: C 210 continue
545: C end if
546: C
547: C
548: C -----
549: C check distance
550: C -----
551: C
552: C INN = 0
553: *VOCL LOOP,NOVREC
554: do 220 I = 1, NN2
555:   IP = IKW1(I)
556: C
557: C ... never crossing with the surface (body)
558: C
559: C if ( DI(I).le.DEPS1.or.DI(I).eq.DINF ) then
560: C if ( DI(I).le.0.0d0 .or. DI(I).eq.DINF ) then
561: C   IBNK(IP,KI10) = 0
562: C else
563: C   INN = INN + 1
564: C   X(INN) = X(I)
565: C   Y(INN) = Y(I)
566: C   Z(INN) = Z(I)
567: C   AI(INN) = AI(I)
568: C   BI(INN) = BI(I)
569: C   CI(INN) = CI(I)
570: C   DI(INN) = DI(I)
571: C   IKW1(INN) = IKW1(I)
572: C   IKL(INN) = IKL(I)
573: C end if
574: C 220 continue
575: C
576: C
577: C if ( INN.eq.0 ) go to 290
578: C
579: C
580: *VOCL LOOP,NOVREC
581: do 230 I = 1, INN
582:   X(I) = X(I) + DI(I)*AI(I)
583:   Y(I) = Y(I) + DI(I)*BI(I)
584:   Z(I) = Z(I) + DI(I)*CI(I)
585: C

```

```

586: C      ... save distance !!!
587: C
588: C      DBNK(IWK1(I),KD71) = DBNK(IWK1(I),KD71) + DI(I)
589: C 230 continue
590: C
591: C -----
592: C check for "clipping" pseudo-zone if necessary
593: C -----
594: C
595: C if ( NNS3.gt.NNS2 ) then
596: C
597: C   KZA(1) = 1
598: C   KZA(2) = NNS3 - NNS2 + 1
599: C   JRR = .false.
600: C   call JUDGE( 'FLSURF-2', 1, INN, X, Y, Z, IFI, SDA,
601: C   &          KTSRF(1,NNS2+1), KZA, NSDA, KZA(2)-1, 2, JVMNT, IKW2,
602: C   &          .true., JRR, IDUMMY, IDUMMY, RDUMMY )
603: C
604: C   NN1 = 0
605: *VOCL LOOP,NOVREC
606: do 240 I = 1, INN
607:   if ( IFI(I) ) then
608:     NN1 = NN1 + 1
609:     IKW2(NN1) = IKW1(I)
610:     X2(NN1) = X(I)
611:     Y2(NN1) = Y(I)
612:     Z2(NN1) = Z(I)
613:     AI2(NN1) = AI(I)
614:     BI2(NN1) = BI(I)
615:     CI2(NN1) = CI(I)
616: C
617: C   IKL(NN1) = IKL(I)
618: C   end if
619: C 240 continue
620: C
621: C   NN2 = 0
622: *VOCL LOOP,NOVREC
623: do 250 I = 1, INN
624:   IP = IKW1(I)
625:   if ( .not.IFI(I) ) then
626:     NN2 = NN2 + 1
627:     IBNK(IP,KI10) = -IBNK(IP,KI10)
628:     X(NN2) = X(I)
629:     Y(NN2) = Y(I)
630:     Z(NN2) = Z(I)
631:     AI(NN2) = AI(I)
632:     BI(NN2) = BI(I)
633:     CI(NN2) = CI(I)
634:     IKW1(NN2) = IP
635:   end if
636: C 250 continue
637: C
638: C ... no clipping pseudo-zone
639: C
640: C else
641: C   NN2 = 0
642: C   NN1 = INN
643: C   do 260 I = 1, NN1
644: C     X2(I) = X(I)
645: C     Y2(I) = Y(I)
646: C     Z2(I) = Z(I)
647: C     AI2(I) = AI(I)
648: C     BI2(I) = BI(I)
649: C     CI2(I) = CI(I)
650: C ----- IKL(I) = IKL(I)

```

src/shared/flsurf.f

```

651:          IWK2(I) = IWK1(I)
652: 260    continue
653:    end if
654: C
655: C-----
656: C      calculate cosine to normal vector
657: C-----
658: C
659:      KD72    = KDBNK(7) + NTSRF + ISURF - 1
660: C
661:      if ( NN1.gt.0 ) then
662: *VOCL LOOP,NOVREC
663:      do 270 I = 1, NN1
664:        LS1    = IKL(I)
665:        KSF     = INT(SDA(LS1))
666:        LS      = LS1 + 1
667: C
668: C      .... slab or plane (half space) ....
669: C
670:        if ( KSF.eq.1 .or. KSF.eq.5 ) then
671:          DDC    = AI2(I)*SDA(LS) + BI2(I)*SDA(LS+1) + CI2(I)*
672: &             SDA(LS+2)
673: C      .... Sphere .....
674:        else if ( KSF.eq.2 ) then
675:          DNX     = X2(I) - SDA(LS)
676:          DNY     = Y2(I) - SDA(LS+1)
677:          DNZ     = Z2(I) - SDA(LS+2)
678:          DD      = DNX*DNX + DNY*DNY + DNZ*DNZ
679: C      ... of course, DD should be squared radius ...
680:          DD      = SQRT(DD)
681:          DDC     = (AI2(I)*DNX+BI2(I)*DNY+CI2(I)*DNZ) /DD
682: C      .... Cylinder .....
683:        else if ( KSF.eq.3 ) then
684:          DNX     = X2(I) - SDA(LS)
685:          DNY     = Y2(I) - SDA(LS+1)
686:          DNZ     = Z2(I) - SDA(LS+2)
687:          DDD     = SDA(LS+3)*DNX + SDA(LS+4)*DNY + SDA(LS+5)*DNZ
688:          DNX     = DNX - SDA(LS+3)*DDD
689:          DNY     = DNY - SDA(LS+4)*DDD
690:          DNZ     = DNZ - SDA(LS+5)*DDD
691:          DD      = DNX*DNX + DNY*DNY + DNZ*DNZ
692: C      ... of course, DD should be squared radius ...
693:          DD      = SQRT(DD)
694:          DDC     = (AI2(I)*DNX+BI2(I)*DNY+CI2(I)*DNZ) /DD
695: C
696: C      .... general quadratic surface ....
697: C
698:        else if ( KSF.eq.4 ) then
699:          DNX     = 2.D0*SDA(LS)*X2(I) + SDA(LS+3)*Y2(I) +
700: &             SDA(LS+5)*Z2(I) + SDA(LS+6)
701:          DNY     = 2.D0*SDA(LS+1)*Y2(I) + SDA(LS+4)*Z2(I)
702: &             + SDA(LS+3)*X2(I) + SDA(LS+7)
703:          DNZ     = 2.D0*SDA(LS+2)*Z2(I) + SDA(LS+5)*X2(I)
704: &             + SDA(LS+4)*Y2(I) + SDA(LS+8)
705:          DD      = DNX*DNX + DNY*DNY + DNZ*DNZ
706:          DD      = SQRT(DD)
707:          DDC     = (AI2(I)*DNX+BI2(I)*DNY+CI2(I)*DNZ) /DD
708: C
709: C      .... torus as tally-surface ... is not supported currently ....
710: C
711: CCCCCCCCC else if ( KSF.eq.6 ) then
712: C      end if
713: C
714: C      ... determine sign of cosine by inside/outside flag
715: C
716:          DDC    = SIGN(DDC,DBLE( IBNK(IWK2(I),KI10)))
717:          DBNK(IWK2(I),KD72) = DDC
718: C
719: C      ... save angle on DSAVE for angle bin check
720:          DI(I)   = DDC
721: C
722: 270    continue
723: C
724: C      ... set angle bin ...
725: C
726:          call BS0DDD( ANGLB, NANGLE+1, DI, IKL, NN1 )
727: *VOCL LOOP,NOVREC
728:          do 280 I = 1, NN1
729:            IBNK(IWK2(I),KI10+NTSRF) = MAX(1,MIN(NANGLE,IKL(I)))
730: 280    continue
731: C
732:          end if
733: C
734: C      ... detected surface position is not in clipping pseudo-zone
735: C      find next crossing position if any.
736: C
737:          if ( NN2.gt.0 ) go to 130
738: C
739: C
740: 290 continue
741:          return
742:          end

```


src/shared/fnssc2.f

```

1:      subroutine FNSSC2( IRAND, NS,      I1,      I2,      DATA, X,      Y,
2:      &                  Z,      A,      B,      C,      E,      RWK,      IWK )
3:
4:      C/#IF SOURCE(NEW)
5:
6:      C=====
7:      C PURPOSE : GENERATION OF NEUTRON SOURCE TYPE9 (FNS-SOURCE)
8:      C history : prepared for compatibility with older version
9:      C-----
10:     C arguments (i=input, o=output, w=work )
11:     C io IRAND : seed of random numbers
12:     C i NS : number of source particles to be generated
13:     C i I1, I2 : data are stored from X(I1) to X(I2) etc.
14:     C i DATA(8) : source parameters
15:     C o X,I,Z,A,B,C : position and direction of source particles
16:     C o E : energy of source particles
17:     C w RWK,IWK : working array
18:     C-----
19:     implicit real*8(D)
20:     c
21:     real*8 DATA(8)
22:     real*8 X(NS), Y(NS), Z(NS), A(NS), B(NS), C(NS), E(NS)
23:     c
24:     real RWK(NS,4)
25:     integer IWK(NS)
26:     c
27:     ... common /COUNTS/ include ifnssc (initialized to 0 in ACTION)
28:     c
29:     include '../shared/INC/_COUNTS'
30:     c
31:     .... data for source generation ...
32:     c
33:     C/#IF PARA( SX* )
34:     *      local common /FNSDAT/
35:     *      &      RF, Y2, IY, ED, YY
36:     C/#ELSEIF PARA( CRAY* )
37:     *      task common /FNSDAT/
38:     *      &      RF, Y2, IY, ED, YY
39:     C/#ELSE
40:     common /FNSDAT/ RF,      Y2,      IY,      ED,      YY
41:     C/#ENDIF
42:     c
43:     real RF(0:180), Y2(13)
44:     integer IY(2,13)
45:     real ED(14)
46:     real YY(13)
47:     c
48:     c
49:     ccc DATA ED / 0., 0.025, 0.050, 0.075, 0.100, 0.125, 0.150,
50:     ccc 1      0.175, 0.200, 0.225, 0.250, 0.275, 0.300, 0.325/
51:     ccc DATA YY / 11.85, 97.0, 349.0, 770.0, 1225.0, 1625.0, 1925.0,
52:     ccc 1      2180.0, 2395.0, 2550.0, 2710.0, 2840.0, 2950.0/
53:     ccc data iy(1,1) /0/
54:     c
55:     parameter( XM1 = 2.014102, XM2 = 3.016050, XM3 = 1.008665,
56:     &          XM4 = 4.002603,
57:     &          XMM = XM1**2 + XM2**2 + XM3**2 - XM4**2 )
58:     c
59:     real*8 PAI, DPAI, HPAI, PAI180
60:     parameter( PAI = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
61:     &          0.5D0*PAI, PAI180 = PAI/180D0 )
62:     c
63:     c--- initialization -----
64:     c
65:     if ( IFNSSC.eq.0 ) then

```

```

66:     IFNSSC = 1
67:     c
68:     ED(1) = 0.
69:     ED(2) = 0.025
70:     ED(3) = 0.050
71:     ED(4) = 0.075
72:     ED(5) = 0.100
73:     ED(6) = 0.125
74:     ED(7) = 0.150
75:     ED(8) = 0.175
76:     ED(9) = 0.200
77:     ED(10) = 0.225
78:     ED(11) = 0.250
79:     ED(12) = 0.275
80:     ED(13) = 0.300
81:     ED(14) = 0.325
82:     c
83:     YY(1) = 11.85
84:     YY(2) = 97.0
85:     YY(3) = 349.0
86:     YY(4) = 770.0
87:     YY(5) = 1225.0
88:     YY(6) = 1625.0
89:     YY(7) = 1925.0
90:     YY(8) = 2180.0
91:     YY(9) = 2395.0
92:     YY(10) = 2550.0
93:     YY(11) = 2710.0
94:     YY(12) = 2840.0
95:     YY(13) = 2950.0
96:     c
97:     Y2(1) = YY(1)
98:     do 100 I = 2, 13
99:         Y2(I) = YY(I) - YY(I-1)
100:    100 continue
101:    call BMCMK1( Y2, YY, IY, 13 )
102:     c
103:     TOTRF = 0.
104:     RF(0) = 0.
105:     do 110 I = 1, 180
106:         TH1 = (I-1)*PAI180
107:         TH2 = I*PAI180
108:         RF(I) = 2*PAI*(COS(TH1)-COS(TH2))*(0.9885*PAI180+0.0213*
109:         &          (SIN(TH2)-SIN(TH1))-0.00475*
110:         &          (SIN(2.*TH2)-SIN(2.*TH1)))
111:         TOTRF = TOTRF + RF(I)
112:         RF(I) = TOTRF
113:    110 continue
114:     do 120 I = 1, 180
115:         RF(I) = RF(I) /TOTRF
116:    120 continue
117:     end if
118:     c
119:     c ..... copy input data .....
120:     c
121:     X0 = DATA(1)
122:     Y0 = DATA(2)
123:     Z0 = DATA(3)
124:     XR = DATA(4)
125:     XDIREC = DATA(5)
126:     TH1 = DATA(6)
127:     FA1 = DATA(7)
128:     PU1 = DATA(8)
129:     c
130:     C ..... SAMPLING OF RADIUS/XR .....

```

src/shared/fnssc2.f

```

131: c
132: c   truncated gauss distribution :
133: c
134: c   P(r) = 2 r * exp(-r**2) / (1-exp(-1)) dr
135: c
136: c       0 < r < 1.0
137: c
138: c   call RANU2( IRAND, RWK, NS, ICON )
139: c   DEEE      = 1.0D0 - EXP(-1.0D0)
140: c   do 130 I = I1, I2
141: c       RWK(I,4) = SQRT(-LOG(1.0D0-RWK(I,1)*DEEE))
142: c 130 continue
143: c
144: c ..... STARTING POSITION .....
145: c
146: c   call RANU2( IRAND, RWK, NS, ICON )
147: c   do 140 I = I1, I2
148: c       THETA = RWK(I,1)*DPA1
149: c       X(I)  = X0 + RWK(I,4)*COS(THETA)*XR
150: c       Y(I)  = Y0 + RWK(I,4)*SIN(THETA)*XR
151: c       Z(I)  = Z0
152: c 140 continue
153: c
154: c ..... SAMPLING FROM RF BY B.M.C. METHOD .....
155: c
156: c   call RANU2( IRAND, RWK, NS*4, ICON )
157: c   call BSVINC( RF(1), 180, RWK, IWK, NS )
158: c
159: c   do 150 I = I1, I2
160: c       RF1 = RF(IWK(I))
161: c       RF2 = RF(IWK(I)+1)
162: c       DTHCM = (IWK(I)+(RWK(I,1)-RF1)/(RF2-RF1))*PAI180
163: c
164: c ..... SAMPLING OF DEUTRON ENERGY .....
165: c
166: c
167: c /#IF ROUND OFF(NEAREST)
168: c     N = MIN(INT(13*RWK(I,2))+1,13)
169: c     L = MIN(INT(2-YY(N)+RWK(I,3)),2)
170: c /#ELSE
171: c     N = 13*rwk(I,2)+1
172: c     L = 2.0-YY(N)+rwk(I,3)
173: c /#ENDIF
174: c     N = IY(L,N)
175: c     E0 = (ED(N)+ED(N+1)) / 2.0
176: c
177: c     DGAM = SQRT(1./(10*(0.6+17.6/E0)))
178: c     RAD = ACOS(-DGAM)
179: c     DFAIL = SIN(DTHCM) / SQRT(1.+2.*DGAM*COS(DTHCM)+DGAM*DGAM)
180: c     if ( DTHCM.gt.RAD ) DFAIL = -DFAIL
181: c     if ( ABS(DFAIL).gt.1.0 ) DFAIL = SIGN(1.0D0,DFAIL)
182: c     DFAI = ASIN(DFAIL)
183: c     if ( DTHCM.gt.RAD ) DFAI = DFAI + PAI
184: c     DCF = COS(DFAI)
185: c     DROH = SQRT(1.0-DCF**2)
186: c     DFAI0 = PAI*(2.0*RWK(I,4)-1.0)
187: c     B(I) = DROH*COS(DFAI0)
188: c     A(I) = DROH*SIN(DFAI0)
189: c     C(I) = DCF
190: c     E0 = E0/931.481 + XM1
191: c     ALP = XMM + 2.0*XM2*E0
192: c     AX = (E0+XM2)**2 - (E0**2-XM1**2)*DCF**2
193: c     BX = -ALP*(E0+XM2)
194: c     CX = SQRT((E0**2-XM1**2)*DCF**2*(ALP**2-4.*AX*XM3**2))
195: c     if ( DCF.lt.0.0 ) CX = -CX

```

```

196: c     W3 = (-BX+CX) / 2. / AX
197: c     E(I) = (W3-XM3)*931.481*1.0E+6
198: c 150 continue
199: c
200: c ..... ROTATION .....
201: c
202: c   if ( XDIREC.eq.2. ) then
203: c *VOCL LOOP,NOVREC
204: c   do 160 I = I1, I2
205: c       DAAA = A(I)
206: c       DBBB = B(I)
207: c       DCCC = C(I)
208: c       A(I) = DCCC
209: c       B(I) = DAAA
210: c       C(I) = DBBB
211: c       DXXX = X(I) - X0
212: c       DYYY = Y(I) - Y0
213: c       DZZZ = Z(I) - Z0
214: c       X(I) = DZZZ + X0
215: c       Y(I) = DXXX + Y0
216: c       Z(I) = DYYY + Z0
217: c 160 continue
218: c   else if ( XDIREC.eq.3. ) then
219: c *VOCL LOOP,NOVREC
220: c   do 170 I = I1, I2
221: c       DAAA = A(I)
222: c       DBBB = B(I)
223: c       DCCC = C(I)
224: c       A(I) = DBBB
225: c       B(I) = DCCC
226: c       C(I) = DAAA
227: c       DXXX = X(I) - X0
228: c       DYYY = Y(I) - Y0
229: c       DZZZ = Z(I) - Z0
230: c       X(I) = DYYY + X0
231: c       Y(I) = DZZZ + Y0
232: c       Z(I) = DXXX + Z0
233: c 170 continue
234: c   else if ( XDIREC.ge.4. ) then
235: c       DTH = TH1*PAI180
236: c       DFA = FA1*PAI180
237: c       DPU = PU1*PAI180
238: c       DT11 = COS(DTH)*COS(DFA)*COS(DPU) - SIN(DFA)*SIN(DPU)
239: c       DT12 = -COS(DTH)*COS(DFA)*SIN(DPU) - SIN(DFA)*COS(DPU)
240: c       DT13 = SIN(DTH)*COS(DFA)
241: c       DT21 = COS(DTH)*SIN(DFA)*COS(DPU) + COS(DFA)*SIN(DPU)
242: c       DT22 = -COS(DTH)*SIN(DFA)*SIN(DPU) + COS(DFA)*COS(DPU)
243: c       DT23 = SIN(DTH)*SIN(DFA)
244: c       DT31 = -SIN(DTH)*COS(DPU)
245: c       DT32 = SIN(DTH)*SIN(DPU)
246: c       DT33 = COS(DTH)
247: c *VOCL LOOP,NOVREC
248: c   do 180 I = I1, I2
249: c       DAAA = A(I)
250: c       DBBB = B(I)
251: c       DCCC = C(I)
252: c       A(I) = DT11*DAAA + DT21*DBBB + DT31*DCCC
253: c       B(I) = DT12*DAAA + DT22*DBBB + DT32*DCCC
254: c       C(I) = DT13*DAAA + DT23*DBBB + DT33*DCCC
255: c 180 continue
256: c   if ( XDIREC.ge.5. ) then
257: c *VOCL LOOP,NOVREC
258: c   do 190 I = I1, I2
259: c       DAAA = X(I) - X0
260: c       DBBB = Y(I) - Y0

```

src/shared/fnssc2.f

```
261:          DCCC      = Z(I) - Z0
262:          X(I)       = DT11*DAAA + DT21*DBBB + DT31*DCCC + X0
263:          Y(I)       = DT12*DAAA + DT22*DBBB + DT32*DCCC + Y0
264:          Z(I)       = DT13*DAAA + DT23*DBBB + DT33*DCCC + Z0
265: 190      continue
266:      end if
267:  end if
268: c ... endif for SOURCE(NEW)
269: C/#ENDIF
270:      return
271:  end
```

SAFE

src/shared/fnssoc.f

```

1:      subroutine FNSSOC( IRAND, NSS,   IGG,   XXX,   YYY,   ZZZ,   AAA,
2:      &                  BBB,   CCC,   WWW,   TTT,   R,   EEE,   IWK,
3:      &                  ENGYB, NGROUP, LSDDED, XDIREC, X0,   Y0,   Z0,
4:      &                  XR,   TH1,   FA1,   PU1 )
5: C=====
6: C PURPOSE : GENERATION OF NEUTRON SOURCE TYPE 9 (FNS-SOURCE)
7: C CALLED IN: SOURCE.      CALLS: (NONE)
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C
11: C io IRAND: seed of random number
12: C i NSS:   Number of source particles to be generated
13: C o IGG,XXX,YYY,ZZZ,AAA,BBB,CCC,WWW,TTT,EEE : particle bank data
14: C w R,IWK: working area ( why "R" is on such a position ?!)
15: C i ENGYB: energy boundaries
16: C i NGROUP: number of energy groups
17: C io LSDDED(*): "dead particle stack", from which particle revives!!
18: C
19: C i XDIREC: direction rotation option
20: C i XR: radius of sampling range
21: C i X0,Y0,Z0: position paramater
22: C i TH1,FA1,PU1: direction paramater
23: C-----
24:      implicit real*8(D)
25:      parameter( XM1 = 2.014102, XM2 = 3.016050, XM3 = 1.008665, XM4 =
26:      &          4.002603, XMM = XM1**2 + XM2**2 + XM3**2 + XM4**2 )
27: C
28:      real*8 PAI, PAI180
29:      parameter( PAI = 3.1415926535897932D0 )
30:      parameter( PAI180 = PAI/180D0 )
31: C
32:      include '../shared/INC/_COUNTS'
33: C
34: C .... data for source generation ...
35: C
36: C/#IF PARA( SX* )
37: *      local common /FNSDAT/
38: *      &      RF, Y2, IY, ED, Y
39: C/#ELSEIF PARA( CRAY* )
40: *      task common /FNSDAT/
41: *      &      RF, Y2, IY, ED, Y
42: C/#ELSE
43: common /FNSDAT/  RF,      Y2,      IY,      ED,      Y
44: C/#ENDIF
45: C
46:      real RF(0:180), Y2(13)
47:      integer IY(2,13)
48:      real ED(14), Y(13)
49: C
50:      real ENGYB(1)
51: C
52:      integer LSDDED(NSS)
53:      real*8 XXX(1), YYY(1), ZZZ(1), AAA(1), BBB(1), CCC(1)
54:      real WWW(1), EEE(NSS)
55:      real*8 TTT(1)
56: C
57:      real R(NSS,4)
58:      integer IGG(1), IWK(NSS)
59: C
60: C-----
61: C
62:      if ( IFNSSC.eq.0 ) then
63:          IFNSSC = 1
64: C
65:          ED(1) = 0.

```

```

66:          ED(2) = 0.025
67:          ED(3) = 0.050
68:          ED(4) = 0.075
69:          ED(5) = 0.100
70:          ED(6) = 0.125
71:          ED(7) = 0.150
72:          ED(8) = 0.175
73:          ED(9) = 0.200
74:          ED(10) = 0.225
75:          ED(11) = 0.250
76:          ED(12) = 0.275
77:          ED(13) = 0.300
78:          ED(14) = 0.325
79: C
80:          Y(1) = 11.85
81:          Y(2) = 97.0
82:          Y(3) = 349.0
83:          Y(4) = 770.0
84:          Y(5) = 1225.0
85:          Y(6) = 1625.0
86:          Y(7) = 1925.0
87:          Y(8) = 2180.0
88:          Y(9) = 2395.0
89:          Y(10) = 2550.0
90:          Y(11) = 2710.0
91:          Y(12) = 2840.0
92:          Y(13) = 2950.0
93: C
94: C
95:          Y2(1) = Y(1)
96:          do 100 I = 2, 13
97:              Y2(I) = Y(I) - Y(I-1)
98:          100 continue
99:          call BMCMK1( Y2, Y, IY, 13 )
100: C
101:          TOTRF = 0.
102:          RF(0) = 0.
103:          do 110 I = 1, 180
104:              TH = (I-1)*PAI180
105:              TH2 = I*PAI180
106:              RF(I) = 2*PAI*(COS(TH)-COS(TH2))*(0.9885*PAI180+0.0213*
107:              &          (SIN(TH2)-SIN(TH))-0.00475*(SIN(2.*TH2)-SIN(2.*TH)))
108:              TOTRF = TOTRF + RF(I)
109:              RF(I) = TOTRF
110:          110 continue
111:
112:          do 120 I = 1, 180
113:              RF(I) = RF(I) /TOTRF
114:          120 continue
115:
116:          end if
117: C
118: C ..... SAMPLING OF RADIUS/XR .....
119: C
120: C truncated gauss distribution :
121: C
122: C  $P(r) = 2 r * \exp(-r**2) / (1-\exp(-1)) dr$ 
123: C
124: C      0 < r < 1.0
125: C
126:          call RANU2( IRAND, R, NSS, ICON )
127:          DEEE = 1.0D0 - EXP(-1.0D0)
128:          do 130 I = 1, NSS
129:              R(I,4) = SQRT(-LOG(1.0D0-R(I,1)*DEEE))
130:          130 continue

```

src/shared/fnssoc.f

```

131: C
132: C ..... STARTING POSITION .....
133: C
134:       call RANU2( IRAND, R, NSS, ICON )
135:
136:       do 140 I = 1, NSS
137:           THETA = R(I,1)*2.*PAI
138:           XXX(LSDED(I)) = X0 + R(I,4)*COS(THETA)*XR
139:           YYY(LSDED(I)) = Y0 + R(I,4)*SIN(THETA)*XR
140:           ZZZ(LSDED(I)) = Z0
141: 140 continue
142: C
143: C ..... SAMPLING FROM RF WITH BINARY SEARCH .....
144: C
145:       call RANU2( IRAND, R, NSS*4, ICON )
146:       call BSVINC( RF(1), 180, R, IWK, NSS )
147: C
148:       do 150 I = 1, NSS
149:           RF1 = RF(IWK(I))
150:           RF2 = RF(IWK(I)+1)
151:           DTHCM = (IWK(I)+(R(I,1)-RF1)/(RF2-RF1))*PAI180
152: C
153: C ..... SAMPLING OF DEUTRON ENERGY .....
154: C
155:
156: C/#IF ROUNDOFF(NEAREST)
157:       N = MIN(INT(13*R(I,2))+1,13)
158: C/#ELSE
159:       N = 13*R(I,2)+1
160: C/#ENDIF
161:       III = MIN(2,INT(2.0-Y(N)+R(I,3)))
162:       N = IY(III,N)
163: C
164:       E0 = (ED(N)+ED(N+1)) /2.0
165: C
166:       DGAM = SQRT(1./(10*(0.6+17.6/E0)))
167:       RAD = ACOS(-DGAM)
168:       DFAI = SIN(DTHCM) /SQRT(1.+2.*DGAM*COS(DTHCM)+DGAM*DGAM)
169:       if ( DTHCM.gt.RAD ) DFAI = -DFAI
170:       DFAI = ASIN(DFAI)
171:       if ( DTHCM.gt.RAD ) DFAI = DFAI + PAI
172:       DCF = COS(DFAI)
173:       DROH = SQRT(1.0-DCF**2)
174:       DFAI0 = PAI*(2.0*R(I,4)-1.0)
175:       BBB(LSDED(I)) = DROH*COS(DFAI0)
176:       AAA(LSDED(I)) = DROH*SIN(DFAI0)
177:       CCC(LSDED(I)) = DCF
178:       E0 = E0/931.481 + XM1
179:       ALP = XMM + 2.0*XM2*E0
180:       A = (E0+XM2)**2 - (E0**2-XM1**2)*DCF**2
181:       B = -ALP*(E0+XM2)
182:       C = SQRT((E0**2-XM1**2)*DCF**2*(ALP**2-4.*A*XM3**2))
183:       if ( DCF.lt.0.0 ) C = -C
184:       W3 = (-B+C) /2./A
185:       EEE(I) = (W3-XM3)*931.481*1.0E+6
186: 150 continue
187: C
188: C ..... ENERGY GROUP & WEIGHT .....
189: C
190:       call BSVDEC( ENGYB, NGROUP+1, EEE, IWK, NSS )
191:
192:       do 160 I = 1, NSS
193:           IGG(LSDED(I)) = MIN(NGROUP,MAX(1,IWK(I)))
194:           WWW(LSDED(I)) = 1.0
195:           TTT(LSDED(I)) = 0.0D0

```

```

196: 160 continue
197: C
198: C ..... ROTATION .....
199: C
200:       if ( XDIREC.eq.2. ) then
201: *VOCL LOOP,NOVREC
202:       do 170 I = 1, NSS
203:           K = LSDED(I)
204:           DAAA = AAA(K)
205:           DBBB = BBB(K)
206:           DCCC = CCC(K)
207:           AAA(K) = DCCC
208:           BBB(K) = DAAA
209:           CCC(K) = DBBB
210:           DXXX = XXX(K) - X0
211:           DYYY = YYY(K) - Y0
212:           DZZZ = ZZZ(K) - Z0
213:           XXX(K) = DZZZ + X0
214:           YYY(K) = DXXX + Y0
215:           ZZZ(K) = DYYY + Z0
216: 170 continue
217:       else if ( XDIREC.eq.3. ) then
218: *VOCL LOOP,NOVREC
219:       do 180 I = 1, NSS
220:           K = LSDED(I)
221:           DAAA = AAA(K)
222:           DBBB = BBB(K)
223:           DCCC = CCC(K)
224:           AAA(K) = DBBB
225:           BBB(K) = DCCC
226:           CCC(K) = DAAA
227:           DXXX = XXX(K) - X0
228:           DYYY = YYY(K) - Y0
229:           DZZZ = ZZZ(K) - Z0
230:           XXX(K) = DYYY + X0
231:           YYY(K) = DZZZ + Y0
232:           ZZZ(K) = DXXX + Z0
233: 180 continue
234:       else if ( XDIREC.ge.4. ) then
235:           DTH = TH1*PAI180
236:           DFA = FA1*PAI180
237:           DPU = PU1*PAI180
238:           DT11 = COS(DTH)*COS(DFA)*COS(DPU) - SIN(DFA)*SIN(DPU)
239:           DT12 = -COS(DTH)*COS(DFA)*SIN(DPU) - SIN(DFA)*COS(DPU)
240:           DT13 = SIN(DTH)*COS(DFA)
241:           DT21 = COS(DTH)*SIN(DFA)*COS(DPU) + COS(DFA)*SIN(DPU)
242:           DT22 = -COS(DTH)*SIN(DFA)*SIN(DPU) + COS(DFA)*COS(DPU)
243:           DT23 = SIN(DTH)*SIN(DFA)
244:           DT31 = -SIN(DTH)*COS(DPU)
245:           DT32 = SIN(DTH)*SIN(DPU)
246:           DT33 = COS(DTH)
247: *VOCL LOOP,NOVREC
248:       do 190 I = 1, NSS
249:           K = LSDED(I)
250:           DAAA = AAA(K)
251:           DBBB = BBB(K)
252:           DCCC = CCC(K)
253:           AAA(K) = DT11*DAAA + DT21*DBBB + DT31*DCCC
254:           BBB(K) = DT12*DAAA + DT22*DBBB + DT32*DCCC
255:           CCC(K) = DT13*DAAA + DT23*DBBB + DT33*DCCC
256: 190 continue
257:       if ( XDIREC.ge.5. ) then
258: *VOCL LOOP,NOVREC
259:       do 200 I = 1, NSS
260:           K = LSDED(I)

```

src/shared/fnssoc.f

```
261:          DAAA   = XXX(K) - X0
262:          DBBB   = YYY(K) - Y0
263:          DCCC   = ZZZ(K) - Z0
264:          XXX(K)  = DT11*DAAA + DT21*DBBB + DT31*DCCC + X0
265:          YYY(K)  = DT12*DAAA + DT22*DBBB + DT32*DCCC + Y0
266:          ZZZ(K)  = DT13*DAAA + DT23*DBBB + DT33*DCCC + Z0
267: 200      continue
268:      end if
269:  end if
270: C
271:      return
272: C
273:      end
```

DATA

src/shared/fssamp.f

```

1:      subroutine FSSAMP( A, H, X, Y, Z, AA, BB, CC, W,
2:      &                  E, NS, NUSE, JNUCS, JWGT, XXXF, YYYY,
3:      &                  ZZZF, INUF, EEEF, WWWF, NNUCID, KNUCID,
4:      &                  IRAND, IWRK1, IWRK2, RWRK, IERR )
5: C=====
6: C purpose : sample sources from fissions using coordinates, nuclides
7: C           and energies from file.
8: C called in: syssrc
9: C-----
10: C arguments ( i=in, o=out, w=work )
11: C
12: C io A(*) : dynamic memory array (task shared)
13: C io H(*) : dynamic memory array (task local)
14: C
15: C o X(NS),Y(NS),Z(NS) : coordinate to be sampled
16: C o AA(NS),BB(NS),CC(NS) : directions to be sampled
17: C o W(NS) : neutron weights
18: C o E(NS) : energies to be sampled
19: C i NS : number of source to be sampled
20: C i NUSE : number of source data used
21: C i JNUCS : if non-zero, sample exact energy using INUF & EEEF
22: C           (MVP only)
23: C i JWGT : weight data WWWF is not given if zero.
24: C i XXXF(NUSE),YYYY(NUSE),ZZZF(NUSE) : coordinate from file.
25: C i INUF(NUSE),EEEF(NUSE) : colliding nuclide #'s, and energies
26: C           induced fission reaction
27: C           (MVP only)
28: C i WWWF(NUSE) : neutron weights
29: C i NNUCID : number of nuclides in INUF
30: C i KNUCID(NNUCID) : list of nuclide #'s in INUF
31: C io IRAND : seed of random numbers
32: C w IWRK1(NS), IWRK2(NS) : working array (integer)
33: C           assinged.
34: C w RWRK(*) : working array (real) at least NS*5 elements must be
35: C           assinged.
36: C-----
37:      implicit real*8(D)
38: C
39:      real A(*)
40:      real H(*)
41: C
42:      real*8 X(NS), Y(NS), Z(NS), AA(NS), BB(NS), CC(NS)
43:      real*8 W(NS)
44:      real*8 E(NS)
45: C
46:      real*8 XXXF(NUSE), YYYY(NUSE), ZZZF(NUSE)
47:      integer INUF(NUSE)
48:      real EEEF(NUSE)
49:      real WWWF(NUSE)
50: C
51:      integer IWRK1(NS)
52:      integer IWRK2(NS)
53:      real RWRK(5*NS)
54: C
55:      real*8 PI, PI2
56:      parameter( PI = 3.1415926535897932D0 )
57:      parameter( PI2 = 2.0D0*PI )
58: C
59: C-----
60: C
61: C determine source particle #'s to be used
62: C
63:      if ( NUSE.gt.NS ) then
64: C
65: C ... make random permutation of integers from 1 to NS

```

```

66: C      on IWRK1(1:NS)...
67: C
68:      do 100 I = 1, NS
69:          IWRK1(I) = I
70:      100 continue
71:      call RANU2( IRAND, RWRK(1), NS, ICOD )
72:      call MSORT( NS, RWRK(1), IWRK1, RWRK(NS+1), IWRK2 )
73: C
74: C ... replace integers from 1 to NS with integers from NS+1 to NUSE
75: C      at random.
76: C
77:      do 120 K = NS + 1, NUSE, NS
78:          K2 = MIN(K+NS-1,NUSE)
79:          NNN = K2 - K + 1
80:          call RANU2( IRAND, RWRK(1), NNN, ICOD )
81:          do 110 II = K, K2
82:              IP = MIN(II,INT(RWRK(II-K+1)*II+1))
83:              if ( IP.le.NS ) IWRK1(IP) = II
84:          110 continue
85:      120 continue
86: C
87:      else
88:          call RANU2( IRAND, RWRK(1), NS, ICOD )
89:          do 130 I = 1, NS
90:              IWRK1(I) = MIN(NUSE,INT(RWRK(I)*NUSE+1))
91:          130 continue
92:      end if
93: C
94: C ... set coordinates & sample directions isotropically
95: C
96:      call RANU2( IRAND, RWRK(1), 2*NS, ICOD )
97: C
98:      do 140 I = 1, NS
99:          IP = IWRK1(I)
100:          X(I) = XXXF(IP)
101:          Y(I) = YYYY(IP)
102:          Z(I) = ZZZF(IP)
103: C
104:          DW = 1. - 2.*RWRK(I)
105:          DA = SQRT(1.-DW*DW)
106:          DPHI = PI2*RWRK(NS+I)
107:          DU = DA*COS(DPHI)
108:          DV = DA*SIN(DPHI)
109:          DS = 1./SQRT(DU*DU+DV*DV+DW*DW)
110:          AA(I) = DU*DS
111:          BB(I) = DV*DS
112:          CC(I) = DW*DS
113:      140 continue
114: C
115: C ... sample weights ...
116: C
117:      if ( JWGT.ne.0 ) then
118:          DWSUM = 0.0D0
119:          do 150 I = 1, NS
120:              IP = IWRK1(I)
121:              W(I) = WWWF(IP)
122:              DWSUM = DWSUM + W(I)
123:          150 continue
124:          DWSUM = DBLE(NS) /DWSUM
125:          do 160 I = 1, NS
126:              W(I) = W(I)*DWSUM
127:          160 continue
128:      else
129:          do 170 I = 1, NS
130:              W(I) = 1.0D0

```

src/shared/fssamp.f

```
131: 170    continue
132:    end if
133: C
134: C ... sample energy ....
135: C
136: C === calls NUCFIS routine once per source ===
137: C
138:    if ( JNUCS.ne.0 ) then
139:      do 180 I = 1, NS
140:        call NUCFIS( A, H,
141:          &          E(I), 1, INUF(IWRK1(I)), EEFF(IWRK1(I)), IRAND,
142:          &          IERR, RWK )
143: 180    continue
144:    end if
145: C
146:    return
147:    end
```


src/shared/geomin.f

```

1: *VOCL TOTAL,SCALAR
2: subroutine GEOMIN( ICALL, A, LIMIT, CHA, LIMITC, LAST,
3: & MAXSF, JDEBG, JHLAT, JLATT, JREFL, JSIMP,
4: & JTLLT, JFISX, JSTGR )
5: C<MVP>=====
6: C PURPOSE: INPUT GEOMETRY RELATED DATA (BODY & ZONE DATA)
7: C CALLED IN: INTRO
8: C CALLS: FREADS BOD??? CCOMP CELTR CNTERR HEADER ITRGSP KEEP
9: C KEEPC KEPV LATTIN LATTR LBZCNT MEMERR MRZONE NAMLST NESTCK
10: C PRSTOP REMAIN REMAINC RESET RESIZE RESIZEC RIUNIT RSTBOD
11: C RWIND SPNUMS SPREG TLOREG TYPBOD ZLBZ ZONEIN
12: C-----
13: C arguments (i=input, o=output, w=work)
14: C
15: C o ICALL : set to non zero when called ... thats all
16: C i o A(*) : dynamic memory array
17: C i LIMIT : effective memory size of A(*)
18: C i o CHA(*) : character dynamic memory array
19: C i LIMITC : size of CHA(*)
20: C x LAST : probably has historical meaning only ??
21: C o MAXSF : maximum number of necessary surface data to compose a zone
22: C i(o) JDEBG, JHLAT, JLATT, JREFL, JSIMP, JTLLT, JFISX, JSTGR :
23: C option flags (some may change value).
24: C see (mvp/gmvp)/INC/_FLAGS.
25: C=====
26: CCCC include '../shared/INC/_ARRAY'
27: C
28: C INTEGER IA(LIMIT)
29: C
30: C real A(*)
31: C character*4 CHA(*)
32: C
33: C include 'INC/_LISTOFF'
34: C include 'INC/_PGEOM'
35: C include 'INC/_SIZES'
36: C include 'INC/_PTLSP'
37: C include 'INC/_LISTON'
38: C include 'INC/_WORDL'
39: C include 'INC/_LNAM'
40: C include 'INC/_IUNIT'
41: C include 'INC/_LISTON'
42: C
43: C
44: C integer JDEBG(*)
45: C
46: C
47: C character TYPE*3, TYPE0*3, CTYP*4
48: C character TYP*32, LINE*72
49: C
50: C .... TEMPORARY INPUT ARRAY OF BODY & DATA .....
51: C
52: C parameter( MXLBD = 35 )
53: C real*8 BD(MXLBD)
54: C
55: C ICALL = 1
56: C
57: C-----
58: C .... CHECK WHETHER DEPS & DINF WERE INPUT OR NOT .....
59: C-----
60: C
61: C if ( DEPS.eq.0.0 ) then
62: C DEPS = 1.0D-5
63: C write(IOG,7000) DEPS
64: C 7000 format(/IX,'<<MESSAGE>> The minimum distance (DEPS) ',
65: & 'has not been input, or given zero value. '//

```

```

66: C##<2007/03/14:PN3:
67: C## & ' Set to ',E12.5,' !')
68: C## & ' Set to ',1P,E12.5,' !')
69: C##>
70: C call CNTERR( 'MESSAGE' )
71: C end if
72: C
73: C if ( DINF.eq.0.0 .or. DINF.gt.2.0D0**128 ) then
74: C DINF = 1.0D30
75: C write(IOG,7020) DINF
76: C 7020 format(/IX,'<<MESSAGE>> The infinite distance (DINF) ',
77: & 'has not been input or given too large value. '//
78: C##<2007/03/14:PN3:
79: C## & ' Set to ',E12.5,' !')
80: C## & ' Set to ',1P,E12.5,' !')
81: C##>
82: C call CNTERR( 'MESSAGE' )
83: C end if
84: C
85: C-----
86: C .... CHECK WHETHER NMEMO WAS INPUT OR NOT .....
87: C-----
88: C IF(NMEMO.LE.0) THEN
89: C NMEMO = 5
90: C WRITE(IOG,*) ' !! NMEMO WAS SET TO 5 IN GEOMIN. '
91: C ENDIF
92: C-----
93: C .... READ LATTICE DATA .....
94: C-----
95: C if ( JLATT.ne.0 ) then
96: C call RWIND( IUG1 )
97: C NLATT = 0
98: C
99: C 100 call CPLIN( IOIN, LINE, NCHAR, IEND )
100: C
101: C if ( IEND.ne.0 ) then
102: C write(IMG,*) 'XXX Unexpected end of input-data ',
103: & 'in lattice data block.'
104: C write(IMG,*) ' Probably missing "END" line.'
105: C call CNTERR( 'FATAL' )
106: C call PRSTOP( 0, 'Geometry data input error.' )
107: C stop 888
108: C end if
109: C
110: C ... IDLAT(*) must be the first input entry on a line.
111: C
112: C if ( LINE(1:1).ne.'%' .and. INDEX(LINE(:NCHAR), 'IDLAT') .ne.0 )
113: C & then
114: C KKK = INDEX(LINE(:NCHAR), 'IDLAT')
115: C if ( KKK.eq.1 .or. (KKK.gt.1 .and. LINE(1:KKK-1).eq.' ') )
116: C & then
117: C NLATT = NLATT + 1
118: C end if
119: C end if
120: C write(IUG1, '(A)') LINE
121: C if ( LINE(1:4).eq.'END ' ) go to 110
122: C
123: C go to 100
124: C
125: C 110 call RWIND( IUG1 )
126: C
127: C call RESET
128: C
129: C
130: C call KEPV( A(1), 'IDLAT', LIDLAT, NLATT, 'I4', LAST, 0, JDEBG )

```

src/shared/geomin.f

```

131:      call KEPV( A(1), 'LTYP', LLTYP, NLATT, 'I4', LAST, 0, JDEBG )
132:      call KEPV( A(1), 'NVLAT', LNVLAT, 4*NLATT, 'I4', LAST, 0, JDEBG
133:      &
134:      call KEPV( A(1), 'SZLAT', LSZLAT, 8*NLATT, 'R8', LAST, 0.0D0,
135:      & JDEBG )
136:      call KEEP( 'IPLAT', LIPLAT, NLATT+1, 'I4', LAST, JDEBG )
137: C
138: C ..... SIZE OF KLATT ARRAY WILL BE DETERMINED DYNAMICALLY IN LATTIN.
139: C
140:      call LATTIN( IUG1, A, A, LIMIT, CHA, LIMITC, NLATT, A(LIDLAT),
141:      & A(LLTYP), A(LNVLAT), A(LSZLAT), A(LIPLAT), NPNND,
142:      & LKSLAT, LKLATT, LKHLAT, LPNND, LKNND,
143:      & NPPPF, LPPPF, LKPPF,
144:      & JHLAT, JFISX, JSTGR, JDEBG )
145:      end if
146: C
147: C -----
148: C .... SET START ADDRESS OF SURFACE DATA
149: C -----
150: C
151:      call REMAIN( 'SDA', NLTEMP, 'R8', LAST )
152:      call KEEP( 'SDA', LSDA, NLTEMP, 'R8', LAST, JDEBG )
153:      LSDAT = LSDA
154: C
155:      NBODY = 0
156:      NBODI = 0
157:      NSURF = 0
158: C
159: C
160:      call RWIND( IUB )
161: C
162: C .... WRITE MESSAGE .....
163: C
164:      call HEADER( IOG, 'BODY DATA' )
165: C
166:      write(IOG,7040) (I,I=1,8)
167: C
168: c7040 format(//1X,'NO. TYPE ID',8(' DATA',I2,5X)/1X,
169: c & '-----',8('-----'))
170: 7040 format(//1X,' NO. TYPE ID',8(' DATA',I2,5X)/1X,
171: & '-----',8('-----'))
172: C
173: 7060 format(1X,4X,1X,3X,1X,6X,: ' < ',A4,' > ',: ' < ',A4,' > ',:
174: & ' < ',A4,' > ',: ' < ',A4,' > ',: ' < ',A4,' > ',:
175: & ' < ',A4,' > ',: ' < ',A4,' > ',: ' < ',A4,' > ')
176: c7080 format(1X,I4,1X,A3,1X,I6,1P,8(E13.5:))/(1X,15X,1P,8E13.5:)
177: c7100 format(1X,I4,1X,A3,1X,I6,1P,8(:3x,I7:,3X)/
178: c & (:1X,15X,1P,8(3x,I7:,3X)))
179: c7120 format(1X,'-----',8('-----'))
180: 7080 format(1X,I5,1X,A3,1X,I6,1P,8(E13.5:))/(1X,15X,1P,8E13.5:)
181: 7100 format(1X,I5,1X,A3,1X,I6,1P,8(:3x,I7:,3X)/
182: & (:1X,16X,1P,8(3x,I7:,3X)))
183: 7120 format(1X,'-----',8('-----'))
184: C
185: C
186: C .... READ BODY TYPE .....
187: C
188: C
189:      TYPE0 = ' '
190: C
191: 120 call FREADS( TYP, NLEN, IEND )
192:      if ( IEND.ne.0 ) then
193:          write(IMG,*) 'XXX Unexpected end of data in $GEOMETRY',
194:          & ' block.'
195:          write(IMG,*)

```

```

196:      & ' Probably missing a termination line of the data block.'
197:      call CNTERR( 'FATAL' )
198:      call PRSTOP( 0, 'Geometry data input error.' )
199:      stop 888
200:      end if
201: C
202:      TYPE = TYP(1:3)
203:      if ( TYPE.eq.'END' ) go to 140
204: C
205:      do 130 I = 1, MXLBD
206:          BD(I) = 0.0
207:      130 continue
208: C
209: C -----
210: C BODY DATA --> SURFACE DATA
211: C -----
212: C
213:      ICAL = 0
214: C
215: C .... ICAL IS (NUMBER OF SDA DATA FOR A BODY)+1 .....
216: C
217: C
218: C .... initialize guess for geometry extent ....
219: C
220:      GRANGE(1) = DINF
221:      GRANGE(2) = -DINF
222:      GRANGE(3) = DINF
223:      GRANGE(4) = -DINF
224:      GRANGE(5) = DINF
225:      GRANGE(6) = -DINF
226: C
227:      if ( TYPE.eq.'RPP' ) call BODRPP( ICAL, A(LSDAT), LSDA, BD, IBODN,
228:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
229:      if ( TYPE.eq.'BOX' ) call BODBOX( ICAL, A(LSDAT), LSDA, BD, IBODN,
230:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
231:      if ( TYPE.eq.'SPH' ) call BODSPH( ICAL, A(LSDAT), LSDA, BD, IBODN,
232:      & KBSDA, NS, LSDAT, GRANGE, ICODE )
233:      if ( TYPE.eq.'RCC' ) call BODRCC( ICAL, A(LSDAT), LSDA, BD, IBODN,
234:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
235:      if ( TYPE.eq.'CYL' ) call BODCYL( ICAL, A(LSDAT), LSDA, BD, IBODN,
236:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
237:      if ( TYPE.eq.'RCL' ) call BODRCL( ICAL, A(LSDAT), LSDA, BD, IBODN,
238:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
239:      if ( TYPE.eq.'TRC' ) call BODTRC( ICAL, A(LSDAT), LSDA, BD, IBODN,
240:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
241:      if ( TYPE.eq.'RHP' ) call BODRHP( ICAL, A(LSDAT), LSDA, BD, IBODN,
242:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
243:      if ( TYPE.eq.'HEX' ) call BODHEX( ICAL, A(LSDAT), LSDA, BD, IBODN,
244:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
245:      if ( TYPE.eq.'ELL' ) call BODELL( ICAL, A(LSDAT), LSDA, BD, IBODN,
246:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
247:      if ( TYPE.eq.'ARB' ) call BODARB( ICAL, A(LSDAT), LSDA, BD, IBODN,
248:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
249:      if ( TYPE.eq.'WED' .or. TYPE.eq.'RAW' ) call BODWED( ICAL,
250:      & A(LSDAT), LSDA, BD, IBODN, KBSDA, NS, LSDAT, DINF, GRANGE,
251:      & ICODE )
252:      if ( TYPE.eq.'HAF' )
253:      & call BODHAF( ICAL, A(LSDAT), LSDA, BD, IBODN, KBSDA, NS, LSDAT,
254:      & ICODE )
255:      if ( TYPE.eq.'GEL' ) call BODGEL( ICAL, A(LSDAT), LSDA, BD, IBODN,
256:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
257: C ... ELLIPTIC TORUS 11/14/1990 & 2/18/1993 .....
258:      if ( TYPE.eq.'ELT' ) call BODELT( ICAL, A(LSDAT), LSDA, BD, IBODN,
259:      & KBSDA, NS, LSDAT, DINF, GRANGE, ICODE )
260:      if ( TYPE.eq.'TEC' ) call BODTEC( ICAL, A(LSDAT), LSDA, BD, IBODN,

```

src/shared/geomin.f

```

261:      & KBSDA, NS, LSDAT, DINP, GRANGE, ICODE )
262:      if ( TYPE.eq.'GQS' )
263:      & call BODGQS( ICAL, A(LSDAT), LSDA, BD, IBODN, KBSDA, NS, LSDAT,
264:      & ICODE )
265: C
266:      if ( TYPE.eq.'BBC' ) call BODBBC( ICAL, A(LSDAT), LSDA, BD, IBODN,
267:      & KBSDA, NS, LSDAT, DINP, GRANGE, ICODE )
268: C
269: C
270:      if ( ICAL.eq.0 ) then
271:      write(IMG,*) 'XXX Body type <', TYPE, '> is not supported!!'
272:      call CNTERR( 'FATAL' )
273:      else if ( IBODN.lt.0 ) then
274:      write(IMG,*) 'XXX Body ', IBODN,
275:      & ' : Body ID must be positive!!'
276:      call CNTERR( 'FATAL' )
277:      else if ( TYPE.ne.'BBC'.and.NS.eq.0 ) then
278:      write(IMG,*) 'XXX Body ', TYPE, ' of ID ', INT(BD(1)),
279:      & ' does not have any bounding surfaces.'
280:      write(IMG,*) ' : Maybe some geometry parameters of the body'
281:      & ' have invalid or critical values.'
282:      call CNTERR( 'FATAL' )
283:      else
284: C
285:      if ( TYPE.ne.TYPE0 ) then
286:      if ( TYPE.eq.'BXX' ) write(IOG,7060) 'CX', 'CY', 'CZ',
287:      & 'V1X', 'V1Y', 'V1Z', 'V2X', 'V2Y', 'V2Z', 'V3X', 'V3Y',
288:      & 'V3Z'
289:      if ( TYPE.eq.'RPP' ) write(IOG,7060) 'X1', 'X2', 'Y1', 'Y2',
290:      & 'Z1', 'Z2'
291:      if ( TYPE.eq.'SPH' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'R'
292:      if ( TYPE.eq.'RCC' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'HX',
293:      & 'HY', 'HZ', 'R'
294:      if ( TYPE.eq.'RCL' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'HX',
295:      & 'HY', 'HZ', 'R', 'VX', 'VY', 'VZ'
296:      if ( TYPE.eq.'CYL' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'H',
297:      & 'R'
298:      if ( TYPE.eq.'ELL' ) write(IOG,7060) 'F1X', 'F1Y', 'F1Z',
299:      & 'F2X', 'F2Y', 'F2Z', 'D'
300:      if ( TYPE.eq.'GEL' ) write(IOG,7060) 'CX', 'CY', 'CZ',
301:      & 'V1X', 'V1Y', 'V1Z', 'V2X', 'V2Y', 'V2Z', 'R1', 'R2',
302:      & 'R3'
303:      if ( TYPE.eq.'TRC' ) write(IOG,7060) 'C1X', 'C1Y', 'C1Z',
304:      & 'C2X', 'C2Y', 'C2Z', 'R1', 'R2'
305:      if ( TYPE.eq.'RHP' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'H',
306:      & 'DX'
307:      if ( TYPE.eq.'HEX' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'HX',
308:      & 'HY', 'HZ', 'D', 'VX', 'VY', 'VZ'
309:      if ( TYPE.eq.'ARB' ) write(IOG,7060) 'C1X', 'C1Y', 'C1Z',
310:      & 'C2X', 'C2Y', 'C2Z', 'C3X', 'C3Y', 'C3Z', 'C4X', 'C4Y',
311:      & 'C4Z', 'C5X', 'C5Y', 'C5Z', 'C6X', 'C6Y', 'C6Z', 'C7X',
312:      & 'C7Y', 'C7Z', 'C8X', 'C8Y', 'C8Z', 'S1', 'S2', 'S3',
313:      & 'S4', 'S5', 'S6'
314:      if ( TYPE.eq.'WED'.or. TYPE.eq.'RAW' ) write(IOG,7060)
315:      & 'CX', 'CY', 'CZ', 'V1X', 'V1Y', 'V1Z', 'V2X', 'V2Y',
316:      & 'V2Z', 'V3X', 'V3Y', 'V3Z'
317:      if ( TYPE.eq.'HAF' ) write(IOG,7060) 'NX', 'NY', 'NZ', 'D'
318:      if ( TYPE.eq.'ELT' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'NX',
319:      & 'NY', 'NZ', 'RM', 'R1', 'R2', 'TLT'
320:      if ( TYPE.eq.'TEC' ) write(IOG,7060) 'CX', 'CY', 'CZ', 'HX',
321:      & 'HY', 'HZ', 'AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'RT'
322:      if ( TYPE.eq.'GQS' ) write(IOG,7060) 'X2', 'Y2', 'Z2', 'XY',
323:      & 'YZ', 'ZX', 'X ', 'Y ', 'Z ', 'C '
324:      if ( TYPE.eq.'BBC' ) write(IOG,7060) 'BOD1', 'BOD2', '...'
325:      TYPE0 = TYPE

```

```

326:      end if
327: C
328:      NSRFO = NSURF
329:      NSURF = NSURF + NS
330:      NBODY = NBODY + 1
331: C ... number of body geometry data as input ...
332:      NBODI = NBODI + (ICAL-1)
333:
334:      if ( TYPE.eq.'BBC' ) then
335:      write(IOG,7100) NBODY, TYPE, INT(BD(1)),
336:      & (NINT(BD(I)),I=2,ICAL)
337:      else
338:      write(IOG,7080) NBODY, TYPE, INT(BD(1)), (BD(I),I=2,ICAL)
339:      end if
340: C
341: C write(IUB) KBSDA, IBODN
342: C ... output first surface # instead of SDA pointer
343: C
344:      call TYPBOD( TYPE, '>', IBTYPE )
345: C
346:      write(IUB) NSRFO + 1, IBODN, IBTYPE, GRANGE, ICAL - 1
347:      write(IUB) (BD(I),I=2,ICAL)
348:      end if
349: C
350: C-----
351: C ... TOO MANY BODIES!! STOP WITH MESSAGE
352: C-----
353: C
354: C
355:      if ( LSDAT-LSDA.gt.NLTEMP ) then
356:      write(IMG,*) 'XXX You need more memory to process body data!'
357:      write(IMG,*) ' Used memory for surface data = ', LSDAT
358:      & - LSDA, '(words)'
359:      write(IMG,*) ' Necessary at least ', LSIZ(LSDAT) - 1,
360:      & '(words)'
361:      write(IMG,*) ' (currently LIMIT = ', LIMIT, ' ) '
362:      call PRSTOP( 1, 'MEMORY OVER.' )
363:      stop 999
364:      end if
365: C
366:      go to 120
367: C
368:      140 call RESET
369:      write(IOG,7120)
370: C
371: C-----
372: C ... CALCULATE LENGTH OF SDA ARRAY
373: C-----
374: C
375:      NSDA = (LSDAT-LSDA)/MWORD
376:      call RESIZE( 'SDA', LSDA, NSDA, 'R8', LAST )
377: C
378: C-----
379: C TO SAVE SDA POINTER & ID'S OF EACH BODY AS 'IBSDA' DATA
380: C-----
381: C
382:      call KEEP( 'IBSDA', LIBSDA, 3*(NBODY+1), 'I4', LAST, JDEBG )
383:      call KEEP( 'IPSDA', LIPSDA, 3*(NSURF+1), 'I4', LAST, JDEBG )
384:      call KEEP( 'IBODI', LIBODI, NBODY+1, 'I4', LAST, JDEBG )
385:      call KEEP( 'GRBOD', LGRBOD, 6*NBODY, 'R8', LAST, JDEBG )
386:      call KEEP( 'DBODI', LDBODI, NBODI, 'R8', LAST, JDEBG )
387:
388:      if ( LSIZ(LAST).gt.LIMIT ) then
389:      call MEMERR( 'GEOMIN',
390:      & ' "Body-Surface matching tables(IBSDA,IPSDA).',

```

src/shared/geomin.f

```

391:      &          LSIZ(LAST), LIMIT )
392:      end if
393: C
394: C ... restore BODN & BSDA DATA ....
395: C ... I/O unit IUB is read in RSTBOD
396: C
397:      call RSTBOD( NBODY, A(LIBSDA), A(LGRBOD), NBODI, A(LIBODI),
398:      &          A(LDBODI), A(LIPSDA), NSURF, A(LSDA), NSDA )
399: C
400: C-----
401: C Check duplicate use of Body ID ( IBSDA(2,i) ) ...
402: C-----
403: C
404: c      call CHKDUP( A(LIBSDA+1), NBODY, 3, NERR, IOG, 'BODY ID' )
405: c      if ( NERR.gt.0 ) call CNTERR( 'FATAL' )
406: C
407: C-----
408: C .... SUMMARY OF BODY INPUT .....
409: C-----
410: C
411:      write(IOG,7140) NBODY, NSURF, LSDA, LSDAT - 1, NSDA, MWORD
412: c##<2007/03/14:PN3:
413: c7140 format(/3X,'==== SUMMARY OF BODY INPUTS =====')
414: c## &          3X,' TOTAL NUMBER OF BODIES      =',I6/3X,
415: c## &          ' TOTAL NUMBER OF SURFACES      =',I6/3X,
416: c## &          ' SURFACE DATA (SDA) IS FROM ',I12/3X,
417: c## &          ' TO ',I12/3X,
418: c## &          ' (ADDRESSES IN ARRAY AREA) '/3X,
419: c## &          ' LENGTH OF SDA ARRAY      = ',I6,'*',I1,' (WORD)'/3X,
420: c## &          '=====')
421: 7140 format(
422:      &/3X,'==== SUMMARY OF BODY INPUTS ====='
423:      &/3X,' TOTAL NUMBER OF BODIES      =',I12
424:      &/3X,' TOTAL NUMBER OF SURFACES      =',I12
425:      &/3X,' SURFACE DATA (SDA) IS FROM ',I12
426:      &/3X,' TO ',I12
427:      &/3X,' (ADDRESSES IN ARRAY AREA)'
428:      &/3X,' LENGTH OF SDA ARRAY      =',I10,'*',I1,' (WORDS)'
429:      &/3X,'=====')
430: c##>
431: C
432: C-----
433: C .... READ ZONE DATA & COUNT THE NUMBER OF INPUT ZONES .....
434: C and prepare area to save input-zone names ...
435: C (DATA ARE SEND TO UNIT 'IUG1' AND ARE RE-READ IN THERE).
436: C-----
437: C
438:      NINPZ = 0
439:      NCELL = 0
440:      call RWIND( IUG1 )
441:      IJSP = 0
442: C
443:      call REMAINC( 'IZNAM', NLTEMP, 'C12', LASTC )
444:      call KEEPC( 'IZNAM', LIZNAM, NLTEMP, 'C12', LASTC, JDEBG )
445: C
446: C .... Implicit loop to transfer zone input to working file ...
447: C
448: 150 call CPLIN( IOIN, LINE, NCHAR, IEND )
449: C
450:      if ( IEND.ne.0 ) then
451:          write(IMG,*) 'XXX Unexpected end of data in $GEOMETRY',
452:          &          ' block.'
453: C
454:          write(IMG,*)
455:          &          ' Probably missing termination line of the data block.'

```

```

456:      call CNTERR( 'FATAL' )
457:      call PRSTOP( 0, 'Geometry data input error.' )
458:      stop 888
459: end if
460: C
461:      if ( LINE(1:1).eq.'%' ) go to 160
462: C
463:      if ( LINE(1:4).eq.'END ' ) then
464:          call CNTERR( 'WARNING' )
465:          write(IMG,7160)
466: 7160 format(/1X,'!!! Termination of geometry(tally ',
467:      &          'region) input by "END" line will be obsolete. '//
468:      &          ' Please use "$END GEOMETRY" line.//')
469:      end if
470: C
471: C .... exit implicit loop ...
472: C (put '$END' to mark end of data)
473: C
474:      if ( LINE(1:4).eq.'END ' .or. LINE(1:7).eq.'#REGION'
475:      &      .or. LINE(1:13).eq.'#TALLY REGION'
476:      &      .or. ISQST2(LINE,'$END','GEOM*').ne.0 ) then
477:          write(IUG1,'(A)') '$END'
478:          go to 170
479:      end if
480: C
481: C
482: C .... number of input zones is counted here ...
483: C
484:      if ( LINE(1:5).ne.' ' .and.LINE(1:1).ne.'#' .and.IJSP.eq.0 ) then
485:          NINPZ = NINPZ + 1
486:      end if
487: C
488: C
489: C .... CELL OR SUBFRAME NAME DATA ...
490: C
491:      if ( LINE(1:1).eq.'#' ) then
492:          if ( LINE(1:6).eq.'#SPACE' .or. LINE(1:9).eq.'#SUBFRAME' ) then
493:              IJSP = 1
494:          else if ( LINE(1:10).eq.'#END SPACE'
495:          &          .or. LINE(1:13).eq.'#END SUBFRAME' ) then
496:              IJSP = 0
497:          else if ( LINE(1:5).eq.'#CELL' .or. LINE(1:2).eq.'# ' ) then
498:              NCELL = NCELL + 1
499:          end if
500:      end if
501: C
502: 160 write(IUG1,'(A)') LINE
503: C
504:      go to 150
505: C
506: C ..... END OF ZONE ( & SUBFRAME ) DATA TRANSFER .....
507: C
508: 170 continue
509: C
510:      if ( NINPZ.gt.NLTEMP ) then
511:          write(IMG,7180) (LIZNAM-1)*LEN(CHA(1)) + NINPZ*12
512: 7180 format(/1X,'XXX(GEOMIN) Character memory is insufficient to ',
513:      &          'save input-zone names.'/1X,' At least ',I12,
514:      &          ' bytes are necessary.//')
515:          call CNTERR( 'FATAL' )
516:          call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
517:          stop 999
518:      end if
519: C
520:      call RESIZEC( 'IZNAM', LIZNAM, NINPZ, 'C12', LASTC )

```

src/shared/geomin.f

```

521: C
522: C
523: C-----
524: C FLAG FOR SPECIAL REGION SPECIFICATION --> UNIT IUG2 IF ANY
525: C-----
526:       JSREG = 0
527:       if ( LINE(1:7).eq.'#REGION' .or. LINE(1:13).eq.'#TALLY REGION' )
528:         & then
529:           call RWIND( IUG2 )
530:           JSREG = 1
531:         end if
532: C
533:       call KEEP( 'KREG', LKREG, NINPZ, 'I4', LAST, JDEBG )
534:       call KEVP( A(1), 'KMAT', LKMAT, NINPZ*2, 'I4', LAST, 0, JDEBG )
535: C
536:       if ( LSIZ(LAST).gt.LIMIT ) then
537:         call MEMERR( 'GEOMIN', 'Material ID for input-zones.',
538:           & LSIZ(LAST), LIMIT )
539:       end if
540: C
541: C .. REGION NAME IN 'LNAM' CHARACTERS ....
542: C
543:       CTYP = ' '
544:       write(CTYP, '(('C',I3))' ) LNAM
545:       call CCOMP( CTYP, LEN(CTYP), CTYP, IEL )
546:       LLC = ICLEN(CTYP)
547: C
548:       call KEEPC( 'KREGN', LKREGN, NINPZ, CTYP(:LLC), LASTC, JDEBG )
549: C
550:       if ( LASTC.gt.LIMITC ) then
551:         write(IMG,7200) (LASTC-1)*LEN(CHA(1))
552: 7200 format(/1X,'XXX(GEOMIN) Character memory is insufficient to ',
553:         & 'save region names.//1X,' At least ',I12,' bytes',
554:         & ' are necessary.//')
555:         call CNTERR( 'FATAL' )
556:         call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
557:         stop 999
558:       end if
559: C
560: C-----
561: C GET AREAS FOR LATTICE CELL DATA
562: C-----
563: C
564:       if ( JLATT.ne.0 ) then
565:         call KEEP( 'IDCEL', LIDCEL, NCELL, 'I4', LAST, JDEBG )
566:         call KEEP( 'IPCEL', LIPCEL, NCELL+1, 'I4', LAST, JDEBG )
567:         call KEEP( 'IPCLI', LIPCLI, NCELL+1, 'I4', LAST, JDEBG )
568:         call KEEP( 'ICTYP', LICTYP, NCELL, 'I4', LAST, JDEBG )
569:         call KEEP( 'CELSZ', LCELSZ, 6*NCELL, 'R8', LAST, JDEBG )
570:         if ( LSIZ(LAST).gt.LIMIT ) then
571:           call MEMERR( 'GEOMIN', 'LATTICE/CELL DATA.', LSIZ(LAST),
572:             & LIMIT )
573:         end if
574:       end if
575: C
576: C-----
577: C DATA & PROCESS ZONE DATA FOR TRACKING
578: C-----
579: C
580:       call REMAIN( 'KZDA', NLTEMP, 'I4', LAST )
581:       call KEEP( 'KZDA', LKZDA, NLTEMP, 'I4', LAST, JDEBG )
582: C
583:       NWRK = NLTEMP
584: C
585:       call ZONEIN( A, LIMIT, CHA, LIMITC, JDEBG, JLATT, JSIMP, JTLT,

```

```

586:       & JSTGR, JFISX, A(LSDA), A(LIPSDA), A(LKZDA), A(LIBSDA),
587:       & A(LDBODI), A(LIBODI), NBODI, A(LKREG), A(LKMAT), LKINPZ,
588:       & LKZDA, LKZAA, LNKZAA, LKSFBD, NBODY, NINPZ, NZONE, NSURF,
589:       & NSDA, NZDA, LKBZL, LIBZL, NBZL, LAST, NWRK, MAXSF, LINE,
590:       & NCELL, A(LIDCEL), A(LICTYP), A(LIPCEL), A(LIPCLI),
591:       & CHA(LKREGN), NLATT, A(LIDLAT), A(LLTYP), A(LNLVLAT),
592:       & NLTSPC, NSMSPC, CHA(LIZNAM) )
593: C
594: C =====
595: C call HEADER( IOG, "SUBFRAME" & "TALLY REGION" INPUT' )
596: C =====
597: C
598: C
599: C-----
600: C .... COPY TALLY-REGION SPECIFICATION TO UNIT 'IUG2' IF ANY
601: C-----
602: C
603:       if ( JSREG.ne.0 ) then
604:         write(IUG2,'(A)') '#REGION'
605:         180 call CPLIN( IOIN, LINE, NCHAR, IEND )
606: C
607:         if ( IEND.ne.0 ) then
608:           write(IMG,*) 'XXX Unexpected end of data in tally-region',
609:             & ' specification section.'
610: C
611:           write(IMG,*)
612:           & ' Probably missing termination line of the data block.'
613:           call CNTERR( 'FATAL' )
614:           call PRSTOP( 0, 'Tally-region specification error.' )
615:           stop 888
616:         end if
617: C
618:       if ( ISQST2(LINE,'$END','GEOM*').eq.0.and.LINE(1:4).ne.'END ' )
619:         & then
620:           write(IUG2,'(A)') LINE
621:           go to 180
622:         else
623:           if ( LINE(1:4).eq.'END ' ) then
624:             call CNTERR( 'WARNING' )
625:             write(IMG,7220)
626:             7220 format(/1X,'!!! Termination of geometry(tally region)',
627:             & ' input by "END" line will be obsolete.//1X,
628:             & ' Please use "$END GEOMETRY" line.//')
629:           end if
630: C
631:           ... (put '$END' to mark end of data)
632: C
633:           write(IUG2,'(A)') '$END'
634:         end if
635:       end if
636: C
637: C-----
638: C .... CREATE REGION (FRAME/SUBFRAME) NAME TABLE .....>
639: C-----
640: C
641: C
642:       if ( JTLT.ne.0 ) then
643:         call KEVP( A(1), 'KLTSP', LKLTSP, 2*NLTSPC, 'I4', LAST, 0,
644:           & JDEBG )
645:         call KEVP( A(1), 'ILTSP', LILTSP, NLTSPC+1, 'I4', LAST, 0,
646:           & JDEBG )
647:         call KEVP( A(1), 'KCLSP', LKCLSP, NSMSPC, 'I4', LAST, 0, JDEBG
648:           & )
649:         call KEVP( A(1), 'KCLST', LKCLST, NSMSPC, 'I4', LAST, 0, JDEBG
650:           & )

```

src/shared/geomin.f

```

651:
652:     if ( LSIZ(LAST).gt.LIMIT ) then
653:         call MEMERR( 'GEOMIN',
654:             &         'FRAME/SUBFRAME-NAME DATA FOR LATTICES.',
655:             &         LSIZ(LAST), LIMIT )
656:     end if
657: end if
658: CTYP = ' '
659: write(CTYP, '( "C", I3 )') LNAM
660: call CCOMP( CTYP, LEN(CTYP), CTYP, IFL )
661: LLC = ICLEN(CTYP)
662: C
663: C call REMAIN( 'TNAMS', NLTEMP, CTYP(:LLC), LAST )
664: C call REMAINC( 'TNAMS', NLTEMP, CTYP(:LLC), LASTC )
665: C
666: C call KEEP( 'TNAMS', LTNAMS, NLTEMP, CTYP(:LLC), LAST, JDEBG )
667: C call KEEPC( 'TNAMS', LTNAMS, NLTEMP, CTYP(:LLC), LASTC, JDEBG )
668: C
669: C call NAMLST( JLATT, JTLLT, JSREG, NNAMES, CHA(LTNAMS), LTNAMS,
670: & NLTEMP, A(LKLTSP), A(LILTSP), A(LKCLSP), A(LKCLST), NCREG,
671: & NCREGS, LIMIT, CHA(LKREGN), NINPZ, A(LIDLAT), A(LIPLAT),
672: & A(LKLATT), A(LNVLAT), A(LLTYP), NLATT, IUG2, NLTPSP,
673: & NSMSPC, IUG1, LINE )
674: C
675: C
676: C
677: C -----
678: C ..... PREPARE ZONE WISE MAT # & REGION # AND SOME FLAGS .....
679: C -----
680: C
681: C
682: C call KEPV( A(1), 'KREGI', LKREGI, NINPZ, 'I4', LAST, 0, JDEBG )
683: C call KEPV( A(1), 'KZMAT', LKZMAT, NZONE*2, 'I4', LAST, 0, JDEBG )
684: C call KEPV( A(1), 'KZREG', LKZREG, NZONE, 'I4', LAST, 0, JDEBG )
685: C
686: C if ( LSIZ(LAST).gt.LIMIT ) then
687: C     call MEMERR( 'GEOMIN',
688: C         &         'REGION & MATERIAL NUMBER FOR EACH ZONE.', LSIZ(LAST),
689: C         &         LIMIT )
690: C end if
691: C
692: C call MRZONE( A(LKZMAT), A(LKZREG), A(LKMAT), A(LKREG), A(LKINPZ),
693: C & NINPZ, NZONE, CHA(LTNAMS), NNAMES, CHA(LKREGN), A(LKREGI),
694: C & NREG, A(LIDLAT), A(LLTYP), NLATT, JREFL, JLATT, JTLLT,
695: C & JSTGR )
696: C
697: C -----
698: C ..... LATTICE PARAMETER CHECK ETC. ....
699: C -----
700: C
701: C if ( JLATT.ne.0 ) then
702: C
703: C =====
704: C     call HEADER( IOG,
705: C         &         'CHECK & SUMMARY OF STRUCTURED(LATTICE) GEOMETRY' )
706: C     =====
707: C
708: C     call LBZCNT( NLBZ, A(LKZMAT), NZONE )
709: C
710: C     call KEPV( A(1), 'KZLBZ', LKZLBZ, NLBZ, 'I4', LAST, 0, JDEBG )
711: C     call KEPV( A(1), 'MLBZZ', LMLBZZ, NZONE, 'I4', LAST, 0, JDEBG )
712: C
713: C     call ZLBZ( IOG, A(LKZMAT), NZONE, A(LKZLBZ), NLBZ, A(LMLBZZ) )
714: C
715: C     call KEPV( A(1), 'KCELL', LKCELL, NZONE, 'I4', LAST, 0, JDEBG )

```

```

716: C     call KEPV( A(1), 'KCELI', LKCELI, NINPZ, 'I4', LAST, 0, JDEBG )
717: C     call KEPV( A(1), 'KDALT', LKDALT, NLBZ, 'I4', LAST, 0, JDEBG )
718: C     if ( LSIZ(LAST).gt.LIMIT ) then
719: C         call MEMERR( 'GEOMIN', 'LATTICE GEOMETRY DATA.', LSIZ(LAST),
720: C             &         LIMIT )
721: C     end if
722: C
723: C     ..... ONLY TO DETERMINE STARTING POINT OF 'DALT' .....
724: C
725: C     call REMAIN( 'DALT', NLTEMP, 'R8', LAST )
726: C     call KEEP( 'DALT', LDALT, NLTEMP, 'R8', LAST, JDEBG )
727: C -----
728: C ..... CALCULATE TRANSFORMATION PARAMETERS FOR LATTICE GEOMETRIES .....
729: C -----
730: C
731: C     call LATTR( JDEBG, NLATT, NZDA, NZONE, NLBZ, DEPS, A(LIDLAT),
732: C         &         A(LNVLAT), A(LSZLAT), A(LKZDA), A(LKZAA), A(LKSFB),
733: C         &         A(LIBSDA), NBODY, A(LSDA), A(LKZMAT), A(LKMAT),
734: C         &         A(LKINPZ), NINPZ, A(LLTYP), A(LMLBZZ), A(LKDALT),
735: C         &         A(LKLATT), A(LDALT), NLTEMP, JSTGR, LDALT, LAST,
736: C         &         A(LIPLAT), JHLAT, A(LKHLAT), LIMIT )
737: C
738: C     call CELTR( JTLLT, NLATT, NCELL, A(LKCELL), A(LIDCEL),
739: C         &         A(LCELSZ), A(LKCELI), A(LKINPZ), NINPZ, A(LKZMAT),
740: C         &         A(LKZREG), A(LIPCEL), A(LIDLAT), A(LSZLAT), A(LIPLAT),
741: C         &         A(LKLATT), NZDA, NZONE, A(LKZDA), A(LKZAA), A(LSDA),
742: C         &         A(LICTYP), A(LLTYP), DEPS )
743: C
744: C -----
745: C ..... SEARCH THE MAXIMUM NESTING-LEVEL OF LATTICE GEOMETRY. ....
746: C -----
747: C
748: C     call NESTCK( NEST, NLATT, NCELL, NINPZ, A(LKCELI), A(LIDCEL),
749: C         &         A(LIPCLI), A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT),
750: C         &         A(LLTYP), A(LKMAT) )
751: C     end if
752: C
753: C -----
754: C ..... PROCESSING FOR FRAME-DEPENDENT-TALLY (TALLY-LATTICE) OPTION .....
755: C -----
756: C
757: C     if ( JTLLT.ne.0 ) then
758: C         call KEPV( A(1), 'KSUZN', LKSUZN, NINPZ*2, 'I4', LAST, 0, JDEBG
759: C         &
760: C         call KEPV( A(1), 'KUNV', LKUNV, NINPZ, 'I4', LAST, 0, JDEBG )
761: C         call KEPV( A(1), 'KZUNV', LKZUNV, NZONE, 'I4', LAST, 0, JDEBG )
762: C
763: C         (... ONLY TO SET STARTING ADDRESS ...)
764: C
765: C         call KEEP( 'ISPNM', LISPNM, 0, 'I4', LAST, JDEBG )
766: C
767: C         if ( LSIZ(LAST).gt.LIMIT ) then
768: C             call MEMERR( 'GEOMIN', 'FRAME-STRUCTURE DATA.', LSIZ(LAST),
769: C                 &         LIMIT )
770: C         end if
771: C
772: C         (... argument IOG removed on Jan 2000)
773: C
774: C         call SPNUMS( NSPACE, LAST, A(LISPNM), A(LKREG), A(LKUNV), NUNV,
775: C             &         NSUZON, A(LKSUZN), A(LKZUNV), NNAMES, CHA(LTNAMS),
776: C             &         NLATT, NCELL, NEST, NINPZ, NZONE, A(LIDLAT), A(LKLTSP),
777: C             &         A(LILTSP), A(LKCLSP), A(LKCLST), LIMIT, CHA(LKREGN),
778: C             &         A(LKREGI), NLTPSP, NSMSPC, A(LKMAT), A(LKCELI),
779: C             &         A(LLTYP), A(LKLATT), A(LIPLAT), A(LIPCLI), A(LKINPZ) )
780: C

```

src/shared/geomin.f

```

781:      call RESIZE( 'ISPNM ', LISPNM, 2*(NEST+1)*NSPACE, 'I4', LAST )
782: C
783: C      .... FRAME, REGION TABLES FOR NESTED REGION TALLY ....
784: C
785: C      call KEPV( A(1), 'ISUSP', LISUSP, NSUZON*NSPACE, 'I4', LAST, 0,
786: &              JDEBG )
787: &
788: &      call KEPV( A(1), 'KSPSU', LKSPSU, NUNV*(NSPACE+1), 'I4', LAST,
789: &              0, JDEBG )
790: C
791: C      if ( LSIZ(LAST).gt.LIMIT ) then
792: C      call MEMERR( 'GEOMIN', 'FRAME-STRUCTURE DATA.', LSIZ(LAST),
793: &              LIMIT )
794: C      call PRSTOP( 1, 'MEMORY OVER.' )
795: C      stop 999
796: C      end if
797: C      call REMAIN( 'KTCSP', NLTEMP, 'I4', LAST )
798: C      call KEEP( 'KTCSP', LKTCSP, NLTEMP, 'I4', LAST, JDEBG )
799: C
800: C      (... argument IOG removed Jan 2000)
801: C      call SPREG( NREG, A(LKTCSP), NLTEMP, NKTCSP, A(LKSPSU),
802: &              A(LISUSP), A(LKREG), A(LKZREG), NUNV, NSUZON, NSPACE,
803: &              NNAME, CHA(LTNAMS), NZONE, NLATT, NCELL, NEST,
804: &              A(LKLTSP), A(LILTSP), A(LKCLSP), A(LKCLST), A(LISPNM),
805: &              A(LKUNV), LIMIT, CHA(LKREGN), A(LKREGI), NINPZ, NITSPC,
806: &              NSMSPC, A(LKMAT), A(LKCELI), A(LKINPZ), A(LLTYP),
807: &              A(LKCLATT), A(LIPLAT), A(LIPCLI), A(LIDLAT), A(LIDCEL) )
808: C
809: C      call RESIZE( 'KTCSP', LKTCSP, NKTCSP, 'I4', LAST )
810: C
811: C
812: C      .... FRAME # & NAME # TABLE FOR EACH REGION ...
813: C
814: C
815: C      call KEPV( A(1), 'IRGSP', LIRGSP, 2*NREG, 'I4', LAST, 0, JDEBG
816: &              )
817: C      if ( LSIZ(LAST).gt.LIMIT ) then
818: C      call MEMERR( 'GEOMIN', 'FRAME-STRUCTURE DATA.', LSIZ(LAST),
819: &              LIMIT )
820: C      call PRSTOP( 1, 'MEMORY OVER.' )
821: C      stop 999
822: C      end if
823: C
824: C      call ITRGSP( IOG, A(LIRGSP), A(LISUSP), NREG, NSUZON, NSPACE,
825: &              A(LKSUZON), A(LKREGI), NINPZ )
826: C      end if
827: C
828: C-----
829: C      PROCESS RE-DEFINED REGION ( NCREG REGIONS )
830: C-----
831: C
832: C      call KEEP( 'IPTRG', LIPTRG, NCREG+NREG+1, 'I4', LAST, JDEBG )
833: C      call KEEP( 'ITRNM', LITRNM, NCREG+NREG, 'I4', LAST, JDEBG )
834: C      call REMAIN( 'LPTRG', NLTEMP, 'I4', LAST )
835: C      call KEEP( 'LPTRG', LLPTRG, NLTEMP, 'I4', LAST, JDEBG )
836: C
837: C      DETERMINE TALLY-REGIONS FROM '#REGION' DATA STORED IN FILE
838: C      'IUG2'.
839: C
840: C      (... argument IOG removed Jan 2000)
841: C
842: C      call TLOREG( JTLLT, NTREG, A(LIPTRG), A(LLPTRG), NLTEMP, LLPTRG,
843: &              A(LITRNM), LAST, IUG2, LINE, NCREG, NREG, NINPZ, NSPACE,
844: &              NSUZON, NZONE, NEST, A(LISPNM), A(LISUSP), A(LKMAT),
845: &              A(LKREG), A(LKREGI), A(LKCELI), A(LKSUZON), A(LIRGSP),
846: &              CHA(LTNAMS), NNAME, LIMIT )
847: C
848: C      call RESIZE( 'LPTRG', LLPTRG, LLPTRG, 'I4', LAST )
849: C      call RIUNIT( IOIN )
850: C
851: C      return
852: C      end

```

src/shared/grconv.f

```

1:      subroutine GRCONV( GR, PX,    PY,    PZ,    AX,    AY,    AZ,
2:      &                  BX,    BY,    BZ,    DINF )
3: C=====
4: C Purpose: translate geometry XYZ range (GR(6)) defined in local
5: C   coordinate (origin(PX,PY,PZ),X-axis(AX,AY,AZ),Y-axis(BX,BY,BZ))
6: C   to range in global coordinates.
7: C
8: C   (PX,PY,PZ),(AX,AY,AZ) and (BX,BY,BZ) remain unchanged.
9: C
10: C-----
11: C   if (BX,BY,BZ)=0
12: C   (BX,BY,BZ) is generated automatically.
13: C-----
14: C arguments (i=input, o=output, w=work)
15: C io GR(6) : geometry range
16: C i  PX,PY,PZ,AX,AY,AZ
17: C i  DINF : large distance
18: C-----
19:      implicit real*8(D)
20:      real*8 GR(6), PX, PY, PZ, AX, AY, AZ, BX, BY, BZ
21:      real*8 DINF
22: C
23: C-----
24: C
25:      DD      = SQRT(AX*AX+AY*AY+AZ*AZ)
26:      if ( DD.eq.0.0D0 ) return
27:      DAX      = AX/DD
28:      DAY      = AY/DD
29:      DAZ      = AZ/DD
30:
31:      DD      = SQRT(BX*BX+BY*BY+BZ*BZ)
32: C
33:      if ( DD.eq.0.0 ) then
34:      if ( DAX.eq.1.0D0 ) then
35:          DBX = 0.0
36:          DBY = 1.0
37:          DBZ = 0.0
38:      else if ( DAY.eq.1.0D0 ) then
39:          DBX = 0.0
40:          DBY = 0.0
41:          DBZ = 1.0
42:      else if ( DAZ.eq.1.0D0 ) then
43:          DBX = 1.0
44:          DBY = 0.0
45:          DBZ = 0.0
46:      else if ( abs(DAZ).lt.0.9 ) then
47:          DBX = -DAY
48:          DBY = DAX
49:          DBZ = 0.0d0
50:      else
51:          DBX = 0.0d0
52:          DBY = 1.0d0
53:          DBZ = 0.0d0
54:      end if
55:      else
56:          DBX = BX
57:          DBY = BY
58:          DBZ = BZ
59:      end if
60:      DD      = SQRT(DBX*DBX+DBY*DBY+DBZ*DBZ)
61:      if ( DD.eq.0.0D0 ) return
62:      DBX      = DBX/DD
63:      DBY      = DBY/DD
64:      DBZ      = DBZ/DD
65: C
66:      DAB      = DAX*DBX + DAY*DBY + DAZ*DBZ
67:      DBX      = DBX - DAB*DAX
68:      DBY      = DBY - DAB*DAY
69:      DBZ      = DBZ - DAB*DAZ
70:      DD      = SQRT(DBX*DBX+DBY*DBY+DBZ*DBZ)
71:      if ( DD.eq.0.0D0 ) return
72:      DBX      = DBX/DD
73:      DBY      = DBY/DD
74:      DBZ      = DBZ/DD
75: C
76:      DCX      = DAY*DBZ - DAZ*DBY
77:      DCY      = DAZ*DBX - DAX*DBZ
78:      DCZ      = DAX*DBY - DAY*DBX
79:      DD      = SQRT(DCX*DCX+DCY*DCY+DCZ*DCZ)
80:      if ( DD.eq.0.0D0 ) return
81:      DCX      = DCX/DD
82:      DCY      = DCY/DD
83:      DCZ      = DCZ/DD
84: C
85:      DG1      = DINF
86:      DG2      = -DINF
87:      DG3      = DINF
88:      DG4      = -DINF
89:      DG5      = DINF
90:      DG6      = -DINF
91: C
92:      do 100 N = 1, 8
93:          I      = MOD(N,2)
94:          J      = MOD(N/2,2)
95:          K      = MOD(N/4,2)
96:          DX      = GR(1+I)
97:          DY      = GR(3+J)
98:          DZ      = GR(5+K)
99: C
100:          DGX     = DX*DAX + DY*DBX + DZ*DCX + PX
101:          DGY     = DX*DAY + DY*DBY + DZ*DCY + PY
102:          DGZ     = DX*DAZ + DY*DBZ + DZ*DCZ + PZ
103: C
104:          DG1     = MIN(DG1,DGX)
105:          DG2     = MAX(DG2,DGX)
106:          DG3     = MIN(DG3,DGY)
107:          DG4     = MAX(DG4,DGY)
108:          DG5     = MIN(DG5,DGZ)
109:          DG6     = MAX(DG6,DGZ)
110:      100 continue
111: C
112:      GR(1)     = DG1
113:      GR(2)     = DG2
114:      GR(3)     = DG3
115:      GR(4)     = DG4
116:      GR(5)     = DG5
117:      GR(6)     = DG6
118: C
119:      return
120:      end

```


src/shared/gtsour.f

```

1:      subroutine GTSOUR( SOUR,  IDSRC, SRCSP, NSOUR )
2: C=====
3: C purpose : gather source generation ratio and source set ID
4: C   for all source sets and normalize for source set selection.
5: C called in : srcinp
6: C calls: pctrw cnterr label r8prnt unpknd pctlb
7: C-----
8: C arguments (i=input, o=output, w=work)
9: C o SOUR(NSOUR) : source set fraction
10: C o IDSRC(NSOUR) : source set ID
11: C o SRCSP(*) : source data
12: C*i IPR : message printout I/O unit (removed Jan 2000)
13: C-----
14:      real*8 SOUR(NSOUR)
15:      integer IDSRC(NSOUR)
16:      integer SRCSP(*)
17: C
18:      character*32 LBL
19: C
20:      real*8 SUM
21:      integer IDUM(4)
22: C
23:      include 'INC/_IUNIT'
24: C
25: C-----
26: C
27:      call PCTRW( SRCSP, 'GTSOUR1', IRET )
28: C
29:      do 100 N = 1, NSOUR
30:         write(LBL,('( '&'',I5,' 'SPEC''))' ) N
31:         call PCTLB( SRCSP, 'SPEC0', LBL(:10), IRET1 )
32: C
33:         call UNPKND( SRCSP, 'SPEC1-2', 'I4', IDUM, 2, ND, IRET2 )
34: Ccccccc call UNPKND( SRCSP, 'RATIO', 'R4', SOUR(N), 1, ND, IRET3 )
35:         call UNPKND( SRCSP, 'RATIO', 'R8', SOUR(N), 1, ND, IRET3 )
36:         call UNPKND( SRCSP, 'ID', 'I4', IDSRC(N), 1, ND, IRET4 )
37: C
38:         if ( ABS(IRET1)+ABS(IRET2)+ABS(IRET3)+ABS(IRET4).ne.0 ) then
39:            write(IMG,*) 'XXX CANNOT RETRIEVE SOURCE GENERATION',
40:            &          'RATIO FROM SOURCE INFORMATION XXX'
41:            call CNTERR( 'FATAL' )
42:         end if
43:      100 continue
44: C
45:         SUM      = 0.0D0
46:         do 110 N = 1, NSOUR
47:            SUM      = SUM + ABS(SOUR(N))
48:         110 continue
49: C
50:         if ( NSOUR.eq.1.and.SUM.eq.0.0 ) then
51:            SOUR(1) = 1.0D0
52:         else
53:            do 120 N = 1, NSOUR
54:               SOUR(N) = SOUR(N) /SUM
55:            120 continue
56:         end if
57: C
58:         call LABEL( IPR, 'SOURCE FRACTION (NORMALIZED)' )
59:         call R8PRNT( 'RATIO ', SOUR, NSOUR, 1, 'SOURCE #', ' ', IPR )
60: C
61:         do 140 N = 2, NSOUR
62:            if ( IDSRC(N).ne.0 ) then
63:               do 130 NN = 1, N - 1
64:                  if ( IDSRC(N).eq.IDSRC(NN) ) then
65:                     write(IMG,7000) N, NN, IDSRC(N)

```

```

66:            call CNTERR( 'WARNING' )
67:            end if
68:         130 continue
69:         end if
70:      140 continue
71:      7000 format(2X,'!!! ID of ',I4,' 'th source set is identical to ',
72:      &          'that of ',I4,' 'th one :',I8)
73: C
74:      return
75:      end

```

src/shared/hextr0.f

```

1:      subroutine HEXTR0( SZLAT, LTYP,  IRC )
2: C=====
3: C  PURPOSE :
4: C
5: C      TRANSFORMATION OF 'SZLAT' (LATTICE PARAMETER) TO MATCH THE
6: C      REQUIREMENT OF 'LATTR' ROUTINE.
7: C
8: C      A REVISION IN THE WAY OF INPUT FOR HEXAGONAL LATTICE ON NOVEMBER
9: C      1990 MADE THIS SUBROUTINE NECESSARY.
10: C      ( MADE BY  M.SASAKI )
11: C
12: C  CALLED IN : LATTIN
13: C
14: C  CALLS : (NONE)
15: C
16: C-----
17: C arguments (i=input, o=output, w=work)
18: C
19: C i o  SZLAT(8) : lattice parameter of a lattice.
20: C i   LTYP      : lattice type of a lattice.
21: C i   IRC(2)   : reference cell position index (RCCELL(*) input)
22: C-----
23: C
24: C   < REVISED MEANING OF INPUT SZLAT >
25: C
26: C   .... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
27: C
28: C   (REVISED FOR HEXAGONAL LATTICE (NOVEMBER 1990))
29: C
30: C SZLAT *      TYPE 1      *      TYPE 2      *      TYPE 3      *      TYPE 4
31: C-----*-----*-----*-----*-----*-----*-----
32: C  1 * CELL-SIZE(X)*                CELL PITCH
33: C  2 *      (Y) *      ANGLE OF CELL ARRAY (DEGREE) - 30 DEGREE
34: C  3 *      (Z) *      CELL HEIGHT
35: C-----*-----*-----*-----*-----*-----
36: C
37: C (SZHEX)
38: C  4 *      ... *      DEVIATION OF REFERENCE CELL CENTER
39: C      *      *      FROM THE CENTER OF THE LATTICE FRAME.
40: C  5 *      ... *      (VECTOR IN THE FRAME-COORDINATE-SYSTEM)
41: C      *      *
42: C-----*-----*-----*-----*-----*-----
43: C      (LATTICE FRAME SIZE)
44: C-----*-----*-----*-----*-----*-----
45: C  6 *      ... * LATTICE- * CYLINDER- * WIDTH (X)
46: C      *      *      WIDTH * RADIUS *
47: C  7 *      ... * HEIGHT * HEIGHT * WIDTH (Y)
48: C  8 *      ... *      ... *      ... * WIDTH (Z)
49: C-----*-----*-----*-----*-----*-----
50: C
51: C * FOR USE IN 'LATTR' ROUTINE,
52: C (SZLAT(4), SZLAT(5)) MUST BE THE ORIGIN OF THE CELL-ARRAY
53: C IN THE FRAME-ATTACHED COORDINATE.
54: C
55: C
56: C      implicit real*8(D)
57: C      real*8 SZLAT(8)
58: C      integer IRC(2)
59: C
60: C-----
61: C
62: C      if ( LTYP.le.1 ) return
63: C
64: C      SZLAT(2) = SZLAT(2) - 30.0D0
65: C      DT      = SZLAT(2) /180.0D0*3.141592653589793238D0

```

```

66:      DS      = SIN(DT)
67:      DC      = COS(DT)
68: C
69: C      .... POSITION OF REFERENCE CELL IN CELL-COORDINATE ....
70: C
71: C      DX      = (dble(IRC(1)-1)+dble(IRC(2)-1)*0.5d0)*SZLAT(1)
72: C      DY      = dble(IRC(2)-1)*SQRT(3.0D0) /2.0d0*SZLAT(1)
73: C
74: C      .... ANTI-ROTATION TO FRAME-COORDINATE ....
75: C
76: C      DXX     = DX*DC - DY*DS
77: C      DYY     = DX*DS + DY*DC
78: C
79: C      .... HEXAGONAL FRAME .....
80: C
81: C      if ( LTYP.eq.2 ) then
82: C          SZLAT(4) = -DXX + SZLAT(4) + SZLAT(6)*0.5d0
83: C          CCCCCCCC SZLAT(5) = -DYY + SZLAT(5) + SZLAT(6)*0.5/SQRT(3.0)
84: C          SZLAT(5) = -DYY + SZLAT(5) + SZLAT(6)*0.5d0/SQRT(3.0D0)
85: C
86: C      .... CYLINDRICAL FRAME .....
87: C
88: C      else if ( LTYP.eq.3 ) then
89: C          SZLAT(4) = -DXX + SZLAT(4)
90: C          SZLAT(5) = -DYY + SZLAT(5)
91: C
92: C      .... RECTANGULAR CELL .....
93: C
94: C      else if ( LTYP.eq.4 ) then
95: C          SZLAT(4) = -DXX + SZLAT(4) + SZLAT(6)*0.5d0
96: C          SZLAT(5) = -DYY + SZLAT(5) + SZLAT(7)*0.5d0
97: C      end if
98: C
99: C      return
100: C      end

```

src/shared/i4prnt.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine I4PRNT( VNAME, I4,    N0,    N1,    N2,    J,
3:     &                HD1, HD2, IPRT)
4: C
5: C   JAERI MONTE CARLO CODE UTILITY
6: C
7: C=====
8: C PURPOSE: Print out two dimensional integer array
9: C CALLED IN: various place
10: C CALLS: (NONE)
11: C-----
12: C arguments (i=input, o=output, w=work)
13: C i VNAME : data name string if any
14: C i I4(N0,N1,N2) : integer array to be printed
15: C i N0, N1, N2 : dimension size ( why three dimensions ? N0 is equal to
16: C               1 to print contiguous data elements. )
17: C i J : offset for the first dimension (=1 for most case)
18: C i HD1, HD2 : label for 1st and 2nd dimension.
19: C i IPRT : printout I/O unit.
20: C=====
21:   integer I4(N0,N1,N2)
22:   character VNAME*(*), HD1*(*), HD2*(*)
23: C ... local ...
24:   character HEAD1*8, HEAD2*8
25: C
26:   HEAD1 = HD1
27:   HEAD2 = HD2
28: C
29:   if ( N1*N2.eq.0 ) return
30: C
31:   write(IPRT,7000) VNAME, N1, N2
32:   7000 format(/1X,'<<< ARRAY ',A,'(I,J) (I=1,',I4,',J=1,',I4,',') >>>')
33: C
34:   ML      = 12
35:   do 130 K = 1, N1, ML
36:     K2     = MIN(K+ML-1,N1)
37:
38:     write(IPRT,7020) ('-----',KK=K,K2)
39:     7020 format(1X,'-----',12A9)
40:
41:     write(IPRT,7040) HEAD2, HEAD1, (KK,KK=K,K2)
42:     7040 format(1X,A8,'| ',A8/1X,8X,'| ',12(I8,1X))
43:     write(IPRT,7060) ('-----',KK=K,K2)
44:     7060 format(1X,'-----+',12A9)
45:     write(IPRT,7100) 1, (I4(J,KK,1),KK=K,K2)
46:     JJ      = 1
47:     do 120 IJ = 2, N2
48:       do 100 KK = K, K2
49:         100 if ( I4(J,KK,IJ).ne.I4(J,KK,JJ) ) go to 110
50:         go to 120
51:       110 if ( IJ.gt.JJ+2 ) write(IPRT,7080) HEAD2, JJ + 1, IJ - 1,
52:         &      HEAD2, JJ
53:       if ( IJ.eq.JJ+2 ) write(IPRT,7100) IJ - 1,
54:         &      (I4(J,KK,IJ-1),KK=K,K2)
55:       write(IPRT,7100) IJ, (I4(J,KK,IJ),KK=K,K2)
56:       JJ      = IJ
57:     120 continue
58:     if ( JJ.lt.N2-1 ) write(IPRT,7080) HEAD2, JJ + 1, N2, HEAD2, JJ
59:     if ( JJ.eq.N2-1 ) write(IPRT,7100) N2, (I4(J,KK,N2),KK=K,K2)
60:     write(IPRT,7060) ('-----',KK=K,K2)
61:   130 continue
62:   7080 format(1X,8X,'| ',A8,I5,' TO ',I5,' ARE SAME AS ',A8,I5,' ) ')
63:   7100 format(1X,I6,2X,'| ',12(I8,1X))
64:   return
65: end
```

src/shared/ibtile.f

```

1:      subroutine IBTILE( IBIN, NBIN, N,      IBLIST,NBLIST )
2:      C=<MVP/GMVP>=====
3:      C purpose: construct a domain "tiling" table, from arbitrary number of
4:      C      integer list.
5:      C
6:      C ex.
7:      C      iblist-1 : - 1 1 1 - 1 - - - -
8:      C      iblist-2 : - - 2 - 2 2 2 - -
9:      C      iblist-3 : - - - - - 3 3 - 3 -
10:     C      iblist-4 : - - - - - - 4 - - 4
11:     C
12:     C      ibin      : 0 1 3 1 2 5 6 2 4 7
13:     C
14:     C      ibin is tiled in 7 parts by combination of four iblist's.
15:     C      Each tiling domain may not be contiguous.
16:     C
17:     C      Used to make direct-tally bin table from edited-tally bin
18:     C      configuration.
19:     C
20:     C      This routine must be called more than one times such as n=1,2,3,...
21:     C
22:     C called in: TALIN1
23:     C -----
24:     C arguments (i=input, o=output, w=work)
25:     C
26:     C i o IBIN(NBIN) : constructed table
27:     C i NBIN      : length of table
28:     C i N          : in n'th construction step.
29:     C i IBLIST(NBLIST) : input to n'th step.
30:     C              IBLIST(*) shows position in ibin(*).
31:     C i NBLIST     : size of iblist.
32:     C
33:     C=====
34:     integer IBIN(NBIN)
35:     integer IBLIST(NBLIST)
36:     C
37:     include 'INC/_IOUNIT'
38:     C
39:     data MMAX /0/
40:     C-----
41:     if ( N.eq.1 ) then
42:       do 100 I = 1, NBIN
43:         IBIN(I) = 0
44:       100 continue
45:       MMAX = 0
46:     end if
47:     C
48:     do 110 J = 1, NBLIST
49:       if ( IBLIST(J).lt.1 .or. IBLIST(J).gt.NBIN ) then
50:         IER = IBLIST(J)
51:         go to 130
52:       end if
53:       IBIN(IBLIST(J)) = IBIN(IBLIST(J)) + MMAX + 1
54:     110 continue
55:     C
56:     M0 = 0
57:     do 120 I = 1, NBIN
58:       M0 = MAX(M0,IBIN(I))
59:     120 continue
60:     MMAX = M0
61:     C
62:     C ... adjust values if mmax is to large ...
63:     if ( MMAX.gt.2**15 ) then
64:       call IORDER( IBIN, NBIN, MMAX )
65:     end if

```

```

66:
67:       return
68:     C
69:     130 write(IMG,*) 'XXX(IBTILE) Position index is out of range.',
70:       &      ' (should be 0 to ',nbin,', but value is ',IER,')'
71:       write(IMG,*) ' IBLIST ',(IBLIST(J),J=1,NBLIST)
72:       stop 666
73:     C
74:       end

```

src/shared/iioset.f

```
1:      subroutine IIOSET
2: C=====
3: C Purpose: set initial value of I/O units on common area
4: C
5: C   I/O units in MVP/GMVP (and in CGVIEW) are declared as parameter
6: C   constants or as common variables. (see src/shared/INC/_IOUNIT)
7: C
8: C   Default value of I/O units should be set in this routine in
9: C   startup phase of program
10: C   And some of them may change its value depending on codes or
11: C   execution mode (parallel processing etc.).
12: C
13: C -----
14: C I/O units preset in this routine.
15: C
16: C   INP  : standard input (normally, unit 5)
17: C   IOIN : text data input. Text form input is normally copied from
18: C         standard input (unit 5) on this unit (default 55).
19: C         In CGVIEW code, this unit is reset to standard input itself
20: C         afterwards (=5).
21: C
22: C   IPR,IOG,IOW,IOT,IOF,IMSG : printout units. Normally they are all 6
23: C
24: C   IPR  : standard output (printout)
25: C   IMSG : message output (printout)
26: C   IOG  : printout unit in geometry processing phase
27: C   IOW  : printout unit in random walk phase
28: C   IOT  : printout unit in tally data output phase (other than flux)
29: C   IOF  : printout unit of calculated flux
30: C
31: C=====
32: C
33: C   include 'INC/_IOUNIT'
34: C
35: C -----
36: C
37: C   IOIN    = 55
38: C
39: C /#IF PARA(VPP)
40: C
41: C   === IOW may have task-wise value in VPP mode
42: C
43: C   INP=5
44: C   IPR=6
45: C   IOSTDO = 6
46: C   IMSG=6
47: C   IOW=6
48: C   IOG=6
49: C   IOT=6
50: C   IOF=6
51: C
52: C /#ELSEIF SYSTEM(PARAGON)
53: C
54: C   === Do not printout directly on stdout on PARAGON
55: C
56: C   INP=5
57: C   IPR =7
58: C   IOSTDO = 7
59: C   IMSG =7
60: C   IOW =6
61: C   IOG =7
62: C   IOT =7
63: C   IOF =7
64: C
65: C /#ELSEIF SYSTEM(FACOMAP3K)
```

```
66: C
67: C === The aprun command on MPI/AP does not guarantee the standard
68: C   input/output in the outside between MPI_INIT and MPI_finalize.
69: C   A MVP input file is read from unit 3 instead of unit 5
70: C
71: C   INP    = 3
72: C   IPR    = 6
73: C   IOSTDO = 6
74: C   IMSG   = 6
75: C   IOW    = 6
76: C   IOG    = 6
77: C   IOT    = 6
78: C   IOF    = 6
79: C
80: C /#ELSE
81: C
82: C   INP    = 5
83: C   IPR    = 6
84: C   IOSTDO = 6
85: C   IMSG   = 6
86: C   IOW    = 6
87: C   IOG    = 6
88: C   IOT    = 6
89: C   IOF    = 6
90: C
91: C /#ENDIF
92: C
93: C   return
94: C   end
```

src/shared/inplst.f

```

1:      subroutine INPLST( IPR, MSG,   INP,   IOIN, NTASK0,MLIMIT )
2: C=<MVP/GMVP UTILITY>=====
3: C PURPOSE: Input file listing & copy input data to 'IOIN'.
4: C CALLED IN: INTRO
5: C-----
6: C arguments (i=input, o=output, w=work)
7: C i IPR : printout I/O unit
8: C i MSG : error/warning message printout I/O unit
9: C i INP : input data stream from this I/O unit
10: C i IOIN : copy input data stream on this I/O unit
11: C io NTASK0 : number of tasks in multi task mode
12: C io MLIMIT : default value of dynamic memory size
13: C-----
14: C
15: C ... linpl : length of line input buffer
16: C ... leff : length of effective parts in input lines
17: C
18: C      parameter( LINPL   = 80, LEFF  = 72 )
19: C
20: C      character LINE*(LINPL)
21: C      character DTYP*16
22: C      logical OPD
23: C
24: C      character*16 CDUMMY, CWRK
25: C
26: C-----
27: C
28: C      NL      = 0
29: C      call HEADER( IPR, 'INPUT DATA LISTING' )
30: C
31: C
32: C      write(IPR,7000) (I,I=1,LEN(LINE)/10)
33: C      7000 format(/1X,5X,' 0',8('....+....',I1),4X,'DATA TYPE'/1X,'LINE#')
34: C
35: C
36: C-----
37: C      STATUS : ISTAT
38: C      ISTAT = 1 : TITLE
39: C      = 2 : OPTIONS
40: C      = 3 : OTHERS
41: C
42: C      echo output status : IECHO (0/1=do not echo/echo)
43: C-----
44: C
45: C
46: C      ISTAT  = 1
47: C      NEXTRA = 0
48: C      IECHO  = 1
49: C      IECHO0 = 1
50: C
51: C      100 read(INP,'(A)',end =120) LINE
52: C
53: C      if ( LINE(1:6).eq.'*@ECHO' ) then
54: C          call CCOMP(CWRK,len(CWRK),LINE(8:),IFL )
55: C          if ( CWRK(1:3).eq.'OFF'.or.CWRK(1:2).eq.'NO' ) then
56: C              IECHO = 0
57: C          else if ( CWRK(1:2).eq.'ON'.or.CWRK(1:3).eq.'YES' ) then
58: C              IECHO = 1
59: C          end if
60: C      end if
61: C
62: C      ... line block @PREINP --- @END [PREINP] is lines passed to
63: C      'PREINP' routine which translate extra options.
64: C
65: C      if ( ISTAT.eq.1.and.LINE(1:7).eq.'@PREINP' ) then

```

```

66:      write(IPR,
67:      &      '(/1X,' '!!! @PREINP block is placed before input.' '))'
68: C
69: C      IPREIN = 1
70: C
71: C      110 read(INP,'(A)',end =120) LINE
72: C      if ( LINE(1:4).eq.'@END' ) then
73: C          write(IPR,'(/1X,' ' @PREINP Block ended.' '))'
74: C          go to 100
75: C      else
76: C
77: C      ... call preinp routine.
78: C
79: C      Attention: MVP and GMVP have their own 'PREINP' routine.
80: C
81: C          call PREINP( IPREIN, LINE, NTASK0, CDUMMY, MLIMIT )
82: C          go to 110
83: C      end if
84: C  end if
85: C
86: C
87: C      NL      = NL + 1
88: C
89: C      if ( NL.eq.1.and.INP.ne.IOIN ) then
90: C          inquire(IOIN,opened =OPD)
91: C          if ( .not.OPD ) open( IOIN, status = 'UNKNOWN', form =
92: C      &      'FORMATTED' )
93: C      end if
94: C
95: C      IEXTRA = 0
96: C
97: C      if ( NL.le.2 ) then
98: C          DTYP = 'TITLE'
99: C      else
100: C          if ( LINE(1:1).eq.'*' ) then
101: C              DTYP = 'COMMENT'
102: C          else
103: C              if ( LINE(LEFF+1:).ne.' ' ) then
104: C                  IEXTRA = 1
105: C                  NEXTRA = NEXTRA + 1
106: C              end if
107: C              if ( LINE(1:1).eq.'%' ) then
108: C                  DTYP = 'PARAMETER LINE'
109: C              else
110: C                  if ( ISTAT.eq.1 ) ISTAT = 2
111: C                  if ( ISTAT.eq.2 ) then
112: C                      if ( LINE(1:LEFF).ne.' ' ) then
113: C                          DTYP = 'OPTION'
114: C                      else
115: C                          DTYP = ' '
116: C                          ISTAT = 3
117: C                      end if
118: C                  else if ( ISTAT.eq.3 ) then
119: C                      DTYP = 'DATA'
120: C                  end if
121: C              end if
122: C          end if
123: C      end if
124: C
125: C      == Echo lines ( but skipped if IECHO is zero )
126: C
127: C      if ( IECHO0.eq.1.or.IECHO.eq.1 ) then
128: C          if ( IEXTRA.eq.0 ) then
129: C              write(IPR,'(1X,I5,' ' : ',A,' ' : ',A)') NL, LINE, DTYP
130: C          else

```

src/shared/inplst.f

```
131:         write(IPR,'(1X,I5,'':>'',A,'': ''',A)') NL, LINE, DTYP
132:         end if
133:     end if
134: C
135:     if ( IECHO0.eq.1.and.IECHO.eq.0 ) then
136:         write(IPR,'(/1X,5x,A)') '+++ listing is skipped +++'
137:     end if
138:     IECHO0 = IECHO
139: C
140:     if ( INP.ne.IOIN ) then
141:         write(IOIN,'(A)') LINE(1:LEFF)
142:     end if
143: C
144:     go to 100
145: C
146: C
147: 120 write(IPR,7020) (1,I=1,LEN(LINE)/10), NL
148: 7020 format(/1X,' ',0',8('....+....',I1))// ' TOTAL LINES : ',18/)
149: C
150:     if ( NEXTRA.gt.0 ) then
151:         write(IMG,'(1x,a,i3,a/1x,a)')
152:         & ' !!! Some input lines have extra characters after ',
153:         & LEFF, ' 'th column.',
154:         & ' Such lines are marked as "line#:>".'
155:         call CNTERR( 'WARNING' )
156:     end if
157: C
158:     rewind IOIN
159: C
160:     return
161: end
```

src/shared/intplc.f

```
1:      subroutine INTPLC( SYM,  INTP,  IDIFF )
2: c=====
3: c  purpose: Translate interpolation symbol to interpolation code or
4: c           vice versa.
5: c
6: c           Translate interpolation symbol 'sym' to interplation code
7: c           'intp' if 'sym' is not blank.
8: c           When 'sym' is blank, this routine returns interpolation symbol
9: c           that corresponds to 'intp' in 'sym'.
10: c           For both of the case above, valid difference between number of
11: c           data points & that of probability values is returned in 'idiff'.
12: c
13: c  called from: smpinf
14: c  history: programmed by M.Sasaki (June 1993)
15: c-----
16: c  arguments (i=input, o=output, w=work)
17: c
18: c  i SYM: interpolation symbol
19: c  o INTP: interpolation code
20: c  o IDIFF: difference between number of data points & number of
21: c           probability values given
22: c=====
23:      character*(*) SYM
24:      integer INTP
25: c
26: c .... interpolation symbols .....
27: c
28:      parameter( NINTP      = 5 )
29:      character*12 INTSYM(NINTP)
30: c
31: c .... valid difference between number of data point &
32: c           that of probability ...
33: c
34:      integer NDIFF(NINTP)
35: c
36:      data(INTSYM(I),NDIFF(I),I=1,NINTP) /
37: &      'STEP      ', 1,
38: &      'DISCRETE   ', 0,
39: &      'LINEAR     ', 0,
40: &      'LOG-LINEAR ', 0,
41: &      'LOG-STEP   ', 1/
42: c
43:      if ( SYM.ne.' ' ) then
44:        do 100 INTP = 1, NINTP
45:          if ( SYM.eq.INTSYM(INTP) ) then
46:            IDIFF = NDIFF(INTP)
47:            return
48:          end if
49: 100    continue
50:        INTP = -1
51:
52:      else
53:        if ( INTP.lt.1 .or. INTP.gt.NINTP ) return
54:        SYM = INTSYM(INTP)
55:        IDIFF = NDIFF(INTP)
56:        return
57:      end if
58:      return
59:      end
```


src/shared/iorder.f

```
1:      subroutine IORDER( IT,      N,      M )
2: C=<MVP/GMVP>=====
3: C purpose: convert list of non-negative integer (IT(1:N)) into ordered
4: C      list.
5: C called in: anywhere
6: C-----
7: C arguments (i=input, o=output)
8: C
9: C io IT(1:N) : array of non-negative integer data.
10: C i N      : size of array IT.
11: C o M      : maximum of ordered number (number of different non-zero
12: C            elements in the original IT array )
13: C-----
14: C
15: C Ex.      IT(1:11) = 0 2 5 0 12 0 9 2 2 5 10
16: C
17: C          --> 0 1 2 0 5 0 3 1 1 2 4   ( MODE=0 )
18: C
19: C      parameter(nN=1000)
20: C      integer IT(nN)
21: C data  IT / 0, 2, 5, 0, 12, 0, 9, 2, 2, 5, 10 /
22: C do i=1,nN
23: C     read(*,*,end=99) it(i)
24: C end do
25: C i = i-1
26: C 99 n = i
27: C call iorder( it, N, m)
28: C write(*,*) m
29: C write(*,*) (it(i),i=1,n)
30: C end
31: C
32: C=====
33:      integer IT(N)
34: C
35:      MX      = 0
36:      do 100 I = 1, N
37:          MX      = MAX(MX,IT(I))
38: 100 continue
39:      if ( MX.eq.0 ) then
40:          M      = 0
41:          return
42:      end if
43: C
44:      M      = 0
45:      do 130 K = 1, N
46:          M0      = MX
47:          do 110 I = 1, N
48:              if ( IT(I).gt.M ) M0      = MIN(M0,IT(I))
49: 110 continue
50:          M      = M + 1
51:          do 120 I = 1, N
52:              if ( IT(I).eq.M0 ) IT(I)      = M
53: 120 continue
54: C
55:          if ( M0.eq.MX ) return
56: C
57: 130 continue
58: C
59:      return
60:      end
```

src/shared/iounud.f

```
1:      subroutine IOUNUD(IU, IS, IE)
2: C=====
3: C purpose:  Return an unused I/O unit number from unit IS to IE.
4: C
5: C   I/O unit 0,5,6 & 7 is excluded, and return negative number
6: C   when no available I/O unit is found.
7: C   if both IE & IS are zero, search from 1 to 99.
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C o  IU : unused I/O unit. Negative if none found.
11: C i  IS, IE : check I/O unit from IS to IE
12: C=====
13:      logical OPD
14: C
15:      I1 = 1
16:      I2 = 99
17:      if( IS.ne.0 .or. IE.ne.0 ) then
18:          I1 = IS
19:          I2 = IE
20:      end if
21:      if( I1.gt.I2 ) then
22:          II = I1
23:          I1 = I2
24:          I2 = II
25:      end if
26: C
27:      do IUU = I1, I2
28:          if ( IUU.ne.0.and.IUU.ne.5.and.IUU.ne.6.and.IUU.ne.7 ) then
29:              inquire(unit =IUU,opened =OPD)
30:              if ( .not.OPD ) then
31:                  IU = IUU
32:                  goto 100
33:              end if
34:          end if
35:      end do
36: C
37:      IU = -1
38: C
39:      100 return
40:      end
```

src/shared/istrcm.f

```

1:      function ISTRCM(STR1,STR2)
2:      C
3:      C      GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C      PURPOSE:  COMPARE TWO CHARACTER STRINGS ON A ORDER-RULES
7:      C                  FOR GMVP/MVP'S NAMES.
8:      C
9:      C      <<RULES>>
10:     C
11:     C      RETURNS   -1   IF  STR1 < STR2
12:     C      RETURNS    0   IF  STR1 = STR2
13:     C      RETURNS    1   IF  STR1 > STR2
14:     C
15:     C      STR1 = STR2  WHEN
16:     C      LENGTH OF STR1 = THAT OF STR2
17:     C      AND  STR(1:L1) = STR(1:L1)  ( L1 = LENGTH OF STR1 )
18:     C
19:     C      STR1 >< STR2  IN CASE 1 OR 2.
20:     C
21:     C      1.  LENGTH OF STR1 >< THAT OF STR2
22:     C
23:     C      2.  LENGTH OF STR1 = THAT OF STR2 , AND
24:     C          FOR SOME K ( =< LENGTH OF L1 )
25:     C
26:     C          STR1(1:K-1) = STR2(1:K-1)  AND
27:     C
28:     C          INDEX( CHARS , STR1(K:K) ) > < INDEX( CHARS, STR2(K:K) )
29:     C
30:     C=====
31:     C      character*(*) STR1, STR2
32:     C
33:     C
34:     C      .... CHARACTER LIST .....
35:     C
36:     C
37:     C      character*53 CHARS
38:     C      data CHARS /' @ $ % & - ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'/'
39:     C      .....2.....3.....4.....5.....6.....
40:     C
41:     C
42:     C      L1 = INDEX(STR1,' ') - 1
43:     C      IF( L1 .LT. 0 ) L1 = LEN(STR1)
44:     C      L2 = INDEX(STR2,' ') - 1
45:     C      IF( L2 .LT. 0 ) L2 = LEN(STR2)
46:     C
47:     C      L1      = LEN(STR1)
48:     C      L2      = LEN(STR2)
49:     C
50:     C      if ( L1.lt.L2 ) then
51:     C          ISTRCM = -1
52:     C          return
53:     C      else if ( L1.gt.L2 ) then
54:     C          ISTRCM = 1
55:     C          return
56:     C      end if
57:     C
58:     C
59:     C
60:     C      do 100 I = 1, LEN(STR1)
61:     C          K1      = INDEX(CHARS,STR1(I:I))
62:     C          K2      = INDEX(CHARS,STR2(I:I))
63:     C          if ( K1.gt.K2 ) then
64:     C              ISTRCM = 1
65:     C          return
66:     C          else if ( K1.lt.K2 ) then
67:     C              ISTRCM = -1
68:     C              return
69:     C          end if
70:     C      100 continue
71:     C          ISTRCM = 0
72:     C          return
73:     C      end

```

src/shared/itrmsp.f

```
1:      subroutine ITRGSP( IOG,   IRGSP, ISUSP, NREG,  NSUZON,NSPACE,
2:      &                  KSUZN, KREGI, NINPZ )
3: C=====
4: C PURPOSE: PREPARE SUBFRAME # & "REGION" NAME # TABLE FOR EACH REGION.
5: C CALLED IN: GEOMIN
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C i IOG : message printout I/O unit
9: C o IRGSP :
10: C i ISUSP, NREG, NSPACE, KSUZN, KREGI, NINPZ:
11: C      -> see INC/_PGEOM
12: C=====
13:      integer IRGSP(2,NREG)
14:      integer ISUSP(NSUZON,NSPACE), KSUZN(NINPZ,2), KREGI(NINPZ)
15: C
16:      do 110 N = 1, NSPACE
17:      do 100 I = 1, NSUZON
18:          IR = ISUSP(I,N)
19:          if ( IR.ne.0 ) then
20: C
21:              IRGSP(1,IR) = N
22: C
23: C ... KSUZN(I,1) : INPUT-ZONE # OF I'TH terminal-input-zones in cells
24: C
25:              IRGSP(2,IR) = KREGI(KSUZN(I,1))
26:          end if
27:      100 continue
28:      110 continue
29:      return
30:      end
```

src/shared/judge.f

```

1:      subroutine JUDGE( SUBNM, KZ,   INX,   X,   Y,   Z,   IFI,
2:      &                 SDA,   KZDA, KZAA, NSDA, NZDA, NZ1, JVMNT,
3:      &                 IFL, JSIMP, JRR,   KSREF, ISRF, FXYZ )
4: C=====
5: C <NVP/GMVP>
6: C PURPOSE: Judge whether points (X(I),Y(I),Z(I)) are in zone KZ or not
7: C CALLED IN: SEADONE,PICASO etc.
8: C CALLS:   NONE
9: C-----
10: C arguments ( i=input o=output w= work )
11: C
12: C i SUBNM : subroutine name called this routine
13: C i KZ   : target zone # to judge in/out
14: C i INX  : number of particles whose position is checked
15: C i X(INX) : x-coordinates
16: C i Y(INX) : y-coordinates
17: C i Z(INX) : z-coordinates
18: C o IFI   : result of judgement ( .true. = in, .false. = out )
19: C i SDA(NSDA) : surface data
20: C i KZDA(2,NZDA) : inside/outside flags and pointers to 'SDA'
21: C      KZDA(1,I) : sign * <pointer to 'SDA' array>
22: C      sign = 1 : inside of the surface
23: C      sign = -1 : outside of the surface
24: C      KZDA(2,I) :
25: C      0 = SURFACE IS simply connected BY 'AND'.
26: C      1 = SURFACE IS ONE OF THOSE COMPOSING A BODY WHOSE
27: C      SIGN IS 'MINUS' AND HAVING MORE THAN ONE SURFACES.
28: C      2 = SIMILAR TO KZDA(2,I) = 1, BUT THE LAST SURFACE
29: C      OF THE BODY.
30: C      -1 = this surface is not used for zone shape definition.
31: C i KZAA(NZONE+1) : kзда(1:2,kzaa(i)) to kзда(1:2,kzaa(i+1)-1)
32: C      represents surface composition of i"th zone.
33: C i NSDA : length of 'SDA' array
34: C i NZDA : length of 'NZDA' array
35: C i NZ1 : nzone + 1
36: C i JVMNT : (unused now)
37: C w IFL(INX) : logical working area
38: C i JSIMP : (unused now)
39: C
40: C i JRR : flags to memorize nearest surface to points or not
41: C
42: C < the following arguments are active only when 'JRR' is true. >
43: C
44: C i KSREF(nzda) : If KSREF(i) is non-zero, KZDA(1,i) points a surface
45: C      which compose reflective boundary (reflective surface) and
46: C      KSREF(i) is reflective surface number.
47: C
48: C o ISRF(inx) : return nearest reflective surface # in this array
49: C w FXYZ(inx) : working area to find nearest surface.
50: C o FXYZ(inx) : distance form current point to the nearest surface.
51: C
52: C=====
53:      implicit real*8(D)
54: C
55:      character*6 SUBNM
56:      real*8 X(INX), Y(INX), Z(INX)
57: C
58: C ..... RESULTS OF JUDGEMENT .....
59: C
60:      logical IFI(INX)
61:      integer ISRF(INX)
62: C
63: C ..... GEOMETRY .....
64: C
65:      real*8 SDA(NSDA)

```

```

66:      integer KZDA(2,NZDA), KZAA(NZ1), KSREF(NZDA)
67: C
68: C ..... WORKING AREA .....
69: C
70:      real FXYZ(INX)
71:      logical IFL(INX), LD1, JRR
72: C
73:      include 'INC/_IOUNIT'
74: C
75:      parameter( EPS = 0.0 )
76: C
77: C-----
78: C
79:      do 100 I = 1, INX
80:         IFL(I) = .false.
81:         IFI(I) = .true.
82:         if ( JRR ) FXYZ(I) = 1.0E30
83:      100 continue
84: C
85:      do 230 N = KZAA(KZ), KZAA(KZ+1) - 1
86:         LLG = KZDA(2,N)
87:         if ( LLG.lt.0 ) go to 230
88: C
89:         JS = -SIGN(1,KZDA(1,N))
90:         LS = ABS(KZDA(1,N))
91:         KSF = INT(SDA(LS))
92:         LS = LS + 1
93: C
94: C << Surface type 1 >>
95: C ..... SLAB .....
96: C
97: C-----
98: C      Space between two parallel planes:
99: C
100: C      A*X + B*X + C*Y - P1 = 0
101: C      A*X + B*X + C*Y - P2 = 0
102: C
103: C      A**2 + B**2 + C**2 = 1
104: C
105: C      SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
106: C      SDA(LS+4) = P1, SDA(LS+5) = P2
107: C-----
108: C
109:      if ( KSF.eq.1 ) then
110:         if ( JRR ) then
111:            *VOCL LOOP,NOVREC
112:            do 110 I = 1, INX
113: C
114:            D1 = SDA(LS)*X(I) + SDA(LS+1)*Y(I) + SDA(LS+2)*
115:            &      Z(I)
116:            D1 = (D1-SDA(LS+3))*(D1-SDA(LS+4)) *JS
117:            D3 = (D1-SDA(LS+3))
118:            D4 = (D1-SDA(LS+4))
119:            CM LD1 = D1 .GE. EPS
120:            LD1 = D3*D4*JS.ge.EPS
121:            CM D1 = ABS(D1)
122: C
123: C      ... D1 is distance to the nearest point on the surface.
124: C
125:            D1 = MIN(ABS(D3),ABS(D4))
126:            if ( D1.lt.FXYZ(I).and.LD1 ) then
127:               ISRF(I) = KSREF(N)
128:               FXYZ(I) = D1
129:            end if
130:

```

src/shared/judge.f

```

131:         if ( LLG.eq.0 ) then
132:             IFI(I) = IFI(I) .and.LD1
133:         else
134:             IFL(I) = IFL(I) .or. LD1
135:             if ( LLG.eq.2 ) then
136:                 IFI(I) = IFI(I) .and.IFL(I)
137:                 IFL(I) = .false.
138:             end if
139:         end if
140: 110      continue
141:     else
142: *VOCL LOOP,NOVREC
143:         do 120 I = 1, INX
144:
145:             D1 = SDA(LS)*X(I) + SDA(LS+1)*Y(I) + SDA(LS+2)*
146: &             Z(I)
147:             LD1 = (D1-SDA(LS+3))*(D1-SDA(LS+4))*JS.ge.EPS
148:
149:             if ( LLG.eq.0 ) then
150:                 IFI(I) = IFI(I) .and.LD1
151:             else
152:                 IFL(I) = IFL(I) .or. LD1
153:                 if ( LLG.eq.2 ) then
154:                     IFI(I) = IFI(I) .and.IFL(I)
155:                     IFL(I) = .false.
156:                 end if
157:             end if
158: 120      continue
159:         end if
160: C
161: C << Surface type 2 >>
162: C ..... SPHERE .....
163: C -----
164: C (X-A)**2 + (Y-B)**2 + (Z-C)**2 - R**2 < 0
165: C
166: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
167: C SDA(LS+3) = R**2
168: C -----
169:         else if ( KSF.eq.2 ) then
170:             if ( JRR ) then
171:                 D3 = 0.5D0/(SQRT(SDA(LS+3))+1.0D-30)
172:                 do 130 I = 1, INX
173:                     D1 = ((SDA(LS)-X(I))**2+(SDA(LS+1)-Y(I))**2
174: &                     +(SDA(LS+2)-Z(I))**2-SDA(LS+3))*JS
175:                     LD1 = D1.ge.EPS
176: CM                     D1 = ABS(D1)
177: CS                     D1 = ABS(D1) /SQRT(SDA(LS+3))
178: C
179: C ... D1 is approximately distance to the surface when the point is
180: C near the surface.
181: C D1 = |P**2-R**2|/(2R) = |P+R||P-R|/(2R) => |P-R|
182: C
183:             D1 = ABS(D1)*D3
184:             if ( D1.lt.FXYZ(I).and.LD1 ) then
185:                 ISRF(I) = KSREF(N)
186:                 FXYZ(I) = D1
187:             end if
188:
189:             if ( LLG.eq.0 ) then
190:                 IFI(I) = IFI(I) .and.LD1
191:             else
192:                 IFL(I) = IFL(I) .or. LD1
193:                 if ( LLG.eq.2 ) then
194:                     IFI(I) = IFI(I) .and.IFL(I)
195:                     IFL(I) = .false.

```

```

196:         end if
197:     end if
198: 130      continue
199:     else
200:         do 140 I = 1, INX
201:             LD1 = ((SDA(LS)-X(I))**2+(SDA(LS+1)-Y(I))**2
202: &             +(SDA(LS+2)-Z(I))**2-SDA(LS+3))*JS.ge.EPS
203:             if ( LLG.eq.0 ) then
204:                 IFI(I) = IFI(I) .and.LD1
205:             else
206:                 IFL(I) = IFL(I) .or. LD1
207:                 if ( LLG.eq.2 ) then
208:                     IFI(I) = IFI(I) .and.IFL(I)
209:                     IFL(I) = .false.
210:                 end if
211:             end if
212: 140      continue
213:         end if
214: C
215: C << Surface type 3 >>
216: C ..... CYLINDER .....
217: C -----
218: C (X-A)**2 + (Y-B)**2 + (Z-C)**2
219: C -( (X-A)*K + (Y-B)*L + (Z-C)*M )**2 - R**2 < 0
220: C
221: C Where K**2 + L**2 + M**2 = 1
222: C
223: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
224: C SDA(LS+3) = K, SDA(LS+4) = L, SDA(LS+5) = M
225: C SDA(LS+6) = R**2
226: C -----
227:         else if ( KSF.eq.3 ) then
228:             if ( JRR ) then
229:                 D3 = 0.5D0/(SQRT(SDA(LS+6))+1.0D-30)
230:                 do 150 I = 1, INX
231:                     DX = X(I) - SDA(LS)
232:                     DY = Y(I) - SDA(LS+1)
233:                     DZ = Z(I) - SDA(LS+2)
234:
235:                     D1 = JS*(DX**2+DY**2+DZ**2-SDA(LS+6)
236: &                     -(SDA(LS+3)*DX+SDA(LS+4)*DY+SDA(LS+5)*DZ)**2)
237:                     LD1 = D1.ge.EPS
238: CM                     D1 = ABS(D1)
239: CS                     D1 = ABS(D1) /SQRT(SDA(LS+6))
240: C
241: C ... D1 is approximately distance to the surface when the point is
242: C near the surface.
243: C D1 = |P**2-R**2|/(2R) = |P+R||P-R|/(2R) => |P-R|
244: C
245:             D1 = ABS(D1)*D3
246:             if ( D1.lt.FXYZ(I).and.LD1 ) then
247:                 ISRF(I) = KSREF(N)
248:                 FXYZ(I) = D1
249:             end if
250:
251:             if ( LLG.eq.0 ) then
252:                 IFI(I) = IFI(I) .and.LD1
253:             else
254:                 IFL(I) = IFL(I) .or. LD1
255:                 if ( LLG.eq.2 ) then
256:                     IFI(I) = IFI(I) .and.IFL(I)
257:                     IFL(I) = .false.
258:                 end if
259:             end if
260: 150      continue

```

src/shared/judge.f

```

261:      else
262:      do 160 I = 1, INX
263:      DX = X(I) - SDA(LS)
264:      DY = Y(I) - SDA(LS+1)
265:      DZ = Z(I) - SDA(LS+2)
266:
267:      LD1 = JS*(DX**2+DY**2+DZ**2-SDA(LS+6)
268:      & -(SDA(LS+3)*DX+SDA(LS+4)*DY+SDA(LS+5)*DZ)**2)
269:      & .ge.EPS
270:
271:      if ( LLG.eq.0 ) then
272:      IFL(I) = IFL(I) .and.LD1
273:      else
274:      IFL(I) = IFL(I) .or. LD1
275:      if ( LLG.eq.2 ) then
276:      IFI(I) = IFI(I) .and.IFL(I)
277:      IFL(I) = .false.
278:      end if
279:      end if
280:      continue
281:      end if
282: C
283: C << Surface type 4 >>
284: C ..... GENERAL QUADRATIC SURFACE
285: C -----
286: C A*X**2 + B*Y**2 + C*Z**2 +
287: C D*X*Y + E*Y*Z + F*Z*X +
288: C G*X + H*Y + I*Z + J < 0
289: C
290: C or
291: C
292: C
293: C ( X Y Z 1 ) | A D F G | | X |
294: C | 0 B E H | | Y |
295: C | 0 0 0 J | | Z |
296: C | 0 0 0 J | | 1 |
297: C
298: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
299: C SDA(LS+3) = D, SDA(LS+4) = E, SDA(LS+5) = F
300: C SDA(LS+6) = G, SDA(LS+7) = H, SDA(LS+8) = I
301: C SDA(LS+9) = J
302: C
303: C << Distance to the surface >>
304: C
305: C When current point {p} is near the surface F{x} = 0, the surface is
306: C approximately a plane like;
307: C
308: C ([grad F{x}]{x=p},{X}) + F{p} = 0
309: C
310: C where {X} is displacement vector from {p}.
311: C
312: C So the distance is approximately;
313: C
314: C D = |F{p} / grad F{p}|
315: C
316: C -----
317:      else if ( KSF.eq.4 ) then
318:      if ( JRR ) then
319:
320:      do 170 I = 1, INX
321:      D1 = (
322:      & X(I)*( X(I)*SDA(LS) +Y(I)*SDA(LS+3) +Z(I)*SDA(LS+5) +SDA(LS+6))
323:      & +Y(I)*( Y(I)*SDA(LS+1) +Z(I)*SDA(LS+4) +SDA(LS+7))
324:      & +Z(I)*( Z(I)*SDA(LS+2) +SDA(LS+8))
325:      & +

```

```

326:      & ) * JS
327:      LD1 = D1.ge.EPS
328:
329:      DGF =
330:      & 2*X(I)*SDA(LS) +Y(I)*SDA(LS+3) +Z(I)*SDA(LS+5) + SDA(LS+6)
331:      DGFY =
332:      & 2*Y(I)*SDA(LS+1)+Z(I)*SDA(LS+4) + X(I)*SDA(LS+3) +SDA(LS+7)
333:      DGFZ =
334:      & 2*Z(I)*SDA(LS+2)+X(I)*SDA(LS+5)+Y(I)*SDA(LS+4)+SDA(LS+8)
335:
336: C ... |grad F{p}|
337: DGF = SQRT(DGF*DGFX+DGFY*DGFY+DGFZ*DGFZ)+1.0D-30
338: CS D1 = ABS(D1)
339: D1 = ABS(D1)/DGF
340: if ( D1.lt.FXYZ(I).and.LD1 ) then
341: ISRF(I) = KSREF(N)
342: FXYZ(I) = D1
343: end if
344:
345: if ( LLG.eq.0 ) then
346: IFI(I) = IFI(I) .and.LD1
347: else
348: IFL(I) = IFL(I) .or. LD1
349: if ( LLG.eq.2 ) then
350: IFI(I) = IFI(I) .and.IFL(I)
351: IFL(I) = .false.
352: end if
353: end if
354: 170 continue
355: else
356: do 180 I = 1, INX
357:
358: LD1 = (
359: & X(I)*( X(I)*SDA(LS) +Y(I)*SDA(LS+3) +Z(I)*SDA(LS+5) +SDA(LS+6))
360: & +Y(I)*( Y(I)*SDA(LS+1) +Z(I)*SDA(LS+4) +SDA(LS+7))
361: & +Z(I)*( Z(I)*SDA(LS+2) +SDA(LS+8))
362: & +
363: & ) * JS .GE. EPS
364:
365: if ( LLG.eq.0 ) then
366: IFI(I) = IFI(I) .and.LD1
367: else
368: IFL(I) = IFL(I) .or. LD1
369: if ( LLG.eq.2 ) then
370: IFI(I) = IFI(I) .and.IFL(I)
371: IFL(I) = .false.
372: end if
373: end if
374: 180 continue
375: end if
376: C
377: C << Surface type 5 >>
378: C ..... PLANE (HALF SPACE)
379: C -----
380: C A*X + B*Y + C*Z - D < 0
381: C
382: C A**2 + B**2 + C**2 = 1
383: C
384: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
385: C SDA(LS+3) = D
386: C -----
387:      else if ( KSF.eq.5 ) then
388:      if ( JRR ) then
389:      do 190 I = 1, INX
390:      D1 = JS*

```

src/shared/judge.f

```

391:      &              (X(I)*SDA(LS)+Y(I)*SDA(LS+1)+Z(I)*SDA(LS+2)
392:      &              -SDA(LS+3))
393:      LD1 = D1.ge.EPS
394:      D1 = ABS(D1)
395:      if ( D1.lt.FXYZ(I).and.LD1 ) then
396:        ISRF(I) = KSREF(N)
397:        FXYZ(I) = D1
398:      end if
399:
400:      if ( LLG.eq.0 ) then
401:        IFI(I) = IFI(I) .and.LD1
402:      else
403:        IFL(I) = IFL(I) .or. LD1
404:        if ( LLG.eq.2 ) then
405:          IFI(I) = IFI(I) .and.IFL(I)
406:          IFL(I) = .false.
407:        end if
408:      end if
409: 190      continue
410:    else
411:      do 200 I = 1, INX
412:        LD1 = JS*
413:        &      (X(I)*SDA(LS)+Y(I)*SDA(LS+1)+Z(I)*SDA(LS+2)
414:        &      -SDA(LS+3)) .ge.EPS
415:        if ( LLG.eq.0 ) then
416:          IFI(I) = IFI(I) .and.LD1
417:        else
418:          IFL(I) = IFL(I) .or. LD1
419:          if ( LLG.eq.2 ) then
420:            IFI(I) = IFI(I) .and.IFL(I)
421:            IFL(I) = .false.
422:          end if
423:        end if
424: 200      continue
425:    end if
426:  C
427:  C << Surface type 6 >>
428:  C ..... TORUS .....
429:  C-----
430:  C
431:  C      2      2
432:  C      B T + (R-A) + C T(R-A) - D = 0
433:  C
434:  C Where
435:  C
436:  C T = NX*(X-VX) + NY*(Y-VY)+NZ*(Z-VZ)
437:  C (VX,VY,VZ) : Center of the torus
438:  C (NX,NY,NZ) : Rotation axis vector of the torus (unit vector)
439:  C
440:  C R : Distance from rotation axis
441:  C      2      2      2
442:  C      R = (X-VX) + (Y-VY) + (Z-VZ) - T
443:  C
444:  C A : Rotation radius
445:  C
446:  C SDA(LS) = VX, SDA(LS+1) = VY, SDA(LS+2) = VZ
447:  C SDA(LS+3) = NX, SDA(LS+4) = NY, SDA(LS+5) = NZ
448:  C SDA(LS+6) = A
449:  C SDA(LS+7) = B
450:  C SDA(LS+8) = C
451:  C SDA(LS+9) = D
452:  C
453:  C << Distance to the surface >>
454:  C
455:  C When current point {p} is near the surface of the torus;

```

```

456:  C gradient of F{x} in (T,R,theta) coordinates is approximated by
457:  C
458:  C grad F{T,R,theta} = ( dF/dT, dF/dR, 0 )
459:  C So the norm of grad F is approximately;
460:  C
461:  C |grad F| => sqrt[(2B T+C*(R-A))**2 + (2*(R-A)+C*T)**2]
462:  C
463:  C-----
464:  C
465:      else if ( KSF.eq.6 ) then
466:        if ( JRR ) then
467:          do 210 I = 1, INX
468:            DX = X(I) - SDA(LS)
469:            DY = Y(I) - SDA(LS+1)
470:            DZ = Z(I) - SDA(LS+2)
471:            DXX = SDA(LS+3)*DX + SDA(LS+4)*DY + SDA(LS+5)*DZ
472:            DX2 = DXX*DXX
473:            DR = SQRT(DX**2+DY**2+DZ**2-DX2) - SDA(LS+6)
474:            D1 = JS*
475:            &      (SDA(LS+7)*DX2+DR*(DR+DXX*SDA(LS+8))-SDA(LS+9)
476:            &      )
477:            LD1 = D1.ge.EPS
478:            cccccccc ----- IF(JRR) THEN
479:
480:            C      ... |grad F{p}|
481:            C      DGF = SQRT(
482:            &      (2.0d0*SDA(LS+7)*DXX+SDA(LS+8)*DR)**2
483:            &      +(2.0d0*DR + SDA(LS+8)*DXX)**2) + 1.0D-30
484:
485:            CS      D1 = ABS(D1)
486:            D1 = ABS(D1)/DGF
487:            if ( D1.lt.FXYZ(I).and.LD1 ) then
488:              ISRF(I) = KSREF(N)
489:              FXYZ(I) = D1
490:            end if
491:            cccccccc ----- ENDIF
492:          C
493:          C ..... ONLY LLG=0 IS ALLOWED FOR TORUS .....
494:          C
495:          IFI(I) = IFI(I) .and.LD1
496: 210      continue
497:        else
498:          do 220 I = 1, INX
499:            DX = X(I) - SDA(LS)
500:            DY = Y(I) - SDA(LS+1)
501:            DZ = Z(I) - SDA(LS+2)
502:            DXX = SDA(LS+3)*DX + SDA(LS+4)*DY + SDA(LS+5)*DZ
503:            DX2 = DXX*DXX
504:            DR = SQRT(DX**2+DY**2+DZ**2-DX2) - SDA(LS+6)
505:            D1 = JS*
506:            &      (SDA(LS+7)*DX2+DR*(DR+DXX*SDA(LS+8))-SDA(LS+9)
507:            &      )
508:            LD1 = D1.ge.EPS
509:          C
510:          C ..... ONLY LLG=0 IS ALLOWED FOR TORUS .....
511:          C
512:          IFI(I) = IFI(I) .and.LD1
513: 220      continue
514:        end if
515:      C
516:      C
517:      else
518:        write(IMG,* ) ' !! (JUDGE) INVALID SURFACE TYPE ', KSF, KZ
519:        write(IMG,* ) ' (JUDGE called from ', SUBNM, ' )'
520:        stop 666

```


src/shared/judge.f

```
521:         end if
522: C
523: 230 continue
524: return
525: end
```

SAFE

src/shared/keep2.f

```
1:      subroutine KEEP2( VNAME, LX,      NL,      TYPE,  LX2,  JDEBG )
2: C=====
3: C  PURPOSE: SET NEW POINTER IN A ALREDY KEPT ARRAY AREA.
4: C  CALLED IN: INTRO
5: C  HISTORY: ADDED BY M.SASAKI   JUNE 12 (1991)
6: C           TO SET BANK POINTERS OF IMAGINARY PARTICLES FOR NEXT EVENT
7: C           ESTIMATOR.
8: C=====
9:      character VNAME*6, TYPE*2
10:     integer JDEBG(*)
11:     include 'INC/_WORDL'
12:     include 'INC/_IUNIT'
13: C
14:     if ( TYPE.eq.'R8' ) then
15:         LX2      = LX + MWORD*NL
16:     else
17:         LX2      = LX + NL
18:     end if
19: C
20:     if ( JDEBG(1).ne.0 ) then
21:         write(IPR, '(1x,a,a,a,i10)') ' <', VNAME,'> AREA KEPT FROM ',LX2
22:     end if
23: C
24:     return
25: end
```

src/shared/keepc.f

```

1:      subroutine KEEPC( VNAME, LX,      NL,      TYPE,  LAST,  JDEBG )
2:      C
3:      C      GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C PURPOSE: KEEPC... Reserve character memory from index LX by NL units
7:      C              in array in common /CARRAY/
8:      C
9:      C Unit of reservation is character*(CHAUNT), where CHAUNT is
10:     C integer constant defined near common /CARRAY/. (currently CHAUNT=4)
11:     C
12:     C Related routines:
13:     C
14:     C REMAINC : check the size of remaining amount of memory
15:     C RESIZEC : resize a memory area already reserved by 'KEEPC'
16:     C SETLASTC : set the last position of available memory
17:     C GETLASTC : get the last position of available memory
18:     C
19:     C (memory reservation with data initialization like KEPCV routine for
20:     C non-character data does not exist.)
21:     C-----
22:     C arguments ( i = input, o = output, c = constant, w = work )
23:     C
24:     C i VNAME : name of data reserved etc.
25:     C oi LX : memory pointer as array index ( CHA(*) of common /CARRAY/)
26:     C io NL : size of memory reserved (of reset, remaining ...)
27:     C i TYPE : character length specification ( 'Cxxx' )
28:     C          uses as character*(xxx) array.
29:     C oi LAST : array index of the starting address of free area
30:     C          ( CHA(*) of common /CARRAY/)
31:     C i JDEBG : option for debugging printout.
32:     C i V : data value with which memory area is initialized.
33:     C-----
34:     C <common variable refers or defines>
35:     C
36:     C CHA(*) : in /CARRAY/
37:     C LIMITC : in /CARRAY/
38:     C-----
39:     C CALLED IN: MAINLY IN 'INTRO' BUT ALSO IN OTHER ROUTINES
40:     C CALLS: IGTF LG (FUNCTION)
41:     C
42:     C
43:     C=====
44:     C
45:     C include 'INC/_ARRAY'
46:     C include 'INC/_IOUNIT'
47:     C
48:     C character VNAME*(*), TYPE*(*)
49:     C integer JDEBG(*)
50:     C include 'INC/_WORDL'
51:     C character*6 CWRK
52:     C-----
53:     C
54:     C LX = IAINFC(2)
55:     C
56:     C if ( TYPE(1:1).eq.'C' ) then
57:     C CWRK = TYPE(2:)
58:     C read(CWRK,'(BN,I6)') LC
59:     C LCW = (LC*NL+CHAUNT-1) /CHAUNT
60:     C
61:     C LAST = LX + LCW
62:     C else
63:     C write(IMG,*) 'XXX(KEEPC) Invalid data type <', TYPE, '>'
64:     C stop 666
65:     C end if

```

```

66:     C
67:     C IAINFC(2) = LAST
68:     C
69:     C if ( LAST+IAINFC(1)-1.gt.LIMITC+1 ) then
70:     C write(IMG,'(/lx,a,a,a,a,i10,a,i10,a,a,i10,a/)')
71:     C & 'XXX MEMORY OVER IN KEEPING <', VNAME(:ICLEN2(VNAME)),
72:     C & '> AS ', ' CHARACTER ARRAY (LENGTH = ', LAST - LX,
73:     C & ' (UNIT)) BY ', LAST - LIMITC, ' (UNIT) ', ' LIMITC=',
74:     C & LIMITC, ' XXX'
75:     C IKK = 1
76:     C call CNTERR( 'FATAL' )
77:     C IAINFC(3) = MAX(IAINFC(3),LAST-IAINFC(1)+1)
78:     C end if
79:     C
80:     C if ( JDEBG(1).ne.0 .or. IGTF LG('JMCHK').ne.0 )
81:     C & write(IPR,'(/lx,a,a,a,a,i10,a,i10/)') ' <',
82:     C & VNAME(:ICLEN2(VNAME)), '> CHARACTER AREA KEPT FROM ', LX,
83:     C & ' TO ', LAST - 1
84:     C
85:     C end
86:     C
87:     C
88:     C=====
89:     C KEEPC: Keep memory and assign value (not prepared for character)
90:     C=====
91:     C
92:     C subroutine KEPVC( VNAME, LX, NL, TYPE, LAST, V, JDEBG )
93:     C
94:     C include 'INC/_ARRAY'
95:     C include 'INC/_IOUNIT'
96:     C
97:     C character VNAME*(*), TYPE*(*)
98:     C integer JDEBG(*)
99:     C character*(*) V
100:    C include 'INC/_WORDL'
101:    C
102:    C character*6 CWRK
103:    C
104:    C-----
105:    C
106:    C call KEEPC( VNAME, LX, NL, TYPE, LAST, JDEBG )
107:    C
108:    C cwrk = type(2:)
109:    C read(cwrk,'(bn,i6)') lc
110:    C klc = lc / chaunt
111:    C
112:    C if ( LAST-IAINFC(1).le.LIMITC+1 ) then
113:    C do 100 I=LX,LAST-1
114:    C CHA(I) = ' '
115:    C 100 continue
116:    C end if
117:    C return
118:    C end
119:    C
120:    C
121:    C=====
122:    C REMAINC: Return remaining array memory size
123:    C=====
124:    C
125:    C subroutine REMAINC( VNAME, NL, TYPE, LAST )
126:    C
127:    C include 'INC/_ARRAY'
128:    C include 'INC/_IOUNIT'
129:    C
130:    C character VNAME*(*), TYPE*(*)

```

src/shared/keepc.f

```

131:      include 'INC/_WORDL'
132: C
133:      character*6 CWRK
134: C-----
135: C
136:      LL      = IAINFC(2)
137:      LAST    = IAINFC(2)
138: C
139:      LTY      = INDEX(TYPE,' ') - 1
140:      if ( LTY.lt.0 ) LTY = LEN(TYPE)
141: C
142:      if ( TYPE(1:1).eq.'C' ) then
143:          CWRK  = TYPE(2:)
144:          read(CWRK,'(bn,i6)') LC
145:      else
146:          write(IMG,*) 'XXX(REMAING) Invalid data type <', TYPE, '>'
147:          stop 666
148:      end if
149: C
150:      NL      = ((LIMITC-(LL-IAINFC(1)+1)+1) / ((LC+CHAUNT-1)/CHAUNT))
151: C
152:      return
153:      end
154: C
155: C=====
156: C RESIZEC:
157: C ===== Resize character array and get last pointer =====
158: C      ( Do not call this entry for variable arrays of
159: C        NON-HIGHEST addresses. )
160: C=====
161: C
162:      subroutine RESIZEC( VNAME, LX,      NL,      TYPE, LAST )
163: C
164:      include 'INC/_ARRAY'
165:      include 'INC/_IOUNIT'
166: C
167:      character VNAME*(*), TYPE*(*)
168:      include 'INC/_WORDL'
169: C
170:      character*6 CWRK
171: C-----
172: C
173:      if ( TYPE(1:1).eq.'C' ) then
174:          CWRK  = TYPE(2:)
175:          read(CWRK,'(bn,i6)') LC
176:          LCW   = (LC*NL+CHAUNT-1) /CHAUNT
177:          LAST  = LX + LCW
178:      else
179:          write(IMG,*) 'XXX(RESIZEC) Invalid data type <', TYPE, '>'
180:          stop 666
181:      end if
182: C
183:      IAINFC(2)  = LAST
184: C
185:      return
186:      end
187: C
188: C
189: C=====
190: C STLASTC : SET LAST POINTER to LX
191: C=====
192: C
193: C ( used to discard temporary data . etc. )
194: C
195:      subroutine STLASTC( VNAME, LX,      LAST )

```

```

196: C
197:      include 'INC/_ARRAY'
198:      include 'INC/_IOUNIT'
199: C
200:      character VNAME*(*)
201: C-----
202: C
203:      if ( LX.lt.IAINFC(1) .or. LX.gt.IAINFC(1)+LIMITC-1 ) then
204:          write(IMG, '(/lx,a,a)') 'XXX TRIED TO RESET <LAST> INDEX OF ',
205:          &      ' CHARACTER MEMORY AREA IN /CARRAY/ WITH INVALID ADDRESS '
206:          write(IMG, '(lx,a,a,i10)') ' Place <', VNAME(:ICLEN2(VNAME)),
207:          &      ' >      index = ', LX
208:          write(IMG, '(lx,a,i10,a,i10/)') ' valid index is from ',
209:          &      IAINFC(1), ' to ', LIMITC
210:          call PRSTOP( 1, 'ERROR IN CHARACTER MEMORY HANDLING' )
211:          stop 666
212:      end if
213: C
214:      IAINFC(2)  = LX
215:      LAST      = IAINFC(2)
216:      return
217:      end
218: C
219: C
220: C=====
221: C GTLASTC: Get the last index =====
222: C=====
223: C
224: C
225:      subroutine GTLASTC( LAST )
226: C
227:      include 'INC/_ARRAY'
228: C
229: C-----
230:      LAST      = IAINFC(2)
231:      return
232:      end

```

src/shared/keep.f

```

1:      subroutine KEEP( VNAME, LX,      NL,      TYPE,  LAST,  JDEBG )
2:      C
3:      C  GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C  PURPOSE: KEEP... RESERVE MEMORY FROM ADDRESS LX BY NL WORD
7:      C              IN ARRAY IN COMMON /ARRAY/
8:      C              KEPV... LIKE KEEP, MOREOVER ASSIGN VALUE V TO MEMORY
9:      C
10:     C-----
11:     C  arguments ( i = input, o = output, c =constant, w = work )
12:     C
13:     C i  VNAME : name of data reserved etc.
14:     C oi LX   : memory pointer as array index ( A(*) of common /ARRAY/)
15:     C io NL   : size of memory reserved ( or reset, remaining ...)
16:     C i  TYPE : data type ( 'I4','R4','R8','C...' 'I4D','R4D' )
17:     C oi LAST : memory pointer of the starting address of free area
18:     C              as array index ( A(*) of common /ARRAY/)
19:     C i  JDEBG : option for debugging printout.
20:     C i  V     : data value with which memory area is initialized.
21:     C-----
22:     C <common variable refers or defines>
23:     C
24:     C  A(*) : in /ARRAY/
25:     C  IAINF(3) : in /ARRAY/
26:     C-----
27:     C CALLED IN: MAINLY IN 'INTRO' BUT ALSO IN OTHER ROUTINES
28:     C CALLS: IGTF LG (FUNCTION)
29:     C
30:     C  Related routines:
31:     C
32:     C      KEPV : keep memory with value initialization.
33:     C      REMAIN : check the size of remaining amount of memory
34:     C      RESIZE : resize a memry area already reserved by 'KEEP'.
35:     C      stlast : set the last position of available memory
36:     C      gtlast : get the last position of available memory
37:     C
38:     C=====
39:     C
40:     C  include 'INC/_ARRAY'
41:     C  include 'INC/_IOUNIT'
42:     C
43:     C  character VNAME*(*), TYPE*(*)
44:     C  integer JDEBG(*)
45:     C  include 'INC/_WORDL'
46:     C-----
47:     C  logical JDBND
48:     C
49:     C ---- CHECK FOR INVALID ASSIGNMENT ----
50:     C  parameter( MAXCHK = 1024, AM1 = -0.987E21, AM2 = 1.98769E-30 )
51:     C  common /CHKMEM/ NCHK, LCHK(MAXCHK)
52:     C  common /CHKMM2/ VCHK(MAXCHK)
53:     C  common /CHKFLG/ IFCHK
54:     C  character*8 VCHK
55:     C
56:     C  real*8 WSIZOF
57:     C
58:     C  LX      = IAINF(2)
59:     C
60:     C  if ( IFCHK.eq.0.and.IGTF LG('JMCHK').ne.0.and.NCHK.lt.MAXCHK ) then
61:     C      NCHK = NCHK + 1
62:     C      LCHK(NCHK) = LX
63:     C      VCHK(NCHK) = VNAME
64:     C      A(LX) = AM1
65:     C      A(LX+1) = AM2

```

```

66:     C      LX      = LX + 2
67:     C  end if
68:     C
69:     C  LTY      = INDEX(TYPE,' ') - 1
70:     C  if ( LTY.lt.0 ) LTY = LEN(TYPE)
71:     C
72:     C  ... requires double word boundary ...
73:     C
74:     C  JDBND = TYPE(1:2) .eq.'R8'
75:     C  if ( TYPE(LTY:LTY).eq.'D' ) then
76:     C      JDBND = .true.
77:     C      LTY = LTY - 1
78:     C  end if
79:     C
80:     C  if ( MWORD.gt.1 ) then
81:     C      if ( JDBND.and.MOD(LX,MWORD).eq.0 ) LX = LX + 1
82:     C  end if
83:     C
84:     C  ... TYPE = 'R8' : DOUBLE PRECISION/REAL = 32/64 BIT MACHIENS
85:     C
86:     C  if ( TYPE(:LTY).eq.'R8' ) then
87:     C      LAST = LX + NL*WSIZOF('R8')
88:     C
89:     C  ... TYPE = 'CXX' : CHARACTER OF XX BYTES .....
90:     C
91:     C  else if ( TYPE(1:1).eq.'C' ) then
92:     C      LAST = LX + NINT(DBLE(NL*WSIZOF(TYPE(:LTY)))+0.1D0)
93:     C
94:     C  ... from Nov. 1997 ...
95:     C      write(IMG,*)
96:     C      & 'XXX(KEEP) Program Error: Memory reservation of character data',
97:     C      & ' is not allowed (use KEEPC instead!!) VNAME <',
98:     C      & VNAME(:ICLEN2(VNAME)), '>'
99:     C      stop 666
100:    C
101:    C  ... TYPE = 'R4'/'I4' : REAL/INTEGER
102:    C
103:    C  else
104:    C      LAST = LX + NL*WSIZOF(TYPE(:LTY))
105:    C  end if
106:    C
107:    C  IAINF(2) = LAST
108:    C
109:    C  if ( LSIZ(LAST).gt.LIMIT+1 ) then
110:    C      write(IMG,('/lx,a,a,a,i10,a,a,i10,a,a,i10,a/a/'))
111:    C      & 'XXX MEMORY OVER IN KEEPING <', VNAME(:ICLEN2(VNAME)),
112:    C      & '> (LENGTH = ', LAST - LX, ' (WORD)) BY ',
113:    C      & LSIZ(LAST) - LIMIT - 1, ' (WORD) ', ' LIMIT=', LIMIT,
114:    C      & ' XXX'
115:    C      IKK = 1
116:    C      call CNTERR( 'FATAL' )
117:    C      IAINF(3) = MAX(IAINF(3),LSIZ(LAST))
118:    C  end if
119:    C
120:    CCCC if ( IGTF LG('JDEBG').ne.0 .or. IGTF LG('JMCHK').ne.0 )
121:    C      if ( JDEBG(1).ne.0 .or. IGTF LG('JMCHK').ne.0 ) then
122:    C          write(IPR,7020) VNAME(:ICLEN2(VNAME)),LX,LAST-1,LSIZ(LX),
123:    C          & LSIZ(LAST-1)
124:    C          7020 format(/lx,<'a,>'> AREA KEPT FROM ',i10,' TO ',i10,
125:    C          & '(',i10,' TO ',i10,')'/)
126:    C      end if
127:    C
128:    C  return
129:    C
130:    C  end

```

src/shared/keep.f

```

131:
132: C
133: C=====
134: C  KEPV: KEEP MEMORY AND ASSIGN VALUE
135: C=====
136: C
137:       subroutine KEPV(AA, VNAME, LX, NL, TYPE,  LAST,  V,      JDEBG )
138: C
139: C     ... array AA must be the array A(*) in common /ARRAY/ or
140: C     address pointed by CRAY type pointer in the common when
141: C     fixed memory size mode or CRAY pointer mode. Otherwise
142: C     it points an array dynamically allocated somewhere in the
143: C     program and passed to routines only as a dummy argument.
144: C
145:       real AA(*)
146: C
147:       include 'INC/_ARRAY'
148:       include 'INC/_IUNIT'
149: C
150:       character VNAME*(*), TYPE*(*)
151:       integer JDEBG(*)
152:       real V
153:       include 'INC/_WORDL'
154: C
155:       logical JDBND
156: C
157:       character*8 BLANK
158:       data BLANK /' ' /
159: C-----
160: C
161: C
162:       call KEEP( VNAME, LX, NL, TYPE, LAST, JDEBG )
163: C
164: C
165:       LTY      = INDEX(TYPE,' ') - 1
166:       if ( LTY.lt.0 ) LTY = LEN(TYPE)
167: C
168: C     ... requires double word boundary ....
169: C
170:       JDBND     = TYPE(1:2) .eq.'R8'
171:       if ( TYPE(LTY:LTY).eq.'D' ) then
172:         JDBND    = .true.
173:         LTY      = LTY - 1
174:       end if
175: C
176: C     FOR CHARACTER TYPE, BLANK INITIALIZATION ONLY.
177: C
178:       if ( LSIZ(LAST).le.LIMIT+1 ) then
179:         if ( TYPE(1:1).eq.'C' ) then
180: C
181: C     ... from Nov. 1997 ...
182:         write(IMSG,*)
183:         & 'XXX(KEPV) Program Error: Memory reservation of character data',
184:         & ' ' is not allowed (use KEEPC instead!!)  VNAME <',
185:         & VNAME(:ICLEN2(VNAME)), '>'
186:         stop 666
187:       if ( MWORD.eq.2 ) then
188:         do 100 L = LX, LAST - 1
189:           read(BLANK(:4),fmt ='(A4)') A(L)
190: C 100       continue
191:       else
192:         do 110 L = LX, LAST - 1
193:           read(BLANK(:8),fmt ='(A8)') A(L)
194: C 110       continue
195:       end if

```

```

196: C
197:       else if ( TYPE(:LTY).eq.'R8' ) then
198:         call PUTVD( A(LX), LAST-LX, V )
199:         call PUTVD( A(LX), NL, V )
200:         call PUTVD( AA(LX), NL, V )
201:       else if ( TYPE(:LTY).eq.'R4' ) then
202:         call PUTV( A(LX), LAST-LX, V )
203:         call PUTV( A(LX), NL, V )
204:         call PUTV( AA(LX), NL, V )
205:       else if ( TYPE(:LTY).eq.'I4' ) then
206:         call PUTVI( A(LX), LAST-LX, V )
207:         call PUTVI( A(LX), NL, V )
208:         call PUTVI( AA(LX), NL, V )
209:       end if
210:     end if
211:   return
212: end
213: C
214: C
215: C=====
216: C REMAIN: RETURN REMAINING MEMORY SIZE
217: C=====
218: C
219:       subroutine REMAIN( VNAME, NL,      TYPE,  LAST )
220: C
221:       include 'INC/_ARRAY'
222:       include 'INC/_IUNIT'
223: C
224:       character VNAME*(*), TYPE*(*)
225:       include 'INC/_WORDL'
226: C-----
227: C
228:       logical JDBND
229:       real*8 WSIZOF
230: C
231:       LL      = IAINF(2)
232:       LAST    = IAINF(2)
233: C
234:       LTY      = INDEX(TYPE,' ') - 1
235:       if ( LTY.lt.0 ) LTY = LEN(TYPE)
236: C
237: C     ... requires double word boundary ....
238: C
239:       JDBND     = TYPE(1:2) .eq.'R8'
240:       if ( TYPE(LTY:LTY).eq.'D' ) then
241:         JDBND    = .true.
242:         LTY      = LTY - 1
243:       end if
244: C
245: C     ... from Nov. 1997 ...
246:       if ( TYPE(1:1).eq.'C' ) then
247:         write(IMSG,*)
248:         & 'XXX(REMAIN) Program Error: Memory handling of character data',
249:         & ' ' is not allowed (use REMAINC instead!!)  VNAME <',
250:         & VNAME(:ICLEN2(VNAME)), '>'
251:         stop 666
252:       end if
253: C
254: C
255: C     .... TYPE = 'R8' : DOUBLE PRECISION/REAL = 32/64 BIT MACHIENS
256: C
257:       if ( JDBND ) then
258:         if ( MWORD.gt.1.and.MOD(LL,MWORD).eq.0 ) LL = LL + 1
259:       end if
260: C

```

src/shared/keep.f

```

261: C      AM      = WSIZOF(TYPE(:LTY))
262: C      NL      = (LIMIT-LL+1) /AM
263: C      NL      = (LIMIT-LL+1) /WSIZOF(TYPE(:LTY))
264: C      NL      = (LIMIT-LSIZ(LL)+1) /WSIZOF(TYPE(:LTY))
265: C
266: C      return
267: C      end
268: C
269: C=====
270: C RESIZE:
271: C ===== RESIZE ARRAY AND GET LAST POINTER =====
272: C      ( DO NOT CALL THIS ENTRY FOR VARIABLE ARRAYS OF
273: C        NON-HIGHEST ADDRESSES. )
274: C=====
275: C
276: C      subroutine RESIZE( VNAME, LX,      NL,      TYPE,  LAST )
277: C
278: C      include 'INC/_ARRAY'
279: C      include 'INC/_IOUNIT'
280: C
281: C      character VNAME*(*), TYPE*(*)
282: C      include 'INC/_WORDL'
283: C      real*8 WSIZOF
284: C-----
285: C      LTY      = INDEX(TYPE,' ') - 1
286: C      if ( LTY.lt.0 ) LTY = LEN(TYPE)
287: C
288: C      ... requires double word boundary ....
289: C
290: C      if ( TYPE(LTY:LTY).eq.'D' ) then
291: C          LTY      = LTY - 1
292: C      end if
293: C      if ( TYPE(1:1).eq.'C' ) then
294: C          LAST      = LX + NINT(DBLE(NL*WSIZOF(TYPE(:LTY)))+0.1D0)
295: C
296: C      ... from Nov 1997 ...
297: C          write(IMG,*)
298: C          & 'XXX(RESIZE) Program Error: Memory handling of character data',
299: C          & ' is not allowed (use RESIZEC instead!!) VNAME <',
300: C          & VNAME(:ICLEN2(VNAME)), '>'
301: C          stop 666
302: C
303: C      else
304: C          LAST      = LX + NL*WSIZOF(TYPE(:LTY))
305: C      end if
306: C
307: C      IAINF(2)      = LAST
308: C
309: C      return
310: C      end
311: C
312: C=====
313: C STLAST : SET LAST POINTER to LX
314: C=====
315: C
316: C      ( used to discard temporary data . etc. )
317: C
318: C      subroutine STLAST( VNAME, LX,      LAST )
319: C
320: C      include 'INC/_ARRAY'
321: C      include 'INC/_IOUNIT'
322: C
323: C      character VNAME*(*)
324: C-----
325: C

```

```

326: C      if ( LX.lt.IAINF(1) .or. LSIZ(LX).gt.LIMIT ) then
327: C          write(IMG,('(1x,a,a)') 'XXX TRIED TO RESET <LAST> INDEX OF ',
328: C          & ' VARIABLE MEMORY AREA IN /ARRAY/ WITH INVALID ADDRESS '
329: C          write(IMG,('(1x,a,a,a,i10)') ' Place <', VNAME(:ICLEN2(VNAME)),
330: C          & '>' index = ', LX
331: C          write(IMG,('(1x,a,i10,a,i10/)') ' valid index is from ',
332: C          & IAINF(1), ' to ', LIMIT
333: C          call PRSTOP( 1, 'ERROR IN DYNAMIC MEMORY HANDLING' )
334: C          stop 666
335: C      end if
336: C
337: C      IAINF(2)      = LX
338: C      LAST          = IAINF(2)
339: C      return
340: C      end
341: C
342: C=====
343: C GTLAST: GET LAST POINTER =====
344: C=====
345: C
346: C      subroutine GTLAST( LAST )
347: C
348: C      include 'INC/_ARRAY'
349: C
350: C-----
351: C      LAST          = IAINF(2)
352: C      return
353: C      end
354: C
355: C=====
356: C LSTCK: Check whether memory over error has occurred or not.
357: C
358: C      ex.
359: C      if ( LSTCK(0).lt.0 ) then
360: C          write(*,*) '== Memory over error has occurred.'
361: C      end if
362: C=====
363: C
364: C      function LSTCK( LJ )
365: C
366: C      include 'INC/_ARRAY'
367: C
368: C-----
369: C      LSTCK          = LIMIT + 1 - IAINF(3)
370: C      return
371: C      end

```

src/shared/kgpset.f

```
1:      subroutine KGPSET( KNGP, KENGP, NGP, JKPAR, KPLIM )
2: C=====
3: C Purpose: set arrays KNGP and KENGP to point energy dependent arrays
4: C         by particle species.
5: C
6: C         This should be called before procedure that may require
7: C         definition of energu dependent arrays of all particle species.
8: C
9: C called in: INTRO, INTRO2
10: C-----
11: C o KNGP(KPLIM+1) : KNGP(J) is < Sum of NGP(i),i=1,J-1> + 1.
12: C         It is also used to calculate energy group # for each
13: C         particle species (from 1 to NGP(particleID)) from
14: C         "unified" energy group # (from 1 to NGROUP).
15: C o KENGP(KPLIM+1) : KENGP(J) is < Sum of (NGP(i)+1),i=1,J-1> + 1.
16: C         It is used to point arrays on which data for each particle
17: C         species having length of <NGP(*)+1> (such as energy group
18: C         boundaries) are packed in the order of particle species ID.
19: C i NGP(KPLIM) : number of energy groups for each particle species.
20: C i JKPAR(KPLIM) : flag for each particle species.
21: C i KPLIM : number of possible particle species.
22: C=====
23:      include 'INC/_IOUNIT'
24: C
25:      integer KNGP(KPLIM+1)
26:      integer KENGP(KPLIM+1)
27: C
28:      integer NGP(KPLIM)
29:      integer JKPAR(KPLIM)
30: C-----
31: C
32: C ... check bin(group) numbers of each particles
33: C
34:      KNGP(1) = 1
35:      KENGP(1) = 1
36:      do 100 IK=1,KPLIM
37:         if ( JKPAR(IK).ne.0 ) then
38:            KNGP(IK+1) = KNGP(IK) + NGP(IK)
39:            KENGP(IK+1) = KENGP(IK) + NGP(IK) + 1
40:         else
41:            KNGP(IK+1) = KNGP(IK)
42:            KENGP(IK+1) = KENGP(IK)
43:         end if
44:      100 continue
45:      return
46:      end
```


src/shared/kpsort.f

```
1:      subroutine KPSORT( KKP,  NBANK, KPLIM, IBP,  NN, KS,  
2:      &                  KSORT, IKSORT,IWK )  
3: C=====   
4: C Purpose: sort index array IBP(1:NN) by KKP(IBP(1:N)).  
5: C          Value of KKP(*) must be from 1 to KPLIM.  
6: C   
7: C-----   
8: C arguments (i=input,o=output,w=work)  
9: C   
10: C i  KKP(NBANK) : a bank array variable(typically it includes particle  
11: C                  species ID).  
12: C i  KPLIM : value range of KKP(*) must be from 1 to KPLIM.  
13: C i  NBANK : bank array size  
14: C io IBP(NN) : index pointer to array KKP and it is sorted by KKP.  
15: C i  NN : array size of IBP  
16: C o  KS : number of different values in KKP(IBP(1:N)).  
17: C o  KSORT(KPLIM) : KSORT(1:KS) has KKP values found in KKP(IBP(1:N)).  
18: C o  IKSORT(KPLIM+1): element KKP(IBP(IKSORT(i):IKSORT(i+1)-1)) has  
19: C          value KSORT(i).  
20: C w  IWK(NN) : working array.  
21: C-----   
22:      integer KKP(NBANK)  
23:      integer IBP(NN)  
24:      integer KSORT(KPLIM), IKSORT(KPLIM)  
25:   
26:      integer IWK(NN)  
27:   
28: C-----   
29:      KS      = 0  
30: C   
31:      KK      = 0  
32:      IKSORT(1) = 1  
33:      do 110 IK = 1, KPLIM  
34:          KK2   = KK  
35: *VOCL LOOP,NOVREC  
36:          do 100 I = 1, NN  
37:              if ( KKP(IBP(I)).eq.IK ) then  
38:                  KK2   = KK2 + 1  
39:                  IWK(KK2) = IBP(I)  
40:              end if  
41: 100      continue  
42:          if ( KK2.gt.KK ) then  
43:              KS      = KS + 1  
44:              KSORT(KS) = IK  
45:              IKSORT(KS+1) = KK2+1  
46:          end if  
47:          if ( KK2.ge.NN ) go to 120  
48:          KK      = KK2  
49: 110 continue  
50: 120 continue  
51: C   
52:      do 130 I = 1, NN  
53:          IBP(I) = IWK(I)  
54: 130 continue  
55: C   
56:      return  
57:      end
```

src/shared/kzrstr.f

```

1:      subroutine KZRSTR( JDEBG, NZONE, KZDA,  NZDA,  NBODY, NSURF,
2:      &                  KINPZ, KZAA,
3:      &                  NKZAA, IBSDA, IPSDA,  KSFBD,
4:      &                  KBZL, IBZL, NBZL )
5: C=<MVP/GMVP>=====
6: C  PURPOSE: Restore KINPZ & KZAA array from working file on unit 'IUB'.
7: C          And debug print of KZDA, KINPZ & KZAA when required.
8: C          Also construct ksfbd(nzda) array which is body ID's
9: C          corresponding to KZDA data.
10: C  CALLED IN: zonein
11: C-----
12: C arguments (i=input, o=output, w=work)
13: C
14: C i JDEBG : debug print flag
15: C i NZONE : nuber of zones
16: C i KZDA(2,NZDA) : zone/surface table.
17: C i NBODY : number of bodies.
18: C i NSURF : number of surfaces.
19: C o KINPZ(NZONE) : input zone # of each zone.
20: C o KZAA(NZONE+1) : pointer to KZDA array for each zone.
21: C o NKZAA(NZONE) : number of elements in KZDA array for each zone.
22: C i IBSDA(3,NBODY) : first surface # and body ID for each body.
23: C i IPSDA(3,NSURF+1): pointer to SDA array & body ID/# for each surface.
24: C o KSFBD(NZDA) : body ID corresponding surfaces in KDZA array.
25: C o KBZL(NZBL) : list of body#*sign to define zones.
26: C o IBZL(NZONE+1) : pointers to KBZL(*) for each zone
27: C=====
28:      integer KZDA(2,NZDA), KINPZ(NZONE), KZAA(NZONE+1), NKZAA(NZONE)
29: C
30:      integer IBSDA(3,NBODY), KSFBD(NZDA)
31:      integer IPSDA(3,NSURF+1)
32:      integer KBZL(NBZL), IBZL(NZONE+1)
33: C
34:      integer JDEBG(*)
35: C
36:      include 'INC/_IOUNIT'
37: C
38: C-----
39: C
40:      call RWIND( IUB )
41: C
42:      IBZL(1) = 1
43:      do 100 I = 1, NZONE
44:          read(IUB) KZAA(I), KINPZ(I)
45:          read(IUB) IZB
46:          read(IUB) (KBZL(J),J=IBZL(I),IBZL(I)+IZB-1)
47:          IBZL(I+1) = IBZL(I) + IZB
48:      100 continue
49:      KZAA(NZONE+1) = NZDA + 1
50:      do 110 I = 1, NZONE
51:          NKZAA(I) = KZAA(I+1) - KZAA(I)
52:      110 continue
53: C-----
54: C ... KSFBD : body ID's corresponding to KZDA data.
55: C          used to indicate body for lost particles etc.
56: C-----
57:      do 140 I = 1, NZDA
58:          ISDA = ABS(KZDA(1,I))
59:          do 120 K = 1, NSURF
60:              if ( ISDA.eq.IPSDA(1,K) ) then
61:                  KSFBD(I) = IPSDA(2,K)
62:                  goto 140
63:              end if
64:          120 continue
65: C

```

```

66: C          do 120 K = 1, NBODY
67: C              if ( ISDA.lt.IBSDA(1,K) ) go to 130
68: C 120          continue
69: C          K = NBODY + 1
70: C 130          KSFBD(I) = IBSDA(2,K-1)
71: C
72:      140 continue
73: C
74: C-----
75: C ..... DEBUG PRINT .....
76: C-----
77: C
78:      if ( JDEBG(1).ne.0 ) then
79:          write(IOG,7000) ' KZDA ARRAY', KZDA
80:          write(IOG,7000) ' KINPZ ARRAY', KINPZ
81:          write(IOG,7000) ' KZAA ARRAY', KZAA
82:          write(IOG,7000) ' NKZAA ARRAY', KZAA
83:          write(IOG,7000) ' KSFBD ARRAY', KSFBD
84:      7000      format(/1X,A/(1X,10I8))
85:      end if
86: C
87:      return
88:      end

```

src/shared/latdw2.f

```

1:      subroutine LATDW2(IOW, JDEBG, JHLAT, JFISX,
2:      N NBANK ,NZONE ,NLATT ,NCELL ,NLBZ ,NEST ,
3:      N DINF ,DEPS ,
4:      S LSK ,NN ,JSTG ,NN2 ,
5:      I LEVL0 ,LZZ0 ,LPOS0 ,LCRS0 ,XYZ0 ,
6:      B AAA ,BBB ,CCC ,
7:      B LEVL ,LZZ ,LPOS ,LCRS ,
8:      B DBNK ,KDBNK ,MDBNK ,IBNK ,KIBNK ,MIBNK,
9:      S LSEDED ,NDEAD ,
10:     G SDA ,KZDA ,KZAA ,
11:     G KZMAT ,KCELL ,KZLBZ ,MLBZZ ,CELSZ ,SZLAT ,
12:     G IDLAT ,IPLAT ,KLATT ,KSLAT ,NVLAT ,IPCEL ,
13:     G LTYP ,ICTYP ,KDALT ,DALT ,KZREG ,
14:     W X ,Y ,Z ,AI ,BI ,CI ,
15:     W DI ,DLOC1 ,DLOC2 ,IKL ,IKL2 )
16: C=====
17: C PURPOSE: calculate flight direction and distance to frame boundary
18: C by using common nest structure.
19: C
20: C      (AAA(LSK(I)),BBB(LSK(I)),CCC(LSK(I)))
21: C      DBNK(LSK(I),KDBNK(4)),DBNK(LSK(I),KDBNK(6))
22: C      IBNK(LSK(I),KIBNK(6))
23: C      And set
24: C      LEVL(LSK(I))
25: C      LZZ(LSK(I)),LPOS(LSK(I)),LCRS(LSK(I))
26: C
27: C <<Comment>>
28: C
29: C CALLED IN:      CALLS: (NONE)
30: C UPDATE:
31: C 14 Mar 2007: clear IBNK data for KIBNK(17:19), for photonuclear.
32: C      (K.Kosako)
33: C-----
34: C arguments ( i=input o=output w=work )
35: C
36: C i NBANK: size of the first dimension of arrays
37: C i NN : number of particles to be processed
38: C io LSK(NN) : stack ( bank pointer )
39: C      remove LOST particles.
40: C o NN2 : number of remaining particles.
41: C o JSTG : flag to indicate this region is STG region.
42: C io LSEDED : dead particle stack.
43: C io NDEAD : number of dead particles.
44: C i LEVL0,LZZ0(NEST),LPOS0(NEST),LCRS0 : lattice data.
45: C i XYZ0(3,0:NEST) : cell coordinates in each level.
46: C io AAA(NN0),BBB(NN0),CCC(NN0) : direction
47: C o LEVL,LZZ,LPOS,LCRS : LATTICE PARAMETER BANK
48: C o DBNK : optional bank data.
49: C o IBNK : optional bank data.
50: C=====
51: C      implicit real*8(D)
52: C
53: C      integer JDEBG(*)
54: C
55: C .... NEST DATA .....
56: C
57: C      integer LZZ0(NEST), LPOS0(NEST),LCRS0(NEST)
58: C      real*8 XYZ0(3,0:NEST)
59: C
60: C .... PARTICLE BANK .....
61: C
62: C      real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
63: C      integer LEVL(NBANK), LZZ(NBANK,NEST),
64: C      & LPOS(NBANK,NEST), LCRS(NBANK,NEST)
65: C      integer KDBNK(0:MDBNK), KIBNK(0:MIBNK)

```

```

66:      real*8 DBNK(NBANK,*)
67:      integer IBNK(NBANK,*)
68: C
69: C .... STACKS .....
70: C
71: C      integer LSK(NN)
72: C      integer LSEDED(NBANK)
73: C
74: C .... ZONE GEOMETRY DATA
75: C
76: C      real*8 SDA(*)
77: C      integer KZDA(2,*), KZAA(*)
78: C      real*8 DINF, DEPS
79: C
80: C .... GEOMETRY ARRAY DATA
81: C
82: C      integer KZMAT(NZONE), KZLBZ(NLBZ), MLBZZ(NZONE), KCELL(NZONE),
83: C      & IDLAT(NLATT), IPLAT(NLATT+1), KLATT(*), KSLAT(*),
84: C      & NVLAT(4,NLATT), IPCEL(NCELL), LTYP(NLATT), ICTYP(NCELL),
85: C      & KDALT(NLBZ)
86: C      real*8 CELSZ(6,NCELL), SZLAT(8,NLATT), DALT(*)
87: C
88: C      integer KZREG(NZONE)
89: C
90: C .... WORKING ARRAYS .....
91: C
92: C      real*8 X(NN), Y(NN), Z(NN)
93: C      real*8 AI(NN), BI(NN), CI(NN)
94: C      ... necessary when JFISX .ne.0 (and JSTGR.ne.0)
95: C      real*8 DI(NN), DLOC1(NN), DLOC2(NN)
96: C      integer IKL(NN), IKL2(NN)
97: C
98: C .... ROTATION VECTOR: ROT(N,1) = COS(-PAI/3 *N)
99: C      ROT(N,2) = SIN(-PAI/3 *N)
100: C      common /HEXROT/ DROT
101: C      real*8 DROT(0:5,2)
102: C
103: C      parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
104: C      parameter( DPAI = 2.0D0*3.141592653589793238D0 )
105: C
106: C      include 'INC/_PMLATT'
107: C      include 'INC/_SFLATT'
108: C
109: C check
110: C      write(6,*) ' latde2: lev10,lzz0=',lev10,lzz0
111: C      write(6,*) ' nest=',nest,' nn=',nn
112: C      write(6,*) ' latde2: lev10,lpos0=',lev10,lpos0
113: C      write(6,*) ' latde2: lev10,lcrs0=',lev10,lcrs0
114: C      write(6,*) ' xyz0:', xyz0
115: C      write(6,*) ' lcrs=',lcrs
116: C
117: C
118: C-----
119: C ..... Gather positions and directions .....
120: C-----
121: C
122: C      NN2 = NN
123: C      JSTG = 0
124: C
125: C      if ( LEVL0.gt.0 ) then
126: C      do 100 I = 1, NN2
127: C          AI(I) = AAA(LSK(I))
128: C          BI(I) = BBB(LSK(I))
129: C          CI(I) = CCC(LSK(I))
130: C      100 continue

```

src/shared/latdw2.f

```

131:      end if
132: C
133: C
134:      do 200 LV = 1, LEVL0
135: C
136: C      ... MZ : ZONE # ... MLT: LATTICE # ...
137: C      ... JKSF : LATTICE Buffer Zone #
138: C      ... IP : Position in lattice ... KHLAT : frame boundary
139: C
140:      MZ      = LZZ0(LV)
141: CCCC MLT      = ABS(KZMAT(MZ))
142:      MLT      = LATNM(KZMAT(MZ))
143:      JKSF      = MLBZZ(MZ)
144:      IP        = LPOS0(LV)
145: check
146: C      write(6,*) ' lv,mz,mlt,jksf,ip=',lv,mz,mlt,jksf,ip
147: C
148: C      --- This is a STG region. ---
149: C      if ( IP.lt.0 ) then
150: C          JSTG = 1
151: C          LVL = LV - 1
152: C          go to 400
153: C      end if
154: C      if ( JHLAT.ne.0 ) KHLAT = LCRS0(LV)
155: C
156: C-----
157: C ..... Calculate distance to frame boundary for particles entering
158: C lattice geometry region for the case allowing
159: C overlapping of cells or frames.
160: C (typically cases using STG particle region)
161: C-----
162: C
163: C      if ( JFISX.ne.0 ) then
164: C          do 210 I = 1, NN2
165: C              IKL(I) = 0
166: C              DI(I) = DINF
167: C              X(I) = XYZ0(1,LV-1)
168: C              Y(I) = XYZ0(2,LV-1)
169: C              Z(I) = XYZ0(3,LV-1)
170: C          210 continue
171: C              JSS = 0
172: C              call SPEAR1( MZ, KZDA, KZAA, SDA, NN2, NBANK, JSS,
173: C              &          X, Y, Z, AI, BI, CI, DI,
174: C              &          DLOC1, DLOC2, IKL, IKL2,
175: C              &          DUMMY1, DINF, DEPS )
176: C
177: C          .... check lost particle: NXLT might be changed
178: C
179: C          IIL = 0
180: C          do 220 I = 1, NN2
181: C              if ( DI(I).eq.DINF ) IIL = IIL + 1
182: C          220 continue
183: C
184: C          if ( IIL.gt.0 ) then
185: C              C/#IF PARA(SX* CRAY)
186: C              * call MVPSYNC_LOCK(2)
187: C              C/#ENDIF
188: C              write(IOW,'(1x,a,i5,2a/1x,3(a,i5))')
189: C              &          '(LATDW2) ', IIL,
190: C              &          ' particles are lost in calculation of distance to ',
191: C              &          'lattice frame.', ' Frame zone# ', MZ,
192: C              &          ' Lattice # ', MLT, ' lattice ID ', IDLAT(MLT)
193: C              II = 0
194: C              do 230 I = 1, NN2
195: C                  if ( DI(I).eq.DINF ) then

```

```

196:      LL      = LSK(I)
197:      write(IOW,7000) X(I), Y(I), Z(I),
198:      &          AI(I), BI(I), CI(I), LL
199:      II      = II + 1
200:      LSDED(NDEAD+II) = LSK(I)
201: C##<2007/03/14:PN3:
202:      if ( KIBNK(17).ne.0 ) IBNK(LSK(I),KIBNK(17)) = 0
203:      if ( KIBNK(18).ne.0 ) IBNK(LSK(I),KIBNK(18)) = 0
204:      if ( KIBNK(19).ne.0 ) IBNK(LSK(I),KIBNK(19)) = 0
205: C##>
206:      end if
207:      230 continue
208: C/#IF PARA(SX* CRAY)
209: *      call MVPSYNC_UNLOCK(2)
210: C/#ENDIF
211:      NDEAD = NDEAD + II
212:      NLOST = NLOST + II
213:      JJ = 0
214: *VOCL LOOP,NOVREC
215:      do 240 I = 1, NN2
216:          if ( DI(I).ne.DINF ) then
217: C              JJ = JJ + 1
218: C              AI(JJ) = AI(I)
219: C              BI(JJ) = BI(I)
220: C              CI(JJ) = CI(I)
221: C              X(JJ) = X(I)
222: C              Y(JJ) = Y(I)
223: C              Z(JJ) = Z(I)
224: C              DI(JJ) = DI(I)
225: C              LSK(JJ) = LSK(I)
226:          end if
227:          240 continue
228:          NN2 = JJ
229:      end if
230: C
231: C      ... all particles have the same lattice level ...
232: C
233: C      KD4 = KDBNK(4)
234: C      KD41 = KDBNK(4) - 1
235: C      KI6 = KIBNK(6)
236: C      KD6 = KDBNK(6)
237: C      KK1 = KD4 + LV - 1
238: C      KK6 = KD6 + 6*(LV-1)
239: C
240: C      ... store distance to frame, position to be reached
241: C      and direction in level LV ...
242: C
243: *VOCL LOOP,NOVREC
244:      do 250 I = 1, NN2
245: C          DBNK(LSK(I),KK1) = DI(I)
246: C          DBNK(LSK(I),KK6) = X(I) + AI(I)*DI(I)
247: C          DBNK(LSK(I),KK6+1) = Y(I) + BI(I)*DI(I)
248: C          DBNK(LSK(I),KK6+2) = Z(I) + CI(I)*DI(I)
249: C          DBNK(LSK(I),KK6+3) = AI(I)
250: C          DBNK(LSK(I),KK6+4) = BI(I)
251: C          DBNK(LSK(I),KK6+5) = CI(I)
252:      250 continue
253: C
254: C      ... set level # giving smallest frame distance in the
255: C      next level ...
256: C
257:      if ( LV.eq.1 ) then
258: C          do 260 I = 1, NN2
259: C              IBNK(LSK(I),KI6) = 1
260:      260 continue

```

src/shared/latdw2.f

```

261:      else
262:      *VOCL LOOP,NOVREC
263:      do 270 I = 1, NN2
264:      if ( DBNK(LSK(I),KD41+IBNK(LSK(I),KI6+LV-2))
265:      &      .lt.DI(I) ) then
266:      IBNK(LSK(I),KI6+LV-1) = IBNK(LSK(I),KI6+LV-2)
267:      else
268:      IBNK(LSK(I),KI6+LV-1) = LV
269:      end if
270:      continue
271:      end if
272:      end if
273: C
274: 7000 format(1X,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',E12.5,
275: &      ' ETA=',E12.5,' XI=',E12.5,' NO=',I6)
276: C
277: C
278: C-----
279: C      ... M : POINTER TO DALT ARRAY
280: C-----
281: C      M      = KDALT(JKSF)
282: C
283: C-----
284: C      RECTANGULAR LATTICE (LTYP(MLT) = 1)
285: C-----
286: C
287: C      if ( LTYP(MLT).eq.1 ) then
288: C
289: C-----
290: C      ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
291: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
292: C-----
293: C
294: C      ICEL      = KLATT(IPLAT(MLT)+IP)
295: C      ISS       = KSLAT(IPLAT(MLT)+IP)
296: C      IDRX      = ISS/100
297: C      IDRY      = (ISS-IDRX*100) /10
298: C      IDRZ      = MOD(ISS,10)
299: C      JSX       = 1 - 2*IDRX
300: C      JSY       = 1 - 2*IDRY
301: C      JSZ       = 1 - 2*IDRZ
302: C
303: *VOCL LOOP,NOVREC
304:      do 280 I = 1, NN2
305: C
306: C-----
307: C      ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
308: C      ( ASSUMING THE BODY OF LATTICE ZONE IS 'RPP' OR 'BOX' )
309: C-----
310: C
311: C      DA      = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*
312: C      &      CI(I)
313: C      DB      = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) +
314: C      &      DALT(M+5)*CI(I)
315: C      DC      = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) +
316: C      &      DALT(M+8)*CI(I)
317: *VOCL STMT,IF(100)
318:      if ( ICEL.ne.0 ) then
319: C
320: C      LZZ(LSK(I),LV) = MZ
321: C      LPOS(LSK(I),LV) = IP
322: C      if ( JHLAT.ne.0 ) LCRS(LSK(I),LV) = 0
323: C
324: C-----
325: C      ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)

```

```

326: C-----
327: C      AAA(LSK(I)) = JSX*DA
328: C      BBB(LSK(I)) = JSY*DB
329: C      CCC(LSK(I)) = JSZ*DC
330: C      AI(I) = JSX*DA
331: C      BI(I) = JSY*DB
332: C      CI(I) = JSZ*DC
333: C
334: C-----
335: C      ..... CELL # = 0 : OUT OF LATTICE DEFINITION
336: C-----
337: C      else
338: C      IPC      = MZ
339: C      end if
340: C
341: 280      continue
342: C
343: C-----
344: C      HEXAGONAL LATTICE ( LTYP(MLT) = 2,3,4 )
345: C-----
346: C
347: C      else if ( LTYP(MLT).ge.2.and.LTYP(MLT).le.4 ) then
348: C
349: C-----
350: C      ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
351: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
352: C-----
353: C
354: C      IC      = IPLAT(MLT) + IP
355: C      ICEL     = KLATT(IC)
356: C      ISS      = KSLAT(IC)
357: C      IDRX     = ISS/100
358: C      JSX      = 1 - 2*IDRX
359: C      IDRY     = (ISS-IDRX*100) /10
360: C      JSZ      = 1 - 2*MOD(ISS,10)
361: C
362: *VOCL LOOP,NOVREC
363:      do 290 I = 1, NN2
364: C
365: C-----
366: C      ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
367: C      ( ASSUMING THE BODY OF LATTICE ZONE IS 'RHP' OR 'HEX' )
368: C-----
369: C
370: C      DA      = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*
371: C      &      CI(I)
372: C      DB      = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) +
373: C      &      DALT(M+5)*CI(I)
374: C      DC      = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) +
375: C      &      DALT(M+8)*CI(I)
376: C
377: *VOCL STMT,IF(100)
378:      if ( ICEL.ne.0 ) then
379: C
380: C-----
381: C      ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL BUFFER-ZONE
382: C-----
383: C      LZZ(LSK(I),LV) = MZ
384: C      LPOS(LSK(I),LV) = IP
385: C      LCRS(LSK(I),LV) = KHLAT
386: C
387: C-----
388: C      ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
389: C-----
390: C

```

src/shared/latdw2.f

```
391:          DA      = JSX*DA
392: C          AAA(LSK(I)) = DROT(IDRY,1)*DA - DROT(IDRY,2)*DB
393: C          BBB(LSK(I)) = DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
394: C          CCC(LSK(I)) = JSZ*DC
395:          AI(I) = DROT(IDRY,1)*DA - DROT(IDRY,2)*DB
396:          BI(I) = DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
397:          CI(I) = JSZ*DC
398: C
399: C-----
400: C          ..... CELL # = 0 : OUT OF LATTICE DEFINITION
401: C-----
402: C          else
403: C             IPC      = MZ
404: C          end if
405: C
406: C          290 continue
407: C
408: C-----
409: C          STG region (LTYP(MLT) = 10)
410: C-----
411: C
412: C          else if ( LTYP(MLT).eq.10 ) then
413: C             LVL      = LV - 1
414: C             JSTG      = 1
415: C             go to 400
416: C          end if
417: C          200 continue
418: C
419: C
420: C          400 continue
421: C
422: C          ... JSTG=0 : not in STG REGION
423: C
424: C          if ( JSTG.eq.0 ) LVL = LEVL0
425: C
426: C          do 410 I = 1, NN2
427: C             LEVL(LSK(I)) = LVL
428: C          410 continue
429: C
430: C          if ( LEVL0.gt.0 ) then
431: C             do 420 I = 1, NN2
432: C                AAA(LSK(I)) = AI(I)
433: C                BBB(LSK(I)) = BI(I)
434: C                CCC(LSK(I)) = CI(I)
435: C             420 continue
436: C          end if
437: C
438: C          return
439: C          end
```

src/shared/latdwn.f

```

1:      subroutine LATDWN(IOW, JDEBG ,JVMNT,JTLLT,JHLAT,
2:      N NBANK,NZONE,NLATT,NCELL,NLBZ,NEST,NSPACE,NUNV,DEPS,
3:      B XXX ,YYY ,ZZZ ,JTRDIR,AAA ,BBB ,CCC ,
4:      B IZZ ,LEVL ,LZZ ,LPOS ,LCRS,IBREG,IBSPC,
5:      S LSSRC ,IZNXT ,NNXT ,LSLAT ,IZLAT ,NXLT ,
6:      S LSEND ,NEND ,
7:      G KZMAT ,KCELL ,KZLBZ ,MLBZZ ,CELSZ ,SZLAT ,
8:      G IDLAT ,IPLAT ,KLATT ,KSLAT ,NVLAT ,IPCEL ,
9:      G LTYP ,ICTYP ,KDALT ,DALT ,KZREG,KSPSU,KTCSP,NKTCSP,KHLAT,
10:     W X ,Y ,Z ,AI ,BI ,CI ,
11:     W IWK ,JKSF ,IPZONE )
12: C=====
13: C PURPOSE: To find lattice cells which particles in lattice-buffer-zone
14: C enter but does not treat cell --> cell movement.
15: C (only "downward" for lattice geometry level)
16: C
17: C <<Comment>>
18: C Particle bank variables such as XXX,YYY,ZZZ,AAA,... may not be
19: C the globally accessed "real" particle bank, but some kind of a
20: C "temporary" bank used within "underground" modules such as ACTA2Z.
21: C And the size 'NBANK' may be different from the global one defined in
22: C the common area /SIZES/.
23: C
24: C <<Comment>>
25: C STG-region (lattice) is not treated in this routine.
26: C The particle in STG region is moved into end particle stack.
27: C
28: C CALLED IN: ACTA2Z CALLS: (NONE)
29: C-----
30: C === INTER-STACK DATA FLOW ===
31: C
32: C LATTICE-SEARCH STACK -----> NEXT-ZONE-SEARCH STACK
33: C (LSLAT,IZLAT,NXLT) | (LSSRC,IZNXT,NNXT)
34: C -----> END-PARTICLE STACK
35: C (LSEND)
36: C
37: C === BANK DATA TO BE UPDATED ===
38: C
39: C (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #
40: C LEVL,LZZ,LPOS,LCRS : LATTICE PARAMETER BANK
41: C
42: C=====
43: C implicit real*8(D)
44: C
45: C integer JDEBG(*)
46: C
47: C .... PARTICLE BANK .....
48: C
49: C real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
50: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
51: C integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
52: C & LPOS(NBANK,NEST), LCRS(NBANK,NEST)
53: C integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
54: C
55: C .... STACKS .....
56: C
57: C integer LSSRC(NBANK), IZNXT(NBANK), NNXT(NZONE+1), LSLAT(NBANK),
58: C & IZLAT(NBANK), NXLT(NLBZ+1), LSEND(NBANK)
59: C
60: C .... GEOMETRY ARRAY DATA
61: C
62: C integer KZMAT(NZONE), KZLBZ(NLBZ), MLBZZ(NZONE), KCELL(NZONE),
63: C & IDLAT(NLATT), IPLAT(NLATT+1), KLATT(*), KSLAT(*),
64: C & NVLAT(4,NLATT), IPCEL(NCELL), LTYP(NLATT), ICTYP(NCELL),
65: C & KDALT(NLBZ), KHLAT(*)

```

```

66:      real*8 CELSZ(6,NCELL), SZLAT(8,NLATT), DALT(*)
67: C
68: C integer KZREG(NZONE)
69: C integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
70: C
71: C .... WORKING ARRAYS .....
72: C
73: C integer IWK(NBANK), JKSF(NZONE+1), IPZONE(NZONE+1)
74: C real*8 X(NBANK), Y(NBANK), Z(NBANK)
75: C real*8 AI(NBANK), BI(NBANK), CI(NBANK)
76: C
77: C .... ROTATION VECTOR: ROT(N,1) = COS(-PAI/3 *N)
78: C ROT(N,2) = SIN(-PAI/3 *N)
79: C common /HEXROT/ DROT
80: C real*8 DROT(0:5,2)
81: C Ccccc DATA DROT / 1.0D0 , 0.5D0 , -0.5D0 , -1.0D0 , -0.5D0 , 0.5D0,
82: C Ccccc@ 0.0, -0.866025403784438652D0, -0.866025403784438652D0, 0.0,
83: C Ccccc@ 0.866025403784438652D0, 0.866025403784438652D0 /
84: C
85: C parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
86: C
87: C include 'INC/_PMLATT'
88: C include 'INC/_SFLATT'
89: C
90: C-----
91: C
92: C if ( JVMNT.ne.0 ) call VMNTRI( 7, NXLT(NLBZ+1) )
93: C
94: C .... REORDER PARTICLES BY LATTICE-BUFFER-ZONE (LB-ZONE) NUMBERS
95: C
96: C NZK = 0
97: C do 100 K = 1, NLBZ
98: C if ( NXLT(K).ne.0 ) then
99: C NZK = NZK + 1
100: C JKSF(NZK) = K
101: C end if
102: C 100 continue
103: C
104: C IPZONE(1) = 1
105: C do 110 NK = 1, NZK
106: C IPZONE(NK+1) = IPZONE(NK) + NXLT(JKSF(NK))
107: C 110 continue
108: C KKK = 0
109: C do 130 NK = 1, NZK
110: C do 120 I = 1, NXLT(NLBZ+1)
111: C if ( IZLAT(I).eq.JKSF(NK) ) then
112: C KKK = KKK + 1
113: C IWK(KKK) = LSLAT(I)
114: C end if
115: C 120 continue
116: C 130 continue
117: C
118: C-----
119: C ..... GATHER POSITIONS .....
120: C-----
121: C
122: C do 140 I = 1, NXLT(NLBZ+1)
123: C X(I) = XXX(IWK(I))
124: C Y(I) = YYY(IWK(I))
125: C Z(I) = ZZZ(IWK(I))
126: C 140 continue
127: C
128: C if ( JTRDIR.ne.0 ) then
129: C do 150 I = 1, NXLT(NLBZ+1)
130: C AI(I) = AAA(IWK(I))

```

src/shared/latdwn.f

```

131:      BI(I) = BBB(IWK(I))
132:      CI(I) = CCC(IWK(I))
133: 150 continue
134: end if
135: C
136: do 210 KZ = 1, NZK
137: C
138:      ... MZ : ZONE # .... MLT: LATTICE # ....
139: C
140:      MZ = KZLBZ(JKSF(KZ))
141: CCC MLT = ABS(KZMAT(MZ))
142: C =====
143: C ..... PARTICLES ARE ENTERING A LATTICE .....
144: C =====
145: CCCC if ( KZMAT(MZ).le.-1.and.KZMAT(MZ).ge.-998 ) then
146:      if ( KSLATT(KZMAT(MZ)) ) then
147:          MLT = LATTNM(KZMAT(MZ))
148:          DSX = SZLAT(1,MLT)
149:          DSY = SZLAT(2,MLT)
150:          DSZ = SZLAT(3,MLT)
151: C -----
152: C      ... M : POINTER TO DALT ARRAY
153: C -----
154:      M = KDALT(JKSF(KZ))
155:      INN = NNXT(NZONE+1) - IPZONE(KZ) + 1
156:      DA = 1.0D0
157:      DB = 0.0D0
158:      DC = 0.0D0
159: C
160: C -----
161: C      RECTANGULAR LATTICE (LTYP(MLT) = 1)
162: C -----
163: C
164:      if ( LTYP(MLT).eq.1 ) then
165: C
166: *VOCL LOOP,NOVREC
167: do 160 I = IPZONE(KZ), IPZONE(KZ+1) - 1
168: C
169: C -----
170: C ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
171: C ( ASSUMING THE BODY OF LATTICE ZONE IS 'RPP' OR 'BOX' )
172: C -----
173: C
174:      DX = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*
175:      & Z(I) - DALT(M+9)
176:      DY = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*
177:      & Z(I) - DALT(M+10)
178:      DZ = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*
179:      & Z(I) - DALT(M+11)
180:      if ( JTRDIR.ne.0 ) then
181:          DA = DALT(M)*AI(I) + DALT(M+1)*BI(I)
182:          & + DALT(M+2)*CI(I)
183:          DB = DALT(M+3)*AI(I) + DALT(M+4)*BI(I)
184:          & + DALT(M+5)*CI(I)
185:          DC = DALT(M+6)*AI(I) + DALT(M+7)*BI(I)
186:          & + DALT(M+8)*CI(I)
187:      end if
188: C
189: C -----
190: C      ....IP: POSITION IN THE LATTICE .....
191: C -----
192: C
193:      IX = MAX(0,MIN(NVLAT(1,MLT)-1,INT(DX/DSX)))
194:      IY = MAX(0,MIN(NVLAT(2,MLT)-1,INT(DY/DSY)))
195:      IZ = MAX(0,MIN(NVLAT(3,MLT)-1,INT(DZ/DSZ)))

```

```

196:      IP = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
197: C
198: C -----
199: C ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
200: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
201: C -----
202: C
203:      ICEL = KLATT(IPLAT(MLT)+IP)
204:      ISS = KSLAT(IPLAT(MLT)+IP)
205:      IDRX = ISS/100
206:      IDRY = (ISS-IDRX*100)/10
207:      IDRZ = MOD(ISS,10)
208:      JSX = 1 - 2*IDRX
209:      JSY = 1 - 2*IDRY
210:      JSZ = 1 - 2*IDRZ
211: *VOCL STMT,IF(100)
212:      if ( ICEL.ne.0 ) then
213: C
214: C -----
215: C ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL
216: C -----
217:      IPC = IPCEL(ICEL)
218:      IZZ(IWK(I)) = IPC
219:      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
220:      LZZ(IWK(I),LEVL(IWK(I))) = MZ
221:      LPOS(IWK(I),LEVL(IWK(I))) = IP
222:      if ( JHLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
223: C
224: C -----
225: C ..... ENTERING SUBSPACE # .....
226: C -----
227: C
228:      if ( JTLLT.gt.0 ) then
229:          IBSP = IBSPC(IWK(I),LEVL(IWK(I))-1)
230:          IBSPC(IWK(I),LEVL(IWK(I))) =
231:      & KTCSP(KSPSU(KZREG(MZ),IBSP)+IP)
232:      end if
233: C
234: C -----
235: C ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
236: C -----
237: C
238:      XXX(IWK(I)) = CELSZ(4,ICEL)
239:      & + JSX*(DX-(IX+0.5D0)*DSX)
240:      YYY(IWK(I)) = CELSZ(5,ICEL)
241:      & + JSY*(DY-(IY+0.5D0)*DSY)
242:      ZZZ(IWK(I)) = CELSZ(6,ICEL)
243:      & + JSZ*(DZ-(IZ+0.5D0)*DSZ)
244:      if ( JTRDIR.ne.0 ) then
245:          AAA(IWK(I)) = JSX*DA
246:          BBB(IWK(I)) = JSY*DB
247:          CCC(IWK(I)) = JSZ*DC
248:      end if
249: C
250: C -----
251: C ..... CELL # = 0 : OUT OF LATTICE DEFINITION
252: C -----
253:      else
254:          IPC = MZ
255:      end if
256: C
257: C -----
258: C ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
259: C -----
260:      LSSRC(I+INN) = IWK(I)

```


src/shared/latdwn.f

```

261:      IZNXT(I+INN)      = IPC
262:      Ccccccc      NNXT(IPC)      = NNXT(IPC) + 1
263:      160      continue
264:      *VOCL LOOP, SCALAR
265:      do 170 I = IPZONE(KZ), IPZONE(KZ+1) - 1
266:      NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
267:      170      continue
268:      NNXT(NZONE+1) = INN + IPZONE(KZ+1) - 1
269:      C
270:      if ( MOD(JDEBG(3),2).ne.0 ) then
271:      write(IOW,*) '(LATDWN) iznxt ',
272:      &      (IZNXT(INN+I), I=IPZONE(KZ), IPZONE(KZ+1)-1)
273:      end if
274:      C
275:      C
276:      C
277:      C      HEXAGONAL LATTICE (LTYP(MLT) = 2,3,4)
278:      C
279:      C
280:      else if ( LTYP(MLT).ge.2 .and. LTYP(MLT).le.4 ) then
281:      C
282:      *VOCL LOOP, NOVREC
283:      do 180 I = IPZONE(KZ), IPZONE(KZ+1) - 1
284:      C
285:      C
286:      C      ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
287:      C      ( ASSUMING THE BODY OF LATTICE ZONE IS 'RHP' OR 'HEX' )
288:      C
289:      C
290:      DX      = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*
291:      &      Z(I) - DALT(M+9)
292:      DY      = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*
293:      &      Z(I) - DALT(M+10)
294:      DZ      = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*
295:      &      Z(I) - DALT(M+11)
296:      if ( JTRDIR.ne.0 ) then
297:      DA      = DALT(M)*AI(I) + DALT(M+1)*BI(I)
298:      &      + DALT(M+2)*CI(I)
299:      DB      = DALT(M+3)*AI(I) + DALT(M+4)*BI(I)
300:      &      + DALT(M+5)*CI(I)
301:      DC      = DALT(M+6)*AI(I) + DALT(M+7)*BI(I)
302:      &      + DALT(M+8)*CI(I)
303:      end if
304:      C
305:      C
306:      C      ..... CELL-POSITION .....
307:      C
308:      Ccccccccc      DXX      = DX + DA*DEPS
309:      Ccccccccc      DYY      = DY + DB*DEPS
310:      Ccccccccc      DZZ      = DZ + DC*DEPS
311:      DXX      = DX
312:      DYY      = DY
313:      DZZ      = DZ
314:      N2      = 2*NVLAT(1,MLT)
315:      IN1      = INT(2.0*DXX/DSX+1.0+N2) - N2
316:      IN2      = INT((DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
317:      IN3      = INT((-DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
318:      IX      = (IN1-IN3+1) /3
319:      IY      = (IN2+IN3) /3
320:      C
321:      IZ      = MAX(0, MIN(NVLAT(3,MLT)-1, INT(DZZ/DSZ)))
322:      C
323:      IP      = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
324:      C
325:      C

```

```

326:      C      ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
327:      C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
328:      C
329:      C
330:      IC      = IPLAT(MLT) + IP
331:      ICEL      = KLATT(IC)
332:      ISS      = KSLAT(IC)
333:      IDRX      = ISS/100
334:      JSX      = 1 - 2*IDRX
335:      IDRY      = (ISS-IDRX*100) /10
336:      JSZ      = 1 - 2*MOD(ISS,10)
337:      *VOCL STMT, IF(100)
338:      if ( ICEL.ne.0 ) then
339:      C
340:      C
341:      C      ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL BUFFER-ZONE
342:      C
343:      IPC      = IPCEL(ICEL)
344:      IZZ(IWK(I)) = IPC
345:      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
346:      LZZ(IWK(I),LEVL(IWK(I))) = MZ
347:      LPOS(IWK(I),LEVL(IWK(I))) = IP
348:      LCRS(IWK(I),LEVL(IWK(I))) = KHLAT(IPLAT(MLT)+IP)
349:      C
350:      C
351:      C      .... ENTERING SUBSPACE # ....
352:      C
353:      C
354:      if ( JTLLT.gt.0 ) then
355:      IBSP      = IBSPC(IWK(I),LEVL(IWK(I))-1)
356:      IBSPC(IWK(I),LEVL(IWK(I))) =
357:      &      KTCSP(KSPSU(KZREG(MZ),IBSP)+IP)
358:      end if
359:      C
360:      C
361:      C
362:      C      ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
363:      C
364:      C
365:      DX      = JSX*(DX-(IX+0.5*IY)*DSX)
366:      DY      = DY - IY*DSX*DHRT3
367:      XXX(IWK(I)) = DROT(IDRY,1)*DX - DROT(IDRY,2)*DY
368:      &      + CELSZ(3,ICEL)
369:      YYY(IWK(I)) = DROT(IDRY,2)*DX + DROT(IDRY,1)*DY
370:      &      + CELSZ(4,ICEL)
371:      ZZZ(IWK(I)) = JSZ*(DZ-(IZ+0.5D0)*DSZ)
372:      &      + CELSZ(5,ICEL)
373:      if ( JTRDIR.ne.0 ) then
374:      DA      = JSX*DA
375:      AAA(IWK(I)) = DROT(IDRY,1)*DA - DROT(IDRY,2)*DB
376:      BBB(IWK(I)) = DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
377:      CCC(IWK(I)) = JSZ*DC
378:      end if
379:      C
380:      C
381:      C      ..... CELL # = 0 : OUT OF LATTICE DEFINITION
382:      C
383:      else
384:      IPC      = MZ
385:      end if
386:      C
387:      C
388:      C      ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
389:      C
390:      LSSRC(I+INN) = IWK(I)

```

src/shared/latdwn.f

```
391:          IZNXT(I+INN)      = IPC
392: Cccccccccccccccccccc NNXT(IPC)      = NNXT(IPC) + 1
393: C
394: 180          continue
395: C
396: *VOCL LOOP, SCALAR
397:          do 190 I = IPZONE(KZ), IPZONE(KZ+1) - 1
398:            NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
399: 190          continue
400:            NNXT(NZONE+1)      = INN + IPZONE(KZ+1) - 1
401: C
402: C
403: C -----
404: C      STG region (LTYP(MLT) = 10)
405: C -----
406: C
407:      else if ( LTYP(MLT).eq.10 ) then
408: C
409:          IZZ(IWK(I)) = 0
410:          LEVL(IWK(I)) = LEVL(IWK(I)) + 1
411:          LZZ(IWK(I),LEVL(IWK(I))) = MZ
412:          LPOS(IWK(I),LEVL(IWK(I))) = -1
413:          if ( JHLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
414: C
415:          do 200 I = IPZONE(KZ), IPZONE(KZ+1) - 1
416:            NEND = NEND + 1
417:            LSEND(NEND) = IWK(I)
418: 200          continue
419: C
420:          end if
421: C
422:          end if
423: C
424: 210 continue
425: C
426: C -----
427: C      ..... CLEAR NXLT
428: C -----
429: C
430:          do 220 I = 1, NLBZ + 1
431:            NXLT(I) = 0
432: 220          continue
433: C
434:          return
435:          end
```

src/shared/latice.f

```

1:      subroutine LATICE( JDEBG ,JVMNT,JTLLT,JHLAT, JFISX, NLOST,
2:      N NBANK,NZONE,NREG,NLATT,NCELL,NLBZ,NEST,NSUZON,NSPACE,NUNV,
3:      N DINF,DEPS,IRAND,NMKREG,
4:      B XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
5:      B IZZ ,LEVL ,LZZ ,LPOS ,LSDED ,NDEAD,LCRS,KLSF,IBREG,IBSPC,
6:      B DBNK,KDBNK,MDBNK,IBNK,KIBNK,MIBNK,
7:      S LSSRC ,IZNXT ,NNXT ,LSLAT ,IZLAT ,NXLT , LSFFL, NFFL, IZFFL,
8:      G SDA, KZDA, KZAA,
9:      G KZMAT ,KCELL ,KZLBZ ,MLBZZ ,CELSZ ,SZLAT ,
10:     G IDLAT ,IPLAT ,KLATT ,KSLAT ,NVLAT ,IPCEL ,
11:     G LTYP ,ICTYP ,KDALT ,DALT ,KZREG,
12:     C ISUSP,KSPSU,KTCSP,NKTCSP,KHLAT,MKREG,
13:     C PPPF ,KPPF ,NPPPF,
14:     W X ,Y ,Z ,AI ,BI ,CI ,
15:     W IWK ,JKSF ,IPZONE, MEM ,DI, DLOC1, DLOC2, IKL, IKL2, R )
16: C=====
17: C PURPOSE: TO FIND CELLS WHICH PARTICLES IN LATTICE-BUFFER-ZONES
18: C ENTER.
19: C CALLED IN ACTION CALLS: (NONE)
20: C=====
21: C
22: C === INTER-STACK DATA FLOW ===
23: C
24: C LATTICE-SEARCH STACK -----> NEXT-ZONE-SEARCH STACK
25: C (LSLAT,IZLAT,NXLT) (LSSRC,IZNXT,NNXT)
26: C -----> Flight stack (STG zone)
27: C
28: C === BANK DATA TO BE UPDATED ===
29: C
30: C (XXX,YYY,ZZZ), (AAA,BBB,CCC), IZZ : POSITION, DIRECTION & ZONE #
31: C LEVL,LZZ,LPOS,LCRS : LATTICE PARAMETER BANK
32: C
33: C=====
34: C implicit real*8(D)
35: C
36: C integer JDEBG(*)
37: C
38: C .... PARTICLE BANK .....
39: C
40: C real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK)
41: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
42: C integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
43: C & LPOS(NBANK,NEST), LCRS(NBANK,NEST)
44: C integer KLSF(NBANK)
45: C integer IBREG(NBANK), IBSPC(NBANK,0:NEST)
46: C integer KDBNK(0:MDBNK), KIBNK(0:MIBNK)
47: C real*8 DBNK(NBANK,*)
48: C integer IBNK(NBANK,*)
49: C
50: C .... TALLY .....
51: C
52: C integer MKREG(NREG,2)
53: C
54: C .... STACKS .....
55: C
56: C integer LSDED(NBANK)
57: C integer LSSRC(NBANK), IZNXT(NBANK), NNXT(NZONE+1)
58: C integer LSLAT(NBANK), IZLAT(NBANK), NXLT(NLBZ+1)
59: C integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
60: C
61: C .... ZONE GEOMETRY DATA
62: C
63: C real*8 SDA(*)
64: C integer KZDA(2,*), KZAA(*)
65: C real*8 DINF, DEPS

```

```

66: C
67: C .... GEOMETRY ARRAY DATA
68: C
69: C integer KZMAT(NZONE), KZLBZ(NLBZ), MLBZZ(NZONE), KCELL(NZONE),
70: C & IDLAT(NLATT), IPLAT(NLATT+1), KLATT(*), KSLAT(*),
71: C & NVLAT(4,NLATT), IPCEL(NCELL), LTYP(NLATT), ICTYP(NCELL),
72: C & KDALT(NLBZ), KHLAT(*)
73: C real*8 CELSZ(6,NCELL), SZLAT(8,NLATT), DALT(*)
74: C
75: C integer KZREG(NZONE)
76: C integer ISUSP(NSUZON,NSPACE)
77: C integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
78: C
79: C ... Partial Packing Fraction for STGM-region
80: C integer KPPF(NPPPF)
81: C real PPPF(NPPPF)
82: C
83: C .... WORKING ARRAYS .....
84: C
85: C integer IWK(NBANK), JKSF(NZONE+1), IPZONE(NZONE+1), MEM(NZONE+1)
86: C real*8 X(NBANK), Y(NBANK), Z(NBANK)
87: C real*8 AI(NBANK), BI(NBANK), CI(NBANK)
88: C ... necessary when JFISX .ne.0 (and JSTGR.ne.0)
89: C real*8 DI(NBANK), DLOC1(NBANK), DLOC2(NBANK)
90: C integer IKL(NBANK), IKL2(NBANK)
91: C real R(3*NBANK)
92: C
93: C .... ROTATION VECTOR: ROT(N,1) = COS(-PAI/3 *N)
94: C ROT(N,2) = SIN(-PAI/3 *N)
95: C common /HEXROT/ DROT
96: C real*8 DROT(0:5,2)
97: C
98: C parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
99: C parameter( DPAI = 2.0D0*3.141592653589793238D0 )
100: C
101: C include 'INC/_IOUNIT'
102: C
103: C include 'INC/_PMLATT'
104: C include 'INC/_SFLATT'
105: C
106: C-----
107: C
108: C if ( JVMNT.ne.0 ) call VMNTRI( 7, NXLT(NLBZ+1) )
109: C
110: C .... REORDER PARTICLES BY LATTICE-BUFFER-ZONE (LB-ZONE) NUMBERS
111: C NZK = 0
112: C do 100 K = 1, NLBZ
113: C if ( NXLT(K).ne.0 ) then
114: C NZK = NZK + 1
115: C JKSF(NZK) = K
116: C end if
117: C 100 continue
118: C
119: C IF(NZK.GT.20) THEN
120: C IPZONE(1) = 1
121: C Cs*VOCL LOOP,NOVREC(MEM)
122: C DO 110 NK=1,NZK
123: C IPZONE(NK+1) = IPZONE(NK) + NXLT(JKSF(NK))
124: C MEM(JKSF(NK)) = IPZONE(NK) -1
125: C Cs110 CONTINUE
126: C Cs*VOCL LOOP,NOVREC(IWK)
127: C DO 120 I=1,NXLT(NLBZ+1)
128: C MEM(IZLAT(I)) = MEM(IZLAT(I)) + 1
129: C KKK = MEM(IZLAT(I))
130: C IWK(KKK) = IZLAT(I)

```

src/shared/latice.f

```

131: Cs120      CONTINUE
132: Cs      ELSE
133:      IPZONE(1) = 1
134:      do 110 NK = 1, NZK
135:          IPZONE(NK+1) = IPZONE(NK) + NXLT(JKSF(NK))
136:      110 continue
137:      KKK = 0
138:      do 130 NK = 1, NZK
139:          do 120 I = 1, NXLT(NLBZ+1)
140:              if ( IZLAT(I).eq.JKSF(NK) ) then
141:                  KKK = KKK + 1
142:                  IWK(KKK) = LSLAT(I)
143:              end if
144:          120 continue
145:      130 continue
146: Cs      ENDIF
147: C
148: C-----
149: C ..... Gather positions and directions .....
150: C-----
151: C
152:      do 140 I = 1, NXLT(NLBZ+1)
153:          AI(I) = AAA(IWK(I))
154:          BI(I) = BBB(IWK(I))
155:          CI(I) = CCC(IWK(I))
156:          X(I) = XXX(IWK(I))
157:          Y(I) = YYY(IWK(I))
158:          Z(I) = ZZZ(IWK(I))
159:      140 continue
160: C
161: C-----
162: C ..... Calculate distance to frame boundary for particles entering
163: C      lattice geometry region for the case allowing
164: C      overlapping of cells or frames.
165: C      (typically cases using STG particle region)
166: C-----
167: C
168:      if ( JFISX.ne.0 ) then
169:          do 150 I = 1, NXLT(NLBZ+1)
170:              IKL(I) = KLSF(IWK(I))
171:              DI(I) = DINF
172:          150 continue
173:          do 260 KZ = 1, NZK
174:              MZ = KZLBZ(JKSF(KZ))
175:              if ( ISLATT(KZMAT(MZ)) ) then
176:                  MLT = LATNM(KZMAT(MZ))
177:
178:                  IS = IPZONE(KZ)
179:                  IS2 = IPZONE(KZ+1) - 1
180:                  NN = IPZONE(KZ+1) - IPZONE(KZ)
181:                  JSS = 0
182:                  call SPEAR1( MZ, KZDA, KZAA, SDA, NN, NBANK, JSS, X(IS),
183:                      &      Y(IS), Z(IS), AI(IS), BI(IS), CI(IS), DI(IS),
184:                      &      DLOC1(IS), DLOC2(IS), IKL(IS), IKL2(IS),
185:                      &      DUMMY1, DINF, DEPS )
186: C
187: C      .... check lost particle: NXLT might be changed
188: C
189:                  IIL = 0
190:                  do 160 I = IS, IS2
191:                      if ( DI(I).eq.DINF ) IIL = IIL + 1
192:                  160 continue
193:
194:                  if ( IIL.gt.0 ) then
195: C/#IF PARA(SX* CRAY)

```

```

196: *      call MVPSYNC_LOCK(2)
197: C/#ENDIF
198:
199:      &      write(IMG, '(//lx,a,i5,2a/lx,3(a,i5))')
200:      &      '(LATICE) ', IIL,
201:      &      ' particles are lost in calculation of distance to ',
202:      &      'lattice frame.', ' Frame zone# ', MZ,
203:      &      ' Lattice # ', MLT, ' lattice ID ', IDLAT(MLT)
204:      II = 0
205:      do 170 I = IS, IS2
206:          if ( DI(I).eq.DINF ) then
207:              LL = IWK(I)
208:              write(IMG,7000) XXX(LL), YYY(LL), ZZZ(LL),
209:              &      AAA(LL), BBB(LL), CCC(LL), KLSF(LL), LL
210:              II = II + 1
211:              LSDED(NDEAD+II) = IWK(I)
212:
213: c##<2007/03/14:PN3:
214:              if ( KIBNK(17).ne.0 ) IBNK(IWK(I),KIBNK(17)) = 0
215:              if ( KIBNK(18).ne.0 ) IBNK(IWK(I),KIBNK(18)) = 0
216:              if ( KIBNK(19).ne.0 ) IBNK(IWK(I),KIBNK(19)) = 0
217: c##>
218:          end if
219:          170 continue
220: C/#IF PARA(SX* CRAY)
221: *      call MVPSYNC_UNLOCK(2)
222: C/#ENDIF
223:      NDEAD = NDEAD + II
224:      NLOST = NLOST + II
225:      JJ = 0
226:
227: *VOCL LOOP,NOVREC
228:      do 180 I = IS, IS2
229:          if ( DI(I).ne.DINF ) then
230:              AI(IS+JJ) = AI(I)
231:              BI(IS+JJ) = BI(I)
232:              CI(IS+JJ) = CI(I)
233:              X(IS+JJ) = X(I)
234:              Y(IS+JJ) = Y(I)
235:              Z(IS+JJ) = Z(I)
236:              DI(IS+JJ) = DI(I)
237:              IWK(IS+JJ) = IWK(I)
238:              JJ = JJ + 1
239:          end if
240:          180 continue
241:          NXLT(JKSF(KZ)) = NXLT(JKSF(KZ)) - II
242:          IS2 = IS2 - II
243:      end if
244:
245:      ... ILV is the current level, but it is assumed that level will
246:      be incremented in lattice cell placement procedure.
247:
248:      ILV = LEVL(IWK(IS))
249:      do 190 I = IS + 1, IS2
250:          if ( LEVL(IWK(I)).ne.ILV ) then
251:              ILV = -10
252:              go to 200
253:          end if
254:          190 continue
255:          200 continue
256:
257:      KD4 = KDBNK(4)
258:      KD41 = KDBNK(4) - 1
259:      KI6 = KIBNK(6)
260:      KD6 = KDBNK(6)

```

src/shared/lattice.f

```

261: C      ... all particles have the same lattice level ...
262: C
263: C      if ( ILV.ge.0 ) then
264: C          KK1      = KD4 + ILV
265: C          KK6      = KD6 + 6*ILV
266: C
267: C      ... store distance to frame, position to be reached
268: C      and direction in level ILV + 1 ...
269: C
270: C *VOCL LOOP,NOVREC
271: C      do 210 I = IS, IS2
272: C          DBNK(IWK(I),KK1) = DI(I)
273: C          DBNK(IWK(I),KK6) = X(I) + AI(I)*DI(I)
274: C          DBNK(IWK(I),KK6+1) = Y(I) + BI(I)*DI(I)
275: C          DBNK(IWK(I),KK6+2) = Z(I) + CI(I)*DI(I)
276: C          DBNK(IWK(I),KK6+3) = AI(I)
277: C          DBNK(IWK(I),KK6+4) = BI(I)
278: C          DBNK(IWK(I),KK6+5) = CI(I)
279: C      210 continue
280: C
281: C      ... set level # giving smallest frame distance in the
282: C      next level ...
283: C
284: C      if ( ILV.eq.0 ) then
285: C *VOCL LOOP,NOVREC
286: C          do 220 I = IS, IS2
287: C              IBNK(IWK(I),KI6) = 1
288: C      220 continue
289: C          else
290: C *VOCL LOOP,NOVREC
291: C          do 230 I = IS, IS2
292: C              if ( DBNK(IWK(I),KD4+IBNK(IWK(I),KI6+ILV-1))
293: C                  .lt.DI(I) ) then
294: C                  & IBNK(IWK(I),KI6+ILV) = IBNK(IWK(I),KI6+ILV-1)
295: C              else
296: C                  IBNK(IWK(I),KI6+ILV) = ILV + 1
297: C              end if
298: C      230 continue
299: C          end if
300: C      else
301: C *VOCL LOOP,NOVREC
302: C          do 240 I = IS, IS2
303: C              DBNK(IWK(I),KD4+LEVL(IWK(I))) = DI(I)
304: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I)))
305: C                  = X(I) + AI(I)*DI(I)
306: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I))+1)
307: C                  = Y(I) + BI(I)*DI(I)
308: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I))+2)
309: C                  = Z(I) + CI(I)*DI(I)
310: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I))+3) = AI(I)
311: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I))+4) = BI(I)
312: C              DBNK(IWK(I),KD6+6*LEVL(IWK(I))+5) = CI(I)
313: C      240 continue
314: C *VOCL LOOP,NOVREC
315: C          do 250 I = IS, IS2
316: C              IILV = LEVL(IWK(I))
317: C              if ( IILV.eq.0 ) then
318: C                  IBNK(IWK(I),KI6) = 1
319: C              else
320: C                  if ( DBNK(IWK(I),KD4+IBNK(IWK(I),KI6+IILV-1))
321: C                      .lt.DI(I) ) then
322: C                      & IBNK(IWK(I),KI6+IILV) =
323: C                      & IBNK(IWK(I),KI6+IILV-1)
324: C                  else
325: C                      IBNK(IWK(I),KI6+IILV) = IILV + 1

```

```

326: C      end if
327: C      end if
328: C      250 continue
329: C      end if
330: C      end if
331: C      260 continue
332: C      end if
333: C
334: C      7000 format(1X,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',E12.5,
335: C          & ' ETA=',E12.5,' XI=',E12.5,' SURFACE=',I5,' NO.=',I6)
336: C
337: C
338: C .....
339: C
340: C      SZLAT *      TYPE 1 *      TYPE 2 *      TYPE 3 *      TYPE 4
341: C      -----*-----*-----*-----*-----*-----
342: C      1 *      CELL-SIZE(X)*      CELL PITCH
343: C      2 *      (Y) *      ANGLE OF CELL ARRAY
344: C      3 *      (Z) *      CELL HEIGHT
345: C      -----*-----*-----*-----*-----
346: C      4 *      ... *      ORIGIN OF CELL ARRAY (X)
347: C      5 *      ... *      ORIGIN OF CELL ARRAY (Y)
348: C      -----*-----*-----*-----*-----
349: C      (LATTICE FRAME SIZE)
350: C      -----*-----*-----*-----*-----
351: C      6 *      ... *      LATTICE- *      CYLINDER *      WIDTH (X)
352: C      *      *      *      -RADIUS *
353: C      7 *      ... *      HEIGHT *      HEIGHT *      WIDTH (Y)
354: C      8 *      ... *      ... *      ... *      WIDTH (Z)
355: C      -----*-----*-----*-----*-----
356: C
357: C      <For STG region (LTYP=10) >
358: C
359: C      SZLAT(1) PF : packing fraction.
360: C      SZLAT(2) RSTG : spherical cell radius (cm) (optional)
361: C      SZLAT(3) WNND1 : radius mesh width relative to 2*RSTG for NND-1
362: C      (single value currently)
363: C      SZLAT(4) WNND2 : radius mesh width relative to 2*RSTG for NND-2
364: C      (single value currently)
365: C      SZLAT(5) WNND3 : radius mesh width relative to 2*RSTG for NND-3
366: C      (single value currently)
367: C .....
368: C
369: C      IDEAD = NDEAD
370: C      LSLL = 0
371: C
372: C      do 430 KZ = 1, NZK
373: C
374: C          ... MZ : ZONE # .... ... MLT: LATTICE # ....
375: C
376: C          MZ = KZLBZ(JKSF(KZ))
377: C
378: C          ... NXLT might be changed in frame distance calculation
379: C          so IPZONE(KZ+1)-1 might not be the last element index
380: C          to be processed for this zone.
381: C
382: C          IPZ2 = IPZONE(KZ) + NXLT(JKSF(KZ))
383: C =====
384: C          ..... PARTICLES ARE ENTERING A LATTICE .....
385: C =====
386: C          if ( ISLATT(KZMAT(MZ)) ) then
387: C              MLT = LATNM(KZMAT(MZ))
388: C              DSX = SZLAT(1,MLT)
389: C              DSY = SZLAT(2,MLT)
390: C              DSZ = SZLAT(3,MLT)

```

src/shared/lattice.f

```

391: C-----
392: C      ... M : POINTER TO DALT ARRAY
393: C-----
394: C      M      = KDALT(JKSF(KZ))
395: C      INN     = NNXT(NZONE+1) - IPZONE(KZ) + 1
396: C
397: C-----
398: C      RECTANGULAR LATTICE (LTYP(MLT) = 1)
399: C-----
400: C
401: C      if ( LTYP(MLT).eq.1 ) then
402: C
403: C      *VOCL LOOP,NOVREC
404: C      do 270 I = IPZONE(KZ), IPZ2 - 1
405: C
406: C-----
407: C      ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
408: C      ( ASSUMING THE BODY OF LATTICE ZONE IS 'RPP' OR 'BOX' )
409: C-----
410: C
411: C      DX      = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*
412: C      &      Z(I) - DALT(M+9)
413: C      DY      = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*
414: C      &      Z(I) - DALT(M+10)
415: C      DZ      = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*
416: C      &      Z(I) - DALT(M+11)
417: C      DA      = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*
418: C      &      CI(I)
419: C      DB      = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) +
420: C      &      DALT(M+5)*CI(I)
421: C      DC      = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) +
422: C      &      DALT(M+8)*CI(I)
423: C
424: C-----
425: C      ....IP: POSITION IN THE LATTICE .....
426: C-----
427: C
428: C      IX      =
429: C      &      MAX(0,
430: C      &      MIN(NVLAT(1,MLT)-1,INT((DX+DA*DEPS)/DSX)))
431: C      IY      =
432: C      &      MAX(0,
433: C      &      MIN(NVLAT(2,MLT)-1,INT((DY+DB*DEPS)/DSY)))
434: C      IZ      =
435: C      &      MAX(0,
436: C      &      MIN(NVLAT(3,MLT)-1,INT((DZ+DC*DEPS)/DSZ)))
437: C      IP      = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
438: C
439: C-----
440: C      ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
441: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
442: C-----
443: C
444: C      ICEL     = KLATT(IPLAT(MLT)+IP)
445: C      ISS      = KSLAT(IPLAT(MLT)+IP)
446: C      IDRX     = ISS/100
447: C      IDRY     = (ISS-IDRX*100)/10
448: C      IDRZ     = MOD(ISS,10)
449: C      JSX      = 1 - 2*IDRX
450: C      JSY      = 1 - 2*IDRY
451: C      JSZ      = 1 - 2*IDRZ
452: C      *VOCL STMT,IF(100)
453: C      if ( ICEL.ne.0 ) then
454: C
455: C-----

```

```

456: C      ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL
457: C-----
458: C      IPC      = IPCEL(ICEL)
459: C      IZZ(IWK(I)) = IPC
460: C      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
461: C      LZZ(IWK(I),LEVL(IWK(I))) = MZ
462: C      LPOS(IWK(I),LEVL(IWK(I))) = IP
463: C      if ( JHLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
464: C
465: C-----
466: C      .... ENTERING SUBSPACE # ....
467: C-----
468: C
469: C      if ( JTLLT.gt.0 ) then
470: C      IBSP      = IBSPC(IWK(I),LEVL(IWK(I))-1)
471: C      IBSPC(IWK(I),LEVL(IWK(I))) =
472: C      &      KTCSP(KSPSU(KZREG(MZ),IBSP)+IP)
473: C      end if
474: C
475: C-----
476: C-----
477: C      ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
478: C-----
479: C      XXX(IWK(I)) = CELSZ(4,ICEL)
480: C      &      + JSX*(DX-(IX+0.5D0)*DSX)
481: C      YYY(IWK(I)) = CELSZ(5,ICEL)
482: C      &      + JSY*(DY-(IY+0.5D0)*DSY)
483: C      ZZZ(IWK(I)) = CELSZ(6,ICEL)
484: C      &      + JSZ*(DZ-(IZ+0.5D0)*DSZ)
485: C      AAA(IWK(I)) = JSX*DA
486: C      BBB(IWK(I)) = JSY*DB
487: C      CCC(IWK(I)) = JSZ*DC
488: C
489: C-----
490: C      ..... CELL # = 0 : OUT OF LATTICE DEFINITION
491: C-----
492: C      else
493: C      IPC      = MZ
494: C      end if
495: C
496: C-----
497: C      ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
498: C-----
499: C      LSSRC(I+INN) = IWK(I)
500: C      IZNXT(I+INN) = IPC
501: C      270      continue
502: C      *VOCL LOOP,SCALAR
503: C      do 280 I = IPZONE(KZ), IPZ2 - 1
504: C      NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
505: C      280      continue
506: C      NNXT(NZONE+1) = INN + IPZ2 - 1
507: C
508: C-----
509: C      HEXAGONAL LATTICE ( LTYP(MLT) = 2,3,4 )
510: C-----
511: C
512: C      else if ( LTYP(MLT).ge.2.and.LTYP(MLT).le.4 ) then
513: C
514: C      *VOCL LOOP,NOVREC
515: C      do 290 I = IPZONE(KZ), IPZ2 - 1
516: C
517: C-----
518: C      ..... TRANSFORMATION TO CELL ARRAY COORDINATES & DIRECTION .....
519: C      ( ASSUMING THE BODY OF LATTICE ZONE IS 'RHP' OR 'HEX' )
520: C-----

```

src/shared/lattice.f

```

521: C
522:      DX      = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*
523:      &        Z(I) - DALT(M+9)
524:      DY      = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*
525:      &        Z(I) - DALT(M+10)
526:      DZ      = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*
527:      &        Z(I) - DALT(M+11)
528:      DA      = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*
529:      &        CI(I)
530:      DB      = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) +
531:      &        DALT(M+5)*CI(I)
532:      DC      = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) +
533:      &        DALT(M+8)*CI(I)
534: C
535: C-----
536: C      ... CELL-POSITION ...
537: C-----
538:      DXX      = DX + DA*DEPS
539:      DYY      = DY + DB*DEPS
540:      DZZ      = DZ + DC*DEPS
541:      N2       = 2*NVLAT(1,MLT)
542:      IN1      = INT(2.0*DXX/DSX+1.0+N2) - N2
543:      IN2      = INT((DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
544:      IN3      = INT((-DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
545:      IX       = (IN1+IN3+1) / 3
546:      IY       = (IN2+IN3) / 3
547: C
548:      IZ       = MAX(0,MIN(NVLAT(3,MLT)-1,INT(DZZ/DSZ)))
549: C
550:      IP       = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
551: C
552: C-----
553: C      ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
554: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
555: C-----
556: C
557:      IC       = IPLAT(MLT) + IP
558:      ICEL     = KLATT(IC)
559:      ISS      = KSLAT(IC)
560:      IDRX     = ISS/100
561:      JSX      = 1 - 2*IDRX
562:      IDRY     = (ISS-IDRX*100) / 10
563:      JSZ      = 1 - 2*MOD(ISS,10)
564: *VOCL STMT,IF(100)
565:      if ( ICEL.ne.0 ) then
566: C
567: C-----
568: C      ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL BUFFER-ZONE
569: C-----
570:      IPC      = IPCEL(ICEL)
571:      IZZ(IWK(I)) = IPC
572:      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
573:      LZZ(IWK(I),LEVL(IWK(I))) = MZ
574:      LPOS(IWK(I),LEVL(IWK(I))) = IP
575:      LCRS(IWK(I),LEVL(IWK(I))) = KHLAT(IPLAT(MLT)+IP)
576: C
577: C-----
578: C      .... ENTERING SUBSPACE # ....
579: C-----
580: C
581:      if ( JTLT.gt.0 ) then
582:      IBSP     = IBSPC(IWK(I),LEVL(IWK(I))-1)
583:      IBSPC(IWK(I),LEVL(IWK(I))) =
584:      &        KTCSP(KSPSU(KZREG(MZ),IBSP)+IP)
585:      end if

```

```

586: C
587: C
588: C-----
589: C      ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
590: C-----
591: C
592:      DX       = JSX*(DX-(IX+0.5*IY)*DSX)
593:      DY       = DY - IY*DSX*DHRT3
594:      XXX(IWK(I)) = DROT(IDRY,1)*DX - DROT(IDRY,2)*DY
595:      &        + CELSZ(3,ICEL)
596:      YYY(IWK(I)) = DROT(IDRY,2)*DX + DROT(IDRY,1)*DY
597:      &        + CELSZ(4,ICEL)
598:      ZZZ(IWK(I)) = JSZ*(DZ-(IZ+0.5D0)*DSZ)
599:      &        + CELSZ(5,ICEL)
600:      DA       = JSX*DA
601:      AAA(IWK(I)) = DROT(IDRY,1)*DA - DROT(IDRY,2)*DB
602:      BBB(IWK(I)) = DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
603:      CCC(IWK(I)) = JSZ*DC
604: C
605: C-----
606: C      ..... CELL # = 0 : OUT OF LATTICE DEFINITION
607: C-----
608:      else
609:      IPC      = MZ
610:      end if
611: C
612: C-----
613: C      ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
614: C-----
615:      LSSRC(I+INN) = IWK(I)
616:      IZNXT(I+INN) = IPC
617: C
618:      290      continue
619: *VOCL LOOP,SCALAR
620:      do 300 I = IPZONE(KZ), IPZ2 - 1
621:      NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
622:      300      continue
623:      NNXT(NZONE+1) = INN + IPZ2 - 1
624: C
625: C-----
626: C      STG region (LTYP(MLT) = 10)
627: C-----
628: C
629:      else if ( LTYP(MLT).eq.10 ) then
630:      NNN      = IPZ2 - IPZONE(KZ)
631:      ... packing ratio
632:      DFP      = SZLAT(1,MLT)
633:      K5       = KIBNK(5)
634:      K7       = KIBNK(7)
635:      IFFL     = NFFL(NZONE+1)
636: C
637:      ... Am I falling in STG particle ?
638: C
639:      call RANU2( IRAND, R(IPZONE(KZ)), NNN, ICON )
640:      NFF      = 0
641:      ICEL     = KLATT(IPLAT(MLT))
642: C      ... base material(matrix) zone in DUMMY cell ...
643:      KZIP     = KZREG(IPCEL(ICEL)+1)
644: *VOCL LOOP,NOVREC
645:      do 310 I = IPZONE(KZ), IPZ2 - 1
646: C
647: C      ... level count up here.
648: C
649:      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
650:      LVL       = LEVL(IWK(I))

```

src/shared/latice.f

```

651:      LZZ(IWK(I),LVL)      = MZ
652: C
653:      IKL(I)  = INT(DFP+R(I))
654: C
655: C      ... not in STG particle
656: C      sent to free flight stack of "matrix" zone.
657: C
658: C      if ( IKL(I).eq.0 ) then
659: C          ... cell position is 0 for STGM matrix
660: C          LPOS(IWK(I),LVL)  = 0
661: C
662: C      ... path flight count > 0 => on the outer boundary
663: C      of STG region
664: C      ... STG position is sampled from NND-3
665: C      ... else path is from matrix region ==> NND-2
666: C
667: C      if ( IBNK(IWK(I),K5).gt.0 ) then
668: C          IBNK(IWK(I),K7) = 3
669: C      else
670: C          IBNK(IWK(I),K7) = 2
671: C      end if
672: C
673: C      NFF      = NFF + 1
674: C      LSFFL(IFFL+NFF) = IWK(I)
675: C      IZFFL(IFFL+NFF) = MZ
676: C      IZZ(IWK(I))    = MZ
677: C
678: C      ... check marker region : set previous region to IBRG
679: C
680: C      if ( NMKREG.gt.0 ) then
681: C          IRRG = IBREG(IWK(I))
682: C          if ( IBNK(IWK(I),KIBNK(5)).lt.0 ) IRRG = 0
683: C      end if
684: C
685: C      if ( JTLT.gt.0 ) then
686: C          LVL = LVL-1
687: C          IBSP = IBSPC(IWK(I),LVL)
688: C          ... cell position is 0 for STGM matrix
689: C          IBSPC(IWK(I),LVL) =
690: C      &          KTCSP(KSPSU(KZREG(MZ),IBSP))
691: C          IBREG(IWK(I)) =
692: C      &          ISUSP(KZIP,IBSPC(IWK(I),LVL))
693: C
694: C      ... always set IBREG (from Jan 2000)
695: C      else
696: C          IBREG(IWK(I)) = KZREG(MZ)
697: C      end if
698: C
699: C      ... check escape from marker region
700: C
701: C      if ( NMKREG.gt.0.and.IRRG.ne.0.and.
702: C      &      MKREG(IRRG,1).ne.0.and.IRRG.ne.IBREG(IWK(I)) )
703: C      &      then
704: C          KG = KIBNK(9) + MKREG(IRRG,1) - 1
705: C          IBNK(IWK(I),KG) = IBNK(IWK(I),KG) + 1
706: C      end if
707: C      end if
708: C      continue
709: C      NFFL(MZ)      = NFFL(MZ) + NFF
710: C      NFFL(NZONE+1) = NFFL(NZONE+1) + NFF
711: C
712: C      ... some are in STG cell.
713: C
714: C      if ( NFF.lt.NNN ) then
715: C          II      = IPZONE(KZ) - 1
716: C
717: C      *VOCL LOOP,NOVREC
718: C
719: C      do 320 I = IPZONE(KZ), IPZ2 - 1
720: C          if ( IKL(I).ne.0 ) then
721: C              NNI      = NNI + 1
722: C              IKL2(NNI) = IWK(I)
723: C              X(II+NNI) = X(I)
724: C              Y(II+NNI) = Y(I)
725: C              Z(II+NNI) = Z(I)
726: C          end if
727: C          continue
728: C
729: C      ... STG cells are sampled ---> IKL:cell position in latice
730: C
731: C      NSTG      = KLATT(IPLAT(MLT)+9)
732: C      if ( NSTG.eq.1 ) then
733: C          do 330 I = 1, NNI
734: C              IKL(I) = 1
735: C          continue
736: C      else
737: C          call RANU2( IRAND, R, 2*NNI, ICON )
738: C          LPPF      = KLATT(IPLAT(MLT)+13)
739: C          do 340 I = 1, NNI
740: C              K      = MIN(NSTG-1, INT(NSTG*R(I)))
741: C              if ( R(NNI+I).lt.PPPF(LPPF+K) )
742: C              &          K = KPPF(LPPF+K) - 1
743: C              IKL(I) = K + 1
744: C          continue
745: C      end if
746: C
747: C      ... center position of STG particle is sampled
748: C
749: C      call RANU2( IRAND, R, 3*NNI, ICON )
750: C      KD5      = KDBNK(5)
751: C      IN0      = NNXT(NZONE+1)
752: C
753: C      ... reduce STG radius by DEPS to prevent particle
754: C      to be lost on STG surface when particle is
755: C      nearly on the surface of STG and flight direction
756: C      is outward from STG sphere
757: C
758: C      DR0      = SZLAT(2,MLT)-DEPS
759: C
760: C      do 350 I = 1, NNI
761: C          DRRR      = DR0*(R(I)**(1.0D0/3.0D0))
762: C
763: C          DMU      = 2.0D0*R(NNI+I) - 1.0D0
764: C          DSN      = DRRR*SQRT(1.0D0-DMU*DMU)
765: C          DFI      = DPAI*R(2*NNI+I)
766: C          DX      = DRRR*DMU
767: C          DY      = DSN*COS(DFI)
768: C          DZ      = DSN*SIN(DFI)
769: C          LV      = LEVL(IKL2(I))
770: C          K1      = KD5 + (LV-1)*3
771: C
772: C      ... STG cell position in level LV
773: C
774: C      DBNK(IKL2(I),K1) = X(II+I) + DX
775: C      DBNK(IKL2(I),K1+1) = Y(II+I) + DY
776: C      DBNK(IKL2(I),K1+2) = Z(II+I) + DZ
777: C
778: C      ... position in STG cell coordinats
779: C
780: C      ICEL      = KLATT(IPLAT(MLT)+IKL(I))

```


src/shared/lattice.f

```

781:      XXX(IKL2(I)) = CELSZ(1,ICEL) - DX
782:      YYY(IKL2(I)) = CELSZ(2,ICEL) - DY
783:      ZZZ(IKL2(I)) = CELSZ(3,ICEL) - DZ
784: C
785: C      ... no NND sampling in STG cell.
786: C
787:      IBNK(IKL2(I),K7) = 0
788: C
789: C      ... cell position in "lattice" is IKL for STG cell.
790: C
791:      LPOS(IKL2(I),LV) = IKL(I)
792: C
793:      IPC = IPCEL(ICEL)
794:      LSSRC(INO+I) = IKL2(I)
795:      IZNXT(INO+I) = IPC
796:      IZZ(IKL2(I)) = IPC
797: C
798:      if ( JTLT.gt.0 ) then
799:      IBSP = IBSPC(IKL2(I),LV-1)
800: C      ... cell position is IKL for STG cell
801:      IBSPC(IKL2(I),LV) =
802: &      KTCSP(KSPSU(KZREG(MZ),IBSP)+IKL(I))
803:      end if
804: 350      continue
805: C
806:      if ( NSTG.eq.1 ) then
807:      IPC = IPCEL(KLATT(IPLAT(MLT)+1))
808:      NNXT(IPC) = NNXT(IPC) + NNI
809:      else
810: *VOCL LOOP, SCALAR
811:      do 360 I = 1, NNI
812:      NNXT(IZNXT(INO+I)) = NNXT(IZNXT(INO+I)) + 1
813: 360      continue
814:      end if
815: C
816:      NNXT(NZONE+1) = NNXT(NZONE+1) + NNI
817:      end if
818:      end if
819: C
820: C
821: C =====
822: C      ..... PARTICLES ARE ENTERING ANOTHER CELL OR ESCAPING LATTICE
823: C =====
824: C
825:      else
826:      INN = NNXT(NZONE+1) - IPZONE(KZ) + 1
827:      ICEL = KCELL(MZ)
828: C
829: C -----
830: C      RECTANGULAR LATTICE-CELL (ICTYP = 1)
831: C -----
832: C
833:      if ( ICTYP(ICEL).eq.1 ) then
834: *VOCL LOOP, NOVREC
835:      do 370 I = IPZONE(KZ), IPZ2 - 1
836: C
837: CC ----- ILZ : ZONE # OF LATTICE FRAME -----
838: C
839:      ILZ = LZZ(IWK(I),LEVL(IWK(I)))
840:      IP = LPOS(IWK(I),LEVL(IWK(I)))
841:      MLT = LATNM(KZMAT(ILZ))
842:      M = KDALT(MLBZZ(ILZ))
843: C
844:      DSX = SZLAT(1,MLT)
845:      DSY = SZLAT(2,MLT)

```

```

846:      DSZ = SZLAT(3,MLT)
847: C
848: C -----
849: C      ..... CELL POSITION & CELL DIRECTION .....
850: C      ( NVLAT(4,MLT) = NVLAT(1,MLT)*NVLAT(2,MLT) )
851: C -----
852: C
853:      IZ = IP/NVLAT(4,MLT)
854:      IY = (IP-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
855:      IX = MOD(IP,NVLAT(1,MLT))
856:      ISS = KSLAT(IPLAT(MLT)+IP)
857:      IDRX = ISS/100
858:      IDRY = (ISS-IDRX*100) /10
859:      IDRZ = MOD(ISS,10)
860:      ISGX = 1 - 2*IDRX
861:      ISGY = 1 - 2*IDRY
862:      ISGZ = 1 - 2*IDRZ
863: C -----
864: C      ..... CELL ARRAY COORDINATES .....
865: C -----
866:      X(I) = ISGX*(X(I)-CELSZ(4,ICEL)) + (IX+0.5D0)*DSX
867:      Y(I) = ISGY*(Y(I)-CELSZ(5,ICEL)) + (IY+0.5D0)*DSY
868:      Z(I) = ISGZ*(Z(I)-CELSZ(6,ICEL)) + (IZ+0.5D0)*DSZ
869:      AI(I) = ISGX*AI(I)
870:      BI(I) = ISGY*BI(I)
871:      CI(I) = ISGZ*CI(I)
872:      DX = X(I) + AI(I)*DEPS
873:      DY = Y(I) + BI(I)*DEPS
874:      DZ = Z(I) + CI(I)*DEPS
875: C
876: C -----
877: C      ..... NEW POSITION IN THE LATTICE
878: C -----
879:      IX = INT(DX/DSX+NVLAT(1,MLT)) - NVLAT(1,MLT)
880:      IY = INT(DY/DSY+NVLAT(2,MLT)) - NVLAT(2,MLT)
881:      IZ = INT(DZ/DSZ+NVLAT(3,MLT)) - NVLAT(3,MLT)
882: C
883: C
884:      ICOUT = 0
885:      if ( IX.ge.0.and.IX.lt.NVLAT(1,MLT).and.IY.ge.0
886: &      .and.IY.lt.NVLAT(2,MLT).and.IZ.ge.0
887: &      .and.IZ.lt.NVLAT(3,MLT) ) then
888:      IP = IX + NVLAT(1,MLT)*IY + NVLAT(4,MLT)*IZ
889:      ICOUT = KLATT(IPLAT(MLT)+IP)
890:      end if
891: C
892: C -----
893: C      ..... PARTICLES HAVE ESCAPED THE LATTICE .....
894: C -----
895: C
896:      if ( ICOUT.eq.0 ) then
897: C -----
898: C      ..... TRANSFORMATION TO COORDINATES IN UPPER-LEVEL
899: C -----
900:      X(I) = X(I) + DALT(M+9)
901:      Y(I) = Y(I) + DALT(M+10)
902:      Z(I) = Z(I) + DALT(M+11)
903: C
904:      IK = IWK(I)
905:      XXX(IK) = DALT(M)*X(I) + DALT(M+3)*Y(I) + DALT(M+6)
906: &      *Z(I)
907:      YYY(IK) = DALT(M+1)*X(I) + DALT(M+4)*Y(I)
908: &      + DALT(M+7)*Z(I)
909:      ZZZ(IK) = DALT(M+2)*X(I) + DALT(M+5)*Y(I)
910: &      + DALT(M+8)*Z(I)

```

src/shared/latice.f

```

911:      AAA(IK) = DALT(M)*AI(I) + DALT(M+3)*BI(I)
912:      &      + DALT(M+6)*CI(I)
913:      BBB(IK) = DALT(M+1)*AI(I) + DALT(M+4)*BI(I)
914:      &      + DALT(M+7)*CI(I)
915:      CCC(IK) = DALT(M+2)*AI(I) + DALT(M+5)*BI(I)
916:      &      + DALT(M+8)*CI(I)
917:      IZZ(IWK(I)) = ILZ
918:      LEVL(IWK(I)) = LEVL(IWK(I)) - 1
919: C
920: C-----
921: C ..... PARTICLES ENTER ADJACENT CELLS ( ICOUT > 0 )
922: C-----
923: C
924:      else
925: C-----
926: C ..... CELL DIRECTION .....
927: C-----
928:      ISS = KSLAT(IPLAT(MLT)+IP)
929:      IDRX = ISS/100
930:      IDRY = (ISS-IDRX*100) /10
931:      IDRZ = MOD(ISS,10)
932:      JSX = 1 - 2*IDRX
933:      JSY = 1 - 2*IDRY
934:      JSZ = 1 - 2*IDRZ
935:      LPOS(IWK(I),LEVL(IWK(I))) = IP
936:      if ( JMLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
937: C-----
938: C ..... NEW IN-CELL COORDINATES .....
939: C-----
940:      XXX(IWK(I)) = CELSZ(4,ICOUT)
941:      &      + JSX*(X(I)-(IX+0.5D0)*DSX)
942:      YYY(IWK(I)) = CELSZ(5,ICOUT)
943:      &      + JSY*(Y(I)-(IY+0.5D0)*DSY)
944:      ZZZ(IWK(I)) = CELSZ(6,ICOUT)
945:      &      + JSZ*(Z(I)-(IZ+0.5D0)*DSZ)
946:      AAA(IWK(I)) = JSX*AI(I)
947:      BBB(IWK(I)) = JSY*BI(I)
948:      CCC(IWK(I)) = JSZ*CI(I)
949:      IZZ(IWK(I)) = IPCEL(ICOUT)
950: C
951: C-----
952: C .... NEW SUBSPACE # ...
953: C-----
954: C
955:      if ( JTLTLT.ne.0 ) then
956:      IBSP1 = IBSPC(IWK(I),LEVL(IWK(I))-1)
957:      IBSPC(IWK(I),LEVL(IWK(I))) =
958:      &      KTCSP(KSPSU(KZREG(ILZ),IBSP1)+IP)
959:      end if
960:      end if
961: C
962: C-----
963: C ..... NEXT-ZONE-SEARCH STACK ....
964: C-----
965: C
966:      LSSRC(I+INN) = IWK(I)
967:      NNZ = IZZ(IWK(I))
968:      IZNXT(I+INN) = NNZ
969:      370      continue
970: *VOCL LOOP, SCALAR
971:      do 380 I = IPZONE(KZ), IPZ2 - 1
972:      NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
973:      380      continue
974:      NNXT(NZONE+1) = INN + IPZ2 - 1
975: C

```

```

976: C -----
977: C      HEXAGONAL LATTICE-CELL      (ICTYP = 2)
978: C -----
979: C
980:      else if ( ICTYP(ICEL).eq.2 ) then
981: *VOCL LOOP,NOVREC
982:      do 390 I = IPZONE(KZ), IPZ2 - 1
983: CC ----- ILZ : ZONE # OF LATTICE FRAME -----
984:      LV = LEVL(IWK(I))
985:      ILZ = LZZ(IWK(I),LV)
986:      IP = LPOS(IWK(I),LV)
987:      MLT = LATNM(KZMAT(ILZ))
988:      M = KDALT(MLBZZ(ILZ))
989:      DSX = SZLAT(1,MLT)
990:      DSZ = SZLAT(3,MLT)
991: C
992: C-----
993: C ..... CELL POSITION & CELL DIRECTION .....
994: C      ( NVLAT(4,MLT) = NVLAT(1,MLT)*NVLAT(2,MLT) )
995: C-----
996:      IZ = IP/NVLAT(4,MLT)
997:      IY = (IP-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
998:      IX = MOD(IP,NVLAT(1,MLT))
999:      ISS = KSLAT(IPLAT(MLT)+IP)
1000:      IDRX = ISS/100
1001:      IDRY = (ISS-IDRX*100) /10
1002:      IDRZ = MOD(ISS,10)
1003: C
1004: C-----
1005: C ..... X,Y,Z,AI,BI,CI TO CELL ARRAY COORDINATE .....
1006: C-----
1007:      X(I) = X(I) - CELSZ(3,ICEL)
1008:      Y(I) = Y(I) - CELSZ(4,ICEL)
1009:      Z(I) = Z(I) - CELSZ(5,ICEL)
1010:      DX = DROT(IDRY,1)*X(I) + DROT(IDRY,2)*Y(I)
1011:      DY = -DROT(IDRY,2)*X(I) + DROT(IDRY,1)*Y(I)
1012:      X(I) = (1-2*IDRX)*DX + (IX+0.5D0*IY)*DSX
1013:      Y(I) = DY + DHRT3*IY*DSX
1014:      Z(I) = (1-2*IDRZ)*Z(I) + (IZ+0.5D0)*DSZ
1015:      DA = AI(I)
1016:      DB = BI(I)
1017:      AI(I) = (1-2*IDRX)*(DROT(IDRY,1)*DA+DROT(IDRY,2)*DB)
1018:      BI(I) = -DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
1019:      CI(I) = (1-2*IDRZ)*CI(I)
1020: C
1021: C-----
1022: C ..... CELL POSITION .....
1023: C-----
1024:      DXX = X(I) + AI(I)*DEPS
1025:      DYY = Y(I) + BI(I)*DEPS
1026:      DZZ = Z(I) + CI(I)*DEPS
1027:      N2 = 2*NVLAT(1,MLT)
1028:      IN1 = INT(2.0*DXX/DSX+1.0+N2) - N2
1029:      IN2 = INT((DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
1030:      IN3 = INT((-DXX+DYY*DROOT3)/DSX+1.0+N2) - N2
1031:      IX = (IN1-IN3+1) /3
1032:      IY = (IN2+IN3) /3
1033:      IZ = INT(DZZ/DSZ+NVLAT(3,MLT)) - NVLAT(3,MLT)
1034:      IP = IX + NVLAT(1,MLT)*IY + NVLAT(4,MLT)*IZ
1035:      IC = IPLAT(MLT) + IP
1036:      ICOUT = 0
1037:      if ( IX.ge.0.and.IX.lt.NVLAT(1,MLT).and.IY.ge.0
1038:      &      .and.IY.lt.NVLAT(2,MLT).and.IZ.ge.0
1039:      &      .and.IZ.lt.NVLAT(3,MLT) ) ICOUT = KLATT(IC)
1040: C

```

src/shared/lattice.f

```

1041: C -----
1042: C ---- PARTICLES LEAVE LATTICES ----
1043: C -----
1044:       if ( ICOUT.eq.0 ) then
1045:           X(I) = X(I) + DALT(M+9)
1046:           Y(I) = Y(I) + DALT(M+10)
1047:           Z(I) = Z(I) + DALT(M+11)
1048:           IK = IWK(I)
1049:           XX(X(IK)) = DALT(M)*X(I) + DALT(M+3)*Y(I) + DALT(M+6)
1050:           &           *Z(I)
1051:           YYY(IK) = DALT(M+1)*X(I) + DALT(M+4)*Y(I)
1052:           &           + DALT(M+7)*Z(I)
1053:           ZZZ(IK) = DALT(M+2)*X(I) + DALT(M+5)*Y(I)
1054:           &           + DALT(M+8)*Z(I)
1055:           AAA(IK) = DALT(M)*AI(I) + DALT(M+3)*BI(I)
1056:           &           + DALT(M+6)*CI(I)
1057:           BBB(IK) = DALT(M+1)*AI(I) + DALT(M+4)*BI(I)
1058:           &           + DALT(M+7)*CI(I)
1059:           CCC(IK) = DALT(M+2)*AI(I) + DALT(M+5)*BI(I)
1060:           &           + DALT(M+8)*CI(I)
1061:           IZZ(IWK(I)) = LZZ(IWK(I),LV)
1062:           LEVL(IWK(I)) = LEVL(IWK(I)) - 1
1063: C -----
1064: C ---- PARTICLES ENTER ADJACENT CELLS ----
1065: C -----
1066: C -----
1067:       else
1068:           ISS = XELAT(IPLAT(MLT)+IP)
1069:           IDRX = ISS/100
1070:           IDRY = (ISS-IDRX*100) /10
1071:           JSZ = 1 - 2*MOD(ISS,10)
1072: C -----
1073: C ----
1074: C .... NEW IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
1075: C -----
1076:           DX = (1-2*IDRX)*(X(I)-(IX+0.5*IY)*DSX)
1077:           DY = Y(I) - IY*DSX*DHRT3
1078:           XXX(IWK(I)) = DROT(IDRY,1)*DX - DROT(IDRY,2)*DY
1079:           &           + CELSZ(3,ICEL)
1080:           YYY(IWK(I)) = DROT(IDRY,2)*DX + DROT(IDRY,1)*DY
1081:           &           + CELSZ(4,ICEL)
1082:           ZZZ(IWK(I)) = JSZ*(Z(I)-(IZ+0.5)*DSZ)
1083:           &           + CELSZ(5,ICEL)
1084:           AI(I) = (1-2*IDRX)*AI(I)
1085:           AAA(IWK(I)) = DROT(IDRY,1)*AI(I) - DROT(IDRY,2)*
1086:           &           BI(I)
1087:           BBB(IWK(I)) = DROT(IDRY,2)*AI(I) + DROT(IDRY,1)*
1088:           &           BI(I)
1089:           CCC(IWK(I)) = JSZ*CI(I)
1090:           LPOS(IWK(I),LV) = IP
1091:           LCRS(IWK(I),LV) = KHLAT(IPLAT(MLT)+IP)
1092:           IZZ(IWK(I)) = IPCEL(ICOUT)
1093: C -----
1094: C ----
1095: C .... NEW SUBSPACE # ...
1096: C -----
1097: C -----
1098:       if ( JTLLT.ne.0 ) then
1099:           IBSP1 = IBSPC(IWK(I),LEVL(IWK(I))-1)
1100:           IBSPC(IWK(I),LEVL(IWK(I))) =
1101:           &           KTCSP(KSPSU(KZREG(ILZ),IBSP1)+IP)
1102:       end if
1103: end if
1104: C -----
1105: C -----

```

```

1106: C ..... NEXT-ZONE-SEARCH STACK ....
1107: C -----
1108:           LSSRC(I+INN) = IWK(I)
1109:           NNZ = IZZ(IWK(I))
1110:           IZNXT(I+INN) = NNZ
1111:       390 continue
1112: *VOCL LOOP,SCALAR
1113:       do 400 I = IPZONE(KZ), IPZ2 - 1
1114:           NNXT(IZNXT(INN+I)) = NNXT(IZNXT(INN+I)) + 1
1115:       400 continue
1116:           NNXT(NZONE+1) = INN + IPZ2 - 1
1117: C -----
1118: C -----
1119: C STG sphere as "LATTICE-CELL" (ICTYP = 3)
1120: C -----
1121: C -----
1122: C Particle leaves into base material of STGM region (matrix),
1123: C so lattice level is unchanged.
1124: C No change in direction vector
1125: C -----
1126:       else if ( ICTYP(ICEL).eq.3 ) then
1127:           KI = KDBNK(5)
1128:           KND = KIBNK(7)
1129:           IFFL = NFFL(NZONE+1) - IPZONE(KZ) + 1
1130: *VOCL LOOP,NOVREC
1131:       do 410 I = IPZONE(KZ), IPZ2 - 1
1132:           LV = LEVL(IWK(I))
1133:           K1 = KI + (LV-1)*3
1134:           XXX(IWK(I)) = X(I) - CELSZ(1,ICEL) + DBNK(IWK(I),K1)
1135:           YYY(IWK(I)) = Y(I) - CELSZ(2,ICEL) + DBNK(IWK(I),K1+1)
1136:           ZZZ(IWK(I)) = Z(I) - CELSZ(3,ICEL) + DBNK(IWK(I),K1+2)
1137:           IZZ(IWK(I)) = LZZ(IWK(I),LV)
1138: C -----
1139: C ... LPOS = 0 for matrix and >0 for STG cell
1140: C -----
1141:           LPOS(IWK(I),LV) = 0
1142: C -----
1143: C .... STG distance and position is sampled
1144: C from Nearest Neighbour Distribution (1)
1145: C ---> from a surface of a STG particle.
1146: C -----
1147:           IBNK(IWK(I),KND) = 1
1148: C -----
1149:           LSFFL(1+I) = IWK(I)
1150:           IZFFL(1+I) = IZZ(IWK(I))
1151: C -----
1152: C ... check marker region : set previous region to IRRG
1153: C -----
1154:       if ( NMKREG.gt.0 ) then
1155:           IRRG = IBREG(IWK(I))
1156:           if ( IBNK(IWK(I),KIBNK(5)).lt.0 ) IRRG = 0
1157:       end if
1158: C -----
1159:       if ( JTLLT.ne.0 ) then
1160:           IBSP1 = IBSPC(IWK(I),LV-1)
1161:           ... LPOS = 0 for STGM matrix
1162:           IBSPC(IWK(I),LV) =
1163:           &           KTCSP(KSPSU(KZREG(IZZ(IWK(I))),IBSP1))
1164: C -----
1165: C ... base material(matrix) zone in DUMMY cell ...
1166:           MMLT = LATNM(KZMAT(IZZ(IWK(I))))
1167:           KZIP = KZREG(IPCEL(KLATT(IPLAT(MMLT))))+1)
1168:           IBREG(IWK(I)) =
1169:           &           ISUSP(KZIP,IBSPC(IWK(I),LV))
1170:       else

```

src/shared/latice.f

```

1171: C          ... always set IBREG (from Jan 2000)
1172: C          IBREG(IWK(I)) = KZREG(IZZ(IWK(I)))
1173: C          end if
1174: C
1175: C          ... check escape from marker region
1176: C
1177: C          if ( NMKREG.gt.0.and.IRRG.ne.0.and.
1178: C      &      MKREG(IRRG,1).ne.0.and.IRRG.ne.IBREG(IWK(I)) )
1179: C      &      then
1180: C          KG      = KIBNK(9) + MKREG(IRRG,1) - 1
1181: C          IBNK(IWK(I),KG) = IBNK(IWK(I),KG) + 1
1182: C      end if
1183: C      410 continue
1184: C      *VOCL LOOP, SCALAR
1185: C      do 420 I = IPZONE(KZ) - IPZ2 - 1
1186: C          NFFL(IZFFL(IFFL+I)) = NFFL(IZFFL(IFFL+I)) + 1
1187: C      420 continue
1188: C          NFFL(NZONE+1) = IFFL + IPZ2 - 1
1189: C
1190: C      end if
1191: C
1192: C      end if
1193: C      430 continue
1194: C
1195: C-----
1196: C      ..... CLEAR NXLT
1197: C-----
1198: C
1199: C      do 440 I = 1, NLBZ + 1
1200: C          NXLT(I) = 0
1201: C      440 continue
1202: C      return
1203: C      end
1204: Cc
1205: C
1206: C=====
1207: C      ... initialization of DROT ...
1208: C=====
1209: C
1210: Cccc block data INDROT
1211: C      subroutine INDROT
1212: C
1213: C      .... ROTATION VECTOR: ROT(N,1) = COS(-PAI/3 *N)
1214: C                          ROT(N,2) = SIN(-PAI/3 *N)
1215: C      common /HEXROT/ DROT
1216: C      real*8 DROT(0:5,2)
1217: C
1218: C      data DROT /
1219: C      &      1.0D0, 0.5D0, -0.5D0,
1220: C      &      -1.0D0, -0.5D0, 0.5D0,
1221: C      &      0.0,-0.866025403784438652D0, -0.866025403784438652D0,
1222: C      &      0.0, 0.866025403784438652D0, 0.866025403784438652D0/
1223: C
1224: C      DROT(0,1) = 1.0D0
1225: C      DROT(1,1) = 0.5D0
1226: C      DROT(2,1) = -0.5D0
1227: C      DROT(3,1) = -1.0D0
1228: C      DROT(4,1) = -0.5D0
1229: C      DROT(5,1) = 0.5D0
1230: C      DROT(0,2) = 0.0
1231: C      ... modified on 25 Apr 1999 ,,,
1232: C      CCCC DROT(1,2) = 0.866025403784438652D0
1233: C      DROT(1,2) = -0.866025403784438652D0
1234: C      DROT(2,2) = -0.866025403784438652D0
1235: C      DROT(3,2) = 0.0
1236: C      DROT(4,2) = 0.866025403784438652D0
1237: C      DROT(5,2) = 0.866025403784438652D0
1238: C
1239: C      return
1240: C      end

```

src/shared/lattin.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine LATTIN( IN,  A,    IA,    LIMIT, CHA,    LIMITC,
3:     &                NLATT, IDLAT, LTYP,   NVLAT, SZLAT, IPLAT,
4:     &                NPNND, LKSLAT,LKLATT,LKHLAT,LPNND, LKNND,
5:     &                NPPPF, LPPPF,  LKPPF,
6:     &                JHLAT, JFISX, JSTGR, JDEBG )
7: C=====
8: C PURPOSE: INPUT OF LATTICE SPECIFICATION DATA
9: C CALLED IN: GEOMIN
10: C CALLS: hextr0 header riunit resize fread5 i4read rwind prlat0
11: C        r8read keep reset prstop
12: C-----
13: C arguments ( i=input, o=output, w=work )
14: C
15: C i IN : data input I/O unit (it may not be stdin, so RIUNIT should be
16: C        called)
17: C i o A(*), IA(*) : real/integer array for dynamic memory
18: C i LIMIT : effective memory size (word) of A(*) and IA(*)
19: C i o CHA(LIMITC),LIMITC : character memory (character*4) and its
20: C        element size
21: C i NLATT: number of lattice geometry type
22: C o IDLAT(NLATT) lattice geometry ID's
23: C o LTYP(NLATT): lattice geometry type
24: C o NVLAT(4,NLATT): X,Y,Z array size of lattice geometry
25: C o SZLAT(8,NLATT) : lattice geometry size parameters
26: C o IPLAT(NLATT+1) : starting array index for KLATT and KSLAT
27: C        for each lattice geometry.
28: C o NPNND: size of nearest neighbor distribution data PPNND and KNND
29: C o NPPPF: size of partial packing fraction data PPPF and KPPF
30: C o LKSLAT,LKLATT,LKHLAT,LPNND, LKNND :
31: C        array index pointers on A(*)/IA(*) for arrays KSLATT,KLATT,
32: C        KHLAT , PNND and KNND
33: C i JHLAT : using hexagonal geometry (non zero) or not (0).
34: C i JFISX : lattice frames or cells may intersect each other (non zero)
35: C        or not.
36: C i JSTGR : using STG region (non zero) or not (0).
37: C i JDEBG(*) : flags for debugging.
38: C=====
39: CCC   include '../shared/INC/_ARRAY'
40:       real A(*)
41: C       ... IA must have the same memory location as A(*)
42:       integer IA(*)
43:       character*4 CHA(*)
44: C
45:       include 'INC/_IOUNIT'
46: C
47:       integer JDEBG(*)
48:       integer IDLAT(NLATT), LTYP(NLATT), NVLAT(4,NLATT), IPLAT(NLATT+1)
49:       real*8 SZLAT(8,NLATT)
50: C
51: C ... local data ...
52: C
53:       real*8 SZHEX(5)
54:       character VNM*8
55:       character VNM2*32
56:       integer IRC(2)
57:       character*8 CC8
58: C
59: C ... for character input of cell placement
60:       parameter( MXCSYM = 80 )
61:       character*(MXCSYM) CSYM
62:       integer ICSYM(MXCSYM)
63:       character*128 CWRK
64: C
65: C ... for character input of cell direction

```

```

66:       parameter( MXDSYM = 32 )
67:       character*(MXDSYM) DRSYM
68:       integer IDRSYM(MXDSYM)
69: C
70: C-----
71: C
72:       NL      = 0
73:       JHLAT   = 0
74:       IRC(1)  = 99999
75:       IRC(2)  = 99999
76:       IPLAT(1) = 1
77:       IPNND   = 1
78:       ICPNND  = 0
79:       call RIUNIT( IN )
80:       call RESET
81: C
82:       KJKLAT  = 0
83:       KJKSLT  = 0
84:       KJSZLT  = 0
85:       KJSZHX  = 0
86:       KJRCEL  = 0
87: C
88:       NCSYM   = 0
89:       NDSYM   = 0
90: C
91:       NSTG    = 0
92:       NPCEL   = 0
93:       IPPPF   = 1
94:       MBASE   = 0
95: C
96: C ... input data KLATT, KSLAT etc. are stored on A(*) temporary
97: C        and freed after input of each lattice using LASTL0.
98: C
99:       call GTLAST( LASTL0 )
100: C
101: C
102: C
103: C ..... INPUT LATTICE DATA .....
104: C
105: C *   LTYP : LATTICE TYPE
106: C       = 1 : RECTANGULAR LATTICE
107: C       1 < 1 <=4 : HEXAGONAL LATTICE
108: C       = 10 : random sphere cell (STG-region)
109: C
110: C <For lattice type 1 to 4 (non STG) >
111: C *   NVLAT: LATTICE DIVISION NUMBERS IN X,Y & Z DIRECTIONS .....
112: C *   SZLAT: LATTICE SIZES IN X,Y & Z DIRECTIONS .....
113: C *   KLATT: CELL NUMBERS
114: C *   KSLAT: DIRECTION INDEXES ( DEFAULT VALUES = 0 )
115: C           DIRECTION =  J*100 + K*10 + L
116: C           J = 0/1 = NON-REVERSED/REVERSED IN X-DIRECTION
117: C           K = 0/1 = NON-REVERSED/REVERSED IN Y-DIRECTION
118: C           L = 0/1 = NON-REVERSED/REVERSED IN Z-DIRECTION
119: C *   RCELL: POSITION OF A CELL THAT ARE USED REFERENCE
120: C           POINT OF LATTICE FRAME (LTYP >1)
121: C
122: C Another way of KLATT input using "cell symbols";
123: C
124: C       CELIDS( cell-ID-list )
125: C       CELLSYMS( one-character-cell-symbol-list )
126: C       CKLATT( strings ... )
127: C
128: C CELLSYMS is a list of one-character cell symbols corresponding
129: C with cell-ID list in CELIDS().
130: C CKLATT() contains cell-ID mapping in a lattice using cell

```

src/shared/lattin.f

```

131: C      symbols
132: C
133: C      ex.
134: C
135: C      CELLDIDS( 0 1 12 100 )
136: C      CELLSYMS( 0 1 A B )
137: C      CKLATT( BBBB BBBB
138: C              BAAA111AAAB
139: C              BAA00000AAB
140: C              BAA00000AAB
141: C              BAA00000AAB
142: C              BAAAAAAAAB
143: C              BBBB BBBB )
144: C
145: C      is identical to:
146: C
147: C      KLATT( 11(100)
148: C            100 12 12 12 1 1 1 12 12 12 100
149: C            100 12 12 0 0 0 0 0 12 12 100
150: C            100 12 12 0 0 0 0 0 12 12 100
151: C            100 12 12 0 0 0 0 0 12 12 100
152: C            100 12 12 0 0 0 0 0 12 12 100
153: C            100 12 12 12 12 12 12 12 12 100
154: C            11(100) )
155: C
156: C      Strings in CKLATT can be separated blanks or newline, and
157: C      only the order of characters in strings has meanings.
158: C      So users can use "broken" or "dense" forms as follows:
159: C
160: C      CKLATT( B B B B B B B B B B
161: C              B A A A 1 1 1 A A A B
162: C              B A A 00000 A A B
163: C              B A A 00000 A A B
164: C              B A A 00000 A A B
165: C              B A A A A A A A A B
166: C              B B B B B B B B B B )
167: C
168: C      CKLATT( BBBB BBBB BAAA111AAAB BAA00000AAB BAA00000AAB
169: C              BAA00000AAB BAAAAAAAAB BBBB BBBB )
170: C
171: C      Beware that the repetition or interpolation syntax usable for
172: C      numeric data is not allowed for character list input like
173: C      CELLSYMS and CKLATT.
174: C
175: C
176: C      A similar character mode input is possible for KSLAT as follows:
177: C      (from 18 Nov 1999)
178: C
179: C      CELLDIRS( cell-direction-indices )
180: C      DIRSYMS( one-character-cell-direction-list )
181: C      CKSLAT( strings ... )
182: C
183: C      ex.
184: C
185: C      CELLDIRS( 0 0 100 010 110 )
186: C      DIRSYMS( 0 A X Y M )
187: C      CKSLAT( XXXXX0AAAAA
188: C              XXXXX0AAAAA
189: C              XXXXX0AAAAA
190: C              00000000000
191: C              M0000000000
192: C              M0000000000
193: C              M0000000000 )
194: C
195: C      <For STG region>
196: C
197: C
198: C      * PF : packing fraction. stored in SZLAT(1)
199: C
200: C      * RSTG : spherical cell radius (cm) stored in SZLAT(2)
201: C              (optional)
202: C
203: C      * MBASE( <cell ID for base (matrix) material> )
204: C
205: C      * CELL( < STG cell IDs > )
206: C
207: C      * PCELL( < partial packing fraction of STG cells > )
208: C
209: C      * WNND1 : radius mesh width relative to 2*RSTG for NND-1
210: C              (single value currently) -> SZLAT(3)
211: C
212: C      * FNND1 : cumulative probabilities for NND mesh for NND-1
213: C
214: C      * WNND2 : radius mesh width relative to 2*RSTG for NND-2
215: C              (single value currently) -> SZLAT(4)
216: C
217: C      * FNND2 : cumulative probabilities for NND mesh for NND-2
218: C
219: C      * WNND3 : radius mesh width relative to 2*RSTG for NND-3
220: C              (single value currently) -> SZLAT(5)
221: C
222: C      * FNND3 : cumulative probabilities for NND mesh for NND-3
223: C
224: C      ... for STG, elements of KLATT and KSLAT are used as follows
225: C
226: C      KLATT(1) : matrix material ID,
227: C      KLATT(2:9): STG cell IDs
228: C      KLATT(10) : the number of STG cells
229: C
230: C      KLATT(11) : division number if NND#1 or
231: C                  <=0 means analitic NND
232: C      KSLAT(11) : pointer to PNND(*) for NND#1
233: C
234: C      KLATT(12) : similar to KLATT(11) for NND#2
235: C      KSLAT(12) : similar to KSLAT(11) for NND#2
236: C
237: C      KLATT(13) : similar to KLATT(11) for NND#3
238: C      KSLAT(13) : similar to KSLAT(11) for NND#3
239: C
240: C      KLATT(14) : pointer to PPPF(*) for partial packing fraction
241: C
242: C      =====
243: C      call HEADER( IOG, 'LATTICE GEOMETRY DATA' )
244: C      =====
245: C
246: C      .... rewind working file to store KLATT & KSLAT temporary
247: C      call RWIND( IUB )
248: C
249: C      100 call FREADS( VNM, NLEN, IEND )
250: C
251: C      -----
252: C      ..... Input lattice ID number or end of data .....
253: C      -----
254: C
255: C      if ( VNM(:NLEN).eq.'IDLAT' .or. IEND.ne.0 .or. VNM(:NLEN).eq.'END'
256: C      & ) then
257: C
258: C      ..... PRINTOUT OF LATTICE PARAMETER .....
259: C      ..... PRE-TRANSFORMATION OF SZLAT PARAMETER HERE .....
260: C
261: C      (INTERFACE BETWEEN NEW WAY OF HEXCELL INPUT AND OLD WAY)
262: C      NOVEMBER 1990 M.SASAKI
263: C
264: C      if ( NL.ne.0 ) then

```

src/shared/lattin.f

```

261: C
262:       if ( LTYP(NL).lt.0 ) then
263:         write(IMG,*)
264:       &       'XXX(LATTIN) Lattice type (LTYP()) is not given for',
265:       &       ' lattice ', IDLAT(NL)
266:       call CNTERR( 'FATAL' )
267:     end if
268: C
269:       if ( LTYP(NL).eq.10 ) then
270:         if ( NSTG.le.0 .or. NSTG.gt.8 ) then
271:           write(IMG,*)
272:         &       'XXX(LATTIN) STG cell ID is not given or ',
273:         &       'more than 8 cells are given for STGM lattice ',
274:         &       IDLAT(NL)
275:         write(IMG,*)
276:         &       'The number of STG cell IDs given is ',NSTG
277:         call CNTERR( 'FATAL' )
278:       end if
279:       if ( NSTG.gt.1 .and. NPCEL.ne.NSTG ) then
280:         write(IMG,*)
281:       &       'XXX(LATTIN) The number of STG cells (CELL) ',
282:       &       'differs from that of PCELL ',
283:       &       IDLAT(NL)
284:       call CNTERR( 'FATAL' )
285:       if ( NPCEL.lt.NSTG ) then
286:         NPCEL = NSTG
287:         call KEEP( A(1), 'PCEL', LPCEL, NPCEL, 'R4'
288:       &         LLAST, 1.0, JDEBG )
289:         if ( LSTCK(0).lt.0 ) then
290:           VNM2 = 'PCEL'
291:           go to 260
292:         end if
293:       end if
294:       else if ( NSTG.eq.1 ) then
295:         if ( NPCEL.le.0 ) then
296:           NPCEL = 1
297:           call KEEP( A(1), 'PCEL', LPCEL, NPCEL, 'R4',
298:       &           LLAST, 1.0, JDEBG )
299:           if ( LSTCK(0).lt.0 ) then
300:             VNM2 = 'PCEL'
301:             go to 260
302:           end if
303:         else
304:           NPCEL = 1
305:         end if
306:       end if
307:       if ( MBASE.le.0 ) then
308:         write(IMG,*)
309:       &       'XXX(LATTIN) Base material pseudo cell ID (MBASE) is not'
310:       &       ' given or given non positive value for STGM lattice ',
311:       &       IDLAT(NL)
312:       call CNTERR( 'FATAL' )
313:     end if
314: C
315: C       ... to process the data when input errors are found
316:       NSTG = MAX(0, MIN(8, NSTG))
317: C
318:       IA(LKLT0) = MBASE
319:       do 10 I = 1, NSTG
320:         IA(LKLT0+I) = IA(LCLID+I-1)
321:       10 continue
322:       IA(LKLT0+9) = NSTG
323:       IA(LKLT0+10) = NNND1
324:       IA(LKSLT0+10) = IPNND
325:       IPNND = IPNND + MAX(0,NNND1)

```

```

326:       IA(LKLT0+11) = NNND2
327:       IA(LKSLT0+11) = IPNND
328:       IPNND = IPNND + MAX(0,NNND2)
329:       IA(LKLT0+12) = NNND3
330:       IA(LKSLT0+12) = IPNND
331:       IPNND = IPNND + MAX(0,NNND3)
332: C
333:       IA(LKLT0+13) = IPPPF
334:       IPPPF = IPPPF + MAX(0,NSTG)
335: C
336: C       ... working area to convert PNND's for alias method
337:       NNNDWK = MAX(MAX(0,NNND1),MAX(0,NNND2),MAX(0,NNND3))
338:       if ( NNNDWK.gt.0 ) then
339:         call KEEP( 'KNND1', LKNND1, NNNDWK, 'I4', LLAST, JDEBG
340:       &         )
341:         call KEEP( 'KNND2', LKNND2, NNNDWK, 'I4', LLAST, JDEBG
342:       &         )
343:         call KEEP( 'KNND3', LKNND3, NNNDWK, 'I4', LLAST, JDEBG
344:       &         )
345:         call KEEP( 'ALIAS', LALIAS, NNNDWK, 'R4', LLAST, JDEBG
346:       &         )
347:         if ( LSTCK(0).lt.0 ) then
348:           VNM2 = 'area for aliases of NND'
349:           go to 260
350:         end if
351:       end if
352: C
353: C       ... working area to convert PPF's for alias method
354:       if ( NSTG.gt.0 ) then
355:         call KEEP( 'KCEL', LKCEL, NSTG, 'I4', LLAST, JDEBG
356:       &         )
357:         if ( NSTG.gt.NNNDWK ) then
358:           call KEEP( 'ALIAS', LALIAS, NSTG, 'R4', LLAST,
359:       &           JDEBG )
360:           NNNDWK = NSTG
361:         end if
362:         if ( LSTCK(0).lt.0 ) then
363:           VNM2 = 'area for aliases of PPF'
364:           go to 260
365:         end if
366:       end if
367: C       ... lattice other than STG region
368:       else
369:         if ( NVLAT(1,NL).le.0 .or. NVLAT(2,NL).le.0
370:       &         .or. NVLAT(3,NL).le.0 ) then
371:           write(IMG,*)
372:         &       'XXX(LATTIN) Lattice division number(NVLAT(3)) is not',
373:         &       ' given or given non positive value for lattice ',
374:         &       IDLAT(NL)
375:         call CNTERR( 'FATAL' )
376:       end if
377:       if ( KJKLAT.eq.0 ) then
378:         write(IMG,*)
379:       &       'XXX(LATTIN) Lattice cell IDs (KLATT/CKLATT) are not',
380:       &       ' given for lattice ', IDLAT(NL)
381:       call CNTERR( 'FATAL' )
382:     end if
383:       if ( KJKSLT.eq.0 ) then
384:         write(IMG,*)
385:       &       '!!!(LATTIN) Lattice cell direction indices (KSLATT) are',
386:       &       ' not given for lattice ', IDLAT(NL)
387:       write(IMG,*)
388:       &       ' KSLAT=0 is assumed for all cell position.'
389:       call CNTERR( 'WARNING' )
390:     end if

```

src/shared/lattin.f

```

391:         if ( KJSZLT.eq.0 ) then
392:             write(IMG,*)
393:         &         'XXX(LATTIN) Lattice cell sizes (SZLAT) are not',
394:         &         ' given for lattice ', IDLAT(NL)
395:         call CNTERR( 'FATAL' )
396:     end if
397: end if
398: if ( LTYP(NL).ge.2.and.LTYP(NL).le.4 ) then
399:     if ( KJSZHX.eq.0 ) then
400:         write(IMG,*)
401:     &     'XXX(LATTIN) Hex. lattice cell parameters (SZHEX) are not',
402:     &     ' given for lattice ', IDLAT(NL)
403:     call CNTERR( 'FATAL' )
404:     end if
405:     if ( KJRCEL.eq.0 ) then
406:         write(IMG,*)
407:     &     'XXX(LATTIN) Hex. lattice reference cell position (RCELL) ',
408:     &     ' is not given for lattice ', IDLAT(NL)
409:     call CNTERR( 'FATAL' )
410:     end if
411: end if
412: C
413: KK      = NVLAT(4,NL)*NVLAT(3,NL)
414: IP      = IPLAT(NL)
415: IPS     = IPLAT(NL) * KK
416: call PRLAT( NL, IOG, IDLAT(NL), LTYP(NL), NVLAT(1,NL),
417: &         SZLAT(1,NL), IA(LKLT0), IA(LKSLT0), NNND1, NNND2,
418: &         NNND3, A(LNND1), A(LNND2), A(LNND3), IA(LKNND1),
419: &         IA(LKNND2), IA(LKNND3),
420: &         NSTG, A(LPCEL), IA(LKCEL),
421: &         A(LALIAS), NNNDWK, IRC )
422:
423: C
424: if ( LTYP(NL).eq.2 .or. LTYP(NL).eq.3 .or. LTYP(NL).eq.4 )
425: &     then
426:     call HEXTR0( SZLAT(1,NL), LTYP(NL), IRC )
427: end if
428: C
429: if ( LTYP(NL).eq.10 ) then
430:     if ( NSTG.gt.0 ) then
431:         write(IUB) 'PCEL '
432:         write(IUB) (A(LPCEL+I),I=0,NSTG-1),
433: &         (IA(LKCEL+I),I=0,NSTG-1)
434:     end if
435:     if ( MAX(0,NNND1).gt.0 ) then
436:         write(IUB) 'PNND1 '
437:         write(IUB) (A(LNND1+I),I=0,NNND1-1),
438: &         (IA(LKNND1+I),I=0,NNND1-1)
439:     end if
440:     if ( MAX(0,NNND2).gt.0 ) then
441:         write(IUB) 'PNND2 '
442:         write(IUB) (A(LNND2+I),I=0,NNND2-1),
443: &         (IA(LKNND2+I),I=0,NNND2-1)
444:     end if
445:     if ( MAX(0,NNND3).gt.0 ) then
446:         write(IUB) 'PNND3 '
447:         write(IUB) (A(LNND3+I),I=0,NNND3-1),
448: &         (IA(LKNND3+I),I=0,NNND3-1)
449:     end if
450: end if
451: C
452: C
453: C ... save KLATT & KSLAT in working file
454: C
455: write(IUB) 'KLATT '

```

```

456:         write(IUB) (IA(LKLT0+I),I=0,KK-1)
457: C
458: write(IUB) 'KSLAT '
459: write(IUB) (IA(LKSLT0+I),I=0,KK-1)
460: C
461: end if
462: C
463: C ... free area for KLATT, KSLAT etc. ...
464: C
465: call STLAST( 'LATTIN', LASTL0, LLAST )
466: C
467: if ( VNM(:NLEN).eq.'IDLAT' ) then
468:     NL      = NL + 1
469:     call I4READ( VNM, IDLAT(NL), NA, 1, IERR )
470: C
471:     do 110 IL = 1, NL - 1
472:         if ( IDLAT(NL).eq.IDLAT(IL) ) then
473:             write(IMG,*) 'XXX(LATTIN) Lattice ID ', IDLAT(NL),
474:             &             ' is used for that of ', IL,
475:             &             'th lattice already.'
476:             call CNTERR( 'FATAL' )
477:         end if
478:     110 continue
479: C
480:     write(IUB) 'IDLAT '
481:     write(IUB) IDLAT(NL)
482: C
483: C ... array index pointers for variables which may be input are
484: C reset here.
485: C
486: C LKLT0 : KLATT, LKSLT0 : KSLATT,
487: C
488:     LTYP(NL)      = -9999
489:     NVLAT(1,NL)   = -9999
490:     NVLAT(2,NL)   = -9999
491:     NVLAT(3,NL)   = -9999
492:     LKLT0         = -9999
493:     LKSLT0        = -9999
494:     LNND1         = -9999
495:     LNND2         = -9999
496:     LNND3         = -9999
497:     NNND1         = 0
498:     NNND2         = 0
499:     NNND3         = 0
500: C
501:     NCSYM         = 0
502:     CSYM          = ' '
503: C
504:     NDSYM         = 0
505:     DRSYM         = ' '
506: C
507:     KJKLAT        = 0
508:     KJKSLT        = 0
509:     KJSZLT        = 0
510:     KJSZHX        = 0
511:     KJRCEL        = 0
512: C
513:     LCLID         = -9999
514:     LPCEL         = -9999
515:     NSTG          = 0
516:     NPCEL         = 0
517:     MBASE         = 0
518: C
519: else if ( IEND.ne.0 .or. VNM(:NLEN).eq.'END' ) then
520:     go to 210

```


src/shared/lattin.f

```

521:         end if
522: C
523: C -----
524: C ..... INPUT LATTICE TYPE .....
525: C -----
526: C
527:     else if ( VNM(:NLEN).eq.'LTP' ) then
528:         call I4READ( VNM(:NLEN), LTP(NL), NA, -1, IERR )
529:         if ( IERR.ne.0 ) go to 270
530: C
531:     if ( LTP(NL).eq.1 ) then
532: C
533:     else if ( LTP(NL).ge.2.and.LTP(NL).le.4 ) then
534:         JHLAT = 1
535: C
536: C ... this problem includes STG-region and lattice frames may
537: C intersect
538: C
539:     else if ( LTP(NL).eq.10 ) then
540:         JSTGR = 1
541:         JFISX = 1
542: C
543: C ... for STG, NVLAT is set automatically here
544: C
545:         NVLAT(1,NL) = 15
546:         NVLAT(2,NL) = 1
547:         NVLAT(3,NL) = 1
548:         NVLAT(4,NL) = NVLAT(1,NL)*NVLAT(2,NL)
549:         KK = NVLAT(4,NL)*NVLAT(3,NL)
550:         IPLAT(NL+1) = IPLAT(NL) + KK
551:         call KEPV( A(1), 'KLATT', LKLT0, KK, 'I4', LLAST, 0, JDEBG )
552:         if ( LSTCK(0).lt.0 ) then
553:             VNM2 = 'KLATT'
554:             go to 260
555:         end if
556:         call KEPV( A(1), 'KSLAT', LKSLT0, KK, 'I4', LLAST, 0, JDEBG
557: &
558:         if ( LSTCK(0).lt.0 ) then
559:             VNM2 = 'KSLAT'
560:             go to 260
561:         end if
562:     else
563:         write(IMG,*) 'XXX(LATTIN) Unsupported lattice type LTP=',
564: &         LTP(NL), ' for lattice ID ', IDLAT(NL)
565:         call CNTERR( 'FATAL' )
566:     end if
567: C
568: C -----
569: C ..... INPUT LATTICE DIVISION NUMBER .....
570: C -----
571: C
572:     else if ( VNM(:NLEN).eq.'NVLAT' ) then
573:         call I4READ( VNM(:NLEN), NVLAT(1,NL), NA, -3, IERR )
574:         if ( IERR.ne.0 ) go to 270
575: C
576:         NVLAT(4,NL) = NVLAT(1,NL)*NVLAT(2,NL)
577:         KK = NVLAT(4,NL)*NVLAT(3,NL)
578:         IPLAT(NL+1) = IPLAT(NL) + KK
579: C
580:         call KEPV( A(1), 'KLATT', LKLT0, KK, 'I4', LLAST, 0, JDEBG )
581:         if ( LSTCK(0).lt.0 ) then
582:             VNM2 = 'KLATT'
583:             go to 260
584:         end if
585:         call KEPV( A(1), 'KSLAT', LKSLT0, KK, 'I4', LLAST, 0, JDEBG )

```

```

586:         if ( LSTCK(0).lt.0 ) then
587:             VNM2 = 'KSLAT'
588:             go to 260
589:         end if
590: C
591: C -----
592: C ..... CELL ID 's .....
593: C -----
594: C
595: C
596:     else if ( VNM(:NLEN).eq.'KLATT' ) then
597:         if ( LKLT0.lt.0 ) then
598:             write(IMG,*) 'XXX(LATTIN) KLATT(...) before input of NVLAT.'
599:             call CNTERR( 'FATAL' )
600:             call DMREAD( VNM(:NLEN), NA, NB, IERR )
601:         else
602:             KJKLAT = 1
603:             call I4READ( VNM(:NLEN), IA(LKLT0), NA, -KK, IERR )
604:             if ( IERR.ne.0 ) go to 270
605:         end if
606: C
607: C .... cell ID list for character mode input of KLATT (CKLATT)
608: C
609:     else if ( VNM(:NLEN).eq.'CELLIDS' ) then
610:         NB = MXCSYM
611:         call I4READ( VNM(:NLEN), ICSYM, NA, -NB, IERR )
612:         if ( IERR.ne.0 ) go to 270
613:         if ( NA.gt.NB ) then
614:             write(IMG,*) 'XXX(LATTIN) Too many entry for CELLIDS(...).',
615: &             ' Program limit (MXCSYM) is ', MXCSYM, '.'
616:             call CNTERR( 'FATAL' )
617:         end if
618:         NCSYM = NA
619: C
620: C ... duplicate cell ID's are allowed -->
621: C it may be convenient to make aliases.
622: C
623:         call CHKDUP( ICSYM, NCSYM, 1, NDUP, IOG, 'CELLIDS' )
624:         if ( NDUP.gt.0 ) then
625:             write(IMG,*) 'XXX(LATTIN) Duplicate ID in CELLIDS(...).'
626:             call CNTERR( 'FATAL' )
627:         end if
628: C
629: C ... cell symbol characters for CKLATT
630: C
631:     else if ( VNM(:NLEN).eq.'CELLSYMS' ) then
632:         NCS = 0
633:         call CHREAD( VNM(:NLEN), CC8, ' ', ML, ITERM, IEND1 )
634:         if ( IEND1.ne.0 ) go to 270
635:         if ( ML.gt.0 ) then
636:             NCS = NCS + 1
637:             if ( NCS.gt.NCSYM ) then
638:                 write(IMG,*)
639: & 'XXX(LATTIN) Number of entry for CELLSYMS(...) exceeds that of',
640: & ' CELLIDS(...) or CELLIDS is not input.'
641:                 call CNTERR( 'FATAL' )
642:             else if ( NCS.gt.MXCSYM ) then
643:                 write(IMG,*)
644: & 'XXX(LATTIN) Too many entry for CELLSYMS(...).',
645: & ' Program limit (MXCSYM) is ', MXCSYM, '.'
646:                 call CNTERR( 'FATAL' )
647:             else
648:                 CSYM(NCS:NCS) = CC8(1:1)
649:             end if
650:             if ( ML.gt.1 ) then

```

src/shared/lattin.f

```

651:         write(IMG,*)
652:         & 'XXX(LATTIN) Entry of CELLSYMS(...) must be a single character:',
653:         & ' <', CC8(:ML), '>'
654:         call CNTERR( 'FATAL' )
655:     end if
656: end if
657: if ( ITERM.eq.0 ) go to 120
658: C
659:     if ( NCS.ne.NCSYM ) then
660:         write(IMG,*)
661:         & 'XXX(LATTIN) Number of entry for CELLSYMS(...) is not equal to ',
662:         & 'that of CELLDIRS(...) or CELLIDS is not input.'
663:         call CNTERR( 'FATAL' )
664:     end if
665:     do 130 J = 1, NCSYM - 1
666:         K = INDEX(CSYM(J:NCSYM),CSYM(J:J))
667:         if ( K.gt.0 ) then
668:             write(IMG,*) 'XXX(LATTIN) Using same cell symbol <',
669:             & CSYM(J:J), '> for ', J, 'th and ', J + K,
670:             & 'th symbol.'
671:             call CNTERR( 'FATAL' )
672:         end if
673: 130 continue
674: C
675: C
676:     else if ( VNM(:NLEN).eq.'CKLATT' ) then
677:         ICOK = 1
678:         if ( NCSYM.eq.0 .or. CSYM.eq.' ' ) then
679:             write(IMG,*) 'XXX(LATTIN) CKLATT(...) is found but ',
680:             & 'cell symbol input (CELLIDS/CELLSYMS) is incomplete.'
681:             ICOK = 0
682:         end if
683:         if ( LKLT0.lt.0 ) then
684:             write(IMG,*)
685:             & 'XXX(LATTIN) CKLATT(...) before input of NVLAT.'
686:             ICOK = 0
687:         end if
688:         if ( ICOK.eq.0 ) then
689:             call PRSTOP( 1, 'Unrecoverable error in lattice input' )
690:         end if
691: C
692:         IKK = 0
693:         ICKERR = 0
694: C
695: 140 call CHREAD( VNM(:NLEN), CWRK, ' ', ML, ITERM, IEND1 )
696:         if ( IEND1.ne.0 ) go to 270
697:         if ( ML.gt.0 ) then
698:             do 150 I = 1, ML
699:                 IKK = IKK + 1
700:                 K = INDEX(CSYM(:NCSYM),CWRK(I:I))
701:                 if ( K.eq.0 ) then
702:                     write(IMG,*) 'XXX(LATTIN) Undefined cell symbol <',
703:                     & CWRK(I:I), '> in CKLATT. (', IKK,
704:                     & 'th character'
705:                     call CNTERR( 'FATAL' )
706:                 else if ( IKK.le.IKK ) then
707:                     IA(LKLT0+IKK-1) = ICSYM(K)
708:                 end if
709: 150 continue
710:         end if
711:         if ( ITERM.eq.0 ) go to 140
712: C
713:         if ( IKK.ne.IKK ) then
714:             write(IMG,*) 'XXX(LATTIN) Number of characters in CKLATT (',
715:             & IKK, ' ) does not match required one (', IKK, ' )'

```

```

716:         call CNTERR( 'FATAL' )
717:     end if
718:     KJKLAT = 1
719: C
720: C
721: C -----
722: C ..... CELL DIRECTION INDICATORS .....
723: C -----
724:     else if ( VNM(:NLEN).eq.'KSLAT' ) then
725:         if ( LKSLT0.lt.0 ) then
726:             write(IMG,*) 'XXX(LATTIN) KSLAT(...) before input of NVLAT.'
727:             call CNTERR( 'FATAL' )
728:             call DMREAD( VNM(:NLEN), NA, NB, IERR )
729:         else
730:             KJKSLT = 1
731:             call I4READ( VNM(:NLEN), IA(LKSLT0), NA, -KK, IERR )
732:             if ( IERR.ne.0 ) go to 270
733:         end if
734: C
735: C .... cell direction index list for character mode input of KSLAT
736: C (CKSLAT)
737: C
738:     else if ( VNM(:NLEN).eq.'CELLDIRS' ) then
739:         NB = MXDSYM
740:         call I4READ( VNM(:NLEN), IDRSYM, NA, -NB, IERR )
741:         if ( IERR.ne.0 ) go to 270
742:         if ( NA.gt.NB ) then
743:             write(IMG,*)
744:             & 'XXX(LATTIN) Too many entry for CELLDIRS(...).',
745:             & ' Allowable number (MXDSYM) is ', MXDSYM, ' '
746:             call CNTERR( 'FATAL' )
747:         end if
748:         NDSYM = NA
749: C
750: C ... cell symbol characters for CKSLAT
751: C
752:     else if ( VNM(:NLEN).eq.'DIRSYMS' ) then
753:         NCS = 0
754: 160 call CHREAD( VNM(:NLEN), CC8, ' ', ML, ITERM, IEND1 )
755:         if ( IEND1.ne.0 ) go to 270
756:         if ( ML.gt.0 ) then
757:             NCS = NCS + 1
758:             if ( NCS.gt.NDSYM ) then
759:                 write(IMG,*)
760:                 & 'XXX(LATTIN) Number of entry for DIRSYMS(...) exceeds that of',
761:                 & ' CELLDIRS(...) or CELLDIRS is not input.'
762:                 call CNTERR( 'FATAL' )
763:             else if ( NCS.gt.MXDSYM ) then
764:                 write(IMG,*)
765:                 & 'XXX(LATTIN) Too many entry for DIRSYMS(...).',
766:                 & ' Program limit (MXDSYM) is ', MXDSYM, ' '
767:                 call CNTERR( 'FATAL' )
768:             else
769:                 DRSYM(NCS:NCS) = CC8(1:1)
770:             end if
771:             if ( ML.gt.1 ) then
772:                 write(IMG,*)
773:                 & 'XXX(LATTIN) Entry of DIRSYMS(...) must be a single character:',
774:                 & ' <', CC8(:ML), '>'
775:                 call CNTERR( 'FATAL' )
776:             end if
777:         end if
778:         if ( ITERM.eq.0 ) go to 160
779: C
780:         if ( NCS.ne.NDSYM ) then

```

src/shared/lattin.f

```

781:      write(IMG,*)
782:      & 'XXX(LATTIN) Number of entry for DIRSYMS(...) is not equal to ',
783:      & 'that of CELLDIRS(...) or CELLDIRS is not input.'
784:      call CNTERR( 'FATAL' )
785:      end if
786:      do 170 J = 1, NDSYM - 1
787:        K = INDEX(DRSYM(J+1:NDSYM),DRSYM(J:J))
788:        if ( K.gt.0 ) then
789:          write(IMG,*)
790:          & 'XXX(LATTIN) Using same cell direction symbol <',
791:          & DRSYM(J:J), '> for ', J, 'th and ', J + K,
792:          & 'th symbol.'
793:          call CNTERR( 'FATAL' )
794:        end if
795:      170 continue
796: C
797: C
798:      else if ( VNM(:NLEN).eq.'CKSLAT' ) then
799:        ICOK = 1
800:        if ( NDSYM.eq.0 .or. DRSYM.eq.' ' ) then
801:          write(IMG,*) 'XXX(LATTIN) CKSLAT(...) is found but ',
802:          & 'cell direction symbol input (CELLDIRS/CELLDIRS) is incomplete.'
803:          ICOK = 0
804:        end if
805:        if ( LKSLT0.lt.0 ) then
806:          write(IMG,*)
807:          & 'XXX(LATTIN) CKSLAT(...) before input of NVLAT.'
808:          ICOK = 0
809:        end if
810:        if ( ICOK.eq.0 ) then
811:          call PRSTOP( 1, 'Unrecoverable error in lattice input' )
812:        end if
813: C
814:      IKK = 0
815:      ICKERR = 0
816: C
817: 180 call CHREAD( VNM(:NLEN), CWRK, ' ', ML, ITERM, IEND1 )
818:      if ( IEND1.ne.0 ) go to 270
819:      if ( ML.gt.0 ) then
820:        do 190 I = 1, ML
821:          IKK = IKK + 1
822:          K = INDEX(DRSYM(:NDSYM),CWRK(I:I))
823:          if ( K.eq.0 ) then
824:            write(IMG,*)
825:            & 'XXX(LATTIN) Undefined cell direction symbol <',
826:            & CWRK(I:I), '> in CKSLAT. ( ', IKK,
827:            & 'th character' )
828:            call CNTERR( 'FATAL' )
829:          else if ( IKK.le.KK ) then
830:            IA(LKSLT0+IKK-1) = IDRSYM(K)
831:          end if
832: 190 continue
833:      end if
834:      if ( ITERM.eq.0 ) go to 180
835: C
836:      if ( IKK.ne.KK ) then
837:        write(IMG,*) 'XXX(LATTIN) Number of characters in CKSLAT ( ',
838:        & IKK, ' ) does not match required one ( ', KK, ' )'
839:        call CNTERR( 'FATAL' )
840:      end if
841:      KJKSLT = 1
842: C
843: C
844: C
845: C -----

```

```

846: C      .... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
847: C -----
848: C
849: C
850: C      (REVISED FOR HEXAGONAL LATTICE (NOVEMVER 1990))
851: C
852: C      SZLAT *      TYPE 1      *      TYPE 2      *      TYPE 3      *      TYPE 4
853: C -----*-----*-----*-----*-----
854: C      1 *      CELL-SIZE(X)*      CELL PITCH
855: C      2 *      (Y) *      ANGLE OF CELL ARRAY (DEGREE) - 30 DEGREE
856: C      3 *      (Z) *      CELL HEIGHT
857: C -----*-----*-----*-----
858: C
859: C      ( SZLAT(4:8) are input as SZHEX(1:5) )
860: C
861: C      4 *      ... *      DEVIATION OF REFERENCE CELL CENETER
862: C      *      *      FROM THE CENTER OF THE LATTICE FRAME.
863: C      5 *      ... *      (VECTOR IN THE FRAME-COORDINATE-SYSTEM)
864: C      *      *
865: C -----
866: C      (LATTICE FRAME SIZE)
867: C -----
868: C      6 *      ... *      LATTICE- *      CYLINDER- *      WIDTH (X)
869: C      *      *      *      WIDTH *      RADIUS *
870: C      7 *      ... *      HEIGHT *      HEIGHT *      WIDTH (Y)
871: C      8 *      ... *      ... *      ... *      WIDTH (Z)
872: C -----
873: C
874:      else if ( VNM(:NLEN).eq.'SZLAT' ) then
875:        KJSZLT = 1
876:        call R8READ( VNM(:NLEN), SZLAT(1,NL), NA, -3, IERR )
877:        if ( IERR.ne.0 ) go to 270
878: C
879: C
880: C -----
881: C      .... SZHEX : SIZE-RELATED PARAMETERS OF HEXAGONAL LATTICE
882: C -----
883: C
884:      else if ( VNM(:NLEN).eq.'SZHEX' ) then
885:        KJSZHX = 1
886:        call R8READ( VNM(:NLEN), SZHEX, NA, -5, IERR )
887:        if ( IERR.ne.0 ) go to 270
888:        do 200 I = 1, 5
889:          SZLAT(I+3,NL) = SZHEX(I)
890: 200 continue
891: C
892: C -----
893: C      .... REFERENCE CELL POSITION .....
894: C -----
895: C
896:      else if ( VNM(:NLEN).eq.'RCELL' ) then
897:        KJRCEL = 1
898:        call I4READ( VNM(:NLEN), IRC, NA, 2, IERR )
899: C
900: C =====
901: C      For STG-region
902: C =====
903: C
904: C      .... PF ( obsolescnet keyword "FP" ) ....
905: C
906:      else if ( VNM(:NLEN).eq.'PF' .or. VNM(:NLEN).eq.'FP' ) then
907:        if ( LTYP(NL).ne.10 ) then
908:          write(IMG,7000) VNM(:NLEN), IDLAT(NL)
909:          call CNTERR( 'FATAL' )
910:        end if

```

src/shared/lattin.f

```

911: 7000 format(1X,'XXX(LATTIN) data ',A,'() is for lattice type ',
912: & ' "STG-region" (LTP=10). Lattice ID ',I8)
913: call R8READ( VNM(:NLEN), SZLAT(1,NL), NA, -1, IERR )
914: if ( IERR.ne.0 ) go to 270
915: C
916: C ..... RSTG .....
917: C
918: else if ( VNM(:NLEN).eq.'RSTG' ) then
919: if ( LTP(NL).ne.10 ) then
920: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
921: call CNTERR( 'FATAL' )
922: end if
923: C
924: call R8READ( VNM(:NLEN), SZLAT(2,NL), NA, -1, IERR )
925: if ( IERR.ne.0 ) go to 270
926: C
927: C ..... MBASE .....
928: C
929: else if ( VNM(:NLEN).eq.'MBASE' ) then
930: if ( LTP(NL).ne.10 ) then
931: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
932: call CNTERR( 'FATAL' )
933: end if
934: C
935: call I4READ( VNM(:NLEN), MBASE, NA, -1, IERR )
936: C
937: C ..... STGP or CELL .....
938: C
939: else if ( VNM(:NLEN).eq.'STGP' .or. VNM(:NLEN).eq.'CELL' ) then
940: if ( LTP(NL).ne.10 ) then
941: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
942: call CNTERR( 'FATAL' )
943: end if
944: C
945: KERR = 0
946: call ARAYIN( VNM(:NLEN), A, ICALL, 'I4', LCLID, KERR, JDEBG,
947: & LAST, NSTG, 'CELL' )
948: C
949: C ..... PCELL .....
950: C
951: else if ( VNM(:NLEN).eq.'PCELL' ) then
952: if ( LTP(NL).ne.10 ) then
953: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
954: call CNTERR( 'FATAL' )
955: end if
956: C
957: KERR = 0
958: call ARAYIN( VNM(:NLEN), A, ICALL, 'R4', LPCEL, KERR, JDEBG,
959: & LAST, NPCEL, 'PCELL' )
960: C
961: C ..... FNND1 .....
962: C
963: else if ( VNM(:NLEN).eq.'FNND1' ) then
964: if ( LTP(NL).ne.10 ) then
965: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
966: call CNTERR( 'FATAL' )
967: end if
968: KERR = 0
969: call ARAYIN( VNM(:NLEN), A, ICALL, 'R4', LNND1, KERR, JDEBG,
970: & LAST, NNND1, 'NNND1' )
971: C
972: C ..... FNND2 .....
973: C
974: else if ( VNM(:NLEN).eq.'FNND2' ) then
975: if ( LTP(NL).ne.10 ) then

```

```

976: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
977: call CNTERR( 'FATAL' )
978: end if
979: KERR = 0
980: call ARAYIN( VNM(:NLEN), A, ICALL, 'R4', LNND2, KERR, JDEBG,
981: & LAST, NNND2, 'NNND2' )
982: C
983: C ..... FNND3 .....
984: C
985: else if ( VNM(:NLEN).eq.'FNND3' ) then
986: if ( LTP(NL).ne.10 ) then
987: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
988: call CNTERR( 'FATAL' )
989: end if
990: KERR = 0
991: call ARAYIN( VNM(:NLEN), A, ICALL, 'R4', LNND3, KERR, JDEBG,
992: & LAST, NNND3, 'NNND3' )
993: C
994: C .... WNND1 ....
995: C
996: else if ( VNM(:NLEN).eq.'WNND1' ) then
997: if ( LTP(NL).ne.10 ) then
998: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
999: call CNTERR( 'FATAL' )
1000: end if
1001: call R8READ( VNM(:NLEN), SZLAT(3,NL), NA, -1, IERR )
1002: if ( IERR.ne.0 ) go to 270
1003: C
1004: C .... WNND2 ....
1005: C
1006: else if ( VNM(:NLEN).eq.'WNND2' ) then
1007: if ( LTP(NL).ne.10 ) then
1008: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
1009: call CNTERR( 'FATAL' )
1010: end if
1011: call R8READ( VNM(:NLEN), SZLAT(4,NL), NA, -1, IERR )
1012: if ( IERR.ne.0 ) go to 270
1013: C
1014: C .... WNND3 ....
1015: C
1016: else if ( VNM(:NLEN).eq.'WNND3' ) then
1017: if ( LTP(NL).ne.10 ) then
1018: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
1019: call CNTERR( 'FATAL' )
1020: end if
1021: call R8READ( VNM(:NLEN), SZLAT(5,NL), NA, -1, IERR )
1022: if ( IERR.ne.0 ) go to 270
1023: C
1024: C .... COPYNND ( lattice-ID ) ....
1025: C
1026: else if ( VNM(:NLEN).eq.'COPYNND' ) then
1027: ICPNND = ICPNND + 1
1028: if ( LTP(NL).ne.10 ) then
1029: write(IMG,7000) VNM(:NLEN), IDLAT(NL)
1030: call CNTERR( 'FATAL' )
1031: end if
1032: call I4READ( VNM(:NLEN), NNNDLT, NA, -1, IERR )
1033: NNND1 = -ABS(NNNDLT)
1034: NNND2 = -ABS(NNNDLT)
1035: NNND3 = -ABS(NNNDLT)
1036: C
1037: C
1038: C XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1039: C ... unknown data item.
1040: C XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

src/shared/lattin.f

```

1041: C
1042:     else
1043:         write(IMG,*) 'XXX(LATTIN) Unrecognizable input <', VNM(:NLLEN),
1044:         &         '> is found in input for Lattice ID ', IDLAT(NL)
1045:         call CNTERR( 'FATAL' )
1046:         call DMREAD( VNM(:NLLEN), NA, NB, IERR )
1047:         if ( IERR.ne.0 ) go to 270
1048:     end if
1049: C
1050:     go to 100
1051: C
1052: C
1053: C=====
1054: C === end of lattice data input ===
1055: C=====
1056: C
1057: C
1058: 210 continue
1059:     write(IUB) 'END '
1060: C
1061:     NPNND = IPNND - 1
1062:     NKLATT = IPLAT(NLATT+1) - 1
1063:     NPPPF = IPPPF - 1
1064: C
1065:     call RIUNIT( IOIN )
1066:     call RESET
1067: C
1068: C
1069: C .... Keep memory for KSLAT & KHLAT (and PNND if any) here.
1070: C
1071: C
1072:     call KEEP( 'KLATT', LKLATT, NKLATT, 'I4', LAST, JDEBG )
1073:     call KEEP( 'KSLAT', LKSLAT, NKLATT, 'I4', LAST, JDEBG )
1074: C
1075:     if ( JHLAT.ne.0 ) then
1076:         call KEEP( 'KHLAT', LKHLAT, NKLATT, 'I4', LAST, JDEBG )
1077:     end if
1078: C
1079:     if ( NPNND.gt.0 ) then
1080:         call KEEP( 'PNND', LPNND, NPNND, 'R4', LAST, JDEBG )
1081:         call KEEP( 'KNND', LKNND, NPNND, 'I4', LAST, JDEBG )
1082:     end if
1083: C
1084:     if ( NPPPF.gt.0 ) then
1085:         call KEEP( 'PPPF', LPPPF, NPPPF, 'R4', LAST, JDEBG )
1086:         call KEEP( 'KPPF', LKPPF, NPPPF, 'I4', LAST, JDEBG )
1087:     end if
1088: C
1089:     if ( LSIZE(LAST).gt.LIMIT ) then
1090:         write(IMG,7020) LSIZE(LAST) - 1
1091: 7020 format('XXX MEMORY INSUFFICIENT FOR PREPARATION OF ',
1092:         &         'LATTICE DATA !!! (',I9,' WORDS NECESSARY) !! STOP ')
1093:         call PRSTOP( 1, 'MEMORY OVER.' )
1094:         stop 999
1095:     end if
1096: C
1097: C .... restore KLATT & KSLAT from working file
1098: C
1099:     call RWIND( IUB )
1100:     N = 0
1101: 220 continue
1102:     read(IUB) CC8
1103:     if ( CC8.eq.'IDLAT' ) then
1104:         N = N + 1
1105:         read(IUB)

```

```

1106:         go to 220
1107:     else if ( CC8.eq.'KLATT' ) then
1108:         read(IUB) (IA(LKLATT+I-1),I=IPLAT(N),IPLAT(N+1)-1)
1109:         go to 220
1110:     else if ( CC8.eq.'KSLAT' ) then
1111:         read(IUB) (IA(LKSLAT+I-1),I=IPLAT(N),IPLAT(N+1)-1)
1112:         go to 220
1113:     else if ( CC8.ne.'END' ) then
1114:         read(IUB)
1115:         go to 220
1116:     end if
1117: C
1118: C .... restore NND from working file if any
1119: C
1120:     if ( NPNND.gt.0 ) then
1121:         call RWIND( IUB )
1122:         N = 0
1123:         II1 = LKLATT
1124:         II2 = LKSLAT
1125: 230 continue
1126:         read(IUB) CC8
1127:         if ( CC8.eq.'IDLAT' ) then
1128:             N = N + 1
1129:             read(IUB)
1130:             II1 = LKLATT + IPLAT(N) - 1
1131:             II2 = LKSLAT + IPLAT(N) - 1
1132:             go to 230
1133:         else if ( CC8.eq.'PNND1' ) then
1134:             NNND1 = IA(II1+10)
1135:             LNND1 = IA(II2+10)
1136:             read(IUB) (A(LPNNND+LNND1-1+I),I=0,NNND1-1),
1137:             &             (IA(LKNND+LNND1-1+I),I=0,NNND1-1)
1138:             go to 230
1139:         else if ( CC8.eq.'PNND2' ) then
1140:             NNND2 = IA(II1+11)
1141:             LNND2 = IA(II2+11)
1142:             read(IUB) (A(LPNNND+LNND2-1+I),I=0,NNND2-1),
1143:             &             (IA(LKNND+LNND2-1+I),I=0,NNND2-1)
1144:             go to 230
1145:         else if ( CC8.eq.'PNND3' ) then
1146:             NNND3 = IA(II1+12)
1147:             LNND3 = IA(II2+12)
1148:             read(IUB) (A(LPNNND+LNND3-1+I),I=0,NNND3-1),
1149:             &             (IA(LKNND+LNND3-1+I),I=0,NNND3-1)
1150:             go to 230
1151:         else if ( CC8.ne.'END' ) then
1152:             read(IUB)
1153:             go to 230
1154:         end if
1155:     end if
1156: C
1157: C .... process pointers for copied NND ...
1158: C
1159:     if ( ICPNND.gt.0 ) then
1160:         do 240 N = 1, NLATT
1161:             if ( LTYPE(N).eq.10 ) then
1162:                 II1 = LKLATT + IPLAT(N) - 1
1163:                 II2 = LKSLAT + IPLAT(N) - 1
1164:                 NNND1 = IA(II1+10)
1165:                 NNND2 = IA(II1+11)
1166:                 NNND3 = IA(II1+12)
1167: C
1168:                 if ( NNND1.lt.0 ) then
1169:                     call NNDCPY( N, 1, NNND1, IDLAT, LTYPE, NLATT,
1170:                     &                     IA(LKLATT), IA(LKSLAT), IPLAT )

```

src/shared/lattin.f

```
1171:           end if
1172: C
1173:           if ( NNND2.lt.0 ) then
1174:             call NNDCPY( N, 2, NNND2, IDLAT, LTYP, NLATT,
1175: &               IA(LKLATT), IA(LKSLAT), IPLAT )
1176:           end if
1177: C
1178:           if ( NNND3.lt.0 ) then
1179:             call NNDCPY( N, 3, NNND3, IDLAT, LTYP, NLATT,
1180: &               IA(LKLATT), IA(LKSLAT), IPLAT )
1181:           end if
1182:         end if
1183: 240      continue
1184:     end if
1185: C
1186: C ... restore PPF from working file if any
1187: C
1188:     if ( NPPPF.gt.0 ) then
1189:       call RWIND( IUB )
1190:       N = 0
1191:       I11 = LKLATT
1192: 250      continue
1193:       read(IUB) CC8
1194:       if ( CC8.eq.'IDLAT' ) then
1195:         N = N + 1
1196:         read(IUB)
1197:         I11 = LKLATT + IPLAT(N) - 1
1198:         go to 250
1199:       else if (CC8.eq.'PCEL' ) then
1200:         NSTG = IA(I11+9)
1201:         LPPF = IA(I11+13)
1202:         read(IUB) (A(LPPPF+LPPF-1+I),I=0,NSTG-1),
1203: &               (IA(LKPPF+LPPF-1+I),I=0,NSTG-1)
1204: Check %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1205: C       write(IOG,*) '%% PPF ',NSTG,LPPF,LPPPF,LKPPF
1206: C       write(IOG,*) (A(LPPPF+LPPF-1+I),I=0,NSTG-1)
1207: C       write(IOG,*) (IA(LKPPF+LPPF-1+I),I=0,NSTG-1)
1208:         go to 250
1209:       else if (CC8.ne.'END' ) then
1210:         read(IUB)
1211:         go to 250
1212:       end if
1213:     end if
1214: C
1215:     return
1216: C
1217: C XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1218: C   Handling of Fatal error ....
1219: C XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1220: C
1221: 260 write(IMG,*) 'XXX(LATTIN) Insufficient memory to store ',
1222: &       'lattice data (', VNM2(:ICLEN2(VNM2)),
1223: &       ') for lattice ID ', IDLAT(NL), '. !!! STOP prosessing'
1224:       call PRSTOP( 1, 'MEMORY OVER.' )
1225:       stop 999
1226: C
1227: 270 write(IMG,7040)
1228: 7040 format(1X,'XXX(LATTIN) Unexpected end of data or data read error',
1229: &       ' in reading lattice data.')
1230:       call PRSTOP( 1, 'ERROR DURING READING LATTICE DATA.' )
1231:       stop 888
1232: C
1233:     end
```

src/shared/lattr.f

```

1: *VOCL TOTAL,SCALAR
2: subroutine LATTR( JDEBG, NLATT, NZDA, NZONE, NLBZ, DEPS,
3: & IDLAT, NVLAT, SZLAT, KZDA, KZAA, KSFBD, IBSDA,
4: & NBODY, SDA, KZMAT, KMAT, KINPZ, NINPZ, LTYP,
5: & MLBZZ, KDALT, KLATT, DALT, NLTEMP, JSTGR, LDALT,
6: & LAST, IPLAT, JHLAT, KHLAT, LIMIT )
7: C=<MVP>=====
8: C PURPOSE: To check data for lattice geometry and
9: C transform ID-number data of each lattice geometry to
10: C sequential numbers convenient for use in the code.
11: C CALLED IN: GEOMIN
12: C
13: C-----
14: C implicit real*8(D)
15: C
16: C integer JDEBG(*)
17: C
18: C integer KZDA(2,NZDA), KZAA(*), KSFBD(NZDA)
19: C integer IBSDA(3,NBODY)
20: C integer KMAT(NINPZ), KINPZ(NZONE), KZMAT(NZONE,2)
21: C integer IDLAT(NLATT), NVLAT(4,NLATT), LTYP(NLATT), KDALT(NLBZ),
22: C & MLBZZ(NZONE), KHLAT(*), IPLAT(NLATT+1), KLATT(*)
23: C real*8 DEPS, SZLAT(8,NLATT), SDA(*), DALT(*)
24: C
25: C parameter( DPAI = 3.14159265358979323846 )
26: C parameter( DPAI18 = DPAI/180.0D0 )
27: C
28: C --- WORD LENGTH PARAMETER ( MWORD = 2 / 1 ... VP, SX, CRAY )
29: C SEE subroutine WDBL
30: C include 'INC/_WORDL'
31: C
32: C ... local data
33: C
34: C real*8 DHEX(2,6)
35: C data DHEX /0.5D0, 0.288675134594812893D0, 0.0D0,
36: C & 0.577350269189625787D0, -0.5D0, 0.288675134594812893D0,
37: C & -0.5D0, -0.288675134594812893D0, 0.0D0,
38: C & -0.577350269189625787D0, 0.5D0, -0.288675134594812893D0/
39: C
40: C character*4 BODTYP
41: C
42: C include 'INC/_IOUNIT'
43: C
44: C include 'INC/_PMLATT'
45: C include 'INC/_SFLATT'
46: C
47: C-----
48: C
49: C *****
50: C call LABEL( IOG, 'LATTICE GEOMETRY DATA TRANSFORMATION' )
51: C *****
52: C
53: C LDA = 1
54: C
55: C ..... LATTICE ID NUMBERS IN KZMAT --> LATTICE SEQUENTIAL NUMBERS
56: C & LATTICE SIZE CHECK.
57: C MATRIXES & CONSTANTS FOR COORDINATE TRANSFORMATION TO
58: C 'DALT' ARRAY.
59: C
60: C
61: C ..... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
62: C
63: C SZLAT * TYPE 1 * TYPE 2 * TYPE 3 * TYPE 4
64: C -----*-----
65: C 1 * CELL-SIZE(X)* CELL PITCH

```

```

66: C 2 * (Y) * ANGLE OF CELL ARRAY
67: C 3 * (Z) * CELL HEIGHT
68: C -----*-----
69: C 4 * ... * ORIGIN OF CELL ARRAY (X)
70: C 5 * ... * ORIGIN OF CELL ARRAY (Y)
71: C -----*-----
72: C (LATTICE FRAME SIZE)
73: C -----*-----
74: C 6 * ... * LATTICE- * CYLINDER * WIDTH (X)
75: C * * * WIDTH * -RADIUS *
76: C 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
77: C 8 * ... * * * * WIDTH (Z)
78: C -----*-----
79: C
80: C DEPS1 = DEPS*1.0D-5
81: C IFF = 0
82: C do 140 KZ = 1, NZONE
83: C CCCCC if ( KZMAT(KZ,1).le.-1.and.KZMAT(KZ,1).ge.-998 ) then
84: C if ( ISLATT(KZMAT(KZ,1)) ) then
85: C KLBZ = MLBZZ(KZ)
86: C
87: C ILAT = LATNM(KZMAT(KZ,1))
88: C do 100 I = 1, NLATT
89: C CCCCC if ( KZMAT(KZ,1).eq.-IDLAT(I) ) then
90: C if ( ILAT.eq.IDLAT(I) ) then
91: C go to 110
92: C end if
93: C 100 continue
94: C
95: C IFF = 1
96: C CCCCC write(IOG,7000) - KZMAT(KZ,1)
97: C write(IMG,7000) LATNM(KZMAT(KZ,1))
98: C 7000 format(1X,'XXX Any lattice whose ID is ',I5,
99: C & ' does not exist !!')
100: C & call CNTERR( 'FATAL' )
101: C go to 140
102: C
103: C 110 continue
104: C CCCCC KZMAT(KZ,1) = -I
105: C ... MLTOFF : material # offset for lattice frame region
106: C KZMAT(KZ,1) = MLTOFF - I
107: C KMAT(KINPZ(KZ)) = KZMAT(KZ,1)
108: C
109: C ... check body type of the first body (surface) of the zone
110: C
111: C IBID = KSFBD(KZAA(KZ))
112: C IBTYPE = 0
113: C do 120 IB = 1, NBODY
114: C if ( IBID.eq.IBSDA(2,IB) ) then
115: C IBTYPE = IBSDA(3,IB)
116: C go to 130
117: C end if
118: C 120 continue
119: C 130 call TYPBOD( BODTYP, '<', IBTYPE )
120: C
121: C =====
122: C ..... RECTANGULAR LATTICE .....
123: C =====
124: C
125: C if ( LTYP(I).eq.1 ) then
126: C
127: C if ( BODTYP.ne.'RPP'.and.BODTYP.ne.'BOX' ) then
128: C write(IMG,7020) KINPZ(KZ), BODTYP, IBID
129: C 7020 format(/1X,'XXX A rectangular lattice in input zone ',
130: C & I5,' is defined with a wrong body type <',A,

```

src/shared/lattr.f

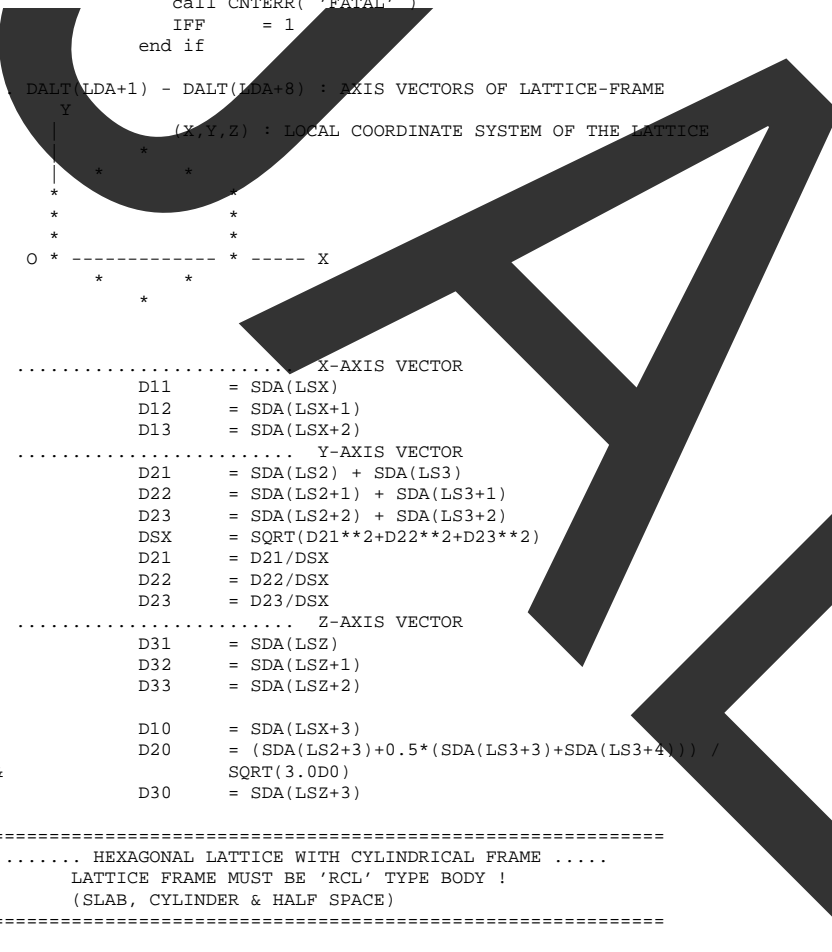
[illegible][illegible]

src/shared/lattr.f

```

261:      write(IMG,7120) ABS(SDA(LSX+4)-SDA(LSX+3))
262:      &      - SZLAT(6,I), ABS(SDA(LS2+4)-SDA(LS2+3))
263:      &      - SZLAT(6,I), ABS(SDA(LS3+4)-SDA(LS3+3))
264:      &      - SZLAT(6,I), ABS(SDA(LSZ+4)-SDA(LSZ+3))
265:      &      - NVLAT(3,I)*SZLAT(3,I)
266: 7120      format(/1X,
267:      &      '      Deviations from expected lattice size ',
268:      &      /1X,'      X : ',E12.5,' N2 : ',E12.5,
269:      &      ' N3 : ',E12.5,' Z : ',E12.5//)
270: C
271:      call CNTERR( 'FATAL' )
272:      IFF = 1
273:      end if
274: C
275: C ... DALT(LDA+1) - DALT(LDA+8) : AXIS VECTORS OF LATTICE-FRAME
276: C
277: C      (X,Y,Z) : LOCAL COORDINATE SYSTEM OF THE LATTICE
278: C
279: C
280: C
281: C
282: C
283: C
284: C
285: C
286: C
287: C
288: C ..... X-AXIS VECTOR
289:      D11 = SDA(LSX)
290:      D12 = SDA(LSX+1)
291:      D13 = SDA(LSX+2)
292: C ..... Y-AXIS VECTOR
293:      D21 = SDA(LS2) + SDA(LS3)
294:      D22 = SDA(LS2+1) + SDA(LS3+1)
295:      D23 = SDA(LS2+2) + SDA(LS3+2)
296:      DSX = SQRT(D21**2+D22**2+D23**2)
297:      D21 = D21/DSX
298:      D22 = D22/DSX
299:      D23 = D23/DSX
300: C ..... Z-AXIS VECTOR
301:      D31 = SDA(LSZ)
302:      D32 = SDA(LSZ+1)
303:      D33 = SDA(LSZ+2)
304: C
305:      D10 = SDA(LSX+3)
306:      D20 = (SDA(LS2+3)+0.5*(SDA(LS3+3)+SDA(LS3+4))) /
307:      &      SQRT(3.0D0)
308:      D30 = SDA(LSZ+3)
309: C
310: C =====
311: C ..... HEXAGONAL LATTICE WITH CYLINDRICAL FRAME .....
312: C      LATTICE FRAME MUST BE 'RCL' TYPE BODY !
313: C      (SLAB, CYLINDER & HALF SPACE)
314: C =====
315: C
316:      else if ( LTYPE(I).eq.3 ) then
317:      if ( BODTYP.ne.'RCL' ) then
318:      write(IMG,7140) KINPZ(KZ), BODTYP, IBID
319: 7140      format(/1X,
320:      &      'XXX A type 3 hexagonal lattice in input zone ',
321:      &      I5,' is defined with a wrong body type <',
322:      &      A,'>,' (ID=',I6,')'/1X,
323:      &      ' Definition of frame zone of type 3',
324:      &      ' lattice must begin with body RCL.')
325:      call CNTERR( 'FATAL' )

```



```

326:      IFF = 1
327:      end if
328: C
329: C ..... POINTER TO SLAB (TOP & BOTTOM SURFACE)
330:      LS1 = ABS(KZDA(1,KZAA(KZ))) + 1
331: C
332: C ..... POINTER TO CYLINDER
333: C
334:      LS2 = ABS(KZDA(1,KZAA(KZ)+1)) + 1
335: C
336: C ..... POINTER TO HALF SPACE WHOSE NORMAL
337: C      VECTOR IS X-AXIS VECTOR
338: C
339:      LS3 = ABS(KZDA(1,KZAA(KZ)+2)) + 1
340: C
341: C ..... INVALID LATTICE FRAME ...CHANGED ON MAY 14,1993
342: C
343:      DSR = SQRT(SDA(LS2+6)) - SZLAT(6,I)
344:      DSZ = ABS(SDA(LS1+4)-SDA(LS1+3)) - SZLAT(7,I)
345:      DS1 = ABS(SDA(LS1+4)-SDA(LS1+3)) - NVLAT(3,I)*
346:      &      SZLAT(3,I)
347:      if ( ABS(DSR).gt.DEPS1 .or. DSR.gt.DEPS1
348:      &      .or. ABS(DSZ).gt.DEPS1 .or. DSZ.gt.DEPS1
349:      &      .or. ABS(DS1).gt.DEPS1 .or. DS1.gt.DEPS1 ) then
350: C
351:      write(IMG,7160) KINPZ(KZ), IDLAT(I)
352:      write(IMG,7180) DSR, DSZ, DS1
353: 7160      format(/1X,'XXX A lattice in input-zone ',I5,
354:      &      ' has invalid FRAME size as lattice ',I4,
355:      &      ' (Invalid RCL body size)')
356: 7180      format(/1X,
357:      &      '      Deviations from expected lattice size ',
358:      &      /1X,'      Radius : ',E12.5,' Height : ',
359:      &      E12.5,E12.5//)
360:      call CNTERR( 'FATAL' )
361:      IFF = 1
362:      end if
363: C
364: C ..... X-AXIS VECTOR .....
365:      D11 = SDA(LS3)
366:      D12 = SDA(LS3+1)
367:      D13 = SDA(LS3+2)
368:      D10 = D11*SDA(LS2) + D12*SDA(LS2+1)
369:      &      + D13*SDA(LS2+2)
370: C
371: C ..... Y-AXIS VECTOR
372:      DX = SDA(LS3+2)*SDA(LS1+1) - SDA(LS3+1)*
373:      &      SDA(LS1+2)
374:      DY = SDA(LS3)*SDA(LS1+2) - SDA(LS3+2)*SDA(LS1)
375:      DZ = SDA(LS3+1)*SDA(LS1) - SDA(LS3)*SDA(LS1+1)
376:      DH = SQRT(DX**2+DY**2+DZ**2)
377:      D21 = DX/DH
378:      D22 = DY/DH
379:      D23 = DZ/DH
380:      D20 = D21*SDA(LS2) + D22*SDA(LS2+1)
381:      &      + D23*SDA(LS2+2)
382: C
383: C ..... Z-AXIS VECTOR
384:      D31 = SDA(LS1)
385:      D32 = SDA(LS1+1)
386:      D33 = SDA(LS1+2)
387:      D30 = D31*SDA(LS2) + D32*SDA(LS2+1)
388:      &      + D33*SDA(LS2+2)
389: C
390: C =====

```

src/shared/lattr.f

```

391: C      .... HEXAGONAL LATTICE WITH RECTANGULAR FRAME ....
392: C      LATTICE FRAME MUST BE 'RPP' OR 'BOX' !
393: C      =====
394: C
395: C      else if ( LTYP(I).eq.4 ) then
396: C          if ( BODTYP.ne.'RPP'.and.BODTYP.ne.'BOX' ) then
397: C              write(IMG,7200) KINPZ(KZ), BODTYP, IBID
398: C              format(/1X,
399: C                  'XXX A type 4 hexagonal lattice in input zone ',
400: C                  'I5,' is defined with a wrong body type < ',
401: C                  A,'>', '(ID=',I6,')'/1X,
402: C                  ' Definition of frame zone of type 4',
403: C                  ' lattice must begin with body RPP or BOX.'
404: C                  )
405: C              call CNTERR( 'FATAL' )
406: C              IFF = 1
407: C          end if
408: C          LSX = ABS(KZDA(1,KZAA(KZ))) + 1
409: C          LSY = ABS(KZDA(1,KZAA(KZ)+1)) + 1
410: C          LSZ = ABS(KZDA(1,KZAA(KZ)+2)) + 1
411: C
412: C          IF(
413: C              @ ABS(ABS(SDA(LSX+4)-SDA(LSX+3))-SZLAT(6,I))
414: C              @ .GT. DEPS .OR.
415: C              @ ABS(ABS(SDA(LSY+4)-SDA(LSY+3))-SZLAT(7,I))
416: C              @ .GT. DEPS .OR.
417: C              @ ABS(ABS(SDA(LSZ+4)-SDA(LSZ+3))-SZLAT(8,I))
418: C              @ .GT. DEPS ) THEN
419: C              C1993-6-17 ...
420: C              ..... INVALID LATTICE FRAME ....CHANGED ON MAY 14,1993
421: C
422: C              DSX = ABS(SDA(LSX+4)-SDA(LSX+3)) - SZLAT(6,I)
423: C              DSY = ABS(SDA(LSY+4)-SDA(LSY+3)) - SZLAT(7,I)
424: C              DSZ = ABS(SDA(LSZ+4)-SDA(LSZ+3)) - SZLAT(8,I)
425: C              if ( ABS(DSX).gt.DEPS .or. DSX.gt.DEPS1
426: C                  & .or. ABS(DSY).gt.DEPS .or. DSY.gt.DEPS1
427: C                  & .or. ABS(DSZ).gt.DEPS .or. DSZ.gt.DEPS1 ) then
428: C
429: C                  write(IMG,7040) KINPZ(KZ), IDLAT(I)
430: C                  write(IMG,7120) DSX, DSY, DSZ
431: C                  call CNTERR( 'FATAL' )
432: C                  IFF = 1
433: C              end if
434: C
435: C              D11 = SDA(LSX)
436: C              D12 = SDA(LSX+1)
437: C              D13 = SDA(LSX+2)
438: C              D21 = SDA(LSY)
439: C              D22 = SDA(LSY+1)
440: C              D23 = SDA(LSY+2)
441: C              D31 = SDA(LSZ)
442: C              D32 = SDA(LSZ+1)
443: C              D33 = SDA(LSZ+2)
444: C              D10 = SDA(LSX+3)
445: C              D20 = SDA(LSY+3)
446: C              D30 = SDA(LSZ+3)
447: C          end if
448: C
449: C      .... Transformation matrix is synthesized from two matrices.
450: C      One of them is for transformation to the lattice-frame coordinate,
451: C      and the other is for transformation to the coordinate system
452: C      attached to the lattice-cell array.
453: C
454: C      *      *      *      *      *
455: C      * DC DS 0 * * D11 D12 D13 * * DALT(0) DALT(1) DALT(2) *

```

```

456: C      *-DS DC 0 * * D21 D22 D23 * = * DALT(3) DALT(4) DALT(5) *
457: C      * 0 0 1 * * D31 D32 D33 * * DALT(6) DALT(7) DALT(8) *
458: C      *      *      *      *      *
459: C      |      |      |
460: C      B      A      =      BA
461: C      *      *      *      *      *
462: C      * X' *      * X *      * D10 *
463: C      * Y' * = A * Y * - * D20 * ( 1'ST STEP )
464: C      * Z' *      * Z *      * D30 *
465: C      *      *      *      *      *
466: C
467: C      *      *      *      *      *
468: C      * X'' *      * X' - SZLAT(4) *      * X *      * D10+SZLAT(4) *
469: C      * Y'' * = B * Y' - SALAT(5) * = BA * Y * - B* D20+SZLAT(5) *
470: C      * Z'' *      * Z'      * Z *      * D30 *
471: C      *      *      *      *      *
472: C
473: C      DC = COS(SZLAT(2,I)*DPAI18)
474: C      DS = SIN(SZLAT(2,I)*DPAI18)
475: C      DALT(LDA) = DC*D11 + DS*D21
476: C      DALT(LDA+1) = DC*D12 + DS*D22
477: C      DALT(LDA+2) = DC*D13 + DS*D23
478: C      DALT(LDA+3) = -DS*D11 + DC*D21
479: C      DALT(LDA+4) = -DS*D12 + DC*D22
480: C      DALT(LDA+5) = -DS*D13 + DC*D23
481: C      DALT(LDA+6) = D31
482: C      DALT(LDA+7) = D32
483: C      DALT(LDA+8) = D33
484: C      D10 = D10 + SZLAT(4,I)
485: C      D20 = D20 + SZLAT(5,I)
486: C      DALT(LDA+9) = DC*D10 + DS*D20
487: C      DALT(LDA+10) = -DS*D10 + DC*D20
488: C      DALT(LDA+11) = D30
489: C      DALT(LDA+12) = DC
490: C      DALT(LDA+13) = DS
491: C      LDA0 = LDA
492: C      LDA = LLDA
493: C
494: C      =====
495: C      ... STG-region as a lattice : DALT is not necessary
496: C      =====
497: C
498: C      KZMAT(*,2) is matrix material ID
499: C
500: C      else if ( LTYP(I).eq.10 ) then
501: C          KDALT(KLBZ) = LDA
502: C          CCCCCCCCCCCCCCCCCC KZMAT(KZ,2) = KLATT(IPLAT(I)+1)
503: C          LLDA = LDA
504: C          LDA0 = LDA
505: C      end if
506: C
507: C      if ( JDEBG(1).ne.0 ) then
508: C          write(IOG,7220) KINPZ(KZ), IDLAT(I),
509: C              (DALT(KK),KK=LDA0,LLDA-1)
510: C          7220 format(/1X,' ** Lattice transformation parameters',
511: C              & ' of input zone ',I5,' (lattice ',I3,')'//
512: C              & (1X,1P,10(E12.5:)))
513: C      end if
514: C
515: C      go to 140
516: C      end if
517: C      140 continue
518: C
519: C      .... LDA : STARTING POINT OF AN ARRAY NEXT TO 'DALT'
520: C

```

src/shared/lattr.f

```

521:      call RESIZE( 'DALT', LDALT, LLDA-1, 'R8', LAST )
522: C
523: C
524: C-----
525: C ..... CHECK LATTICE-CELLS CROSSING WITH LATTICE-FRAME
526: C          (HEXAGONAL LATTICES ONLY)
527: C-----
528: C
529: C
530: C      if ( JHLAT.ne.0 ) then
531: C          do 150 I = 1, IPLAT(NLATT+1) - 1
532: C              KHLAT(I) = 0
533: C          150 continue
534: C
535: C
536: C      DHEX(1:2,K) K = 1 to 6 is X-Y coordinates of corner points of
537: C      a hexagon having unit width.
538: C
539: C
540: C
541: C
542: C      3 *
543: C      *
544: C      *
545: C      *
546: C      4 *
547: C      *
548: C      *
549: C      *
550: C
551: C      DHEX(1,1) = 0.5D0
552: C      DHEX(2,1) = 0.5D0*TAN(30D0*DPAI18)
553: C      DHEX(1,2) = 0.0D0
554: C      DHEX(2,2) = 0.5D0/COS(30D0*DPAI18)
555: C      DHEX(1,3) = -DHEX(1,1)
556: C      DHEX(2,3) = DHEX(2,1)
557: C      DHEX(1,4) = -DHEX(1,1)
558: C      DHEX(2,4) = -DHEX(2,1)
559: C      DHEX(1,5) = DHEX(1,2)
560: C      DHEX(2,5) = -DHEX(2,2)
561: C      DHEX(1,6) = DHEX(1,1)
562: C      DHEX(2,6) = -DHEX(2,1)
563: C
564: C      do 210 N = 1, NLATT
565: C          if ( LTYP(N).ge.2.and.LTYP(N).le.4 ) then
566: C              DC = COS(SZLAT(2,N)*DPAI18)
567: C              DS = SIN(SZLAT(2,N)*DPAI18)
568: C              do 180 K = 1, 6
569: C                  do 170 IY = 0, NVLAT(2,N) - 1
570: C                      do 160 IX = 0, NVLAT(1,N) - 1
571: C
572: C                      IPP = IY*NVLAT(1,N) + IX
573: C                      DX = SZLAT(1,N)*(IX+IY*0.5D0+DHEX(1,K))
574: C                      DY = SZLAT(1,N)*
575: C                      & (IY*SQRT(3.0D0)*0.5D0+DHEX(2,K))
576: C                      DXX = DC*DX - DS*DY
577: C                      DYY = DS*DX + DC*DY
578: C                      DX = DXX + SZLAT(4,N)
579: C                      DY = DYY + SZLAT(5,N)
580: C
581: C          ..... HEXAGONAL FRAME .....
582: C              if ( LTYP(N).eq.2 ) then
583: C                  DD1 = (DX+SQRT(3.0D0)*DY)*0.5D0
584: C                  DD2 = (-DX+SQRT(3.0D0)*DY+SZLAT(6,N))*
585: C                  & 0.5D0

```

```

586: C                  if ( DX.lt.0.0 .or. DX.gt.SZLAT(6,N)
587: C                  & .or. DD1.lt.0.0 .or. DD1.gt.SZLAT(6,N)
588: C                  & .or. DD2.lt.0.0 .or. DD2.gt.SZLAT(6,N) )
589: C                  & KHLAT(IPLAT(N)+IPP) = 1
590: C
591: C          ..... CYLINDRICAL FRAME .....
592: C              else if ( LTYP(N).eq.3 ) then
593: C                  if ( DX*DX+DY*DY.gt.SZLAT(6,N)**2 )
594: C                  & KHLAT(IPLAT(N)+IPP) = 1
595: C
596: C          ..... RECTANGULAR FRAME .....
597: C              else if ( LTYP(N).eq.4 ) then
598: C                  if ( DX.lt.0.0 .or. DX.gt.SZLAT(6,N)
599: C                  & .or. DY.lt.0.0 .or. DY.gt.SZLAT(7,N) )
600: C                  & KHLAT(IPLAT(N)+IPP) = 1
601: C                  end if
602: C
603: C          160 continue
604: C          170 continue
605: C          180 continue
606: C              if ( NVLAT(3,N).gt.1 ) then
607: C                  do 200 IZ = 1, NVLAT(3,N) - 1
608: C                      NN = IPLAT(N) + IZ*NVLAT(4,N)
609: C                      do 190 IXY = 0, NVLAT(4,N) - 1
610: C                          KHLAT(NN+IXY) = KHLAT(IPLAT(N)+IXY)
611: C                      190 continue
612: C                  200 continue
613: C                  end if
614: C
615: C          .... PRINTOUT KHLAT ....
616: C
617: C              write(IOG,7240) IDLAT(N)
618: C              format(/1X,' === Flags for CELL & LATTICE-FRAME',
619: C              & ' crossing === LATTICE :',I5//)
620: C              call PRLATT( IOG, NVLAT(1,N), NVLAT(2,N), NVLAT(3,N),
621: C              & KHLAT(IPLAT(N)) )
622: C              end if
623: C          210 continue
624: C          end if
625: C
626: C          return
627: C
628: C
629: C-----
630: C
631: C      220 write(IMG,7260) LIMIT, LSIZ(LLDA+LDALT)
632: C      7260 format(/1X,'XXX Memory over in subroutine 'LATTR' (LIMIT=',I10,
633: C      & ' ) At least more ',I10,' words are required !!')
634: C      call PRSTOP( 1, 'MEMORY OVER.' )
635: C      stop 999
636: C
637: C      end

```

src/shared/lattyp.f

```
1:      subroutine LATTYP( LTYPE, DIREC, ILTYP )
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C PURPOSE :   Lattice type symbol <=> Lattice type number
7: C             ( DIREC   '>' : SYMBOL -> NUMBER )
8: C             '<' : SYMBOL <- NUMBER )
9: C=====
10:      character LTYPE*(*), DIREC*1
11: C
12:      if ( DIREC.eq.'>' ) then
13:      if ( LTYPE.eq.'RECT' ) then
14:          ILTYP = 1
15:      else if ( LTYPE.eq.'HEX2' ) then
16:          ILTYP = 2
17:      else if ( LTYPE.eq.'HEX3' ) then
18:          ILTYP = 3
19:      else if ( LTYPE.eq.'HEX4' ) then
20:          ILTYP = 4
21:      else if ( LTYPE.eq.'STGR' ) then
22:          ILTYP = 10
23:      else
24:          ILTYP = 0
25:      endif
26:      else if ( DIREC.eq.'<' ) then
27:      if ( ILTYP.eq.1 ) then
28:          LTYPE = 'RECT'
29:      else if ( ILTYP.eq.2 ) then
30:          LTYPE = 'HEX2'
31:      else if ( ILTYP.eq.3 ) then
32:          LTYPE = 'HEX3'
33:      else if ( ILTYP.eq.4 ) then
34:          LTYPE = 'HEX3'
35:      else if ( ILTYP.eq.10 ) then
36:          LTYPE = 'STGR'
37:      else
38:          LTYPE = ' '
39:      end if
40:      end if
41: C
42:      return
43: C
44:      end
```

src/shared/latup2.f

```

1:      subroutine LATUP2( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
2:      N      NZONE, NN, NN0, JDIR, JLS, LS,
3:      B      XX, YY, ZZ, AA, BB, CC,
4:      B      XXX0, YYY0, ZZZ0,
5:      B      AAA0, BBB0, CCC0,
6:      B      LEVL0, LZZ0, LPOS0, LCRS0,
7:      B      DBNK0, KDBNK0, MDBNK0,
8:      G      KZMAT, KDALT, DALT, NVLAT, SZLAT,
9:      G      CELSZ, IPLAT, KLATT, KSLAT, LTP,
10:     G      ICTYP, MLBZZ,
11:     W      IP )
12: C=====
13: C PURPOSE: calculate absolute coordinates of particles in lattice cell.
14: C based on LATUPW routine.
15: C (XXX0(LS(I)),YYY0(LS(I)),ZZZ0(LS(I)))
16: C ----> (XX(I),YY(I),ZZ(I)) (JLS=0)
17: C ----> (XX(LS(I)),YY(LS(I)),ZZ(LS(I))) (JLS=1)
18: C (AAA0(LS(I)),BBB0(LS(I)),CCC0(LS(I)))
19: C ----> (AA(I),BB(I),CC(I)) (JDIR=1,JLS=0)
20: C ----> (AA(LS(I)),BB(LS(I)),CC(LS(I))) (JDIR=1,JLS=1)
21: C
22: C <<Comment>>
23: C
24: C CALLED IN: CALLS: (NONE)
25: C UPDATE:
26: C 18 Aug 2003: for LTP=10 (more than 1 cells in STG-region)
27: C-----
28: C arguments ( i=input o=output w=work )
29: C
30: C i NN : number of particles to be processed
31: C i JDIR : get direction in absolute coordinates if non zero.
32: C i JLS : store the results in bank indicated by ls if non zero.
33: C i NN0 : size of the first dimension of arrays
34: C o XX(NN0),YY(NN0),ZZ(NN0) : absolute coordinates returned.
35: C o AA(NN0),BB(NN0),CC(NN0) : absolute direction returned(JDIR.ne.0).
36: C i LS(NN) : stack ( bank pointer )
37: C i XXX0(NN0),YYY0(NN0),ZZZ0(NN0) : in cell coordinates.
38: C i AAA0(NN0),BBB0(NN0),CCC0(NN0) : in cell coordinates.(JDIR.ne.0)
39: C i LEVL0(NN0),LZZ0(NN0,NEST),LPOS0(NN0,NEST),LCRS0(NN0,NEST) :
40: C lattice data.
41: C i DBNK0 : optional bank data. Currently, STG cell position
42: C is the only effective data when JSTGR.ne.0.
43: C w IP
44: C=====
45: C implicit real*8(D)
46: C
47: C integer JDEBG(*)
48: C
49: C .... absolute coordinates and directions .....
50: C
51: C real*8 XX(NN0), YY(NN0), ZZ(NN0)
52: C real*8 AA(NN0), BB(NN0), CC(NN0)
53: C
54: C .... stack ( bank pointer ) .....
55: C
56: C integer LS(NN)
57: C
58: C .... particle attributes in lattice .....
59: C
60: C real*8 XXX0(NN0), YYY0(NN0), ZZZ0(NN0)
61: C real*8 AAA0(NN0), BBB0(NN0), CCC0(NN0)
62: C integer LEVL0(NN0), LZZ0(NN0,NEST), LPOS0(NN0,NEST),
63: C & LCRS0(NN0,NEST)
64: C
65: C real*8 DBNK0(NN0,*)

```

```

66: C integer KDBNK0(0:MDBNK0)
67: C
68: C .... geometry .....
69: C
70: C real*8 DALT(*), SZLAT(8,NLATT), CELSZ(6,1)
71: C integer KZMAT(NZONE), KDALT(NLBZ), MLBZZ(NLBZ), LTP(NLATT),
72: C & ICTYP(NCELL), NVLAT(4,NLATT), IPLAT(NLATT+1), KSLAT(*),
73: C & KLATT(*)
74: C
75: C .....
76: C
77: C integer IP(NN)
78: C
79: C common /HEXROT/ DROT
80: C real*8 DROT(0:5,2)
81: C
82: C parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
83: C
84: C include 'INC/_PMLATT'
85: C include 'INC/_SFLATT'
86: C
87: C-----
88: C
89: C LEVLM = 0
90: C do 100 I = 1, NN
91: C if ( LEVL0(LS(I)).gt.LEVLM ) LEVLM = LEVL0(LS(I))
92: C 100 continue
93: C
94: C if ( JLS.eq.0 ) then
95: C
96: C *VOCL LOOP,NOVREC
97: C do 200 I = 1, NN
98: C XX(I) = XXX0(LS(I))
99: C YY(I) = YYY0(LS(I))
100: C ZZ(I) = ZZZ0(LS(I))
101: C 200 continue
102: C
103: C if ( JDIR.ne.0 ) then
104: C *VOCL LOOP,NOVREC
105: C do 210 I = 1, NN
106: C AA(I) = AAA0(LS(I))
107: C BB(I) = BBB0(LS(I))
108: C CC(I) = CCC0(LS(I))
109: C 210 continue
110: C end if
111: C
112: C do 220 LV = LEVLM, 1, -1
113: C
114: C N1 = 0
115: C *VOCL LOOP,NOVREC
116: C do 230 I = 1, NN
117: C if ( LV.le.LEVL0(LS(I)) ) then
118: C N1 = N1 + 1
119: C IP(N1) = I
120: C end if
121: C 230 continue
122: C
123: C *VOCL LOOP,NOVREC
124: C do 240 J = 1, N1
125: C I = IP(J)
126: C LZ = LZZ0(LS(I),LV)
127: C LPS = LPOS0(LS(I),LV)
128: C MLT = LATNM(KZMAT(LZ))
129: C M = KDALT(MLBZZ(LZ))
130: C

```

src/shared/latup2.f

```

131: C      .... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
132: C
133: C
134: C
135: C    SZLAT *        TYPE 1          *        TYPE 2          *        TYPE 3          *        TYPE 4
136: C -----*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
137: C    1 * CELL-SIZE(X) *                                CELL PITCH
138: C    2 *              (Y) *                          ANGLE OF CELL ARRAY
139: C    3 *              (Z) *                          CELL HEIGHT
140: C -----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
141: C    4 * ... * ORIGIN OF CELL ARRAY (X)
142: C    5 * ... * ORIGIN OF CELL ARRAY (Y)
143: C -----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
144: C             (LATTICE FRAME SIZE)
145: C -----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
146: C    6 * ... * LATTICE- * CYLINDER- * WIDTH (X)
147: C             WIDTH * RADIUS *
148: C    7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
149: C    8 * ... * ... * ... * WIDTH (Z)
150: C -----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
151: C
152: C            if ( LTYP(MLT).eq.1 ) then
153: C                IZ = LPS/NVLAT(4,MLT)
154: C                IY = (LPS-IZ*NVLAT(4,MLT))/NVLAT(1,MLT)
155: C                IX = MOD(LPS,NVLAT(1,MLT))
156: C                KSL = SZLAT(IPLAT(MLT)+LPS)
157: C                ICEL = KLATT(IPLAT(MLT)+LPS)
158: C                IDRX = KSL/100
159: C                IDRY = (KSL-IDRX*100)/10
160: C                IDRZ = MOD(KSL,10)
161: C
162: C      ... transformation to cell array coordinates ...
163: C
164: C                DSX = SZLAT(1,MLT)
165: C                DSY = SZLAT(2,MLT)
166: C                DSZ = SZLAT(3,MLT)
167: C                ISGX = 1 - 2*IDRX
168: C                ISGY = 1 - 2>IDRY
169: C                ISGZ = 1 - 2>IDRZ
170: C                DX1 = ISGX*(XX(I)-CELSZ(4,ICEL))
171: C                & + (IX+0.5D0)*DSX
172: C                DY1 = ISGY*(YY(I)-CELSZ(5,ICEL))
173: C                & + (IY+0.5D0)*DSY
174: C                DZ1 = ISGZ*(ZZ(I)-CELSZ(6,ICEL))
175: C                & + (IZ+0.5D0)*DSZ
176: C
177: C      ... transformation to upper level coordinates ...
178: C
179: C                DX1 = DX1 + DALT(M+9)
180: C                DY1 = DY1 + DALT(M+10)
181: C                DZ1 = DZ1 + DALT(M+11)
182: C                XX(I) =
183: C                & DALT(M)*DX1 + DALT(M+3)*DY1 + DALT(M+6)*DZ1
184: C                YY(I) =
185: C                & DALT(M+1)*DX1 + DALT(M+4)*DY1 + DALT(M+7)*DZ1
186: C                ZZ(I) =
187: C                & DALT(M+2)*DX1 + DALT(M+5)*DY1 + DALT(M+8)*DZ1
188: C
189: C            if ( JDIR.ne.0 ) then
190: C                DA1 = ISGX*AA(I)
191: C                DB1 = ISGY*BB(I)
192: C                DC1 = ISGZ*CC(I)
193: C                AA(I) =
194: C                & DALT(M)*DA1 + DALT(M+3)*DB1 + DALT(M+6)*DC1
195: C                BB(I) =

```

```

196:      &          DALT(M+1)*DA1 + DALT(M+4)*DB1 + DALT(M+7)*DC1
197:      CC(I) =
198:      &          DALT(M+2)*DA1 + DALT(M+5)*DB1 + DALT(M+8)*DC1
199:      end if
200:      else if ( LTYPE(MLT).ge.2.and.LTYPE(MLT).le.4. ) then
201:          IZ = LPS/NVLAT(4,MLT)
202:          IY = (LPS-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
203:          IX = MOD(LPS,NVLAT(1,MLT))
204:          KSL = KSLAT(IPLAT(MLT)+LPS)
205:          ICEL = KLATT(IPLAT(MLT)+LPS)
206:          IDRX = KSL/100
207:          IDRY = (KSL-IDRX*100) /10
208:          IDRZ = MOD(KSL,10)
209:          DSX = SZLAT(1,MLT)
210:          DSZ = SZLAT(3,MLT)
211:          DX2 = XX(I) - CELSZ(3,ICEL)
212:          DY2 = YY(I) - CELSZ(4,ICEL)
213:          DZ2 = ZZ(I) - CELSZ(5,ICEL)
214:          DX = DROT(IDRY,1)*DX2 + DROT(IDRY,2)*DY2
215:          DY = -DROT(IDRY,2)*DX2 + DROT(IDRY,1)*DY2
216:          DX2 = (1-2*IDRX)*DX + (IX+0.5D0*IY)*DSX
217:          DY2 = DY + DHRT3*IY*DSX
218:          DZ2 = (1-2*IDRZ)*DZ2 + (IZ+0.5D0)*DSZ
219:      c
220:      c      ... transformation to upper level coordinates ...
221:      c
222:          DX2 = DX2 + DALT(M+9)
223:          DY2 = DY2 + DALT(M+10)
224:          DZ2 = DZ2 + DALT(M+11)
225:          XX(I) =
226:      &          DALT(M)*DX2 + DALT(M+3)*DY2 + DALT(M+6)*DZ2
227:          YY(I) =
228:      &          DALT(M+1)*DX2 + DALT(M+4)*DY2 + DALT(M+7)*DZ2
229:          ZZ(I) =
230:      &          DALT(M+2)*DX2 + DALT(M+5)*DY2 + DALT(M+8)*DZ2
231:      c
232:      if ( JDIR.ne.0 ) then
233:          DA = AA(I)
234:          DB = BB(I)
235:          DA2 = (1-2*IDRX)*
236:      &          (DROT(IDRY,1)*DA+DROT(IDRY,2)*DB)
237:          DB2 =
238:      &          -DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
239:          DC2 = (1-2*IDRZ)*CC(I)
240:          AA(I) =
241:      &          DALT(M)*DA2 + DALT(M+3)*DB2 + DALT(M+6)*DC2
242:          BB(I) =
243:      &          DALT(M+1)*DA2 + DALT(M+4)*DB2 + DALT(M+7)*DC2
244:          CC(I) =
245:      &          DALT(M+2)*DA2 + DALT(M+5)*DB2 + DALT(M+8)*DC2
246:      end if
247:      c
248:      c      ... STGM region ....
249:      c
250:      else if ( LTYPE(MLT).eq.10.and.LPS.gt.0 ) then
251:          K5X = KDBNK0(5) + (LV-1)*3
252:          ICEL = KLATT(IPLAT(MLT)+LPS)
253:          XX(I) = XX(I) - CELSZ(1,ICEL)
254:      &          + DBNK0(LS(I),K5X)
255:          YY(I) = YY(I) - CELSZ(2,ICEL)
256:      &          + DBNK0(LS(I),K5X+1)
257:          ZZ(I) = ZZ(I) - CELSZ(3,ICEL)
258:      &          + DBNK0(LS(I),K5X+2)
259:      end if
260:      continue

```

```

261:      220      continue
262: c
263:      else
264: c
265: *VOCL LOOP,NOVREC
266:      do 300 I = 1, NN
267:          XX(LS(I)) = XXX0(LS(I))
268:          YY(LS(I)) = YYY0(LS(I))
269:          ZZ(LS(I)) = ZZZ0(LS(I))
270:      300      continue
271: c
272:      if ( JDIR.ne.0 ) then
273: *VOCL LOOP,NOVREC
274:      do 310 I = 1, NN
275:          AA(LS(I)) = AAA0(LS(I))
276:          BB(LS(I)) = BBB0(LS(I))
277:          CC(LS(I)) = CCC0(LS(I))
278:      310      continue
279:      end if
280: c
281:      do 320 LV = LEVLM, 1, -1
282: c
283:          N1 = 0
284:          do 330 I = 1, NN
285:              if ( LV.le.LEVL0(LS(I)) ) then
286:                  N1 = N1 + 1
287:                  IP(N1) = LS(I)
288:              end if
289:          330      continue
290: c
291: *VOCL LOOP,NOVREC
292:      do 340 J = 1, N1
293:          I = IP(J)
294:          LZ = LZZ0(I,LV)
295:          LPS = LPOS0(I,LV)
296:          MLT = LATNM(KZMAT(LZ))
297:          M = KDALT(MLBZZ(LZ))
298: c
299: c ..... COORDINATE TRANSFORMATION TO IN-LATTICE COORDINATES
300: c
301: c ..... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
302: c
303: c SZLAT *      TYPE 1      *      TYPE 2      *      TYPE 3      *      TYPE 4
304: c -----
305: c 1 * CELL-SIZE(X)*          CELL PITCH
306: c 2 * (Y) *                  ANGLE OF CELL ARRAY
307: c 3 * (Z) *                  CELL HEIGHT
308: c -----
309: c 4 * ... *                  ORIGIN OF CELL ARRAY (X)
310: c 5 * ... *                  ORIGIN OF CELL ARRAY (Y)
311: c -----
312: c (LATTICE FRAME SIZE)
313: c -----
314: c 6 * ... * LATTICE- * CYLINDER- * WIDTH (X)
315: c * * * WIDTH * RADIUS *
316: c 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
317: c 8 * ... * ... * ... * WIDTH (Z)
318: c -----
319: c
320:      if ( LTYPE(MLT).eq.1 ) then
321:          IZ = LPS/NVLAT(4,MLT)
322:          IY = (LPS-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
323:          IX = MOD(LPS,NVLAT(1,MLT))
324:          KSL = KSLAT(IPLAT(MLT)+LPS)
325:          ICEL = KLATT(IPLAT(MLT)+LPS)

```

```

326:      IDRX      = KSL/100
327:      IDRY      = (KSL-IDRX*100) /10
328:      IDRZ      = MOD(KSL,10)
329: c
330: c      ... transformation to cell array coordinates ...
331: c
332:      DSX      = SZLAT(1,MLT)
333:      DSY      = SZLAT(2,MLT)
334:      DSZ      = SZLAT(3,MLT)
335:      ISGX      = 1 - 2*IDRX
336:      ISGY      = 1 - 2*IDRY
337:      ISGZ      = 1 - 2*IDRZ
338:      DX1      = ISGX*(XX(I)-CELSZ(4,ICEL))
339:      &        + (IX+0.5D0)*DSX
340:      DY1      = ISGY*(YY(I)-CELSZ(5,ICEL))
341:      &        + (IY+0.5D0)*DSY
342:      DZ1      = ISGZ*(ZZ(I)-CELSZ(6,ICEL))
343:      &        + (IZ+0.5D0)*DSZ
344: c
345: c      ... transformation to upper level coordinates ...
346: c
347:      DX1      = DX1 + DALT(M+9)
348:      DY1      = DY1 + DALT(M+10)
349:      DZ1      = DZ1 + DALT(M+11)
350:      XX(I)    =
351:      &        DALT(M)*DX1 + DALT(M+3)*DY1 + DALT(M+6)*DZ1
352:      YY(I)    =
353:      &        DALT(M+1)*DX1 + DALT(M+4)*DY1 + DALT(M+7)*DZ1
354:      ZZ(I)    =
355:      &        DALT(M+2)*DX1 + DALT(M+5)*DY1 + DALT(M+8)*DZ1
356: c
357:      if ( JDIR.ne.0 ) then
358:      DA1      = ISGX*AA(I)
359:      DB1      = ISGY*BB(I)
360:      DC1      = ISGZ*CC(I)
361:      AA(I)    =
362:      &        DALT(M)*DA1 + DALT(M+3)*DB1 + DALT(M+6)*DC1
363:      BB(I)    =
364:      &        DALT(M+1)*DA1 + DALT(M+4)*DB1 + DALT(M+7)*DC1
365:      CC(I)    =
366:      &        DALT(M+2)*DA1 + DALT(M+5)*DB1 + DALT(M+8)*DC1
367:      end if
368:      else if ( LTPY(MLT).ge.2.and.LTYP(MLT).le.4. ) then
369:      IZ      = LPS/NVLAT(4,MLT)
370:      IY      = (LPS-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
371:      IX      = MOD(LPS,NVLAT(1,MLT))
372:      KSL     = KSLAT(IPLAT(MLT)+LPS)
373:      ICEL    = KLATT(IPLAT(MLT)+LPS)
374:      IDRX    = KSL/100
375:      IDRY    = (KSL-IDRX*100) /10
376:      IDRZ    = MOD(KSL,10)
377:      DSX     = SZLAT(1,MLT)
378:      DSZ     = SZLAT(3,MLT)
379:      DX2     = XX(I) - CELSZ(3,ICEL)
380:      DY2     = YY(I) - CELSZ(4,ICEL)
381:      DZ2     = ZZ(I) - CELSZ(5,ICEL)
382:      DX      = DROT(IDRY,1)*DX2 + DROT(IDRY,2)*DY2
383:      DY      = -DROT(IDRY,2)*DX2 + DROT(IDRY,1)*DY2
384:      DX2     = (1-2*IDRX)*DX + (IX+0.5D0*IY)*DSX
385:      DY2     = DY + DHRT3*IY*DSX
386:      DZ2     = (1-2*IDRZ)*DZ2 + (IZ+0.5D0)*DSZ
387: c
388: c      ... transformation to upper level coordinates ...
389: c
390:      DX2     = DX2 + DALT(M+9)

```

src/shared/latup2.f

```
391:          DY2      = DY2 + DALT(M+10)
392:          DZ2      = DZ2 + DALT(M+11)
393:          XX(I)     =
394:      &          DALT(M)*DX2 + DALT(M+3)*DY2 + DALT(M+6)*DZ2
395:          YY(I)     =
396:      &          DALT(M+1)*DX2 + DALT(M+4)*DY2 + DALT(M+7)*DZ2
397:          ZZ(I)     =
398:      &          DALT(M+2)*DX2 + DALT(M+5)*DY2 + DALT(M+8)*DZ2
399: C
400:          if ( JDIR.ne.0 ) then
401:              DA      = AA(I)
402:              DB      = BB(I)
403:              DA2      = (1-2*IDRX)*
404:      &          (DROT(IDRY,1)*DA+DROT(IDRY,2)*DB)
405:              DB2      =
406:      &          DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
407:              DC2      = (1-2*IDRZ)*CC(I)
408:              AA(I)    =
409:      &          DALT(M)*DA2 + DALT(M+3)*DB2 + DALT(M+6)*DC2
410:              BB(I)    =
411:      &          DALT(M+1)*DA2 + DALT(M+4)*DB2 + DALT(M+7)*DC2
412:              CC(I)    =
413:      &          DALT(M+2)*DA2 + DALT(M+5)*DB2 + DALT(M+8)*DC2
414:          end if
415: C
416: C ... STGM region ....
417: C
418:          else if ( LTYP(MLT).eq.10.and.LPS.gt.0 ) then
419:              K5X      = KDBNK0(5) + (LV-1)*3
420:              ICEL      = KLATT(IPLAT(MLT)+LPS)
421:              XX(I)     = XX(I) - CELSZ(1,ICEL)
422:      &              + DBNK0(I,K5X)
423:              YY(I)     = YY(I) - CELSZ(2,ICEL)
424:      &              + DBNK0(I,K5X+1)
425:              ZZ(I)     = ZZ(I) - CELSZ(3,ICEL)
426:      &              + DBNK0(I,K5X+2)
427:          end if
428:      340      continue
429:      320      continue
430: C
431:          end if
432: C
433:          return
434:          end
```


src/shared/latupw.f

```

1:      subroutine LATUPW( IOW, JDEBG, NEST, NLATT, NCELL, NLBZ,
2:      N      NZONE, NN, NN0, JDIR,
3:      B      XX      ,YY      ,ZZ      ,AA      ,BB      ,CC,
4:      B      XXX0,   YYY0,   ZZZ0,
5:      B      AAA0,   BBB0,   CCC0,
6:      B      LEVL0, LZZ0, LPOS0, LCRS0,
7:      B      DBNK0,KDBNK0,MDBNK0,
8:      G      KZMAT , KDALT, DALT , NVLAT, SZLAT, CELSZ, IPLAT,
9:      G      KLATT , KSLAT, LTYP , ICTYP,
10:     G      MLBZZ )
11: C=====
12: C  PURPOSE: Calculate absolute coordinates of particles in lattice cell.
13: C
14: C <<Comment>>
15: C  This routine should not be used for processing which requires
16: C  speed (not vectorizable).
17: C  Currently used only to get absolute coordinates of fission particles
18: C  in lattice geometry to save in fission particle file.
19: C
20: C  CALLED IN:      CALLS: (NONE)
21: C-----
22: C arguments ( i=input o=output w=work )
23: C
24: C i NN : number of particles to be processed
25: C i JDIR : get direction in absolute coordinates if non zero.
26: C i NN0 : size of the first dimension of arrays
27: C o XX(NN0),YY(NN0),ZZ(NN0) : absolute coordinates returned.
28: C o AA(NN0),BB(NN0),CC(NN0) : absolute direction returned(JDIR.ne.0).
29: C i XXX0(NN0),YYY0(NN0),ZZZ0(NN0) : in cell coordinates.
30: C i AAA0(NN0),BBB0(NN0),CCC0(NN0) : in cell coordinates.(JDIR.ne.0)
31: C i LEVL0(NN0),LZZ0(NN0,NEST),LPOS0(NN0,NEST),LCRS0(NN0,NEST) :
32: C      lattice data.
33: C i DBNK0 : optional bank data. Currently, STG cell position
34: C      is the only effective data when JSTGR.ne.0.
35: C=====
36:      implicit real*8(D)
37: C
38:      integer JDEBG(*)
39: C
40: C .... absolute coordinates and directions .....
41: C
42:      real*8 XX(NN0), YY(NN0), ZZ(NN0)
43:      real*8 AA(NN0), BB(NN0), CC(NN0)
44: C
45: C .... particle attributes in lattice .....
46: C
47:      real*8 XXX0(NN0), YYY0(NN0), ZZZ0(NN0)
48:      real*8 AAA0(NN0), BBB0(NN0), CCC0(NN0)
49:      integer LEVL0(NN0), LZZ0(NN0,NEST), LPOS0(NN0,NEST),
50:      &      LCRS0(NN0,NEST)
51: C
52:      real*8 DBNK0(NN0,*)
53:      integer KDBNK0(0:MDBNK0)
54: C
55: C .... geometry .....
56: C
57:      real*8 DALT(*), SZLAT(8,NLATT), CELSZ(6,1)
58:      integer KZMAT(NZONE), KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT),
59:      &      ICTYP(NCELL), NVLAT(4,NLATT), IPLAT(NLATT+1), KSLAT(*),
60:      &      KLATT(*)
61: C
62: C .....
63: C
64:      common /HEXROT/ DROT
65:      real*8 DROT(0:5,2)

```

```

66: C
67:      parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
68: C
69:      include 'INC/_PMLATT'
70:      include 'INC/_SFLATT'
71: C
72: C-----
73: C
74: C *VOCL LOOP,NOVREC
75:       do 100 I = 1, NN
76:         XX(I) = XXX0(I)
77:         YY(I) = YYY0(I)
78:         ZZ(I) = ZZZ0(I)
79:       100 continue
80: C
81:       if ( JDIR.ne.0 ) then
82: C *VOCL LOOP,NOVREC
83:         do 110 I = 1, NN
84:           AA(I) = AAA0(I)
85:           BB(I) = BBB0(I)
86:           CC(I) = CCC0(I)
87:         110 continue
88:       end if
89: C
90:       DXX = 0.0D0
91:       DYY = 0.0D0
92:       DZZ = 0.0D0
93:       DAA = 1.0D0
94:       DBB = 0.0D0
95:       DCC = 0.0D0
96:       do 130 I = 1, NN
97:         do 120 LV = LEVL0(I), 1, -1
98:           C
99:             LZ = LZZ0(I,LV)
100:            LPS = LPOS0(I,LV)
101:            MLT = LATNM(KZMAT(LZ))
102:            M = KDALT(MLBZZ(LZ))
103: C
104: C ..... COORDINATE TRANSFORMATION TO IN-LATTICE COORDINATES
105: C
106: C ..... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
107: C
108: C SZLAT *      TYPE 1      *      TYPE 2      *      TYPE 3      *      TYPE 4
109: C -----*-----*-----*-----*-----
110: C 1 * CELL-SIZE(X) * CELL PITCH
111: C 2 * (Y) * ANGLE OF CELL ARRAY
112: C 3 * (Z) * CELL HEIGHT
113: C -----*-----*-----*-----
114: C 4 * ... * ORIGIN OF CELL ARRAY (X)
115: C 5 * ... * ORIGIN OF CELL ARRAY (Y)
116: C -----*-----*-----*-----
117: C (LATTICE FRAME SIZE)
118: C -----*-----*-----*-----
119: C 6 * ... * LATTICE- * CYLINDER- * WIDTH (X)
120: C * * * WIDTH * RADIUS *
121: C 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
122: C 8 * ... * ... * ... * WIDTH (Z)
123: C -----*-----*-----*-----
124: C
125:       if ( LTYP(MLT).ge.1.and.LTYP(MLT).le.4 ) then
126:         IZ = LPS/NVLAT(4,MLT)
127:         IY = (LPS-IZ*NVLAT(4,MLT)) /NVLAT(1,MLT)
128:         IX = MOD(LPS,NVLAT(1,MLT))
129:         KSL = KSLAT(IPLAT(MLT)+LPS)
130:         ICEL = KLATT(IPLAT(MLT)+LPS)

```

src/shared/latupw.f

```

131:          IDRX   = KSL/100
132:          IDRY   = (KSL-IDRX*100) /10
133:          IDRZ   = MOD(KSL,10)
134: C
135: C    ... transformation to cell array coordinates ...
136: C
137:          if ( ICTYP(ICEL).eq.1 ) then
138:             DSX   = SZLAT(1,MLT)
139:             DSY   = SZLAT(2,MLT)
140:             DSZ   = SZLAT(3,MLT)
141:             ISGX  = 1 - 2*IDRX
142:             ISGY  = 1 - 2*IDRY
143:             ISGZ  = 1 - 2*IDRZ
144:             DXX   = ISGX*(XX(I)-CELSZ(4,ICEL)) + (IX+0.5D0)*DSX
145:             DYY   = ISGY*(YY(I)-CELSZ(5,ICEL)) + (IY+0.5D0)*DSY
146:             DZZ   = ISGZ*(ZZ(I)-CELSZ(6,ICEL)) + (IZ+0.5D0)*DSZ
147:             if ( JDIR.ne.0 ) then
148:                DAA = ISGX*AA(I)
149:                DBB = ISGY*BB(I)
150:                DCC = ISGZ*CC(I)
151:             end if
152:          else if ( ICTYP(ICEL).eq.2 ) then
153:             DSX   = SZLAT(1,MLT)
154:             DSZ   = SZLAT(3,MLT)
155:             DXX   = XX(I) - CELSZ(3,ICEL)
156:             DYY   = YY(I) - CELSZ(4,ICEL)
157:             DZZ   = ZZ(I) - CELSZ(5,ICEL)
158:             DX    = DROT(IDRY,1)*DXX + DROT(IDRY,2)*DYY
159:             DY    = -DROT(IDRY,2)*DXX + DROT(IDRY,1)*DYY
160:             DXX   = (1-2*IDRX)*DX + (IX+0.5D0*IY)*DSX
161:             DYY   = DY + DHRT3*IY*DSX
162:             DZZ   = (1-2*IDRZ)*DZZ + (IZ+0.5D0)*DSZ
163:             if ( JDIR.ne.0 ) then
164:                DA  = AA(I)
165:                DB  = BB(I)
166:                DAA = (1-2*IDRX)*
167:                &      (DROT(IDRY,1)*DA+DROT(IDRY,2)*DB)
168:                DBB = -DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
169:                DCC = (1-2*IDRZ)*CC(I)
170:             end if
171:          end if
172: C
173: C    ... transformation to upper level coordinates ...
174: C
175:          DXX   = DXX + DALT(M+9)
176:          DYY   = DYY + DALT(M+10)
177:          DZZ   = DZZ + DALT(M+11)
178:          XX(I) = DALT(M)*DXX + DALT(M+3)*DYY + DALT(M+6)*DZZ
179:          YY(I) = DALT(M+1)*DXX + DALT(M+4)*DYY + DALT(M+7)*DZZ
180:          ZZ(I) = DALT(M+2)*DXX + DALT(M+5)*DYY + DALT(M+8)*DZZ
181: C
182:          if ( JDIR.ne.0 ) then
183:             AA(I) = DALT(M)*DAA + DALT(M+3)*DBB + DALT(M+6)*DCC
184:             BB(I) = DALT(M+1)*DAA + DALT(M+4)*DBB + DALT(M+7)*
185:             &      DCC
186:             CC(I) = DALT(M+2)*DAA + DALT(M+5)*DBB + DALT(M+8)*
187:             &      DCC
188:          end if
189: C
190: C    ... STGM region ....
191: C
192:          else if ( LTYP(MLT).eq.10 ) then
193:             if ( LPS.gt.0 ) then
194:                K5X = KDBNK0(5) + (LV-1)*3
195:                ICEL = KLATT(IPLAT(MLT)+LPS)
196:                XX(I) = XX(I) - CELSZ(1,ICEL) + DBNK0(I,K5X)
197:                YY(I) = YY(I) - CELSZ(2,ICEL) + DBNK0(I,K5X+1)
198:                ZZ(I) = ZZ(I) - CELSZ(3,ICEL) + DBNK0(I,K5X+2)
199:            end if
200:          end if
201:          120 continue
202:          130 continue
203: C
204:          return
205:          end

```

src/shared/lbzcnt.f

```
1:      subroutine LBZCNT( NLBZ, KZMAT, NZONE )
2: C=====
3: C PURPOSE: COUNT THE NUMBER OF LATTICE-BUFFER ZONES ( -> NLBZ )
4: C CALLED IN: GEOMIN
5: C-----
6: C arguments (i=input, o=output, w=work)
7: C o NLBZ : number of lattice-buffer zones
8: C o KZMAT(NZONE) : material # for zones
9: C=====
10:      integer KZMAT(NZONE)
11: C
12:      include 'INC/_PMLATT'
13:      include 'INC/_SFLATT'
14: C
15:      NLBZ = 0
16:      do 100 KZ = 1, NZONE
17: CCCC      if ( KZMAT(KZ).ie.-1.and.KZMAT(KZ).ge.-999 ) NLBZ = NLBZ + 1
18:      if ( ISLATT(KZMAT(KZ)).or.KZMAT(KZ).eq.-999 ) NLBZ = NLBZ + 1
19: 100 continue
20:      return
21:      end
```

src/shared/lframe.f

```

1:      subroutine LFRAME( IOW, JDEBG, NBANK, NEST, NLATT, NLBZ,
2:      N      NZONE, DINF, IBP, IPZONE, IZS, IZE, DI, IFC,
3:      O      XUP, YUP, ZUP, AUP, BUP, CUP,
4:      B      LEVL, LZZ, LPOS, LCRS,
5:      G      KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT,
6:      G      KLATT, KSLAT, LTY, MLBZZ,
7:      W      X, Y, Z, AI, BI, CI, MLT )
8:      C=====
9:      C PURPOSE:  CALCULATE DISTANCES TO FRAME OF HEXAGONAL LATTICE,
10:     C           COMPARE THEM WITH THOSE TO ZONE BOUNDARY,
11:     C           AND CALCULATE COORDINATES & DIRECTIONS IN UPPER LEVEL.
12:     C           IFC = N/0 = CROSS THE FRAME/ NO CROSSING
13:     C           (N = ZONE NUMBER IN UPPER LEVEL)
14:     C CALLED IN: FLIGHT, FLIONE      CALLS:  (NONE)
15:     C
16:     C-----
17:     C arguments (i=input, o=output, w=work)
18:     C i IOW: message printout I/O unit.
19:     C i JDEBG : debugging flags
20:     C i NBANK,NEST,NLATT,NLBZ,NZONE: see discription in common definition
21:     C           file.
22:     C i IBP(NBANK) : particle bank pointers
23:     C i IPZONE(NZONE+1) : particles indicated by IBP(*) is assumed to
24:     C           be sorted by zone # (but the "zone #" may be a
25:     C           fake numbers ...)
26:     C i IZS, IZE : starting/ending "zone #" ie. IBP(IPZONE(IZS)) to
27:     C           IBP(IPZONE(IZE+1)-1) are processed in this routine.
28:     C *i IPZ1 : (obsolete)
29:     C i o DI(NBANK) : distance to zone boundary. set to that of frame
30:     C           boundary when frame boundary is nearer than zone boundary
31:     C o IFC(NBANK) : zone number in upper level when frame boundary is
32:     C           nearer than zone boundary.
33:     C o XUP,YUP,ZUP,AUP,BUP,CUP : position and direction in upper level
34:     C i LEVL, LZZ, LPOS, LCRS: lattice data bank
35:     C i KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT,
36:     C KLATT, KSLAT, LTY, MLBZZ: (lattice) geometry data
37:     C w X,Y,Z,AI,BI,CI,MLT : working array
38:     C=====
39:     C implicit real*8(D)
40:     C
41:     C integer IPZONE(NZONE+1)
42:     C integer IBP(NBANK)
43:     C
44:     C integer JDEBG(*)
45:     C
46:     C .... DISTANCE, COORDINATES & FLAG .....
47:     C
48:     C real*8 DI(NBANK), XUP(NBANK), YUP(NBANK), ZUP(NBANK)
49:     C real*8 AUP(NBANK), BUP(NBANK), CUP(NBANK)
50:     C integer IFC(NBANK)
51:     C
52:     C .... PARTICLE BANK .....
53:     C
54:     C integer LEVL(NBANK), LZZ(NBANK,NEST), LPOS(NBANK,NEST),
55:     C & LCRS(NBANK,NEST)
56:     C
57:     C .... GEOMETRY .....
58:     C
59:     C real*8 DALT(*), SZLAT(8,NLATT), CELSZ(6,1)
60:     C integer KZMAT(NZONE), KDALT(NLBZ), MLBZZ(NLBZ), LTY(NLATT),
61:     C & NVLAT(4,NLATT), IPLAT(NLATT+1), KSLAT(*), KLATT(*)
62:     C
63:     C .... WORK AREA (LENGTH = NBANK) .....
64:     C
65:     C real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),

```

```

66:     C & CI(NBANK)
67:     C integer MLT(NBANK)
68:     C
69:     C .....
70:     C
71:     C common /HEXROT/ DROT
72:     C real*8 DROT(0:5,2)
73:     C
74:     C cccc DATA DROT / 1.0D0, 0.5D0, -0.5D0, -1.0D0, -0.5D0, 0.5D0,
75:     C cccc @ 0.0, -0.866025403784438652D0, -0.866025403784438652D0, 0.0,
76:     C cccc @ 0.866025403784438652D0, 0.866025403784438652D0 /
77:     C
78:     C parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5 )
79:     C
80:     C include 'INC/_PMLATT'
81:     C include 'INC/_SFLATT'
82:     C
83:     C-----
84:     C
85:     C THIS SUBROUTINE ASSUMES THAT PARTICLES ARE ORDERED IN ASCENDING ZONE
86:     C NUMBERS (EVENT-SELECTION) OR ALL IN A ZONE (ZONE SELECTION).
87:     C IZS : SMALLEST ZONE NUMBER IZE : LARGEST-ZONE NUMBER
88:     C (IZS = IZE FOR ZONE-SELECTION)
89:     C
90:     C IBP : BANK POINTER
91:     C IPZONE(I) : HEAD POSITION OF I'TH ZONE IN PARTICLE ARRAYS
92:     C DFR(I) : DISTANCE TO FRAMES
93:     C
94:     C .....
95:     C << REVISED 2/1/1991 >> M.SASAKI
96:     C ADDED WORKING ARRAY MLT TO SAVE RESIDING LATTICE #'S FOR EACH
97:     C PARTICLE, AND A DO-LOOP WITH LATTICE # AS LOOP-COUNTER WAS ATACHED
98:     C AS THE OUTERMOST LOOP.
99:     C THIS MODIFICATION IS TO AVOID MEMORY-ACESS CONFLICTS FOR LATTICE
100:    C PARAMETERS (SZLAT) AND MAKE SPPED-UP.
101:    C .....
102:    C
103:    C
104:    C do 100 I = 1, IPZ1
105:    C IFC(I) = 0
106:    C 100 continue
107:    C
108:    C ..... CHECK LATTICE # AND SAVE IT IN MLT(I) .....
109:    C
110:    C NLL = 0
111:    C do 110 I = IPZONE(IZS), IPZONE(IZE+1) - 1
112:    C IFC(I) = 0
113:    C MLT(I) = 0
114:    C LP = IBP(I)
115:    C LV = LEVL(LP)
116:    C if ( LV.gt.0.and.LCRS(LP,LV).gt.0 ) then
117:    C NLL = NLL + 1
118:    C LZ = LZZ(LP,LV)
119:    C CCC MLT(I) = ABS(KZMAT(LZ))
120:    C MLT(I) = LATNM(KZMAT(LZ))
121:    C end if
122:    C 110 continue
123:    C if ( NLL.eq.0 ) return
124:    C
125:    C ..... OUTERMOST LOOP BY LATTICE NUMBER .....
126:    C
127:    C do 150 N = 1, NLATT
128:    C
129:    C
130:    C ..... COORDINATE TRANSFORMATION TO IN-LATTICE COORDINATES

```

```
src/shared/lframe.f
```

```

131: C
132: C ..... SZLAT : SIZE-RELATED PARAMETERS OF LATTICE
133: C
134: C SZLAT * TYPE 1 * TYPE 2 * TYPE 3 * TYPE 4
135: C -----
136: C 1 * CELL-SIZE(X) * CELL PITCH
137: C 2 * (Y) * ANGLE OF CELL ARRAY
138: C 3 * (Z) * CELL HEIGHT
139: C -----
140: C 4 * ... * ORIGIN OF CELL ARRAY (X)
141: C 5 * ... * ORIGIN OF CELL ARRAY (Y)
142: C -----
143: C (LATTICE FRAME SIZE)
144: C -----
145: C 6 * ... * LATTICE- * CYLINDER- * WIDTH (X)
146: C * * * WIDTH * RADIUS *
147: C 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
148: C 8 * ... * * * * WIDTH (Z)
149: C -----
150: C
151: C
152: C... LTYP = 2 : HEXAGONAL FRAME .....
153: C
154: C if ( LTYP(N).eq.2 ) then
155: C do 120 I = IPZONE(L2S), IPZONE(L2E+1) - 1
156: C if ( MLT(I).eq.N ) then
157: C
158: C LP = IBP(I)
159: C LV = LEVL(LP)
160: C LZ = LZZ(LP,LV)
161: C LPS = LPOS(LP,LV)
162: C M = KDALT(MLBZZ(LZ))
163: C
164: C IZ = LPS/NVLAT(4,N)
165: C IY = (LPS-IZ*NVLAT(4,N)) /NVLAT(1,N)
166: C IX = MOD(LPS,NVLAT(1,N))
167: C KSL = KSLAT(IPLAT(N)+LPS)
168: C ICEL = KLATT(IPLAT(N)+LPS)
169: C IDRX = KSL/100
170: C IDRY = (KSL-IDRX*100) /10
171: C IDRZ = MOD(KSL,10)
172: C
173: C DXX = X(I) - CELSZ(3,ICEL)
174: C DYY = Y(I) - CELSZ(4,ICEL)
175: C DZZ = Z(I) - CELSZ(5,ICEL)
176: C DX = DROT(IDRY,1)*DXX + DROT(IDRY,2)*DYY
177: C DY = -DROT(IDRY,2)*DXX + DROT(IDRY,1)*DYY
178: C DX0 = (1-2*IDRX)*DX + (IX+0.5*IY)*SZLAT(1,N)
179: C DY0 = DY + IY*SZLAT(1,N)*DHRT3
180: C
181: C... DALT12 = COS(SZLAT2), DALT(M+13) = SIN(SZLAT2) ...
182: C ( SEE SUBROUTINE 'LATTR' )
183: C
184: C ..... GATHER CONFLICTING PARAMETERS (DALT(M+12),DALT(M+13))
185: C DALT12 = DALT(M+12)
186: C DALT13 = DALT(M+13)
187: C
188: C DX = DALT12*DX0 - DALT13*DY0 + SZLAT(4,N)
189: C DY = DALT13*DX0 + DALT12*DY0 + SZLAT(5,N)
190: C DZ = (1-2*IDRZ)*DZZ + (IZ+0.5)*SZLAT(3,N)
191: C
192: C DA0 = (1-2*IDRX)*
193: C & (DROT(IDRY,1)*AI(I)+DROT(IDRY,2)*BI(I))
194: C DB0 = -DROT(IDRY,2)*AI(I) + DROT(IDRY,1)*BI(I)
195: C DA = DALT12*DA0 - DALT13*DB0

```

```

196:          DB      = DAL T13*DA0 + DAL T12*DB0
197:          DC      = (1-2*IDRZ)*CI(I)
198: C
199:          DFR      = DINF
200: C
201: C          ..... specific to LTYP(N) = 2
202: C
203:          if ( DA.ne.0.0 ) DFR      =
204: &          MIN(DFR,MAX(-DX/DA, (SZLAT(6,N)-DX)/DA))
205: CC
206:          DUV      = (DA+DROOT3*DB)*0.5
207:          if ( DUV.ne.0.0 ) then
208:            DUP      = -(DX+DROOT3*DY)*0.5
209:            DFR      =
210: &            MIN(DFR,MAX(DUP/DUV, (SZLAT(6,N)+DUP)/DUV))
211:          end if
212: CC
213:          DUV      = -DA + DROOT3*DB
214:          if ( DUV.ne.0.0 ) then
215:            DUP      = DX - DROOT3*DY
216:            DFR      =
217: &            MIN(DFR,
218: &            MAX((-SZLAT(6,N)+DUP)/DUV, (SZLAT(6,N)
219: &            +DUP)/DUV))
220:          end if
221: CC
222:          if ( DC.ne.0.0 ) DFR      =
223: &          MIN(DFR,MAX(-DZ/DC, (SZLAT(7,N)-DZ)/DC))
224: C
225: C      .... COMPARE DFR WITH DI(I) .....
226: C
227:          if ( DFR.lt.0.0 ) DFR      = 0.0
228:          if ( DFR.le.DI(I) ) then
229:            IFC(I)  = LZ
230:            DI(I)   = DFR
231: C
232: C      .... CALCULATE COORDINATES IN UPPER LEVEL .....
233: C          DX0,DY0,DZ : IN-LATTICE COORDINATES.
234: C
235:          DX0      = DX0 + DAL T(M+9)
236:          DY0      = DY0 + DAL T(M+10)
237:          DZ       = DZ + DAL T(M+11)
238:          XUP(I)   = DAL T(M)*DX0 + DAL T(M+3)*DY0 + DAL T(M+6)*
239: &          DZ
240:          YUP(I)   = DAL T(M+1)*DX0 + DAL T(M+4)*DY0 + DAL T(M+7)
241: &          *DZ
242:          ZUP(I)   = DAL T(M+2)*DX0 + DAL T(M+5)*DY0 + DAL T(M+8)
243: &          *DZ
244:          AUP(I)   = DAL T(M)*DA0 + DAL T(M+3)*DB0 + DAL T(M+6)*
245: &          DC
246:          BUP(I)   = DAL T(M+1)*DA0 + DAL T(M+4)*DB0 + DAL T(M+7)
247: &          *DC
248:          CUP(I)   = DAL T(M+2)*DA0 + DAL T(M+5)*DB0 + DAL T(M+8)
249: &          *DC
250:          end if
251:          end if
252: 120      continue
253: C
254: C
255: C.... LTYP = 3 : CYLINDRICAL FRAME .....
256: C
257:          else if ( LTYP(N).eq.3 ) then
258: C
259:          do 130 I = IPZONE(IZS), IPZONE(IZE+1) - 1
260:            if ( MLT(I).eq.N ) then

```

src/shared/lframe.f

```

261: C
262:         LP      = IBP(I)
263:         LV      = LEVL(LP)
264:         LZ      = LZZ(LP,LV)
265:         LPS     = LPOS(LP,LV)
266:         M       = KDALT(MLBZZ(LZ))
267: C
268:         IZ      = LPS/NVLAT(4,N)
269:         IY      = (LPS-IZ*NVLAT(4,N)) /NVLAT(1,N)
270:         IX      = MOD(LPS,NVLAT(1,N))
271:         KSL     = KSLAT(IPLAT(N)+LPS)
272:         ICEL    = KLATT(IPLAT(N)+LPS)
273:         IDRX    = KSL/100
274:         IDRY    = (KSL-IDRX*100) /10
275:         IDRZ    = MOD(KSL,10)
276: CC
277:         DX      = X(I) - CELSZ(3,ICEL)
278:         DYY     = Y(I) - CELSZ(4,ICEL)
279:         DZZ     = Z(I) - CELSZ(5,ICEL)
280:         DX      = DROT(IDRY,1)*DXX + DROT(IDRY,2)*DYY
281:         DY      = -DROT(IDRY,2)*DXX + DROT(IDRY,1)*DYY
282:         DX0     = (1-2*IDRX)*DX + (IX+0.5*IY)*SZLAT(1,N)
283:         DY0     = DY + IY*SZLAT(1,N)*DHRT3
284: C
285: C.... DALT12 = COS(SZLAT2), DALT(M+13) = SIN(SZLAT2) ...
286: C      ( SEE SUBROUTINE 'LATTR' )
287: C
288: C ..... GATHER CONFLICTING PARAMETERS (DALT(M+12),DALT(M+13))
289:         DALT12  = DALT(M+12)
290:         DALT13  = DALT(M+13)
291: C
292:         DX      = DALT12*DX0 - DALT13*DY0 + SZLAT(4,N)
293:         DY      = DALT13*DX0 + DALT12*DY0 + SZLAT(5,N)
294:         DZ      = (1-2*IDRZ)*DZZ + (IZ+0.5)*SZLAT(3,N)
295: C
296:         DA0     = (1-2*IDRX)*
297: &              (DROT(IDRY,1)*AI(I)+DROT(IDRY,2)*BI(I))
298:         DB0     = -DROT(IDRY,2)*AI(I) + DROT(IDRY,1)*BI(I)
299:         DA      = DALT12*DA0 - DALT13*DB0
300:         DB      = DALT13*DA0 + DALT12*DB0
301:         DC      = (1-2*IDRZ)*CI(I)
302: C
303:         DFR     = DINF
304: C
305: C .... LTYPE = 3 : CYLINDRICAL FRAME ....
306: C
307:         if ( DC.ne.0.0 ) DFR =
308: &          MIN(DFR,MAX(-DZ/DC, (SZLAT(7,N)-DZ)/DC))
309: CC
310:         DAA     = 1.0 - DC*DC
311:         DBB     = DA*DX + DB*DY
312:         DD      = DBB*DBB - DAA*(DX**2+DY**2-SZLAT(6,N)**2)
313: cccc @          DX**2-DY**2-DZ**2+SZLAT(6,N)**2+DZ**2
314:         if ( DAA.gt.0.0.and.DD.ge.0.0 ) DFR =
315: &          MIN(DFR, (-DBB+SQRT(DD))/DAA)
316: C
317: C ..... COMPARE DFR WITH DI(I) .....
318: C
319:         if ( DFR.lt.0.0 ) DFR = 0.0
320:         if ( DFR.le.DI(I) ) then
321:             IFC(I) = LZ
322:             DI(I)  = DFR
323: C
324: C ..... CALCULATE COORDINATES IN UPPER LEVEL .....
325: C         DX0,DY0,DZ : IN-LATTICE COORDINATES.

```

```

326: C
327:         DX0     = DX0 + DALT(M+9)
328:         DY0     = DY0 + DALT(M+10)
329:         DZ      = DZ + DALT(M+11)
330:         XUP(I)  = DALT(M)*DX0 + DALT(M+3)*DY0 + DALT(M+6)*
331: &              DZ
332:         YUP(I)  = DALT(M+1)*DX0 + DALT(M+4)*DY0 + DALT(M+7)
333: &              *DZ
334:         ZUP(I)  = DALT(M+2)*DX0 + DALT(M+5)*DY0 + DALT(M+8)
335: &              *DZ
336:         AUP(I)  = DALT(M)*DA0 + DALT(M+3)*DB0 + DALT(M+6)*
337: &              DC
338:         BUP(I)  = DALT(M+1)*DA0 + DALT(M+4)*DB0 + DALT(M+7)
339: &              *DC
340:         CUP(I)  = DALT(M+2)*DA0 + DALT(M+5)*DB0 + DALT(M+8)
341: &              *DC
342:         end if
343:         end if
344: 130         continue
345: C
346: C.... LTYPE = 4 : RECTANGULAR FRAME .....
347: C
348:         else if ( LTYPE(N).eq.4 ) then
349: C
350:         do 140 I = IPZONE(IZS), IPZONE(IZE+1) - 1
351:             if ( MLT(I).eq.N ) then
352: C
353:                 LP      = IBP(I)
354:                 LV      = LEVL(LP)
355:                 LZ      = LZZ(LP,LV)
356:                 LPS     = LPOS(LP,LV)
357:                 M       = KDALT(MLBZZ(LZ))
358: C
359:                 IZ      = LPS/NVLAT(4,N)
360:                 IY      = (LPS-IZ*NVLAT(4,N)) /NVLAT(1,N)
361:                 IX      = MOD(LPS,NVLAT(1,N))
362:                 KSL     = KSLAT(IPLAT(N)+LPS)
363:                 ICEL    = KLATT(IPLAT(N)+LPS)
364:                 IDRX    = KSL/100
365:                 IDRY    = (KSL-IDRX*100) /10
366:                 IDRZ    = MOD(KSL,10)
367: CC
368:                 DX      = X(I) - CELSZ(3,ICEL)
369:                 DYY     = Y(I) - CELSZ(4,ICEL)
370:                 DZZ     = Z(I) - CELSZ(5,ICEL)
371:                 DX      = DROT(IDRY,1)*DXX + DROT(IDRY,2)*DYY
372:                 DY      = -DROT(IDRY,2)*DXX + DROT(IDRY,1)*DYY
373:                 DX0     = (1-2*IDRX)*DX + (IX+0.5*IY)*SZLAT(1,N)
374:                 DY0     = DY + IY*SZLAT(1,N)*DHRT3
375: C
376: C.... DALT12 = COS(SZLAT2), DALT(M+13) = SIN(SZLAT2) ...
377: C      ( SEE SUBROUTINE 'LATTR' )
378: C
379: C ..... GATHER CONFLICTING PARAMETERS (DALT(M+12),DALT(M+13))
380:         DALT12  = DALT(M+12)
381:         DALT13  = DALT(M+13)
382: C
383:         DX      = DALT12*DX0 - DALT13*DY0 + SZLAT(4,N)
384:         DY      = DALT13*DX0 + DALT12*DY0 + SZLAT(5,N)
385:         DZ      = (1-2*IDRZ)*DZZ + (IZ+0.5)*SZLAT(3,N)
386: C
387:         DA0     = (1-2*IDRX)*
388: &              (DROT(IDRY,1)*AI(I)+DROT(IDRY,2)*BI(I))
389:         DB0     = -DROT(IDRY,2)*AI(I) + DROT(IDRY,1)*BI(I)
390:         DA      = DALT12*DA0 - DALT13*DB0

```

src/shared/lframe.f

```
391:          DB      = DALT13*DA0 + DALT12*DB0
392:          DC      = (1-2*IDRZ)*CI(I)
393: C
394:          DFR      = DINF
395: C
396: C      .... LTYP = 4 : RECTANGULAR FRAME .....
397: C
398:          if ( DA.ne.0.0 ) DFR      =
399:          &      MIN(DFR,MAX(-DX/DA,(SZLAT(6,N)-DX)/DA))
400:          if ( DB.ne.0.0 ) DFR      =
401:          &      MIN(DFR,MAX(-DY/DB,(SZLAT(7,N)-DY)/DB))
402:          if ( DC.ne.0.0 ) DFR      =
403:          &      MIN(DFR,MAX(-DZ/DC,(SZLAT(8,N)-DZ)/DC))
404: C
405: C      .... COMPARE DFR WITH DI(I) .....
406: C
407:          if ( DFR.lt.0.0 ) DFR      = 0.0
408:          if ( DFR.le.DI(I) ) then
409:              IFC(I) = LZ
410:              DI(I)  = DFR
411: C
412: C      .... CALCULATE COORDINATES IN UPPER LEVEL .....
413: C      DX0,DY0,DZ : IN-LATTICE COORDINATES.
414: C
415:          DX0      = DX0 + DALT(M+9)
416:          DY0      = DY0 + DALT(M+10)
417:          DZ       = DZ + DALT(M+11)
418:          XUP(I)   = DALT(M)*DX0 + DALT(M+3)*DY0 + DALT(M+6)*
419:          &         DZ
420:          YUP(I)   = DALT(M+1)*DX0 + DALT(M+4)*DY0 + DALT(M+7)
421:          &         *DZ
422:          ZUP(I)   = DALT(M+2)*DX0 + DALT(M+5)*DY0 + DALT(M+8)
423:          &         *DZ
424:          AUP(I)   = DALT(M)*DA0 + DALT(M+3)*DB0 + DALT(M+6)*
425:          &         DC
426:          BUP(I)   = DALT(M+1)*DA0 + DALT(M+4)*DB0 + DALT(M+7)
427:          &         *DC
428:          CUP(I)   = DALT(M+2)*DA0 + DALT(M+5)*DB0 + DALT(M+8)
429:          &         *DC
430:          end if
431:          end if
432:          140      continue
433: C
434:          end if
435: C
436:          150      continue
437:          return
438:          end
```

src/shared/lkeep.f

```

1:      subroutine LKEEP( VNAME, LX,      NL,      TYPE,  LAST,  JDEBG )
2:      C
3:      C  GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C  PURPOSE: LKEEP... RESERVE MEMORY FROM ADDRESS LX BY NL WORD
7:      C              IN ARRAY IN COMMON /LARRAY/, WHICH IS PRESERVED AS
8:      C              "TASK LOCAL" MEMORY WHEN MULTI TASKING CALCULTION
9:      C              LKEPV... LIKE 'LKEEP', MOREOVER ASSIGN VALUE V TO MEMORY
10:     C
11:     C      Almost smilar to KEEP,KEPV.
12:     C
13:     C-----
14:     C  arguments ( i = input, o = output, c =constant, w = work )
15:     C
16:     C  i  VNAME : name of data reserved etc.
17:     C  o  LX   : memory pointer as array index ( A(*) of common /ARRAY/)
18:     C  i  o  NL   : size of memory reserved ( or reset, remaining...)
19:     C  i  TYPE  : data type ( 'I4','R4','R8','C...' 'I4D','R4D' )
20:     C  o  LAST  : memory pointer of the starting address of free area
21:     C              as array index ( A(*) of common /ARRAY/)
22:     C  i  JDEBG : option for debugging printout.
23:     C  i  V     : data value with which memory area is initialized.
24:     C-----
25:     C <common variable refers or defines>
26:     C
27:     C  H(*) : in /LARRAY/
28:     C  IAINFL(3) : in /ARRAY/
29:     C-----
30:     C  CALLED IN: MAINLY IN 'WRKARY' BUT ALSO IN OTHER ROUTINES
31:     C  CALLS: IGTFGL (FUNCTION)
32:     C
33:     C  Related routines:
34:     C
35:     C      LKEPV   : keep memory with value initialization.
36:     C      LRMAIN  : check the size of remaining amount of memory
37:     C      LRSIZE  : resize a memry area already reserved by 'KEEP'.
38:     C      Lstlst  : set the last position of available memory
39:     C      Lgtlst  : get the last position of available memory
40:     C
41:     C=====
42:     C
43:     C      include 'INC/_ARRAY'
44:     C      include 'INC/_IOUNIT'
45:     C
46:     C      character VNAME*(*), TYPE*(*)
47:     C      integer JDEBG(*)
48:     C      include 'INC/_WORDL'
49:     C-----
50:     C      logical JDBND
51:     C      real*8 WSIZOF
52:     C
53:     C
54:     C      LX      = IAINFL(2)
55:     C
56:     C      LTY      = INDEX(TYPE,' ') - 1
57:     C      if ( LTY.lt.0 ) LTY = LEN(TYPE)
58:     C
59:     C      ... requires double word boundary ....
60:     C
61:     C      JDBND    = TYPE(1:2) .eq.'R8'
62:     C      if ( TYPE(LTY:LTY).eq.'D' ) then
63:     C          JDBND = .true.
64:     C          LTY    = LTY - 1
65:     C      end if

```

```

66:     C
67:     C      if ( MWORD.gt.1 ) then
68:     C          if ( JDBND.and.MOD(LX,MWORD).eq.0 ) LX  = LX + 1
69:     C      end if
70:     C
71:     C      ... TYPE = 'R8' : DOUBLE PRECISION/REAL = 32/64 BIT MACHIENS
72:     C
73:     C      if ( TYPE(:LTY).eq.'R8' ) then
74:     C          LAST  = LX + NL*WSIZOF('R8')
75:     C
76:     C      ... TYPE = 'CXX' : CHARACTER OF XX BYTES .....
77:     C
78:     C      else if ( TYPE(1:1).eq.'C' ) then
79:     C          LAST  = LX + NINT(DBLE(NL*WSIZOF(TYPE(:LTY)))+0.1D0)
80:     C
81:     C      ... TYPE = 'R4'/'I4' : REAL/INTEGER
82:     C
83:     C      else
84:     C          LAST  = LX + NL*WSIZOF(TYPE(:LTY))
85:     C      end if
86:     C
87:     C      IAINFL(2)  = LAST
88:     C
89:     C      if ( LSIZL(LAST).gt.LIMITL+1 ) then
90:     C          write(IMSG, '(//1X,A,A,A,I10,A,I10,A,A,I10,A//)')
91:     C          &      'XXX MEMORY OVER IN KEEPING <', VNAME(:ICLEN2(VNAME)),
92:     C          &      '> (LENGTH = ', LAST - LX, ' (WORD)) BY ', LSIZL(LAST)
93:     C          &      - LIMITL - 1, ' (WORD) ', ' LIMITL=', LIMITL, ' XXX'
94:     C          IKK    = 1
95:     C          call CNTERR( 'FATAL' )
96:     C          IAINFL(3) = MAX(IAINFL(3),LSIZL(LAST))
97:     C      end if
98:     C      CCCC if ( IGTFGL('JDEBG').ne.0 .or. IGTFGL('JMCHK').ne.0 )
99:     C
100:     C      if ( JDEBG(1).ne.0 .or. IGTFGL('JMCHK').ne.0 ) then
101:     C          write(IPR,7020) VNAME(:ICLEN2(VNAME)),LX,LAST-1,LSIZL(LX),
102:     C          &      LSIZL(LAST-1)
103:     C          7020 format(/1X,<','A,> AREA KEPT IN LOCAL MEMORY FROM ',I10,
104:     C          &      ' TO ',I10,','(','I10,' TO ',I10,')//)
105:     C      end if
106:     C
107:     C      return
108:     C      end
109:     C
110:     C
111:     C
112:     C=====
113:     C  LKEPV: KEEP MEMORY AND ASSIGN VALUE
114:     C=====
115:     C
116:     C      subroutine LKEPV(HH,VNAME, LX, NL, TYPE,  LAST,  V, JDEBG
117:     C          &
118:     C
119:     C      ... array HH must be the array H(*) in common /LARRAY/ or
120:     C      address pointed by CRAY type pointer in the common when
121:     C      fixed memory size mode or CRAY pointer mode. Otherwise
122:     C      it points an array dynamically allocated somewhere in the
123:     C      priogram and passed to routines only as a dummy argument.
124:     C
125:     C      real HH(*)
126:     C
127:     C      include 'INC/_ARRAY'
128:     C      include 'INC/_IOUNIT'
129:     C
130:     C      character VNAME*(*), TYPE*(*)

```


src/shared/lkeep.f

```

131:      real V
132:      integer JDEBG(*)
133: C
134:      logical JDBND
135: C
136:      include 'INC/_WORDL'
137: C
138:      character*8 BLANK
139:      data BLANK /'          '/
140: C-----
141: C
142: C
143:      call LKEEP( VNAME, LX, NL, TYPE, LAST, JDEBG )
144: C
145: C
146:      LTY = INDEX(TYPE,' ') - 1
147:      if ( LTY.lt.0 ) LTY = LEN(TYPE)
148: C
149: C      ... requires double word boundary ....
150: C
151:      JDBND = TYPE(1:2) .eq.'R8'
152:      if ( TYPE(LTY:LTY).eq.'D' ) then
153:          JDBND = .true.
154:          LTY = LTY - 1
155:      end if
156: C
157: C      .... ASSIGN VALUE TO MEMORY .....
158: C
159: C      FOR CHARACTER TYPE, BLANK INITIALIZATION ONLY.
160: C
161:      if ( LSIZL(LAST).le.LIMITL+1 ) then
162:          if ( TYPE(1:1).eq.'C' ) then
163:              write(IMG,*)
164:              & 'XXX(LKEPV) Program Error: Memory reservation of character data',
165:              & ' is not allowed (use LKEEPC instead!!) VNAME <',
166:              & VNAME(:ICLEN2(VNAME)), '>'
167:              stop 666
168: C
169: C          if ( MWORD.eq.2 ) then
170: C              do 100 L = LX, LAST - 1
171: C                  read(BLANK(:4),fmt ='(A4)') H(L)
172: C                  continue
173: C              else
174: C                  do 110 L = LX, LAST - 1
175: C                      read(BLANK(:8),fmt ='(A8)') H(L)
176: C                      continue
177: C                  end if
178: C              else if ( TYPE(:LTY).eq.'R8' ) then
179: C                  call PUTVD( H(LX), LAST-LX, V )
180: C                  call PUTVD( H(LX), NL, V )
181: C                  call PUTVD( HH(LX), NL, V )
182: C              else if ( TYPE(:LTY).eq.'R4' ) then
183: C                  call PUTV( H(LX), LAST-LX, V )
184: C                  call PUTV( H(LX), NL, V )
185: C                  call PUTV( HH(LX), NL, V )
186: C              else if ( TYPE(:LTY).eq.'I4' ) then
187: C                  call PUTVI( H(LX), LAST-LX, V )
188: C                  call PUTVI( H(LX), NL, V )
189: C                  call PUTVI( HH(LX), NL, V )
190: C              end if
191: C          end if
192: C          return
193: C
194: C
195: C

```

```

196: C=====
197: C LRMAIN: RETURN REMAINING MEMORY SIZE =====
198: C=====
199: C
200: Ccccc ENTRY REMAIN( VNAME, NL, TYPE, LAST )
201: C      subroutine LRMAIN( VNAME, NL, TYPE, LAST )
202: C
203: C          include 'INC/_ARRAY'
204: C          include 'INC/_IOUNIT'
205: C
206: C          character VNAME*(*), TYPE*(*)
207: C          include 'INC/_WORDL'
208: C-----
209: C          logical JDBND
210: C          real*8 WSIZOF
211: C
212: C
213: C          LL = IAINFL(2)
214: C          LAST = IAINFL(2)
215: C
216: C          LTY = INDEX(TYPE,' ') - 1
217: C          if ( LTY.lt.0 ) LTY = LEN(TYPE)
218: C
219: C          ... requires double word boundary ....
220: C
221: C          JDBND = TYPE(1:2) .eq.'R8'
222: C          if ( TYPE(LTY:LTY).eq.'D' ) then
223: C              JDBND = .true.
224: C              LTY = LTY - 1
225: C          end if
226: C
227: C          .... TYPE = 'R8' : DOUBLE PRECISION/REAL = 32/64 BIT MACHIENS
228: C
229: C          if ( JDBND ) then
230: C              if ( MWORD.gt.1.and.MOD(LL,MWORD).eq.0 ) LL = LL + 1
231: C          end if
232: C
233: C          AM = WSIZOF(TYPE(:LTY))
234: C          NL = (LIMITL-LL+1) /AM
235: C          NL = (LIMITL-LL+1) /WSIZOF(TYPE(:LTY))
236: C          NL = (LIMITL-LSIZL(LL)+1) /WSIZOF(TYPE(:LTY))
237: C
238: C          return
239: C          end
240: C
241: C=====
242: C LRSIZE:
243: C      ===== RESIZE ARRAY AND GET LAST POINTER =====
244: C      ( DO NOT CALL THIS ENTRY FOR VARIABLE ARRAYS OF
245: C          NON-HIGHEST ADDRESSES. )
246: C=====
247: C
248: C      subroutine LRSIZE( VNAME, LX, NL, TYPE, LAST )
249: C
250: C          include 'INC/_ARRAY'
251: C          include 'INC/_IOUNIT'
252: C
253: C          character VNAME*(*), TYPE*(*)
254: C          include 'INC/_WORDL'
255: C          real*8 WSIZOF
256: C-----
257: C          LTY = INDEX(TYPE,' ') - 1
258: C          if ( LTY.lt.0 ) LTY = LEN(TYPE)
259: C
260: C          ... requires double word boundary ....

```

src/shared/lkeep.f

```
261: C
262:   if ( TYPE(LTY:LTY).eq.'D' ) then
263:     LTY      = LTY - 1
264:   end if
265:   if ( TYPE(1:1).eq.'C' ) then
266:     LAST      = LX + NINT(DBLE(NL*WSIZOF(TYPE(:LTY)))+0.1D0)
267:   else
268:     LAST      = LX + NL*WSIZOF(TYPE(:LTY))
269:   end if
270: C
271:   IAINFL(2)   = LAST
272:   return
273: end
274:
275: C
276: C=====
277: C LSTLST: SET LAST POINTER to LX =====
278: C=====
279: C
280: C ( used to discard temporary data . etc. )
281: C
282:   subroutine LSTLST( VNAME, LX,   LAST )
283: C
284:   include 'INC/_ARRAY'
285:   character VNAME*(*)
286: C-----
287: C
288:   IAINFL(2)   = LX
289:   LAST        = IAINFL(2)
290:   return
291: end
292:
293: C
294: C=====
295: C LGTLST: GET LAST POINTER  =====
296: C=====
297: C
298:   subroutine LGTLST( LAST )
299: C
300:   include 'INC/_ARRAY'
301: C
302: C-----
303: C
304:   LAST        = IAINFL(2)
305:
306:   end
307: C
308: C=====
309: C LSTCKL: Check whether memory over error has occurred or not.
310: C
311:   ex.
312:   if ( LSTCKL(0).lt.0 ) then
313:     write(*,*) '== Memory over error has occurred.'
314:   end if
315: C=====
316: C
317:   function LSTCKL( LJ )
318: C
319:   include 'INC/_ARRAY'
320: C
321: C-----
322:   LSTCKL      = LIMITL + 1 - IAINFL(3)
323:   return
324: end
```

src/shared/lsiz.f

```
1:      function LSIZ(LINDEX)
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C   PURPOSE: array index handling for dynamically used memory area
7: C       starts in A(1) in common /ARRAY/ or H(1) in common /LARRAY/.
8: C
9: C       Real starting array index IAINF(1)/IAONFL(1) may not be 1
10: C       if dynamic allocation without "POINTER" syntax is required
11: C       and starting address of allocated area is pointed as relative
12: C       index to A(0) or H(0).
13: C
14: C   LSIZ( LINDEX ) returns really used memory size (word)
15: C       for array index LINDEX for array A(*) in common /ARRAY/.
16: C   LSIZL( LINDEX ) works similarly for task local memory.
17: C   LSIZC( LINDEX ) works similarly for task shared character memory.
18: C
19: C   ILSIZ( I ) returns array index of I'th element of
20: C       really used part of A(*)
21: C   ILSIZL( I ) works similarly for task local memory.
22: C
23: C   LIMGT() return size limit of A(*)
24: C   LIMGTL() return size limit of H(*)
25: C   LIMGTC() return size limit of H(*)
26: C
27: C-----
28: C argument (i=input, o=output, w=work)
29: C
30: C i   LINDEX : array index for A(*) ( LSIZL : for H(*) )
31: C-----
32: C <common variable refers or defines>
33: C
34: C   a(*) : in /ARRAY/
35: C   iainf(3) : in /ARRAY/
36: C-----
37:       include 'INC/_ARRAY'
38:       LSIZ = LINDEX - IAINF(1) + 1
39:       return
40: end
41: C
42:       function LSIZL(LINDEX)
43:       include 'INC/_ARRAY'
44:       LSIZL = LINDEX - IAINFL(1) + 1
45:       return
46: end
47: C
48:       function LSIZC(LINDEX)
49:       include 'INC/_ARRAY'
50:       LSIZC = LINDEX - IAINFC(1) + 1
51:       return
52: end
53: C
54: C
55: C
56:       function ILSIZ(I)
57:       include 'INC/_ARRAY'
58:       ILSIZ = IAINF(1) + I - 1
59:       return
60: end
61: C
62:       function ILSIZL(I)
63:       include 'INC/_ARRAY'
64:       ILSIZL = IAINFL(1) + I - 1
65:       return
66:
67: C
68: C
69: C
70:       function LIMGT()
71:       include 'INC/_ARRAY'
72:       LIMGT = LIMIT
73:       return
74: end
75: C
76:       function LIMGTL()
77:       include 'INC/_ARRAY'
78:       LIMGTL = LIMITL
79:       return
80: end
81: C
82:       function LIMGTC()
83:       include 'INC/_ARRAY'
84:       LIMGTC = LIMITC
85:       return
86: end
```

src/shared/lthrgy.f

```
1:      subroutine LTHRGY( E,      DLTH,  E0,    N )
2: C=====
3: C purpose: create equi-lethargy and decreasing data array.
4: C-----
5: C arguments ( i=input, o=output )
6: C o E(N) : array on which equi-lethargy data are created.
7: C i DLTH : lethargy width (positive)
8: C i E0   : start value
9: C i N    : number of data
10: C-----
11:      real E(N)
12: C
13:      do 100 I = 1, N
14:          E(I) = E0*EXP(-REAL(I-1)*DLTH)
15:      100 continue
16:      return
17:      end
```

src/shared/memalc.f

```

1:      subroutine MEMALC( IOUT, LSTART,LSTRTL,IERR )
2: C=====
3: C  PURPOSE: CHECK MEMORY SIZE (LIMIT) OF COMMON /ARRAY/
4: C           OR ALLOCATE MEMORY IN DYNAMIC ALLOCATION MODE.
5: C           RETURNS REAL STARTING POSITION OF DYNAMIC MEMORY AS LSTART.
6: C
7: C  CALLED IN: INTRO
8: C  CALLS: CNTERR
9: C-----
10: C Arguments  ( i=input, o=output, c=constant, w=work )
11: C  i iout : message printout I/O unit
12: C  o lstart: starting position array index of task shared memory.
13: C  o lstrtl: starting position array index of task local memory.
14: C  o ierr  : error code ( 0 = no error, non zero = error )
15: C-----
16: C/#IF CUTIL.AND.MS_VISUAL
17: C
18: C ... This interface block is for CUTIL & MS-Visual tools.
19: C
20: C  interface
21: C    subroutine ADTOIDX( A, LIMIT, LSTART, JERR )
22: C      real    A
23: C      integer LIMIT, LSTART, JERR
24: C*DEC$  ATTRIBUTES C :: ADTOIDX
25: C*DEC$  ATTRIBUTES REFERENCE :: A, LIMIT, LSTART, JERR
26: C    end subroutine
27: C  end interface
28: C/#ENDIF
29: C
30: C  include 'INC/_ARRAY'
31: C  include 'INC/_WORDL'
32: C
33: C  include 'INC/_IOUNIT'
34: C
35: C/#IF ADRSIZE(64)
36: C    integer*8 LBYTE
37: C    integer*8 MALLOC
38: C/#  IF MALLOC64
39: C    integer*8 MALLOC64
40: C/#  ENDIF
41: C/#ENDIF
42: C/#IF MOD(STRICT)
43: C    integer*8 LPOS
44: C/#ENDIF
45: C/#IF INTEGER8
46: C    integer*8 INT8
47: C/#ELSE
48: C    integer  INT8
49: C/#ENDIF
50: C    external  INT8
51: C
52: C ... default memory size in word in dynamic allocation ...
53: C    ( 1 Mword )
54: C
55: C    parameter ( MDFLT = 104 8576 )
56: C
57: C-----
58: C
59: C ... LWORD : byte length of single precision variables.
60: C
61: C    LWORD = 8 / MWORD
62: C
63: C    IERR = 0
64: C
65: C/#IF DYNAMIC

```

```

66:
67:      write(IOUT,
68: & '(/lx, ' == MEMORY FOR VARIABLE-SIZED DATA IS ALLOCATED ',
69: & ' 'DYNAMICALLY =='/)')
70:
71: C/#  IF .NOT.NOPOINTER
72: C
73: C
74: C ... FOR DYNAMIC ALLOCATION MODE (CRAY/ HP9000 / SUN / NEC-SX ) ....
75: C
76: C
77: C    MEMORY SIZE (WORD) IS SPECIFIED BY 'DYNAMIC-MEMORY' OPTION.
78: C
79: C
80: C/#  IF PARA( CRAY* SX* )
81: C
82: C    if( LIMIT.gt.0 .and. LIMITL.eq.0 ) then
83: C      LIMITL = 0.3D0 * LIMIT
84: C      LIMIT = LIMIT - LIMITL
85: C      write(IMG,*) '!!! NO SPECIFICATION FOR TASK LOCAL MEMORY SIZE'
86: C      write(IMG,*) ' SIZE PARAMETER GIVEN AS TASK SHARED MEMORY',
87: C & ' IS SPLIT IN TWO PARTS BY DEFALULT RATIO.'
88: C      write(IMG,*)
89: C & ' TASK SHARED : ',limit,' WORDS ',
90: C & ' TASK LOCAL  : ',limitl,' WORDS '
91: C      call CNTERR('WARNING')
92: C    end if
93: C
94: C/#  ENDIF
95: C == For task shared memory ==
96: C
97: C    if( LIMIT .eq. 0 ) then
98: C      LIMIT = MDFLT
99: C    end if
100: C
101: C .... lpa is connected to A(1) by pointer statement ....
102: C
103: C    LBYTE = LWORD * LIMIT
104: C
105: C/#IF LFMALLOC
106: C    LPA = LFMALLOC ( LBYTE )
107: C/#ELSEIF SYSTEM(CRAY*)
108: C    call HPALLOC( LPA, LIMIT, JLERR, 0)
109: C    if ( JLERR.lt.0 ) then
110: C      write(IMG, '(' '!!! MEMORY ALLOCATION ERROR ON "CRAY" TYPE',
111: C & ' "HPALLOC" ROUTINE. CODE=' ',i5)') JLERR
112: C    LPA = 0
113: C    end if
114: C/#ELSEIF SYSTEM(SX*)
115: C/#  IF ADRSIZE(64)
116: C    call MALLOCW2( INT8(LIMIT), LPA )
117: C/#  ELSE
118: C    call MALLOCW( LIMIT, LPA )
119: C/#  ENDIF
120: C/#ELSEIF MALLOC64
121: C    LPA = MALLOC64( LBYTE )
122: C/#ELSE
123: C    LPA = MALLOC ( LBYTE )
124: C/#ENDIF
125: C
126: C    if( LPA .eq. 0 ) then
127: C      write(IMG, '(' '!!! FAILED TO ALLOCATE MEMORY DYNAMICALLY',
128: C & ' WITH SPECIFIED SIZE (' ',i12,' WORDS)') ) LIMIT
129: C      call CNTERR('WARNING')
130: C      IERR = 1

```

src/shared/memalc.f

```

131: c
132: c ... try to allocate default size ...
133: c
134:         LIMIT = MDFLT
135:         LBYTE = LWORD * LIMIT
136: C/#IF LFMALLOC
137: *         LPA = LFMALLOC ( LBYTE )
138: C/#ELSEIF SYSTEM(CRAY*)
139: *         call HPALLOC( LPA, LIMIT, JLERR, 0)
140: *         if ( JLERR.lt.0 ) then
141: *             WRITE(IMG,('!!! MEMORY ALLOCATION ERROR ON "CRAY" TYPE'',
142: * & ' "HPALLOC" ROUTINE. CODE=','i5')) JLERR
143: *             LPA = 0
144: *         end if
145: C/#ELSEIF SYSTEM(SX*)
146: C/# IF ADRSIZE(64)
147: *         call MALLOCW2( INT8(LIMIT), LPA )
148: C/# ELSE
149: *         call MALLOCW( LIMIT, LPA )
150: C/# ENDIF
151: C/#ELSEIF MALLOC64
152: *         LPA = MALLOC64( LBYTE )
153: C/#ELSE
154: *         LPA = MALLOC ( LBYTE )
155: C/#ENDIF
156:         if( LPA .eq. 0 ) then
157:             write(IMG,*)
158:             & 'XXX TRIED TO ALLOCATE MEMORY OF DEFAULT SIZE (',
159:             & lbyte,') BUT FAILED. XXX'
160:             call PRSTOP(1,'MEMORY ALLOCATION ERROR.')
161:             stop 999
162:         end if
163:         end if
164:         write(IOUT,(/5x,A,I20,' ( ',Z16,' )'))
165:         & ' STARTING ADDRESS OF DYNAMIC MEMORY ',LPA,LPA
166: c
167:         LSTART = 1
168: c
169: C/#IF PARA( CRAY* SX* )
170: C
171: C === For task local memory ===
172: C
173: C
174: *         if( LIMITL .eq. 0 ) then
175: *             LIMITL = MDFLT
176: *         end if
177: c
178: C .... lph is connected to A(1) by pointer statement ....
179: C
180: *         LBYTE = LWORD * LIMITL
181: C/#IF LFMALLOC
182: *         LPH = LFMALLOC ( LBYTE )
183: C/#ELSEIF SYSTEM(CRAY*)
184: *         call HPALLOC( LPH, LIMITL, JLERR, 0)
185: *         if ( JLERR.lt.0 ) then
186: *             write(IMG,('!!! MEMORY ALLOCATION ERROR ON "CRAY" TYPE'',
187: * & ' "HPALLOC" ROUTINE. CODE=','i5')) JLERR
188: *             LPH = 0
189: *         end if
190: C/#ELSEIF SYSTEM(SX*)
191: C/# IF ADRSIZE(64)
192: *         call MALLOCW2( INT8(LIMITL), LPH )
193: C/# ELSE
194: *         call MALLOCW( LIMITL, LPH )
195: C/# ENDIF

```

```

196: C/#ELSE
197: *         LPH = MALLOC ( LBYTE )
198: C/#ENDIF
199: *
200: *         if( LPH .eq. 0 ) then
201: *             WRITE(IMG,
202: * & ' ( '!!! FAILED TO ALLOCATE TASK LOCAL MEMORY DYNAMICALLY'
203: * & ' , WITH SPECIFIED SIZE ( ',I12,' WORDS)' ) limitl
204: *             call CNTERR('WARNING')
205: *             IERR = 1
206: c
207: c ... try to allocate default size ...
208: c
209: *         LIMITL = MDFLT
210: *         LBYTE = LWORD * LIMITL
211: C/#IF LFMALLOC
212: *         LPH = LFMALLOC ( LBYTE )
213: C/#ELSEIF SYSTEM(CRAY*)
214: *         call HPALLOC( LPH, LIMITL, JLERR, 0)
215: *         if ( JLERR.lt.0 ) then
216: *             write(IMG,('!!! MEMORY ALLOCATION ERROR ON "CRAY" TYPE'',
217: * & ' "HPALLOC" ROUTINE. CODE=','i5')) JLERR
218: *             LPH = 0
219: *         end if
220: C/#ELSEIF SYSTEM(SX*)
221: C/# IF ADRSIZE(64)
222: *         call MALLOCW2( INT8(LIMITL), LPH )
223: C/# ELSE
224: *         call MALLOCW( LIMITL, LPH )
225: C/# ENDIF
226: C/#ELSE
227: *         LPH = MALLOC ( LBYTE )
228: C/#ENDIF
229: *         if( LPH .eq. 0 ) then
230: *             write(IMG,*)
231: *             & 'XXX TRIED TO ALLOCATE',
232: *             & ' TASK LOCAL MEMORY OF DEFAULT SIZE (',
233: *             & lbyte,') BUT FAILED. XXX'
234: *             call PRSTOP(1,'TASK LOCAL MEMORY ALLOCATION ERROR.')
235: *             stop 999
236: *         end if
237: *         end if
238: *         write(IOUT,(/5x,a,i20,' ( ',z16,' )'))
239: *         & ' STARTING ADDRESS OF TASK LOCAL DYNAMIC MEMORY ',LPH,LPH
240: c
241: *         LSTRTL = 1
242: C/#ELSE
243: *         LSTRTL = LSTART
244: C/#ENDIF
245: *         IAINFL(1) = LSTRTL
246: *         IAINFL(3) = 0
247: c
248: C/#IF WORD(32)
249: C
250: C ... A(LSTART) must be on a double word boundary ...
251: C
252: C/# IF MOD(STRICT)
253: *         LPOS = LOC( A(LSTART) )
254: *         IRES = LPOS - INT8(LPOS/8)*8
255: *         if( IRES .ne. 0 ) LSTART = LSTART+1
256: C/# ELSE
257: *         if( MOD( LOC( A(LSTART) ), 8 ) .ne. 0 ) LSTART = LSTART+1
258: C/# ENDIF
259: C/# IF PARA( SX* )
260: *         if( MOD( LOC( H(LSTRTL) ), 8 ) .ne. 0 ) LSTRTL = LSTRTL+1

```

src/shared/memalc.f

```

261: C/# ENDIF
262: C/#ENDIF
263:
264: C/# ELSE
265: C
266: C ... dynamic memory allocation for compilers not having pointer.
267: C
268: C === For task shared memory ===
269: C
270:     if( LIMIT .eq. 0 ) then
271:         LIMIT = MDFLT
272:     endif
273: C
274:     call ADTOIDX( A(1), LIMIT, LSTART, JERR )
275:
276:     if( JERR .ne. 0 ) then
277:         write(IMG, '(//!!! FAILED TO ALLOCATE MEMORY DYNAMICALLY'
278:         & ' WITH SPECIFIED SIZE (',I2,' WORDS)')') limit
279:         call CNTERR('WARNING')
280:         IERR = 1
281: C
282: C ... try to allocate default size ...
283: C
284:         LIMIT = MDFLT
285:         call ADTOIDX( A(1), LIMIT, LSTART, JERR )
286:         if( JERR .ne. 0 ) then
287:             write(IMG,*)
288:             & 'XXX TRIED TO ALLOCATE MEMORY OF DEFAULT SIZE (',
289:             & ' lbyte,') BUT FAILED. XXX'
290:             call PRSTOP(1,'MEMORY ALLOCATION ERROR.')
291:             stop 999
292:         end if
293:     end if
294:     write(IOUT, '(//5X,A,I16)')
295:     & ' STARTING ARRAY INDEX OF DYNAMIC MEMORY : ',lstart
296:     & ' LAST ARRAY INDEX OF DYNAMIC MEMORY : ',lstart+limit-1
297:
298:     LSTRTL = LSTART
299:     IAINFL(1) = LSTRTL
300:     IAINFL(3) = 0
301:
302: C/# ENDIF
303:
304: C/#ELSE
305: C
306: C NON-DYNAMIC MEMORY ALLOCATION:
307: C ASSUMES MEMORY WORD SIZE ( LIMIT IN COMMON /ARRAY/ or
308: C LIMITL IN COMMON /LARRAY/ ) IS
309: C ALREADY DETERMINED.
310: C
311: * write(IOUT,
312: * & '/1X,' ' === MEMORY FOR VARIABLE-SIZED DATA IS FIXED SIZE ' ',
313: * & ' ' ==='/'')
314: *
315: * LSTART = 1
316: * LSTRTL = 1
317: C/#ENDIF
318: C
319:     write(IOUT, '(//10X,' '***** MEMORY FOR VARIABLE-SIZED DATA : ' ',
320:     & ' I12,' ' WORDS (',I2,' BYTES/WORD )'/' /
321:     & ' 15X,' '( ' ',F10.2,' ' MEGA-BYTES )'/'/'')
322:     & LIMIT , LWORD , DBLE( LIMIT ) / 2**20 * LWORD
323:
324: C/#IF PARA( CRAY* SX* )
325: * write(IOUT,

```

```

326: * & ' (/10X,' '***** LOCAL MEMORY FOR VARIABLE-SIZED DATA : ' ',
327: * & ' I12,' ' WORDS (',I2,' BYTES/WORD )'/' /
328: * & ' 15X,' '( ' ',F10.2,' ' MEGA-BYTES )'/'/'')
329: * & LIMITL, LWORD , DBLE( LIMITL ) / 2**20 * LWORD
330: C/#ENDIF
331: C
332:     if( LIMIT .le. 0 ) then
333:         LIMIT = 0
334:         write(IMG, '(//10X,' 'XXX MEMORY SIZE FOR VARIABLE-SIZED DATA' ',
335:         & ' ' IS INVALID.'')')
336:         call PRSTOP(1,'MEMORY ALLOCATION ERROR.')
337:         stop 999
338:     end if
339:
340: C/#IF PARA( CRAY* SX* )
341: * if( LIMITL .le. 0 ) then
342: * LIMITL = 0
343: * write( IMG,
344: * & ' (/10X,' 'XXX LOCAL MEMORY SIZE FOR VARIABLE SIZED DATA' ',
345: * & ' ' IS INVALID.'')')
346: * call PRSTOP(1,'LOCAL MEMORY ALLOCATION ERROR.')
347: * stop 999
348: * end if
349: C/#ENDIF
350: C
351:     IAINF(1) = LSTART
352:     IAINFL(1) = LSTRTL
353:     IAINF(3) = 0
354:     IAINFL(3) = 0
355:     call STLAST('MEMALC',IAINF(1),LAST)
356:     call LSTLST('MEMALC',IAINFL(1),LAST)
357: C
358: C/#IF PARA( SX* CRAY* )
359: C
360: C ... clear task local memory ( ouch !! ) ....
361: C
362: * call PUTV(H(IAINFL(1)),LIMITL,0.0)
363: C/#ENDIF
364: return
365: end

```

src/shared/memerr.f

```
1:      subroutine MEMERR( SUBNAM,MESSG, NDATA, LIMIT )
2: C
3: C   JAERI MONTE CALRO CODE UTILITY
4: C
5: C=====
6: C PURPOSE:  ISSUE ERROR MESSAGE FOR MEMORY OVERFLOW ERRORS.
7: C-----
8: C arguments (i=input, o=output, w=work)
9: C i SUBNAM : subroutine name or other "name" passed from caller.
10: C i MESSG : message string
11: C i NDATA : memory word size necessary
12: C i LIMIT : current memory size limit
13: C-----
14: C
15: C   SUBROUTINE NAME CALLED
16: C
17: C   character*(*) SUBNAM
18: C
19: C   MESSAGE
20: C
21: C   character*(*) MESSG
22: C
23: C   include 'INC/_IUNIT'
24: C
25: C-----
26: C
27: C   write(IMG,7000) SUBNAM
28: 7000 format(//4X,'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'/4X,
29: &          ' MEMORY INSUFFICIENT (SUBROUTINE <'A,> )'/4X,
30: &          'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'/)
31: C
32: C
33: C   write(IMG,7020) MESSG
34: 7020 format(/1X,3X,' MESSAGE: ',A/)
35: C
36: C
37: C   write(IMG,7040) NDATA, LIMIT
38: 7040 format(1X,3X,' RECOVERY: '/1X,3X,
39: &          ' YOU MUST SET MEMORY SIZE OF COMMON/ARRAY/ ',
40: &          'AT LEAST MORE THAN ',I10,' WORDS '/1X,3X,
41: &          ' (LIMIT = ',I10,' WORDS NOW ) !!!! STOP ')
42: C
43: C   call PRSTOP( 1, 'MEMORY OVER.' )
44: C   stop 999
45: C   end
```


src/shared/memmem.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine MEMMEM( IOW,  KMEMO, MEMC,  MEMZ,  NZONE, NMEMO,
3:     &                NBATCH )
4: C=====
5: C PURPOSE:  OPTIMIZATION OF KMEMO ARRAY
6: C CALLED IN: ACTION      CALLS: (NONE)
7: C-----
8: C arguments (i=input, o=output, w=work)
9: C i IOW : printout I/O unit
10: C o KMEMO(NZONE,NMEMO) : next zone memory
11: C o MEMC(NZONE,NMEMO) : counter of next zone memory
12: C i MEMZ(NZONE) : number of memorized zones
13: C i NBATCH : current batch number
14: C=====
15:   integer MEMC(NZONE,NMEMO), MEMZ(NZONE), KMEMO(NZONE,NMEMO)
16:   include '../shared/inc/_TASKDT'
17: C
18:   if ( NBATCH.eq.0 ) return
19: C/
20: C/#IF PARA(SX* CRAY)
21: *   call MVPSYNC_LOCK( 2 )
22: C/#ENDIF
23:
24:   write(IOW,7000)
25: 7000 format(/1X,' == OPTIMIZATION OF NEXT ZONE MEMORY (KMEMO) == ')
26:
27: C/#IF PARA(SX* CRAY)
28: *   call MVPSYNC_UNLOCK( 2 )
29: C/#ENDIF
30:
31:   call CPUTM( T0 )
32:   MMM      = 0
33:   MTT      = 0
34:   do 120 NK = 1, NZONE
35:     MMM     = MAX(MMM,MEMZ(NK))
36:     MTT     = MTT + MEMZ(NK)
37:     II      = 0
38:     do 110 K = 1, MEMZ(NK)
39:       do 100 L = 1, MEMZ(NK) - 1
40:         if ( MEMC(NK,L).lt.MEMC(NK,L+1) ) then
41:           M      = KMEMO(NK,L)
42:           KMEMO(NK,L) = KMEMO(NK,L+1)
43:           KMEMO(NK,L+1) = M
44:           M      = MEMC(NK,L)
45:           MEMC(NK,L) = MEMC(NK,L+1)
46:           MEMC(NK,L+1) = M
47:           II     = 1
48:         end if
49:       100   continue
50:     if ( II.eq.0 ) go to 120
51:     110   continue
52:   120 continue
53: C
54: C/#IF PARA(SX* CRAY)
55: *   call MVPSYNC_LOCK( 2 )
56: C/#ENDIF
57:
58:   130 write(IOW,'(1X ,'' MAX OF USED KMEMO DATA = '' ,I5,
59:     & '' UTILIZATION RATE OF KMEMO = '' ,F8.3,''' % '' )'
60:     & ) MMM, (MTT*100.0) / (NZONE*NMEMO)
61:
62: C/#IF PARA(SX* CRAY)
63: *   call MVPSYNC_UNLOCK( 2 )
64: C/#ENDIF
65:

```

```

66: CC   CALL CLOCK(T1,0,1)
67: CC   WRITE(IOW,*) '      (CPU = ',T1-T0,', (SEC) ) '
68:      return
69:      end

```

src/shared/memuse.f

```

1:      subroutine MEMUSE( ITEM, NSHARD,NLOCAL, LIMIT, LIMITL, LIMITC )
2: C=====
3: C purpose: register and output memory utilization
4: C-----
5: C arguments (i=input, o=output, w=work):
6: C i item : memory item name. If blank, print memory utilization summary.
7: C          and nshard & nlocal should be the total size.
8: C i nshard : task-shared memory size (word).
9: C i nlocal : task-local memory size (word).
10: C i LIMIT : size of dynamic memory array (task shared)
11: C i LIMITL : size of dynamic memory array (task local)
12: C i LIMITC : size of dynamic character memory array (task shared)
13: C=====
14:      character*(*) ITEM
15: CCCC include 'INC/_ARRAY'
16:      include 'INC/_IOUNIT'
17:      include 'INC/_WORDL'
18: C
19: C ... unsaved local data
20: C
21:      character*64 CWRK
22: C
23: C ... saved local data
24: C
25:      parameter( MXITEM = 16 )
26:      character*32 ITEMS(MXITEM)
27:      integer KSHARD(MXITEM)
28:      integer KLOCAL(MXITEM)
29: C
30:      data NITEM /0/
31:      data ITEMS /MXITEM*' '/
32:      data KSHARD /MXITEM*0/
33:      data KLOCAL /MXITEM*0/
34: C
35: C-----
36: C
37:      if ( ITEM.ne.' ' ) then
38:          if ( NITEM.lt.MXITEM ) then
39:              NITEM = NITEM + 1
40:              ITEMS(NITEM) = ITEM
41:              KSHARD(NITEM) = NSHARD
42:              KLOCAL(NITEM) = NLOCAL
43:          else
44:              write(IMG,*)
45:              & '!!!(MEMUSE) tried to register too many memory usage items'
46:              & ' (MXITEM=', MXITEM, ' )'
47:              call CNTERR( 'WARNING' )
48:          end if
49: C
50: C .... output summary when called with a blank "ITEM"
51: C
52:      else
53: C
54:          call LABEL( IPR, 'MEMORY UTILIZATION SUMMARY' )
55: C
56: C .... Non Character Memory
57: C
58:          KTOTS = 0
59:          KTOTL = 0
60:          do 100 I=1, NITEM
61:              KTOTS = KTOTS + KSHARD(I)
62:              KTOTL = KTOTL + KLOCAL(I)
63: 100      continue
64: C
65:          KTTS = NSHARD

```

```

66:          KTTL = NLOCAL
67: C
68:          if ( NITEM.lt.MXITEM ) then
69:              NITEM = NITEM + 1
70:              ITEMS(NITEM) = 'Others'
71:              KSHARD(NITEM) = KTTS - KTOTS
72:              KLOCAL(NITEM) = KTTL - KTOTL
73:          end if
74: C
75:          MM = 0
76:          do 110 I = 1, NITEM
77:              MM = MAX(MM, ICLEN2(ITEMS(I)))
78: 110      continue
79: C
80:          CWRK = 'Item'
81:          write(IPR,7000) CWRK(:MM)
82:          format(2X,A,5X,'Task shared (word) ',5X,'Task local (word)'/)
83: C
84:          do 120 I = 1, NITEM
85:              CWRK = ITEMS(I)
86:              write(IPR,7020) CWRK(:MM),
87:              & KSHARD(I), (DBLE(KSHARD(I))/KTTS)*100E0,
88:              & KLOCAL(I), (DBLE(KLOCAL(I))/KTTL)*100E0
89: 120      continue
90:          format(2X,A,2(2X,I12,' (',F5.1,'% )'))
91: C
92:          CWRK = 'Total'
93:          write(IPR,7040) CWRK(:MM), KTTS, KTTL
94:          format(/2X,A,2(2X,I12,: ' ',5X,' '))
95: C
96:          CWRK = 'Available'
97:          write(IPR,7040) CWRK(:MM), LIMIT, LIMITL
98: C
99: C .... Character Memory
100: C
101:          write(IPR,7060)
102:          format(/3X,'< Character Memory (4bytes/unit)>')
103: C
104:          call GTLASTC(LASTC)
105:          CWRK = 'Total'
106: CCCC write(IPR,7040) CWRK(:MM), LASTC-IAINFC(1)
107:          write(IPR,7040) CWRK(:MM), LSIZE(LASTC)
108:          CWRK = 'Available'
109:          write(IPR,7040) CWRK(:MM), LIMITC
110: C
111:      end if
112: C
113:      return
114: end

```

src/shared/mid2nm.f

```

1:      subroutine MID2NM( CODE, KZMAT, KMAT, KINPZ, NINPZ, NZONE,
2:      & JIMAG, IDMAT, DSMAT, NMAT )
3: C=====
4: C  PURPOSE: MAT ID # IN KMAT & KZMAT --> MAT #
5: C  CALLED IN: INTRO
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C i CODE : name of program using this routine ('MVP' or 'GMVP')
9: C o KZMAT(NZONE,2) : mat ID -> mat # of zones
10: C o KMAT(NINPZ,2) : mat ID -> mat # of input-zones
11: C i JIMAG : IMAGINARY-PARTICLE option. Set all material to inner void.
12: C i IDMAT(NMAT) : material ID
13: C i DSMAT(NMAT) : sum of number densities in each material
14: C-----
15:      character*(*) CODE
16:      integer KZMAT(NZONE,2), KMAT(NINPZ,2), KINPZ(NZONE), IDMAT(NMAT)
17:      real DSMAT(NMAT)
18: C
19:      include 'INC/_IOUNIT'
20: C
21: C-----
22: C  CHANGE MATERIAL ID'S TO MATERIAL NUMBERS
23: C-----
24:      IFMAT = 0
25:      do 140 I = 1, NZONE
26:        if ( KZMAT(I,1).ge.1 ) then
27:          do 100 M = 1, NMAT
28:            if ( KZMAT(I,1).eq.IDMAT(M) ) then
29:              KZMAT(I,1) = M
30:              go to 110
31:            end if
32:          continue
33: c##<2007/03/14:PN3:
34: c##      write(IMG,*) 'XXX Material ID ', KZMAT(I,1),
35:      write(IMG,*) 'XXX(mid2nm) Material ID ', KZMAT(I,1),
36: c##>
37:      &      ' in input-zone ', KINPZ(I), ' is not defined !!'
38:      call CNTERR( 'FATAL' )
39:      IFMAT = IFMAT + 1
40:      110      continue
41:      end if
42:      if ( KZMAT(I,2).ge.1 ) then
43:        do 120 M = 1, NMAT
44:          if ( KZMAT(I,2).eq.IDMAT(M) ) then
45:            KZMAT(I,2) = M
46:            go to 130
47:          end if
48:        continue
49: c##<2007/03/14:PN3:
50: c##      write(IMG,*) 'XXX Material ID ', KZMAT(I,2),
51:      write(IMG,*) 'XXX(mid2nm) Material ID ', KZMAT(I,2),
52: c##>
53:      &      ' specified as base material of STG reg. etc.,',
54:      &      ' in input-zone ', KINPZ(I), ' is not defined !!'
55:      call CNTERR( 'FATAL' )
56:      IFMAT = IFMAT + 1
57:      130      continue
58:      end if
59:      140      continue
60: C
61:      do 170 I = 1, NINPZ
62:        if ( KMAT(I,1).ge.1 ) then
63:          do 150 M = 1, NMAT
64:            if ( KMAT(I,1).eq.IDMAT(M) ) then
65:              KMAT(I,1) = M

```

```

66:              KMAT(I,2) = M
67:              go to 160
68:            end if
69:          150      continue
70:          end if
71:          160      continue
72:          170      continue
73: C
74: C  ... for base material zone of STG region, set KMAT(*,2) to
75: C  material # of the base material.
76: C
77:      do 180 I = 1, NZONE
78:        if ( KZMAT(I,1).lt.0.and.KZMAT(I,2).ge.0 ) then
79:          KMAT(KINPZ(I),2) = KZMAT(I,2)
80:          end if
81:        180      continue
82: C-----
83: C  .... CHANGE MATERIAL OF ZONES to inner-void if "DSMAT" of
84: C  the material is zero. (for MVP only)
85: C-----
86:      if ( CODE.eq.'MVP' ) then
87:        do 190 I = 1, NINPZ
88:          if ( KMAT(I,1).gt.0.and.DSMAT(KMAT(I,1)).eq.0.0 ) then
89:            write(IMG, '/lx,a,i5,a,i5,a/') '!!! material ',
90:            &      IDMAT(KMAT(I,1)), ' of ', I,
91:            &      'th input-zone is replaced with inner-void.'
92:            call CNTERR( 'WARNING' )
93:            KMAT(I,1) = 0
94:            end if
95:          190      continue
96:          do 200 I = 1, NZONE
97:            if ( KZMAT(I,1).gt.0.and.DSMAT(KZMAT(I,1)).eq.0.0 ) then
98:              KZMAT(I,1) = 0
99:            end if
100:            if ( KZMAT(I,2).gt.0.and.DSMAT(KZMAT(I,2)).eq.0.0 ) then
101:              KZMAT(I,2) = 0
102:            end if
103:          200      continue
104:          end if
105: C-----
106: C  .... CHANGE ALL ZONE WITH REAL MATERIAL TO INNER VOID FOR
107: C  'IMAGINARY-PARTICLE' OPTION
108: C-----
109:      if ( JIMAG.eq.1 ) then
110:        write(IPR,7000)
111:        7000      format(/lx, ' <<< All zones with real medium (KZMAT>0) are ',
112:        &      'changed to inner-void zone for ',
113:        &      'IMAGINARY-PARTICLE' option !! >> '/')
114:        do 210 I = 1, NZONE
115:          if ( KZMAT(I,1).gt.0 ) KZMAT(I,1) = 0
116:          if ( KZMAT(I,2).gt.0 ) KZMAT(I,2) = 0
117:        210      continue
118:        end if
119:        return
120:      end

```

src/shared/mirror.f

```

1:      subroutine MIRROR( JMNTR, JDEBG, JVMNT, JLATT,
2:      N      NBANK ,NREFS ,NSDA ,IRAND ,
3:      N      NLBZ ,NZONE ,
4:      B      XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
5:      B      WWW ,IZZ ,IBNK ,KIBNK,MIBNK,
6:      S      NCNTR ,WCNTR ,LSFFL ,NFFL ,IZFFL ,
7:      S      LSSRC ,NNXT ,IZNXT ,
8:      S      LSREF ,ISREF ,IZREF ,NBREF ,SDA ,KKREF ,
9:      S      LSLAT ,IZLAT ,NXLT ,MLBZZ ,
10:     W      KKSFl ,KKSf2 ,JKSF ,IPSURF,
11:     W      X ,Y ,Z ,AI ,BI ,CI ,
12:     W      IR,IBP ,IZT ,DSDA0 ,DSDA1 ,DSDA2 ,
13:     W      DSDA3 ,DSDA4 ,DSDA5 ,DSDA6 ,DSDA7 ,DSDA8 )
14: C===== MIRROR >=====
15: C PURPOSE: REFLECTION ON BOUNDARIES
16: C CALLED IN: ACTION
17: C CALLS:
18: C-----
19: C arguments (i=input, o=output, w=work)
20: C i JMNTR, JDEBG, JVMNT, JLATT: option flags(see [mvp|gmvp]/INC/_FLAGS)
21: C i NBANK ,NREFS ,NSDA ,NLBZ,NZONE: sizes (see INC/_SIZES etc.)
22: C io IRAND: seed of random numbers
23: C-----
24: C
25: C === INTER-STACK DATA FLOW ===
26: C
27: C REFLECTION STACK -----> FREE-FLIGHT STACK
28: C (LSREF,ISREF,IZREF,NBREF) | (LSFFL,IZFFL,NFFL)
29: C | +-----> LATTICE-SEARCH STACK
30: C | | (LSLAT,IZLAT,NXLT)
31: C | +-----> NEXT-ZONE-SEARCH STACK
32: C | | (LSSRC,IZNXT,NNXT)
33: C
34: C === BANK DATA TO BE UPDATED ===
35: C
36: C (AAA,BBB,CCC),IZZ : DIRECTION & ZONE #
37: C
38: C === BANK DATA ADDED NEWLY ===
39: C
40: C (NOTHING)
41: C-----
42: C implicit real*8(D)
43: C parameter( DPAI = 2.0D0*3.141592653589793D0 )
44: C
45: C integer JDEBG(*)
46: C
47: C .... PARTICLE BANK .....
48: C
49: C real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK)
50: C real*8 AAA(NBANK), BBB(NBANK), CCC(NBANK)
51: C real WWW(NBANK)
52: C integer IZZ(NBANK)
53: C integer IBNK(NBANK,*)
54: C integer KIBNK(0:MIBNK)
55: C
56: C .... STACKS .....
57: C
58: C integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
59: C integer LSSRC(NBANK), NNXT(NZONE+1), IZNXT(NBANK)
60: C integer LSREF(NBANK), NBREF(NREFS+1), ISREF(NBANK), IZREF(NBANK)
61: C integer LSLAT(NBANK), IZLAT(NBANK), NXLT(1)
62: C
63: C .... GEOMETRY .....
64: C
65: C real*8 SDA(NSDA)

```

```

66:      integer KKREF(2,NREFS), MLBZZ(NZONE)
67: C
68: C .... TALLY .....
69: C
70: C real*8 WCNTR(*), NCNTR(*)
71: C
72: C .... WORKING AREA ( LENGTH OF R = 2*NBANK ) .....
73: C
74: C real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
75: C & CI(NBANK), DSDA0(NBANK), DSDA1(NBANK), DSDA2(NBANK),
76: C & DSDA3(NBANK), DSDA4(NBANK), DSDA5(NBANK), DSDA6(NBANK),
77: C & DSDA7(NBANK), DSDA8(NBANK)
78: C real R(2*NBANK)
79: C integer IR(2*NBANK)
80: C integer IBP(NBANK), IZT(NBANK), JKSF(NREFS+1), IPSURF(NREFS+1),
81: C & KKSFl(NREFS+1), KKSf2(NREFS+1)
82: C
83: C include 'INC/_IOUNIT'
84: C
85: C -----
86: C
87: C if ( JVMNT.ne.0 ) call VMNTR1( 6, NBREF(NREFS+1) )
88: C NZS = 0
89: C NZ1 = NZONE + 1
90: C NR1 = NREFS + 1
91: C JPRDB = 0
92: C JREFB = 0
93: C do 100 K = 1, NREFS
94: C   if ( NBREF(K).ne.0 ) then
95: C     NZS = NZS + 1
96: C     JKSF(NZS) = K
97: C   end if
98: C 100 continue
99: C
100: C   if ( NZS.gt.20 ) then
101: C     IPSURF(1) = 1
102: C     do 110 NK = 1, NZS
103: C       IPSURF(NK+1) = IPSURF(NK) + NBREF(JKSF(NK))
104: C     110 continue
105: C *VOCL LOOP,NOVREC
106: C   do 120 NK = 1, NZS
107: C     KKSFl(JKSF(NK)) = IPSURF(NK) - 1
108: C   120 continue
109: C
110: C Cs*VOCL LOOP,NOVREC(IBP,IZT)
111: C   DO 162 I=1,NBREF(NR1)
112: C     KKSFl(ISREF(I)) = KKSFl(ISREF(I)) + 1
113: C     KKK = KKSFl(ISREF(I))
114: C     IBP(KKK) = LSREF(I)
115: C     IZT(KKK) = IZREF(I)
116: C   Cs 162 CONTINUE
117: C Cc*VOCL LOOP,SCALAR
118: C
119: C   do 130 I = 1, NBREF(NR1)
120: C     KKSFl(ISREF(I)) = KKSFl(ISREF(I)) + 1
121: C     IR(I) = KKSFl(ISREF(I))
122: C   130 continue
123: C *VOCL LOOP,NOVREC
124: C   do 140 I = 1, NBREF(NR1)
125: C     IBP(IR(I)) = LSREF(I)
126: C     IZT(IR(I)) = IZREF(I)
127: C   140 continue
128: C
129: C else
130: C

```

src/shared/mirror.f

```

131:      IPSURF(1) = 1
132:      do 150 NK = 1, NZS
133:        IPSURF(NK+1) = IPSURF(NK) + NBREF(JKSF(NK))
134:      150 continue
135:      KKK = 0
136:      do 170 NK = 1, NZS
137:        *VOCL LOOP,NOVREC
138:        do 160 I = 1, NBREF(NR1)
139:          if ( ISREF(I).eq.JKSF(NK) ) then
140:            KKK = KKK + 1
141:            IBP(KKK) = LSREF(I)
142:            IZT(KKK) = IZREF(I)
143:          end if
144:        160 continue
145:      170 continue
146:    end if
147:  C .....
148:    do 180 I = 1, NBREF(NR1)
149:      X(I) = XXX(IBP(I))
150:      Y(I) = YYY(IBP(I))
151:      Z(I) = ZZZ(IBP(I))
152:      AI(I) = AAA(IBP(I))
153:      BI(I) = BBB(IBP(I))
154:      CI(I) = CCC(IBP(I))
155:      IR(I) = 0
156:    180 continue
157:  C
158:  C IR : PERIODIC BOUNDARY ---->1
159:  C
160:  C=====
161:  C .... CALCULATE NORMAL VECTOR & DIRECTIONS OF REFLECTED PARTICLES ....
162:  C=====
163:  C
164:  C-----
165:  C .... GATHER DATA OF SURFACES .....
166:  C
167:  C KKREF(1,K) : SDA POINTER OF K'TH REFLECTIVE SURFACE
168:  C KKREF(2,K) : MATERIAL BEYOND K'TH REFLECTIVE SURFACE
169:  C-----
170:  C
171:    do 220 NK = 1, NZS
172:      K = JKSF(NK)
173:      LSP = KKREF(1,K)
174:  C
175:  C-----
176:  C ..... KKSFl : TYPE OF SURFACES, KKSf2 : MATERIAL
177:  C-----
178:    KKSFl(NK) = INT(ABS(SDA(LSP)))
179:    KKSf2(NK) = KKREF(2,K)
180:  C
181:  C ..... SLAB, SPHERE, HALF SPACE .....
182:  C
183:    if ( KKSFl(NK).eq.1 .or. KKSFl(NK).eq.2 .or. KKSFl(NK).eq.5 )
184:      & then
185:        do 190 I = IPSURF(NK), IPSURF(NK+1) - 1
186:          DSDA0(I) = SDA(LSP+1)
187:          DSDA1(I) = SDA(LSP+2)
188:          DSDA2(I) = SDA(LSP+3)
189:          if ( KKSf2(NK).eq.-4000 ) then
190:            DSDA3(I) = SDA(LSP+4)
191:            DSDA4(I) = SDA(LSP+5)
192:          end if
193:        190 continue
194:  C
195:  C ..... CYLINDER .....

```

```

196:  C
197:    else if ( KKSFl(NK).eq.3 ) then
198:      do 200 I = IPSURF(NK), IPSURF(NK+1) - 1
199:        DSDA0(I) = SDA(LSP+1)
200:        DSDA1(I) = SDA(LSP+2)
201:        DSDA2(I) = SDA(LSP+3)
202:        DSDA3(I) = SDA(LSP+4)
203:        DSDA4(I) = SDA(LSP+5)
204:        DSDA5(I) = SDA(LSP+6)
205:      200 continue
206:  C
207:  C ..... QUADRATIC SURFACE & TORUS .....
208:  C
209:    else if ( KKSFl(NK).eq.4 .or. KKSFl(NK).eq.6 ) then
210:      do 210 I = IPSURF(NK), IPSURF(NK+1) - 1
211:        DSDA0(I) = SDA(LSP+1)
212:        DSDA1(I) = SDA(LSP+2)
213:        DSDA2(I) = SDA(LSP+3)
214:        DSDA3(I) = SDA(LSP+4)
215:        DSDA4(I) = SDA(LSP+5)
216:        DSDA5(I) = SDA(LSP+6)
217:        DSDA6(I) = SDA(LSP+7)
218:        DSDA7(I) = SDA(LSP+8)
219:        DSDA8(I) = SDA(LSP+9)
220:      210 continue
221:    end if
222:  220 continue
223:  C
224:    KKK = 1
225:  230 KSFS = KKSFl(KKK)
226:    KSFM = KKSf2(KKK)
227:    do 240 NK = KKK, NZS
228:      if ( KKSFl(NK).ne.KSFS .or. KKSf2(NK).ne.KSFM ) go to 250
229:    240 continue
230:    NK = NZS + 1
231:  250 KKK2 = NK
232:    IP1 = 0
233:    IP2 = 0
234:    if ( KSFM.eq.-3000 ) then
235:      call RANU2( IRAND, R, 2*(IPSURF(KKK2)-IPSURF(KKK)), ICON )
236:      IP1 = 1 - IPSURF(KKK)
237:      IP2 = IP1 + IPSURF(KKK2) - IPSURF(KKK)
238:    end if
239:  C
240:  C=====
241:  C .... PERIODIC BOUNDARY CONDITION ....
242:  C=====
243:  C
244:    if ( KSFM.eq.-4000 ) then
245:      if ( KSFS.eq.1 ) then
246:        JPRDB = 1
247:        do 260 I = IPSURF(KKK), IPSURF(KKK2) - 1
248:          IR(I) = 1
249:          DL = DSDA0(I)*X(I) + DSDA1(I)*Y(I) + DSDA2(I)*Z(I)
250:          DQX = X(I)-DSDA0(I)*DL
251:          DQY = Y(I)-DSDA1(I)*DL
252:          DQZ = Z(I)-DSDA2(I)*DL
253:          DL2 = DSDA3(I)
254:          if ( abs(DSDA3(I)-DL).lt.abs(DSDA4(I)-DL) ) then
255:            DL2 = DSDA4(I)
256:          end if
257:          XXX(IBP(I)) = DSDA0(I)*DL2 + DQX
258:          YYY(IBP(I)) = DSDA1(I)*DL2 + DQY
259:          ZZZ(IBP(I)) = DSDA2(I)*DL2 + DQZ
260:        260 continue

```

src/shared/mirror.f

```

261: C
262: C .... PERIODIC BOUNDARY IS NOT AVAILABLE FOR OTHER THAN SLAB ....
263:       else
264:         write(IPR,*) ' '
265:         write(IPR,*) ' XXX PERIODIC BOUNDARY IS FOUND ',
266:           & 'FOR UNPERMITTED ZONE ',IZT(IPSURF(KKK))
267:       stop
268:     end if
269: C
270: C=====
271: C .... REFLECTION BOUNDARY CONDITION .....
272: C=====
273: C
274:       else
275:         JREB = 1
276: C
277: C-----
278: C ..... SLAB OR PLANE (HALF SPACE) .....
279: C-----
280: C
281:       if ( KSFS.eq.1 .or. KSFS.eq.5 ) then
282:         do 270 I = IPSURF(KKK), IPSURF(KKK2) - 1
283:           DUN = DSDA0(I)*AI(I) + DSDA1(I)*BI(I) + DSDA2(I)*CI(I)
284: C
285: C ..... MIRROR REFLECTION .....
286: C
287:         if ( KSFM.eq.-2000 ) then
288:           AI(I) = AI(I) - 2.0D0*DUN*DSDA0(I)
289:           BI(I) = BI(I) - 2.0D0*DUN*DSDA1(I)
290:           CI(I) = CI(I) - 2.0D0*DUN*DSDA2(I)
291: C
292: C ..... WHITE REFLECTION .....
293: C
294:         else if ( KSFM.eq.-3000 ) then
295: C           DAA = 2.0*R(I+IP1) - 1.0
296: C           DDD = SQRT(1.0-DAA*DAA)
297: C           TH = DPAI *R(I+IP2)
298: C           DBB = DDD*SIN(TH)
299: C           DCC = DDD*COS(TH)
300: C           DDD = DSDA0(I)*DAA + DSDA1(I)*DBB + DSDA2(I)*DCC
301: C           DS = SIGN(1.0D0,-DDD*DUN)
302: C           AI(I) = DAA*DS
303: C           BI(I) = DBB*DS
304: C           CI(I) = DCC*DS
305: C           DAA = R(I+IP1)
306: C           DDD = SQRT(1.0D0-DAA)
307: C           DAA = SQRT(DAA)
308: C           TH = DPAI*R(I+IP2)
309: C           DBB = DDD*SIN(TH)
310: C           DCC = DDD*COS(TH)
311: C           DS = SIGN(1.0D0,-DUN)
312: C           DN0 = DSDA0(I)*DS
313: C           DN1 = DSDA1(I)*DS
314: C           DN2 = DSDA2(I)*DS
315: C           DSN = 1.0D0 - DN0*DN0
316: C
317:         if ( DSN.gt.0. ) then
318: C           DST = SQRT(DSN)
319: C           DCF = DN1/DST
320: C           DSF = DN2/DST
321:         else
322: C           DST = 0.0D0
323: C           DCF = 1.0D0
324: C           DSF = 0.0D0
325:         end if

```

```

326:
327:           AI(I) = DN0*DAA - DCC*DST
328:           BI(I) = DN1*DAA + DN0*DCF*DCC - DSF*DBB
329:           CI(I) = DN2*DAA + DN0*DSF*DCC + DCF*DBB
330:         end if
331:       270 continue
332:     end if
333: C
334: C-----
335: C ..... SPHERE .....
336: C-----
337: C
338:       if ( KSFS.eq.2 ) then
339:         do 280 I = IPSURF(KKK), IPSURF(KKK2) - 1
340: C           DNX = X(I) - DSDA0(I)
341: C           DNY = Y(I) - DSDA1(I)
342: C           DNZ = Z(I) - DSDA2(I)
343: C           DH = DNX*DNX + DNY*DNY + DNZ*DNZ
344: C           DUN = DNX*AI(I) + DNY*BI(I) + DNZ*CI(I)
345: C
346: C ..... MIRROR REFLECTION .....
347: C
348:         if ( KSFM.eq.-2000 ) then
349: C           AI(I) = AI(I) - 2.0D0*DUN*DNX/DH
350: C           BI(I) = BI(I) - 2.0D0*DUN*DNY/DH
351: C           CI(I) = CI(I) - 2.0D0*DUN*DNZ/DH
352: C
353: C ..... WHITE REFLECTION .....
354: C
355:         else if ( KSFM.eq.-3000 ) then
356: C           DAA = R(I+IP1)
357: C           DDD = SQRT(1.0D0-DAA)
358: C           DAA = SQRT(DAA)
359: C           TH = DPAI*R(I+IP2)
360: C           DBB = DDD*SIN(TH)
361: C           DCC = DDD*COS(TH)
362: C           DS = SIGN(1.0D0,-DUN)
363: C           DHI = 1.0D0/SQRT(DH)
364: C           DN0 = DNX*DS*DHI
365: C           DN1 = DNY*DS*DHI
366: C           DN2 = DNZ*DS*DHI
367: C           DSN = 1.0D0 - DN0*DN0
368: C           if ( DSN.gt.0.0D0 ) then
369: C             DST = SQRT(DSN)
370: C             DCF = DN1/DST
371: C             DSF = DN2/DST
372: C           else
373: C             DST = 0.0D0
374: C             DCF = 1.0D0
375: C             DSF = 0.0D0
376: C           end if
377: C           AI(I) = DN0*DAA - DCC*DST
378: C           BI(I) = DN1*DAA + DN0*DCF*DCC - DSF*DBB
379: C           CI(I) = DN2*DAA + DN0*DSF*DCC + DCF*DBB
380:         end if
381:       280 continue
382:     end if
383: C
384: C-----
385: C ..... CYLINDER .....
386: C-----
387: C
388:       if ( KSFS.eq.3 ) then
389:         do 290 I = IPSURF(KKK), IPSURF(KKK2) - 1
390: C           DNX = X(I) - DSDA0(I)

```

src/shared/mirror.f

```

391:      DNY = Y(I) - DSDA1(I)
392:      DNZ = Z(I) - DSDA2(I)
393:      DDD = DSDA3(I)*DNX + DSDA4(I)*DNY + DSDA5(I)*DNZ
394:      DNX = DNX - DSDA3(I)*DDD
395:      DNY = DNY - DSDA4(I)*DDD
396:      DNZ = DNZ - DSDA5(I)*DDD
397:      DH = DNX**2 + DNY**2 + DNZ**2
398:      DUN = DNX*AI(I) + DNY*BI(I) + DNZ*CI(I)
399: C
400: C      ..... MIRROR REFLECTION .....
401: C
402: C      if ( KSFM.eq.-2000 ) then
403: C          AI(I) = AI(I) - 2.0*DUN*DNX/DH
404: C          BI(I) = BI(I) - 2.0*DUN*DNY/DH
405: C          CI(I) = CI(I) - 2.0*DUN*DNZ/DH
406: C
407: C      ..... WHITE REFLECTION .....
408: C
409: C      else if ( KSFM.eq.-3000 ) then
410: C          DAA = R(I+IP1)
411: C          DDD = SQRT(1.0D0-DAA)
412: C          DAA = SQRT(DAA)
413: C          TH = DPAI*R(I+IP2)
414: C          DBB = DDD*SIN(TH)
415: C          DCC = DDD*COS(TH)
416: C          DS = SIGN(1.0D0,-DUN)
417: C          DHI = 1.0D0/SQRT(DH)
418: C          DN0 = DNX*DS*DHI
419: C          DN1 = DNY*DS*DHI
420: C          DN2 = DNZ*DS*DHI
421: C          DSN = 1.0D0 - DN0*DN0
422: C          if ( DSN.gt.0.0D0 ) then
423: C              DST = SQRT(DSN)
424: C              DCF = DN1/DST
425: C              DSF = DN2/DST
426: C          else
427: C              DST = 0.0D0
428: C              DCF = 1.0D0
429: C              DSF = 0.0D0
430: C          end if
431: C          AI(I) = DN0*DAA - DCC*DST
432: C          BI(I) = DN1*DAA + DN0*DCF*DCC - DSF*DBB
433: C          CI(I) = DN2*DAA + DN0*DSF*DCC + DCF*DBB
434: C      end if
435: C      290      continue
436: C      end if
437: C
438: C-----
439: C      .... QUADRATIC EXPRESSION .....
440: C      S0*X**2 + S1*Y**2 + S2*Z**2 + S3*X*Y + S4*Y*Z + S5*Z*X
441: C      S6*X      + S7*Y      + S8*Z      + S9 = 0
442: C-----
443: C
444: C      if ( KSFS.eq.4 ) then
445: C          do 300 I = IPSURF(KKK), IPSURF(KKK2) - 1
446: C              DNX = 2.0D0*DSDA0(I)*X(I) + DSDA3(I)*Y(I) + DSDA5(I)*Z(I)
447: C              &      + DSDA6(I)
448: C              DNY = 2.0D0*DSDA1(I)*Y(I) + DSDA4(I)*Z(I) + DSDA3(I)*X(I)
449: C              &      + DSDA7(I)
450: C              DNZ = 2.0D0*DSDA2(I)*Z(I) + DSDA5(I)*X(I) + DSDA4(I)*Y(I)
451: C              &      + DSDA8(I)
452: C              &      DH = DNX**2 + DNY**2 + DNZ**2
453: C              DUN = DNX*AI(I) + DNY*BI(I) + DNZ*CI(I)
454: C
455: C      ..... MIRROR REFLECTION .....

```

```

456: C
457: C      if ( KSFM.eq.-2000 ) then
458: C          AI(I) = AI(I) - 2.0D0*DUN*DNX/DH
459: C          BI(I) = BI(I) - 2.0D0*DUN*DNY/DH
460: C          CI(I) = CI(I) - 2.0D0*DUN*DNZ/DH
461: C
462: C      ..... WHITE REFLECTION .....
463: C
464: C      else if ( KSFM.eq.-3000 ) then
465: C          DAA = R(I+IP1)
466: C          DDD = SQRT(1.0D0-DAA)
467: C          DAA = SQRT(DAA)
468: C          TH = DPAI*R(I+IP2)
469: C          DBB = DDD*SIN(TH)
470: C          DCC = DDD*COS(TH)
471: C          DS = SIGN(1.0D0,-DUN)
472: C          DHI = 1.0D0/SQRT(DH)
473: C          DN0 = DNX*DS*DHI
474: C          DN1 = DNY*DS*DHI
475: C          DN2 = DNZ*DS*DHI
476: C          DSN = 1.0D0 - DN0*DN0
477: C          if ( DSN.gt.0.0D0 ) then
478: C              DST = SQRT(DSN)
479: C              DCF = DN1/DST
480: C              DSF = DN2/DST
481: C          else
482: C              DST = 0.0D0
483: C              DCF = 1.0D0
484: C              DSF = 0.0D0
485: C          end if
486: C          AI(I) = DN0*DAA - DCC*DST
487: C          BI(I) = DN1*DAA + DN0*DCF*DCC - DSF*DBB
488: C          CI(I) = DN2*DAA + DN0*DSF*DCC + DCF*DBB
489: C      end if
490: C      300      continue
491: C      end if
492: C
493: C-----
494: C      .... TORUS      11/15/1990      ....
495: C-----
496: C
497: C      if ( KSFS.eq.6 ) then
498: C          do 310 I = IPSURF(KKK), IPSURF(KKK2) - 1
499: C              DX = X(I) - DSDA0(I)
500: C              DY = Y(I) - DSDA1(I)
501: C              DZ = Z(I) - DSDA2(I)
502: C              DXT = DSDA3(I)*DX + DSDA4(I)*DY + DSDA5(I)*DZ
503: C              DR = SQRT(DX**2+DY**2+DZ**2-DXT**2)
504: C              DB = (2.0D0*(DR-DSDA6(I))+DXT*DSDA8(I))/DR
505: C              DA = 2.0D0*DSDA7(I) - DB
506: C              DC = DSDA8(I)*(DR-DSDA6(I))
507: C              DXY = DSDA3(I)*DSDA4(I)
508: C              DZX = DSDA3(I)*DSDA5(I)
509: C              DYZ = DSDA4(I)*DSDA5(I)
510: C
511: C              DNX = DB*DX + DA*(DSDA3(I)**2*DX+DXY*DY+DZX*DZ)
512: C              &      + DC*DSDA3(I)
513: C              DNY = DB*DY + DA*(DSDA4(I)**2*DY+DYZ*DZ+DXY*DX)
514: C              &      + DC*DSDA4(I)
515: C              DNZ = DB*DZ + DA*(DSDA5(I)**2*DZ+DZX*DX+DYZ*DY)
516: C              &      + DC*DSDA5(I)
517: C
518: C              DH = DNX**2 + DNY**2 + DNZ**2
519: C              DUN = DNX*AI(I) + DNY*BI(I) + DNZ*CI(I)
520: C

```

src/shared/mirror.f

```

521: C          ..... MIRROR REFLECTION .....
522: C
523: C          if ( KSFMEQ.-2000 ) then
524: C              AI(I) = AI(I) - 2.0D0*DUN*DNX/DH
525: C              BI(I) = BI(I) - 2.0D0*DUN*DNY/DH
526: C              CI(I) = CI(I) - 2.0D0*DUN*DNZ/DH
527: C
528: C          ..... WHITE REFLECTION .....
529: C
530: C          else if ( KSFMEQ.-3000 ) then
531: C              DAA = R(I+IP1)
532: C              DDD = SQRT(1.0D0-DAA)
533: C              DAA = SQRT(DAA)
534: C              TH = DPAI*R(I+IP2)
535: C              DBB = DDD*SIN(TH)
536: C              DCC = DDD*COS(TH)
537: C              DS = SIGN(1.0D0,-DUN)
538: C              DHI = 1.0D0/SQRT(DH)
539: C              DN0 = DNX*DS*DHI
540: C              DN1 = DNY*DS*DHI
541: C              DN2 = DNZ*DS*DHI
542: C              DSN = 1.0D0 - DN0*DN0
543: C              if ( DSN.gt.0.0D0 ) then
544: C                  DST = SQRT(DSN)
545: C                  DCF = DN1/DST
546: C                  DSF = DN2/DST
547: C              else
548: C                  DST = 0.0D0
549: C                  DCF = 1.0D0
550: C                  DSF = 0.0D0
551: C              end if
552: C              AI(I) = DN0*DAA - DCC*DST
553: C              BI(I) = DN1*DAA + DN0*DCF*DCC - DSF*DBB
554: C              CI(I) = DN2*DAA + DN0*DSF*DCC + DCF*DBB
555: C          end if
556: C          continue
557: C      end if
558: C
559: C      end if
560: C
561: C      if ( KKK2.le.NZS ) then
562: C          KKK = KKK2
563: C          go to 230
564: C      end if
565: C
566: C=====
567: C .... CASE OF REFLECTION BOUNDARY CONDITION ONLY ....
568: C=====
569: C      if ( JPRDB.eq.0 ) then
570: C-----
571: C ..... SEND PARTICLES TO FLIGHT STACK .....
572: C-----
573: C      if ( JLATT.eq.0 ) then
574: C          IFFL = NFFL(NZ1)
575: C          do 320 I = 1, NBREF(NR1)
576: C              LSFFL(IFFL+I) = IBP(I)
577: C              IZFFL(IFFL+I) = IZT(I)
578: C          continue
579: C      *VOCL LOOP,SCALAR
580: C          do 330 I = 1, NBREF(NR1)
581: C              NFFL(IZT(I)) = NFFL(IZT(I)) + 1
582: C          continue
583: C
584: C          NCNTR(17) = NCNTR(17) + NBREF(NR1)
585: C          NFFL(NZ1) = IFFL + NBREF(NR1)

```

```

586: C
587: C-----
588: C ..... SEND PARTICLES TO FLIGHT STACK OR LATTICE-SEARCH STACK .....
589: C-----
590: C      else if ( JLATT.ne.0 ) then
591: C          IFFL = NFFL(NZ1)
592: C          ILAT = NXLT(NLBZ+1)
593: C          do 340 I = 1, NBREF(NR1)
594: C              LBZ = MLBZZ(IZT(I))
595: C              if ( LBZ.eq.0 ) then
596: C                  IFFL = IFFL + 1
597: C                  LSFFL(IFFL) = IBP(I)
598: C                  IZFFL(IFFL) = IZT(I)
599: C              else
600: C                  ILAT = ILAT + 1
601: C                  LSLAT(ILAT) = IBP(I)
602: C                  IZLAT(ILAT) = LBZ
603: C              end if
604: C          continue
605: C
606: C      *VOCL LOOP,SCALAR
607: C          do 350 I = NFFL(NZ1) + 1, IFFL
608: C              NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
609: C          continue
610: C
611: C      *VOCL LOOP,SCALAR
612: C          do 360 I = NXLT(NLBZ+1) + 1, ILAT
613: C              NXLT(IZLAT(I)) = NXLT(IZLAT(I)) + 1
614: C          continue
615: C
616: C          NCNTR(17) = NCNTR(17) + IFFL - NFFL(NZ1)
617: C          NFFL(NZ1) = IFFL
618: C          NXLT(NLBZ+1) = ILAT
619: C      end if
620: C-----
621: C ..... UPDATE DIRECTIONS IN BANK .....
622: C-----
623: C      *VOCL LOOP,NOVREC
624: C          do 370 I = 1, NBREF(NR1)
625: C              DD = SQRT(AI(I)**2+BI(I)**2+CI(I)**2)
626: C              AAA(IBP(I)) = AI(I) /DD
627: C              BBB(IBP(I)) = BI(I) /DD
628: C              CCC(IBP(I)) = CI(I) /DD
629: C          continue
630: C
631: C=====
632: C .... CASE OF PERIODIC BOUNDARY CONDITION ONLY ....
633: C=====
634: C      else if ( JREFB.eq.0 ) then
635: C-----
636: C ..... SEND PARTICLES TO SEARCH STACK .....
637: C-----
638: C          INXT = NNXT(NZ1)
639: C          do 380 I = 1, NBREF(NR1)
640: C              LSSRC(INXT+I) = IBP(I)
641: C              IZNXT(INXT+I) = IZT(I)
642: C          continue
643: C      *VOCL LOOP,SCALAR
644: C          do 390 I = 1, NBREF(NR1)
645: C              NNXT(IZT(I)) = NNXT(IZT(I)) + 1
646: C          continue
647: C
648: C          NNXT(NZ1) = INXT + NBREF(NR1)
649: C
650: C=====

```


src/shared/mirror.f

```

651: C .... CASE OF BOTH BOUNDARY CONDITIONS ....
652: C=====
653:       else
654: C-----
655: C ..... SEND PARTICLES TO FLIGHT STACK OR SEARCH STACK .....
656: C-----
657:       if ( JLATT.eq.0 ) then
658:         IFFL = NFFL(NZ1)
659:         INXT = NNXT(NZ1)
660:         do 400 I = 1, NBREF(NR1)
661:           if ( IR(I).eq.0 ) then
662:             IFFL = IFFL + 1
663:             LSFFL(IFFL) = IBP(I)
664:             IZFFL(IFFL) = IZT(I)
665:           else
666:             INXT = INXT + 1
667:             LSSRC(INXT) = IBP(I)
668:             IZNXT(INXT) = IZT(I)
669:           end if
670:         400 continue
671: C
672: *VOCL LOOP,SCALAR
673:       do 410 I = NFFL(NZ1) + 1, IFFL
674:         NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
675:       410 continue
676: C
677: *VOCL LOOP,SCALAR
678:       do 420 I = NNXT(NZ1) + 1, INXT
679:         NNXT(IZNXT(I)) = NNXT(IZNXT(I)) + 1
680:       420 continue
681: C
682:         NCNTR(17) = NCNTR(17) + IFFL - NFFL(NZ1)
683:         NFFL(NZ1) = IFFL
684:         NNXT(NZ1) = INXT
685: C-----
686: C ..... SEND PARTICLES TO FLIGHT, SEARCH OR LATTICE-SEARCH STACK .....
687: C-----
688:       else if ( JLATT.ne.0 ) then
689:         IFFL = NFFL(NZ1)
690:         INXT = NNXT(NZ1)
691:         ILAT = NXLT(NLBZ+1)
692:         do 430 I = 1, NBREF(NR1)
693:           LBZ = MLBZZ(IZT(I))
694:           if ( IR(I).eq.1 ) then
695:             INXT = INXT + 1
696:             LSSRC(INXT) = IBP(I)
697:             IZNXT(INXT) = IZT(I)
698:           else if ( LBZ.eq.0 ) then
699:             IFFL = IFFL + 1
700:             LSFFL(IFFL) = IBP(I)
701:             IZFFL(IFFL) = IZT(I)
702:           else
703:             ILAT = ILAT + 1
704:             LSLAT(ILAT) = IBP(I)
705:             IZLAT(ILAT) = LBZ
706:           end if
707:         430 continue
708: C
709: *VOCL LOOP,SCALAR
710:       do 440 I = NFFL(NZ1) + 1, IFFL
711:         NFFL(IZFFL(I)) = NFFL(IZFFL(I)) + 1
712:       440 continue
713: C
714: *VOCL LOOP,SCALAR
715:       do 450 I = NXLT(NLBZ+1) + 1, ILAT

```

```

716:         NXLT(IZLAT(I)) = NXLT(IZLAT(I)) + 1
717:       450 continue
718: C
719: *VOCL LOOP,SCALAR
720:       do 460 I = NNXT(NZ1) + 1, INXT
721:         NNXT(IZNXT(I)) = NNXT(IZNXT(I)) + 1
722:       460 continue
723: C
724:         NCNTR(17) = NCNTR(17) + IFFL - NFFL(NZ1)
725:         NFFL(NZ1) = IFFL
726:         NXLT(NLBZ+1) = ILAT
727:         NNXT(NZ1) = INXT
728:       end if
729: C-----
730: C ..... UPDATE DIRECTIONS IN BANK .....
731: C-----
732: *VOCL LOOP,NOVREC
733:       do 470 I = 1, NBREF(NR1)
734:         if ( IR(I).eq.0 ) then
735:           DD = SQRT(AI(I)**2+BI(I)**2+CI(I)**2)
736:           AAA(IBP(I)) = AI(I) /DD
737:           BBB(IBP(I)) = BI(I) /DD
738:           CCC(IBP(I)) = CI(I) /DD
739:         endif
740:       470 continue
741:     endif
742: C
743: C ... clear flight path counter
744: C
745: *VOCL LOOP,NOVREC
746:       if ( KIBNK(5).ne.0 ) then
747:         do 475 I = 1, NBREF(NR1)
748:           IBNK(IBP(I),KIBNK(5)) = 0
749:         475 continue
750:       end if
751: C
752:         NCNTR(18) = NCNTR(18) + NBREF(NR1)
753:         do 480 I = 1, NBREF(NR1)
754:           IZZ(IBP(I)) = IZT(I)
755:           WCNTR(18) = WCNTR(18) + WWW(IBP(I))
756:         480 continue
757: C
758:         do 490 I = 1, NR1
759:           NBREF(I) = 0
760:         490 continue
761:       return
762:     end

```

src/shared/mrzone.f

```

1:      subroutine MRZONE( KZMAT, KZREG, KMAT, KREG, KINPZ,
2:      &                  NINPZ, NZONE, TNAMS, NNAMES, KREGN, KREGI, NREG,
3:      &                  IDLAT, LTY, NLATT,
4:      &                  JREFL, JLATT, JTLLT, JSTGR )
5:      C=====
6:      C PURPOSE: MAT # & REGION # , INPUT-ZONE-WISE --> ZONE-WISE
7:      C
8:      C   KMAT(*) must be material ID as input (not material or lattice #'s)
9:      C
10:     C CALLED IN: GEOMIN
11:     C -----
12:     C arguments (i=input, o=output, w=work)
13:     C
14:     C *i IOG : message output I/O unit (removed Jan 2000)
15:     C o KZMAT(NZONE) : material ID of zones
16:     C o KZREG(NZONE) : region # of zones
17:     C i KMAT(NINPZ) : material ID of input-zones
18:     C o KREG(NINPZ) : region # of input-zones
19:     C -----
20:     integer KZMAT(NZONE), KZREG(NZONE), KMAT(NINPZ), KREG(NINPZ),
21:     &      KINPZ(NZONE), KREGI(NINPZ)
22:     integer IDLAT(NLATT), LTY(NLATT)
23:     C
24:     character*12 TNAMS(NNAMES), KREGN(NINPZ)
25:     C
26:     include 'INC/_IUNIT'
27:     C
28:     include 'INC/_PMLATT'
29:     include 'INC/_SPLATT'
30:     C
31:     C -----
32:     C MAKE ZONE-WISE MATERIAL/REGION TABLE FROM INPUT-ZONE-WISE DATA
33:     C -----
34:     JREFL = 0
35:     do 100 I = 1, NZONE
36:       KZMAT(I) = KMAT(KINPZ(I))
37:     C ..... THERE EXIST SOME REFLECTIVE SURFACES .....
38:       if ( KZMAT(I).lt.-1000.and..not.ISLATT(KZMAT(I)) ) JREFL = 1
39:     100 continue
40:     C
41:     C -----
42:     C ..... CHECK MATERIAL NUMBER .....
43:     C -----
44:     C
45:     IFMAT = 0
46:     if ( JLATT.eq.0 ) then
47:       do 110 I = 1, NZONE
48:         if ( KZMAT(I).lt.0.and.KZMAT(I).ne.-1000
49:         & .and.KZMAT(I).ne.-2000.and.KZMAT(I).ne.-3000
50:         & .and.KZMAT(I).ne.-4000 ) then
51:           IFMAT = 1
52:           write(IMG,7000) KZMAT(I), I
53:         end if
54:       110 continue
55:     end if
56:     7000 format(/1X,' XXX INVALID MATERIAL ',I6,' IN ZONE ',I6,' !! ')
57:     C
58:     C -----
59:     CC ..... STOP IF INVALID MATERIALS ARE SPECIFIED .....
60:     C -----
61:     if ( IFMAT.ne.0 ) then
62:       write(IMG,'(/1X,' XXX INVALID MATERIAL !!')')
63:       call PRSTOP( 1, 'INVALID MATERIAL #' )
64:       stop 888
65:     end if

```

```

66:     C
67:     C -----
68:     C ..... KREGI(1:NINPZ) --> KREGN(1:NINPZ) IN TNAMS TABLE.
69:     C ..... AND PREPARE KZREG FOR NON-UNIVERSE DEPENDENT TALLY.
70:     C -----
71:     C
72:     do 120 I = 1, NINPZ
73:       call CBSINC( TNAMS, NNAMES, KREGN(I), KREGI(I) )
74:     120 continue
75:     C
76:     C
77:     C -----
78:     C ..... DETERMINE REGION #'S WHEN JTLLT = 0 .....
79:     C -----
80:     C
81:     C
82:     if ( JTLLT.eq.0 ) then
83:       NREG = 0
84:       do 140 I = 1, NINPZ
85:         KREG(I) = 0
86:         KKM = KMAT(I)
87:       C
88:       if ( KKM.ge.0 ) then
89:         do 130 J = 1, I - 1
90:           if ( KREG(J).ne.0.and.KREGI(J).eq.KREGI(I) ) then
91:             KREG(I) = KREG(J)
92:             go to 140
93:           end if
94:         130 continue
95:         NREG = NREG + 1
96:         KREG(I) = -1
97:       end if
98:     140 continue
99:     C
100:    C ..... ASSIGN REGION #'S IN ASCENDING ORDER OF KREGN ....
101:    C
102:    do 170 N = 1, NREG
103:      MINN = NNAMES + 1
104:      do 150 I = 1, NINPZ
105:        if ( KREG(I).lt.0 ) then
106:          MINN = MIN(KREGI(I),MINN)
107:        end if
108:      150 continue
109:      do 160 I = 1, NINPZ
110:        if ( KREGI(I).eq.MINN ) KREG(I) = N
111:      160 continue
112:    170 continue
113:    C
114:    do 180 I = 1, NZONE
115:      KZREG(I) = KREG(KINPZ(I))
116:    180 continue
117:    C
118:    C *****
119:    call LABEL( IOG, 'REGION NUMBER FOR INPUT-ZONES' )
120:    C *****
121:    C
122:    write(IOG,7020)
123:    7020 format(/' INPUT-ZONE NAME REGION MATERIAL'
124:    & /' ----- ----- ----- -----')
125:    C
126:    write(IOG,7040) (I,KREGN(I),KREG(I),KMAT(I),I=1,NINPZ)
127:    7040 format( 4X, I5, 6X, A, 6X, I4, 7X, I6 )
128:    C
129:    write(IOG,7060)
130:    7060 format(' ----- ----- ----- -----')

```

src/shared/mrzone.f

```
131: C
132:     write(IOG,'(///'      ** TOTAL NUMBER OF REGION : '' ,I4/)' ) NREG
133:     end if
134: C
135:     return
136:     end
```

SAE

src/shared/mvpcommu.f

```

1: C/#IF PARA(PVM MPI)
2: C=====
3: C   MVP/GMVP inter-task communication interface.
4: C
5: C purpose: interface routines to message passing calls for MVP/GMVP
6: C           system.
7: C
8: C   PVM & MPI calls for message passing are masked by these routines.
9: C
10: C-----
11: C
12: C
13: C <procedures> .....
14: C
15: C
16: C * subroutine mvpcom_send_i4( itsk, msgtag, ia, ndata, ierr)
17: C * subroutine mvpcom_send_r4( itsk, msgtag, ra, ndata, ierr)
18: C * subroutine mvpcom_send_r8( itsk, msgtag, da, ndata, ierr)
19: C * subroutine mvpcom_send_ch( itsk, msgtag, ch, ndata, ierr)
20: C
21: C   Send integer/real/real*8/character data to task ITSK
22: C   with message tag MSGTAG.
23: C
24: C * subroutine mvpcom_rcv_i4( itsk, msgtag, ia, ndata, ierr)
25: C * subroutine mvpcom_rcv_r4( itsk, msgtag, ra, ndata, ierr)
26: C * subroutine mvpcom_rcv_r8( itsk, msgtag, da, ndata, ierr)
27: C * subroutine mvpcom_rcv_ch( itsk, msgtag, ch, ndata, ierr)
28: C
29: C   Receive integer/real/real*8/character data from task ITSK
30: C   with message tag MSGTAG.
31: C
32: C * subroutine mvpcom_bcast_i4( itsk, msgtag, ia, ndata, ierr)
33: C * subroutine mvpcom_bcast_r4( itsk, msgtag, ra, ndata, ierr)
34: C * subroutine mvpcom_bcast_r8( itsk, msgtag, da, ndata, ierr)
35: C * subroutine mvpcom_bcast_ch( itsk, msgtag, ch, ndata, ierr)
36: C
37: C   Broadcast integer/real/real*8/character data from task ITSK
38: C   with message tag MSGTAG.
39: C   Both sender and receiver must call this.
40: C
41: C * subroutine mvpcom_sum_i4( itsk, msgtag, ia, ndata, i4buf, nbuf,
42: C   ierr)
43: C * subroutine mvpcom_sum_r4( itsk, msgtag, ra, ndata, r4buf, nbuf,
44: C   ierr)
45: C * subroutine mvpcom_sum_r8( itsk, msgtag, da, ndata, r8buf, nbuf,
46: C   ierr)
47: C
48: C   Take element-wise sum of an array of type integer, real and
49: C   real*8 on task ITSK.
50: C
51: C   I4BUF, R4BUF and R8BUF is a working buffer array of size
52: C   NBUF. NBUF can be an arbitrary non-zero size.
53: C
54: C
55: C * subroutine mvpcom_probe( itsk, msgtag, info )
56: C
57: C   Check if message from task ITSK and having message tag MSGTAG
58: C   has arrived. ( info=0 : not arrived, info > 0 : arrived )
59: C   Non positive ITSK and/or MSGTAG matches any task and/or tag
60: C   and returns task-ID(MVP) and/or message tag of probed message.
61: C
62: C * subroutine mvpcom_exit( info )
63: C
64: C   Terminate parallel node connection of the calling process.
65: C

```

```

66: C <arguments> .....
67: C
68: C
69: C   ITSK : task no. Counted from 1 to NTASK in MVP/GMVP.
70: C
71: C   MSGTAG : message tag.
72: C           Any positive integer, or a negative integer which matches
73: C           any tag.
74: C
75: C   IERR : returns zero on success, non-zero if an error occurred.
76: C
77: C   IA/RA/DA/CH : data sent or received.
78: C
79: C   NDATA : number of data elements sent or received.
80: C
81: C
82: C   All calls except "bcast" group can accept ITSK=-1 and/or MSGTAG=-1
83: C   which match any task and/or message tag.
84: C
85: C
86: C <comment> .....
87: C
88: C
89: C   Interface routines here are not aiming to be a "light-speed"
90: C   communication interface but to be a stable and fool-proof
91: C   one. Do not use them in communication time bound programs.
92: C
93: C-----
94: C   PVM specific :
95: C
96: C   TASKID(1:NTASK) must be defined as "task ID" list.
97: C
98: C   MPI specific :
99: C
100: C   ITSK-1 is the rank of task.
101: C   Communication in MPI communicator MVP_COMM.
102: C   Status array MPISTAT(MPI_STATUS_SIZE) may be necessary.
103: C=====
104: C
105: C -----
106: C   subroutine MVPCOM_SEND_I4( ITSK, MSGTAG, IA, NDATA, IERR )
107: C -----
108: C   integer IA(NDATA)
109: C   include 'INC/_IOUNIT'
110: C/#IF PARA(PVM)
111: C   include 'INC/_PVMPPARA'
112: C/#ELSEIF PARA(MPI)
113: C   include 'mpif.h'
114: C/#ENDIF
115: C   include 'INC/_TASKDT'
116: C
117: C   Check %%%%%%%%%
118: C   write(*,*) '(MVPCOM_SEND_I4) itsk ',itsk,' msgtag ',msgtag,
119: C   & ' ndata ',ndata,' idtask ',idtask
120: C   %%%%%%%%%
121: C
122: C/#IF PARA(PVM)
123: C
124: C   call PVMFINITSEND( PVMDEFAULT, INFO )
125: C   call PVMFPACK( INTEGER4, IA, NDATA, 1, INFO )
126: C   call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
127: C
128: C/#ELSEIF PARA(MPI)
129: C/# IF DOUBLE_MPI_DATATYPE
130: C   call MPI_SEND( IA, NDATA, MPI_INTEGER8, ITSK-1, MSGTAG, MVP_COMM,

```

src/shared/mvpcommu.f

```

131: *      &      IERR )
132: C/# ELSE
133:      call MPI_SEND( IA, NDATA, MPI_INTEGER, ITSK-1, MSGTAG, MVP_COMM,
134:      &      IERR )
135: C/# ENDIF
136: C/#ENDIF
137:      return
138:      end
139: C
140: C -----
141:      subroutine MVPCOM_SEND_I8( ITSK, MSGTAG, I8A, NDATA, IERR )
142: C -----
143: C/#IF INTEGER8
144: *      integer*8 I8A(NDATA)
145: C/#ELSE
146:      integer I8A(NDATA)
147: C/#ENDIF
148:      include 'INC/_IOUNIT'
149: C/#IF PARA(PVM)
150:      include 'INC/_PVMPARA'
151: C/#ELSEIF PARA(MPI)
152: *      include 'mpif.h'
153: C/#ENDIF
154:      include 'INC/_TASKDT'
155: C
156: Check %%%%%%%%%
157: C      write(*,*) '(MVPCOM_SEND_I4) itsk ',itsk,' msgtag ',msgtag
158: C      &      ' ndata ',ndata,' idtask ',idtask
159: C %%%%%%%%%
160: C
161: C/#IF PARA(PVM)
162: C
163:      call PVMFINITSEND( PVMDEFAULT, INFO )
164: C/# IF INTEGER8
165: *      call PVMFPACK( INTEGER8, I8A, NDATA, 1, INFO )
166: C/# ELSE
167:      call PVMFPACK( INTEGER4, I8A, NDATA, 1, INFO )
168: C/# ENDIF
169:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
170: C
171: C/#ELSEIF PARA(MPI)
172: C/# IF INTEGER8
173: *      call MPI_SEND( I8A, NDATA, MPI_INTEGER8, ITSK-1, MSGTAG, MVP_COMM,
174: *      &      IERR )
175: C/# ELSE
176:      call MPI_SEND( I8A, NDATA, MPI_INTEGER , ITSK-1, MSGTAG, MVP_COMM,
177:      &      IERR )
178: C/# ENDIF
179: C/#ENDIF
180:      return
181:      end
182: C
183: C
184: C -----
185:      subroutine MVPCOM_SEND_R4( ITSK, MSGTAG,RA, NDATA, IERR )
186: C -----
187:      real RA(NDATA)
188:      include 'INC/_IOUNIT'
189: C/#IF PARA(PVM)
190:      include 'INC/_PVMPARA'
191: C/#ELSEIF PARA(MPI)
192: *      include 'mpif.h'
193: C/#ENDIF
194:      include 'INC/_TASKDT'
195: C

```

```

196: C
197: C/#IF PARA(PVM)
198: C
199:      call PVMFINITSEND( PVMDEFAULT, INFO )
200:      call PVMFPACK( REAL4, RA, NDATA, 1, INFO )
201:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
202: C
203: C/#ELSEIF PARA(MPI)
204: C/# IF DOUBLE_MPI_DATATYPE
205: *      call MPI_SEND(RA, NDATA, MPI_REAL8, ITSK-1, MSGTAG, MVP_COMM, IERR
206: *      &      )
207: C/# ELSE
208:      call MPI_SEND( RA, NDATA, MPI_REAL, ITSK-1, MSGTAG, MVP_COMM, IERR
209:      &      )
210: C/# ENDIF
211: C/#ENDIF
212:      return
213: C
214:      end
215: C
216: C -----
217:      subroutine MVPCOM_SEND_R8( ITSK, MSGTAG,DA, NDATA, IERR )
218: C -----
219:      real*8 DA(NDATA)
220:      include 'INC/_IOUNIT'
221: C/#IF PARA(PVM)
222:      include 'INC/_PVMPARA'
223: C/#ELSEIF PARA(MPI)
224: *      include 'mpif.h'
225: C/#ENDIF
226:      include 'INC/_TASKDT'
227: C
228: C
229: C/#IF PARA(PVM)
230: C
231:      call PVMFINITSEND( PVMDEFAULT, INFO )
232:      call PVMFPACK( REAL8, DA, NDATA, 1, INFO )
233:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
234: C
235: C/#ELSEIF PARA(MPI)
236:      call MPI_SEND( DA, NDATA, MPI_DOUBLE_PRECISION, ITSK-1, MSGTAG,
237:      &      MVP_COMM, IERR )
238: C/#ENDIF
239:      return
240: C
241:      end
242: C
243: C -----
244:      subroutine MVPCOM_SEND_CH( ITSK, MSGTAG,CH, NDATA, IERR )
245: C -----
246:      character*(*) CH
247:      include 'INC/_IOUNIT'
248: C/#IF PARA(PVM)
249:      include 'INC/_PVMPARA'
250: C/#ELSEIF PARA(MPI)
251: *      include 'mpif.h'
252: C/#ENDIF
253:      include 'INC/_TASKDT'
254: C
255: C
256: C/#IF PARA(PVM)
257: C
258:      call PVMFINITSEND( PVMDEFAULT, INFO )
259:      call PVMFPACK( STRING, CH, NDATA, 1, INFO )
260:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )

```

src/shared/mvpcommu.f

```
261: C
262: C/#ELSEIF PARA(MPI)
263:   call MPI_SEND( CH, NDATA, MPI_CHARACTER, ITSK-1, MSGTAG, MVP_COMM,
264:     & IERR )
265: C/#ENDIF
266:   return
267: C
268:   end
269: C
270: C -----
271:   subroutine MVPCOM_RECV_I4( ITSK, MSGTAG,IA, NDATA, IERR )
272: C -----
273:   integer IA(NDATA)
274:   include 'INC/_IOUNIT'
275: C/#IF PARA(PVM)
276:   include 'INC/_PVM PARA'
277: C/#ELSEIF PARA(MPI)
278:   * include 'mpif.h'
279:   integer MPISTAT(MPI_STATUS_SIZE)
280: C/#ENDIF
281:   include 'INC/_TASKDT'
282: C
283:   Check %%%%%%%%%
284:   C   write(*,*) '(MVPCOM_RECV_I4) itsk ',itsk,' msgtag ',msgtag,
285:   C   & ' ndata ',ndata,' idtask ',idtask
286:   C %%%%%%%%%
287:   C
288: C/#IF PARA(PVM)
289:   C
290:   IT = -1
291:   if ( ITSK.gt.0 ) IT = TASKID(ITSK)
292:   call PVMFREC( IT, MSGTAG, IERR )
293:   call PVMFUNPACK( INTEGER4, IA, NDATA, 1, IERR )
294: C
295: C/#ELSEIF PARA(MPI)
296:   IT = MPI_ANY_SOURCE
297:   if ( ITSK.gt.0 ) IT = ITSK - 1
298: C/# IF DOUBLE_MPI_DATATYPE
299:   * call MPI_RECV( IA, NDATA, MPI_INTEGER8, IT, MSGTAG, MVP_COMM,
300:   * & MPISTAT, IERR )
301: C/# ELSE
302:   call MPI_RECV( IA, NDATA, MPI_INTEGER, IT, MSGTAG, MVP_COMM,
303:   & MPISTAT, IERR )
304: C/# ENDIF
305: C/#ENDIF
306:   return
307: C
308:   end
309: C
310: C -----
311:   subroutine MVPCOM_RECV_I8( ITSK, MSGTAG, I8A, NDATA, IERR )
312: C -----
313: C/#IF INTEGER8
314:   * integer*8 I8A(NDATA)
315: C/#ELSE
316:   integer I8A(NDATA)
317: C/#ENDIF
318:   include 'INC/_IOUNIT'
319: C/#IF PARA(PVM)
320:   include 'INC/_PVM PARA'
321: C/#ELSEIF PARA(MPI)
322:   * include 'mpif.h'
323:   integer MPISTAT(MPI_STATUS_SIZE)
324: C/#ENDIF
325:   include 'INC/_TASKDT'
```

```
326: C
327:   Check %%%%%%%%%
328:   C   write(*,*) '(MVPCOM_RECV_I8) itsk ',itsk,' msgtag ',msgtag,
329:   C   & ' ndata ',ndata,' idtask ',idtask
330:   C %%%%%%%%%
331:   C
332: C/#IF PARA(PVM)
333:   C
334:   IT = -1
335:   if ( ITSK.gt.0 ) IT = TASKID(ITSK)
336:   call PVMFREC( IT, MSGTAG, IERR )
337: C/# IF INTEGER8
338:   * call PVMFUNPACK( INTEGER8, I8A, NDATA, 1, IERR )
339: C/# ELSE
340:   call PVMFUNPACK( INTEGER4, I8A, NDATA, 1, IERR )
341: C/# ENDIF
342: C
343: C/#ELSEIF PARA(MPI)
344:   IT = MPI_ANY_SOURCE
345:   if ( ITSK.gt.0 ) IT = ITSK - 1
346: C/# IF INTEGER8
347:   * call MPI_RECV( I8A, NDATA, MPI_INTEGER8, IT, MSGTAG, MVP_COMM,
348:   * & MPISTAT, IERR )
349: C/# ELSE
350:   call MPI_RECV( I8A, NDATA, MPI_INTEGER, IT, MSGTAG, MVP_COMM,
351:   & MPISTAT, IERR )
352: C/# ENDIF
353: C/#ENDIF
354:   return
355: C
356:   end
357: C
358: C -----
359:   subroutine MVPCOM_RECV_R4( ITSK, MSGTAG,RA, NDATA, IERR )
360: C -----
361:   real RA(NDATA)
362:   include 'INC/_IOUNIT'
363: C/#IF PARA(PVM)
364:   include 'INC/_PVM PARA'
365: C/#ELSEIF PARA(MPI)
366:   * include 'mpif.h'
367:   integer MPISTAT(MPI_STATUS_SIZE)
368: C/#ENDIF
369:   include 'INC/_TASKDT'
370: C
371:   Check %%%%%%%%%
372:   C   write(*,*) '(MVPCOM_RECV_R4) itsk ',itsk,' msgtag ',msgtag,
373:   C   & ' ndata ',ndata,' idtask ',idtask
374:   C %%%%%%%%%
375:   C
376: C/#IF PARA(PVM)
377:   C
378:   IT = -1
379:   if ( ITSK.gt.0 ) IT = TASKID(ITSK)
380:   call PVMFREC( IT, MSGTAG, IERR )
381:   call PVMFUNPACK( REAL4, RA, NDATA, 1, IERR )
382: C
383: C/#ELSEIF PARA(MPI)
384:   IT = MPI_ANY_SOURCE
385:   if ( ITSK.gt.0 ) IT = ITSK - 1
386: C/# IF DOUBLE_MPI_DATATYPE
387:   * call MPI_RECV( RA, NDATA, MPI_REAL8, IT, MSGTAG, MVP_COMM,
388:   * & MPISTAT, IERR )
389: C/# ELSE
390:   call MPI_RECV( RA, NDATA, MPI_REAL, IT, MSGTAG, MVP_COMM,
```

src/shared/mvpcommu.f

```
391:      &      MPISTAT, IERR )
392: C/# ENDIF
393: C/#ENDIF
394:      return
395: C
396:      end
397: C
398: C -----
399:      subroutine MVPCOM_RECV_R8( ITSK, MSGTAG,DA,      NDATA, IERR )
400: C -----
401:      real*8 DA(NDATA)
402:      include 'INC/_IUNIT'
403: C/#IF PARA(PVM)
404:      include 'INC/_PVM PARA'
405: C/#ELSEIF PARA(MPI)
406: *      include 'mpif.h'
407:      integer MPISTAT(MPI_STATUS_SIZE)
408: C/#ENDIF
409:      include 'INC/_TASKDT'
410: C
411: Check %%%%%%%%%
412: C      write(*,*) '(MVPCOM_RECV_R8) itsk ',itsk,' msgtag ',msgtag,
413: C      &      ' ndata ',ndata,' idtask ',idtask
414: C %%%%%%%%%
415: C
416: C/#IF PARA(PVM)
417: C
418:      IT      = -1
419:      if ( ITSK.gt.0 ) IT = TASKID(ITSK)
420:      call PVMFREC( IT, MSGTAG, IERR )
421:      call PVMFUNPACK( REAL8, DA, NDATA, 1, IERR )
422: C
423: C/#ELSEIF PARA(MPI)
424:      IT      = MPI_ANY_SOURCE
425:      if ( ITSK.gt.0 ) IT = ITSK - 1
426:      call MPI_RECV( DA, NDATA, MPI_DOUBLE_PRECISION, IT, MSGTAG,
427:      &      MVP_COMM, MPISTAT, IERR )
428: C/#ENDIF
429:      return
430: C
431:      end
432: C
433: C -----
434:      subroutine MVPCOM_RECV_CH( ITSK, MSGTAG,CH,      NDATA, IERR )
435: C -----
436:      character*(*) CH
437:      include 'INC/_IUNIT'
438: C/#IF PARA(PVM)
439:      include 'INC/_PVM PARA'
440: C/#ELSEIF PARA(MPI)
441: *      include 'mpif.h'
442:      integer MPISTAT(MPI_STATUS_SIZE)
443: C/#ENDIF
444:      include 'INC/_TASKDT'
445: C
446: C
447: C/#IF PARA(PVM)
448: C
449:      IT      = -1
450:      if ( ITSK.gt.0 ) IT = TASKID(ITSK)
451:      call PVMFREC( IT, MSGTAG, IERR )
452:      call PVMFUNPACK( STRING, CH, NDATA, 1, IERR )
453: C
454: C/#ELSEIF PARA(MPI)
455:      IT      = MPI_ANY_SOURCE
```

```
456:      if ( ITSK.gt.0 ) IT = ITSK - 1
457:      call MPI_RECV( CH, NDATA, MPI_CHARACTER, IT, MSGTAG, MVP_COMM,
458:      &      MPISTAT, IERR )
459: C/#ENDIF
460:      return
461: C
462:      end
463: C
464: C
465: C -----
466:      subroutine MVPCOM_BCAST_I4( ITSK, MSGTAG,IA,      NDATA, IERR )
467: C -----
468:      integer IA(NDATA)
469:      include 'INC/_IUNIT'
470: C/#IF PARA(PVM)
471:      include 'INC/_PVM PARA'
472: C/#ELSEIF PARA(MPI)
473: *      include 'mpif.h'
474: C/#ENDIF
475:      include 'INC/_TASKDT'
476: C
477: C
478: C/#IF PARA(PVM)
479: C
480:      if ( IDTASK.eq.ITSK ) then
481: check %%%%%%%%%
482: c      write(*,*) idtask,
483: c      &      '%%% bcast_i4 itsk ',itsk,' msgtag ',msgtag,
484: c      &      ' ndata ',ndata
485: c %%%%%%%%%
486:      call PVMFINITSEND( PVMDEFAULT, IERR )
487:      call PVMFPACK( INTEGER4, IA, NDATA, 1, IERR )
488:      call PVMFMCAST( NTASK, TASKID(1), MSGTAG, IERR )
489: check %%%%%%%%%
490: c      write(*,*) '%%% bcast_i4 sent '
491: c %%%%%%%%%
492: C
493:      do 100 I = 1, NTASK
494:      if ( I.ne.IDTASK ) then
495: check %%%%%%%%%
496: c      write(*,*) '%%% bcast_i4 receive iok '
497: c %%%%%%%%%
498:      call PVMFREC( TASKID(I), MSGTAG, INFO )
499:      call PVMFUNPACK( INTEGER4, IOK, 1, 1, IERR )
500:      end if
501: 100      continue
502:      else
503: check %%%%%%%%%
504: c      write(*,*) idtask,
505: c      &      '%%% bcast_i4 itsk ',itsk,' msgtag ',msgtag,
506: c      &      ' ndata ',ndata
507: c %%%%%%%%%
508:      call PVMFREC( TASKID(ITSK), MSGTAG, IERR )
509:      call PVMFUNPACK( INTEGER4, IA, NDATA, 1, IERR )
510: check %%%%%%%%%
511: c      write(*,*) '%%% bcast_i4 received'
512: c %%%%%%%%%
513: C
514:      IOK      = 0
515: check %%%%%%%%%
516: c      write(*,*) '%%% bcast_i4 send iok'
517: c %%%%%%%%%
518:      call PVMFINITSEND( PVMDEFAULT, IERR )
519:      call PVMFPACK( INTEGER4, IOK, 1, 1, IERR )
520:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
```

src/shared/mvpcommu.f

```

521:         end if
522: C
523: C/#ELSEIF PARA(MPI)
524: C/# IF DOUBLE_MPI_DATATYPE
525: *       call MPI_BCAST( IA, NDATA, MPI_INTEGER8, ITSK-1, MVP_COMM, IERR )
526: C/# ELSE
527:       call MPI_BCAST( IA, NDATA, MPI_INTEGER, ITSK-1, MVP_COMM, IERR )
528: C/# ENDIF
529: C/#ENDIF
530:       return
531: C
532:       end
533: C
534: C -----
535:       subroutine MVPCOM_BCAST_I8( ITSK, MSGTAG, I8A, NDATA, IERR )
536: C -----
537: C/#IF INTEGER8
538: *       integer*8 I8A(NDATA)
539: C/#ELSE
540:       integer I8A(NDATA)
541: C/#ENDIF
542:       include 'INC/_IOUNIT'
543: C/#IF PARA(PVM)
544:       include 'INC/_PVM PARA'
545: C/#ELSEIF PARA(MPI)
546: *       include 'mpif.h'
547: C/#ENDIF
548:       include 'INC/_TASKDT'
549: C
550: C
551: C/#IF PARA(PVM)
552: C
553:       if ( IDTASK.eq.ITSK ) then
554: check %%%%%%%%%
555: c       write(*,*) idtask,
556: c &       '%%% bcast_i4 itsk ',itsk,' msgtag ',msgtag,
557: c &       ' ndata ',ndata
558: c %%%%%%%%%
559:       call PVMFINITSEND( PVMDEFAULT, IERR )
560: C/# IF INTEGER8
561: *       call PVMFPACK( INTEGER8, I8A, NDATA, 1, IERR )
562: C/# ELSE
563:       call PVMFPACK( INTEGER4, I8A, NDATA, 1, IERR )
564: C/# ENDIF
565:       call PVMFMCAST( NTASK, TASKID(1), MSGTAG, IERR )
566: check %%%%%%%%%
567: c       write(*,*) '%%% bcast_i4 sent '
568: c %%%%%%%%%
569: C
570:       do 100 I = 1, NTASK
571:       if ( I.ne.IDTASK ) then
572: check %%%%%%%%%
573: c       write(*,*) '%%% bcast_i4 receive iok '
574: c %%%%%%%%%
575:       call PVMFRCV( TASKID(I), MSGTAG, INFO )
576:       call PVMFUNPACK( INTEGER4, IOK, 1, 1, IERR )
577:       end if
578:       100 continue
579:       else
580: check %%%%%%%%%
581: c       write(*,*) idtask,
582: c &       '%%% bcast_i4 itsk ',itsk,' msgtag ',msgtag,
583: c &       ' ndata ',ndata
584: c %%%%%%%%%
585:       call PVMFRCV( TASKID(ITSK), MSGTAG, IERR )

```

```

586: C/# IF INTEGER8
587: *       call PVMFUNPACK( INTEGER8, I8A, NDATA, 1, IERR )
588: C/# ELSE
589:       call PVMFUNPACK( INTEGER4, I8A, NDATA, 1, IERR )
590: C/# ENDIF
591: check %%%%%%%%%
592: c       write(*,*) '%%% bcast_i4 received'
593: c %%%%%%%%%
594: C
595:       IOK = 0
596: check %%%%%%%%%
597: c       write(*,*) '%%% bcast_i4 send iok'
598: c %%%%%%%%%
599:       call PVMFINITSEND( PVMDEFAULT, IERR )
600:       call PVMFPACK( INTEGER4, IOK, 1, 1, IERR )
601:       call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
602:       end if
603: C
604: C/#ELSEIF PARA(MPI)
605: C/# IF INTEGER8
606: *       call MPI_BCAST( I8A, NDATA, MPI_INTEGER8, ITSK-1, MVP_COMM, IERR )
607: C/# ELSE
608:       call MPI_BCAST( I8A, NDATA, MPI_INTEGER, ITSK-1, MVP_COMM, IERR )
609: C/# ENDIF
610: C/#ENDIF
611:       return
612: C
613:       end
614: C
615: C
616: C -----
617:       subroutine MVPCOM_BCAST_R4( ITSK, MSGTAG,RA, NDATA, IERR )
618: C -----
619:       real RA(NDATA)
620:       include 'INC/_IOUNIT'
621: C/#IF PARA(PVM)
622:       include 'INC/_PVM PARA'
623: C/#ELSEIF PARA(MPI)
624: *       include 'mpif.h'
625: C/#ENDIF
626:       include 'INC/_TASKDT'
627: C
628: C
629: C/#IF PARA(PVM)
630: C
631:       if ( IDTASK.eq.ITSK ) then
632:       call PVMFINITSEND( PVMDEFAULT, IERR )
633:       call PVMFPACK( REAL4, RA, NDATA, 1, IERR )
634:       call PVMFMCAST( NTASK, TASKID(1), MSGTAG, IERR )
635: C
636:       do 100 I = 1, NTASK
637:       if ( I.ne.IDTASK ) then
638:       call PVMFRCV( TASKID(I), MSGTAG, INFO )
639:       call PVMFUNPACK( REAL4, IOK, 1, 1, IERR )
640:       end if
641:       100 continue
642: C
643:       else
644: C
645:       call PVMFRCV( TASKID(ITSK), MSGTAG, IERR )
646:       call PVMFUNPACK( REAL4, RA, NDATA, 1, IERR )
647: C
648:       IOK = 0
649:       call PVMFINITSEND( PVMDEFAULT, IERR )
650:       call PVMFPACK( INTEGER4, IOK, 1, 1, IERR )

```


src/shared/mvpcommu.f

```

651:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
652:      end if
653: C
654: C/#ELSEIF PARA(MPI)
655: C/# IF DOUBLE_MPI_DATATYPE
656: *      call MPI_BCAST( RA, NDATA, MPI_REAL8, ITSK-1, MVP_COMM, IERR )
657: C/# ELSE
658:      call MPI_BCAST( RA, NDATA, MPI_REAL, ITSK-1, MVP_COMM, IERR )
659: C/# ENDIF
660: C/#ENDIF
661:      return
662: C
663:      end
664: C
665: C
666: C -----
667:      subroutine MVPCOM_BCAST_R8( ITSK, MSGTAG, DA, NDATA, IERR )
668: C -----
669:      real*8 DA(NDATA)
670:      include 'INC/_IUNIT'
671: C/#IF PARA(PVM)
672:      include 'INC/_PVMPPARA'
673: C/#ELSEIF PARA(MPI)
674: *      include 'mpif.h'
675: C/#ENDIF
676:      include 'INC/_TASKDT'
677: C
678: C
679: C/#IF PARA(PVM)
680: C
681:      if ( IDTASK.eq.ITSK ) then
682:      call PVMFINITSEND( PVMDEFAULT, IERR )
683:      call PVMFPACK( REAL8, DA, NDATA, 1, IERR )
684:      call PVMFMCAST( NTASK, TASKID(1), MSGTAG, IERR )
685: C
686:      do 100 I = 1, NTASK
687:      if ( I.ne.IDTASK ) then
688:      call PVMFRCV( TASKID(I), MSGTAG, INFO )
689:      call PVMFUNPACK( INTEGER4, IOK, 1, 1, IERR )
690:      end if
691: 100 continue
692:      else
693:      call PVMFRCV( TASKID(ITSK), MSGTAG, IERR )
694:      call PVMFUNPACK( REAL8, DA, NDATA, 1, IERR )
695: C
696:      IOK = 0
697:      call PVMFINITSEND( PVMDEFAULT, IERR )
698:      call PVMFPACK( INTEGER4, IOK, 1, 1, IERR )
699:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
700:      end if
701: C
702: C/#ELSEIF PARA(MPI)
703:      call MPI_BCAST( DA, NDATA, MPI_DOUBLE_PRECISION, ITSK-1, MVP_COMM,
704:      & IERR )
705: C/#ENDIF
706:      return
707: C
708:      end
709: C
710: C
711: C -----
712:      subroutine MVPCOM_BCAST_CH( ITSK, MSGTAG, CH, NDATA, IERR )
713: C -----
714:      character CH*(*)
715:      include 'INC/_IUNIT'

```

```

716: C/#IF PARA(PVM)
717:      include 'INC/_PVMPPARA'
718: C/#ELSEIF PARA(MPI)
719: *      include 'mpif.h'
720: C/#ENDIF
721:      include 'INC/_TASKDT'
722: C
723: C
724: C/#IF PARA(PVM)
725: C
726:      check %%%%%%%%%
727: c      write(*,*) idtask,
728: c      & '%%% bcast_ch itsk ',itsk,' msgtag ',msgtag,
729: c      & ' ndata ',ndata
730: c %%%%%%%%%
731:      if ( IDTASK.eq.ITSK ) then
732:      call PVMFINITSEND( PVMDEFAULT, IERR )
733:      call PVMFPACK( STRING, CH, NDATA, 1, IERR )
734:      call PVMFMCAST( NTASK, TASKID(1), MSGTAG, IERR )
735:      check %%%%%%%%%
736: c      write(*,*) '%%% bcast_ch sent '
737: c %%%%%%%%%
738: C
739:      do 100 I = 1, NTASK
740:      check %%%%%%%%%
741: c      write(*,*) '%%% bcast_ch iok ',i
742: c %%%%%%%%%
743:      if ( I.ne.IDTASK ) then
744:      call PVMFRCV( TASKID(I), MSGTAG, INFO )
745:      call PVMFUNPACK( INTEGER4, IOK, 1, 1, IERR )
746:      end if
747: 100 continue
748:      else
749:      call PVMFRCV( TASKID(ITSK), MSGTAG, IERR )
750:      call PVMFUNPACK( STRING, CH, NDATA, 1, IERR )
751:      check %%%%%%%%%
752: c      write(*,*) '%%% bcast_ch received '
753: c %%%%%%%%%
754: C
755:      IOK = 0
756:      call PVMFINITSEND( PVMDEFAULT, IERR )
757:      call PVMFPACK( INTEGER4, IOK, 1, 1, IERR )
758:      call PVMFSEND( TASKID(ITSK), MSGTAG, IERR )
759:      check %%%%%%%%%
760: c      write(*,*) '%%% bcast_ch sent iok '
761: c %%%%%%%%%
762:      end if
763: C
764: C/#ELSEIF PARA(MPI)
765:      call MPI_BCAST( CH, NDATA, MPI_CHARACTER, ITSK-1, MVP_COMM, IERR )
766: C/#ENDIF
767:      return
768: C
769:      end
770: C
771: C * subroutine mvpcom_sum_i4( itsk, msgtag, ia, ndata, i4buf, bufsize,
772: C ierr)
773: C * subroutine mvpcom_sum_i8( itsk, msgtag, i8a, ndata, i4buf, bufsize,
774: C ierr)
775: C * subroutine mvpcom_sum_r4( itsk, msgtag, ra, ndata, r4buf, bufsize,
776: C ierr)
777: C * subroutine mvpcom_sum_r8( itsk, msgtag, da, ndata, r8buf, bufsize,
778: C ierr)
779: C
780: C -----

```

src/shared/mvppcommu.f

```

781:      subroutine MVPCOM_SUM_I4( ITSK, MSGTAG,IA,   NDATA,
782:      &                          WORK, NWORK, IERR )
783: C -----
784:      integer IA(NDATA)
785:      integer WORK(NWORK)
786:      include 'INC/_IUNIT'
787: C/#IF PARA(PVM)
788:      include 'INC/_PVMPARA'
789: C/#ELSEIF PARA(MPI)
790: *      include 'mpif.h'
791: C/#ENDIF
792:      include 'INC/_TASKDT'
793: C
794: C/#IF PARA(PVM)
795: C
796: C ---- take summation tree (tournament) method on the first task task.
797: C
798:      IPID = IDTASK - 1
799:      do 130 M=1,NDATA,NWORK
800:          M2 = MIN(NDATA,M+NWORK-1)
801:          NM = M2 - M + 1
802:          M1 = M - 1
803: C
804:          IWIN = 1
805:          IDIGIT = 1
806: C
807:      100      continue
808:
809:          if( IDIGIT.le.NTASK .and. IWIN.eq.1 ) then
810: C
811: C          ... Ibit = and( idigit, ipid )
812: C
813: ccc          IBIT = MOD(IPID,2*IDIGIT)/IDIGIT
814:          IBIT = IAND( IPID, IDIGIT )
815: C
816: C          ... Imatch = or( idigit, ipid )
817: C
818: ccc          IMATCH = IPID + IBIT*IDIGIT
819:          IMATCH = IOR( IPID, IDIGIT )
820: C ---- ibit = 0 : get data ...
821:          if( IBIT.eq.0 .and. IMATCH.lt.NTASK ) then
822:              call MVPCOM_RECV_I4( IMATCH+1, MSGTAG, WORK, NM, INFO )
823:              do 110 i=1,NM
824:                  IA(M1+I) = IA(M1+I) + WORK(I)
825:              110      continue
826: C ---- ibit = 1 : send data ...
827:              else if( IBIT.ne.0 ) then
828:                  IWIN = 0
829:                  IMATCH = IPID - IDIGIT
830:                  call MVPCOM_SEND_I4( IMATCH+1, MSGTAG, IA, NM, INFO )
831: C ---- Exit tournament ..
832:                  goto 120
833:              endif
834: C ---- Advance to higher digit
835:                  IDIGIT = IDIGIT * 2
836:                  goto 100
837:              endif
838:      120      continue
839:      130      continue
840:
841:      IT0 = 1
842:      if ( ITSK.ne.IT0 ) then
843:          if( IDTASK.eq.ITSK ) then
844:              call MVPCOM_RECV_I4( IT0, MSGTAG, IA, NDATA, IERR )
845:          else if( IDTASK.eq.IT0 ) then

```

```

846:              call MVPCOM_SEND_I4( ITSK, MSGTAG, IA, NDATA, IERR )
847:          end if
848:      end if
849: C
850: C/#ELSEIF PARA(MPI)
851: C
852:      do 210 M=1,NDATA,NWORK
853:          M2 = min(NDATA,M+NWORK-1)
854:          NM = M2 - M + 1
855:          call MPI_REDUCE( IA(M), WORK, NM, MPI_INTEGER, MPI_SUM,
856:          &                          ITSK-1, MVP_COMM, IERR )
857:          if ( IDTASK.eq.ITSK ) then
858:              M1 = M - 1
859:              do 200 J=1,NM
860:                  IA(M1+J) = WORK(J)
861:              200      continue
862:          end if
863:      210      continue
864: C
865: C/#ENDIF
866: C
867:      return
868: C
869:      end
870: C
871: C -----
872:      subroutine MVPCOM_SUM_I8( ITSK, MSGTAG, I8A, NDATA,
873:      &                          WORK, NWORK, IERR )
874: C -----
875: C/#IF INTEGER8
876: *      integer*8 I8A(NDATA)
877: *      integer*8 WORK(NWORK)
878: C/#ELSE
879:      integer I8A(NDATA)
880:      integer WORK(NWORK)
881: C/#ENDIF
882:      include 'INC/_IUNIT'
883: C/#IF PARA(PVM)
884:      include 'INC/_PVMPARA'
885: C/#ELSEIF PARA(MPI)
886: *      include 'mpif.h'
887: C/#ENDIF
888:      include 'INC/_TASKDT'
889: C
890: C/#IF PARA(PVM)
891: C
892: C ---- take summation tree (tournament) method on the first task task.
893: C
894:      IPID = IDTASK - 1
895:      do 130 M=1,NDATA,NWORK
896:          M2 = MIN(NDATA,M+NWORK-1)
897:          NM = M2 - M + 1
898:          M1 = M - 1
899: C
900:          IWIN = 1
901:          IDIGIT = 1
902: C
903:      100      continue
904:
905:          if( IDIGIT.le.NTASK .and. IWIN.eq.1 ) then
906: C
907: C          ... Ibit = and( idigit, ipid )
908: C
909: ccc          IBIT = MOD(IPID,2*IDIGIT)/IDIGIT
910:          IBIT = IAND( IPID, IDIGIT )

```

src/shared/mvpcommu.f

```

911: C
912: C      ... Imatch = or( idigit, ipid )
913: C
914: ccc      IMATCH = IPID + IBIT*IDIGIT
915:      IMATCH = IOR( IPID, IDIGIT )
916: c ---- ibit = 0 : get data ...
917:      if( IBIT.eq.0 .and. IMATCH.lt.NTASK ) then
918:          call MVPCOM_RECV_I8( IMATCH+1, MSGTAG, WORK, NM, INFO )
919:          do 110 i=1,NM
920:              I8A(M1+I) = I8A(M1+I) + WORK(I)
921:          110 continue
922: c ---- ibit = 1 : send data ...
923:      else if( IBIT.ne.0 ) then
924:          IWIN = 0
925:          IMATCH = IPID - IDIGIT
926:          call MVPCOM_SEND_I8( IMATCH+1, MSGTAG, I8A, NM, INFO )
927: c ---- Exit tournament ..
928:          goto 120
929:      endif
930: c ---- Advance to higher digit
931:      IDIGIT = IDIGIT * 2
932:      goto 100
933:      endif
934: 120 continue
935: 130 continue
936:
937:      IT0 = 1
938:      if ( ITSK.ne.IT0 ) then
939:          if( IDTASK.eq.ITSK ) then
940:              call MVPCOM_RECV_I8( IT0, MSGTAG, I8A, NDATA, IERR )
941:          else if( IDTASK.eq.IT0 ) then
942:              call MVPCOM_SEND_I8( ITSK, MSGTAG, I8A, NDATA, IERR )
943:          end if
944:      end if
945: C
946: C/#ELSEIF PARA(MPI)
947: C
948:      do 210 M=1,NDATA,NWORK
949:          M2 = min(NDATA,M+NWORK-1)
950:          NM = M2 - M + 1
951:          call MPI_REDUCE( I8A(M), WORK, NM, MPI_INTEGER8, MPI_SUM,
952:              & ITSK-1, MVP_COMM, IERR )
953:          if ( IDTASK.eq.ITSK ) then
954:              M1 = M - 1
955:              do 200 J=1,NM
956:                  I8A(M1+J) = WORK(J)
957:              200 continue
958:          end if
959:      210 continue
960: C
961: C/#ENDIF
962: C
963:      return
964: C
965:      end
966: C
967: C -----
968:      subroutine MVPCOM_SUM_R4( ITSK, MSGTAG, RA, NDATA,
969:          & WORK, NWORK, IERR )
970: C -----
971:      real RA(NDATA)
972:      real WORK(NWORK)
973:      include 'INC/_IOUNIT'
974: C/#IF PARA(PVM)
975:      include 'INC/_PVMPARA'

```

```

976: C/#ELSEIF PARA(MPI)
977: *      include 'mpif.h'
978: C/#ENDIF
979:      include 'INC/_TASKDT'
980: C
981: C/#IF PARA(PVM)
982: C
983: C ---- take summation tree (tournament) method on the first task task.
984: C
985:      IPID = IDTASK - 1
986:      do 130 M=1,NDATA,NWORK
987:          M2 = MIN(NDATA,M+NWORK-1)
988:          NM = M2 - M + 1
989:          M1 = M - 1
990: c
991:          IWIN = 1
992:          IDIGIT = 1
993: c
994: 100      continue
995:          if( IDIGIT.le.NTASK .and. IWIN.eq.1 ) then
996: C
997: C      ... Ibit = and( idigit, ipid )
998: C
999: ccc      IBIT = MOD(IPID,2*IDIGIT)/IDIGIT
1000:          IBIT = IAND( IPID, IDIGIT )
1001: C
1002: C      ... Imatch = or( idigit, ipid )
1003: C
1004: ccc      IMATCH = IPID + IBIT*IDIGIT
1005:          IMATCH = IOR( IPID, IDIGIT )
1006: c ---- ibit = 0 : get data ...
1007:          if( IBIT.eq.0 .and. IMATCH.lt.NTASK ) then
1008:              call MVPCOM_RECV_R4( IMATCH+1, MSGTAG, WORK, NM, INFO )
1009:              do 110 i=1,NM
1010:                  RA(M1+I) = RA(M1+I) + WORK(I)
1011:              110 continue
1012: c ---- ibit = 1 : send data ...
1013:          else if( IBIT.ne.0 ) then
1014:              IWIN = 0
1015:              IMATCH = IPID - IDIGIT
1016:              call MVPCOM_SEND_R4( IMATCH+1, MSGTAG, RA, NM, INFO )
1017: c ---- Exit tournament ..
1018:              goto 120
1019:          endif
1020: c ---- Advance to higher digit
1021:          IDIGIT = IDIGIT * 2
1022:          goto 100
1023:          endif
1024: 120 continue
1025: 130 continue
1026: C
1027:      IT0 = 1
1028:      if ( ITSK.ne.IT0 ) then
1029:          if( IDTASK.eq.ITSK ) then
1030:              call MVPCOM_RECV_R4( IT0, MSGTAG, RA, NDATA, IERR )
1031:          else if( IDTASK.eq.IT0 ) then
1032:              call MVPCOM_SEND_R4( ITSK, MSGTAG, RA, NDATA, IERR )
1033:          end if
1034:      end if
1035: C
1036: C/#ELSEIF PARA(MPI)
1037: C
1038:      do 210 M=1,NDATA,NWORK
1039:          M2 = min(NDATA,M+NWORK-1)
1040:          NM = M2 - M + 1

```

src/shared/mvpcommu.f

```

1041:      call MPI_REDUCE( RA(M), WORK, NM, MPI_REAL, MPI_SUM,
1042:      &                ITSK-1, MVP_COMM, IERR )
1043:      if ( IDTASK.eq.ITSK ) then
1044:        M1 = M - 1
1045:        do 200 J=1,NM
1046:          RA(M1+J) = WORK(J)
1047:        200 continue
1048:      end if
1049:    210 continue
1050: C
1051: C/#ENDIF
1052: C
1053: return
1054: C
1055: end
1056: C
1057: C -----
1058:      subroutine MVPCOM_SUM_R8( ITSK, MSGTAG,R8,  NDATA,
1059:      &                        WORK, NWORK, IERR )
1060: C -----
1061:      real*8 R8(NDATA)
1062:      real*8 WORK(NWORK)
1063:      include 'INC/_IOUNIT'
1064: C/#IF PARA(PVM)
1065:      include 'INC/_PVMPARA'
1066: C/#ELSEIF PARA(MPI)
1067:      * include 'mpif.h'
1068: C/#ENDIF
1069:      include 'INC/_TASKDT'
1070: C
1071: C/#IF PARA(PVM)
1072: C
1073: C ---- take summation tree (tournament) method on the first task task.
1074: C
1075:      IPID = IDTASK - 1
1076:      do 130 M=1,NDATA,NWORK
1077:        M2 = MIN(NDATA,M+NWORK-1)
1078:        NM = M2 - M + 1
1079:        M1 = M - 1
1080: C
1081:        IWIN = 1
1082:        IDIGIT = 1
1083: C
1084:    100 continue
1085:      if( IDIGIT.le.NTASK .and. IWIN.eq.1 ) then
1086: C
1087:        ... Ibit = and( idigit, ipid )
1088: C
1089:        ccc      IBIT = MOD(IPID,2*IDIGIT)/IDIGIT
1090:        IBIT = IAND( IPID, IDIGIT )
1091: C
1092:        ... Imatch = or( idigit, ipid )
1093: C
1094:        ccc      IMATCH = IPID + IBIT*IDIGIT
1095:        IMATCH = IOR( IPID, IDIGIT )
1096: C ---- ibit = 0 : get data ...
1097:        if( IBIT.eq.0 .and. IMATCH.lt.NTASK ) then
1098:          call MVPCOM_RECV_R8( IMATCH+1, MSGTAG, WORK, NM, INFO )
1099:          do 110 i=1,NM
1100:            R8(M1+I) = R8(M1+I) + WORK(I)
1101:          110 continue
1102: C ---- ibit = 1 : send data ...
1103:        else if( IBIT.ne.0 ) then
1104:          IWIN = 0
1105:          IMATCH = IPID - IDIGIT

```

```

1106:      call MVPCOM_SEND_R8( IMATCH+1, MSGTAG, R8, NM, INFO )
1107: C ---- Exit tournament ..
1108:      goto 120
1109:    endif
1110: C ---- Advance to higher digit
1111:      IDIGIT = IDIGIT * 2
1112:      goto 100
1113:    endif
1114:  120 continue
1115:  130 continue
1116: C
1117:      IT0 = 1
1118:      if ( ITSK.ne.IT0 ) then
1119:        if( IDTASK.eq.ITSK ) then
1120:          call MVPCOM_RECV_R8( IT0, MSGTAG, R8, NDATA, IERR )
1121:        else if( IDTASK.eq.IT0 ) then
1122:          call MVPCOM_SEND_R8( ITSK, MSGTAG, R8, NDATA, IERR )
1123:        end if
1124:      end if
1125: C
1126: C/#ELSEIF PARA(MPI)
1127: C
1128:      do 210 M=1,NDATA,NWORK
1129:        M2 = min(NDATA,M+NWORK-1)
1130:        NM = M2 - M + 1
1131:        call MPI_REDUCE( R8(M), WORK, NM, MPI_DOUBLE_PRECISION,
1132:        &                MPI_SUM,
1133:        &                ITSK-1, MVP_COMM, IERR )
1134:      &      if ( IDTASK.eq.ITSK ) then
1135:        M1 = M - 1
1136:        do 200 J=1,NM
1137:          R8(M1+J) = WORK(J)
1138:        200 continue
1139:      end if
1140:    210 continue
1141: C
1142: C/#ENDIF
1143: C
1144: return
1145: C
1146: end
1147: C
1148: C
1149: C -----
1150:      subroutine MVPCOM_PROBE( ITSK, MSGTAG,INFO )
1151: C -----
1152: C ... returns probed task ID and/or message tag (Feb 2000)
1153: C
1154:      include 'INC/_IOUNIT'
1155: C/#IF PARA(PVM)
1156:      include 'INC/_PVMPARA'
1157:      integer IBUFINFO
1158: C/#ELSEIF PARA(MPI)
1159:      * include 'mpif.h'
1160:      logical FLAG
1161:      integer MPISTAT(MPI_STATUS_SIZE)
1162: C/#ENDIF
1163:      include 'INC/_TASKDT'
1164: C
1165:      INFO = 0
1166: C
1167: C/#IF PARA(PVM)
1168: C
1169:      IT = -1
1170:      if ( ITSK.gt.0 ) IT = TASKID(ITSK)

```

src/shared/mvpcommu.f

```

1171:      MT      = -1
1172:      if ( MSGTAG.gt.0 ) MT  = MSGTAG
1173:      call PVMFPROBE( IT, MT, IBUFINFO )
1174:      if ( IBUFINFO.lt.0 ) then
1175:        INFO = -1
1176:      else
1177:        INFO = 0
1178: C
1179: C      ... get tag / task-id from probed buffer
1180: C
1181: C      Current MVMCOM_* routines does not export PVM buffer ID.
1182: C
1183: C      if ( ITSK.le.0 .or. MSGTAG.le.0 ) then
1184: C      call PVMFBUFINFO( IBUFINFO, IBYTES, MT1, ITID1, INFO1 )
1185: C      if ( INFO1.eq.0 ) then
1186: C      if ( ITSK.le.0 ) then
1187: C
1188: C      ... converting into MVP task ID, stupid ;} yes ...
1189: C
1190: C      do 100 I = 1, NTASK
1191: C      if ( ITID1.eq.TASKID(I) ) then
1192: C      ITSK = I
1193: C      go to 110
1194: C      end if
1195: C      continue
1196: C      ... PVM task ID is not in MVP task ID ???
1197: C      INFO = -3
1198: C      continue
1199: C      end if
1200: C      if ( MSGTAG.le.0 ) then
1201: C      MSGTAG = MT1
1202: C      end if
1203: C      ... failed to get bufinfo
1204: C      else
1205: C      INFO = -2
1206: C      end if
1207: C      end if
1208: C      end if
1209: C
1210: C/#ELSEIF PARA(MPI)
1211: C
1212: C      IT      = MPI_ANY_SOURCE
1213: C      if ( ITSK.gt.0 ) IT = ITSK - 1
1214: C      MT      = MPI_ANY_TAG
1215: C      if ( MSGTAG.gt.0 ) MT  = MSGTAG
1216: C
1217: C      call MPI_IPROBE( IT, MT, MVP_COMM, FLAG, MPISTAT, IERR )
1218: C
1219: C      if ( FLAG ) then
1220: C      INFO      = 0
1221: C
1222: C      ... get MVP task ID and/or message tag when "MPI_ANY_*" is used.
1223: C
1224: C      if ( ITSK.le.0 ) then
1225: C      ITSK = MPISTAT(MPI_SOURCE) + 1
1226: C      end if
1227: C      if ( MSGTAG.le.0 ) then
1228: C      MSGTAG = MPISTAT(MPI_TAG)
1229: C      end if
1230: C      else
1231: C      INFO      = -1
1232: C      end if
1233: C/#ENDIF
1234: C      return
1235: C      end

```

```

1236: C
1237: C
1238: C -----
1239: C      subroutine MVMCOM_BARRIER( INFO )
1240: C -----
1241: C
1242: C      include 'INC/_IOUNIT'
1243: C/#IF PARA(PVM)
1244: C      include 'INC/_PVMPPARA'
1245: C/#ELSEIF PARA(MPI)
1246: C      *      include 'mpif.h'
1247: C/#ENDIF
1248: C      include 'INC/_TASKDT'
1249: C
1250: C      INFO = 0
1251: C      if ( NTASK.le.1 ) return
1252: C
1253: C/#IF PARA(PVM)
1254: C
1255: C      ... PVM: simulating barrier by send-receive
1256: C      of course, it should not be used for busy jobs ...
1257: C
1258: C      Why not using group function PVMFBARRIER() ???
1259: C      ... It's only lazyness of a programmer ...
1260: C
1261: C      MTAG = 65531
1262: C
1263: C      if ( IDTASK.eq.1 ) then
1264: C
1265: C      do 100 I = 2, NTASK
1266: C      call PVMFRECV( -1, MTAG, IERR )
1267: C      call PVMFUNPACK( INTEGER4, IDUMMY, 1, 1, INFO1 )
1268: C      continue
1269: C
1270: C      call PVMFINITSEND( PVMDEFAULT, IBUF )
1271: C      call PVMFPACK( INTEGER4, 9999, 1, 1, INFO1 )
1272: C      call PVMFMCAST( NTASK-1, TASKID(2), MTAG, INFO1 )
1273: C
1274: C      else
1275: C
1276: C      call PVMFINITSEND( PVMDEFAULT, IBUF )
1277: C      call PVMFPACK( INTEGER4, 9999, 1, 1, INFO1 )
1278: C      call PVMFSEND( TASKID(1), MTAG, INFO1 )
1279: C
1280: C      call PVMFRECV( TASKID(1), MTAG, INFO1 )
1281: C      call PVMFUNPACK( INTEGER4, IDUMMY, 1, 1, INFO1 )
1282: C
1283: C      end if
1284: C
1285: C/#ELSEIF PARA(MPI)
1286: C
1287: C      ... Hey, it's easy on MPI !
1288: C
1289: C      call MPI_BARRIER( MVP_COMM, INFO )
1290: C
1291: C/#ENDIF
1292: C
1293: C      return
1294: C      end
1295: C
1296: C -----
1297: C      subroutine MVMCOM_EXIT( INFO )
1298: C -----
1299: C
1300: C      include 'INC/_IOUNIT'

```

src/shared/mvpcommu.f

```
1301: C/#IF PARA(PVM)
1302:   include 'INC/_PVMPARA'
1303: C/#ELSEIF PARA(MPI)
1304: *   include 'mpif.h'
1305: C/#ENDIF
1306:   include 'INC/_TASKDT'
1307: C
1308: C
1309: C/#IF PARA(PVM)
1310:   call PVMFEXIT( INFO )
1311: C/#ELSEIF PARA(MPI)
1312:   call MPI_FINALIZE( INFO )
1313: C/#ENDIF
1314:   return
1315:   end
1316: C
1317: C
1318: C/#ENDIF
1319: C
1320: C   ... a dummy routine not to make empty program source ...
1321: C
1322:   subroutine MVPCDM( )
1323:   return
1324:   end
```

src/shared/mvpsync.f

```

1: C/#IF PARA(SX* CRAY*)
2: C=====
3: C   MVP task synchronization interface.
4: C
5: C purpose: interface routines to task synchronization on SMP systems.
6: C
7: C=====
8: C
9: C
10: C <procedures> .....
11: C
12: C
13: C * subroutine mvpsync_assign_barrier( i, ierr )
14: C
15: C   Assign i'th barrier.
16: C
17: C * subroutine mvpsync_barrier( i )
18: C
19: C   Barrier Synchronization by i'th barrier.
20: C
21: C * subroutine mvpsync_assign_lock( i, ierr )
22: C
23: C   Assign i'th lock.
24: C
25: C * subroutine mvpsync_lock( i )
26: C
27: C   Lock i'th lock.
28: C
29: C * subroutine mvpsync_unlock( i )
30: C
31: C   Unlock i'th lock.
32: C
33: C * subroutine mvpsync_assign_event( i, ierr )
34: C
35: C   Assign i'th post/wait event.
36: C
37: C * subroutine mvpsync_post( i )
38: C
39: C   post i'th post/wait event.
40: C
41: C * subroutine mvpsync_wait( i )
42: C
43: C   wait (and clear) i'th post/wait event.
44: C
45: C=====
46: C
47: C -----
48: C   subroutine mvpsync_assign_barrier( IB, IERR )
49: C -----
50: C
51: C   include 'INC/_TASKDT'
52: C
53: C/#IF PARA(SX*)
54: C   call PBASGN( IBARR(IB), NTASK, 0 )
55: C/#ELSEIF PARA(CRAY*)
56: C   call BARASGN( IBARR(IB), NTASK )
57: C/#ENDIF
58: C   IERR = 0
59: C   return
60: C   end
61: C
62: C -----
63: C   subroutine mvpsync_assign_lock( IL, IERR )
64: C -----
65: C

```

```

66: C   include 'INC/_TASKDT'
67: C
68: C/#IF PARA(SX*)
69: C   call PLASGN( ILOCK(IL), 0 )
70: C/#ELSEIF PARA(CRAY*)
71: C   call LOCKASGN( ILOCK(IL) )
72: C/#ENDIF
73: C   IERR = 0
74: C   return
75: C   end
76: C
77: C -----
78: C   subroutine mvpsync_assign_event( IE, IERR )
79: C -----
80: C
81: C   include 'INC/_TASKDT'
82: C
83: C/#IF PARA(SX*)
84: C   call PEASGN(IPWEV(IE),1)
85: C/#ELSEIF PARA(CRAY*)
86: C   call EVASGN(IPWEV(IE))
87: C/#ENDIF
88: C   IERR = 0
89: C   return
90: C   end
91: C
92: C -----
93: C   subroutine mvpsync_barrier( IB )
94: C -----
95: C
96: C   include 'INC/_TASKDT'
97: C
98: C/#IF PARA(SX*)
99: C   call PBSYCN( IBARR(IB) )
100: C/#ELSEIF PARA(CRAY*)
101: C   call BARSYCN( IBARR(IB) )
102: C/#ENDIF
103: C   IERR = 0
104: C   return
105: C   end
106: C
107: C -----
108: C   subroutine mvpsync_lock( IL )
109: C -----
110: C
111: C   include 'INC/_TASKDT'
112: C
113: C/#IF PARA(SX*)
114: C   call PLLOCK( ILOCK(IL) )
115: C/#ELSEIF PARA(CRAY*)
116: C   call LOCKON( ILOCK(IL) )
117: C/#ENDIF
118: C
119: C   return
120: C   end
121: C
122: C -----
123: C   subroutine mvpsync_unlock( IL )
124: C -----
125: C
126: C   include 'INC/_TASKDT'
127: C
128: C/#IF PARA(SX*)
129: C   call PLUNLOCK( ILOCK(IL) )
130: C/#ELSEIF PARA(CRAY*)

```

src/shared/mvpsync.f

```
131:      call LOCKOFF( ILOCK(IL) )
132: C/#ENDIF
133: C
134:      return
135:      end
136: C
137: C -----
138:      subroutine mvpsync_post( IE )
139: C -----
140: C
141:      include 'INC/_TASKDT'
142: C
143: C/#IF PARA(SX*)
144:      call PEPOST( IPWEV(IE) )
145: C/#ELSEIF PARA(CRAY*)
146:      call EVPOST( IPWEV(IE) )
147: C/#ENDIF
148: C
149:      return
150:      end
151: C
152: C -----
153:      subroutine mvpsync_wait( IE )
154: C -----
155: C
156:      include 'INC/_TASKDT'
157: C
158: C/#IF PARA(SX*)
159:      call PEWAIT( IPWEV(IE) )
160:      call PECLEAR( IPWEV(IE) )
161: C/#ELSEIF PARA(CRAY*)
162:      call EVWAIT( IPWEV(IE) )
163:      call EVCLEAR( IPWEV(IE) )
164: C/#ENDIF
165: C
166:      return
167:      end
168: C
169: C/#ENDIF
170: C
171: C ... a dummy routine not to make empty program source ...
172: C
173:      subroutine MVPSNC( )
174:      return
175:      end
```


src/shared/namlst.f

```

1:      subroutine NAMLIST( JLATT, JTLLT, JSREG, NNames, TNAMS, LTNAMS,
2:      &                    NLTEMP, KLTSP, ILTSP, KCLSP, KCLST, NCREG,
3:      &                    NCREGS, LIMIT, KREGN, NINPZ, IDLAT, IPLAT,
4:      &                    KLATT, NVLAT, LTYP, NLATT, IUSP, NLTSPC,
5:      &                    NSMSPC, IUSP2, LINE )
6: C=====
7: C PURPOSE : CREATE REGION NAME LIST "TNAMS(NNames)" (CHARACTER*(LNAM)
8: C FROM REGION NAMES OR "FRAME" NAMES STORED IN KREGN,
9: C AND "SUBFRAME" NAMES DESCRIBED IN TEMPORARY FILE (UNIT 75)
10: C IF NECESSARY.
11: C * COUNT NUMBER OF TALLY-REGIONS BY COMBINATION OF
12: C REGIONS.
13: C
14: C CALLED IN : GEOMIN
15: C CALLS : GTLLIN (TO GET A RECORD FROM CURRENT INPUT FILE)
16: C DENTAK (TO GET NUMERIC VALUE OF A CHARACTER STRING )
17: C GTVVAL (TO GET NUMERIC VALUE OF A PROGRAM-VARIABLE )
18: C-----
19: C ARGUMENTS :
20: C
21: C === OUTPUT OR INPUT/OUTPUT VARIABLES ===
22: C
23: C NNames : TOTAL NUMBER OF NAMES (TO BE COUNTED IN THIS ROUTINE)
24: C TNAMS(NNames)
25: C : NAMES STORED AS CHARACTER*12 DATA.
26: C BEFORE CALLING THIS ROUTINE,
27: C STARTING POSITION IN ARRAY-DATA STORAGE ONLY GIVEN.
28: C LTNAMS : Starting position of TNAMS in character memory area
29: C ** LAST : STARTING POSITION OF AVAILABLE ARRAY-DATA STORAGE.
30: C **
31: C ** IT MUST BE THE STARTING POSITION OF TNAMS AT THE BEGINNING
32: C **
33: C ** OF THIS ROUTINE AND SET TO LTNAMS + NNames*(WORDS FOR 12
34: C ** CHARACTERS).
35: C KLTSP(2,NLTSPC)
36: C : (FRAME,LATTICE) -> FRAME NAME INDEX(KREGN) &
37: C LATTICE # (NAME INDEX WILL BE CHANGED TO INDEX TO TNAMS)
38: C ILTSP(NLTSPC+1)
39: C : POINTER TO KSPCL ARRAY FOR EACH (FRAME, LATTICE)
40: C COMBINATION. (MEMORY ALREADY RESEVED)
41: C KCLSP(NSMSPC)
42: C : SUBFRAME NAME TABLE (POINTERS TO TNAMS TABLE)
43: C FOR SUBFRAME-CELL COMBINATIONS (MEMORY ALREADY RESERVED)
44: C
45: C === REFERENCE ONLY ===
46: C
47: C LIMIT : LIMIT OF ARRAY-DATA STOREGE IN WORD.
48: C KREGN : REGION OR SUBFRAME NAMES INPUT IN ZONEIN ROUTINE.
49: C NINPZ : NUMBER OF INPUT-ZONES
50: C IPLAT(NLATT+1) : LATTICE CELL LIST POINTER
51: C NLATT : NUMBER OF LATTICE
52: C IUSP : I/O UNIT(=75). DATA OF SUBFRAME NAME OF EACH LATTICE-CELL
53: C BY (FRAME, LATTICE) COMBINATION. (COPIED FROM
54: C INPUT DATA IN ZONEIN ROUTINE).
55: C IUSP2 : WORKING I/O UNIT (=15) TO STORE NAMES APPEARED IN
56: C NAMES(...) LIST.
57: C NLTSPC : NUMBER OF (FRAME , LATTICE) COMBINATION
58: C (COUNTED IN ZONEIN)
59: C NSMSPC : TOTAL NUMBER OF SUBFRAME-CELL SPECIFICATIONS.
60: C (SUMMED IN ZONEIN)
61: C=====
62:      include 'INC/_LNAM'
63: C
64:      integer NNames
65: C

```

```

66: C .... NLTEMP IS TEMPORARY SIZE OF 'TNAMS' ARRAY
67: C
68:      character*(LNAM) TNAMS(NLTEMP)
69:      integer KLTSP(2,NLTSPC), ILTSP(NLTSPC+1), KCLSP(NSMSPC),
70:      &          KCLST(NSMSPC)
71: C
72:      integer IDLAT(NLATT), KLATT(*), NVLAT(4,NLATT), LTYP(NLATT)
73:      integer IPLAT(NLATT+1)
74: C
75:      character*(LNAM) KREGN(NINPZ)
76: C
77:      include 'INC/_WORDL'
78:      include 'INC/_IUNIT'
79: C
80:      character*72 LINE
81: C
82: C .... TEMPORARY WORKING STORAGE TO STORE SPACE NAME TEMPORARY ....
83:      parameter( MXNAME = 30000 )
84:      character*(LNAM) TNAME(MXNAME)
85:      character*(LNAM) TOKEN
86:      character*72 SNAME
87:      character*4 CTYP
88: C
89: C ... for character input of space
90:      parameter( MXCSYM = 128 )
91:      character*(MXCSYM) CSYM
92:      character*128 CWRK
93: C
94: C-----
95: C
96: C call GTLAST( LAST )
97: C
98: C if ( JTLLT.eq.0 ) then
99: C *****
100: C call LABEL( IOG, 'NAMES OF REGIONS' )
101: C *****
102: C else
103: C *****
104: C call LABEL( IOG, 'REGION NAMES IN ROOT-CELL (SUBSPACE #0)' )
105: C *****
106: C end if
107: C
108:      NNames = 0
109:      do 110 I = 1, NINPZ
110:          do 100 J = 1, NNames
111:              if ( TNAMS(J).eq.KREGN(I) ) go to 110
112:          100 continue
113: C
114:          NNames = NNames + 1
115: C
116:          if ( NNames.gt.NLTEMP ) then
117:              write(IMG,*) 'XXX Character memory is insufficient ',
118:              &          'to construct a list of region/space names.'
119:              write(IMG,*) ' Error occured ', I, 'th input-zone (',
120:              &          ' total input zone ', NINPZ, ')'
121:              call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
122:              stop 999
123:          end if
124: C
125:          TNAMS(NNames) = KREGN(I)
126:          110 continue
127: C
128:          write(IOG,'(5(2X,I4,1X,' <'>'> '))' ) (I,TNAMS(I),I=1,NNames)
129: C
130: C

```

src/shared/namlst.f

```

131: C=====
132: C Subframe space specification processing
133: C
134: C #SUBFRAME data were extracted in ZONEIN routine and stored on
135: C working I./O unit IUSP
136: C=====
137: C
138: C ITEM PN = 0
139: C
140: C NCSYM = 0
141: C CSYM = ' '
142: C
143: C KJNAME = 0
144: C KJSPC = 0
145: C
146: C ILTSPC = 0
147: C ILTI = 0
148: C
149: C if ( JTLT.ne.0 ) then
150: C
151: C *****
152: C call LABEL( IOG, 'NAME OF EACH SUBFRAME' )
153: C *****
154: C
155: C call RWIND( IUSP )
156: C call RWIND( IUSP2 )
157: C
158: C ...change input I/O unit to IUSP
159: C
160: C call RIUNIT( IUSP )
161: C ILTSPC = 0
162: C ILTSP(1) = 1
163: C
164: C
165: C 120 call FREADS( TOKEN, LTK, IEND )
166: C
167: C if ( IEND.ne.0 .or. TOKEN(:LTK).eq.'$END' ) go to 300
168: C if ( JSREG.ne.0.and.TOKEN(:LTK).eq.'#REGION' ) go to 300
169: C
170: C if ( TOKEN(:LTK).eq.'@SUBFRAME' ) then
171: C
172: C -----
173: C ..... GET FRAME NAME ....
174: C AND LATTICE ID # .....
175: C -----
176: C
177: C call CHREAD( ' ', TOKEN, '( ', LTK, ITERM, IEND )
178: C
179: C if ( IEND.ne.0 ) then
180: C write(IMG,*)
181: C & 'XXX UNEXPECTED END OF DATA IN GETTING SUBFRAME NAME.'
182: C go to 440
183: C end if
184: C
185: C call I4READ( ' ', ILT, NA, 1, IERR )
186: C
187: C if ( LTK.lt.LNAM ) TOKEN(LTK+1:LNAM) = ' '
188: C
189: C write(IOG,7000) ILT, TOKEN(:LTK)
190: C 7000 format(/4X,70('=')/5X,'UNIVERSE(LATTICE-ID) ',I6,
191: C & ' : FRAME-NAME ',<'>',A,'>'/4X,70('=')/)
192: C
193: C do 140 I = 1, ILTSPC
194: C if ( TOKEN(1:LNAM).eq.KREGN(KLTSP(1,I))
195: C & .and.ILT.eq.IDLAT(KLTSP(2,I)) ) then

```

```

196: C
197: C write(IMG,*) 'XXX SUBFRAME DATA FOR ', '( ',
198: C & KREGN(KLTSP(1,I)), ' LATTICE=', ILT, ' ) IS '
199: C write(IMG,*) ' DEFINED DOUBLEY OR MORE.'
200: C call CNTERR( 'FATAL' )
201: C
202: C ..... SKIP TO THE NEXT DATA SECTION ...
203: C
204: C 130 read(IUSP,'(A)') LINE
205: C if ( LINE(1:4).ne.'@END' ) go to 130
206: C go to 120
207: C end if
208: C 140 continue
209: C
210: C
211: C ILTSPC = ILTSPC + 1
212: C
213: C do 150 IZ = 1, NINPZ
214: C if ( TOKEN(1:LNAM).eq.KREGN(IZ) ) go to 160
215: C continue
216: C
217: C 160 KLTSP(1,ILTSPC) = IZ
218: C do 170 IL = 1, NLATT
219: C if ( ILT.eq.IDLAT(IL) ) then
220: C ILTI = IL
221: C go to 180
222: C end if
223: C 170 continue
224: C ILTI = NLATT
225: C 180 KLTSP(2,ILTSPC) = ILTI
226: C
227: C ... STG region(lattice) has "cell"s (base material and STGs)
228: C
229: C if ( LTYP(ILTI).eq.10 ) then
230: C ILTSP(ILTSPC+1) = ILTSP(ILTSPC)
231: C & + KLATT(IPLAT(ILTI)+9) + 1
232: C else
233: C ILTSP(ILTSPC+1) = ILTSP(ILTSPC) + IPLAT(ILTI+1)
234: C & - IPLAT(ILTI)
235: C end if
236: C
237: C ITEM PN = 0
238: C ITERM = 0
239: C
240: C NCSYM = 0
241: C CSYM = ' '
242: C
243: C KJNAME = 0
244: C KJSPC = 0
245: C
246: C go to 120
247: C
248: C -----
249: C INPUT SUBFRAME NAME-LIST & SUBFRAME-CELL DATA
250: C -----
251: C
252: C ..... SUBFRAME NAME LIST .....
253: C
254: C -----
255: C
256: C else if ( TOKEN(1:LTK).eq.'NAMES' ) then
257: C
258: C 190 call CHREAD( 'NAMES', TOKEN, ' )', LTK, ITERM, IEND )
259: C
260: C if ( IEND.ne.0 ) then

```

src/shared/namlst.f

```

261:      write(IMG,*)
262:      &      'XXX UNEXPECTED END OF DATA IN "NAMES(...)" INPUT.'
263:      go to 440
264:      end if
265: C
266:      if ( LTK.ne.0 ) then
267:      ITEMPN = ITEMPN + 1
268:      if ( ITEMPN.gt.MXNAME ) then
269:      write(IMG,*) 'XXX TOO MANY SUBFRAME NAMES IN ', '( ',
270:      &      KREGN(KLTSP(1,ILTSPEC)), ', ', KLTSP(2,ILTSPEC),
271:      &      ' )'. (MXNAME=', MXNAME, ' ')
272:      write(IMG,*) ' PLEASE MODIFY NAMLIST ',
273:      &      'ROUTINE AND RECOMPILE '
274:      call PRSTOP(1,
275:      &      'INTERNAL MEMORY OVERFLOW IN "NAMLIST" ROUTINE.'
276:      &      )
277:      stop 888
278:      end if
279: C
280:      TNAME(ITEMP) = TOKEN(1:LTK)
281:      write(10G,7020) ITEMPN, TNAME(ITEMP)
282: 7020      format(6X,'SUBFRAME-NAME NO. ',I3,' <',A,' > ')
283: C
284:      end if
285: C
286:      if ( ITERM.eq.0 ) go to 190
287: C
288:      KJNAME = 1
289: C
290:      go to 120
291: C
292: C
293: C
294: C -----
295: C ... SUBFRAME NAME # FOR EACH CELL ....
296: C -----
297:      else if ( TOKEN(1:LTK).eq.'SPACE' ) then
298: C
299:      NB      = ILTSP(ILTSPEC+1) - ILTSP(ILTSPEC)
300: C
301:      call I4READ( ' ', KCLSP(ILTSP(ILTSPEC)), NA, -NB, IERR )
302: C
303:      if ( NA.lt.NB ) then
304:      write(IMG,*) 'XXX INVALID NUMBER OF SUBFRAME '
305:      &      'DATA FOR ', ' ', ( ', KREGN(KLTSP(1,ILTSPEC)), ', ',
306:      &      KLTSP(2,ILTSPEC), ' )'.
307:      write(IMG,*) ' REQUIRED ', NB, ' DATA BUT ',
308:      &      'GIVEN ', NA, ' DATA. '
309:      call CNTERR( 'FATAL' )
310:      else if ( NA.gt.NB ) then
311:      write(IMG,*) ' !!! NUMBER OF SUBFRAME DATA FOR ', ' ', ( ',
312:      &      KREGN(KLTSP(1,ILTSPEC)), ', ', KLTSP(2,ILTSPEC),
313:      &      ' )'. EXCEEDS REQUIRED NUMBER.'
314:      write(IMG,*) ' ( REQUIRED ', NB, ' DATA BUT ',
315:      &      'GIVEN ', NA, ' DATA.) '
316:      call CNTERR( 'WARNING' )
317:      end if
318: C
319:      KJSPC = 1
320:      go to 120
321: C
322: C
323: C -----
324: C ... Subframe name symbol character ....
325: C -----

```

```

326: C
327:      else if ( TOKEN(1:LTK).eq.'CNAMES' ) then
328: C
329:      NCS      = 0
330: C
331:      200      call CHREAD( TOKEN(:LTK), CWRK, ' )', ML, ITERM, IEND1 )
332:      if ( IEND1.ne.0 ) then
333:      write(IMG,*)
334:      &      'XXX UNEXPECTED END OF DATA IN "CNAMES(...)" INPUT.'
335:      go to 440
336:      end if
337: C
338:      if ( ML.gt.0 ) then
339:      NCS      = NCS + 1
340:      if ( NCS.gt.ITEMP ) then
341:      write(IMG,*)
342:      &      'XXX(NAMLIST) Number of entry in CNAMES(...) exceeds',
343:      &      ' that of NAMES(...) or NAMES is not input yet.'
344:      call CNTERR( 'FATAL' )
345:      else if ( NCS.gt.MXCSYM ) then
346:      write(IMG,*)
347:      &      'XXX(LATTIN) Too many entry for CNAMES(...).',
348:      &      ' Program limit (MXCSYM) is ', MXCSYM, ' '
349:      call CNTERR( 'FATAL' )
350:      else
351:      CSYM(NCS:NCS) = CWRK(1:1)
352:      end if
353: C
354:      NCSYM = MIN(ITEMP,NCS,MXCSYM)
355: C
356:      if ( ML.gt.1 ) then
357:      write(IMG,*)
358:      &      'XXX(NAMLIST) Entry of CNAMES(...) must be a single character:',
359:      &      ' <', CWRK(:ML), '>'
360:      call CNTERR( 'FATAL' )
361:      end if
362:      end if
363: C
364:      if ( ITERM.eq.0 ) go to 200
365: C
366:      if ( NCS.ne.ITEMP ) then
367:      write(IMG,*)
368:      &      'XXX(NAMLIST) Number of entry for CNAMES(...) is not equal to ',
369:      &      'that of NAMES(...) or NAMES is not input.'
370:      call CNTERR( 'FATAL' )
371:      end if
372: C
373:      go to 120
374: C
375: C -----
376: C ... Subframe number for each cell position: symbol character
377: C -----
378: C
379:      else if ( TOKEN(:LTK).eq.'CSPACE' ) then
380: C
381:      if ( ITEMPN.eq.0 .or. CSYM.eq.' ' ) then
382:      write(IMG,*) 'XXX(NAMLIST) CSPACE(...) is found but ',
383:      &      ' space name (NAMES) and/or space symbol (CNAMES) ',
384:      &      ' is not input.'
385:      call CNTERR( 'FATAL' )
386:      end if
387: C
388:      KK      = ILTSP(ILTSPEC+1) - ILTSP(ILTSPEC)
389: C
390:      IKK      = 0

```

src/shared/namlst.f

```

391:      ICKERR = 0
392: C
393: 210      call CHREAD( TOKEN(:LTK), CWRK, ' ', ML, ITERM, IEND1 )
394:      if ( IEND1.ne.0 ) then
395:          write(IMG,*)
396:          &      'XXX UNEXPECTED END OF DATA IN "CSPACE(...)" INPUT.'
397:          go to 440
398:      end if
399:
400:      if ( ML.gt.0 ) then
401:          do 220 I = 1, ML
402:              IKK = IKK + 1
403:              K = INDEX(CSYM(:NCSYM),CWRK(I:I))
404:              if ( K.eq.0 ) then
405:                  write(IMG,*)
406:                  &      'XXX(NAMLST) Undefined space symbol <',
407:                  &      CWRK(I:I), '> in CSPACE. (', IKK,
408:                  &      'th character)'
409:                  call CNTERR( 'FATAL' )
410:              else if ( IKK.le.KK ) then
411:                  KCLSP(ILTSP(ILTSPC)+IKK-1) = K
412:              end if
413: 220          continue
414:      end if
415:      if ( ITERM.eq.0 ) go to 210
416: C
417:      if ( IKK.ne.KK ) then
418:          write(IMG,*)
419:          &      'XXX(NAMLST) Number of characters in CSPACE (',
420:          &      IKK, ') does not match required one (', KK, ')
421:          call CNTERR( 'FATAL' )
422:      end if
423: C
424:      KJSPC = 1
425:      go to 120
426: C
427: C-----
428: C      process subframe/space input
429: C-----
430: C
431:      else if ( TOKEN(1:LTK).eq.'@END' ) then
432: C
433: C
434: C
435: C      check KCLSP() ( input by SPACE()/CSPACE() )
436: C
437: C      if ( KJSPC.eq.0 ) then
438:          write(IMG,*)
439:          &      'XXX(NAMLST) No space data SPACE()/CSPACE() ',
440:          &      ' for this subframe.'
441:          call CNTERR( 'FATAL' )
442:      end if
443: C
444: C      ... REVERSE KCLSP DATA IN Y/Z DIVISION ...
445: C
446:      call RVLAT( NVLAT(1,ILTI), NVLAT(2,ILTI), NVLAT(3,ILTI),
447:          &      KCLSP(ILTSP(ILTSPC)) )
448: C
449:      ICHK1 = 0
450:      ICHK2 = 0
451: C
452:      do 230 J = ILTSP(ILTSPC), ILTSP(ILTSPC+1) - 1
453:          I = IPLAT(ILTI) + J - ILTSP(ILTSPC)
454: C
455: C      ... giving valid subframe name for empty cell position

```

```

456: C      is changed to "warning" for ease of CSPACE()
457: C      (Nov 1999)
458: C
459:      if ( KCLSP(J).ne.0.and.KLATT(I).eq.0 ) then
460:          KCLSP(J) = -KCLSP(J)
461:          if ( ICHK1.eq.0 ) then
462:              write(IMG,*) '!!! A "SUBFRAME" name is given ',
463:              &      'for a lattice position with no cell-assignment.'
464:              write(IMG,*) ' (indicated by minus sign in ',
465:              &      'the following list.)'
466:              call CNTERR( 'WARNING' )
467:          end if
468:          ICHK1 = ICHK1 + 1
469: C
470:      else if ( KCLSP(J).eq.0.and.KLATT(I).ne.0 ) then
471:          write(IMG,*) 'XXX "SUBFRAME" name is not given ',
472:          &      'for a lattice-cell.'
473:          write(IMG,*) ' (indicated by ', -999,
474:          &      'in the following list.)'
475:          KCLSP(J) = -999
476:          call CNTERR( 'FATAL' )
477:          ICHK2 = ICHK2 + 1
478:      end if
479: 230      continue
480: C
481:      write(IOG,
482:          &      '(/6X, '== SUBFRAME NAME #''''S FOR EACH CELL == '/')' )
483: C
484: C      .... for STGM region
485: C
486:      if ( LTYP(ILTI).eq.10 ) then
487:          KK = ILTSP(ILTSPC+1) - ILTSP(ILTSPC)
488:          call PRLATT( IOG, KK, 1, 1, KCLSP(ILTSP(ILTSPC)) )
489: C
490:          MBASE = KCLSP(ILTSP(ILTSPC)+KK-1)
491:          do 235 I = KK-1, 1, -1
492:              KCLSP(ILTSP(ILTSPC)+I) = KCLSP(ILTSP(ILTSPC)+I-1)
493:          235      continue
494:          KCLSP(ILTSP(ILTSPC)) = MBASE
495: C
496: C      .... for non STGM region
497: C
498:      else
499:          call PRLATT( IOG, NVLAT(1,ILTI), NVLAT(2,ILTI),
500:              &      NVLAT(3,ILTI), KCLSP(ILTSP(ILTSPC)) )
501: C
502: C      end if
503: C
504:      if ( ICHK2.ne.0.or.ICHK1.ne.0 ) then
505:          write(IOG, '(/1x,a,a)')
506:          &      ' "-999" is a cell to which subspace assignment failed',
507:          &      ' And other Minus signed data in this list indicate',
508:          &      ' positions with subspace name but no actual cell',
509:          &      ' is assigned in lattice input (KLATT).)'
510: C
511:      do 240 I = ILTSP(ILTSPC), ILTSP(ILTSPC+1) - 1
512:          if ( KCLSP(I).eq.-999 ) then
513:              KCLSP(I) = 0
514:          else if ( KCLSP(I).lt.0 ) then
515: C
516: C      ... subspace # in positions with KLATT()=0 is
517: C      set to zero.
518: C
519:          KCLSP(I) = 0
520:      end if

```

src/shared/namlst.f

```

521:      240      continue
522:      end if
523: C
524: C
525: C -----
526: C      convert KCLSP() to space #
527: C -----
528: C
529: C ..... IF 'NAMES' DATA DOES NOT EXIST. NUMBERS IN 'SPACE' EXCEPT
530: C      '0' ARE TAKEN AS SUBFRAME-NAME.
531: C
532: C      if ( ITEMPN.eq.0 ) then
533: C      do 260 I = ILTSP(ILTSPC), ILTSP(ILTSPC+1) - 1
534: C      if ( KCLSP(I).ne.0 ) then
535: C      TOKEN = ' '
536: C      write(TOKEN,'(i12)') KCLSP(I)
537: C      call CCOMP( TOKEN, LNAM, TOKEN, IFL )
538: C
539: C      do 250 J = 1, ITEMPN
540: C      if ( TOKEN(:LNAM).eq.TNAME(J) ) then
541: C      KCLSP(I) = J
542: C      go to 260
543: C      end if
544: C      250      continue
545: C      ITEMPN = ITEMPN + 1
546: C
547: C
548: C      if ( ITEMPN.gt.MXNAME ) then
549: C      write(IMG,*) ' XXX TOO MANY SUBFRAME NAMES ',
550: C      &      ' IN ', '( ', KREGN(KLTSP(1,ILTSPC)), ', ',
551: C      &      KLTSP(2,ILTSPC), ')'. MXNAME=, MXNAME
552: C      write(IMG,*) ' PLEASE MODIFY "NAMLST" '
553: C      &      ' ROUTINE AND RECOMPILE '
554: C      call PRSTOP( 1,
555: C      &      ' INTERNAL MEMORY OVERFLOW IN "NAMLST" ROUTINE.'
556: C      &      )
557: C      stop 888
558: C      end if
559: C
560: C      TNAME(ITEMPN) = TOKEN(:LNAM)
561: C      KCLSP(I) = ITEMPN
562: C      end if
563: C      260      continue
564: C      else
565: C      IMX = 0
566: C      do 270 I = ILTSP(ILTSPC), ILTSP(ILTSPC+1) - 1
567: C      IMX = MAX(KCLSP(I),IMX)
568: C      270      continue
569: C
570: C      if ( IMX.gt.ITEMPN ) then
571: C      write(IMG,*) ' XXX SUBFRAME # IN "SPACE(...)" DATA ',
572: C      &      ' EXCEEDS THE NUMBER OF NAMES IN NAME-LIST ',
573: C      &      '( ', KREGN(KLTSP(1,ILTSPC)), ', ',
574: C      &      KLTSP(2,ILTSPC), ')'. '
575: C      write(IMG,*) ' NAME LIST HAS ONLY ', ITEMPN,
576: C      &      ' NAMES BUT ', IMX, ' NAME WAS SPECIFIED.'
577: C      call CNTERR( 'FATAL' )
578: C      end if
579: C      if ( IMX.eq.0 ) then
580: C      write(IMG,*) ' XXX ANY SUBFRAME # DATA ',
581: C      &      ' IS NOT GIVEN ', '( ', KREGN(KLTSP(1,ILTSPC)),
582: C      &      ', ', KLTSP(2,ILTSPC), ')'. '
583: C      call CNTERR( 'FATAL' )
584: C      end if
585: C      end if

```

```

586: C
587: C ..... OUTPUT SUBFRAME NAMES ON TEMPORARY FILE ...
588: C
589: C      write(IUSP2,*) ITEMPN
590: C      write(IUSP2,'(6A12)') (TNAME(I),I=1,ITEMPN)
591: C
592: C ..... ADD SUBFRAME NAMES TO NAME LIST TNAMS
593: C
594: C      do 290 I = 1, ITEMPN
595: C      do 280 J = 1, NNAMES
596: C      if ( TNAME(I).eq.TNAMS(J) ) go to 290
597: C      280      continue
598: C
599: C      NNAMES = NNAMES + 1
600: C
601: C      if ( NNAMES.gt.NLTEMP ) then
602: C      write(IMG,*) 'XXX Character memory is insufficient ',
603: C      &      ' to construct a list of region/space names.'
604: C      call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
605: C      stop 999
606: C      end if
607: C
608: C      TNAMS(NNAMES) = TNAME(I)
609: C
610: C      290      continue
611: C
612: C      go to 120
613: C
614: C      .... data which cannot be specified in #SUBFRAME block
615: C
616: C      else
617: C      write(IMG,*) 'XXX INVALID DATA ITEM <', TOKEN(1:LTK),
618: C      &      '> FATAL ERROR '
619: C      call CNTERR( 'FATAL' )
620: C      go to 120
621: C      end if
622: C      go to 120
623: C
624: C      300      continue
625: C      end if
626: C
627: C
628: C
629: C =====
630: C      Tally region processing
631: C =====
632: C
633: C
634: C -----
635: C      ADD NAMES FOR USER-DEFINED TALLY REGIONS BY COMBINATIONS OF REGIONS.
636: C
637: C      ADD SPECIAL CHARACTER '@' ON THE HEAD OF NAME OF THE USER-DEFINED
638: C      TALLY REGION IF NECESSARY..
639: C
640: C      DATA TYPE :
641: C
642: C      1. ADD <REGION-NAME> ( REGIONS ... )
643: C      2. DEFINE <REGION-NAME> ( REGIONS ... ) (DEFINE -> DEF OK! )
644: C -----
645: C
646: C
647: C      NCREG = 0
648: C      NCREGS = 0
649: C      if ( JSREG.ne.0 ) then
650: C      if ( JTLLT.eq.0 ) then

```

src/shared/namlst.f

```

651:      call RWIND( IUSP )
652: 310      read(IUSP,'(A)') LINE
653:          K      = INDEX(LINE,'#REGION')
654:          if ( K.eq.0 .or. (K.gt.1.and.LINE(:K-1).ne.' ') ) go to 310
655:          call RIUNIT( IUSP )
656:      end if
657: C
658: C *****
659: C call LABEL( IOG,
660: C &      'TALLY REGION NAMES SPECIFIED IN "#TALLY REGION" DATA BLOCK'
661: C &      )
662: C *****
663: C
664: 320      LINE      = ' '
665: C
666: C ---- GET MODE (ADD/DEFINE) OR TALLY-REGION NAME ----
667: C
668: C call CHREAD( ' ', LINE, '(', LTK, ITERM, IEND )
669: C
670: C ... IEND >< 0 means end of temporary file here, and handling
671: C for this end-of-data in unnecessary.
672: C
673: C if ( IEND.eq.0.and.LINE(:LTK).ne.'$END' ) then
674: C
675: C if ( LINE(:LTK).eq.'#REGION' ) go to 320
676: C
677: C ..... DEFINITION MODE :
678: C ADD : ADD AS NEW REGION & TALLIES OF COMPONENT REGION
679: C WILL BE SAVED.
680: C DEFINE : DEFINE NEW REGION & TALLIES OF COMPONENT REGION
681: C ARE NOT TAKEN.
682: C
683: C IF( LTK.GE.3 .AND.
684: C @      (LINE(1:3).EQ.'ADD' .OR. LINE(1:3).EQ.'DEF' )) GOTO 2000
685: C
686: C .... MODE STRING FOUND ..... ( NOT TERMINATED BY '(' )
687: C
688: C if ( ITERM.eq.0 ) then
689: C if ( LTK.lt.3
690: C &      .or.
691: C &      (LTK.ge.3
692: C &      .and.(LINE(1:3).ne.'ADD'.and.LINE(1:3).ne.'DEF' ) )
693: C &      then
694: C &          write(IMG,'(/lx,a,a,a/)')
695: C &          '!!! INVALID TALLY-REGION MODE <', LINE(:LTK),
696: C &          '>. IGNORED.'
697: C &          call CNTERR( 'WARNING' )
698: C &          end if
699: C &          go to 320
700: C &          end if
701: C
702: C .... TALLY-REGION NAME FOUND ... ( TERMINATED BY '(' )
703: C
704: C SNAME      = LINE(:LTK)
705: C LSN        = LTK
706: C
707: C NCREG      = NCREG + 1
708: C
709: C if ( LTK.gt.LNAM .or. LINE(1:1).ne.'@'.and.LTK.gt.LNAM-1 )
710: C &      then
711: C &          write(IMG,'(/lx,a,a,a/)')
712: C &          '!!! TRUNCATE TALLY-REGION NAME <', LINE(:LTK),
713: C &          '> BECAUSE IT IS TOO LONG.'
714: C &          call CNTERR( 'WARNING' )
715: C

```

```

716:      end if
717: C
718: C if ( LINE(1:1).ne.'@' ) then
719: C do 330 I = LTK, 1, -1
720: C     LINE(I+1:I+1) = LINE(I:I)
721: 330 C     continue
722: C     LTK      = LTK + 1
723: C     LINE(1:1) = '@'
724: C     LTK      = MIN(LNAM,LTK)
725: C end if
726: C
727: C
728: C ... add tally region name to name list ...
729: C
730: C do 340 J = 1, NNAMES
731: C if ( LINE(1:LTK).eq.TNAMS(J) ) then
732: C write(IMG,'(/lx,a,a,a/)') '!!! TALLY-REGION NAME <',
733: C &      LINE(1:LTK),
734: C &      '> IS ALREADY USED. CHANGING THE NAME IS RECOMMENDED.'
735: C call CNTERR( 'WARNING' )
736: C write(IMG,7040) NCREG, TNAMS(J)
737: C go to 350
738: C end if
739: 340 C continue
740: C
741: C
742: C NNAMES = NNAMES + 1
743: C
744: C if ( NNAMES.gt.NLTEMP ) then
745: C
746: C LM      = LAST + LNAM*NNAMES/(8/MWORD) + 2
747: C call MEMERR( 'NAMLST', ' TALLY-REGION NAME LIST. ',
748: C &      LSIZ(LM), LIMIT )
749: C stop 999
750: C
751: C write(IMG,*) 'XXX Character memory is insufficient ',
752: C &      'to construct a list of region/space names.'
753: C call PRSTOP( 1, 'CHARACTER MEMORY OVER.' )
754: C stop 999
755: C end if
756: C
757: C TNAMS(NNAMES) = LINE(:LTK)
758: C write(IOG,7040) NCREG, TNAMS(NNAMES)
759: C
760: 7040 C format(/7X,'TALLY-REGION ',I3,': NAME <',A,'>'/7X,
761: C &      ' COMPONENTS; '/')
762: C
763: 350 C continue
764: C
765: C .... INPUT (COUNT) NAMES OF REGIONS COMPOSING TALLY-REGION ...
766: C
767: 360 C call CHREAD( ' ', LINE, ')', LTK, ITERM, IEND )
768: C
769: C if ( IEND.ne.0 ) then
770: C write(IMG,'(/lx,a/)')
771: C & 'XXX UNEXPECTED END OF DATA IN GETTING TALLY-REGION COMPONENTS.'
772: C go to 440
773: C end if
774: C
775: C NCREGS = NCREGS + 1
776: C write(IOG,'(7X,' REGION(S): ',A)') LINE(:LTK)
777: C
778: C if ( ITERM.ne.0 ) go to 320
779: C go to 360
780: C end if

```

src/shared/namlst.f

```

781:      end if
782: C
783: C-----
784: C   SORT NAME LIST IN ASCENDING ORDER
785: C-----
786: C
787: C   ..... ADDRESS OF END OF TNAMS ARRAY (LAST POSITION OF ARRAY
788: C           MEMORY AREA )
789: C
790: C   CTYPE = ' '
791: C   write(CTYPE,'(''C'',I3)') LNAM
792: C   call CCOMP( CTYPE, LEN(CTYPE), CTYPE, IPL )
793: C   LLC = ICLEN(CTYPE)
794: C
795: C   call RESIZEC( 'TNAMS', LTNAMS, NNAMES, CTYPE(:LLC), LASTC )
796: C
797: C   do 380 I = 1, NNAMES - 1
798: C
799: C       KMIN = I
800: C       LI = ICLEN(TNAMS(KMIN))
801: C
802: C       do 370 J = I + 1, NNAMES
803: C           LJ = ICLEN(TNAMS(J))
804: C           if ( ISTRCM(TNAMS(J)(:LI),TNAMS(KMIN)(:LI)).lt.0 ) then
805: C               KMIN = J
806: C               LI = LJ
807: C           end if
808: C       370 continue
809: C       if ( KMIN.ne.I ) then
810: C           TOKEN = TNAMS(I)
811: C           TNAMS(I) = TNAMS(KMIN)
812: C           TNAMS(KMIN) = TOKEN
813: C       end if
814: C   380 continue
815: C
816: C-----
817: C   KCLSP ( TEMPORARY SUBFRAME NAME # ) --> NAME # IN TNAMS LIST
818: C-----
819: C
820: C   if ( JTLT.ne.0 ) then
821: C       call RWIND( IUSP2 )
822: C       do 420 K = 1, ILTSPC
823: C           read(IUSP2,*) ITEMPN
824: C           read(IUSP2,'(6A12)') (TNAME(I),I=1,ITEMPN)
825: C           NN = ILTSP(K) - 1
826: C           do 400 N = 0, ITEMPN
827: C               do 390 I = ILTSP(K), ILTSP(K+1) - 1
828: C                   J = I - ILTSP(K)
829: C                   if ( KCLSP(I).eq.N ) then
830: C                       NN = NN + 1
831: C                       KCLST(NN) = J
832: C                   end if
833: C               390 continue
834: C           400 continue
835: C
836: C       do 410 I = ILTSP(K), ILTSP(K+1) - 1
837: C           if ( KCLSP(I).ne.0 ) then
838: C               call CBSINC( TNAMS, NNAMES, TNAME(KCLSP(I)), IPOS )
839: C               if ( IPOS.eq.0 ) then
840: C                   write(IMG,*) 'XXX SUBFRAME NAME MISMATCH !! <',
841: C                       & TNAME(KCLSP(I)),
842: C                       & '> NOT FOUND IN NAMELIST.'
843: C                   call CNTERR( 'FATAL' )
844: C               end if
845: C               KCLSP(I) = IPOS

```

```

846:      end if
847:      410 continue
848:      420 continue
849: C
850: C-----
851: C   .... KLTSP(1,I) --> POSITION IN TNAMS
852: C-----
853: C
854: C       do 430 I = 1, NLTSPC
855: C           call CBSINC( TNAMS, NNAMES, KREGN(KLTSP(1,I)), IPOS )
856: C           KLTSP(1,I) = IPOS
857: C       430 continue
858: C
859: C   end if
860: C
861: C   call RIUNIT( IOIN )
862: C
863: C
864: C   *****
865: C   if ( JTLT.ne.0 ) then
866: C       call LABEL( IOG, 'REGION/FNAME NAMES APPEARED IN INPUT DATA' )
867: C   end if
868: C   *****
869: C
870: C   write(IOG,'('' TOTAL NUMBER OF NAMES : ''I4//)') NNAMES
871: C   write(IOG,'(5(3X,I4,1X,''' <''A,''' >''))' ) (I,TNAMS(I),I=1,NNAMES)
872: C
873: C   return
874: C
875: C   ... generic 'END OF DATA' handling ...
876: C
877: C   440 call PRSTOP( 1, 'UNEXPECTED END OF INPUT DATA' )
878: C   stop 888
879: C   end

```

src/shared/neefli.f

```

1:      subroutine NEEFLI( IOW, JVMNT, JFISX, JHLAT, JDEBG, JJJNND,
2:      &
3:      &
4:      &
5:      &
6:      &
7:      &
8:      &
9:      &
10:     &
11:     &
12:     &
13:     &
14:     &
15:     &
16:     &
17:     C
18:     C
19: C====< FLIGHT >=====
20: C PURPOSE:  FREE-FLIGHT OF ONE-ZONE LOGIC FOR NEXT EVENT ESTIMATOR.
21: C          TAKE TALLY OF POINT DETECTOR.
22: C
23: C CALLED IN:  NXTFLI
24: C CALLS:     LFRAME,(DFRAME),RANUS,VMNTR0,VMNTR1,VMNTR
25: C
26: C=====
27: C
28: C === INTER-STACK DATA FLOW ===
29: C
30: C   FREE-FLIGHT STACK -----+-----> NEXT-ZONE-SEARCH STACK
31: C   (IMSFL,IMZFL,IMNFL)      |      (IMSNX,IMZNX,IMNNX)
32: C   (AFTER TALLY) +-----> DEAD PARTICLE STACK
33: C                        (IMDED,NDIMPT)
34: C
35: C === BANK DATA TO BE UPDATED ===
36: C
37: C   (XXXI,YYI,ZZZI) : POSITION
38: C   (AAAI,BBBI,CCCI) : DIRECTION (IF HEXAGONAL-LATTICE EXISTS)
39: C   LEVLI           : LATTICE LEVEL (IF HEXAGONAL-LATTICE EXISTS)
40: C   KLSFI           : SURFACE DATA POINTER (IF ON-SURFACE)
41: C
42: C === main arguments ===
43: C i   IBP : bank pointer
44: C i   SIGT(IBP(I) : total cross sections
45: C o   NDil : starting bank pointer of particles to be tallied
46: C       by next event estimator.
47: C o   ITRM : number of particles to be tallied by next event estimator.
48: C o   OPT(1:ITRM) : contribution of i'th particle
49: C=====
50:     implicit real*8(D)
51:     include '../shared/INC/_SIZES'
52:     include '../shared/INC/_TASKDT'
53: C
54: C .... PARTICLE BANK .....
55: C
56:     real*8 XXXI(IMPMAX), YYI(IMPMAX), ZZZI(IMPMAX), AAAI(IMPMAX),
57:     &
58:     real WWI(IMPMAX)
59:     real*8 TTTI(IMPMAX)
60:     integer ITTI(IMPMAX)
61:     integer IGGI(IMPMAX), LEVLI(IMPMAX), LZZI(IMPMAX,NEST),
62:     &
63:     &
64: C
65: C

```

```

66:     integer KDETP(IMPMAX)
67:     real*8 OPTI(IMPMAX), PATH(IMPMAX)
68: C
69: C
70: C
71: C
72: C
73: C
74: C
75: C
76: C
77: C
78: C
79: C
80: C
81: C
82: C .... STACKS .....
83: C
84: C
85: C
86: C
87: C
88: C .... GEOMETRY .....
89: C
90: C
91: C
92: C
93: C
94: C
95: C
96: C
97: C
98: C
99: C
100: C
101: C
102: C
103: C
104: C
105: C
106: C
107: C .... WORKING AREA (LENGTH = IMPMAX) .....
108: C
109: C
110: C
111: C
112: C
113: C
114: C
115: C
116: C
117: C
118: C
119: C
120: C
121: C
122: C
123: C
124: C
125: C
126: C
127: C
128: C .... GATHER VECTORS .....
129: C
130: C

```


src/shared/neefli.f

```

131:      do 110 I = 1, IMNFL(MZONE)
132:        X(I) = XXXI(IBP(I))
133:        Y(I) = YYYY(IBP(I))
134:        Z(I) = ZZZI(IBP(I))
135:        AI(I) = AAAI(IBP(I))
136:        BI(I) = BBBI(IBP(I))
137:        CI(I) = CCCI(IBP(I))
138:        OPT(I) = OPTI(IBP(I))
139:        IGI(I) = IGGI(IBP(I))
140:        IKL(I) = KLSFI(IBP(I))
141:        IKL2(I) = 0
142:        DINF = DINF
143: C
144:        ILST(I) = 0
145:      110 continue
146: C
147: C-----
148: C .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
149: C-----
150: C
151:      if ( JFISX.eq.0 .or. JJJNND.eq.0 ) then
152:        JSS = 0
153:        call SPEAR1( MZONE, KZDA, KZAA, SDA, IMNFL(MZONE), IMPMAX, JSS,
154:          & X, Y, Z, AI, BI, CI, DI, DLOC1, DLOC2, IKL, IKL2,
155:          & DUMMY1, DINF, DEPS )
156: C
157: C ... "lost" particle index to "1"
158: C
159: *VOCL LOOP,NOVREC
160:      do 120 I = 1, IMNFL(MZONE)
161:        if ( DI(I).eq.DINF ) then
162:          ILST(I) = 1
163:        end if
164:      120 continue
165:    end if
166: C
167: C ( DLOC1 and DLOC2 are free to use hereafter )
168: C-----
169: C ----- HEXAGONAL LATTICE -----
170: C ... CALCULATE DISTANCE TO LATTICE FRAME & COORDINATES IN UPPER LEVEL
171: C IFC(I) = N/0 = CROSS THE OUTER FRAME (N=ZONE# IN UPPER LEVEL)/ NO
172: C VALUE OF DI(I) MAY BE CHANGED.
173: C ( IZS=1, IZE=1, LLS AS IFC, DSDA0 TO DSDA5 AS XUP TO CUP )
174: C-----
175: C
176:      IHF = 0
177: C
178:      if ( JFISX.eq.0.and.JHLAT.ne.0 ) then
179:        if ( KCELL(MZONE).gt.0.and.ICTYP(KCELL(MZONE)).eq.2 ) then
180:          IHF = 1
181:          IPZZ(1) = 1
182:          IPZZ(2) = IMNFL(MZONE) + 1
183: C
184:          if ( JVMNT.ne.0 ) call VMNTR0( 11, IOW )
185:          call LFRAME( IOW, JDEBG, IMPMAX, NEST, NLATT, NLBZ, NZONE,
186:            & DINF, IBP, IPZZ, 1, 1, DI, IFC, XUP, YUP, ZUP, AUP,
187:            & BUP, CUP,
188:            & LEVLI, LZZI, LPOSI, LCRSI, KZMAT, KDALT, DALT,
189:            & NVLAT, SZLAT, CELSZ, IPLAT, KLATT, KSLAT, LTYP,
190:            & MLBZZ, X, Y, Z, AI, BI, CI, R )
191: C
192: *VOCL LOOP,NOVREC
193:      do 130 I = 1, IMNFL(MZONE)
194: C
195: C .... this case moves only one upper lattice level

```

```

196: C
197:        ILUP(I) = LEVLI(IBP(I)) - 1
198: C
199: C ... lost in frame distance calculation
200: C
201:        if ( DI(I).eq.DINF ) then
202:          ILST(I) = ILST(I) + 2
203:        end if
204:      130 continue
205:        if ( JVMNT.ne.0 ) call VMNTR2( 11 )
206: C
207: C ... woking area IKL is never used after this call
208: C
209: C
210: C
211: C
212: C ... distance to frames until current lattice level is
213: C calculated if flight path count is zero.
214: C
215:      else if ( JFISX.ne.0.and.(KCELL(MZONE).gt.0.or.JJJNND.gt.0) ) then
216:        IHF = 1
217:        KK = 0
218: *VOCL LOOP,NOVREC
219:      do 140 I = 1, IMNFL(MZONE)
220:        IFC(I) = 0
221:        if ( IBNK(IBP(I),KIBNK(5)).eq.0 ) then
222:          KK = KK + 1
223:        end if
224:      140 continue
225:        if ( KK.gt.0 ) then
226:          write(IOW,*) 'XXX(NEEFLI) Program error ?',
227:            & ' Invalid path flight counter(=0) '
228:          stop 666
229: C
230:          if ( JVMNT.ne.0 ) call VMNTR0( 11, IOW )
231:          call DFRAME( IOW, MZONE, IMNFL(MZONE), JDEBG, IMPMAX, NEST,
232:            & NLATT, NLBZ, NZONE, DINF, DEPS, X, Y, Z, AI, BI, CI,
233:            & IBP, DI, IFC, XUP, YUP, ZUP, AUP, BUP, CUP, ILUP,
234:            & ILST, LEVLI, LZZI, LPOSI, LCRSI, DBNK, KDBNK, MDBNK,
235:            & IBNK, KIBNK, MIBNK, KZMAT, KZDA, KZAA, SDA, KCELL,
236:            & KDALT, DALT, NVLAT, SZLAT, CELSZ, IPLAT, KLATT,
237:            & KSLAT, LTYP, MLBZZ, KZLBZ, KPLT, JKSF, IBP0, IBP1,
238:            & IBP2, XYZ1(1,1), XYZ1(1,2), XYZ1(1,3), ABC1(1,1),
239:            & ABC1(1,2), ABC1(1,3), XYZ2(1,1), XYZ2(1,2),
240:            & XYZ2(1,3), ABC2(1,1), ABC2(1,2), ABC2(1,3), DDI,
241:            & IKL, IKL2, DLOC1, DLOC2 )
242:          if ( JVMNT.ne.0 ) call VMNTR2( 11 )
243:        end if
244: C
245: C ... get frame distance and upper level postion etc. from
246: C banked data
247: C
248:      if ( IMNFL(MZONE)-KK.gt.0 ) then
249: *VOCL LOOP,NOVREC
250:      do 150 I = 1, IMNFL(MZONE)
251:        KL = IBNK(IBP(I),KIBNK(6))+LEVLI(IBP(I))-1
252:        KI = KDBNK(4) + KL - 1
253:        DDDD = DBNK(1,IBP(I),KI)
254:        if ( IBNK(1,IBP(I),KIBNK(5)).ne.0.and.DDDD.lt.DI(I) ) then
255:          IFC(I) = LZZI(1,IBP(I),KI)
256:          ILUP(I) = KL - 1
257:          K6 = KDBNK(6) + 6*(KL-1)
258:          AUP(I) = DBNK(1,IBP(I),K6+3)
259:          BUP(I) = DBNK(1,IBP(I),K6+4)
260:          CUP(I) = DBNK(1,IBP(I),K6+5)
261:          XUP(I) = DBNK(1,IBP(I),K6) - DDDD*AUP(I)

```

src/shared/neefli.f

```

261:          YUP(I) = DBNK(IBP(I),K6+1) - DDDD*BUP(I)
262:          ZUP(I) = DBNK(IBP(I),K6+2) - DDDD*CUP(I)
263:          DI(I) = DDDD
264:        end if
265:      150    continue
266:    end if
267:  end if
268: C-----
269: C .... CHECK LOST PARTICLE .....
270: C-----
271:       ISAFE = IMNFL(MZONE)
272:       II = 0
273:       do 160 I = 1, IMNFL(MZONE)
274:         if ( ILST(I).ne.0 ) II = II + 1
275:       160 continue
276: C
277: C .... COUNT THE NUMBER OF FREE FLIGHT ....
278: CCCI NCNTR(16) = NCNTR(16) + ISAFE - II
279: C
280:       if ( II.gt.0 ) then
281: C/#IF PARA(SX* CRAY)
282: *      call MVPSYNC_LOCK(2)
283: C/#ENDIF
284:       write(IOW,*) '!!!(NEEFL) NONE ', MZONE, ' ', II,
285: &      ' PARTICLES ARE LOST !!!'
286:       NLOST = NLOST + II
287:       ILOST = NDIMPT
288:       do 170 I = 1, IMNFL(MZONE)
289:         if ( ILST(I).ne.0 ) then
290:           ILOST = ILOST + 1
291:           IMDED(ILOST) = IBP(I)
292:         end if
293:       170 continue
294:       do 180 I = NDIMPT + 1, ILOST
295:         LL = IMDED(I)
296:         write(IOW,7000) LL, XXXI(LL), YYI(LL), ZZI(LL), AAI(LL),
297: &         BBBI(LL), CCCI(LL), KLSFI(LL)
298:       180 continue
299:       7000    format(1X,I6,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',
300: &         E12.5,' ETA=',E12.5,' XI=',E12.5,' SURFACE=',I6)
301: C/#IF PARA(SX* CRAY)
302: *      call MVPSYNC_UNLOCK(2)
303: C/#ENDIF
304:       NDIMPT = ILOST
305:       NIMPT = IMPMAX - NDIMPT
306: C
307: C-----
308: C ..... DELETE LOST PARTICLES .....
309: C-----
310:       ISAFE = 0
311: *VOCL LOOP,NOVREC
312:       do 190 I = 1, IMNFL(MZONE)
313:         if ( ILST(I).eq.0 ) then
314:           ISAFE = ISAFE + 1
315:           IBP(ISAFE) = IBP(I)
316:           DI(ISAFE) = DI(I)
317:           IKL2(ISAFE) = IKL2(I)
318:           if ( IHF.ne.0 ) then
319:             ILUP(ISAFE) = ILUP(I)
320:             IFC(ISAFE) = IFC(I)
321:             XUP(ISAFE) = XUP(I)
322:             YUP(ISAFE) = YUP(I)
323:             ZUP(ISAFE) = ZUP(I)
324:             AUP(ISAFE) = AUP(I)
325:             BUP(ISAFE) = BUP(I)

```

```

326:          CUP(ISAFE) = CUP(I)
327:        end if
328:      end if
329:    190    continue
330: C
331: C-----
332: C ..... GATHER UNLOST PARTICLES AGAIN .....
333: C-----
334:       do 200 I = 1, ISAFE
335:         X(I) = XXXI(IBP(I))
336:         Y(I) = YYI(IBP(I))
337:         Z(I) = ZZI(IBP(I))
338:         AI(I) = AAI(IBP(I))
339:         BI(I) = BBBI(IBP(I))
340:         CI(I) = CCCI(IBP(I))
341:         IGI(I) = IGGI(IBP(I))
342:         OPT(I) = OPTI(IBP(I))
343:       200 continue
344:     end if
345: C
346: C-----
347: C .... sampling of STG sphere position from
348: C       Nearest Neighbour Distribution (NND)
349: C-----
350: C
351:       if ( JJJNND.ne.0 ) then
352:         MLT = LATTNM(KZMAT(MZONE,1))
353:         NN = ISAFE
354: C
355: C       ... ISTG (cell buffer zone of STG cell region) is returned
356: C       ... IKL (cell position in STG lattice) is returned
357: C       if sampled distance to STG is smaller than DI(I)
358: C
359: C       XUP,YUP,ZUP here is not a coordinate in "upper level" in this
360: C       case.
361: C
362:       call SMPSTG( IOW, MLT, X, Y, Z, AI, BI, CI, DI, ISTG, IBP,
363: &       NN, PNND, KNND, NPNND, IPLAT, KLATT, KSLAT, SZLAT,
364: &       NLATT, CELSZ, IPCEL, PPPF, KPPF, NPPPF,
365: &       IRAND, LEVLI, DBNK, KDBNK, MDBNK, IBNK,
366: &       KIBNK, MIBNK, IMPMAX, XUP, YUP, ZUP, IKL, DLOC1, DLOC2,
367: &       R )
368: C
369: C ... override frame distance flag
370: C
371:       do 210 I = 1, ISAFE
372:         if ( ISTG(I).ne.0 ) then
373:           IFC(I) = ISTG(I)
374:           AUP(I) = AI(I)
375:           BUP(I) = BI(I)
376:           CUP(I) = CI(I)
377:           ILUP(I) = LEVLI(IBP(I))
378:         end if
379:       210 continue
380:     end if
381: C
382: C=====
383: C       Check whether particles pass through the current zone
384: C=====
385: C
386:       INZ1 = IMNNX(NZONE+1)
387:       INXT = 0
388:       NDII = NDIMPT
389:       ITRM = 0
390: C

```

src/shared/neefli.f

```

391: C      .... No particle crosses frame boundaries.
392: C
393: C      if ( IHF.eq.0 ) then
394: *VOCL LOOP,NOVREC
395:       do 220 I = 1, ISAFE
396: C
397: C-----
398: C      ..... PASS THROUGH THE CURRENT ZONE .....
399: C-----
400: C
401: C      if ( DI(I).lt.PATH(IBP(I)) ) then
402:       INXT = INXT + 1
403:       IMSNX(INZ1+INXT) = IBP(I)
404:       IMZNX(INZ1+INXT) = MZONE
405:       ILST(INXT) = I
406:       XXXI(IBP(I)) = X(I) + AI(I)*DI(I)
407:       YYI(IBP(I)) = Y(I) + BI(I)*DI(I)
408:       ZZI(IBP(I)) = Z(I) + CI(I)*DI(I)
409:       PATH(IBP(I)) = PATH(IBP(I)) - DI(I)
410:       OPTI(IBP(I)) = OPTI(IBP(I)) + DI(I)*SIGT(IBP(I))
411: C
412: C-----
413: C      ..... REACHED AIMING POINTS .....
414: C-----
415: C
416: C      else
417:       ITRM = ITRM + 1
418:       OPT(ITRM) = EXP(-(OPT(I)+PATH(IBP(I))*SIGT(IBP(I))))
419:       & * WWI(IBP(I))
420:       IMDED(NDI1+ITRM) = IBP(I)
421:     end if
422: 220 continue
423: C
424:       IMNNX(MZONE) = IMNNX(MZONE) + INXT
425: C
426: C      .... Some particles might crosse frame boundaries.
427: C
428: C      else
429:       IHH = 0
430: C
431: *VOCL LOOP,NOVREC
432:       do 230 I = 1, ISAFE
433: C
434: C-----
435: C      ..... PASS THROUGH THE CURRENT ZONE .....
436: C-----
437: C
438: C      if ( DI(I).lt.PATH(IBP(I)) ) then
439:       INXT = INXT + 1
440:       IMSNX(INZ1+INXT) = IBP(I)
441:       ILST(INXT) = I
442:       PATH(IBP(I)) = PATH(IBP(I)) - DI(I)
443:       OPTI(IBP(I)) = OPTI(IBP(I)) + DI(I)*SIGT(IBP(I))
444:       if ( IFC(I).eq.0 ) then
445:       IMZNX(INZ1+INXT) = MZONE
446:       XXXI(IBP(I)) = X(I) + AI(I)*DI(I)
447:       YYI(IBP(I)) = Y(I) + BI(I)*DI(I)
448:       ZZI(IBP(I)) = Z(I) + CI(I)*DI(I)
449:       IKL2(I) = 0
450:     else
451:       IHH = IHH + 1
452:       IMZNX(INZ1+INXT) = IFC(I)
453:       IFC(I) = -IFC(I)
454:       XXXI(IBP(I)) = XUP(I) + AUP(I)*DI(I)
455:       YYI(IBP(I)) = YUP(I) + BUP(I)*DI(I)

```

```

456:       ZZI(IBP(I)) = ZUP(I) + CUP(I)*DI(I)
457:       AAII(IBP(I)) = AUP(I)
458:       BBII(IBP(I)) = BUP(I)
459:       CCCI(IBP(I)) = CUP(I)
460:       LEVLI(IBP(I)) = ILUP(I)
461:     endif
462: C
463: C-----
464: C      ..... REACHED AIMING POINTS .....
465: C-----
466: C
467: C      else
468:       ITRM = ITRM + 1
469: c <<note>>
470: c gfortran ver.5 outputs a warning message about
471: c "IEEE_UNDERFLOW_FLAG". To eliminate the message, the following
472: c if statetment is introduced.
473: c      OPT(ITRM) = EXP(-(OPT(I)+PATH(IBP(I))*SIGT(IBP(I))))
474: c      & * WWI(IBP(I))
475:       DPOW = -(OPT(I)+PATH(IBP(I))*SIGT(IBP(I)))
476:       if(DPOW.lt.-500) then
477:       OPT(ITRM) = 0.d0
478:     else
479:       OPT(ITRM) = EXP(DPOW)*WWI(IBP(I))
480:     end if
481:     IMDED(NDI1+ITRM) = IBP(I)
482:   end if
483: 230 continue
484: C
485:       IMNNX(MZONE) = IMNNX(MZONE) + INXT - IHH
486: C
487:       if ( IHH.gt.0 ) then
488: *VOCL LOOP,SCALAR
489:       do 250 I = 1, ISAFE
490:       if ( IFC(I).lt.0 ) IMNNX(-IFC(I)) =
491:       & IMNNX(-IFC(I)) + 1
492: 250 continue
493:     end if
494: C
495: C      ... "cell position" LPOS is changed here for STG cell
496: C
497:       if ( JJJNND.ne.0 ) then
498: *VOCL LOOP,NOVREC
499:       do 260 I = 1, ISAFE
500:       if ( IFC(I).lt.0.and.ISTG(I).ne.0 ) then
501:       LPOSI(IBP(I),LEVLI(IBP(I))) = IKL(I)
502:     end if
503: 260 continue
504:   end if
505: end if
506: C
507:       IMNNX(NZONE+1) = IMNNX(NZONE+1) + INXT
508:       NDIMPT = NDI1 + ITRM
509:       NIMPT = NIMPT - ITRM
510: C
511: C ==== update of remaining bank data =====
512: C      bank pointer : IMSNX(INZ1+1) ---> IMSNX(INZ1+INXT)
513: C      working array pointer : ILST(I),I=1,INXT
514: C
515:       do 270 I = 1, INXT
516:       KLSFI(IMSNX(INZ1+I)) = IKL2(ILST(I))
517: 270 continue
518: C
519: C      .... count up path flight counter here
520: C      not necessary to change bank data in KI7 because of no collision

```

src/shared/neefli.f

```
521: C
522:     KI5      = KIBNK(5)
523: C     KI7      = KIBNK(7)
524:     if ( KI5.ne.0 ) then
525: *VOCL LOOP,NOVREC
526:         do 280 I = 1, INXT
527:             IBPI = IMSNX(INZ1+I)
528:             if ( IBNK(IBPI,KI5).lt.0 ) then
529:                 IBNK(IBPI,KI5) = 1
530:             else
531:                 IBNK(IBPI,KI5) = IBNK(IBPI,KI5) + 1
532:             end if
533: 280     continue
534:     end if
535: C
536: C ... reduce distance to frame in lattice levels
537: C
538:     if ( JFISX.ne.0 ) then
539:         MXLV = 0
540:         KK = 0
541: *VOCL LOOP,NOVREC
542:         do 290 I = 1, INXT
543:             IBPI = IMSNX(INZ1+I)
544:             if ( IBNK(IBPI,KI5).ne.0 ) then
545:                 MXLV = MAX(MXLV,LEVLI(IBPI))
546:                 KK = KK + 1
547:             end if
548: 290     continue
549:             if ( MXLV.gt.0.and.KK.gt.0 ) then
550:                 KD4 = KDBNK(4)
551:                 do 300 LV = 1, MXLV
552: *VOCL LOOP,NOVREC
553:                     do 310 I = 1, INXT
554:                         IBPI = IMSNX(INZ1+I)
555:                         if ( IBNK(IBPI,KI5).ne.0.and.LEVLI(IBPI).ge.LV
556: &                             ) then
557:                             DBNK(IBPI,KD4+LV-1) = DBNK(IBPI,KD4+LV-1)
558: &                             - DI(ILST(I))
559:                         end if
560: 310                     continue
561: 300                 continue
562:             end if
563:         end if
564: C
565: C
566: C
567: C ==== return with the data for tallies =====
568: C     bank pointer : IMDED(NDI1+1) ---> IMDED(NDI1+ITRM)
569: C     contribution : OPT(I),I=1,ITRM
570: C
571: C
572: C-----
573: C .... DELETE PARTICLES FROM FLIGHT STACK .....
574: C-----
575: C
576:     II = 0
577: *VOCL LOOP,NOVREC
578:     do 400 N = 1, IMNFL(NZONE+1)
579:         if ( IMZFL(N).ne.MZONE ) then
580:             II = II + 1
581:             IMSFL(II) = IMSFL(N)
582:             IMZFL(II) = IMZFL(N)
583:         end if
584: 400     continue
585: C
```

```
586:     IMNFL(NZONE+1) = IMNFL(NZONE+1) - IMNFL(MZONE)
587:     IMNFL(MZONE) = 0
588: C
589:     return
590:     end
```

src/shared/neesea.f

```

1:      SUBROUTINE NEESEA(IOW, JVMNT, JLATT, JSIMP, MZONE, NIMPT, NDIMPT,
2:      E      NLOST ,DEPS ,JMEM ,KMEMO ,MEMC ,MEMZ ,
3:      B      XXXI ,YYI ,ZZI ,AAAI ,BBBI ,CCCI ,
4:      B      WWI ,IZI ,IGGI ,TTTI ,LEVLI ,LZZI ,
5:      B      LPOSI ,LCRSI ,KLSFI ,
6:      S      IMSFL ,IMNFL ,IMZFL ,IMSNX ,IMNNX ,IMZNX ,
7:      S      IMDED ,IMDEFR ,
8:      *      IMSLT ,IMZLT ,IMNLT ,
9:      G      SDA ,KZMAT ,KZDA ,KZAA ,
10:     *      KCELL ,IPCEL ,MLBZZ ,
11:     T      NCNTR ,WCNTR ,
12:     W      X ,Y ,Z ,W ,IBP ,IFI ,
13:     W      FXYZ ,ISRF ,IZI ,IFL )
14: C=====
15: C  PURPOSE: NEXT ZONE SEARCH IN NEXT EVENT ESTIMATOR
16: C  CALLED IN: NXTEE
17: C  CALLS: JUDGE
18: C=====
19: C
20: C  === INTER-STACK DATA FLOW ===
21: C
22: C  NEXT-ZONE SEARCH STACK---+---+---> FREE-FLIGHT STACK
23: C  (IMSNX,IMZNX,IMNNX) | | (IMSFL,IMZFL,IMNFL)
24: C  DEAD PARTICLE STACK ---+---> LATTICE-SEARCH STACK
25: C  (IMDED,NDIMPT) | | (IMSLT,IMZLT,IMNLT)
26: C  +---+ DEAD-PARTICLE STACK
27: C  (IMDED,NDIMPT)
28: C  === BANK DATA TO BE UPDATED ===
29: C
30: C  IZZI : ZONE #
31: C
32: C=====
33: C  implicit real*8(D)
34: C  include 'INC/_SIZES'
35: C
36: C  ..... BANK .....
37: C
38: C  real*8 XXXI(IMPMAX), YYI(IMPMAX), ZZI(IMPMAX), AAI(IMPMAX),
39: C  & BBBI(IMPMAX), CCCI(IMPMAX)
40: C  real WWI(IMPMAX)
41: C  real*8 TTTI(IMPMAX)
42: C  integer IZZI(IMPMAX), IGGI(IMPMAX), LEVLI(IMPMAX),
43: C  & LZZI(IMPMAX,NEST), LPOSI(IMPMAX,NEST), LCRSI(IMPMAX,NEST),
44: C  & KLSFI(IMPMAX)
45: C
46: C  ..... STACK .....
47: C
48: C  integer IMSFL(IMPMAX), IMNFL(NZONE+1), IMZFL(IMPMAX),
49: C  & IMSNX(IMPMAX), IMNNX(NZONE+1), IMZNX(IMPMAX),
50: C  & IMDED(IMPMAX), IMSLT(IMPMAX), IMZLT(IMPMAX), IMNLT(*)
51: C
52: C  ..... GEOMETRY .....
53: C
54: C  real*8 SDA(NSDA)
55: C  integer KZMAT(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
56: C  & KMEMO(NZONE,NMEMO), MEMC(NZONE,NMEMO), MEMZ(NZONE),
57: C  & KCELL(NZONE), IPCEL(1), MLBZZ(NZONE)
58: C
59: C  ..... WORKING AREA .....
60: C
61: C  real*8 X(IMPMAX), Y(IMPMAX), Z(IMPMAX), W(IMPMAX)
62: C  real FXYZ(IMPMAX)
63: C  integer IBP(IMPMAX), IZI(IMPMAX), ISRF(IMPMAX)
64: C  logical IFI(IMPMAX), IFL(IMPMAX), JMEM
65: C

```

```

66: C
67: C  real*8 NCNTR(*), WCNTR(*)
68: C
69: C  include '../shared/INC/_PMLATT'
70: C  include '../shared/INC/_SFLATT'
71: C
72: C-----
73: C
74: C .... MZONE: > 0 : SEARCH FOR ZONE MZONE, < 0 : DEFERRED PARTICLES
75: C
76: C  if ( MZONE.lt.0 ) go to 290
77: C  if ( JVMNT.ne.0 ) call VMNTRI( 4, IMNNX(MZONE) )
78: C
79: C .... GATHER VECTORS .....
80: C
81: C  II = 0
82: C  do 100 I = 1, IMNNX(NZONE+1)
83: C    if ( IMZNX(I).eq.MZONE ) then
84: C      II = II + 1
85: C      IBP(II) = IMSNX(I)
86: C    end if
87: C  100 continue
88: C  do 110 I = 1, IMNNX(MZONE)
89: C    X(I) = XXXI(IBP(I)) + DEPS*AAAI(IBP(I))
90: C    Y(I) = YYI(IBP(I)) + DEPS*BBBI(IBP(I))
91: C    Z(I) = ZZI(IBP(I)) + DEPS*CCCI(IBP(I))
92: C    W(I) = WWI(IBP(I))
93: C  110 continue
94: C
95: C
96: C
97: C=====
98: C  .... BEGIN NEXT-ZONE-SEARCH .....
99: C=====
100: C
101: C
102: C .... SEARCH ZONE # KKZ+1 TO MMZ. ....
103: C
104: C
105: C
106: C  if ( JLATT.ne.0 ) then
107: C    ICEL = KCELL(MZONE)
108: C    MMZ = IPCEL(ICEL+1) - 1
109: C    if ( ICEL.gt.0 ) then
110: C      KKZ = IPCEL(ICEL) - 1
111: C    else
112: C      KKZ = 0
113: C    end if
114: C  else
115: C    ICEL = 0
116: C    KKZ = 0
117: C    MMZ = NZONE
118: C  end if
119: C
120: C  MMX = MIN(MMZ-KKZ,NMEMO)
121: C
122: C
123: C
124: C .... IMEMO = 1 MEANS THAT ALL ZONES IN KMEMO ARRAY HAS NOT BEEN
125: C  SEARCHED.
126: C
127: C
128: C  IMEMO = 1
129: C  INX = IMNNX(MZONE)
130: C  INFL = IMNFL(NZONE+1)

```

src/shared/neesea.f

```

131: CCCCC IF(JREFL.NE.0) IRCF = NBREF(NR1)
132:       ILAT = 0
133:       if ( JLATT.ne.0 ) ILAT = IMNLT(NLBZ+1)
134: C
135: C
136:       do 210 M = 1, MMX
137: C
138: C
139: C
140: C .... RETURN HERE IF KMEMO(MZONE,M)=0 AND THERE ARE NO PARTICLES WHICH
141: C BELONGS TO ZONE 'KZ'. DETERMINE NEXT 'KZ' WITHOUT INCREMENTING
142: C LOOP COUNTER 'M'.
143: C
144: C
145: C
146: 120   if ( INX.eq.0 ) go to 220
147: C
148: C .... DETERMINE ZONE TO BE CHECKED (KZ) .....
149: C
150: C
151: C ..... CHECK A MEMORIZED ZONE .....
152: C
153:       if ( KMEMO(MZONE,M).ne.0 ) then
154:           KZ = KMEMO(MZONE,M)
155:       else
156: C
157: C ..... CHECKED ALL MEMORIZED ZONES, SO SELECT ANOTHER ZONE .....
158: C
159:           IMEMO = 0
160: C
161: C
162: C << SELECTION RULE OF THE ZONE CHECKED >>
163: C
164: C * OMIT MZONE.
165: C * NOT INCLUDED IN KMEMO.
166: C
167: 130   KKZ = KKZ + 1
168:       if ( KKZ.eq.MZONE ) go to 130
169: C
170:       do 140 MM = 1, MMX
171:           if ( KKZ.eq.KMEMO(MZONE,MM) ) go to 130
172:       continue
173: C
174:       KZ = KKZ
175:       if ( KZ.gt.MMZ ) go to 220
176: C
177: C
178: end if
179: C
180: C
181:       MAT = KZMAT(KZ)
182: C
183: C
184: C
185: C .... CHECK FOR EACH PARTICLE. IFI(I) = .FALSE./.TRUE. = OUT/IN
186: C
187: C
188: C
189:       call JUDGE( 'NEESEa', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
190:       &          NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, .false., DUM1,
191:       &          DUM2, DUM3 )
192: C
193: C
194: C
195: C .... COMPRESS PARTICLE DATA & UPDATE FLIGHT STACK .....

```

```

196: C
197: C
198: C
199:       KK = 0
200:       do 150 I = 1, INX
201:           if ( IFI(I) ) KK = KK + 1
202: 150   continue
203:       if ( KK.ne.0 ) then
204:           II = 0
205:           IFFL = 0
206:           IDEAD = NDIMPT
207: C
208: C
209: C
210: C ..... SEND PARTICLES TO FLIGHT STACK .....
211: C
212: C
213: C
214:       if ( MAT.ge.0 ) then
215: *VOCL LOOP,NOVREC
216:           do 160 I = 1, INX
217:               if ( IFI(I) ) then
218:                   IFFL = IFFL + 1
219:                   IMSFL(INFL+IFFL) = IBP(I)
220:               end if
221: 160   continue
222:           do 170 I = 1, IFFL
223:               IMZFL(INFL+I) = KZ
224: 170   continue
225:           IMNFL(KZ) = IMNFL(KZ) + IFFL
226:           INFL = INFL + IFFL
227: C
228: C
229: C
230: C ..... ESCAPING LATTICE OR CELL .....
231: C
232: C
233: C
234: CCC   else if ( MAT.le.-1.and.MAT.ge.-999 ) then
235:       else if ( ISLATT(MAT).or.MAT.eq.-999 ) then
236: *VOCL LOOP,NOVREC
237:           do 180 I = 1, INX
238:               if ( IFI(I) ) then
239:                   ILAT = ILAT + 1
240:                   IMSLT(ILAT) = IBP(I)
241:                   IMZLT(ILAT) = MLBZZ(KZ)
242:                   IZZI(IBP(I)) = KZ
243:               end if
244: 180   continue
245:           IMNLT(MLBZZ(KZ)) = IMNLT(MLBZZ(KZ)) + KK
246: C
247: C
248: C
249: C ..... UNPENETRATABLE MATERIAL (TERMINATE WITHOUT TALLY )
250: C
251: C
252: C
253:       else if ( MAT.lt.0 ) then
254: *VOCL LOOP,NOVREC
255:           do 190 I = 1, INX
256:               if ( IFI(I) ) then
257:                   IDEAD = IDEAD + 1
258:                   IMDED(IDEAD) = IBP(I)
259: CCCCCCCCCC   WCNTR(7) = WCNTR(7) + W(I)
260:           end if

```

src/shared/neesea.f

```

261: 190          continue
262:          if ( IDEAD.gt.NDIMPT ) then
263: CCCCCCCCCC      NCNTR(7) = NCNTR(7) + IDEAD - NDIMPT
264: CCCCCCCCCC      NCNTR(17) = NCNTR(17) + IDEAD - NDIMPT
265:          NDIMPT = IDEAD
266:          NIMPT = IMPMAX - NDIMPT
267:          end if
268:        end if
269: C
270: C
271: C
272: C ..... COMPRESS POINTERS & TEMPORALY ARRAYS .....
273: C
274: C
275: C
276: *VOCL LOOP,NOVREC
277:       do 200 I = 1, INX
278:         if ( .not.IFI(I) ) then
279:           II = II + 1
280:           IBP(II) = IBP(I)
281:           X(II) = X(I)
282:           Y(II) = Y(I)
283:           Z(II) = Z(I)
284:           W(II) = W(I)
285:         end if
286:       200      continue
287:       INX = II
288: C
289: C
290: C
291: C ..... SETTING OF KMEMO ARRAY IF NECESSARY .....
292: C
293: C       ( FOUND A NEW ZONE TO BE MEMORIZED )
294: C
295: C880901 **** IF(IMEMO.EQ.0) KMEMO(M,MZONE) = KZ
296: C       if ( .not.JMEM ) then
297: C         if ( IMEMO.eq.0 ) KMEMO(MZONE,M) = KZ
298: C
299: C
300: C       ..... WITH TALLY FOR OPTIMIZATION OF DATA ORDER IN KMEMO
301: C       (ACCUMULATE NUMBER OD ENTERING PARTCLES FOR EACH
302: C       MEMORIZED ZONE)
303: C
304: C
305: C       else
306: C         if ( IMEMO.eq.0 ) then
307: C           KMEMO(MZONE,M) = KZ
308: C           MEMC(MZONE,M) = MEMC(MZONE,M) + KK
309: C           MEMZ(MZONE) = M
310: C         else if ( M.le.NMEMO ) then
311: C           MEMC(MZONE,M) = MEMC(MZONE,M) + KK
312: C         end if
313: C       end if
314: C
315: C       .... NO PARTICLES ENTER THIS ZONE !! .... ( KK = 0 )
316: C
317: C       else if ( IMEMO.eq.0 ) then
318: C         go to 120
319: C       end if
320: C       210 continue
321: C
322: C
323: C =====
324: C ..... END OF NEXT-ZONE-SEARCH .....
325: C =====

```

```

326: C
327: C
328: C
329: C .... UPDATE ZONE # IN BANK
330: C
331: C
332: C
333: *VOCL LOOP,NOVREC
334:       220 do 230 I = IMNFL(NZONE+1) + 1, INFL
335:         IZZI(IMSFL(I)) = IMZFL(I)
336:       230 continue
337: CCCC NCNTR(17) = NCNTR(17) + INFL - IMNFL(NZONE+1)
338: C
339: C
340: C
341: C .... COMPRESS SEARCH STACK .....
342: C
343: C
344: C
345: C       II = 0
346: *VOCL LOOP,NOVREC
347:       do 240 I = 1, IMNNX(NZONE+1)
348:         if ( IMZNX(I).ne.MZONE ) then
349:           II = II + 1
350:           IMSNX(II) = IMSNX(I)
351:           IMZNX(II) = IMZNX(I)
352:         end if
353:       240 continue
354: C
355: C
356: C =====
357: C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, TREAT THEM
358: C AS DEFERRED PARTICLES OR LOST PARTICLES.
359: C =====
360: C
361: C
362: C       if ( INX.ne.0 ) then
363: C
364: C
365: C       .... DEFERRED PARTICLES .....
366: C
367: C
368: CCCC IF(IMEMO.EQ.1.OR. IMEMO.EQ.0.AND.MMX.LT.NZONE) THEN
369: C       if ( IMEMO.eq.1 .or. IMEMO.eq.0.and.MMX.lt.(MMZ-KKZ) ) then
370: C         do 250 I = 1, INX
371: C           IMSNX(II+I) = IBP(I)
372: C           IMZNX(II+I) = -MZONE
373: C       250      continue
374: C           IMDEFR = IMDEFR + INX
375: C           IMNNX(NZONE+1) = IMNNX(NZONE+1) + INX
376: C
377: C
378: C       .... LOST PARTICLES OR TANGENTIAL PARTICLES
379: C       ELSE IF(INX.NE.0.AND.IMEMO.EQ.0) THEN
380: C
381: C
382: C       else
383: C         if ( KZMAT(MZONE).ge.0 ) then
384: C       .... CHECK FOR PREVIOUS ZONE. IFI(I) = .FALSE./.TRUE. = OUT/IN
385: C         call JUDGE( 'NEESEA', MZONE, INX, X, Y, Z, IFI, SDA,
386: C           &          KZDA, KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL,
387: C           &          JSIMP, .false., DUM1, DUM2, DUM3 )
388: C
389: C           IDEAD = 0
390: C           IFFL = 0

```

src/shared/neesea.f

```

391: *VOCL LOOP,NOVREC
392:       do 260 I = 1, INX
393: C
394: C ..... TANGENTIAL .....
395: C
396:       if ( IFI(I) ) then
397:         IFFL = IFFL + 1
398:         IMSFL(INFL+IFFL) = IBP(I)
399:         IMZFL(INFL+IFFL) = MZONE
400:         XXXI(IBP(I)) = X(I)
401:         YYI(IBP(I)) = Y(I)
402:         ZZI(IBP(I)) = Z(I)
403:         KLSFI(IBP(I)) = 0
404: C
405: C ..... LOST .....
406: C
407:       else
408:         IDEAD = IDEAD + 1
409:         IMDED(NDIMPT+IDEAD) = IBP(I)
410:       end if
411: 260   continue
412:       IMNFL(MZONE) = IMNFL(MZONE) + IFFL
413:       INFL = INFL + IFFL
414: C
415:       else
416:         do 270 I = 1, INX
417:           IFI(I) = .false.
418:           IMDED(NDIMPT+I) = IBP(I)
419: 270   continue
420:           IDEAD = INX
421:         end if
422: C
423:         NDIMPT = NDIMPT + IDEAD
424:         NLOST = NLOST + IDEAD
425: C
426:         if ( IDEAD.gt.0 ) then
427:           write(IOW,*) ' !! (NEESEA) ', IDEAD,
428:           & ' PARTICLES ARE LOST !! ZONE # = ', MZONE
429:           do 280 I = 1, INX
430:             if ( .not.IFI(I) ) then
431:               write(IOW,9000) IBP(I), X(I), Y(I), Z(I),
432:               & AAAI(IBP(I)), BBBI(IBP(I)), CCCI(IBP(I))
433:               if ( JLATT.ne.0.and.LEVLI(IBP(I)).ne.0 ) write(IOW,
434:               & 9020)
435:               & (L,LZZI(IBP(I),L),LPOSI(IBP(I),L),L=1,
436:               & LEVLI(IBP(I)))
437:             end if
438: 280   continue
439:           end if
440: 9000   format(1X,I5,' X=',1P,E12.5,' Y=',E12.5,' Z=',E12.5,' MU=',
441:           & E12.5,' ETA=',E12.5,' XI=',E12.5)
442: 9020   format(1X,' LEVLI = ',I3,' LZZI = ',I5,' LPOSI = ',I5)
443:         end if
444:       end if
445: C
446: C .... ADJUSTMENT OF NUMBER OF PARTICLE IN STACKS .....
447: C
448:       IMNNX(NZONE+1) = IMNNX(NZONE+1) - IMNNX(MZONE)
449:       IMNNX(MZONE) = 0
450:       IMNFL(NZONE+1) = INFL
451:       if ( JLATT.ne.0 ) IMNLT(NLBZ+1) = ILAT
452:       NIMPT = IMPMAX - NDIMPT
453: C
454:       return
455: C

```

```

456: C
457: C
458: C =====
459: C .... TREATMENT OF DEFERRED PARTICLE .....
460: C =====
461: C
462: C
463: 290 II = 0
464:       do 300 I = 1, IMNNX(NZONE+1)
465:         if ( IMZNX(I).lt.0 ) then
466:           II = II + 1
467:           IBP(II) = IMSNX(I)
468:           IZI(II) = -IMZNX(I)
469:         end if
470: 300   continue
471: C
472:       if ( II.ne.IMDEFR ) then
473:         write(IOW,*) ' !! (NEESEA) NUMBER OF DEFERRED PARTICLES IN ',
474:         & ' SEARCH STACK IS DIFFERENT FROM IMDEFR !! ', II,
475:         & IMDEFR
476:       stop 666
477:       end if
478:       do 310 I = 1, IMDEFR
479:         X(I) = XXXI(IBP(I)) + DEPS*AAAI(IBP(I))
480:         Y(I) = YYI(IBP(I)) + DEPS*BBBI(IBP(I))
481:         Z(I) = ZZI(IBP(I)) + DEPS*CCCI(IBP(I))
482:         W(I) = WWI(IBP(I))
483: 310   continue
484: C
485: C .... CHECK FOR ALL ZONES .....
486: C
487:       INX = IMDEFR
488:       IDEAD = NDIMPT
489:       if ( JLATT.ne.0 ) ILAT = IMNLT(NLBZ+1)
490:       KKZ = 0
491:       MMZ = NZONE
492:       do 350 KZ = 1, NZONE
493:         if ( INX.eq.0 ) go to 360
494: C
495:         call JUDGE( 'NEESEA', KZ, INX, X, Y, Z, IFI, SDA, KZDA, KZAA,
496:         & NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, .false., DUM1,
497:         & DUM2, DUM3 )
498:
499:         II = 0
500:         if ( JLATT.ne.0 ) then
501:           ICEL = KCELL(KZ)
502:           MMZ = IPCEL(ICEL+1) - 1
503:           if ( ICEL.gt.0 ) then
504:             KKZ = IPCEL(ICEL) - 1
505:           else
506:             KKZ = 0
507:           end if
508:         end if
509: C
510:         if ( KZMAT(KZ).lt.0 ) then
511:           *VOCL LOOP,NOVREC
512:           do 320 I = 1, INX
513:             if ( (.not.IFI(I)) .or. IZI(I).eq.KKZ .or. IZI(I).le.KKZ
514:             & .or. IZI(I).gt.MMZ ) then
515:               II = II + 1
516:               IBP(II) = IBP(I)
517:               X(II) = X(I)
518:               Y(II) = Y(I)
519:               Z(II) = Z(I)
520:               W(II) = W(I)

```


src/shared/neesea.f

```

521:          IZI(II) = IZI(I)
522:          else
523: C
524: C
525: C      .... LATTICE-SEARCH .....
526: C
527: C
528: CCCC      if ( KZMAT(KZ).le.-1.and.KZMAT(KZ).ge.-999 ) then
529: C          if ( ISLATT(KZMAT(KZ)).or.KZMAT(KZ).eq.-999 ) then
530: C              ILAT      = ILAT + 1
531: C              IMSLT(ILAT) = IBP(I)
532: C              IMZLT(ILAT) = MLBZZ(KZ)
533: C              IMNLT(MLBZZ(KZ)) = IMNLT(MLBZZ(KZ)) + 1
534: C          else if ( KZMAT(KZ).lt.0 ) then
535: C              NDIMPT = NDIMPT + 1
536: C              IMDED(NDIMPT) = IBP(I)
537: C              CCCICC      WCNTR(7) = WCNTR(7) + W(I)
538: C          end if
539: C      end if
540: C      320      continue
541: C      else
542: C011691 ...
543: C          IFFL = 0
544: C      *VOCL LOOP,NOVREC
545: C      do 330 I = 1, INX
546: C          if ( (.not.IBP(I)) .or. IZI(I).le.KZ .or. IZI(I).gt.MMZ
547: C      &      ) then
548: C              II = II + 1
549: C              IBP(II) = IBP(I)
550: C              X(II) = X(I)
551: C              Y(II) = Y(I)
552: C              Z(II) = Z(I)
553: C              W(II) = W(I)
554: C              IZI(II) = IZI(I)
555: C          else
556: C
557: C
558: C      .... FREE-FLIGHT .....
559: C
560: C
561: C          IFFL = IFFL + 1
562: C          IMSFL(IMNFL(NZONE+1)+IFFL) = IBP(I)
563: C          IZZI(IBP(I)) = KZ
564: C
565: C
566: C      ..... TANGENTIAL PARTICLES
567: C
568: C
569: C          if ( IZI(I).eq.KZ ) then
570: C              XXXI(IBP(I)) = X(I)
571: C              YYI(IBP(I)) = Y(I)
572: C              ZZI(IBP(I)) = Z(I)
573: C          end if
574: C      end if
575: C      330      continue
576: C      if ( IFFL.gt.0 ) then
577: C          do 340 I = 1, IFFL
578: C              340      IMZFL(IMNFL(NZONE+1)+I) = KZ
579: C              IMNFL(NZONE+1) = IMNFL(NZONE+1) + IFFL
580: C              IMNFL(KZ) = IMNFL(KZ) + IFFL
581: C          end if
582: C      end if
583: C011691 ... END
584: C
585: C          INX = II

```

```

586: C011691      IF(IFFL.GT.0) THEN
587: C          DO 420 I=1,IFFL
588: C      420      IMZFL(IMNFL(NZONE+1)+I) = KZ
589: C          IMNFL(NZONE+1) = IMNFL(NZONE+1) + IFFL
590: C          IMNFL(KZ) = IMNFL(KZ) + IFFL
591: C          ENDIF
592: C
593: C      350 continue
594: C
595: C      CC440 NCNTR(7) = NCNTR(7) + NDIMPT - IDEAD
596: C      360 continue
597: C
598: C
599: C      ..... LOST PARTICLES .....
600: C
601: C
602: C          if ( INX.ne.0 ) then
603: C              do 370 I = 1, INX
604: C                  IMDED(NDIMPT+I) = IBP(I)
605: C      370      continue
606: C              NDIMPT = NDIMPT + INX
607: C              NLOST = NLOST + INX
608: C              write(IOW,*) ' !! (NEESEA) ', INX, ' PARTICLES ARE LOST IN ',
609: C      &      'DEFERRED SEARCH. '
610: C              do 380 I = 1, INX
611: C                  write(IOW,9000) X(I), Y(I), Z(I), AAAI(IBP(I)),
612: C      &      BBBI(IBP(I)), CCCI(IBP(I))
613: C                  if ( JLATT.ne.0.and.LEVLI(IBP(I)).ne.0 ) write(IOW,9020)
614: C      &      (L,LZZI(IBP(I),L),L,LOSI(IBP(I),L),L=1,LEVLI(IBP(I)))
615: C      380      continue
616: C          end if
617: C
618: C
619: C      .... COUNT THE NUMBER OF BOUNDARY CROSSING
620: C      CCCI NCNTR(17) = NCNTR(17) + IMDEFR - INX
621: C
622: C
623: C
624: C      .... COMPRESS SEARCH STACK .....
625: C
626: C
627: C          II = 0
628: C      *VOCL LOOP,NOVREC
629: C      do 390 I = 1, IMNNX(NZONE+1)
630: C          if ( IMZNX(I).ge.0 ) then
631: C              II = II + 1
632: C              IMSNX(II) = IMSNX(I)
633: C              IMZNX(II) = IMZNX(I)
634: C          end if
635: C      390 continue
636: C          IMNNX(NZONE+1) = IMNNX(NZONE+1) - IMDEFR
637: C          if ( JLATT.ne.0 ) IMNLT(NLBZ+1) = ILAT
638: C          NIMPT = IMPMAX - NDIMPT
639: C          IMDEFR = 0
640: C          return
641: C
642: C      end

```

src/shared/nestck.f

```

1:      subroutine NESTCK( NEST, NLATT, NCELL, NINPZ, KCELI,
2:      &                  IDCEL, IPCLI, IDLAT, IPLAT, KLATT, KSLAT, LTYPE,
3:      &                  KMAT )
4: C=====
5: C  PURPOSE: TO CHECK THE DEPTH OF NEST IN LATTICE-GEOMETRY
6: C            RETURN MAXIMUM NESTING LEVEL IN 'NEST'.
7: C  CALLED IN: GEOMIN
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C*i IOG : message output I/O unit (removed Jan 2000)
11: C o NEST : geometry nest depth
12: C
13: C i ... others (geometry data, see INC/_PGEOM)
14: C=====
15:      integer KCELI(NINPZ), IDCEL(NCELL), IPCLI(NCELL+1), IDLAT(NLATT),
16:      &      LTYPE(NLATT), IPLAT(NLATT+1), KLATT(*), KSLAT(*),
17:      &      KMAT(NINPZ)
18: C
19: C
20:      parameter( MAXLVL = 4 )
21: C
22: C      .... LOCAL WORKING ARRAYS TO SWEEP NESTING STRUCTURE
23: C
24:      integer IZP(0:MAXLVL), IZL(0:MAXLVL), IZC(0:MAXLVL),
25:      &      IZCP(0:MAXLVL)
26:      character TEMP6*6, TEMP62*6
27: C
28:      character*4 LTYPE
29: CCCC data LTYPE /'RECT', 'HEX2', 'HEX3', 'HEX4'/
30: C
31:      include 'INC/_IOUNIT'
32: C
33:      include 'INC/_PMLATT'
34:      include 'INC/_SFLATT'
35: C
36: C=====
37: C
38:      call LABEL( IOG, 'LATTICE GEOMETRY STRUCTURE' )
39: C
40:      NEST = 0
41: C
42: C      .... SEARCH ZONES AND PRINT LATTICE NESTING STRUCTURE. ....
43: C
44:      LV = 0
45:      IZP(0) = 0
46:      IZC(0) = 0
47:      IZL(0) = 0
48: C
49:      write(IOG,7000) (I,I=1,MAXLVL)
50:      write(IOG,7020) ('-----', '-----', I=1,MAXLVL)
51:      write(IOG,7020) (' LATTICE ', ' CELL ', I=1,MAXLVL)
52:      write(IOG,7020) (' (TYPE) ', '(DIRECTION) ', I=1,MAXLVL)
53:      write(IOG,7020) ('-----', '-----', I=1,MAXLVL)
54:      7000 format(//4X,5('          LEVEL ',I3,'          '))
55:      7020 format(4X,5(A12,A12:))
56:      7040 format(4X,5(A6,A6,12X:))
57:      7060 format(4X,5(12X,A6,A6:))
58: C
59: C
60:      100 IZP(LV) = IZP(LV) + 1
61: C
62: C -----
63: C      .... END OF A CELL
64: C -----
65: C

```

```

66:      if ( IZP(LV).gt.NINPZ .or. (LV.gt.0.and.KCELI(IZP(LV)).ne.IZC(LV))
67:      & ) then
68: C
69: C
70:      110 IZCP(LV) = IZCP(LV) + 1
71: C
72: C
73: C      .... END OF LATTICE CELL ...
74: C
75: C
76:      if ( LTYPE(IZL(LV)).eq.10.and.
77:      &      IZCP(LV).gt.IPLAT(IZL(LV))+KLATT(IPLAT(IZL(LV))+9).or.
78:      &      LTYPE(IZL(LV)).ne.10.and.IZCP(LV).ge.IPLAT(IZL(LV)+1) )
79:      &      then
80:          LV = LV - 1
81:          go to 100
82: C
83: C      .... ENTER NEXT CELL .....
84: C
85:      else
86:          if ( KLATT(IZCP(LV)).eq.0 ) go to 110
87: C
88: C      ... ALREADY CHECKED CELL OR NOT ? ...
89: C
90:          IFFF = 0
91:          do 120 K = IPLAT(IZL(LV)), IZCP(LV) - 1
92:              if ( KLATT(K).eq.KLATT(IZCP(LV)) ) then
93:                  IFFF = 1
94: C
95:                  if ( KSLAT(K).eq.KSLAT(IZCP(LV)) ) IFFF = 2
96:                  end if
97:          120 continue
98: C
99: C      .... ALREADY IN LATTICE IN THE SAME DIRECTION ...
100: C
101:      if ( IFFF.eq.2 ) go to 110
102: C
103:      write(TEMP6,'(I6)') IDCEL(KLATT(IZCP(LV)))
104:      write(TEMP62,'(''('',i4,'')'')' KSLAT(IZCP(LV)))
105: C
106:      write(IOG,7060) (' ', ' ', I=1, LV-1), TEMP6, TEMP62
107: C
108: C      .... ALREADY IN LATTICE BUT DIFFERENT DIRECTION ...
109: C
110:      if ( IFFF.eq.1 ) go to 110
111: C
112: C      ... NEXT CELL IN CURRENT LATTICE ...
113: C
114:      IZC(LV) = KLATT(IZCP(LV))
115:      IZP(LV) = IPCLI(IZC(LV))
116:      go to 100
117:      end if
118:      end if
119: C
120: C
121: C
122:      IMAT = KMAT(IZP(LV))
123:      if ( LV.eq.0.and.IMAT.eq.-999 ) go to 140
124: C
125: C
126: C -----
127: C      .... ENTERING A LATTICE (OR LATTICE-FRAME) ....
128: C -----
129: C
130: C

```

src/shared/nestck.f

```

131:      if ( ISLATT(IMAT) ) then
132:        LV      = LV + 1
133:        NEST     = MAX(NEST,LV)
134: C
135:      if ( LV.gt.MAXLVL ) then
136:        write(IMG,7080) LV, MAXLVL
137: 7080      format(/1X,' XXX TOO DEEPLY NESTED GEOMETRY !! ',
138:      &          ' ( NEST =',I3,' ALLOWED ',I3,' ). ' /
139:      &          ' YOU MAY HAVE MADE A RECURSIVE ',
140:      &          'GEOMETRY SPECIFICATION. '//
141:      &          ' IF YOU WANT TO CONSTRUCT A GEOMETRY ',
142:      &          'WHICH EXCEEDS CURRENT LIMIT OF ',
143:      &          'NEST DEPTH, INCREASE ''MAXLVL'' PARAMETER'/
144:      &          ' IN SUBROUTINE ''NESTCHK''.')
145:        call PRSTOP( 1, 'GEOMETRY NEST IS TOO DEEP OR RECURSIVE.' )
146:        stop 888
147:      end if
148: C
149: C      .... LATTICE # ( ASSUME 'KMAT' IS LATTICE #, NOT ID )
150: C
151:      IZL(LV) = LATTNM(IMAT)
152: C
153: C      .... CELL ARRAY POINTER.
154:      IZCP(LV) = IPLAT(IZL(LV))
155: C
156: C      .... CELL #
157: 130      IZC(LV) = KLATT(IZCP(LV))
158:      if ( IZC(LV).eq.0 ) then
159:        IZCP(LV) = IZCP(LV) + 1
160:        if ( LTYPE(IZL(LV)).eq.10.and.
161:      &      IZCP(LV).gt.IPLAT(IZL(LV))+KLATT(IPLAT(IZL(LV))+9).br.
162:      &      LTYPE(IZL(LV)).ne.10.and.IZCP(LV).ge.IPLAT(IZL(LV)+1) )
163:      &      then
164:        LV      = LV - 1
165:        write(IMG,*) '!!! NO EFFECTIVE CELLS IN ',
166:      &          'THIS LATTICE !! '
167:        call CNTERR( 'WARNING' )
168:        go to 100
169:      end if
170:      go to 130
171:    end if
172: C
173: C      .... CELL STARING ZONE ( KMAT = -999 )
174: C
175:      IZP(LV) = IPCLI(IZC(LV))
176: C
177:      write(TEMP6,'(I6)') IDLAT(IZL(LV))
178:      call LATTYP(LTYPE,'<',LTYPE(IZL(LV)))
179:      TEMP62 = '('/LTYPE//')'
180:      write(IOG,7040) ( '      ',I=1,LV-1), TEMP6, TEMP62
181: C
182:      write(TEMP6,'(I6)') IDCEL(IZC(LV))
183:      write(TEMP62,'(''('',I4,'')'')') KSLAT(IZCP(LV))
184: C
185:      write(IOG,7060) ( '      ',I=1,LV-1), TEMP6, TEMP62
186: C
187:      go to 100
188:    end if
189:    go to 100
190: 140 continue
191: C
192:      write(IOG,7020) ('-----','----- ',I=1,MAXLVL)
193:      write(IOG,7100) NEST
194: 7100 format(/' * MAXIMUM NESTING LEVEL : ',I3/)
195: C
196:      if ( NEST.eq.0 ) then
197:        write(IMG,*) '!!! NO GEOMETRY NEST WITH LATTICE GEOMETRY'
198:        call CNTERR( 'WARNING' )
199:      end if
200: C
201:      return
202:    end

```

src/shared/ngpchk.f

```
1:      subroutine NGPCHK( CODE, NGP, JKPAR, KPSYM, KPLIM,  
2:      &                NGP1, NGP2, NGROUP )  
3: C=====
```

4: C Purpose: check (or set if necessary) number of energy groups
5: C (or energy bins) for newly implemented multiple particle
6: C treatment scheme.

7: C
8: C This should be called before procedure that may require
9: C definition of energy bin numbers (of all particle species).

10: C
11: C called in: INTRO, INTRO2

12: C-----

13: C i CODE : program name (MVP/GMVP)
14: C io NGP(KPLIM) : number of energy groups for each particle species.
15: C i JKPAR(KPLIM) : flag for each particle species.
16: C i KPSYM(2,KPLIM) : particle symbol for each particle species.
17: C i KPLIM : number of possible particle species.

18: C
19: C (JNEUT, IPHOT, NGP1 and NGP2 are for backward compatibility only.
20: C They should be removed in the future.)

21: C
22: C io NGP1 : number of neutron energy bins
23: C io NGP2 : number of photon energy bins

24: C
25: C io NGROUP : total number of energy bins over all particles

26: C-----

27: include 'INC/_IOUNIT'

28: C
29: character*4 CODE
30: integer NGP(KPLIM)
31: integer JKPAR(KPLIM)
32: character*32 KPSYM(2,KPLIM)

33: C-----

34: C
35: C ... check bin(group) numbers of each particles

36: C
37: C
38: JSET = 0
39: NN = 0
40: do 100 IK=1,KPLIM
41: if (JKPAR(IK).ne.0 .and. NGP(IK).le.0) then
42: NGP(IK) = 1
43: write(IMG,7000) KPSYM(1,IK):(ICLEN2(KPSYM(1,IK))),NGP(IK)
44: call CNTERR('WARNING')
45: JSET = 1
46: C
47: C ... for backward compatibility

48: C
49: if (KPSYM(1,IK).eq.'NEUTRON') NGP1 = 1
50: if (KPSYM(1,IK).eq.'PHOTON') NGP2 = 1
51: end if
52: NN = NN + NGP(IK)
53: 100 continue

54: C
55: 7000 format(/lx,'!!!(NGPCHK) Particle <',a,'>: ',
56: & 'Unspecified number of enrgy bin(group). Set to ',i4,'.')
57: C
58: C ... check total number of enrgy bins (groups) NGROUP

59: C
60: if (NGROUP.eq.0) then
61: NGROUP = NN
62: write(IMG,7020) NN
63: call CNTERR('WARNING')
64: else if (NGROUP.ne.NN) then
65: if (JSET.eq.0) then

```
66:            write(IMG,7040) NGROUP, NN  
67:            call CNTERR('FATAL')  
68:         else  
69:            write(IMG,7060) NGROUP, NN  
70:            NGROUP = NN  
71:            call CNTERR('WARNING')  
72:         end if  
73:      end if  
74: C  
75:      7020 format(/lx,'!!!(NGPCHK) Total number of energy bin(group) ',  
76:      & '"NGROUP" is set to ',i4)  
77: C  
78:      7040 format(/lx,'XXX(NGPCHK) Total number of energy bin(group) ',  
79:      & '"NGROUP" (= ',i4,' ) is different from sum of number of ',  
80:      & ' energy bin(group) of particles (= ',i4,' ).')  
81: C  
82:      7060 format(/lx,'!!!(NGPCHK) Total number of energy bin(group) ',  
83:      & '"NGROUP" (= ',i4,' ) is different from sum of number of ',  
84:      & ' energy bin(group) of particles (= ',i4,' )'/lx,  
85:      & ' NGROUP is reset to the summation value.')  
86: C  
87:      return  
88:      end
```

src/shared/nndcpy.f

```

1:      subroutine NNDCPY( N,      INND,  NNND,  IDLAT, LTYP,
2:      &                  NLATT, KLATT, KSLAT, IPLAT )
3: C=====
4: C Purpose: Copy NND pointers etc. for STG region(lattice) whose NND
5: C           is specified as copy of that of some other STG region.
6: C Called in: LATTIN
7: C-----
8: C arguments (i=input, o=output)
9: C
10: C i   IOG: message printout I/O unit
11: C i   N: lattice # of processed lattice.
12: C i   INND: NND type # (1,2 or 3)
13: C i   NNND: negative lattice ID of lattice whose NND is copied.
14: C i   IDLAT(NLATT) : lattice ID
15: C i   LTYP(NLATT) : lattice type
16: C i   NLATT : number of lattices
17: C io  KLATT(*) : lattice parameter 1
18: C io  KSLAT(*) : lattice parameter 2
19: C i   IPLAT(NLATT+1) : pointer to KLATT & KSLAT
20: C-----
21:      integer IDLAT(NLATT)
22:      integer LTYP(NLATT)
23:      integer KLATT(*), KSLAT(*)
24:      integer IPLAT(NLATT+1)
25: C
26:      include 'INC/_IOUNIT'
27: C
28: C-----
29: C
30:      IDLT      = -NNND
31:      do 100 IL = 1, NLATT
32:        if ( IDLAT(IL).eq.IDLT ) then
33:          go to 110
34:        end if
35:      100 continue
36: C
37:      write(IMG,7000) INND, IDLAT(N), IDLT
38: 7000 format(1X,'XXX NND-',I1,' of STG lattice',I7,' is specified as ',
39: &          'copy of that for lattice',I7,', but no such a lattice.')
40:      call CNTERR( 'FATAL' )
41:      return
42: C
43: 110 continue
44:      if ( LTYP(IL).ne.10 ) then
45:        write(IMG,7020) INND, IDLAT(N), IDLT, IDLT
46: 7020 format(1X,'XXX NND-',I1,' of STG lattice',I7,' is specified',
47: &          ' as copy of that for lattice ',I7,'./1X,
48: &          ' But the lattice',I7,' is not a STG lattice.')
49:        call CNTERR( 'FATAL' )
50:        return
51:      end if
52: C
53:      NNDX      = 0
54:      LNNDX      = 0
55:      if ( INND.eq.1 ) then
56:        NNDX      = KLATT(IPLAT(IL)+10)
57:        LNNDX      = KSLAT(IPLAT(IL)+10)
58:      else if ( INND.eq.2 ) then
59:        NNDX      = KLATT(IPLAT(IL)+11)
60:        LNNDX      = KSLAT(IPLAT(IL)+11)
61:      else if ( INND.eq.3 ) then
62:        NNDX      = KLATT(IPLAT(IL)+12)
63:        LNNDX      = KSLAT(IPLAT(IL)+12)
64:      end if
65:      if ( NNDX.lt.0 ) then

```

```

66:        write(IMG,7040) INND, IDLAT(N), IDLT, IDLT, -NNDX
67: 7040 format(1X,'XXX NND-',I1,' of STG lattice',I7,' is specified',
68: &          ' as copy of that for lattice ',I7,'./1X,
69: &          ' But NND of lattice',I7,' is also specified',
70: &          ' as copy of that for lattice ',I7,'.')
71:        call CNTERR( 'FATAL' )
72:        return
73:      end if
74: C
75:      if ( INND.eq.1 ) then
76:        KLATT(IPLAT(N)+10) = NNDX
77:        KSLAT(IPLAT(N)+10) = LNNDX
78:      else if ( INND.eq.2 ) then
79:        KLATT(IPLAT(N)+11) = NNDX
80:        KSLAT(IPLAT(N)+11) = LNNDX
81:      else if ( INND.eq.3 ) then
82:        KLATT(IPLAT(N)+12) = NNDX
83:        KSLAT(IPLAT(N)+12) = LNNDX
84:      end if
85: C
86:      return
87:      end

```

src/shared/oldsrc.f

```

1:      subroutine OLDSRC( TYPE, NS,   I1,   I2,   DATA, NDATA, RWK,
2:      &                  IRAND, X,    Y,    Z,    A,    B,    C,
3:      &                  E,          IERR )
4:
5:      C/#IF SOURCE(NEW)
6:      c
7:      c==<MVP/GMVP>=====
8:      c purpose: generation of source particle from old-fashioned type source.
9:      c           (for upward compatibility with older version)
10:     c=====
11:     c
12:     c TYPE      sampling      control data given
13:     c
14:     c 1 point,isotropic      (x0,y0,z0)
15:     c 2 point,mono-direction (x0,y0,z0),(mu,eta,xi)
16:     c 3 rectangular,isotropic xmin,xmax,ymin,ymax,zmin,zmax
17:     c 4 sphere,isotropic      (x0,y0,z0),r
18:     c 5 disk,monodirectional  (x0,y0,z0),r,(mu,eta,xi)
19:     c 6 rect.,monodirectional (x0,y0,z0),(lx,ly,lz),l,(mu,eta,xi)
20:     c 9 FNS (D,T) source      x0,y0,z0,xr,xdir,thl,fal,pul
21:     c
22:     c =====
23:     implicit real*8(D)
24:     c
25:     character*(*) TYPE
26:     c
27:     ... user supplied data as input ....
28:     c
29:     real*8 DATA(NDATA)
30:     c
31:     ... working area ( at least 5 * ns )
32:     c
33:     real RWK(*)
34:     c
35:     ... parameters to be sampled ....
36:     c
37:     real*8 X(NS), Y(NS), Z(NS), A(NS), B(NS), C(NS)
38:     real*8 E(NS)
39:     c
40:     c
41:     type : source type ('TYPE' )
42:     ns : size of sampled variable array
43:     i1, i2 : sample x(i1) to x(i2) etc.
44:     c
45:     data : samplig control data
46:     ndata : number of samplig control data
47:     c
48:     rwk : working area (for random number storage)
49:     c
50:     x,y,z : source position (unit: cm)
51:     a,b,c : direction cosines (a**2 + b**2 + c**2 = 1.0)
52:     e : energy (unit: eV)
53:     c
54:     set and return an error code as follows:
55:     c
56:     ierr : .eq. 0 if sampling succeeded.
57:     .ne. 0 if any sampling errors occurred.
58:     c
59:     real*8 PAI, DPAI, HPAI
60:     parameter( PAI = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
61:     &          0.5D0*PAI )
62:     c
63:     IERR = 0
64:     c
65:     c-----

```

```

66:     c ... sampling of position ...
67:     c-----
68:     c
69:     NP = I2 - I1 + 1
70:     c
71:     if ( TYPE.eq.'TYPE1' .or. TYPE.eq.'TYPE2' ) then
72:       do 100 I = I1, I2
73:         X(I) = DATA(1)
74:         Y(I) = DATA(2)
75:         Z(I) = DATA(3)
76:       100 continue
77:     else if ( TYPE.eq.'TYPE3' ) then
78:       call RANU2( IRAND, RWK(I1), 3*NP, ICOD )
79:       do 110 I = I1, I2
80:         X(I) = DATA(1) + RWK(I)*(DATA(2)-DATA(1))
81:         Y(I) = DATA(3) + RWK(NP+I)*(DATA(4)-DATA(3))
82:         Z(I) = DATA(5) + RWK(2*NP+I)*(DATA(6)-DATA(5))
83:       110 continue
84:     else if ( TYPE.eq.'TYPE4' ) then
85:       call RANU2( IRAND, RWK(I1), 3*NP, ICOD )
86:       do 120 I = I1, I2
87:         DRR = DATA(4)*RWK(I)**(1.0D0/3.0D0)
88:         DMU = 2.0D0*RWK(NP+I) - 1.0D0
89:         X(I) = DATA(1) + DRR*DMU
90:         DSN = SQRT(1.0D0-DMU*DMU)*DRR
91:         DFI = DPAI*RWK(2*NP+I)
92:         Y(I) = DATA(2) + DSN*COS(DFI)
93:         Z(I) = DATA(3) + DSN*SIN(DFI)
94:       120 continue
95:     else if ( TYPE.eq.'TYPE5' ) then
96:       if ( DATA(5).eq.0. ) then
97:         DAX = 1.0
98:         DAY = 0.0
99:         DAZ = 0.0
100:       else if ( DATA(6).eq.0. ) then
101:         DAX = 0.0
102:         DAY = 1.0
103:         DAZ = 0.0
104:       else if ( DATA(7).eq.0. ) then
105:         DAX = 0.0
106:         DAY = 0.0
107:         DAZ = 1.0
108:       else
109:         DD1 = SQRT(DATA(6)**2+DATA(7)**2)
110:         DAX = 0.0
111:         DAY = DATA(7) /DD1
112:         DAZ = -DATA(6) /DD1
113:       end if
114:       DBX = DATA(6)*DAZ - DATA(7)*DAY
115:       DBY = DATA(7)*DAX - DATA(5)*DAZ
116:       DBZ = DATA(5)*DAY - DATA(6)*DAX
117:       DD2 = SQRT(DBX**2+DBY**2+DBZ**2)
118:       DBX = DBX/DD2
119:       DBY = DBY/DD2
120:       DBZ = DBZ/DD2
121:     c
122:     call RANU2( IRAND, RWK(I1), 2*NP, ICOD )
123:     do 130 I = I1, I2
124:       DR = DATA(4)*SQRT(RWK(I))
125:       DD = DPAI*RWK(NP+I)
126:       D1 = COS(DD)*DR
127:       D2 = SIN(DD)*DR
128:       X(I) = DAX*D1 + DBX*D2 + DATA(1)
129:       Y(I) = DAY*D1 + DBY*D2 + DATA(2)
130:       Z(I) = DAZ*D1 + DBZ*D2 + DATA(3)

```

src/shared/oldsrc.f

```

131: 130 continue
132: else if ( TYPE.eq.'TYPE6' ) then
133:     DBX = DATA(5)*DATA(10) - DATA(6)*DATA(9)
134:     DBY = DATA(6)*DATA(8) - DATA(4)*DATA(10)
135:     DBZ = DATA(4)*DATA(9) - DATA(5)*DATA(8)
136:     DD2 = SQRT(DBX**2+DBY**2+DBZ**2)
137:     DBX = DBX/DD2
138:     DBY = DBY/DD2
139:     DBZ = DBZ/DD2
140: C
141:     call RANU2( IRAND, RWK(I1), 2*NP, ICOD )
142:     do 140 I = I1, I2
143:         D1 = RWK(I)
144:         D2 = DATA(7)*RWK(I+NP)
145:         X(I) = DATA(4)*D1 + DBX*D2 + DATA(1)
146:         Y(I) = DATA(5)*D1 + DBY*D2 + DATA(2)
147:         Z(I) = DATA(6)*D1 + DBZ*D2 + DATA(3)
148: 140 continue
149:     end if
150: C
151: C-----
152: C ... sampling of direction ...
153: C-----
154: C
155: C .... isotropic ...
156: C
157:     if ( TYPE.eq.'TYPE1' .or. TYPE.eq.'TYPE3' .or. TYPE.eq.'TYPE4' )
158:     & then
159:         call RANU2( IRAND, RWK(I1), 2*NP, ICOD )
160: C
161:         do 150 I = I1, I2
162:             C(I) = (DATA(10)-DATA(9))*RWK(I) + DATA(9)
163:             DSN = SQRT(1.0D0-C(I)*C(I))
164:             DFI = DPAT*RWK(NP+I)
165:             B(I) = DSN*COS(DFI)
166:             A(I) = DSN*SIN(DFI)
167: 150 continue
168: C
169: C .... monodirection ...
170: C
171:     else if ( TYPE.eq.'TYPE2' ) then
172:         do 160 I = I1, I2
173:             A(I) = DATA(4)
174:             B(I) = DATA(5)
175:             C(I) = DATA(6)
176: 160 continue
177:     else if ( TYPE.eq.'TYPE5' ) then
178:         do 170 I = I1, I2
179:             A(I) = DATA(5)
180:             B(I) = DATA(6)
181:             C(I) = DATA(7)
182: 170 continue
183:     else if ( TYPE.eq.'TYPE6' ) then
184:         do 180 I = I1, I2
185:             A(I) = DATA(8)
186:             B(I) = DATA(9)
187:             C(I) = DATA(10)
188: 180 continue
189:     end if
190: C
191: C-----
192: C ... FNS (D,T) source ...
193: C-----
194: C
195:     if ( TYPE.eq.'TYPE9' ) then
196:         call FNSSC2( IRAND, NS, I1, I2, DATA, X, Y, Z, A, B, C, E,
197:             & RWK(1), RWK(4*NS+1) )
198:         end if
199: C ... endif for SOURCE(NEW)
200: C/#ENDIF
201:     return
202:     end

```

src/shared/opensc.f

```
1:      subroutine OPENS( PROG )
2: C
3: C=====
4: C PURPOSE: open scratch file if they are not opened
5: C       I/O units are defined in INC/_IUNIT
6: C       This is mainly not to leave files fort.*,ftn* ... in slave task
7: C       of parallel processing mode.
8: C CALLED IN: INTRO
9: C CALLS: OPENF
10: C-----
11: C arguments (i=input, o=output, w=work)
12: C
13: C i PROG : calling program name (MVP/GMVP etc.). currently unused...
14: C=====
15: C
16: C character PROG*(*)
17: C include 'INC/_IUNIT'
18: C
19: C logical OPD
20: C
21: C if ( INP.ne.IOIN ) then
22: C   inquire(IOIN,opened =OPD)
23: C   if ( .not.OPD ) then
24: C     call OPENF( IOIN, ' ', 'FORMATTED', 'SCRATCH', 'UNKNOWN',
25: C &      IERR )
26: C     if ( IERR.ne.0 ) then
27: C       write(IMG,7000) IOIN, 'FORMATTED', IERR
28: C       call CNTERR( 'FATAL' )
29: C     end if
30: C   end if
31: C end if
32: C
33: C inquire(IUG1,opened =OPD)
34: C if ( .not.OPD ) then
35: C   call OPENF( IUG1, ' ', 'FORMATTED', 'SCRATCH', 'UNKNOWN', IERR
36: C &      )
37: C   if ( IERR.ne.0 ) then
38: C     write(IMG,7000) IUG1, 'FORMATTED', IERR
39: C     call CNTERR( 'FATAL' )
40: C   end if
41: C end if
42: C
43: C inquire(IUG2,opened =OPD)
44: C if ( .not.OPD ) then
45: C   call OPENF( IUG2, ' ', 'FORMATTED', 'SCRATCH', 'UNKNOWN', IERR
46: C &      )
47: C   if ( IERR.ne.0 ) then
48: C     write(IMG,7000) IUG2, 'FORMATTED', IERR
49: C     call CNTERR( 'FATAL' )
50: C   end if
51: C end if
52: C
53: C inquire(IUB,opened =OPD)
54: C if ( .not.OPD ) then
55: C   call OPENF( IUB, ' ', 'UNFORMATTED', 'SCRATCH', 'UNKNOWN', IERR
56: C &      )
57: C   if ( IERR.ne.0 ) then
58: C     write(IMG,7000) IUB, 'UNFORMATTED', IERR
59: C     call CNTERR( 'FATAL' )
60: C   end if
61: C end if
62: C
63: C return
64: C
65: 7000 format(1X,'xxx(OPENS) failed to open scratch I/O unit',I3,' in ',
66:      &      ' FORM=',A,' error code =',I7)
67: C
68:      end
```


src/shared/pablo.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine PABLO( A, LIMIT, LWORK,
3:     &               NPICT, NZONE, NSDA, NZDA, NCELL, JLATT,
4:     &               JTLLT, JVMNT, JSIMP, JDEBG, IDCEL, IPCEL, KCELL,
5:     &               SDA, KINPZ, KMAT, KZREG, KZDA, KZAA, PAPER
6:     &               )
7: C=====
8: C  PURPOSE:  DRAWING PICTURES.
9: C  CALLED IN:  INTRO
10: C  CALLS:    DOTMAX, PICASO
11: C=====
12: CCC  include 'INC/_ARRAY'
13: real A(*)
14: CC
15:   integer JDEBG(*)
16:   integer KINPZ(*), KMAT(*), KZREG(*), KZDA(2,*), KZAA(*), KCELL(*)
17:   integer IDCEL(NCELL), IPCEL(NCELL+1)
18:   real*8 SDA(*)
19:   real PAPER(12,NPICT)
20: C
21:   include 'INC/_IUNIT'
22: C
23: C ..... FIND MAXIMUM DOT NUMBER IN PICTURES .....
24:   call DOTMAX( NDMAX, NPICT, PAPER )
25: C
26:   MAXMZ   = 300
27: C
28:   call KEEP( 'X', LX, NDMAX, 'R8', LAST, JDEBG )
29:   call KEEP( 'Y', LY, NDMAX, 'R8', LAST, JDEBG )
30:   call KEEP( 'Z', LZ, NDMAX, 'R8', LAST, JDEBG )
31:   call KEEP( 'IFI', LIFI, NDMAX, 'I4', LAST, JDEBG )
32:   call KEEP( 'IFL', LIFL, NDMAX, 'I4', LAST, JDEBG )
33:   call KEEP( 'KZM', LKZM, NDMAX, 'I4', LAST, JDEBG )
34:   call KEEP( 'LIRG', LIRG, NDMAX, 'I4', LAST, JDEBG )
35:   call KEEP( 'LIMT', LIMT, NDMAX, 'I4', LAST, JDEBG )
36:   call KEEP( 'LIRMWK', LIRMWK, NDMAX, 'I4', LAST, JDEBG )
37:   call KEEP( 'LMULTZ', LMULTZ, MAXMZ*5, 'I4', LAST, JDEBG )
38: C
39:   if ( LSIZ(LAST).gt.LIMIT ) then
40:     write(IMG,
41: &       '(/''!!! INSUFFICIENT MEMORY TO DRAW PICTURES !'')')
42:     write(IMG,'(/'' NECESSARY AT LEAST '' ,I12,
43: &       '' WORDS (LIMIT='' ,I12,')'')' ) LSIZ(LAST), LIMIT
44:     call CNTERR('WARNING')
45:     return
46:   end if
47: C
48:   NDOT     = 120
49: C
50:   call PICASO( NPICT, NDMAX, NDOT, MAXMZ, JSIMP, JLATT, JTLLT,
51: &             JVMNT, NZONE, NCELL, SDA, IDCEL, IPCEL, KCELL, KINPZ,
52: &             KMAT, KZREG, KZDA, NSDA, NZDA, KZAA, PAPER, A(LX), A(LY),
53: &             A(LZ), A(LKZM), A(LIRG), A(LIMT), A(LIRMWK), A(LMULTZ),
54: &             A(LIFI), A(LIFL) )
55: C
56:   return
57: end
```

src/shared/packet.f

```
1: C      parameter( nn = 10000 )
2: C      real      a(nn),b(nn)
3: C      integer   ia(nn),ib(nn)
4: C      real*8    d(nn),bd(nn)
5: C      equivalence (a(1),d(1),ia(1))
6: C      equivalence (b(1),bd(1),ib(1))
7: C
8: C      include 'INC/_WORDL'
9: C
10: C      character*128 ch
11: C
12: C      character*4 type
13: C      integer   iret(100)
14: C
15: C      mword = 2
16: C      mword = 1
17: C
18: C      call pack00( a, 'PACKET CONTAINER TEST', nn, iret(0))
19: C
20: C      n = 100
21: C      do 100 i=1,n
22: C          b(i) = i
23: C          ib(i+n) = i
24: C          bd(i+2*n) = i
25: C 100 continue
26: C
27: C      ch = '[PACKET OF CHARACTER TYPE] '//char(0)
28: C      lch = index(ch,char(0))-1
29: C      call packnd(a,'A', 'R4', b(1), n, iret(1) )
30: C      call packnd(a,'IA', 'I4', ib(n+1), n, iret(2) )
31: C      call packlb(a,'BEFORE D', 'BEFORE D', iret(3) )
32: C      call packnd(a,'D', 'R8', bd(2*n+1), n, iret(4) )
33: C      call packcs(a,'CH', ch(:lch), iret(5) )
34: C
35: C      write(6,*) 'IRETS ',(iret(i),i=1,5)
36: C
37: C      call pctcls(a,'CLOSE', nn2, nsize, ir )
38: C      write(6,*) ' close nn2 nsize ir ',nn2,nsize,ir
39: C
40: C      do 200 i=1,nn
41: C          b(i) = 0
42: C 200 continue
43: C
44: C      ... non existent label ...
45: C      call pctlb(a,'LABEL', 'INVALID',ir )
46: C      write(6,*) ' ir ',ir
47: C      call pctlb(a,'BEFORE D', 'BEFORE D', ir )
48: C      write(6,*) ' ir ',ir
49: C
50: C      call pctchk(a,'D?', type, lp, ndata, ir )
51: C      write(6,*) ir,' type ',type,' lp ', lp,' ndata ',ndata
52: C
53: C      call unpknd(a,'D', 'R8', bd, n, ir )
54: C      write(6,*) ir,' bd ',(bd(i),i=1,n)
55: C
56: C      call unpkpt(a,'CH?', type, lp, ndata, ir )
57: C      write(6,*) ir,' type ',type,' ndata ',ndata
58: C
59: C      call pctrw(a,'REWIND',ir)
60: C      write(6,*) 'rewind ',ir
61: C
62: C      call unpkpt(a,'a?', type, lp, ndata, ir )
63: C      write(6,*) ir,' type ',type,' lp ', lp,' ndata ',ndata
64: C      write(6,*) ir,' a ',(a(i),i=lp,lp+ndata-1)
65: C
```

```
66: C      call unpknd(a, 'ia?', 'I4', ib(1),n ,ndata, ir )
67: C      write(6,*) ir,' ndata ',ndata
68: C      write(6,*) ' ib ',(ib(i),i=1,ndata)
69: C
70: C      call unpknd(a, 'bd?', 'R8', bd(1),n ,ndata, ir )
71: C      write(6,*) ir,' ndata ',ndata
72: C      write(6,*) ' bd ',(bd(i),i=1,ndata)
73: C
74: C      call unpkcs(a, 'ch?',ch,ndata, ir )
75: C      write(6,*) ir,' ndata ',ndata
76: C      write(6,*) ' ch ',ch(:ndata)
77: C
78: C      end
79: C=<MVP/GMVP>=====
80: C purpose : data packing and unpacking into/from a data packet-container
81: C      A data-packet-container is a contiguous memory area on which
82: C      various types of data are stored as "packets" in arbitrary order
83: C      and length.
84: C calls: mvctoi, mvitoc, cmpchi
85: C=====
86: C
87: C      * data packet must begin on a double word boundary in
88: C      32 bit / word architecture.
89: C
90: C      * structure of a "packet"
91: C
92: C*****
93: C*** Obsolete ***** after June 1997 *****
94: C*****
95: C***      Packet header in 2 integer (greater than 30 bit precision !!):
96: C***
97: C***      1. <data-type>*2**24 + <packet-size>
98: C***
99: C***          <data-type>  1  integer    2  real
100: C***                      3  real*8    4  character
101: C***                      5  label
102: C***          <packet size> number of data elements (type 1,2, 3)
103: C***                      or number of charaters (type 4 or 5)
104: C***
105: C***      This data is negative for newly packed packet next time.
106: C***      ( it means current end point of container also )
107: C***      -1 : able to add new packet
108: C***      -2 : cannot add new packet (closed container !)
109: C***
110: C***
111: C***      2. pointer (array index to 'packet' array) of the
112: C***      header of the next packet.
113: C*****
114: C
115: C      Packet header in NPHEAD = 4 integer:
116: C
117: C      1. data-type
118: C
119: C          <data-type>  1  integer    2  real
120: C                      3  real*8    4  character
121: C                      5  label
122: C
123: C      This data is negative for newly packed packet next time.
124: C      ( it means current end point of container also )
125: C      -1 : able to add new packet
126: C      -2 : cannot add new packet (closed container !)
127: C
128: C      2. packet data size
129: C
130: C          number of data elements (type 1,2, 3)
```

src/shared/packet.f

```

131: C          or number of charaters (type 4 or 5)
132: C
133: C          3. pointer (array index to 'packet' array) of the
134: C          header of the next packet.
135: C          (-1 for the last packet)
136: C
137: C          4. pointer (array index to 'packet' array) of the
138: C          header of the previous packet.
139: C          (-1 for the first packet)
140: C
141: C          Data body immediately follows this header. Real*8 data should
142: C          begin on a double word memory boundary.
143: C
144: C
145: C          Routine (ENTRY) s
146: C
147: C          < packing >
148: C
149: C          pack00 : initialize a data packet container.
150: C          packri : re-initialize a data packet container to enable
151: C                  appending more packets.
152: C          packnd : pack numeric data( I4, R4, R8 )
153: C          packnn : pack a numeric data array repeatedly ( I4, R4, R8 )
154: C          packpt : create a specified size of packet and return position
155: C                  of data instead of storing data(I4,R4,R8)
156: C          packcs : pack a character string
157: C          packlb : tuck a label packet
158: C
159: C          < positioning etc.>
160: C
161: C          pctrw : place pointer at the top of a packet container
162: C                  and reset status for use
163: C          pctlb : locate at a packet by giving a label
164: C          pctlb2 : locate at a packet by giving a label (silent)
165: C          pctlbr : locate at a packet by backward label search
166: C          pctcls : close a packet container
167: C          pctchk : get information of current packet
168: C
169: C          < unpacking by copying data >
170: C
171: C          Data packets should be retrieved in order of packing in valid
172: C          data types. Positionig by label may be useful to avoid ambiguity.
173: C
174: C          unpknd : unpack numeric data ( I4, R4, R8 )
175: C          unpkcs : unpack a character string
176: C
177: C          < unpacking or check by getting a pointer >
178: C
179: C          unpkpt : return pointer to packed data & proceed to the next
180: C                  packet
181: C
182: C          < checking status >
183: C
184: C          pcterr : return number of errors
185: C
186: C          < dump >
187: C
188: C          pctdmp : printout contents of container.
189: C
190: C-----
191: C
192: C
193: C          subroutine PACK00( PACKET,MESSAGE,          NPSIZE,IRET )
194: C=====
195: C          Packet container initialization:

```

```

196: C
197: C          packet : memory area used as packet container
198: C                  (it must begin on a double word boundary in 32 bit
199: C                  architecture.)
200: C          npsize : initial size of packet container in word.
201: C                  (on closing container, size information of the container
202: C                  is set to the effective length.)
203: C          iret : return code
204: C                  0 = initialized succesfully
205: C                  1 = not start from double word boundary (not implimented)
206: C                  2 = npsize is insufficient to consturct container.
207: C=====
208: C
209: C          ... packet ....
210: C
211: C          integer PACKET(*)
212: C
213: C          character*(*) MESSAGE
214: C
215: C          ... word length of container header ...
216: C
217: C          parameter( NPHEAD = 4 )
218: C          parameter( LPERR = 5 )
219: C          parameter( LPCUR = 6 )
220: C          parameter( NCHEAD = LPCUR )
221: C
222: C          include 'INC/_IOUNIT'
223: C          include 'INC/_WORDL'
224: C-----
225: C
226: C          ... internal static data ...
227: C
228: C          integer NCON
229: C          data NCON /0/
230: C-----
231: C
232: C          if ( NPSIZE.lt.NCHEAD+NPHEAD ) then
233: C              write(IMG,*) 'XXX NEEDS MORE MEMORY TO PREPARE DATA PACKET',
234: C              & ' CONTAINER.(container no. ', PACKET(1), ' ) <',
235: C              & MESSAGE, '>'
236: C              write(IMG,*) ' size = ', NPSIZE
237: C              IRET = 2
238: C              return
239: C          end if
240: C
241: C          NCON = NCON + 1
242: C          PACKET(1) = NCON
243: C          PACKET(2) = NPSIZE
244: C          ... total number of packets ...
245: C          PACKET(3) = 0
246: C          ... current packet number ...
247: C          PACKET(4) = 0
248: C
249: C          ... lperr : position of error counter ...
250: C
251: C          PACKET(LPERR) = 0
252: C
253: C          ... current packet position ...
254: C
255: C          lpcur : position of current pointer !! ...
256: C
257: C          PACKET(LPCUR) = NCHEAD + 1
258: C
259: C          ... positioning to the first packet ...
260: C

```

src/shared/packet.f

```

261:     PACKET(PACKET(LPCUR)) = -1
262:     PACKET(PACKET(LPCUR)+1) = 0
263:     PACKET(PACKET(LPCUR)+2) = -1
264:     PACKET(PACKET(LPCUR)+3) = -1
265: C
266:     IRET = 0
267:     return
268: end
269: C
270: C
271:     subroutine PACKRI( PACKET,MESSAGE, NPSIZE,IRET )
272: C=====
273: C   Packet container initialization for already close container:
274: C
275: C   packet : memory area used as packet container
276: C   (it must begin on a double word boundary in 32 bit
277: C   architecture.)
278: C   npsize : new initial size of packet container in word.
279: C   (on closing container, size information of the container
280: C   was set to the effective length.)
281: C   iret : return code
282: C   0 = initialized succesfully
283: C   1 = not start from double word boundary (not implimented)
284: C   2 = npsize is insufficient to consturuct container.
285: C=====
286: C
287: C   ... packet ....
288: C
289:     integer PACKET(*)
290: C
291:     character*(*) MESSAGE
292: C
293: C   ... word length of container header ...
294: C
295:     parameter( NPHEAD = 4 )
296:     parameter( LPERR = 5 )
297:     parameter( LPCUR = 6 )
298:     parameter( NCHEAD = LPCUR )
299: C
300:     include 'INC/_IOUNIT'
301:     include 'INC/_WORDL'
302: C-----
303:     IRET = 0
304:     if( NPSIZE.lt.PACKET(2) ) then
305:         write(IMG,*)
306:         & 'XXX(PACKRI) Failed to re-initialize packet container.',
307:         & ' New size ('NPSIZE,') is smaller than current size ('
308:         & PACKET(2),') ',
309:         & ' <',MESSAGE,'>'
310:         IRET = 1
311:         return
312:     end if
313: C
314: C   ... search the last packet checking container integrity ...
315: C
316:     call PCTRW( PACKET,MESSAGE, IRET )
317:     LN = 0
318:     100 LL = PACKET(LPCUR)
319:     JST = PACKET(LL)
320:     LN = LN + 1
321:     if( JST.ne.-2 ) then
322:         if ( JST.eq.-1 ) then
323:             write(IMG,*)
324:             & 'XXX(PACKRI) Found a writable packet position.',
325:             & ' This packet container seems not closed.',

```

```

326:         & ' <',message,'>'
327:         write(IMG,*) ' Number of packets encountered ',LN
328:         IRET = 2
329:         return
330:     end if
331: C
332:     LL0 = LL
333:     LL = PACKET(LL+2)
334: C
335:     if( PACKET(LL+3).ne.LL0 ) then
336:         write(IMG,*)
337:         & 'XXX(PACKRI) packet position link is destroyed?.'
338:         write(IMG,*)
339:         & ' Current packet position: ',ll0
340:         write(IMG,*)
341:         & ' next packet position: ',ll
342:         write(IMG,*)
343:         & ' Recorded current packet pos. in next packet: ',
344:         & packet(ll+3)
345:         write(IMG,*) ' PACKRI is Called with message <',message,'>'
346:         IRET = 3
347:         return
348:     end if
349:     PACKET(LPCUR) = LL
350:     PACKET(4) = PACKET(4) + 1
351:     goto 100
352: endif
353: C
354: C   ... reset the last packet as writable
355: C
356:     PACKET(PACKET(LPCUR)) = -1
357: C
358:     PACKET(2) = NPSIZE
359: C
360:     IRET = 0
361:     return
362: end
363: C-----
364: C
365: C   < packing >
366: C
367:     subroutine PACKND( PACKET,MESSAGE, TYPE, DATA, NDATA, IRET )
368: C=====
369: C   : pack numeric data( I4, R4, R8 )
370: C   message : message string (not stored in container, but for
371: C   error message etc.)
372: C   type : data type packed
373: C   'I4' : integer 'R4' : real 'R8' : real*8
374: C   data : packed data ( declared as integer )
375: C   ndata : number of data packed
376: C   iret : return code
377: C   0 = succesful
378: C   1 = cannot pack. Container size insufficient
379: C   2 = cannot pack. Container already closed
380: C   3 = unsupported data type
381: C   4 = too long data to pack
382: C=====
383: C   ... packet ....
384: C
385:     integer PACKET(*)
386: C
387: C   ... data packed / unpcked ...
388: C
389:     integer DATA(*)
390:     character*(*) MESSAGE

```

src/shared/packet.f

```

391: C
392:     character*(*) TYPE
393: C
394: C ... word length of container header ...
395: C
396:     parameter( NPHEAD = 4 )
397: Cccc parameter( LPCUR = 5 ) ! Sep 1995
398:     parameter( LPERR = 5 )
399:     parameter( LPCUR = 6 )
400:     parameter( NCHEAD = LPCUR )
401: C
402:     include 'INC/_IOUNIT'
403:     include 'INC/_WORDL'
404: C-----
405:     LL = PACKET(LPCUR)
406:     LH1 = PACKET(LL)
407: C
408:     if ( LH1.eq.-2 ) then
409:         write(IMG,*)
410:         & 'XXX(PACKND) you are trying to add new packet to closed',
411:         & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
412:         write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
413:         IRET = 2
414:         PACKET(LPERR) = PACKET(LPERR) + 1
415:         return
416:     end if
417: C
418:     LP = LL + NPHEAD
419:     ND = NDATA
420:     if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
421:         ND = NDATA*MWORD
422:         if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
423:     end if
424: C
425: c if ( NDATA.ge.2**24 ) then
426: c     write(IPR,*) 'XXX tried to pack too long data to ',
427: c & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
428: c & write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
429: c     IRET = 4
430: c     PACKET(LPERR) = PACKET(LPERR) + 1
431: c     return
432: c end if
433: C
434:     LDS = LP + ND
435:     LDLAST = LDS + NPHEAD - 1
436:     if ( LDLAST.gt.PACKET(2) ) then
437:         write(IMG,*) 'XXX(PACKND) packet container overflow.',
438:         & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
439:         write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
440:         IRET = 1
441:         PACKET(LPERR) = PACKET(LPERR) + 1
442:         return
443:     end if
444: C
445:     if ( TYPE.eq.'I4' ) then
446:         ITYP = 1
447:     else if ( TYPE.eq.'R4' ) then
448:         ITYP = 2
449:     else if ( TYPE.eq.'R8' ) then
450:         ITYP = 3
451:     else
452:         write(IMG,*)
453:         & 'XXX(PACKND) tried to pack data of invalid type to the',
454:         & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
455:         write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA

```

```

456:         IRET = 3
457:         PACKET(LPERR) = PACKET(LPERR) + 1
458:         return
459:     end if
460: C
461: CCCC LHNXT = ITYP*2**24 + NDATA
462: C
463: CCCC PACKET(LL) = LHNXT
464: CCCC PACKET(LL+1) = LDS
465: CCCC PACKET(LL) = ITYP
466: CCCC PACKET(LL+1) = NDATA
467: CCCC PACKET(LL+2) = LDS
468: C
469:     PACKET(3) = PACKET(3) + 1
470:     PACKET(4) = PACKET(4) + 1
471:     PACKET(LPCUR) = LDS
472: C
473: C ... for next packet
474: C
475:     PACKET(LDS) = -1
476:     PACKET(LDS+1) = 0
477:     PACKET(LDS+2) = -1
478:     PACKET(LDS+3) = LL
479: C
480:     do 100 I = 1, ND
481:         PACKET(LP+1+I) = DATA(I)
482:     100 continue
483: C
484:     IRET = 0
485:     return
486: end
487: C
488:     subroutine PACKNN( PACKET,MESSAGE, TYPE, DATA, NDATA, NC, IRET )
489: C=====
490: C : pack a numeric array DATA(NDATA) NC times in a packet
491: C ( I4, R4, R8 )
492: C
493: C Used to create a packet having repeated data structure without
494: C copying data in caller side.
495: C
496: C message : message string (not stored in container, but for
497: C error message etc.)
498: C type : data type packed
499: C 'I4' : integer 'R4' : real 'R8' : real*8
500: C data : packed data ( declared as integer )
501: C ndata : size of data array used as a repeated packing pattern.
502: C nc : data repetition count
503: C iret : return code
504: C 0 = succesful
505: C 1 = cannot pack. Container size insufficient
506: C 2 = cannot pack. Container already closed
507: C 3 = unsupported data type
508: C 4 = too long data to pack
509: C 5 = repetition count is negative
510: C=====
511: C ... packet ....
512: C
513:     integer PACKET(*)
514: C
515: C ... data packed / unpcked ...
516: C
517:     integer DATA(*)
518:     character*(*) MESSAGE
519: C
520:     character*(*) TYPE

```

src/shared/packet.f

```

521: C
522: C ... word length of container header ...
523: C
524: parameter( NPHEAD = 4 )
525: Cccc parameter( LPCUR = 5 ) ! Sep 1995
526: parameter( LPERR = 5 )
527: parameter( LPCUR = 6 )
528: parameter( NCHEAD = LPCUR )
529: C
530: include 'INC/_IUNIT'
531: include 'INC/_WORDL'
532: C-----
533: LL = PACKET(LPCUR)
534: LHL = PACKET(LL)
535: C
536: if ( LHL.eq.-2 ) then
537: write(IMG,*)
538: & 'XXX(PACKNN) you are trying to add new packet to closed'
539: & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
540: write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
541: IRET = 2
542: PACKET(LPERR) = PACKET(LPERR) + 1
543: return
544: end if
545: C
546: if ( NC.lt.0 ) then
547: write(IMG,*)
548: & 'XXX(PACKNN) Data repetition count for packed data'
549: & ' is negative. NC=',NC, ' <', MESSAGE, '>'
550: IRET = 5
551: return
552: end if
553: C
554: LP = LL + NPHEAD
555: ND = NDATA * NC
556: if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
557: ND = NDATA*MWORD*NC
558: if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
559: end if
560: C
561: c if ( NDATA.ge.2**24 ) then
562: c write(IPR,*) 'XXX tried to pack too long data to ',
563: c & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
564: c write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
565: c IRET = 4
566: c PACKET(LPERR) = PACKET(LPERR) + 1
567: c return
568: c end if
569: C
570: LDS = LP + ND
571: LDLAST = LDS + NPHEAD - 1
572: if ( LDLAST.gt.PACKET(2) ) then
573: write(IMG,*) 'XXX(PACKNN) packet container overflow.',
574: & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
575: write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
576: IRET = 1
577: PACKET(LPERR) = PACKET(LPERR) + 1
578: return
579: end if
580: C
581: if ( TYPE.eq.'I4' ) then
582: ITYP = 1
583: else if ( TYPE.eq.'R4' ) then
584: ITYP = 2
585: else if ( TYPE.eq.'R8' ) then

```

```

586: ITYP = 3
587: else
588: write(IMG,*) 'XXX tried to pack data of invalid type to the',
589: & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
590: write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
591: IRET = 3
592: PACKET(LPERR) = PACKET(LPERR) + 1
593: return
594: end if
595:
596: CCCC LHNXT = ITYP*2**24 + NDATA
597: C
598: CCCC PACKET(LL) = LHNXT
599: CCCC PACKET(LL+1) = LDS
600: PACKET(LL) = ITYP
601: PACKET(LL+1) = NDATA
602: PACKET(LL+2) = LDS
603: C
604: PACKET(3) = PACKET(3) + 1
605: PACKET(4) = PACKET(4) + 1
606: PACKET(LPCUR) = LDS
607: C
608: C ... for next packet
609: C
610: PACKET(LDS) = -1
611: PACKET(LDS+1) = 0
612: PACKET(LDS+2) = -1
613: PACKET(LDS+3) = LL
614: C
615: JJ = 0
616: if( NC.gt.0 ) then
617: do 110 J = 1, NC
618: do 100 I = 1, ND/NC
619: PACKET(LP-1+JJ+I) = DATA(I)
620: 100 continue
621: JJ = JJ + ND/NC
622: 110 continue
623: end if
624: C
625: IRET = 0
626: return
627: end
628: C
629: subroutine PACKPT( PACKET,MESSAGE, TYPE, LP, NDATA, IRET )
630: C=====
631: C : get an index pointer to store numeric data( I4, R4, R8 ) as
632: C a packet.
633: C
634: C Data placement on the data area should be done afterwards
635: C and take care not to destroy data structure.
636: C This is useful for a case to place data directly from a file;
637: C
638: C ex:
639: C call packpt( packet, 'integer', 'R4', LP, 1000, IRET )
640: C if( IRET.eq.0 ) read(10) (packet(lp+i-1),i=1,1000)
641: C
642: C An index pointer for 'R8' data is returned as an index for
643: C PACKET(*) array declared as double word type.
644: C
645: C ex.
646: C
647: C real PACKET(100000)
648: C real*8 DPACKET(100000)
649: C equivalence (PACKET,DPACKET)
650: C ...

```

src/shared/packet.f

```

651: C      call packpt( packet, 'integer', 'R4', LP1, 1000, IRET )
652: C      if( IRET.eq.0 ) read(10) (PACKET(LP1+I-1),I=1,1000)
653: C      call packpt( packet, 'integer', 'R8', LP2, 200, IRET )
654: C      if( IRET.eq.0 ) read(10) (DPACKET(LP2+I-1),I=1,200)
655: C
656: C .....
657: C
658: C      message : message string (not stored in container, but for
659: C              error message etc.)
660: C      type : data type packed
661: C      'I4' : integer 'R4' : real 'R8' : real*8
662: C      lp : index pointer of data to be packed afterwards.
663: C      ndata : number of data packed
664: C      iret : return code
665: C          0 = successful
666: C          1 = cannot pack. Container size insufficient
667: C          2 = cannot pack. Container already closed
668: C          3 = unsupported data type
669: C          4 = too long data to pack
670: C=====
671: C      ... packet ....
672: C
673: C      integer PACKET(*)
674: C
675: C      ... data packed / unpacked ...
676: C
677: C      ccccc integer DATA(*)
678: C      character*(*) MESSAGE
679: C
680: C      character*(*) TYPE
681: C
682: C      ... word length of container header ...
683: C
684: C      parameter( NPHEAD = 4 )
685: C      cccc parameter( LPCUR = 5 ) ! Sep 1995
686: C      parameter( LPERR = 5 )
687: C      parameter( LPCUR = 6 )
688: C      parameter( NCHEAD = LPCUR )
689: C
690: C      include 'INC/_IOUNIT'
691: C      include 'INC/_WORDL'
692: C-----
693: C      LL = PACKET(LPCUR)
694: C      LH1 = PACKET(LL)
695: C
696: C      if ( LH1.eq.-2 ) then
697: C          write(IMG,*) 'XXX you are trying to add new packet to closed',
698: C          & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
699: C          write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
700: C          IRET = 2
701: C          PACKET(LPERR) = PACKET(LPERR) + 1
702: C          return
703: C      end if
704: C
705: C      LP = LL + NPHEAD
706: C      ND = NDATA
707: C      if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
708: C          ND = NDATA*MWORD
709: C          if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
710: C      end if
711: C
712: C      if ( NDATA.ge.2**24 ) then
713: C          write(IPR,*) 'XXX tried to pack too long data to ',
714: C          & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
715: C          write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA

```

```

716: C          IRET = 4
717: C          PACKET(LPERR) = PACKET(LPERR) + 1
718: C          return
719: C      end if
720: C
721: C      LDS = LP + ND
722: C      LDLAST = LDS + NPHEAD - 1
723: C      if ( LDLAST.gt.PACKET(2) ) then
724: C          write(IMG,*) 'XXX packet container overflow.',
725: C          & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
726: C          write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
727: C          IRET = 1
728: C          PACKET(LPERR) = PACKET(LPERR) + 1
729: C          return
730: C      end if
731: C
732: C      if ( TYPE.eq.'I4' ) then
733: C          ITYP = 1
734: C      else if ( TYPE.eq.'R4' ) then
735: C          ITYP = 2
736: C      else if ( TYPE.eq.'R8' ) then
737: C          ITYP = 3
738: C      else
739: C          write(IMG,*) 'XXX tried to pack data of invalid type to the',
740: C          & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
741: C          write(IMG,*) ' TYPE ', TYPE, ' ndata = ', NDATA
742: C          IRET = 3
743: C          PACKET(LPERR) = PACKET(LPERR) + 1
744: C          return
745: C      end if
746: C
747: C      ccc LHNXT = ITYP*2**24 + NDATA
748: C
749: C      ccc PACKET(LL) = LHNXT
750: C      ccc PACKET(LL+1) = LDS
751: C      PACKET(LL) = ITYP
752: C      PACKET(LL+1) = NDATA
753: C      PACKET(LL+2) = LDS
754: C
755: C      PACKET(3) = PACKET(3) + 1
756: C      PACKET(4) = PACKET(4) + 1
757: C      PACKET(LPCUR) = LDS
758: C
759: C      PACKET(LDS) = -1
760: C      PACKET(LDS+1) = 0
761: C      PACKET(LDS+2) = -1
762: C      PACKET(LDS+3) = LL
763: C
764: C      ... do not pack data here ...
765: C
766: C      do 100 I = 1, ND
767: C          PACKET(LP-1+I) = DATA(I)
768: C      100 continue
769: C
770: C      if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
771: C          LP = LP/MWORD + MWORD - 1
772: C      end if
773: C
774: C      IRET = 0
775: C      return
776: C      end
777: C=====
778: C      packcs : pack a character string
779: C      pcklb : tuck a label-packet
780: C      message : message string (not stored in container, but for

```

src/shared/packet.f

```

781: C          error message etc.)
782: C      chstr : packed character string or label string (assumed length)
783: C      iret  : return code
784: C          0 = succesful
785: C          1 = cannot pack. Container size insufficient
786: C          2 = cannot pack. Container already closed
787: C          4 = too long data to pack
788: C=====
789: C      subroutine PACKCS( PACKET,MESSAGE,      CHSTR, IRET )
790: C
791: C      ... packet ....
792: C
793: C      integer PACKET(*)
794: C
795: C      character*(*) MESSAGE
796: C      character*(*) CHSTR
797: C
798: C      ... word length of container header ...
799: C
800: C      parameter( NPHEAD = 4 )
801: C      CCCCCparameter( LPCUR = 5 ) ! Sep 1995
802: C      parameter( LPERR = 5 )
803: C      parameter( LPCUR = 6 )
804: C      parameter( NCHEAD = LPCUR )
805: C
806: C      include 'INC/_IOUNIT'
807: C      include 'INC/_WORDL'
808: C-----
809: C      ITYP = 4
810: C
811: C      LL = PACKET(LPCUR)
812: C      LH1 = PACKET(LL)
813: C
814: C      if ( LH1.eq.-2 ) then
815: C          write(IMG,*) 'XXX you are trying to add new packet to closed',
816: C      &          ' container no. ', PACKET(1), ' <', MESSAGE, '>'
817: C          write(IMG,*) ' STRING <', CHSTR, '>'
818: C          IRET = 2
819: C          PACKET(LPERR) = PACKET(LPERR) + 1
820: C          return
821: C      end if
822: C
823: C      LP = LL + NPHEAD
824: C
825: C      if ( LEN(CHSTR).ge.2**24 ) then
826: C          write(IPR,*) 'XXX tried to pack too long data to ',
827: C      &          ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
828: C          write(IPR,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
829: C      &          LEN(CHSTR)
830: C          IRET = 4
831: C          PACKET(LPERR) = PACKET(LPERR) + 1
832: C          return
833: C      end if
834: C
835: C      KWD = 8/MWORD
836: C      ND = (LEN(CHSTR)+KWD-1) /KWD
837: C      LDS = LP + ND
838: C      LDLAST = LDS + NPHEAD - 1
839: C      if ( LDLAST.gt.PACKET(2) ) then
840: C          write(IMG,*) 'XXX packet container overflow.',
841: C      &          ' container no. ', PACKET(1), ' <', MESSAGE, '>'
842: C          write(IMG,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
843: C      &          LEN(CHSTR)
844: C          IRET = 1
845: C          PACKET(LPERR) = PACKET(LPERR) + 1

```

```

846: C          return
847: C      end if
848: C
849: C      LHNXT = ITYP*2**24 + LEN(CHSTR)
850: C
851: C      PACKET(LL) = LHNXT
852: C      PACKET(LL+1) = LDS
853: C      PACKET(LL) = ITYP
854: C      PACKET(LL+1) = LEN(CHSTR)
855: C      PACKET(LL+2) = LDS
856: C
857: C      PACKET(3) = PACKET(3) + 1
858: C      PACKET(4) = PACKET(4) + 1
859: C      PACKET(LPCUR) = LDS
860: C
861: C      PACKET(LDS) = -1
862: C      PACKET(LDS+1) = 0
863: C      PACKET(LDS+2) = -1
864: C      PACKET(LDS+3) = LL
865: C
866: C      ... data transfer by internal read-write may cause compilation or
867: C      execution error for too large data.
868: C
869: C      nn = 10
870: C      write(fmt,'(i4,'a'',il)') nn, 8/mword
871: C      do 201 i=1,len(chstr), nn*(8/mword)
872: C      201 packet(lp-1+i) = data(i+1)
873: C
874: C      call MVCTOI( LEN(CHSTR), CHSTR, PACKET(LP) )
875: C
876: C      IRET = 0
877: C      return
878: C      end
879: C=====
880: C      subroutine PACKLB( PACKET,MESSAGE,      CHSTR, IRET )
881: C
882: C      ... packet ....
883: C
884: C      integer PACKET(*)
885: C
886: C      character*(*) MESSAGE
887: C      character*(*) CHSTR
888: C
889: C      ... word length of container header ...
890: C
891: C      parameter( NPHEAD = 4 )
892: C      Ccccparameter( LPCUR = 5 ) ! Sep 1995
893: C      parameter( LPERR = 5 )
894: C      parameter( LPCUR = 6 )
895: C      parameter( NCHEAD = LPCUR )
896: C
897: C      include 'INC/_IOUNIT'
898: C      include 'INC/_WORDL'
899: C-----
900: C      ITYP = 5
901: C      LL = PACKET(LPCUR)
902: C      LH1 = PACKET(LL)
903: C
904: C      if ( LH1.eq.-2 ) then
905: C          write(IMG,*) 'XXX you are trying to add new packet to closed',
906: C      &          ' container no. ', PACKET(1), ' <', MESSAGE, '>'
907: C          write(IMG,*) ' STRING <', CHSTR, '>'
908: C          IRET = 2
909: C          PACKET(LPERR) = PACKET(LPERR) + 1
910: C          return

```


src/shared/packet.f

```

911:      end if
912: C
913:      LP      = LL + NPHEAD
914: C
915: c      if ( LEN(CHSTR).ge.2**24 ) then
916: c          write(IPR,*) 'XXX tried to pack too long data to ',
917: c      &          ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
918: c          write(IPR,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
919: c      &          LEN(CHSTR)
920: c          IRET      = 4
921: c          PACKET(LPERR) = PACKET(LPERR) + 1
922: c          return
923: c      end if
924: C
925:      KWD      = 8/MWORD
926:      ND      = (LEN(CHSTR)+KWD-1)/KWD
927:      LDS      = LP + NP
928:      LDLAST   = LDS + NPHEAD - 1
929: c      if ( LDLAST.gt.PACKET(2) ) then
930: c          write(IMG,*) 'XXX packet container overflow.'
931: c      &          ' container no. ', PACKET(1), ' <', MESSAGE, '>'
932: c          write(IMG,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
933: c      &          LEN(CHSTR)
934: c          IRET      = 1
935: c          PACKET(LPERR) = PACKET(LPERR) + 1
936: c          return
937: c      end if
938: C
939: ccc      LHNXT      = ITYP*2**24 + LEN(CHSTR)
940: C
941: ccc      PACKET(LL) = LHNXT
942: ccc      PACKET(LL+1) = LDS
943: C
944:      PACKET(LL) = ITYP
945:      PACKET(LL+1) = LEN(CHSTR)
946:      PACKET(LL+2) = LDS
947: C
948:      PACKET(3) = PACKET(3) + 1
949:      PACKET(4) = PACKET(4) + 1
950:      PACKET(LPCUR) = LDS
951: C
952:      PACKET(LDS) = -1
953:      PACKET(LDS+1) = 0
954:      PACKET(LDS+2) = -1
955:      PACKET(LDS+3) = LL
956: C
957:      call MVCTOI( LEN(CHSTR), CHSTR, PACKET(LP) )
958: C
959:      IRET      = 0
960:      return
961:      end
962: C-----
963: C
964: C < positioning etc.>
965: C
966:      subroutine PCTRW( PACKET,MESSAGE,      IRET )
967: C=====
968: C : place pointer at the top of a packet container and reset status
969: C for use
970: C=====
971: C
972: C ... packet ....
973: C
974:      integer PACKET(*)
975: C

```

```

976:      character*(*) MESSAGE
977: C
978: C ... word length of container header ...
979: C
980:      parameter( NPHEAD      = 4 )
981:      parameter( LPERR       = 5 )
982:      parameter( LPCUR       = 6 )
983:      parameter( NCHEAD      = LPCUR )
984: C
985:      include 'INC/_IOUNIT'
986:      include 'INC/_WORDL'
987: C-----
988:      PACKET(4) = 0
989:      PACKET(LPCUR) = NCHEAD + 1
990:      IRET      = 0
991:      return
992:      end
993: C-----
994:      subroutine PCTLB( PACKET,MESSAGE,      CHSTR, IRET )
995: C=====
996: C : locate at a packet by giving a label
997: C   chstr : label
998: C   iret  : return code
999: C         0 = found record successfully
1000: C         1 = no such a label
1001: C=====
1002: C
1003: C ... packet ....
1004: C
1005:      integer PACKET(*)
1006: C
1007:      character*(*) MESSAGE
1008:      character*(*) CHSTR
1009:      call PCTLB0( PACKET,MESSAGE,      CHSTR, 0, IRET )
1010:      return
1011:      end
1012: C-----
1013:      subroutine PCTLB2( PACKET,MESSAGE,      CHSTR, IRET )
1014: C=====
1015: C : locate at a packet by giving a label (do not print error message
1016: C   if the label is not found )
1017: C   chstr : label
1018: C   iret  : return code
1019: C         0 = found record successfully
1020: C         1 = no such a label
1021: C=====
1022: C
1023: C ... packet ....
1024: C
1025:      integer PACKET(*)
1026: C
1027:      character*(*) MESSAGE
1028:      character*(*) CHSTR
1029:      call PCTLB0( PACKET,MESSAGE,      CHSTR, 1, IRET )
1030:      return
1031:      end
1032: C-----
1033:      subroutine PCTLB0( PACKET,MESSAGE,      CHSTR, isilnt, IRET )
1034: C=====
1035: C : locate at a packet by giving a label
1036: C   chstr : label
1037: C   isilnt : 1 = do not print error message if specified label is
1038: C             not found.
1039: C             0 = print error message
1040: C   iret  : return code

```

src/shared/packet.f

```

1041: C      0 = found record successfully
1042: C      1 = no such a label
1043: C=====
1044: C
1045: C      ... packet ....
1046: C
1047: C      integer PACKET(*)
1048: C
1049: C      character*(*) MESSAGE
1050: C      character*(*) CHSTR
1051: C
1052: C      ... word length of container header ...
1053: C
1054: C      parameter( NPHEAD = 4 )
1055: C      parameter( LPCUR = 5 ) ! Sep 1995
1056: C      parameter( LPERR = 5 )
1057: C      parameter( LPCUR = 6 )
1058: C      parameter( NCHEAD = LPCUR )
1059: C
1060: C      include 'INC/_IOUNIT'
1061: C      include 'INC/_WORDL'
1062: C-----
1063: C      LC = PACKET(LPCUR)
1064: C      IP = PACKET(4)
1065: C      LROT = 0
1066: C      100 continue
1067: C      LL = PACKET(LPCUR)
1068: C      LH1 = PACKET(LL)
1069: C      if ( LH1.gt.0 ) then
1070: C      cccccc ITYP = LH1/2**24
1071: C      ITYP = LH1
1072: C
1073: C      if ( ITYP.eq.5 ) then
1074: C      IM = 0
1075: C      cccccc ND = LH1 - ITYP*2**24
1076: C      ND = PACKET(LL+1)
1077: C      if ( ND.eq.LEN(CHSTR) )
1078: C      & call CMPCHI( ND, IM, CHSTR, PACKET(LL+NPHEAD) )
1079: C
1080: C      .... label found ....
1081: C
1082: C      if ( IM.eq.1 ) then
1083: C      IRET = 0
1084: C      PACKET(4) = PACKET(4) + 1
1085: C      cccccc PACKET(LPCUR) = PACKET(LL+1)
1086: C      PACKET(LPCUR) = PACKET(LL+2)
1087: C      return
1088: C      end if
1089: C      end if
1090: C      PACKET(4) = PACKET(4) + 1
1091: C      cccccc PACKET(LPCUR) = PACKET(LL+1)
1092: C      PACKET(LPCUR) = PACKET(LL+2)
1093: C      go to 100
1094: C      else
1095: C
1096: C      ... not found ...
1097: C
1098: C      if ( LROT.ne.0 ) then
1099: C      IRET = 1
1100: C      if ( ISILNT.eq.0 ) then
1101: C      PACKET(LPERR) = PACKET(LPERR) + 1
1102: C      write(MSG,*) 'XXX CANNOT FIND LABEL <', CHSTR,
1103: C      & '>' IN DATA PACKET ' ', ' CONTAINER NO. ', PACKET(1),
1104: C      & ' <', MESSAGE, '>'
1105: C      end if

```

```

1106: C      PACKET(4) = IP
1107: C      PACKET(LPCUR) = LC
1108: C      return
1109: C      else
1110: C      LROT = 1
1111: C      PACKET(4) = 1
1112: C      PACKET(LPCUR) = NCHEAD + 1
1113: C      go to 100
1114: C      end if
1115: C      end if
1116: C      IRET = 0
1117: C      return
1118: C      end
1119: C-----
1120: C      subroutine PCTLBR( PACKET,MESSAGE, CHSTR, isilnt, IRET )
1121: C-----
1122: C      : locate at a packet by giving a label and make backward search from
1123: C      current position.
1124: C      chstr : label
1125: C      isilnt : 1 = do not print error message if specified label is
1126: C      not found.
1127: C      0 = print error message
1128: C      iret : return code
1129: C      0 = found record successfully
1130: C      1 = no such a label
1131: C-----
1132: C
1133: C      ... packet ....
1134: C
1135: C      integer PACKET(*)
1136: C
1137: C      character*(*) MESSAGE
1138: C      character*(*) CHSTR
1139: C
1140: C      ... word length of container header ...
1141: C
1142: C      parameter( NPHEAD = 4 )
1143: C      parameter( LPERR = 5 )
1144: C      parameter( LPCUR = 6 )
1145: C      parameter( NCHEAD = LPCUR )
1146: C
1147: C      include 'INC/_IOUNIT'
1148: C      include 'INC/_WORDL'
1149: C-----
1150: C      LC = PACKET(LPCUR)
1151: C      IP = PACKET(4)
1152: C      LROT = 0
1153: C      100 continue
1154: C      LL = PACKET(LPCUR)
1155: C      ITYP = PACKET(LL)
1156: C      check %%%%%%%%%
1157: C      write(*,*) '(PCTLBR) LL ',LL, 'ityp ',ityp
1158: C      %%%%%%%%%
1159: C      if ( ITYP.eq.5 ) then
1160: C      IM = 0
1161: C      ND = PACKET(LL+1)
1162: C      if ( ND.eq.LEN(CHSTR) ) then
1163: C      call CMPCHI( ND, IM, CHSTR, PACKET(LL+NPHEAD) )
1164: C      end if
1165: C
1166: C      .... label found ....
1167: C
1168: C      if ( IM.eq.1 ) then
1169: C      IRET = 0
1170: C      PACKET(4) = PACKET(4) + 1

```

src/shared/packet.f

```

1171:         PACKET(LPCUR) = PACKET(LL+2)
1172:         return
1173:     end if
1174: end if
1175: C
1176:     PACKET(4) = PACKET(4) - 1
1177:     PACKET(LPCUR) = PACKET(LL+3)
1178: C
1179:     if(PACKET(4).gt.0.and.PACKET(LPCUR).gt.0) go to 100
1180: C
1181: C ... not found ...
1182: C
1183:     if( PACKET(4).le.0 ) then
1184:         IRET = 1
1185:         if ( ISILNT.eq.0 ) then
1186:             PACKET(LPERR) = PACKET(LPERR) + 1
1187:             write(IMG,*) 'XXX CANNOT FIND LABEL <', CHSTR,
1188: & ' > IN DATA PACKET ', ' CONTAINER NO. ', PACKET(1),
1189: & ' <', MESSAGE, '>'
1190:         end if
1191:         PACKET(4) = IP
1192:         PACKET(LPCUR) = LC
1193:         return
1194:     end if
1195: C
1196:     IRET = 0
1197:     return
1198: end
1199: C-----
1200:     subroutine PCTCLS( PACKET,MESSAGE,      NPSIZE,NSIZE, IRET )
1201: C=====
1202: C : close a packet container , and return effective size of container.
1203: C
1204: C     npsize : given size in initialization (output)
1205: C     nszie  : effective size (output)
1206: C=====
1207: C
1208: C ... packet ....
1209: C
1210:     integer PACKET(*)
1211: C
1212:     character*(*) MESSAGE
1213: C
1214: C ... word length of container header ...
1215: C
1216:     parameter( NPHEAD = 4 )
1217: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1218:     parameter( LPERR = 5 )
1219:     parameter( LPCUR = 6 )
1220:     parameter( NCHEAD = LPCUR )
1221: C
1222:     include 'INC/_IOUNIT'
1223:     include 'INC/_WORDL'
1224: C-----
1225:     LL = PACKET(LPCUR)
1226:     IP = PACKET(4)
1227:     100 continue
1228:     if ( PACKET(LL).gt.0 ) then
1229: Ccccc LL = PACKET(LL+1)
1230:         LL = PACKET(LL+2)
1231:         IP = IP + 1
1232:         go to 100
1233:     end if
1234: C
1235:     if ( PACKET(LL).eq.-1 ) then

```

```

1236:         NPSIZE = PACKET(2)
1237: CCCC PACKET(2) = LL + 1
1238:     PACKET(2) = LL + NPHEAD - 1
1239:     NSIZE = PACKET(2)
1240:     PACKET(LL) = -2
1241:     IRET = 0
1242:     return
1243:     else if ( PACKET(LL).eq.-2 ) then
1244:         write(IMG,*) 'XXX(PCTCLS) cannot close data packet container ',
1245: & PACKET(1), ' <', MESSAGE, '>'
1246:         write(IMG,*) ' LOCATION ', LL, ' header ', PACKET(LL),
1247: & PACKET(LL+1),PACKET(LL+2),PACKET(LL+3),
1248: & ' PACKET # ', IP
1249:         IRET = 1
1250:         PACKET(LPERR) = PACKET(LPERR) + 1
1251:         return
1252:     else
1253:         write(IMG,*) 'XXX(PCTCLS) invalid packet header ', PACKET(LL),
1254: & ' <',MESSAGE, '>'
1255:         IRET = 999
1256:         PACKET(LPERR) = PACKET(LPERR) + 1
1257:     end if
1258: C
1259:     return
1260: end
1261: C-----
1262: C
1263: C < get information of currently positioned packet >
1264: C
1265:     subroutine PCTCHK( PACKET,MESSAGE,      TYPE, LPT,  NDATA, IRET
1266: & )
1267: C=====
1268: C : check contents of currently positioned packet
1269: C returns pointer & data size but does not proceed to next packet.
1270: C (does not skip label packet )
1271: C
1272: C type : return data type of packet
1273: C 'I4','R4','R8','CH','LB' or blank for end of container
1274: C lpt : pointer to data portion of packet. ( packet(lpt) )
1275: C for 'R8' type, lpt is array index to real*8 array
1276: C having the same starting address with 'packet'.
1277: C ndata : number of data packed in data packet.
1278: C iret : return code
1279: C 0 : successful
1280: C 99 : invalid packet!
1281: C=====
1282: C
1283: C ... packet ....
1284: C
1285:     integer PACKET(*)
1286: C
1287:     character*(*) MESSAGE
1288:     character*(*) TYPE
1289: C
1290: C ... word length of container header ...
1291: C
1292:     parameter( NPHEAD = 4 )
1293: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1294:     parameter( LPERR = 5 )
1295:     parameter( LPCUR = 6 )
1296:     parameter( NCHEAD = LPCUR )
1297: C
1298:     include 'INC/_IOUNIT'
1299:     include 'INC/_WORDL'
1300: C-----

```

src/shared/packet.f

```

1301:      NDATA = 0
1302:      LPT = 0
1303:      LL = PACKET(LPCUR)
1304: C
1305:      LH1 = PACKET(LL)
1306:      if ( LH1.lt.0 ) then
1307:      IRET = 0
1308:      TYPE = ' '
1309:      return
1310:      end if
1311: C
1312: ccc ITYP = LH1/2**24
1313: ITYP = PACKET(LL)
1314: C
1315:      if ( ITYP.lt.1 .or. ITYP.gt.5 ) then
1316:      write(IMG,*) 'XXX UNSUPPORTED TYPE OF DATA IN PACKET ',
1317: & ' packet container ', PACKET(1), ' packet # ',
1318: & PACKET(4), ' < ', MESSAGE, '>'
1319:      write(IMG,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1320:      IRET = 99
1321:      PACKET(LPERR) = PACKET(LPERR) + 1
1322:      return
1323:      end if
1324: C
1325:      if ( ITYP.eq.1 ) TYPE = 'I4'
1326:      if ( ITYP.eq.2 ) TYPE = 'R4'
1327:      if ( ITYP.eq.3 ) TYPE = 'R8'
1328:      if ( ITYP.eq.4 ) TYPE = 'CH'
1329:      if ( ITYP.eq.5 ) TYPE = 'LB'
1330: C
1331: cccc NDATA = LH1 - ITYP*2**24
1332: NDATA = PACKET(LL+1)
1333: LPT = LL + NPHEAD
1334: C
1335:      if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
1336:      if ( MOD(LPT,MWORD).ne.1 ) LPT = LPT + 1
1337:      LPT = LPT/MWORD + MWORD - 1
1338:      end if
1339: C
1340:      IRET = 0
1341:      return
1342:      end
1343: C-----
1344: C
1345: C < return the number of error >
1346: C
1347:      subroutine PCTERR( PACKET,MESSAGE, NERR, IRET )
1348: C-----
1349: C : return the number of error
1350: C
1351: C nerr :
1352: C iret : return code
1353: C 0 : successful
1354: C-----
1355: C
1356: C ... packet ....
1357: C
1358:      integer PACKET(*)
1359: C
1360:      character*(*) MESSAGE
1361: cccc character*(*) TYPE
1362: C
1363: C ... word length of container header ...
1364: C
1365:      parameter( NPHEAD = 4 )

```

```

1366: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1367:      parameter( LPERR = 5 )
1368:      parameter( LPCUR = 6 )
1369:      parameter( NCHEAD = LPCUR )
1370: C
1371: C-----
1372:      IRET = 0
1373:      NERR = PACKET(LPERR)
1374:      return
1375:      end
1376: C-----
1377: C
1378: C < unpacking by copying data >
1379: C
1380: C Data packets should be retrieved in order of packing with valid
1381: C data types. Positioning by label may be useful to avoid ambiguity.
1382: C
1383: C-----
1384:      subroutine UNPKND( PACKET,MESSAGE, TYPE, DATA, NDATA,
1385: & NDATA2,IRET )
1386: C=====
1387: C : unpack numeric data ( I4, R4, R8 )
1388: C message : message string (not stored in container, but for
1389: C error message etc.)
1390: C type : data type packed
1391: C 'I4' : integer 'R4' : real 'R8' : real*8
1392: C data : packed data ( declared as integer )
1393: C ndata : number of data to be unpacked
1394: C ndata2 : number of data really unpacked
1395: C iret : return code
1396: C 0 = successful ( ndata2 <= ndata )
1397: C 1 = no more data packet in this container.
1398: C 2 = data packet includes more data than ndata.
1399: C 3 = packet itself is invalid.
1400: C -N = data type mismatch.
1401: C N is type of data in packet
1402: C ( 1/2/3/4 = I4/R4/R8/CHAR )
1403: C=====
1404: C
1405: C ... packet ....
1406: C
1407:      integer PACKET(*)
1408: C
1409:      character*(*) MESSAGE
1410:      character*(*) TYPE
1411:      integer DATA(*)
1412: C
1413: C ... word length of container header ...
1414: C
1415:      parameter( NPHEAD = 4 )
1416: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1417:      parameter( LPERR = 5 )
1418:      parameter( LPCUR = 6 )
1419:      parameter( NCHEAD = LPCUR )
1420: C
1421:      include 'INC/_IOUNIT'
1422:      include 'INC/_WORDL'
1423: C-----
1424:      IRET = 0
1425:      NDATA2 = 0
1426:      100 LL = PACKET(LPCUR)
1427: C
1428:      LH1 = PACKET(LL)
1429:      if ( LH1.lt.0 ) then
1430:      IRET = 1

```

src/shared/packet.f

```

1431:      PACKET(LPERR) = PACKET(LPERR) + 1
1432:      write(IMG,*) 'XXX CANNOT FIND DATA <', MESSAGE, '> OF TYPE <',
1433:      &      TYPE, '> IN DATA PACKET ', ' CONTAINER NO. ',
1434:      &      PACKET(1), ', END OF CONTAINER.'
1435:      return
1436:    end if
1437:  C
1438:  CCC      ITYP = LH1/2**24
1439:      ITYP = LH1
1440:  C
1441:  C ... skip label ...
1442:  C
1443:      if ( ITYP.eq.5 ) then
1444:  cccc      PACKET(LPCUR) = PACKET(LL+1)
1445:      PACKET(LPCUR) = PACKET(LL+2)
1446:      PACKET(4) = PACKET(4) + 1
1447:      go to 100
1448:    end if
1449:  C
1450:      if ( ITYP.lt.1 .or. ITYP.gt.4 ) then
1451:      write(IMG,*) 'XXX UNSUPPORTED TYPE OF DATA IN PACKET ',
1452:      &      ' packet container ', PACKET(1), ' packet # ',
1453:      &      PACKET(4), ' <', MESSAGE, '>'
1454:      write(IMG,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1455:      IRET = 3
1456:      PACKET(LPERR) = PACKET(LPERR) + 1
1457:  cccc      PACKET(LPCUR) = PACKET(LL+1)
1458:      PACKET(LPCUR) = PACKET(LL+2)
1459:      PACKET(4) = PACKET(4) + 1
1460:      return
1461:    end if
1462:  C
1463:      if ( TYPE.eq.'I4'.and.ITYP.ne.1 .or. TYPE.eq.'R4'
1464:      &      .and.ITYP.ne.2 .or. TYPE.eq.'R8'.and.ITYP.ne.3 ) then
1465:      IRET = -ITYP
1466:      PACKET(LPERR) = PACKET(LPERR) + 1
1467:      write(IMG,*) 'XXX DATA <', MESSAGE, '> OF TYPE <', TYPE,
1468:      &      '> REQUIRED IN DATA PACKET ', ' CONTAINER NO. ',
1469:      &      PACKET(1), ' BUT STORED TYPE IS ', ITYP,
1470:      &      ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1471:  cccc      PACKET(LPCUR) = PACKET(LL+1)
1472:      PACKET(LPCUR) = PACKET(LL+2)
1473:      PACKET(4) = PACKET(4) + 1
1474:      return
1475:    end if
1476:  C
1477:  ccc      ND0 = LH1 - ITYP*2**24
1478:      ND0 = PACKET(LL+1)
1479:      NDATA2 = MIN(ND0,NDATA)
1480:      LP = LL + NPHEAD
1481:      ND = NDATA2
1482:      if ( TYPE(1:2).eq.'R8'.and.MWORD.gt.1 ) then
1483:      ND = NDATA2*MWORD
1484:      if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
1485:  c...unnecessary
1486:  ccc      LP = LP/MWORD + MWORD - 1
1487:    end if
1488:  C
1489:  C
1490:      do 110 I = 1, ND
1491:      DATA(I) = PACKET(LP+I-1)
1492:  110 continue
1493:  C
1494:      if ( NDATA.lt.ND0 ) then
1495:      IRET = 2

```

```

1496:      PACKET(LPERR) = PACKET(LPERR) + 1
1497:      write(IMG,*) 'XXX DATA <', MESSAGE, '> OF TYPE <', TYPE,
1498:      &      '> STORED WITH LENGTH ', ND0, ' IN DATA PACKET ',
1499:      &      ' CONTAINER NO. ', PACKET(1), ' BUT INPUT BUFFER IS ',
1500:      &      ' SMALL TO RESTORE ALL DATA (', NDATA, ')'
1501:    end if
1502:  C
1503:  cccc      PACKET(LPCUR) = PACKET(LL+1)
1504:      PACKET(LPCUR) = PACKET(LL+2)
1505:      PACKET(4) = PACKET(4) + 1
1506:      IRET = 0
1507:      return
1508:    end
1509:  C-----
1510:      subroutine UNPKCS( PACKET,MESSAGE,      CHSTR, NDATA2,IRET )
1511:  C=====
1512:  C : unpack a character string
1513:  C      message : message string (not stored in container, but for
1514:  C      error message etc.)
1515:  C      chstr : unpacked string
1516:  C      ndata2 : number of data really unpacked
1517:  C      iret : return code
1518:  C      0 = successful ( ndata2 <= len(chstr) )
1519:  C      1 = no more data packet in this container.
1520:  C      2 = data packet includes more data than len(chstr).
1521:  C      99 = packet itself is invalid.
1522:  C      -N = data type mismatch.
1523:  C      N is type of data in packet
1524:  C      ( 1/2/3/4 = I4/R4/R8/CHAR )
1525:  C=====
1526:  C
1527:  C ... packet ...
1528:  C
1529:      integer PACKET(*)
1530:  C
1531:      character*(*) MESSAGE
1532:      character*(*) CHSTR
1533:  C
1534:  C ... word length of container header ...
1535:  C
1536:      parameter( NPHEAD = 4 )
1537:  Cccc      parameter( LPCUR = 5 ) ! Sep 1995
1538:      parameter( LPERR = 5 )
1539:      parameter( LPCUR = 6 )
1540:      parameter( NCHEAD = LPCUR )
1541:  C
1542:      include 'INC/_IOUNIT'
1543:      include 'INC/_WORDL'
1544:  C-----
1545:      NDATA2 = 0
1546:      100 LL = PACKET(LPCUR)
1547:  C
1548:      LH1 = PACKET(LL)
1549:      if ( LH1.lt.0 ) then
1550:      IRET = 1
1551:      PACKET(LPERR) = PACKET(LPERR) + 1
1552:      write(IMG,*) 'XXX CANNOT FIND DATA <', MESSAGE,
1553:      &      '> OF STRING TYPE IN DATA PACKET ', ' CONTAINER NO. ',
1554:      &      PACKET(1), ', END OF CONTAINER.'
1555:      return
1556:    end if
1557:  C
1558:  cccc      ITYP = LH1/2**24
1559:      ITYP = LH1
1560:  C

```

src/shared/packet.f

```

1561: C ... skip label ...
1562: C
1563:       if ( ITYP.eq.5 ) then
1564: ccc      PACKET(LPCUR)   = PACKET(LL+1)
1565:          PACKET(LPCUR)   = PACKET(LL+2)
1566:          PACKET(4)       = PACKET(4) + 1
1567:          go to 100
1568:       end if
1569: C
1570:       if ( ITYP.lt.1 .or. ITYP.gt.4 ) then
1571:          write(IMG*,*) 'XXX UNSUPPORTED TYPE OF DATA IN PACKET ',
1572: &      ' CONTAINER NO. ', PACKET(1), ' packet # ', PACKET(4),
1573: &      ' <', MESSAGE, '>'
1574:          write(IMG*,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1575:          IRET = 99
1576:          PACKET(LPERR) = PACKET(LPERR) + 1
1577: ccc      PACKET(LPCUR)   = PACKET(LL+1)
1578:          PACKET(LPCUR)   = PACKET(LL+2)
1579:          PACKET(4)       = PACKET(4) + 1
1580:          return
1581:       end if
1582: C
1583:       if ( ITYP.ne.4 ) then
1584:          IRET = -ITYP
1585:          PACKET(LPERR) = PACKET(LPERR) + 1
1586:          write(IMG*,*) 'XXX DATA <', MESSAGE,
1587: &      '> OF TYPE <CH> REQUIRED IN DATA PACKET ',
1588: &      ' CONTAINER NO. ', PACKET(1), ' BUT STORED TYPE IS ',
1589: &      ITYP, ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1590: ccc      PACKET(LPCUR)   = PACKET(LL+1)
1591:          PACKET(LPCUR)   = PACKET(LL+2)
1592:          PACKET(4)       = PACKET(4) + 1
1593:          return
1594:       end if
1595: C
1596: CCC      ND0           = LH1 - ITYP*2**24
1597:          ND0           = PACKET(LL+1)
1598:          NDATA2        = MIN(ND0,LEN(CHSTR))
1599:          LP             = LL + NPHEAD
1600: C
1601:          call MVITOC( NDATA2, CHSTR, PACKET(LP) )
1602: C
1603:       if ( NDATA2.lt.ND0 ) then
1604:          IRET = 2
1605:          PACKET(LPERR) = PACKET(LPERR) + 1
1606:          write(IMG*,*) 'XXX DATA <', MESSAGE,
1607: &      '> OF TYPE <CH> STORED WITH LENGTH ', ND0,
1608: &      ' IN DATA PACKET ', ' CONTAINER NO. ', PACKET(1),
1609: &      ' BUT INPUT BUFFER IS ', 'SMALL TO RESTORE ALL DATA (',
1610: &      LEN(CHSTR), ') '
1611:       end if
1612: C
1613: cccc     PACKET(LPCUR)   = PACKET(LL+1)
1614:          PACKET(LPCUR)   = PACKET(LL+2)
1615:          PACKET(4)       = PACKET(4) + 1
1616:          IRET            = 0
1617:          return
1618:       end
1619: C-----
1620: C
1621: C < unpacking by getting pointer to data >
1622: C
1623:       subroutine UNPKPT( PACKET,MESSAGE,          TYPE, LPT,  NDATA, IRET
1624: &      )
1625: C=====

```

```

1626: C : unpack numeric/character data by returning pointer ( I4, R4, R8 )
1627: C returns pointer & data size and proceed to next packet.
1628: C (does not skip label packet )
1629: C
1630: C type : data type
1631: C lpt : pointer to data portion of packet. ( packet(ipt) )
1632: C for 'R8' type, lpt is array index to real*8 array
1633: C having the same starting address with 'packet'.
1634: C ndata : number of data packed in data packet.
1635: C iret : return code
1636: C 0 : successful
1637: C 1 : no more packet
1638: C 99 : invalid packet!
1639: C -N : data type mismatch.
1640: C N is type of data in packet
1641: C ( 1/2/3/4/5 = I4/R4/R8/CHAR/LB )
1642: C=====
1643: C
1644: C ... packet ....
1645: C
1646: C integer PACKET(*)
1647: C
1648: C character*(*) MESSAGE
1649: C character*(*) TYPE
1650: C
1651: C ... word length of container header ...
1652: C
1653: C parameter( NPHEAD = 4 )
1654: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1655: C parameter( LPERR = 5 )
1656: C parameter( LPCUR = 6 )
1657: C parameter( NCHEAD = LPCUR )
1658: C
1659: C include 'INC/_IOUNIT'
1660: C include 'INC/_WORDL'
1661: C-----
1662: C
1663: C NDATA = 0
1664: C LPT = 0
1665: C LL = PACKET(LPCUR)
1666: C
1667: C LH1 = PACKET(LL)
1668: C if ( LH1.lt.0 ) then
1669: C IRET = 1
1670: C PACKET(LPERR) = PACKET(LPERR) + 1
1671: C write(IMG*,*) 'XXX CANNOT FIND DATA <', MESSAGE, '> OF TYPE <',
1672: &      TYPE, '> IN DATA PACKET ', ' CONTAINER NO. ',
1673: &      PACKET(1), ' , END OF CONTAINER.'
1674: C return
1675: C end if
1676: C
1677: C ITYP = LH1/2**24
1678: C ITYP = LH1
1679: C
1680: C if ( ITYP.lt.1 .or. ITYP.gt.5 ) then
1681: C write(IMG*,*) 'XXX UNSUPPORTED TYPE OF DATA IN PACKET ',
1682: &      ' packet container ', PACKET(1), ' packet # ',
1683: &      PACKET(4), ' <', MESSAGE, '>'
1684: C write(IMG*,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1685: C IRET = 99
1686: C PACKET(LPERR) = PACKET(LPERR) + 1
1687: ccc      PACKET(LPCUR)   = PACKET(LL+1)
1688:          PACKET(LPCUR)   = PACKET(LL+2)
1689:          PACKET(4)       = PACKET(4) + 1
1690:          return
1691:       end if

```

src/shared/packet.f

```

1691: C
1692:       if ( ITYP.eq.1.and.TYPE.ne.'I4' .or. ITYP.eq.2
1693:       & .and.TYPE.ne.'R4' .or. ITYP.eq.3.and.TYPE.ne.'R8' .or. ITYP.eq.
1694:       & 4.and.TYPE.ne.'CH' .or. ITYP.eq.5.and.TYPE.ne.'LB' ) then
1695:       IRET = -ITYP
1696:       PACKET(LPERR) = PACKET(LPERR) + 1
1697:       write(IMG,*) 'XXX DATA <', MESSAGE, '> OF TYPE <', TYPE,
1698:       & '> REQUIRED IN DATA PACKET ', ' CONTAINER NO. ',
1699:       & PACKET(1), ' BUT STORED TYPE IS ', ITYP,
1700:       & ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1701: ccc   PACKET(LPCUR) = PACKET(LL+1)
1702:       PACKET(LPCUR) = PACKET(LL+2)
1703:       PACKET(4) = PACKET(4) + 1
1704:       return
1705:     end if
1706: C
1707: ccc   NDATA = LH1 - ITYP*2**24
1708:       NDATA = PACKET(LL+1)
1709:       LPT = LL + NPHEAD
1710: C
1711:       if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
1712:       if ( MOD(LPT,MWORD).ne.1 ) LPT = LPT + 1
1713:       LPT = LPT/MWORD + MWORD - 1
1714:     end if
1715: C
1716: ccc   PACKET(LPCUR) = PACKET(LL+1)
1717:       PACKET(LPCUR) = PACKET(LL+2)
1718:       PACKET(4) = PACKET(4) + 1
1719:       IRET = 0
1720:       return
1721: C
1722:     end
1723: C
1724: C
1725: C-----
1726: C
1727:       subroutine PCTDMP( IPR, PACKET,RPCT, DPCT )
1728: C=====
1729: C   dump data packet container
1730: C=====
1731:       integer PACKET(*)
1732: C
1733: C   ... rpct & dpct must be equivalent to packet.
1734: C
1735:       real RPCT(*)
1736:       real*8 DPCT(*)
1737: C
1738:       character*4 TYPE
1739:       character*512 CBUFF
1740: C
1741:       write(IPR,'(//1x,a//)') '***** DATA PACKET CONTAINER DUMP *****'
1742: C
1743:       call PCTRW( PACKET, 'START', IRET )
1744: C
1745:       IP = 0
1746: C
1747: 100 TYPE = ' '
1748:       call PCTCHK( PACKET, 'CHECK', TYPE, LPT, NDATA, IRET )
1749: C
1750:       if ( TYPE.eq.' ' ) go to 120
1751: C
1752:       IP = IP + 1
1753: C
1754:       write(IPR,'(/a,i4,3a)') '-- packet # ',IP,' --- type <',TYPE,>'
1755:       write(IPR,*) ' pointer ', LPT, ' number of data ', NDATA

```

```

1756:
1757: C
1758:       if ( TYPE.eq.'LB' .or. TYPE.eq.'CH' ) then
1759: C
1760:       if ( NDATA.gt.LEN(CBUFF) ) then
1761:       write(IPR,*) ' (character length ', NDATA, ' is longer ',
1762:       & 'than buffer size. output truncated.)'
1763:     end if
1764: C
1765:       if ( TYPE.eq.'CH' ) then
1766:       call UNPKCS( PACKET, 'CH', CBUFF, NDATA, IRET )
1767:     else
1768:       call UNPKPT( PACKET, 'LB', TYPE, LPT, NDATA, IRET )
1769:       NDATA = MIN(NDATA,LEN(CBUFF))
1770:       call MVITOC( NDATA, CBUFF, PACKET(LPT) )
1771:     end if
1772: C
1773:       do 110 I = 1, NDATA, 64
1774:       I2 = MIN(I+63,NDATA)
1775: C
1776:       write(IPR,'(1x,i3,''. ''',i3,''' [ ''',a,''' ]'')') I, I2,
1777:       & CBUFF(I:I2)
1778: 110   continue
1779: C
1780:       else if ( TYPE.eq.'I4' ) then
1781: C
1782:       call UNPKPT( PACKET, 'INT4', 'I4', LP, ND, IRET )
1783:       write(IPR,'(1x,5x,9i7)') (PACKET(LP+I),I=0,ND-1)
1784: C
1785:       else if ( TYPE.eq.'R4' ) then
1786: C
1787:       call UNPKPT( PACKET, 'REAL4', 'R4', LP, ND, IRET )
1788:       write(IPR,'(1x,1p,5e14.5)') (RPCT(LP+I),I=0,ND-1)
1789: C
1790:       else if ( TYPE.eq.'R8' ) then
1791: C
1792:       call UNPKPT( PACKET, 'REAL8', 'R8', LP, ND, IRET )
1793:       write(IPR,'(1x,1p,5E14.5)') (DPCT(LP+I),I=0,ND-1)
1794: C
1795:     end if
1796: C
1797:       go to 100
1798: C
1799: 120 write(IPR,'(/1x,a//)') '***** END OF DATA PACKET CONTAINER *****'
1800:       return
1801:     end

```

src/shared/pcuml2.f

```
1:      subroutine PCUML2( P,      N1,      N2 )
2: C=====
3: C  PURPOSE: MAKE CUMULATIVE DISTRIBUTION
4: C  CALLED IN: (NONE)
5: C-----
6: C arguments (i=input, o=output, w=work)
7: C io P(N1,N2) : two dimensional parobability array.
8: C              converted to cumulative distribution in first dimension
9: C              for each second dimension indices.
10: C i N1, N2 : array dimension size
11: C=====
12:      real P(N1,N2)
13:      do 120 N = 1, N2
14:          do 100 I = 2, N1
15:              P(I,N) = P(I-1,N) + P(I,N)
16:          100 continue
17:          do 110 I = 1, N1
18:              P(I,N) = P(I,N) / P(N1,N)
19:          110 continue
20:      120 continue
21:      return
22:      end
```


src/shared/pesum.f

```
1:      subroutine PESUM4( D,      N )
2: C==<MVP/GMVP>=====
3: C Purpose: take summation of D(1:N) on all PE's of FACOM VPP.
4: C      For Real*4 array.
5: C UPDATE:
6: C=====
7: C/#IF PARA(VPP)
8:      include 'INC/_VPPPARA'
9: C
10:      parameter( LIM = 100000 )
11:      real D(N), TMP(LIM)
12: c
13: !xocl processor p( mpe )
14: !xocl subprocessor sp=p(1:mpe)
15: c
16: c      write(6,*) '@@@@@ n :', n
17: c
18:      NN      = (N+LIM-1) /LIM
19:      do 130 K = 1, NN
20:          K1      = (K-1)*LIM
21:          K2      = MIN(K*LIM,N)
22:          KK      = K2 - K1
23: c
24: !xocl spread do
25:      do 110 I = 1, MPE
26:          do 100 J = 1, KK
27:              TMP(J) = D(J+K1)
28:          100      continue
29:          110      continue
30: !xocl end spread sum(tmp)
31: c
32:      do 120 J = 1, KK
33:          D(J+K1) = TMP(J)
34:      120      continue
35: c
36:      130 continue
37: c
38: C/#ENDIF
39:      return
40:      end
41: c
42:      subroutine PESUM8( D,      N )
43: C==<MVP/GMVP>=====
44: C Purpose: take summation of D(1:N) on all PE's of FACOM VPP.
45: C      For Real*8 array.
46: C UPDATE:
47: C=====
48: C
49: C/#IF PARA(VPP)
50:      include 'INC/_VPPPARA'
51: C
52:      parameter( LIM = 100000 )
53:      real*8 D(N), TMP(LIM)
54: c
55: !xocl processor p( mpe )
56: !xocl subprocessor sp=p(1:mpe)
57: c
58: c      write(6,*) '@@@@@ n :', n
59: c
60:      NN      = (N+LIM-1) /LIM
61:      do 130 K = 1, NN
62:          K1      = (K-1)*LIM
63:          K2      = MIN(K*LIM,N)
64:          KK      = K2 - K1
65: c
```

```
66: !xocl spread do
67:      do 110 I = 1, MPE
68:          do 100 J = 1, KK
69:              TMP(J) = D(J+K1)
70:          100      continue
71:          110      continue
72: !xocl end spread sum(tmp)
73: c
74:      do 120 J = 1, KK
75:          D(J+K1) = TMP(J)
76:      120      continue
77: c
78:      130 continue
79: c
80: C/#ENDIF
81:      return
82:      end
83: c
84:      subroutine PESUMI( D,      N )
85: C==<MVP/GMVP>=====
86: C Purpose: take summation of D(1:N) on all PE's of FACOM VPP.
87: C      For INTEGER array.
88: C UPDATE:
89: C=====
90: C
91: C/#IF PARA(VPP)
92:      include 'INC/_VPPPARA'
93: C
94:      parameter( LIM = 100000 )
95:      integer D(N), TMP(LIM)
96: c
97: !xocl processor p( mpe )
98: !xocl subprocessor sp=p(1:mpe)
99: c
100: c      write(6,*) '@@@@@ n :', n
101: c
102:      NN      = (N+LIM-1) /LIM
103:      do 130 K = 1, NN
104:          K1      = (K-1)*LIM
105:          K2      = MIN(K*LIM,N)
106:          KK      = K2 - K1
107: c
108: !xocl spread do
109:      do 110 I = 1, MPE
110:          do 100 J = 1, KK
111:              TMP(J) = D(J+K1)
112:          100      continue
113:          110      continue
114: !xocl end spread sum(tmp)
115: c
116:      do 120 J = 1, KK
117:          D(J+K1) = TMP(J)
118:      120      continue
119: c
120:      130 continue
121: c
122: C/#ENDIF
123:      return
124:      end
125: c
126:      subroutine PESUMI8( D, N )
127: C==<MVP/GMVP>=====
128: C Purpose: take summation of D(1:N) on all PE's of FACOM VPP.
129: C      For INTEGER array.
130: C=====
```

src/shared/pesum.f

```
131: C
132: C/#IF PARA(VPP)
133:   include 'INC/_VPPPARAM'
134: C
135:   parameter( LIM = 100000 )
136: C/#IF INTEGER8
137: *   integer*8 D(N), TMP(LIM)
138: C/#ELSE
139:   integer D(N), TMP(LIM)
140: C/#ENDIF
141:
142: !xocl processor p( mpe )
143: !xocl subprocessor sp=p(1:mpe)
144:   NN = (N+LIM-1) /LIM
145:   do 130 K = 1, NN
146:     K1 = (K-1)*LIM
147:     K2 = MIN(K*LIM,N)
148:     KK = K2 - K1
149:   c
150: !xocl spread do
151:     do 110 I = 1, MPE
152:       do 100 J = 1, KK
153:         TMP(J) = D(J+K1)
154:       100 continue
155:     110 continue
156: !xocl end spread sum(tmp)
157: c
158:     do 120 J = 1, KK
159:       D(J+K1) = TMP(J)
160:     120 continue
161:   130 continue
162: C/#ENDIF
163:   return
164: end
165:
```

src/shared/picasso.f

```

1:      subroutine PICASSO( NPIC, NDMAX, NDOT, MAXMZ, JSIMP, JLATT,
2:      &                    JTLLT, JVMNT, NZONE, NCELL, SDA, IDCEL,
3:      &                    IPCEL, KCELL, KINPZ, KMAT, KZREG, KZDA, NSDA,
4:      &                    NZDA, KZAA, PAPER, X, Y, Z, KZM,
5:      &                    IRG, IMT, IRMWK, MULTZ, IFI, IFL )
6:      C=====
7:      C <MVP/GMVP>
8:      C PURPOSE:  DRAWING PICTURES.
9:      C CALLED IN:  PABLO
10:     C CALLS:    JUDGE
11:     C=====
12:
13:     real*8 X(NDMAX), Y(NDMAX), Z(NDMAX), SDA(*), DX, DY
14:     real PAPER(12,NPIC)
15:     integer KINPZ(*), KMAT(*), KZREG(*), KZDA(2,*), KZAA(*), KCELL(*)
16:     integer IDCEL(NCELL), IPCEL(NCELL+1)
17:     C
18:     C ... working area .....
19:     C
20:     integer KZM(NDMAX), IRG(NDMAX), IMT(NDMAX), IRMWK(NDMAX)
21:     integer MULTZ(MAXMZ,5)
22:     logical IFI(NDMAX), IFL(NDMAX)
23:     C
24:     C
25:     C ==== region symbols ===
26:     C
27:     C REGSYM(1:rsp) is for special materials
28:     C
29:     integer RSP
30:     parameter( RSP = 3 )
31:     character*64 REGSYM
32:     C
33:     C ==== material symbols ===
34:     C
35:     C MATSYM(1:msp) is for special materials
36:     C
37:     parameter( MSP = 8 )
38:     character*68 MATSYM
39:     C
40:     C
41:     character*130 SCALE1, SCALE2, LINE
42:     C
43:     include 'INC/_IOUNIT'
44:     C
45:     parameter( JUNK = -99999 )
46:     C
47:     C ... local variable ..
48:     C
49:     integer KTEMP(3,8)
50:     C
51:     include 'INC/_PMLATT'
52:     include 'INC/_SFLATT'
53:     C
54:     C-----
55:     C
56:     REGSYM = '?* 123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'//
57:     & 'abcdefghijklmnopqrstuvwxyz'
58:     MATSYM = '-?*%< .#123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'//
59:     & 'abcdefghijklmnopqrstuvwxyz'
60:     C
61:     do 480 N = 1, NPIC
62:     C
63:         IX      = PAPER(10,N)
64:         IY      = PAPER(11,N)
65:     C

```

```

66:     do 100 K = 1, IX*IY
67:         KZM(K) = 0
68:         IRG(K) = JUNK
69:         IMT(K) = JUNK
70:     100 continue
71:     C
72:     MJUNK = 2*JUNK
73:     C
74:     do 120 J = 1, 5
75:         do 110 I = 1, MAXMZ
76:             MULTZ(I,J) = 0
77:         110 continue
78:     120 continue
79:     C
80:     do 140 J = 1, IY
81:         DY = (REAL(J)-0.5) /IY
82:         LL = (J-1)*IX
83:         do 130 I = 1, IX
84:             DX = (REAL(I)-0.5) /IX
85:             X(LL+I) = PAPER(1,N) + DX*PAPER(4,N) + DY*PAPER(7,N)
86:             Y(LL+I) = PAPER(2,N) + DX*PAPER(5,N) + DY*PAPER(8,N)
87:             Z(LL+I) = PAPER(3,N) + DX*PAPER(6,N) + DY*PAPER(9,N)
88:         130 continue
89:     140 continue
90:     C
91:     C ... set range of zone search: kz1,kz2
92:     C
93:     C PAPER(12,n) is cell ID ( 0 means non-cell zones )
94:     C
95:     KZ1 = 1
96:     KZ2 = NZONE
97:     C
98:     ICELL = PAPER(12,N)
99:     IICELL = 0
100:    C
101:    if ( JLATT.ne.0.and.NCELL.gt.0 ) then
102:        if ( ICELL.ne.0 ) then
103:            do 150 K = 1, NCELL
104:                if ( IDCEL(K).eq.ICELL ) then
105:                    IICELL = K
106:                    KZ1 = IPCEL(K)
107:                    KZ2 = IPCEL(K+1) - 1
108:                    go to 160
109:                end if
110:            150 continue
111:        C
112:        write(IMSG, '( /lx,a,i5,a/lx,a,i3,a/ )' )
113:        & '!!! a cell having ', ICELL,
114:        & ' as cell-ID does not exist!',
115:        & ' Skip drawing of ', N, 'th' picture.'
116:        go to 480
117:    C
118:    160 continue
119:    else
120:        KZ1 = 1
121:        KZ2 = IPCEL(1) - 1
122:    end if
123:    end if
124:    C
125:    C =====
126:    C ..... ZONE # .....
127:    C =====
128:    C
129:    MMZ = 0
130:    INX = IX*IY

```

src/shared/picasso.f

```

131: C
132:       do 180 KZ = KZ1, KZ2
133: C
134: C .....
135: C
136:       call JUDGE( 'PICASO', KZ, INX, X, Y, Z, IFI, SDA, KZDA,
137: &               KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
138: &               .false., DUMMY, DUMMY, DUMMY )
139: C
140: C .....
141: C
142:       do 170 I = 1, INX
143:       if ( IFI(I) ) then
144: C
145: C region determination logic here is not valid for lattice geometry
146: C
147:       if ( KZM(I).eq.0 ) then
148:         KZM(I) = KZ
149:         IRG(I) = KZREG(KZ)
150:         IMT(I) = KMAT(KINPZ(KZ))
151:       else
152: C
153: C ... check multi definition of region/material ....
154: C
155:         IIREG = KZREG(KZ)
156:         IIMAT = KMAT(KINPZ(KZ))
157:         JJJJ = 0
158: C
159: C ... overlapping lattice zone is checked in any case
160: C
161: CCCCCC      if ( IMT(I).lt.0.and.IMT(I).gt.-999 ) then
162: C            if ( ISLATT(IMT(I)) ) then
163: C              JJJJ = 2
164: C
165: C ... multi definition of region or material ...
166: C
167:       else if ( IIREG.ne.IRG(I) .or. IIMAT.ne.IMT(I) )
168:       &       then
169:         JJJJ = 1
170: CCCCC      if ( IIMAT.lt.0.and.IIMAT.gt.-999 ) JJJJ = 3
171: C            if ( ISLATT(IIMAT) ) JJJJ = 3
172:       end if
173: C
174: C ... memorize suspicious/invalid multiple zone definition ....
175: C ... point # & zone #
176: C
177:       if ( JJJJ.ne.0 ) then
178:         MMZ = MMZ + 1
179: CCCCCC      if ( MMZ.gt.MAXMZ ) then
180: CCCCCC      if ( MMZ.eq.MAXMZ+1 ) write(IPR,7099)
181: CCCCCC      else
182: C            if ( MMZ.le.MAXMZ ) then
183: C              MULTZ(MMZ,1) = I
184: C              MULTZ(MMZ,2) = KZ
185: C              MULTZ(MMZ,3) = IIREG
186: C              MULTZ(MMZ,4) = IIMAT
187: C
188: C ... jjjj = 1 : overlapping of an ordinary zone/region/material
189: C ... jjjj = 2 : overlapping of two lattice zones
190: C ... jjjj = 3 : overlapping of a lattice zone and an ordinary zone
191: C
192:       MULTZ(MMZ,5) = JJJJ
193:       end if
194:     end if
195:   end if

```

```

196:       end if
197:       continue
198:       180 continue
199: C
200: 7000 format(2X,'!!! TOO MANY MULTIPLE DEFINITION. CANNOT TRACE',
201: &          ' ALL ERRORS EXACTLY.'/)
202: C
203: C =====
204: C ..... REGION MAP .....
205: C =====
206: C
207:       NUNDEF = 0
208:       NWDEF = 0
209: C
210:       MAXRG = 0
211: C
212:       do 220 J = 1, IY
213:       do 210 I = 1, IX
214:         II = (J-1)*IX + I
215: C
216: C .... REGION MARKER: -2 = NOT DEFINED, -1 = multiply DEFINED
217: C
218:         IRMWK(II) = IRG(II)
219: C
220:         if ( IRG(II).eq.JUNK ) then
221:           NUNDEF = NUNDEF + 1
222:           IRMWK(II) = -2
223:         else
224:           do 190 K = 1, MIN(MMZ,MAXMZ)
225:           if ( MULTZ(K,1).eq.II.and.MULTZ(K,3).ne.IRG(II) )
226:           &       then
227:             IRMWK(II) = -1
228:             NWDEF = NWDEF + 1
229:             go to 200
230:           end if
231:           190 continue
232: C
233:           200 continue
234:           MAXRG = MAX(MAXRG,IRG(II))
235:         end if
236:       210 continue
237:       220 continue
238: C
239:       NPAGE = (IX+NDOT-1) /NDOT
240: C
241:       do 290 K = 1, NPAGE
242:       KX1 = (K-1)*NDOT + 1
243:       KX2 = MIN(K*NDOT,IX)
244: C
245:       if ( ICELL.eq.0 ) then
246:         write(IPR,7020) N, 'REGION NUMBER', K, KX1, KX2
247:       else
248:         write(IPR,7040) N, 'REGION NUMBER', ICELL, K, KX1, KX2
249:       end if
250: C
251:       write(IPR,7080) (PAPER(I,N),I=1,9), NINT(PAPER(10,N)),
252:       &       NINT(PAPER(11,N))
253: C
254: 7020 format('1 == PICTURE ',I3,' ( ',A,' ) == ', PART ',I2,
255: &          ' : X-DIVISION ',I4,' TO ',I4)
256: C
257: 7040 format('1 == PICTURE ',I3,' ( ',A,' ) == CELL ID ',I5,
258: &          ' : PART ',I2,' : X-DIVISION ',I4,' TO ',I4)
259: C
260:       if ( JTLT.ne.0.and.ICELL.ne.0 ) write(IPR,7060)

```

src/shared/picaso.f

```

261: 7060      format(7X,'<< REGION SYMBOLS IN THIS PAGE DOES NOT',
262:      &      ' SHOW REGION #'S, BECAUSE "FRAME-DEPENDENT-TALLY"'
263:      &      ', OPTION IS USED.',
264:      &      ' ONLY TO CHECK MULTIPLE- OR MISS-DEFINITION.>>')
265:
266: 7080      format(/1P,7X,' ORIGIN = (',E13.5,',',E13.5,',',E13.5,',')/7X,
267:      &      ' X-AXIS = (',E13.5,',',E13.5,',',E13.5,',')5X,
268:      &      ' Y-AXIS = (',E13.5,',',E13.5,',',E13.5,',')/7X,
269:      &      ' X-DIVISION = ',I4,', Y-DIVISION = ',I4)
270:      write(IPR,7100) X(KX1), Y(KX1), Z(KX1), X(KX2), Y(KX2),
271:      &      Z(KX2), X((IY-1)*IX+KX1), Y((IY-1)*IX+KX1),
272:      &      Z((IY-1)*IX+KX1), X((IY-1)*IX+KX2),
273:      &      Y((IY-1)*IX+KX2), Z((IY-1)*IX+KX2)
274: 7100      format(7X,' UPPER-LEFT CORNER (',1P,2(E12.5,','),E12.5,',')',
275:      &      ' 3X,' UPPER-RIGHT CORNER (',1P,2(E12.5,','),E12.5,',')',
276:      &      '/7X,' LOWER-LEFT CORNER (',1P,2(E12.5,','),E12.5,',')',
277:      &      ' 3X,' LOWER-RIGHT CORNER (',1P,2(E12.5,','),E12.5,',')',
278:      &      ' E12.5,',')')
279:      write(IPR,7120) REGSYM(2:2), REGSYM(1:1)
280: 7120      format(/7X,A1,' : REGION MULTIPLY DEFINED, ',A1,
281:      &      ' : REGION UNDEFINED, ',
282:      &      ' BLANK : LATTICE OR UNTALLIED ZONE.'/)
283: C
284:      MINRG = 999999
285:      MAXRG = -999999
286:
287:      do 230 I = 1, IX*Y
288:      if ( IRMWK(I).gt.0 ) then
289:          MINRG = MIN(MINRG,IRMWK(I))
290:          MAXRG = MAX(MAXRG,IRMWK(I))
291:      end if
292: 230      continue
293:
294:      do 250 M = MINRG, MAXRG, 20
295:          M2 = MIN(MAXRG,M+19)
296:          LINE = ' '
297:          write(LINE,'(7x,'REGION ',i5,' to ',i5,' #'')' ) M,
298:          &          M2
299:          MM = INDEX(LINE,'#')
300:
301:          do 240 I = M, M2
302:              IP = MOD(I-1,LEN(REGSYM)-RSP) + 1 + RSP
303:              LINE(MM+4*(I-M):MM+4*(I-M)+3) = ' < '//REGSYM(IP:IP)
304:              &              //'>'
305: 240          continue
306:
307:          write(IPR,'(a)') LINE
308: 250          continue
309: C
310:          SCALE1 = ' '
311:          SCALE2 = ' '
312:          MRGIN = 6
313:          SCALE2(MRGIN:MRGIN) = ':'
314:
315:          I1IK1 = 1
316:          II = 1
317:          do 260 I = KX1, KX2
318:              II = I - KX1 + 1 + MRGIN
319:              SCALE2(II:II) = '-'
320:              if ( MOD(I,5).eq.0 ) SCALE2(II:II) = '+'
321:              if ( MOD(I,10).eq.0 ) then
322:                  SCALE2(II:II) = ':'
323:                  write(SCALE1(II-2:II+1),'(I4)') I
324:                  I1IK1 = II + 1
325:              end if

```

```

326: 260      continue
327:
328:          I1IK2 = II
329:
330:          if ( KX2.eq.IX ) then
331:              SCALE2(II+1:II+1) = ':'
332:              I1IK2 = II + 1
333:          end if
334:
335:          write(IPR,'(A)') SCALE1(:I1IK1)
336:          write(IPR,'(A)') SCALE2(:I1IK2)
337: C
338: C      .... PRINT REGION MAP ....
339: C
340:          do 280 J = IY, 1, -1
341:              LL = (J-1)*IX
342:              LINE = ' '
343:              if ( MOD(J,10).eq.0 ) then
344:                  write(LINE(1:3),'(i3)') J
345:                  LINE(4:5) = '-:'
346:              else
347:                  LINE(5:5) = ':'
348:              end if
349:
350: C          .... map starts from IL+1'th column.
351: C
352:              IL = 5
353: C
354:              do 270 I = KX1, KX2
355:                  IL = IL + 1
356: C
357: C          .... region undefined or multiply defined
358: C          .... region # 0 remains as blank ...
359: C
360:              if ( IRMWK(LL+I).lt.0 ) then
361:                  IP = IRMWK(LL+I) + RSP
362:                  LINE(IL:IL) = REGSYM(IP:IP)
363:              else if ( IRMWK(LL+I).gt.0 ) then
364:                  IP = MOD(IRMWK(LL+I)-1,LEN(REGSYM)-RSP) + 1
365:                  &                  + RSP
366:                  LINE(IL:IL) = REGSYM(IP:IP)
367:              end if
368: 270          continue
369:          if ( KX2.eq.IX ) then
370:              IL = IL + 1
371:              LINE(IL:IL) = ':'
372:          end if
373:          write(IPR,'(1x,a)') LINE(:IL)
374: C
375: C      CCCC      .... UNDERLINED NUMBER FOR REGIONS WHOSE NUMBER ARE GREATER
376: C      CC          THAN 50 .
377: C      CCC      ..... NOT YET AVAILABLE (1989 8/29 )
378: C
379: 280          continue
380:          write(IPR,'(A)') SCALE2(:I1IK2)
381:          write(IPR,'(A)') SCALE1(:I1IK1)
382: 290          continue
383: C
384:          if ( NUNDEF.gt.0.and.JLATT.eq.0 ) then
385:              write(IMG,7140)
386: 7140          format(/1x,'!!! REGION (ZONE) UNDEFINED ON SOME POINTS!!!'
387:          &          ' I HOPE THOSE ARE UNREACHABLE POINTS.')
388:          &          call CNTERR('WARNING')
389:          end if
390: C

```

src/shared/picaso.f

```

391:         if ( NWDEF.gt.0 ) then
392:             write(IMG,7160)
393: 7160         format(/lx,'!!! REGION MULTIPLY-DEFINED ON SOME POINTS!!!'
394:             &          ' I HOPE THOSE ARE INTENTIONAL.'/)
395:             call CNTERR( 'WARNING' )
396:             if ( MMZ.gt.MAXMZ ) write(IMG,7000)
397:             call CNTERR( 'WARNING' )
398: C
399:             do 310 II = 1, IX*IY
400:
401:                 if ( IRMWK(II).eq.-1 ) then
402:                     write(IPR,7180) 'REGION MULTIPLY DEFINED', X(II),
403:                         &          Y(II), Z(II), MOD(II,IX) + 1, II/IX + 1
404:                     write(IPR,7200) 'REGION', IRG(II), KZM(II)
405:                     do 300 K = 1, MIN(MMZ,MAXMZ)
406:                         if ( MULTZ(K,1).eq.II.and.MULTZ(K,3).ne.IRG(II) )
407:                             &          write(IPR,7200) 'REGION', MULTZ(K,3), MULTZ(K,2)
408:                         continue
409:                     end if
410:                 310             continue
411: C
412:             end if
413: C
414: 7180         format(5X,A,' AT (x,y,z) = (',1P,3E12.5,') (i,j) = (',2I4,')'
415:             &          )
416: 7200         format(9X,A,2X,I6,4,' (ZONE ',I6,')')
417: C
418: C =====
419: C ..... MATERIAL MAP.....
420: C =====
421: C
422:         NUNDEF = 0
423:         NWDEF = 0
424: C
425:         do 350 J = 1, IY
426:             do 340 I = 1, IX
427:                 II = (J-1)*IX + I
428:                 KZ = KZM(II)
429: C
430: C ..... MARKER FOR CHARACTER-PRINT-OUT.
431: C         -7 = outer region of cell ( mat = -999 )
432: C         -6 = NOT DEFINED, -5= MULTIPLY DEFINED
433: C         -4,-3 = REFLECTOR, -2 = OUTER VOID -1 = INNER VOID
434: C         0 = LATTICE
435: C
436:             if ( IMT(II).eq.JUNK ) then
437:                 IRMWK(II) = -6
438:                 NUNDEF = NUNDEF + 1
439:             else
440:                 JWDEF = 0
441:                 do 320 K = 1, MIN(MMZ,MAXMZ)
442:                     if ( MULTZ(K,1).eq.II
443:                         &          .and.
444:                         &          (MULTZ(K,4).ne.IMT(II).or.MULTZ(K,5).eq.2
445:                         &          .or.MULTZ(K,5).eq.3) ) then
446:                         IRMWK(II) = -5
447:                         NWDEF = NWDEF + 1
448:                         JWDEF = 1
449:                         go to 330
450:                     end if
451:                 320             continue
452: C
453:                 330             continue
454: C
455:             if ( JWDEF.eq.0 ) then

```

```

456:
457:             if ( IMT(II).eq.-3000 ) then
458:                 IRMWK(II) = -4
459:             else if ( IMT(II).eq.-2000 ) then
460:                 IRMWK(II) = -3
461:             else if ( IMT(II).eq.-1000 ) then
462:                 IRMWK(II) = -2
463:             else if ( IMT(II).eq.0 ) then
464:                 IRMWK(II) = -1
465: CCCCC
466:             else if ( IMT(II).le.-1.and.IMT(II).gt.-999 ) then
467:                 IRMWK(II) = -0
468:             else if ( IMT(II).eq.-999 ) then
469:                 IRMWK(II) = -7
470:             else
471:                 IRMWK(II) = IMT(II)
472:             end if
473:
474:             end if
475:             end if
476:             340             continue
477:             350             continue
478: C
479:             NPAGE = (IX+NDOT-1) /NDOT
480: C
481:             do 420 K = 1, NPAGE
482:                 KX1 = (K-1)*NDOT + 1
483:                 KX2 = MIN(K*NDOT,IX)
484:
485:                 if ( ICELL.eq.0 ) then
486:                     write(IPR,7020) N, 'MATERIAL NUMBER', K, KX1, KX2
487:                 else
488:                     write(IPR,7040) N, 'MATERIAL NUMBER', ICELL, K, KX1, KX2
489:                 end if
490:
491:                 write(IPR,7080) (PAPER(I,N),I=1,9), NINT(PAPER(10,N)),
492:                     &          NINT(PAPER(11,N))
493: C
494: C         WRITE(IPR,7020) MATSYM(2:2), MATSYM(1:1), MATSYM(3:3),
495: C         &          MATSYM(4:4), MATSYM(6:6),
496: C         &          MATSYM(7:7)
497:
498:                 write(IPR,7220) MATSYM(3:3), MATSYM(2:2), MATSYM(4:4),
499:                     &          MATSYM(5:5), MATSYM(7:7), MATSYM(8:8), MATSYM(1:1)
500: 7220             format(/8X,A1,' : MATERIAL MULTIPLY DEFINED, ',A1,
501:                 &          ' : MATERIAL UNDEFINED, ',A1,
502:                 &          ' : WHITE REFLECTOR, ',A1,' : PERFECT REFLECTOR '
503:                 &          /8X,' BLANK : OUTER VOID, ',A1,' : INNER VOID,',
504:                 &          ' ',A1,' : LATTICE, ',A1,' : OUT-OF CELL.'/)
505: C
506:             MINMAT = 999999
507:             MAXMAT = -999999
508:
509:             do 360 I = 1, IX*IY
510:                 if ( IRMWK(I).gt.0 ) then
511:                     MINMAT = MIN(MINMAT,IRMWK(I))
512:                     MAXMAT = MAX(MAXMAT,IRMWK(I))
513:                 end if
514:             360             continue
515:
516:             do 380 M = MINMAT, MAXMAT, 20
517:                 M2 = MIN(MAXMAT,M+19)
518:                 LINE = ' '
519:                 write(LINE,'(7x,'MATERIAL ',i5,' to ',i5,' : #')')
520:                 &          M, M2

```

src/shared/picasso.f

```

521:
522:      MM      = INDEX(LINE,'#')
523:      do 370 I = M, M2
524:      IP      = MOD(I-1,LEN(MATSYM)-MSP) + 1 + MSP
525:      LINE(MM+4*(I-M):MM+4*(I-M)+3) = ' <'//MATSYM(IP:IP)
526:      &      //>'
527: 370      continue
528:      write(IPR,'(a)') LINE
529: 380      continue
530: C
531:      SCALE1 = ' '
532:      SCALE2 = ' '
533:      MRGIN  = 6
534:      SCALE2(MRGIN:MRGIN) = ':'
535:
536:      I1K1 = 1
537:      II   = 1
538:
539:      do 390 I = KX1, KX2
540:      II = I - KX1 + 1 + MRGIN
541:      SCALE2(II:II) = '-'
542:      if ( MOD(I,5).eq.0 ) SCALE2(II:II) = '+'
543:      if ( MOD(I,10).eq.0 ) then
544:      SCALE2(II:II) = ':'
545:      write(SCALE1(II-2:II+1),'(14)') I
546:      I1K1 = II + 1
547:      end if
548: 390      continue
549:
550:      I1K2 = II
551:      if ( KX2.eq.IX ) then
552:      SCALE2(II+1:II+1) = ':'
553:      I1K2 = II + 1
554:      end if
555:      write(IPR,'(/A)') SCALE1(:I1K1)
556:      write(IPR,'(A)') SCALE2(:I1K2)
557: C
558: C      .... PRINT MATERIAL MAP ....
559: C
560:      do 410 J = IY, 1, -1
561:      LL = (J-1)*IX
562:      LINE = ' '
563:      if ( MOD(J,10).eq.0 ) then
564:      write(LINE(1:3),'(i3)') J
565:      LINE(4:5) = '-:'
566:      else
567:      LINE(5:5) = ':'
568:      end if
569:
570: C      .... map starts from IL+1'th column.
571:
572:      IL = 5
573: C
574:      do 400 I = KX1, KX2
575:      IL = IL + 1
576: C
577: C      .... material undefined or multiply defined
578: C
579:      if ( IRMWK(LL+I).le.0 ) then
580:      IP = IRMWK(LL+I) + MSP
581:      LINE(IL:IL) = MATSYM(IP:IP)
582: C
583: C      .... normal
584: C
585:      else

```

```

586:      IP = MOD(IRMWK(LL+I)-1,LEN(MATSYM)-MSP) + 1
587:      &      + MSP
588:      LINE(IL:IL) = MATSYM(IP:IP)
589:      end if
590: 400      continue
591:      if ( KX2.eq.IX ) then
592:      IL = IL + 1
593:      LINE(IL:IL) = ':'
594:      end if
595:      write(IPR,'(1x,a)') LINE(:IL)
596: 410      continue
597:      write(IPR,'(A)') SCALE2(:I1K2)
598:      write(IPR,'(A)') SCALE1(:I1K1)
599: 420      continue
600: C
601:      if ( NUNDEF.gt.0.and.JLATT.eq.0 ) then
602:      write(IMG,7240)
603:      format(/1x,' !!! MATERIAL NOT DEFINED AT SOME POINTS !'/1x,
604:      &      ' I HOPE THOSE ARE INTENTIONAL.'/)
605:      call CNTERR( 'WARNING' )
606:
607:      do 450 II = 1, IX*IX
608:
609:      if ( IRMWK(II).eq.-6 ) then
610:      III = MOD(II,IX) + 1
611:      JJJ = II/IX + 1
612:      write(IPR,7180) 'MATERIAL NOT DEFINED', X(II), Y(II),
613:      &      Z(II), III, JJJ
614: C
615: C      ... print zones of neighbouring points
616: C
617:      NB = 0
618:      do 440 J1 = MAX(JJJ-1,1), MIN(JJJ+1,IX)
619:      do 430 I1 = MAX(III-1,1), MIN(III+1,IX)
620:      if ( I1.ne.III .or. J1.ne.JJJ ) then
621:      KKK = I1 + IX*(J1-1)
622:      KKK = KZM(KKK)
623:
624:      if ( KKK.ne.0 ) then
625:      NB = NB + 1
626:      KTEMP(1,NB) = I1
627:      KTEMP(2,NB) = J1
628:      KTEMP(3,NB) = KKK
629:      end if
630:
631:      end if
632: 430      continue
633: 440      continue
634:
635:      if ( NB.gt.0 ) then
636:      write(IPR,7260)
637:      &      (KTEMP(1,K),KTEMP(2,K),KTEMP(3,K),K=1,NB)
638:      end if
639: 7260      format(1x,' NEIGHBOURING ZONES :',6
640:      &      :('(',I3,',',I3,',',I5,',','))
641:      end if
642: 450      continue
643:      end if
644: C
645:      if ( NWDEF.gt.0 ) then
646:      write(IMG,
647:      &      '(/' XXX MATERIAL MULTIPLY-DEFINED AT SOME POINTS !'))
648:      if ( JLATT.ne.0 ) write(IPR,
649:      &      '(/' (MAY INCLUDE OVERLAPPED DEFINITION OF LATTICE)'))
650:      &

```

src/shared/picaso.f

```
651: C
652:         call CNTERR( 'FATAL' )
653: C
654:         if ( MMZ.gt.MAXMZ ) write(IMG,7000)
655:         call CNTERR( 'WARNING' )
656: C
657:         do 470 II = 1, IX*IY
658:             if ( IRMWK(II).eq.-5 ) then
659:                 if ( JLATT.eq.0 ) then
660:                     write(IPR,7180) 'MATERIAL MULTIPLY DEFINED', X(II),
661:                                     & Y(II), Z(II), MOD(II,IX) + 1, II/IX + 1
662:                 else
663:                     write(IPR,7180)
664:                         & 'MATERIAL MULTIPLY DEFINED OR OVERLAPPING LATTICE',
665:                         & X(II), Y(II), Z(II), MOD(II,IX) + 1, II/IX
666:                         & + 1
667:                 end if
668: C
669:         CCCCCC if ( IMT(II).lt.0.and.IMT(II).gt.-999 ) then
670:             if ( ISLATT(IMT(II)) ) then
671:                 write(IPR,7200) 'LATTICE', IMT(II), K&M(II)
672:             else
673:                 write(IPR,7200) 'MATERIAL', IMT(II), K&M(II)
674:             end if
675: C
676:             do 460 K = 1, MIN(MMZ,MAXMZ)
677:                 if ( MULTZ(K,1).eq.II ) then
678:                     if ( MULTZ(K,5).eq.1.and.MULTZ(K,4).ne.IMT(II) )
679:                         & then
680:                         write(IPR,7200) 'MATERIAL', MULTZ(K,4),
681:                                     & MULTZ(K,2)
682:                     else if ( MULTZ(K,5).eq.2 ) then
683:                         write(IPR,7200) 'OVERLAPPING LATTICE',
684:                                     & MULTZ(K,4), MULTZ(K,2)
685:                     else if ( MULTZ(K,5).eq.3 ) then
686:                         write(IPR,7200) 'OVERLAPPING LATTICE',
687:                                     & MULTZ(K,4), MULTZ(K,2)
688:                     end if
689:                 end if
690:                 460 continue
691:             end if
692:             470 continue
693: C
694:         end if
695: C
696:         480 continue
697: C
698:         7280 format(1X,I3,'-:',130(A1:))
699:         7300 format(1X,3X,' :',130(A1:))
700:         return
701:     end
```


src/shared/pkkai.f

```
1:      subroutine PKKAI( KKAI,  FKAI,  NGROUP,NMAT,  TEMP )
2: C=====
3: C PURPOSE: NORMALIZE FKAI AND MAKE KKAI FOR B.M.C. SAMPLING
4: C CALLED IN:  INTRO
5: C CALLS:  BMCMK1
6: C-----
7: C arguments (i=input, o=output, w=work)
8: C o KKAI(2,NGROUP,NMAT) : B.M.C sampling index for FKAI
9: C o FKAI(NGROUP,NMAT) : converted into B.M.C sampling form
10: C i NGROUP : number of energy groups
11: C i NMAT : number of materials
12: C w TEMP(NGROUP) : working array
13: C=====
14:      integer KKAI(2,NGROUP,NMAT)
15:      real FKAI(NGROUP,NMAT), TEMP(NGROUP)
16:      include 'INC/_IUNIT'
17:      do 120 M = 1, NMAT
18:          SUM = 0.0
19:          do 100 I = 1, NGROUP
20:              SUM = SUM + FKAI(I,M)
21:          100 continue
22:          write(IPR,7000) M, SUM
23:          if ( SUM.ne.0.0 ) then
24:              do 110 I = 1, NGROUP
25:                  FKAI(I,M) = FKAI(I,M) /SUM
26:              110 continue
27:              call BMCMK1( FKAI(1,M), TEMP, KKAI(1,1,M), NGROUP )
28:              call ATRANS( TEMP, FKAI(1,M), NGROUP )
29:          end if
30:      120 continue
31:      7000 format(1X,' == SUMMATION OF FISSION SPECTRUM (MAT=',I3,' ) = ',
32:          &      E12.5)
33:      return
34:      end
```

src/shared/pnorm.f

```
1:      subroutine PNORM( P,      N )
2: C=====
3: C  PURPOSE: NORMALIZE DISTRIBUTION FUNCTION
4: C  CALLED IN: PRSOUR
5: C=====
6:      real P(N)
7:      TP      = 0.0
8:      do 100 I = 1, N
9:          TP      = TP + P(I)
10: 100 continue
11:      do 110 I = 1, N
12:          P(I)      = P(I) /TP
13: 110 continue
14: C
15:      return
16:      end
17: C
18:      subroutine PNORM8( P,      N )
19: C=====
20: C  PURPOSE: NORMALIZE REAL*8 DISTRIBUTION FUNCTION
21: C  CALLED IN: PRSOUR
22: C=====
23:      real*8 P(N), TP
24:      TP      = 0.0
25:      do 100 I = 1, N
26:          TP      = TP + P(I)
27: 100 continue
28:      do 110 I = 1, N
29:          P(I)      = P(I) /TP
30: 110 continue
31: C
32:      return
33:      end
```

src/shared/prein0.f

```

1:      subroutine PREIN0( IPREIN,CSTRNG,NTASK0,TSKINF,MLIMIT )
2: C=<MVP>=====
3: C Purpose: Interpret command string for file-name/I/O-unit coupling
4: C or default-option overriding etc.
5: C This routine calls PREINP routine of GMVP/MVP, and
6: C Probably called from main routine before calling of 'CENTER'
7: C routine, or called before reading input to MVP/GMVP.
8: C
9: C=====
10: C argument :
11: C
12: C i iprein : counter of calling for this routine given externally.
13: C (when iprein = 1 , initialize data )
14: C i cstrng : command string
15: C
16: C /nn[rfu][:file-name]
17: C
18: C Assign I/O unit nn to a file.
19: C
20: C xxxxyyy
21: C
22: C Execution parameter setting.
23: C
24: C @filename
25: C
26: C MS-DOS-like "linkage" file which includes option parameter
27: C strings.
28: C
29: C o ntask0 : number of task effective in multiprocessing mode
30: C
31: C NTASK=... or ntask=...
32: C
33: C o tskinf : name of task control information file in multiprocessing
34: C mode.
35: C
36: C TASKINFO=... or taskinfo=...
37: C
38: C o mlimit : memory size limit (effective only in dynamic allocation
39: C mode )
40: C
41: C=====
42: C
43: C include '../shared/INC/_IOUNIT'
44: C
45: C ... external data ...
46: C
47: C character*(*) CSTRNG
48: C character*(*) TSKINF
49: C
50: C ... local data ...
51: C
52: C character*256 LKFILE
53: C character*256 LINE
54: C logical OPD
55: C
56: C-----
57: C
58: C if ( CSTRNG(1:1).ne.'@' ) then
59: C call PREINP( IPREIN, CSTRNG, NTASK0, TSKINF, MLIMIT )
60: C else
61: C
62: C .... @filename ...
63: C
64: C LKFILE = CSTRNG(2:)
65: C

```

```

66: C ... find unopened I/O unit
67: C
68: C do 100 IIU = 8, 99
69: C inquire(unit =IIU,opened =OPD)
70: C if ( .not.OPD ) go to 110
71: C 100 continue
72: C
73: C write(IMG,*) 'XXX(PREIN0) Cannot find unopened I/O unit for',
74: C & ' "linkage-file" <', LKFILE(:ICLEN(LKFILE)), '>'
75: C stop 999
76: C
77: C 110 open( unit =IIU, file =LKFILE(:ICLEN(LKFILE)), form =
78: C & 'FORMATTED', status = 'OLD', iostat =IOS )
79: C if ( IOS.ne.0 ) then
80: C write(IMG,*) 'XXX(PREIN0) Cannot open "linkage-file" <',
81: C & LKFILE(:ICLEN(LKFILE)), '>'
82: C write(IMG,*) ' Status code from OPEN statement = ', IOS
83: C stop 888
84: C else
85: C write(IPR,*) '=== open "linkage-file" <',
86: C & LKFILE(:ICLEN(LKFILE)), '> on I/O unit ', IIU
87: C end if
88: C
89: C ... input loop from the file
90: C
91: C 120 LINE = ' '
92: C read(IIU,'(a)',end =160) LINE
93: C if( LINE(1:1).eq.'#' ) goto 120
94: C
95: C IS = 1
96: C
97: C ... extract non-blank tokens ...
98: C
99: C 130 continue
100: C if ( IS.le.LEN(LINE) ) then
101: C IS0 = IS
102: C do 140 IS = IS0, LEN(LINE)
103: C if ( LINE(IS:IS).ne.' ' ) go to 150
104: C 140 continue
105: C go to 120
106: C
107: C 150 IE = ICLEN(LINE(IS:)) + IS - 1
108: C -----
109: C call PREINP( IPREIN, LINE(IS:IE), NTASK0, TSKINF, MLIMIT )
110: C -----
111: C IS = IE + 1
112: C go to 130
113: C end if
114: C
115: C 160 continue
116: C
117: C close(IIU)
118: C write(IPR,*) '=== close "linkage-file" <',
119: C & LKFILE(:ICLEN(LKFILE)), '> on I/O unit ', IIU
120: C
121: C end if
122: C
123: C return
124: C end

```

src/shared/presrc.f

```

1:      subroutine PRESRC( PROG, JNEUT, JPHOT, NSOUR,
2:      &                  NGROUP,NGP1,  NGP2,  NTGX,  LSOUR, LKSOUR,
3:      &                  LPSPAC,LPENRG,LIFISM,SOUR,  KSOUR, PSPAC,
4:      &                  PENRG, IFISM, FKAI,  ENGYB, ENGPB, NUCID, NUC,
5:      &                  IDMAT, NMAT,  EINCD, NERR )
6: C=====
7: C purpose: convert old-type of source information to new type
8: C          ( for compatibility with earlier version )
9: C called in : srcinp
10: C calls : rwind ccomp
11: C-----
12: C
13: C arguments ( i = input, o = output )
14: C
15: C*i iugl : output I/O unit of converted information (removed Jan 2000)
16: C*i ipr : message printout I/O unit (removed Jan 2000)
17: C
18: C i prog : program name ( 'MVP' or 'GMVP' )
19: C i jneut : flag for neutron problem
20: C i jphot : flag for photon problem
21: C
22: C io nsour : number of source sets
23: C i ngroup : number of energy group (GMVP) or energy bin of tally(MVP)
24: C i ngp1 : number of neutron energy group (GMVP)
25: C          or neutron energy bin of tally (MVP)
26: C i ngp2 : number of photon energy group (GMVP)
27: C          or photon energy bin of tally (MVP)
28: C i ntgx : number of energy groups in cross section library
29: C          ( prog='GMVP' )
30: C
31: C i lsour : pointer to source set intensity data
32: C i lksour : pointer to source set type data
33: C i lspac : pointer to source specification data
34: C i lpenrg : pointer to source energy distribution data
35: C i lifism : pointer to initial fission source specification data
36: C
37: C i sour(nsour) : source set relative intensity
38: C i ksour(nsour) : source set type
39: C i pspac(10,nsour) : source set specification data
40: C i penrg(ngroup,nsour) : source energy distribution data
41: C i ifism(nsour) : initial fission source specification data
42: C i fkai(ntgx,nmat) : fission spectrum data (prog='GMVP')
43: C i engyb(ngp1+1) : neutron energy boundaries
44: C i engpb(ngp2+1) : photon energy boundaries
45: C
46: C i nucid(nuc) : nuclide name (prog='MVP')
47: C i nuc : number of nuclides (prog='MVP')
48: C i idmat(nmat) : material ID
49: C i nmat : number of materials (prog='GMVP')
50: C
51: C o nerr : number of processing error
52: C
53: C-----
54:      character*(*) PROG
55: C
56:      real*8 SOUR(NSOUR)
57:      integer KSOUR(NSOUR)
58:      real PSPAC(10,NSOUR)
59:      real PENRG(NGROUP,NSOUR)
60:      integer IFISM(NSOUR)
61:      real FKAI(NTGX,NMAT)
62:      real ENGYB(NGP1+1)
63:      real ENGPB(NGP2+1)
64:      real EINCD
65:      integer IDMAT(NUC)

```

```

66:      character*16 NUCID(NUC)
67: C
68: C ... IUG1, IP and IMSEG is passed by include
69: C include 'INC/_IOUNIT'
70: C
71: C --- local variable ---
72: C character*8 TYPE
73: C character*8 PKIND
74: C
75: C-----
76: C
77: C call RWIND( IUG1 )
78: C
79: 7000 format(A)
80: 7020 format(A,A)
81: 7040 format(A,A,A)
82: 7060 format(10(:1X,I6))
83: 7080 format(5(:1X,1P,E13.6))
84: 7100 format(A,A,1PE13.6,A)
85: 7120 format(A,I6,A)
86: C
87: if ( PROG.ne.'GMVP'.and.PROG.ne.'MVP' ) then
88:   write(IMSG,*) 'XXX(PRESRC) Old form MVP/GMVP source input',
89:   & ' conversion to new form is allowed only for MVP and GMVP!!'
90:   NERR = 1
91:   return
92: end if
93: C
94: write(IUG1,7000) '$SOURCE'
95: C
96: NERR = 0
97: C
98: NNSOUR = 0
99: do 140 N = 1, NSOUR
100: C
101:   N1 = NNSOUR + 1
102:   N2 = NNSOUR + 1
103:   P1 = SOUR(N)
104:   P2 = SOUR(N)
105:   IPHOT = 0
106:   INEUT = 0
107:   IFISS = 0
108:   if ( LPENRG.ne.0 ) then
109: C
110: C ... for photon & neutron coupled problem
111: C
112:   if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
113:     PN = 0.0
114:     PP = 0.0
115:     do 100 I = 1, NGROUP
116:       if ( PENRG(I,N).ne.0 ) then
117:         if ( I.le.NGP1 ) then
118:           PN = PN + PENRG(I,N)
119:           INEUT = 1
120:         else
121:           PP = PP + PENRG(I,N)
122:           IPHOT = 1
123:         end if
124:       end if
125:     continue
126:     P1 = P1*PN/(PN+PP)
127:     P2 = P2*PP/(PN+PP)
128: C
129:   else if ( JNEUT.ne.0 ) then
130:     INEUT = 1

```

src/shared/presrc.f

```

131:         else if ( JPHOT.ne.0 ) then
132:             IPHOT = 1
133:         end if
134:         if ( INEUT.ne.0.and.IPHOT.ne.0 ) N2 = N1 + 1
135:     else if ( LIFISM.ne.0 ) then
136:         INEUT = 1
137:         IFISS = 1
138: C
139: C .. FNS DT Neutron source ....
140: C
141:     else if ( KSOUR(N).eq.9 ) then
142:         INEUT = 1
143:     end if
144: C
145: C
146:     do 130 K = N1, N2
147: C
148: C .... '&' + PARTICLE('NEUTRON'/'PHOTON')
149: C
150:         write(IUG1,7000) '&'
151:         IG1 = 0
152:         IG2 = 0
153:         NOFF = 0
154:         if ( IFISS.ne.0 .or. INEUT.ne.0.and.K.eq.N1 ) then
155:             PKIND = 'NEUTRON'
156:             IG1 = 1
157:             IG2 = NGP1
158:             NOFF = 0
159:         else if ( IPHOT.ne.0.and.K.eq.N2 ) then
160:             PKIND = 'PHOTON'
161:             IG1 = 1
162:             if ( INEUT.ne.0 ) IG1 = NGP1 + 1
163:             IG2 = NGROUP
164:             NOFF = 0
165:             if ( INEUT.ne.0 ) NOFF = NGP1
166:         end if
167: CCCCCCCCC write(IUG1,7000) PKIND
168:         write(IUG1,'(a,a,a)' ) PARTICLE(' ',PKIND(:ICLEN2(PKIND)),')')
169: C
170: C .... RATIO(...)
171: C
172:         if ( PKIND.eq.'NEUTRON' ) then
173:             RATIO = P1
174:         else
175:             RATIO = P2
176:         end if
177: C##<2007/03/14:PN3:
178: C## write(IUG1,'('' RATIO('',E12.5,'')'')' ) RATIO
179: C## write(IUG1,'('' RATIO('',1P,E12.5,'')'')' ) RATIO
180: C##>
181: C
182: C .... @E or @G ...
183: C
184:         if ( IFISS.ne.0 ) then
185:             if ( PROG.eq.'MVP' ) then
186:                 L1 = ICLEN(NUCID(IFISM(N)))
187:                 write(IUG1,7100) '@E = #FISSION(' ,
188:                     & NUCID(IFISM(N)) (:L1), EINCD, ' ) ;'
189:             else if ( PROG.eq.'GMVP' ) then
190:                 do 110 IM = 1, NMAT
191:                     if ( IDMAT(IM).eq.IFISM(N) ) go to 120
192: 110 continue
193:                 write(IMG,*) 'XXX(PRESRC) Material ID IFISM = ',
194:                     & IFISM(N), ' is not found.'
195:                 write(IMG,7140) (IDMAT(IM),IM=1,NMAT)

```

```

196: 7140 format(1X,' --- Available Material IDs ---'/(1X,10I8))
197: call CNTERR( 'FATAL' )
198: IM = 1
199: 120 continue
200:
201: CM write(IUG1,7000) '@G = #TABLE INT(DISCRETE) X('
202: CM write(IUG1,7060) (I,I=1,NTGX)
203: CM write(IUG1,7000) ' ) PX('
204: CM write(IUG1,7080) (FKAI(IG,IM),IG=1,NTGX)
205: CM write(IUG1,7000) ' ) ;'
206: CM write(IUG1,7120) '@G = #FISSION(' , IFISM(N), ' ) ;'
207: end if
208: end if
209: C
210: if ( LPENRG.ne.0 ) then
211:     if ( PROG.eq.'GMVP' ) then
212:         write(IUG1,7000) '@G = #TABLE INT(DISCRETE) X('
213:         write(IUG1,7060) (I,I=IG1-NOFF,IG2-NOFF)
214:         write(IUG1,7000) ' ) PX('
215:         write(IUG1,7080) (PENRG(IG,N),IG=IG1,IG2)
216:         write(IUG1,7000) ' ) ;'
217:     else
218:         write(IUG1,7000) '@E = #TABLE INT(LOG-STEP) X('
219:         if ( PKIND.eq.'NEUTRON' )
220:             & write(IUG1,7080) (ENGYB(I),I=1,NGP1+1)
221:         if ( PKIND.eq.'PHOTON' )
222:             & write(IUG1,7080) (ENGPB(I),I=1,NGP2+1)
223:         write(IUG1,7000) ' ) PX('
224:         write(IUG1,7080) (PENRG(IG,N),IG=IG1,IG2)
225:         write(IUG1,7000) ' ) ;'
226:     end if
227: end if
228: C
229: C .... @(X Y Z A B C [E] ) = #TYPEn( .... ) ....
230: C
231: if ( LPSPAC.ne.0.and.LKSOUR.ne.0 ) then
232:     write(TYPE,'(''TYPE'',i3)' ) KSOUR(N)
233:     call CCOMP( TYPE, LEN(TYPE), TYPE, IFL )
234: C
235: C ... type 9 source samples energy ....
236: C
237: if ( KSOUR(N).eq.9 ) then
238:     write(IUG1,7040) '@(X Y Z A B C E) = #', TYPE, '('
239: else
240:     write(IUG1,7040) '@(X Y Z A B C) = #', TYPE, '('
241: end if
242: C
243: write(IUG1,7080) (PSPAC(I,N),I=1,10)
244: write(IUG1,7000) ' ) ;'
245: end if
246: 130 continue
247: NNSOUR = N2
248: 140 continue
249: C
250: write(IUG1,7000) '$END SOURCE'
251: C
252: NSOUR = NNSOUR + 1
253: C
254: return
255: end

```

src/shared/prlat0.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine PRLAT0( NL,   IOG,   IDLAT, LTY, NVLAT, SZLAT,
3:   &                 KLATT, KSLAT, NNND1, NNND2, NNND3, PNND1,
4:   &                 PNND2, PNND3, KNND1, KNND2, KNND3,
5:   &                 NSTG, PCEL, KCEL,
6:   &                 ALIAS, NNNDWK, IRC )
7: C=====
8: C PURPOSE:
9: C   PRINTOUT INPUT DATA FOR A LATTICE (BEFORE ANY TRANSFORMATION)
10: C
11: C   (THIS PROCEDURE IS MOVED HERE FROM LATTIN. NOVEMBER 1990)
12: C
13: C CALLED IN: LATTIN
14: C CALLS: PRLATT
15: C=====
16:   integer IDLAT, LTY, NVLAT(4), KLATT(*), KSLAT(*)
17:   real*8 SZLAT(8)
18:   real PNND1(NNND1), PNND2(NNND2), PNND3(NNND3)
19:   integer KNND1(NNND1), KNND2(NNND2), KNND3(NNND3)
20:   real PCEL(NSTG)
21:   integer KCEL(NSTG)
22: C
23: C ... work
24:   real ALIAS(NNNDWK)
25: C
26:   integer IRC(2)
27: C
28:   write(IOG,7000) NL, IDLAT, LTY, (NVLAT(I),I=1,4)
29: 7000 format(/3X,'#####'//3X,' LATTICE NO.',I3,' : LATTICE-ID = ',I5,
30:   &        /3X,' ', TYPE = ',I3/3X,
31:   &        '#####'//
32:   &        6X,'== LATTICE DIMENSION == NVLAT = (' ,3I6,' )')
33: C
34: C
35: C
36: C ..... PRINT SZLAT .....
37: C
38:   write(IOG,'(/6X,'== SIZE PARAMETERS ==')')
39: C
40:   if ( LTY.eq.1 ) then
41:     write(IOG,7020) (SZLAT(I),I=1,3)
42: 7020 format(/8X,'X-WIDTH = ',1P,E13.5,', Y-WIDTH = ',E13.5,
43:   &        ' Z-WIDTH = ',E13.5)
44: C
45: C .... Hexa lattice ...
46: C
47:   else if ( LTY.ge.2.and.LTY.le.4 ) then
48:     write(IOG,7040) (SZLAT(I),I=1,5)
49: 7040 format(/8X,'CELL PITCH = ',1P,E13.5/8X,
50:   &        'DIRECTION ANGLE OF CELL ARRAY = ',E13.5,' (DEGREE)'/
51:   &        8X,'CELL HEIGHT = ',E13.5/8X,
52:   &        'DEVIATION OF CELL ARRAY FROM FRAME CENTER = (' ,E13.5,
53:   &        2X,E13.5,' )')
54: C
55:   write(IOG,7060) (IRC(I),I=1,2)
56: 7060 format(/12X,'POSITION OF REFERENCE CELL (RCCELL) = (' ,I4,2X,I4,
57:   &        ' )')
58: C
59:   if ( LTY.eq.2 ) then
60:     write(IOG,7080) (SZLAT(I),I=6,7)
61: 7080 format(/8X,'SIZE OF LATTICE FRAME (RHP) '/8X, ' WIDTH = ',
62:   &        1P,E13.5,' HEIGHT = ',E13.5/)
63: C
64:   else if ( LTY.eq.3 ) then
65:     write(IOG,7100) (SZLAT(I),I=6,7)

```

```

66: 7100 format(/8X,'SIZE OF LATTICE FRAME (CYLINDER) ',12X,
67:   &        ' RADIUS = ',1P,E13.5,' HEIGHT = ',E13.5/)
68: C
69:   else if ( LTY.eq.4 ) then
70:     write(IOG,7120) (SZLAT(I),I=6,8)
71: 7120 format(/8X,'SIZE OF LATTICE FRAME (BOX,RPP) ',12X,
72:   &        ' X-WIDTH = ',1P,E13.5,' Y-WIDTH = ',E13.5,
73:   &        ' Z-WIDTH = ',E13.5/)
74:   end if
75: C
76: C .... STG region as a lattice ...
77: C   ( NND probabilities are normalized and aliased here )
78: C   ( Partial Packing Fraction is normalized and aliased here )
79: C
80:   else if ( LTY.eq.10 ) then
81:     write(IOG,7140) (SZLAT(I),I=1,2), KLATT(1)
82: 7140 format(/8X,'STG-region as a lattice'/8X,'FP = ',1P,E13.5,
83:   &        ' Cell-radius = ',E13.5,
84:   &        ' Cell ID for base material = ',I8)
85:     write(IOG,7260) (KLATT(I+1),I=1,NSTG)
86: 7260 format(/8X,'STG Cell ID = ',8I8)
87:     if ( NSTG.gt.1 ) then
88:       write(IOG,7280) (PCEL(I),I=1,NSTG)
89: 7280 format(/8X,'Partial PF = ',1P8E8.1)
90:       write(IOG,7300)
91: 7300 format(/8X,' == Sum of partial PF is normalized to be unity.')
92:     end if
93:     if ( SZLAT(2).eq.0.0D0 ) then
94:       write(IOG,7160)
95: 7160 format(/8X,' == Cell-radius will be replaced with that of ',
96:   &        'lattice frame body.')
97:     end if
98: C
99:   if ( NSTG.eq.1 ) then
100:     PCEL(1) = 0.5
101:     KCEL(1) = 1
102:   else if ( NSTG.gt.1 ) then
103:     call PNORM( PCEL, NSTG )
104:     call WALIAS( PCEL, ALIAS, KCEL, NSTG )
105:     call ATRANS( ALIAS, PCEL, NSTG )
106:   end if
107: C
108:   if ( NNND1.gt.0 ) then
109:     write(IOG,7180) 1, SZLAT(3), (PNND1(I),I=1,NNND1)
110: C ... convert cumulative distribution to distribution function
111:   do 100 I = NNND1, 2, -1
112:     PNND1(I) = PNND1(I) - PNND1(I-1)
113: 100 continue
114:     call PNORM( PNND1, NNND1 )
115:     write(IOG,7200) (PNND1(I),I=1,NNND1)
116:     call WALIAS( PNND1, ALIAS, KNND1, NNND1 )
117:     call ATRANS( ALIAS, PNND1, NNND1 )
118:   else if ( NNND1.lt.0 ) then
119:     write(IOG,7240) 1, -NNND1
120:   else
121:     write(IOG,7220) 1
122:   end if
123:   if ( NNND2.gt.0 ) then
124:     write(IOG,7180) 2, SZLAT(4), (PNND2(I),I=1,NNND2)
125:     do 110 I = NNND2, 2, -1
126:       PNND2(I) = PNND2(I) - PNND2(I-1)
127: 110 continue
128:     call PNORM( PNND2, NNND2 )
129:     write(IOG,7200) (PNND2(I),I=1,NNND2)
130:     call WALIAS( PNND2, ALIAS, KNND2, NNND2 )

```

src/shared/prlat0.f

```
131:      call ATRANS( ALIAS, PNND2, NNND2 )
132:      else if ( NNND2.lt.0 ) then
133:        write(IOG,7240) 2, -NNND2
134:      else
135:        write(IOG,7220) 2
136:      end if
137:      if ( NNND3.gt.0 ) then
138:        write(IOG,7180) 3, SZLAT(5), (PNND3(I),I=1,NNND3)
139:        do 120 I = NNND3, 2, -1
140:          PNND3(I) = PNND3(I) - PNND3(I-1)
141: 120    continue
142:        call PNORM( PNND3, NNND3 )
143:        write(IOG,7200) (PNND3(I),I=1,NNND3)
144:        call WALIAS( PNND3, ALIAS, NNND3, NNND3 )
145:        call ATRANS( ALIAS, PNND3, NNND3 )
146:      else if ( NNND3.lt.0 ) then
147:        write(IOG,7240) 3, -NNND3
148:      else
149:        write(IOG,7220) 3
150:      end if
151:
152: 7180    format(/8X,' NND-',I1,' is given with width ',E13.5//
153:      &      (4X,10(E12.5:)))
154: 7200    format(/8X,'      * after normalization *'/(4X,10(E12.5:)))
155: 7220    format(/8X,' NND-',I1,' is theoretical NND.')
156: 7240    format(/8X,' NND-',I1,' is same as that for lattice',I7,'.')
157:      end if
158: C
159: C      .... REVERSE ORDER OF KLATT (KSLAT) IN Y & Z DIRECTION ....
160: C
161:      if ( LTYP.eq.1 .or. (LTYP.ge.2.and.LTYP.le.4) ) then
162:        call RVLATT( NVLAT(1), NVLAT(2), NVLAT(3), KLATT )
163:        call RVLATT( NVLAT(1), NVLAT(2), NVLAT(3), KSLAT )
164: C
165: C      .... PRINT KLATT ....
166: C
167:        write(IOG,'(/6X,'== CELL NUMBERS OF THIS LATTICE ==' '/')')
168:        call PRLATT( IOG, NVLAT(1), NVLAT(2), NVLAT(3), KLATT )
169: C
170:        write(IOG,'(/6X,'== DIRECTION INDICES OF THIS LATTICE ==' '/')')
171:        call PRLATT( IOG, NVLAT(1), NVLAT(2), NVLAT(3), KSLAT )
172:      end if
173: C
174:      return
175:      end
```

src/shared/prlatt.f

```

1:      subroutine PRLATT( IOG,  NX,   NY,   NZ,   KLATT )
2: C=====
3: C  PURPOSE:  TO PRINT CELL NUMBERS OR DIRECTIONS IN A LATTICE.
4: C  CALLED IN: LATTIN          CALLS: NONE
5: C
6: C=====
7:      integer KLATT(NX,NY,NZ)
8: C
9:      parameter( MAXC = 120 )
10:     character*(MAXC) LINE
11:     character*8 INN
12:     character*6 ANN
13:     character*2 AMX
14:     character*32 FORMT
15: C
16:     character*16 MINUS
17:     data MINUS /'-----'/
18: C
19:     MKP      = 0
20:     MKM      = 0
21:     do 120 K = 1, NZ
22:       do 110 J = 1, NY
23:         do 100 I = 1, NX
24:           if ( KLATT(I,J,K).gt.0 ) then
25:             MKP = MAX(MKP,KLATT(I,J,K))
26:           end if
27:           if ( KLATT(I,J,K).lt.0 ) then
28:             MKM = MAX(MKM,-KLATT(I,J,K))
29:           end if
30:         continue
31:       110 continue
32:     120 continue
33:
34:     if ( MKP.gt.0 ) MKP = INT(LOG(FLOAT(MKP)+0.1)/LOG(10.0)) + 2
35:     if ( MKM.gt.0 ) MKM = INT(LOG(FLOAT(MKM)+0.1)/LOG(10.0)) + 3
36: C
37:     MDIG      = MAX(3,MAX(MKP,MKM))
38: C
39: C
40: C .... 7 CHARACTERS FOR Y INDEX & 4 DIGITS FOR EACH KLATT VALUE ...
41: C
42: C
43:     LX      = 7 + MDIG*NX
44:     MM      = MAXC/LX
45:     MX      = (MAXC-7) /MDIG
46:     INN     = ' '
47:     ANN     = ' '
48:     write(INN,'(I2,'(I','I2,'':'')'') MX, MDIG
49:     write(ANN,'(I2,'(A:')'') MX
50:     write(AMX,'(I2)'') MX
51: C
52: 7000 format(5X,A)
53:     LINE    = ' '
54: C
55:     if ( MM.le.1 .or. NZ.eq.1 ) then
56:       ML      = (NX-1) /MX + 1
57:       do 150 IZ = NZ, 1, -1
58:         do 140 L = 1, ML
59:           K1    = (L-1)*MX + 1
60:           K2    = MIN(NX,K1+MX-1)
61: C
62:           write(IOG,
63:             &      '(/5X,' ' *** INDEX ' ',I4,' ' IN Z-DIRECTION ****',
64:             &      ' ' X-POSITION ' ',I4,' ' TO ' ',I4,' ' *** ' /)'
65:             &      ) IZ, K1, K2

```

```

66:     FORMT    = ' (' ' -----: ' ', '//AMX// '(a,a:)) '
67:     write(LINE,fmt =FORMT) (MINUS(:MDIG-1),'+',K=K1,K2)
68:     write(IOG,7000) LINE(:ICLEN2(LINE))
69: C
70:     FORMT    = '(1X,I4,' ' : ' ', '//INN//)'
71:     do 130 IY = NY, 1, -1
72:       write(LINE,fmt =FORMT) IY, (KLATT(K,IY,IZ),K=K1,K2)
73:       write(IOG,7000) LINE(:ICLEN2(LINE))
74: 130 continue
75:     FORMT    = ' (' ' -----: ' ', '//AMX// '(a,a:)) '
76:     write(LINE,fmt =FORMT) (MINUS(:MDIG-1),'+',K=K1,K2)
77:     write(IOG,7000) LINE(:ICLEN2(LINE))
78: C
79:     FORMT    = ' (' ' Y / X ' ', '//INN//)'
80:     write(LINE,fmt =FORMT) (K,K=K1,K2)
81:     write(IOG,7000) LINE(:ICLEN2(LINE))
82: 140 continue
83: 150 continue
84: C
85: C ..... MM LEVELS IN-ZDIRECTION      IN ONE LINE .....
86: C
87:     else
88:       do 210 IZ = 1, NZ, MM
89:         IZ2    = MIN(NZ,IZ+MM-1)
90: C
91:         write(IOG,
92:           &      '(/6X,' ' *** INDEX ' ',I4,' ' TO ' ',I4,' ' IN Z-DIRECTION ****' /)'
93:           &      ) IZ, IZ2
94: C
95:         FORMT  = ' (' ' Z ' ',I3,' ' ', '//AMX// '(a,a:)) '
96:         do 160 KZ = IZ, IZ2
97:           L1    = LX*(KZ-IZ) + 1
98:           write(LINE(L1:),fmt =FORMT) KZ,
99:             (MINUS(:MDIG-1),'+',K=1,NX)
100: 160 continue
101:         write(IOG,7000) LINE(:ICLEN2(LINE))
102: C
103: C
104:     FORMT    = '(1X,I4,' ' : ' ', '//INN//)'
105:     do 180 IY = NY, 1, -1
106:       do 170 KZ = IZ, IZ2
107:         L1    = LX*(KZ-IZ) + 1
108:         write(LINE(L1:),fmt =FORMT) IY,
109:           (KLATT(I,IY,KZ),I=1,NX)
110:       &
111:       continue
112:       write(IOG,7000) LINE(:ICLEN2(LINE))
113: 170 continue
114: C
115: C
116:     FORMT    = ' (' ' -----: ' ', '//AMX// '(a,a:)) '
117:     do 190 KZ = IZ, IZ2
118:       L1    = LX*(KZ-IZ) + 1
119:       write(LINE(L1:),fmt =FORMT) (MINUS(:MDIG-1),'+',K=1,NX)
120: 190 continue
121:     write(IOG,7000) LINE(:ICLEN2(LINE))
122: C
123:     FORMT    = ' (' ' Y/X ' ', '//INN//)'
124:     do 200 KZ = IZ, IZ2
125:       L1    = LX*(KZ-IZ) + 1
126:       write(LINE(L1:),FORMT) (K,K=1,NX)
127: 200 continue
128:     write(IOG,7000) LINE(:ICLEN2(LINE))
129: C
130: 210 continue
131:     end if

```


src/shared/prlatt.f

```
131: C
132:   return
133: end
```

SAFE

src/shared/prnspd.f

```
1:      subroutine PRNSPD( IOG,   ISPACE,LEVEL, ISPNM, TNAMS, NNAMES,NEST,
2:      &                  NSPACE )
3: C=====
4: C purpose: to print subspace name
5: C=====
6: C
7:      include 'INC/_LNAM'
8:      character*(LNAM) TNAMS(NNAMES)
9:      integer ISPNM(2,0:NEST,NSPACE)
10: C
11: 7000 format(1X,' LEVEL',I2,' : SUBSPACE',I5,1X,4('!':,A,' :',A))
12:      write(IOG,7000) LEVEL, ISPACE, (
13:      &          TNAMS(ISPNM(1,L,ISPACE))
14:      &          (1:ICLEN(TNAMS(ISPNM(1,L,ISPACE))))),
15:      &          TNAMS(ISPNM(2,L,ISPACE))
16:      &          (1:ICLEN(TNAMS(ISPNM(2,L,ISPACE))))),L=1,ISPNM(1,0,ISPACE))
17:      return
18:      end
```

src/shared/prnuwe.f

```
1: *VOCL TOTAL,SCALAR
2:      subroutine PRNUWE( IOT, NUM, WGT,   NG,   NR, HEAD1, HEAD2,
3:      &                  A, CHA )
4: C=====
5: C  PURPOSE: PRINT NUMBER (NUM) & WEIGHT (WGT)
6: C          NG = GROUP NUMBER, NR = REGION NUMBER   ETC.
7: C  CALLED IN: PRMNTR
8: C  CALLS: REGNM0
9: C=====
10:      real*8 NUM(NG,NR)
11:      real*8 WGT(NG,NR)
12:      character*6 HEAD1, HEAD2
13: C
14:      real A(*)
15:      character*4 CHA(*)
16: C
17:      parameter( NL = 6 )
18:      character*12 RNAME(NL)
19: C
20: C -----
21: C
22:      do 120 N = 1, NR, NL
23:          N2 = MIN(NR,N+NL-1)
24:          write(IOT,7000) HEAD1, (HEAD2,K,K=N,N2)
25: c##<2007/03/14:PN3:
26: c7000      format(/1X,A6,6(5X,A6,I4,6X:))
27: c7000      format(/1X,A6,6(5X,A6,I4,6X:))
28: c##>
29: c
30:          if ( HEAD2.eq.'REGION' ) then
31:              do 100 K = N, N2
32:                  call REGNM0( K, '>', RNAME(K-N+1), LNM, A, CHA )
33:                  if ( LNM.lt.LEN(RNAME(1)) ) RNAME(K-N+1) (LNM+1:) = ' '
34:              100      continue
35:                  write(IOT,7020) (RNAME(K-N+1),K=N,N2)
36: c##<2007/03/14:PN3:
37: c7020      format(/1X,6X,6(' <',A12,'> '))
38: c7020      format(/7X,6(' <',A12,'>':4X))
39: c##>
40:          end if
41: c
42:          write(IOT,7040) (' ',K=N,N2)
43: c##<2007/03/14:PN3:
44: c7040      format(/1X,6X,6(A1,' NUMBER AV.WEIGHT ':))
45: c7040      format(/7X,6(A1,' NUMBER AV.WEIGHT':1X))
46: c##>
47:          do 110 I = 1, NG
48:              write(IOT,7060) I,
49:              &                  (NUM(I,K),WGT(I,K)/MAX(1D0,NUM(I,K)),K=N,N2)
50: c##<2007/03/14:PN3:
51: c7060      format(1X,I5,1X,6(0P,F10.0,1X,1P,D10.3:))
52: c7060      format(I6,1X,6(0P,F10.0,1P,D11.3:))
53: c##>
54:          110      continue
55:          120      continue
56:      return
57:      end
```

src/shared/prpara.f

```
1:      subroutine PRPARA( NTASK, IRANT, IRSUT, IRSMT, IRSLT )
2:      C=====
3:      C purpose:
4:      C   setting of random numbers etc. for VPP-FORTRAN parallel mode.
5:      C called in: INTRO
6:      C calls: RNINIT, RANU2
7:      C   IDVPROC(intrinsic function of VPP Fortran. returns proc ID.)
8:      C=====
9:      C/IF PARA(VPP)
10:     include 'INC/_SIZES'
11:     C
12:     include 'INC/_VPPPARA'
13:     include 'INC/_IOUNIT'
14:     C
15:     include 'INC/_RAND'
16:
17: !xocl processor p( MPE )
18: !xocl subprocessor sp=p(1:MPE)
19:
20:     real      RTMP(10)
21:     integer   IRANT(NTASK)
22:     integer   IRSUT(NTASK), IRSMT(NTASK), IRSLT(NTASK)
23:
24: C----- set IRAND -----
25:
26:     do 20 ITSK = 1, MPE
27:       IRANT(ITSK) = IRAND
28:       IRSUT(ITSK) = IRNSU
29:       IRSMT(ITSK) = IRNSM
30:       IRSLT(ITSK) = IRNSL
31:       call RANU2(IRAND, RTMP, 1 , ICOND)
32: 20    continue
33:
34: !xocl spread do
35:     do 30 ITSK=1, MPE
36:       IRAND = IRANT(ITSK)
37:       IRNSU = IRSUT(ITSK)
38:       IRNSM = IRSMT(ITSK)
39:       IRNSL = IRSLT(ITSK)
40: 30    continue
41: !xocl end spread
42:
43: C----- set skip-number -----
44:
45:     NRSTEP = NTASK
46:     if( mod(NTASK,2) .eq. 0 ) NRSTEP = NRSTEP + 1
47:     call RNINIT( NRSTEP )
48:
49: C----- Change IOW -----
50:
51:     IOW = IVPPPR(idvproc())
52: C/#ENDIF
53:
54:     return
55:     end
```

src/shared/prstop.f

```
1:      subroutine PRSTOP( NERR, MESSG )
2: C=====
3: C Purpose: To print summary input-data error when terminating
4: C         before completing input data processing.
5: C
6: C         nerr : increment count fatal error 'nerr' times.
7: C
8: C CALLED IN: anywhere
9: C CALLS PRNERR
10: C=====
11:      character*(*) MESSG
12:      include 'INC/_IUNIT'
13: C
14:      if ( NERR.gt.0 ) then
15:         do 100 I = 1, NERR
16:            call CNTERR( 'FATAL' )
17:         100 continue
18:      end if
19: C
20:      call PRNERR( IPR, ' ' )
21: C
22:      write(IMG,7000)
23: 7000 format('1'/5X,5('XXXXXXXXXX')//6X,
24: &         ' Your input data has one or more fatal errors which make '
25: &         '//6X,' it impossible to process all of your input data.'//
26: &         5X,5('XXXXXXXXXX')//)
27: C
28:      if ( MESSG.ne.' ' ) then
29:         write(IMG,7020) MESSG
30: 7020 format(/5X,' MESSAGE FROM SUBROUTINE : '//5X,A/)
31:      end if
32:
33: C/#IF PARA( PVM MPI )
34: *      call MVPCOM_EXIT(INFO)
35: C/#ENDIF
36:
37: C/#IF UNIX
38: *      call FLUSHSTD()
39: C/#ENDIF
40:      return
41:      end
```

src/shared/prtxyz.f

```
1:      subroutine PRXYZ( IOPR,  HEAD,  XYZ )
2: C=====
3: C PURPOSE :   Print a coordinate with a header
4: C
5: C Called in : INTRO2
6: C=====
7:      character*(*) HEAD
8:      real*8 XYZ(3)
9: C-----
10:     write(IOPR,7000) HEAD, XYZ(1), XYZ(2), XYZ(3)
11: 7000 format(1X,2X,A,' = (',1P,E13.6,', ',E13.6,', ',E13.6,', ' )')
12:     return
13:     end
```

src/shared/pt2zon.f

```

1:      subroutine PT2ZON( A,      JVMNT, JLATT, JSIMP, JHLAT,
2:      &                  X0,      Y0,      Z0,
3:      &                  XYZ,      NP,      IZON,      IREG,
4:      &                  LEVLP, LZZP, LPOSP, LCRSP,
5:      &                  KZREG, IPCEL, KCELL, KDALT, KZMAT, MLBZZ,
6:      &                  ISUSP, KSPSU, KTCSP,
7:      &                  NSDA, NZDA, NZONE, NEST, NCELL, NLBZ,
8:      &                  NSPACE, NSUZON, NUNV, NKTCSPP )
9: C=====
10: C  PURPOSE:  DETERMINE ZONE # & LATTICE PARAMETER FROM
11: C             ABSOLUTE COORDINATES OF A POINT
12: C             (3/15/1991)
13: C
14: C i  X0,Y0,Z0 : absolute coordinates given XPDET(NP,I=3,5).
15: C
16: C  NP : detector #
17: C
18: C < determined parameters in this routine >
19: C
20: C  XYZ(3,NEST) : detector position in each level.
21: C  IZON : zone #
22: C           IZON=0 when the zone is a STG region.
23: C           IZON<0 if ZONE IS NOT FOUND.
24: C  IREG : region #
25: C  XC,YC,ZC : intra-cell coordinates
26: C  LEVLP : lattice cell nesting depth
27: C  LZZP(NEST) : lattice-buffer zones in each nesting level
28: C  LPOSP(NEST) : cell position in each nesting level
29: C           LPOSP(LEVLP)=-1 when the zone is a STG region.
30: C  LCRSP(NEST) : flags for cell crossing with lattice boundary for
31: C                each nesting level
32: C
33: C
34: C  CALLED IN : RXPDET
35: C  CALLES : JUDGE
36: C=====
37:      implicit real*8(D)
38:      real A(*)
39:
40:      include 'INC/_PGEOM'
41:      include 'INC/_IUNIT'
42: C
43:      real*8 X0, Y0, Z0
44:      real*8 XC, YC, ZC
45:      real*8 XYZ(3,NEST)
46:      integer LZZP(NEST), LPOSP(NEST), LCRSP(NEST)
47: C
48: C ... geometry data ...
49: C
50:      integer IPCEL(NCELL), KCELL(NZONE), KDALT(NLBZ), KZMAT(NZONE),
51:      &        MLBZZ(NZONE), KZREG(NZONE)
52:      integer ISUSP(NSUZON,NSPACE), KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSPP)
53: C
54: C ... local variables .....
55: C
56:      real*8 X1, Y1, Z1
57:      logical IFI, IFL
58:      include '../shared/INC/_PMLATT'
59:      include '../shared/INC/_SFLATT'
60: C
61: C-----
62: C
63: C .... SEARCH ZONE FOR POINT (X0,Y0,Z0) .....
64: C
65:      MMZ      = NZONE

```

```

66:      if ( JLATT.ne.0 ) MMZ      = IPCEL(1) - 1
67: C
68:      IZON      = -1
69: C
70:      do 100 M = 1, MMZ
71:          call JUDGE( 'PT2ZON', M, 1, X0, Y0, Z0, IFI, A(LSDA), A(LKZDA),
72:          &            A(LKZAA), NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP,
73:          &            .false., DUMMY, DUMMY, DUMMY )
74: C
75:          if ( IFI ) then
76:              IZON      = M
77:              go to 110
78:          end if
79:      100 continue
80:      110 continue
81: C
82:      if ( IZON.gt.0 ) then
83:          IREG      = KZREG(IZON)
84: C
85: C .... MATERIAL IN THE ZONE IS NON-REAL MATERIAL.
86: C
87:      if ( JLATT.eq.0.and.KZMAT(IZON).lt.0 ) then
88:          IZON      = -1
89:          IREG      = KZREG(IZON)
90:          go to 150
91:      end if
92:      else
93: C
94: C .... ABNORMAL END 2 : ZONE NOT FOUND!
95: C
96:          IZON      = -2
97:          go to 150
98:      end if
99: C
100:      LEVLP      = 0
101:      XC          = X0
102:      YC          = Y0
103:      ZC          = Z0
104: C
105: C=====
106: C
107: C  FOR CASES WITH LATTICE-GEOMETRY
108: C
109: C=====
110: C
111:      if ( JLATT.ne.0 ) then
112: C
113: C .... NOT IN LATTICE .... NORAL END .....
114: C
115:          if ( KZMAT(IZON).ge.0 ) then
116:              go to 150
117:          end if
118: C
119: C .... IN LATICE REGION .....
120: C
121: CCCCCC if ( KZMAT(IZON).le.-1.and.KZMAT(IZON).ge.-998 ) then
122: C        if ( ISLATT(KZMAT(IZON)) ) then
123: C
124:          IZ      = IZON
125:          ISPC      = 0
126: C
127:          do 140 L = 1, NEST
128: CCCCCCCCC      MLT      = -KZMAT(IZ)
129:                  MLT      = LATNM(KZMAT(IZ))
130:                  M        = KDALT(MLBZZ(IZ))

```

src/shared/pt2zon.f

```

131:          IWK1      = 1
132: C
133:          LEVL1     = L - 1
134:          NBANK1    = 1
135:          INN       = 1
136:          NPT1      = 1
137: C -----
138: C   SINK INTO LATTICE REGION : RETURN LZSP, LPOSP, LCRSP OF LEVEL L.
139: C
140: C -----
141:          X1        = XC
142:          Y1        = YC
143:          Z1        = ZC
144:          DAI       = 0.0
145:          DBI       = 0.0
146:          DCI       = 0.0
147: C
148:          call SINKLT( NPT1, MLT, M, IZ, JHLAT, X1, Y1, Z1, DAI,
149: &                   DBI, DCI, IWK1, IZZ1, XC, YC, ZC, DAI, DBI, DCI,
150: &                   IPC, LEVL1, LZSP, LPOSP, LCRSP, NBANK1, A(LDALT),
151: &                   A(LLTYP), A(LNVLAT), A(LSZLAT), A(LIPLAT),
152: &                   A(LKLATT), A(LKSLAT), A(LKHLAT), A(LIPCEL),
153: &                   A(LCELSZ), DEPS )
154: C
155: C -----
156: C   store detector position in this level
157:          XYZ(1,L) = XC
158:          XYZ(2,L) = YC
159:          XYZ(3,L) = ZC
160: C -----
161: C
162: C   .... ABNORMAL END 3 : CANNOT FIND CELL ! .....
163:          if ( IPC.eq.IZ ) then
164:             IZON      = -3
165:             go to 150
166: C
167: C   .... ZONE IS A STG REGION .....
168:          else if ( IPC.eq.0 ) then
169:             IZON      = 0
170:             LEVLP     = L
171:             go to 150
172:          end if
173: C
174: C   .... FIND ZONE # IN CELL .....
175: C
176:          MZ1         = IPCEL(KCELL(IPC))
177:          MZ2         = IPCEL(KCELL(IPC)+1) - 1
178:          do 120 MZZ = MZ1, MZ2
179:             INX1      = 1
180: C
181:             call JUDGE( 'PT2ZON', MZZ, INX1, XC, YC, ZC, IFI,
182: &                   A(LSDA), A(LKZDA), A(LKZAA), 1, 1, 1, JVMNT,
183: &                   IFL, JSIMP, .false., DUMMY, DUMMY, DUMMY )
184: C
185:             if ( IFI ) go to 130
186: 120        continue
187: C
188: C   .... ABNORMAL END 4 : CANNOT FIND ZONE IN A CELL !
189:          IZON      = -4
190:          go to 150
191: C
192: C   .... SINK INTO DEEPER LEVEL .....
193: C
194: CC130          if ( KZMAT(MZZ).le.-1.and.KZMAT(MZZ).ge.-998 ) then
195: 130          if ( ISLATT(KZMAT(MZZ)) ) then

```

```

196:          IZ        = MZZ
197: C
198: C   .... GET TO REAL ZONE ! .....
199: C
200:          else
201:             IZON     = MZZ
202:             LEVLP    = L
203:             go to 150
204:          end if
205: 140        continue
206:          end if
207:          end if
208: C
209: 150 if ( IZON.lt.-1 ) then
210:    call CNTERR( 'FATAL' )
211:    write(IPR,*) ' XXX CANNOT FIND A ZONE FOR POINT-DETECTOR ', NP,
212: &              ' ERROR CODE = ', IZON
213:    else if ( IZON.eq.-1 ) then
214:      call CNTERR( 'FATAL' )
215:      write(IPR,*) ' XXX POINT-DETECTOR ', NP,
216: &              ' IS LOCATED IN A ZONE FOR BOUNDARY CONDITION.',
217: &              ' ERROR CODE = ', IZON
218: C    else if ( IZON.eq.0 ) then
219: C      call CNTERR( 'FATAL' )
220: C      write(IPR,*) ' XXX POINT-DETECTOR ', NP,
221: C &              ' IS LOCATED IN A STATISTICAL GEOMETRY REGION.',
222: C &              ' ERROR CODE = ', IZON
223:    end if
224: C
225:    return
226:  end

```


src/shared/ptinck.f

```
1:      subroutine PTINCK(IMG, NREG, WGTf)
2: C=<MVP/GMVP>=====
3: C purpose: check WGTf for beta-effective calculation. Currently, WGTf
4: C          must be 1.0.
5: C called in: INTRO
6: C=====
7:      include '../mvp/INC/_PERT'
8:      real    WGTf(NREG,2)
9:
10:     if(NPTBE.eq.0) return
11:     do I = 1, NREG
12:       if((Abs(WGTf(I,1)-1.0).gt.1.0e-6)) then
13:         write(IMG,'(1x,a,a)')
14: &         'XXX(ptinck) Currently, all values of WGTf must be 1.0 for',
15: &         'beta-effective calculation.'
16:         call CNTERR('FATAL')
17:         exit
18:       end if
19:     end do
20:     return
21: end
```

src/shared/ptinp.f

```

1:      subroutine PTINP( CODE,   IPR, ICALL,   A,   IA,   DA,   CHA,
2:      &                H,   IH,   DH, NUCID,   IUB )
3: C=<MVP/GMVP>=====
4: C purpose: Input tally specifications in $PERTURBATION block.
5: C called in: INTRO routine of MVP/GMVP.
6: C=====
7: C arguments ( i=input, o=output, w=work )
8: C i CODE : name of code using this routine ('MVP'/'GMVP')
9: C i IPR : I/O unit number
10: C o ICALL : output with value 1.
11: C i o A : task shared dynamic data area used as "REAL" type.
12: C i o IA : task shared dynamic data area used as "INTEGER" type.
13: C i o DA : task shared dynamic data area used as "REAL*8" type.
14: C i o CHA : task shared dynamic data area used as "CHARACTER*4" type.
15: C i o H : task local dynamic data area used as "REAL" type.
16: C i o IH : task local dynamic data area used as "INTEGER" type.
17: C i o DH : task local dynamic data area used as "REAL*8" type.
18: C i NUCID : nuclide ID.
19: C i IUB : temporary I/O unit.
20: C
21: C (A,IA,DA,H,IH & DH must have the same address as A(*) & H(*) of
22: C common /ARRAY/ & /LARRAY/ )
23: C
24: C=====
25:      character*(*) CODE
26:      real A(*), H(*)
27:      character*4 CHA(*)
28:      integer IA(*), IH(*)
29:      real*8 DA(*), DH(*)
30:
31:      character*16 NUCID(NUC)
32:      character*256 CHAPTIN
33:
34:      include '../mvp/INC/_KPIDS'
35:      include 'INC/_SIZES'
36:      include '../mvp/INC/_CXSEC'
37:      include '../mvp/INC/_PERT'
38:      include '../mvp/INC/_FLAGS'
39: c
40: c ===== local variables =====
41: c
42:      character VNAME*72, VSUBF*1
43: c
44: c ... Items for perturbation tally specification ...
45: c
46:      character*6 TAG
47:      integer ID
48:      character*72 TLABEL
49:      character*12 METHOD
50: c
51:      character CWRK*128
52: c
53: c-----
54: c
55: c .... search parameters in input ..
56: c
57:      ICALL = 1
58:      call HEADER( IPR, 'PERTURBATION INPUT DATA' )
59: c
60:      call GTLAST( LAST )
61: c
62:      NERR = 0
63: c
64:      call RWIND( IUB )
65: c

```

```

66:      KJEVNT = 0
67:      KJPART = 0
68:      KJPTYP = 0
69:      KJPARA = 0
70:      KJMTHD = 0
71:      KJFRAC = 0
72:      NJFRAC = 0
73: c
74:      KJDIFF = 1
75:      KJCORR = 2
76: c
77:      KJTRAC = 1
78:      KJCOLL = 2
79:      KJANAC = 3
80:      KJSURF = 4
81:      KJPOIN = 5
82: c
83:      KJPHOT = 1
84:      KJNEUT = 2
85: c
86:      KJKEFF = 1
87:      KJFLUX = 2
88:      KJMACR = 3
89:      KJMICR = 4
90:      KJBEFF = 5
91: c
92:      KJDENS = 1
93:      KJNUMB = 2
94:      KJTEMP = 3
95: c
96:      NGSP = 10
97:      NORDDSDS = 8
98:      NOPSDDS = 8
99: c
100:      JIDPT = 0
101:      JMTHD = 0
102:      JNGSP = 10
103:      JEVNT = KJTRAC
104:      JPART = KJNEUT
105:      JPTYP = KJKEFF
106:      JPARA = 0
107: c
108:      JMICN = 0
109:      JXTYP = 0
110: c
111:      KJID = 0
112:      TLABEL = ' '
113:      METHOD = ' '
114:      KJPTTR = 0
115:      KJPTNU = 0
116: c
117: c ... size and pointer for working area ...
118: c
119:      NWORK = MAX(NREG,NUC)
120: c MAXDA : Maximum size of DELA array
121:      MAXDA = 10
122:      LJPTTR = 0
123: c
124: c ITAL : number of tallies
125: c IPTDS : number of tallies for differential operator sampling
126: c IPTCS : number of tallies for correlated sampling
127: c (These numbers are automatically counted in this routine.)
128: c IPTBE : number of tallies for beta-effective
129: c NUCPT : number of perturbed nuclides. Currently, NUCPT must be
130: c always 1.

```

src/shared/ptinp.f

```

131:
132:   call KEEP('IWRKX', LIWRKX, NREG, 'I4', LAST, JDEBG)
133:   call KEEP('IWKRK', LIWKRG, NREG, 'I4', LAST, JDEBG)
134:   call KEEP('IWKNU', LIWKNU, NUC, 'I4', LAST, JDEBG)
135:   call KEEP('LRWKA', LRWKA, MAXDA, 'R4D', LAST, JDEBG)
136:
137:   ITAL = 0
138:   IPTDS = 0
139:   IPTCS = 0
140:   IPTBE = 0
141:   NUCPT = 1
142:
143: 100 call CHREAD( VNAME, NLEN, IEND )
144:   if ( IEND.ne.0 ) go to 170
145: c
146: c ... search a character ' '.
147: c
148:   IS = 1
149:   IE = NLEN
150:   NHYP = 0
151:   do 105 I = IS, IE
152:     if ( VNAME(I:I).eq.'-' ) NHYP = NHYP + 1
153:   105 continue
154: c
155: c=====
156: c === data for each perturbation tally ===
157: c=====
158: c
159: c ... ID ...
160: c
161:   if ( NHYP.eq.0 .and. VNAME(:NLEN).eq.'ID' ) then
162:     if ( KJID.ne.0 ) then
163:       write(IPR,*) 'XXX($PERT) Duplicate "ID" input.'
164:       call CNTERR( 'FATAL' )
165:       NERR = NERR + 1
166:     end if
167:     call I4READ( VNAME(:NLEN), KJID, NA, -1, IERR )
168:     JIDPT = KJID
169: c
170: c ... LABEL ...
171: c
172:   else if ( NHYP.eq.0 .and. VNAME(:3).eq.'LAB' ) then
173:     if ( TLABEL.ne.' ' ) then
174:       write(IPR,*) 'XXX($PERT) Duplicate "LABEL" input.'
175:       call CNTERR( 'FATAL' )
176:       NERR = NERR + 1
177:     end if
178: c
179:     IL = 0
180: c
181: 110 call CHREAD( VNAME(:NLEN), TLABEL(IL+1:), ' ', NL, ITERM, IEND
182:   &
183:   if ( IEND.ne.0 ) go to 150
184:   IL = MIN(LEN(TLABEL), IL+NL+1)
185:   if ( ITERM.eq.0 ) go to 110
186: c
187:   LLABEL = IL
188: c
189: c ... METHOD ...
190: c
191:   else if ( VNAME(:6).eq.'METHOD' ) then
192:     if ( METHOD.ne.' ' ) then
193:       write(IPR,*) 'XXX($PERT) Duplicate "METHOD" input.'
194:       call CNTERR( 'FATAL' )
195:       NERR = NERR + 1

```

```

196:   end if
197:   KJMTD = 1
198:
199:   call CHREAD( VNAME(:NLEN), METHOD, ' ', NL, ITERM, IEND )
200:   if ( IEND.ne.0 ) go to 150
201:
202:   LMTHD = NL
203:
204:   if ( METHOD(:4).eq.'DIFF' ) then
205:     IPTDS = IPTDS + 1
206:     JMTHD = KJDIFF
207:   else if ( METHOD(:4).eq.'CORR' ) then
208:     IPTCS = IPTCS + 1
209:     JMTHD = KJCORR
210:   else
211:     write(IPR,*) 'XXX($PERT) "METHOD" must be "DIFF" or "CORR".'
212:     call CNTERR( 'FATAL' )
213:     NERR = NERR + 1
214:   end if
215:
216:   if ( ITERM.eq.0 ) then
217:     NERR = NERR + 1
218:     write(IPR,*) 'XXX($PERT) "METHOD" data ',
219:   &
220:     'has too many items or lacks closing )'.".'
221:     call DMREAD( VNAME(:NLEN), NA, NB, IEND )
222:     if ( IEND.ne.0 ) go to 180
223:   end if
224: c
225: c ... PERTURBED-SOURCE ...
226: c
227:   else if ( NHYP.eq.1 .and. IMATCH('PERT*-SOUR*', VNAME(IS:IE)).eq.1
228:   &
229:   ) then
230:     call I4READ( VNAME(:NLEN), JNGSP, NA, -1, IERR )
231:     if ( JNGSP.gt. NGSP ) then
232:       NGSP = JNGSP
233:     end if
234: c
235: c ... FRACTIONAL-CHANGE ...
236: c
237:   else if ( NHYP.eq.1 .and. IMATCH('FRAC*-CHAN*', VNAME(IS:IE)).eq.1
238:   &
239:   ) then
240:     KJFRAC = 1
241:     do 510 KDA = 1, MAXDA
242:       A(LRWKDA+KDA-1) = 0.0
243:     510 continue
244:     call R4READ( VNAME(:NLEN), A(LRWKDA), NA, -MAXDA, IERR )
245:     NJFRAC = NA
246:     if ( NA.gt.MAXDA ) then
247:       write(IPR,6060) VNAME(:NLEN), NA, MAXDA
248:       format(/1X,'XXX($PERT) size of data in item <',A,'> ',
249:   &
250:       ' exceeds that of a working area for input;/1X,
251:   &
252:       ' (num. of data ',I6,' > limit =',I6,')'/1X,
253:   &
254:       ' Number of data is invalid or you need',
255:   &
256:       ' larger working memory size ( "NWORK" after $PERT ).')
257:       call PRSTOP( 1, 'Data size overflow in $PERT block.' )
258:       stop 888
259:     end if
260:     if ( IERR.ne.0 ) go to 150
261: c
262: c ... PERTURBED-REGION ...
263: c
264:   else if ( NHYP.eq.1 .and. IMATCH('PERT*-REG*', VNAME(IS:IE)).eq.1
265:   &
266:   ) then
267:     KJPTTR = 1
268:     do 520 IREG = 1, NREG

```

src/shared/ptinp.f

```

261:      IA(LIWKRG+IREG-1) = 0
262: 520  continue
263:
264: 120  call CHREAD( VNAME(:NLEN), CHAPTN, ' ', NCH, ITERM, IEND
265: &
266:      if ( IEND.ne.0 ) go to 150
267:      call REGLST( CHAPTN(:NCH), IA(LIWKRX), NREG, NR, NR2, A,
268: &
269:          CHA, IERR )
270:      if ( NR.eq.0 ) then
271:          write(IPR,*) 'XXX($PERT) no region that',
272: &
273:          ' matches a name <', CHAPTN(:NCH), '>'
274:          call CNTERR( 'FATAL' )
275:          NERR = NERR + 1
276:      else if ( NR2.gt.NR ) then
277:          write(IPR,*) 'XXX($PERT) too many regions that',
278: &
279:          ' matches name <', CHAPTN(:NCH), '>'
280:          write(IPR,*) 'A strange error occurred !!!'
281:          call CNTERR( 'FATAL' )
282:          NERR = NERR + 1
283:      else
284:          do 530 IREGX = 1, NR
285:              IREG = IA(LIWKRX+IREGX-1)
286:              IA(LIWKRG+IREG-1) = 1
287:          530 continue
288:      end if
289:      if ( ITERM.eq.0 ) go to 120
290: c
291: c ... PERTURBED-NUCLIDE ...
292: c
293:      else if ( NHYP.eq.1 .and. IMATCH('PERT*-NUC*',VNAME(:NLEN)).eq.1
294: &
295:          ) then
296:          KJPTNU = 1
297:          do 540 KNUC = 1, NUC
298:              IA(LIWKNU+KNUC-1) = 0
299:          540 continue
300:          call CHREAD( VNAME(:NLEN), CHAPTN, ' ', NCH, ITERM, IEND
301: &
302:          )
303:          if ( IEND.ne.0 ) go to 150
304:          IEXNUC = 0
305:          do 550 KNUC = 1, NUC
306:              if ( IMATCH(CHAPTN(:NCH),NUCID(KNUC):(ICLEN(NUCID(KNUC))))
307: &
308:                  .eq.1 ) then
309:                  IA(LIWKNU+KNUC-1) = 1
310:                  IEXNUC = 1
311:              end if
312:          550 continue
313:          if ( IEXNUC .eq. 0 ) then
314:              write(IPR,*) 'XXX($PERT) no nuclide that',
315: &
316:              ' matches a name <', CHAPTN(:NCH), '>'
317:              call CNTERR( 'FATAL' )
318:              NERR = NERR + 1
319:          end if
320:          if ( ITERM.eq.0 ) go to 130
321: c
322: c ... EVENT ...
323: c
324:      else if ( VNAME(:5).eq.'EVENT' ) then
325:          if ( KJEVNT.ne.0 ) then
326:              write(IPR,*) 'XXX($PERT) Duplicate "EVENT" input.'
327:              call CNTERR( 'FATAL' )
328:              NERR = NERR + 1
329:          end if
330:          KJEVNT = 1
331: c

```

```

326:      call CHREAD( VNAME(:NLEN), CHAPTN, ' ', NCH, ITERM, IEND )
327:      if ( IEND.ne.0 ) go to 150
328: c
329: c
330:      if ( CHAPTN(:4).eq.'TRAC' ) then
331:          JEVNT = KJTRAC
332:      else if ( CHAPTN(:4).eq.'COLL' ) then
333:          JEVNT = KJCOLL
334:      else if ( IMATCH('ANA*-COL*',CHAPTN(:NCH)) .eq. 1 ) then
335:          JEVNT = KJANAC
336:      else if ( CHAPTN(:4).eq.'SURF' ) then
337:          JEVNT = KJSURF
338:      else if ( CHAPTN(:4).eq.'POIN' ) then
339:          JEVNT = KJPOIN
340:      else
341:          write(IPR,*) 'XXX($PERT) "EVENT" must be "TRAC" or "COLL" ',
342: &
343:          ' or "ANA-COL" or "SURF" or "POINT".'
344:          call CNTERR( 'FATAL' )
345:          NERR = NERR + 1
346:      end if
347: c
348: c
349:      if ( ITERM.eq.0 ) then
350:          NERR = NERR + 1
351:          write(IPR,*) 'XXX($PERT) "EVENT" data ',
352: &
353:          'has too many items or lacks closing )'.".'
354:          call DMREAD( VNAME(:NLEN), NA, NB, IEND )
355:          if ( IEND.ne.0 ) go to 180
356:      end if
357: c
358: c ... PHOTON|NEUTRON ...
359: c
360:      else if ( VNAME(:4).eq.'PHOT'
361: &
362:          .or. VNAME(:4).eq.'NEUT' ) then
363:          if ( KJPART.ne. 0 ) then
364:              write(IPR,*) 'XXX($PERT) Duplicate "PHOTON|NEUTRON" input ',
365: &
366:              'or exclusive options.', ' ', VNAME(:NLEN), ' '
367:              call CNTERR( 'FATAL' )
368:              NERR = NERR + 1
369:          end if
370:          KJPART = 1
371: c
372: c
373:      if ( VNAME(:4).eq.'PHOT' ) then
374:          JPART = KJPART
375:      else if ( VNAME(:4).eq.'NEUT' ) then
376:          JPART = KJNEUT
377:      end if
378: c
379: c ... KEFF|FLUX|MACRO|MICO ...
380: c
381:      else if ( VNAME(:4).eq.'KEFF'
382: &
383:          .or. VNAME(:4).eq.'BEFF'
384: &
385:          .or. VNAME(:4).eq.'FLUX'
386: &
387:          .or. VNAME(:4).eq.'MACR'
388: &
389:          .or. VNAME(:6).eq.'MICO:' ) then
390:          if ( KJPTYP.ne. 0 ) then
391:              write(IPR,*) 'XXX($PERT) Duplicate "KEFF|FLUX|MACRO|MICO"',
392: &
393:              ' input or exclusive options.', ' ',
394: &
395:              VNAME(:NLEN), ' '
396:              call CNTERR( 'FATAL' )
397:              NERR = NERR + 1
398:          end if
399:          KJPTYP = 1
400: c
401: c
402:      if ( VNAME(:4).eq.'KEFF' ) then
403:          JPTYP = KJKEFF
404:      else if ( VNAME(:4).eq.'BEFF' ) then

```

src/shared/ptinp.f

```

391:      JPTYP = KJBEFF
392:      IPTBE = IPTBE + 1
393:      else if ( VNAME(:4).eq.'FLUX' ) then
394:        JPTYP = KJFLUX
395:      else if ( VNAME(:4).eq.'MACR' ) then
396:        JPTYP = KJMACR
397:      else if ( VNAME(:6).eq.'MICRO:' ) then
398:        JPTYP = KJMICR
399:        if ( NLEN .le. 6 ) then
400:          write(IPR,*) 'XXX($PERT) Option "MICRO" does not specify ',
401:            & ' nuclide.'
402:          call CNTERR( 'FATAL' )
403:          NERR = NERR + 1
404:        else
405:          JMICN = 0
406:          do 560 KNUC = 1, NUC
407:            if ( IMATCH(VNAME(7:NLEN),NUCID(KNUC)
408:              & (:ICLEN(NUCID(KNUC)))) .eq.1 ) then
409:              JMICN = KNUC
410:              go to 125
411:            end if
412: 560      continue
413:      write(IPR,*) 'XXX($PERT) Option "MICRO" specifies',
414:        & ' incorrect nuclide.'
415:        & VNAME(7:NLEN),'.',
416:        call CNTERR( 'FATAL' )
417:        NERR = NERR + 1
418: 125      continue
419:      end if
420:    end if
421:  c
422:    if ( VNAME(:4).eq.'MACR'
423:      & .or. VNAME(:6).eq.'MICRO:' ) then
424:  c
425:      CHAPTIN = ' '
426:      call CHREAD( VNAME(:NLEN), CHAPTIN, ' '), NCH, ITERM, IEND )
427:      if ( IEND.ne.0 ) go to 150
428:  c
429:      if ( CHAPTIN(:4).eq.'TOTA' ) then
430:        JXTYP = 1
431:      else if ( CHAPTIN(:4).eq.'NUFI' ) then
432:        JXTYP = 2
433:      else if ( CHAPTIN(:4).eq.'FISS' ) then
434:        JXTYP = 3
435:      else if ( CHAPTIN(:4).eq.'ELAS' ) then
436:        JXTYP = 4
437:      else if ( CHAPTIN(:4).eq.'CAPT' ) then
438:        JXTYP = 5
439:      else if ( CHAPTIN(:4).eq.'INEL' ) then
440:        JXTYP = 6
441:      else if ( CHAPTIN(:3).eq.'N2N' ) then
442:        JXTYP = 7
443:      else if ( CHAPTIN(:4).eq.'LOSS' ) then
444:        JXTYP = 8
445:      else
446:        write(IPR,*) 'XXX($PERT) "MACRO|MICRO" must be "TOTA",',
447:          & ' "NUFI", "FISS", "ELAS", "CAPT", "INEL", "N2N",',
448:          & ' or "LOSS".'
449:        call CNTERR( 'FATAL' )
450:        NERR = NERR + 1
451:      end if
452:  c
453:      if ( ITERM.eq.0 ) then
454:        NERR = NERR + 1
455:        write(IPR,*) 'XXX($PERT) "MACRO|MICRO" data ',

```

```

456:      & 'has too many items or lacks closing ")'.
457:      call DMREAD( VNAME(:NLEN), NA, NB, IEND )
458:      if ( IEND.ne.0 ) go to 180
459:    end if
460:  c
461:  end if
462:  c
463:  ... PARAMETER ...
464:  c
465:  else if ( VNAME(:4).eq.'PARA' ) then
466:    if ( KJPARA.ne. 0 ) then
467:      write(IPR,*) 'XXX($PERT) Duplicate "PARAMETER" input.'
468:      call CNTERR( 'FATAL' )
469:      NERR = NERR + 1
470:    end if
471:    KJPARA = 1
472:  c
473:  call CHREAD( VNAME(:NLEN), CHAPTIN, ' '), NCH, ITERM, IEND )
474:  if ( IEND.ne.0 ) go to 150
475:  c
476:  if ( CHAPTIN(:4).eq.'DENS' ) then
477:    JPARA = KJDENS
478:  else if ( CHAPTIN(:4).eq.'NUMB' ) then
479:    JPARA = KJNUMB
480:  else if ( CHAPTIN(:4).eq.'TEMP' ) then
481:    JPARA = KJTEMP
482:  else
483:    write(IPR,*) 'XXX($PERT) "PARAMETER" must be "DENS" ',
484:      & 'or "NUMB" or "TEMP".'
485:    call CNTERR( 'FATAL' )
486:    NERR = NERR + 1
487:  end if
488:  c
489:  if ( ITERM.eq.0 ) then
490:    NERR = NERR + 1
491:    write(IPR,*) 'XXX($PERT) "PARAMETER" data ',
492:      & 'has too many items or lacks closing ")'.
493:    call DMREAD( VNAME(:NLEN), NA, NB, IEND )
494:    if ( IEND.ne.0 ) go to 180
495:  end if
496:  c
497:  C=====
498:  C === delimiter of perturbation tally or "$END" ===
499:  C=====
500:  c
501:  else if ( VNAME(:NLEN).eq.'&' .or. VNAME(:NLEN).eq.'$END' ) then
502:  c
503:  ... print & check validity of the previous tally ...
504:  c
505:  if ( ITAL.gt.0 ) then
506:  c
507:  >>>> ID <<<<
508:  c
509:  if(KJID.eq.0) then
510:    write(IPR,*) 'XXX($PERT) ID must be input.'
511:    call CNTERR('FATAL')
512:    NERR = NERR + 1
513:  end if
514:  c
515:  if(KJID.ne.0) write(IPR,'(3x,' ID : ',i5)') KJID
516:  c
517:  >>>> LABEL <<<<
518:  c
519:  if ( TLABEL.ne.' ' )
520:    & write(IPR,'(3x,' LABEL : <'',a,'>'') TLABEL(:LLABEL)
521:  c
522:  >>>> METHOD <<<<
523:  c
524:  if ( METHOD.ne.' ' )

```

src/shared/ptinp.f

```

521:      &          write(IPR,'(3x,'' METHOD : <'' ,a,'>'' )' ) METHOD(:LMTHD)
522: c
523: c >>>> FRACTIONAL-CHANGE <<<<
524:      LSSS = LRWKDA
525:      LEEE = LSSS + MAXDA - 1
526:      if ( KJFRAC.ne.0 ) write(IPR,'(3x,'' FRACTIONAL CHANGE : ''
527:      &          /10(e13.5) )' ) (A(i),i=LSSS,LEEE)
528: c
529: c >>>> PERTURBED REGION <<<<
530:      LSSS = LIWKRG
531:      LEEE = LSSS + NREG - 1
532:      if ( KJPTTR.ne.0 ) write(IPR,'(3x,'' PERTURBED REGION : ''
533:      &          /10(i5) )' ) (IA(i),i=LSSS,LEEE)
534: c
535: c >>>> PERTURBED NUCLIDE <<<<
536:      LSSS = LIWKNU
537:      LEEE = LSSS + NUC - 1
538:      if ( KJPTNU.ne.0 ) write(IPR,'(3x,'' PERTURBED NUCLIDE : ''
539:      &          /10(i5) )' ) (IA(i),i=LSSS,LEEE)
540:
541:      if ( KJPART .eq. 1 .or. KJPART .ne. 1 ) then
542:      if ( JPART .eq. KJPHOT ) then
543:      write(IPR,*) 'XXX($PERT) option "PHOTON" is not ',
544:      &          ' available.'
545:      call CNTERR( 'FATAL' )
546:      NERR = NERR + 1
547:      end if
548:      end if
549: c
550:      if ( KJEVNT.eq.1 ) then
551:      if ( JEVNT.eq.KJTRAC ) then
552:      write(IPR,*) 'XXX(PTINP) Sub-option "TRACK" ',
553:      &          'for "EVENT" option is not available.'
554:      call CNTERR( 'FATAL' )
555:      NERR = NERR + 1
556:      end if
557:      if ( JEVNT.eq.KJANAC ) then
558:      write(IPR,*) 'XXX(PTINP) Sub-option "ANA-COL" ',
559:      &          'for "EVENT" option is not available.'
560:      call CNTERR( 'FATAL' )
561:      NERR = NERR + 1
562:      end if
563:      if ( JEVNT.eq.KJSURF ) then
564:      write(IPR,*) 'XXX(PTINP) Sub-option "SURF" ',
565:      &          'for "EVENT" option is not available.'
566:      call CNTERR( 'FATAL' )
567:      NERR = NERR + 1
568:      end if
569:      if ( JEVNT.eq.KJPOIN ) then
570:      write(IPR,*) 'XXX(PTINP) Sub-option "POINT" ',
571:      &          'for "EVENT" option is not available.'
572:      call CNTERR( 'FATAL' )
573:      NERR = NERR + 1
574:      end if
575:      else
576:      if( JPTYP.ne.KJBEFF ) then
577:      write(IPR,*) '!!!(PTINP) Option "EVENT" is not ',
578:      &          'specified.'
579:      call CNTERR( 'WARNING' )
580:      end if
581:      end if
582: c
583:      if ( KJPTYP .eq. 1 .or. KJPTYP .ne. 1 ) then
584:      if ( JPTYP .eq. KJFLUX ) then
585:      write(IPR,*) 'XXX($PERT) option "FLUX" is not ',

```

```

586:      &          ' available.'
587:      call CNTERR( 'FATAL' )
588:      NERR = NERR + 1
589:      end if
590:      if ( JPTYP .eq. KJMACR ) then
591:      write(IPR,*) 'XXX($PERT) option "MACRO" is not ',
592:      &          ' available.'
593:      call CNTERR( 'FATAL' )
594:      NERR = NERR + 1
595:      end if
596:      if ( JPTYP .eq. KJMICR ) then
597:      write(IPR,*) 'XXX($PERT) option "MICRO" is not ',
598:      &          ' available.'
599:      call CNTERR( 'FATAL' )
600:      NERR = NERR + 1
601:      end if
602:      end if
603: c
604:      if ( KJPARA.eq.1 ) then
605:      if ( JPARA .eq. KJTEMP ) then
606:      write(IPR,*) 'XXX($PERT) sub-option "TEMP" ',
607:      &          'for "PARAMETER" option is not ',
608:      &          ' available.'
609:      call CNTERR( 'FATAL' )
610:      NERR = NERR + 1
611:      end if
612:      if ( JPARA .eq. KJNUMB ) then
613:      if ( KJPTNU .eq. 0 ) then
614:      write(IPR,*) 'XXX($PERT) option "PERT-NUC*"',
615:      &          ' is not specified ',
616:      &          'for "PARAMETER(NUMBER)" option.'
617:      call CNTERR( 'FATAL' )
618:      NERR = NERR + 1
619:      end if
620:      end if
621:      if ( JPARA .eq. KJDENS ) then
622:      if ( KJPTNU .eq. 1 ) then
623:      write(IPR,*) '!!!($PERT) option "PERT-NUC*"',
624:      &          ' is ignored ',
625:      &          'for "PARAMETER(DENSITY)" option.'
626:      write(IPR,*) ' All nuclides are used',
627:      &          ' for PERTURBATION-CALCULATION.'
628:      call CNTERR( 'WARNING' )
629:      end if
630:      do 570 KNUC = 1 , NUC
631:      IA(LIWKNU+KNUC-1) = 1
632:      570 continue
633:      end if
634:      else
635:      if( JPTYP.ne.KJBEFF ) then
636:      write(IPR,*) 'XXX(PTINP) Option "PARAMETER" is not',
637:      &          ' specified.'
638:      call CNTERR( 'FATAL' )
639:      NERR = NERR + 1
640:      end if
641:      end if
642: c
643:      if ( KJMTHD.eq.1 ) then
644:      if ( JMTHD .ne. KJCORR .and. JMTHD .ne. KJDIFF ) then
645:      write(IPR,*) 'XXX($PERT) sub-option "METHOD," ',
646:      &          'for "METHOD" option is not',
647:      &          ' available.'
648:      call CNTERR( 'FATAL' )
649:      NERR = NERR + 1
650:      end if

```

src/shared/ptinp.f

```

651:         else
652:             if( JPTYP.ne.KJBEFF ) then
653:                 write(IPR,*) 'XXX(PTINP) Option "METHOD" is not',
654:                     & ' specified.'
655:                 call CNTERR( 'FATAL' )
656:                 NERR = NERR + 1
657:             end if
658:         end if
659: c
660:         if ( KJPTTR.eq.0 .and. JPTYP.ne.KJBEFF ) then
661:             write(IPR,*) 'XXX(PTINP) Option "PERT-REG*" is not',
662:                 & ' specified.'
663:             call CNTERR( 'FATAL' )
664:             NERR = NERR + 1
665:         end if
666: c
667:         if ( KJFRAC.eq.1 ) then
668:             if ( KJMTHTD.eq.1 .and. JMTHTD.eq.KJCORR ) then
669:                 if ( NJFRAC.gt.1 ) then
670:                     write(IPR,*) '!!!(PTINP) Option "FRAC*-CHAN"',
671:                         & ' specified more than one data',
672:                         & ' for "METHOD(CORRELATED)" option.',
673:                         & ' First data is used.'
674:                     call CNTERR( 'WARNING' )
675:                 end if
676:             end if
677:         else
678:             if( JPTYP.ne.KJBEFF ) then
679:                 write(IPR,*) 'XXX(PTINP) Option "FRAC*-CHAN"',
680:                     & ' is not specified.'
681:                 call CNTERR( 'FATAL' )
682:                 NERR = NERR + 1
683:             end if
684:         end if
685: c
686:         write(IUB) JIDPT, JMTHTD, JNGSP, JEVNT, JPART, JPTYP,
687:             & JPARA, JMICN, JXTYP
688:         write(IUB) (A(LRWKDA+KDA-1),KDA=1,MAXDA)
689:         write(IUB) (IA(LIWKRG+IREG-1),IREG=1,NREG)
690:         write(IUB) (IA(LIWKNU+KNUC-1),KNUC=1,NUC)
691:     end if
692: c
693: c ... set flags to check certain input item is input or not...
694: c
695:         KJID = 0
696:         TLABEL = ' '
697:         METHOD = ' '
698:         KJPTTR = 0
699:         KJPTNU = 0
700:         JIDPT = 0
701:         JMTHTD = 0
702:         JNGSP = 10
703:         JEVNT = KJTRAC
704:         JPART = KJNEUT
705:         JPTYP = KJKEFF
706:         JPARA = 0
707: c
708:         JMICN = 0
709:         JXTYP = 0
710: c
711:         KJEVNT = 0
712:         KJPART = 0
713:         KJPTYP = 0
714:         KJPARA = 0
715:         KJMTHTD = 0

```

```

716:         KJFRAC = 0
717:         NJFRAC = 0
718: c
719: c ... terminate input here
720: c
721:         if ( VNAME(:NLEN).eq.'$END' ) go to 140
722: c
723: c ... increment tally # and put header label in temporary packet.
724: c
725:         ITAL = ITAL + 1
726:         write(IPR,7020) ITAL
727: 7020 format(/3X,'=== Input perturbation No. ',I3,' ===/')
728: c
729: c === invalid data ===
730: c
731:         else
732:             write(IPR,*) 'XXX($PERT) Data <', VNAME(:NLEN), '> is not',
733:                 & ' acceptable as perturbation input data.'
734:             write(IPR,*) ' Perturbation No. ', ITAL, ' ID(', KJID, ') '
735:             write(IPR,*) ' Label <', TLABEL, '>'
736:             call CNTERR( 'FATAL' )
737:             NERR = NERR + 1
738:             call FPROBE( IIP, ' ', CWRK )
739:             if ( IIP.ne.0.and.CWRK(1:1).eq.'(' ) then
740:                 call DMREAD( VNAME(:NLEN), NA, NB, IEND )
741:                 if ( IEND.ne.0 ) go to 180
742:             end if
743:         end if
744:         go to 100
745: c
746: C=====
747: C ===== end of $PERT block
748: C=====
749: c
750: 140 continue
751: c
752:         call RESET
753: c
754: c
755:         if ( NERR.gt.0 ) then
756:             write(IPR,*) 'XXX Error found in $PERT block.',
757:                 & ' Skip processing.'
758:             call CNTERR( 'FATAL' )
759:             return
760:         end if
761: c
762:         if ( ITAL.eq.0 ) then
763:             write(IPR,*) '<<MESSAGE>> No perturbation data is specified',
764:                 & ' in $PERT block.'
765:             call CNTERR( 'MESSAGE' )
766:             return
767:         end if
768: c
769:         if ( NUCPT.eq.0 ) then
770:             write(IPR,*) '<<MESSAGE>> No perturbed nuclide is specified',
771:                 & ' in $PERT block.'
772:             call CNTERR( 'MESSAGE' )
773:             return
774:         end if
775: c
776:         NPTDS = IPTDS
777:         NPTCS = IPTCS
778:         NPTBE = IPTBE
779: c
780: c NPTDS : Number of direct tallies for differential operator sampling.

```

src/shared/ptinp.f

```
781: c NPTCS : Number of direct tallies for correlated sampling.
782:
783:       NPTLEN = NPTDS + NPTCS
784:
785:       call KEEP( 'JPTOP', LJPTOP,MXPTOP*NPTLEN, 'I4', LAST, JDEBG )
786:       call KEEP( 'JPTTR', LJPTTR,  NREG*NPTLEN, 'I4', LAST, JDEBG )
787:       call KEEP( 'JPTNU', LJPTNU,   NUC*NPTLEN, 'I4', LAST, JDEBG )
788:       call KEEP( 'DELA', LDELA,  MAXDA*NPTLEN,'R4D', LAST, JDEBG )
789:
790:       call RWIND( IUB )
791:
792:       do 580 IPT = 1, NPTLEN
793:         LPTR = LJPTOP + MXPTOP*(IPT-1)
794:         read(IUB) (IA(LPTR+III-1),III=1,MXPTOP)
795:         LPTR = LDELA + MAXDA*(IPT-1)
796:         read(IUB) (A(LPTR+KDA-1),KDA=1,MAXDA)
797:         LPTR = LJPTTR + NREG*(IPT-1)
798:         read(IUB) (IA(LPTR+IREG-1),IREG=1,NREG)
799:         LPTR = LJPTNU + NUC*(IPT-1)
800:         read(IUB) (IA(LPTR+KNUC-1),KNUC=1,NUC)
801:       580 continue
802:
803: c ===== post processing of input data =====
804:
805: c       call KEEP('JPTID', LJPTID, NPTLEN, 'I4', LAST, JDEBG)
806: c       do IPT = 1, NPTLEN
807: c         LPTR = LJPTOP + MXPTOP*(IPT-1) ! pointer to 'ID'
808: c         IA(LJPTID+IPT-1) = IA(LPTR)    ! set JPTID array
809: c       end do
810:
811:       return
812: c
813: c ===== error messages ===
814: c
815: c       150 continue
816: c       write(IPR,7040) VNAME(:NLEN)
817: c       7040 format(/1X,'XXX($PERT) Input error on data <',A,'>')
818: c       call PRSTOP( 1, 'Data input error in $PERT block.' )
819: c       stop 888
820: c
821: c       160 continue
822: c       write(IPR,7060) VNAME(:NLEN), NA, NWORK
823: c       7060 format(/1X,'XXX($PERT) size of data in item <',A,'> exceeds',
824: c         &      ' that of a working area for input:/1X,
825: c         &      ' (num. of data ',I6,' > limit =',I6,')'/1X,
826: c         &      ' Number of data is invalid or you need',
827: c         &      ' larger working memory size ( "NWORK" after $PERT ).')
828: c       call PRSTOP( 1, 'Data size overflow in $PERT block.' )
829: c       stop 888
830: c
831: c       170 continue
832: c       write(IPR,7080)
833: c       7080 format(/1X,'XXX Unexpected end of input data during reading',
834: c         &      ' "$PERT" block.')
835: c       call PRSTOP( 1, 'End of input data in "$PERT" block.' )
836: c       stop 888
837: c
838: c       180 write(IPR,7100)
839: c       7100 format(1X,'XXX Unexpected end of data or data read error ',
840: c         &      'during skipping unnecessary or invalid data ($PERT)')
841: c       call PRSTOP( 1, 'ERROR DURING SKIPPING PERTURBATION INPUT DATA.' )
842: c       stop 888
843: c
844:       end
```


src/shared/putv.f

```
1:      subroutine PUTV( A,      N,      V )
2: C=====
3: C PURPOSE:  Put value V in array A
4: C CALLED FROM:  INTRO
5: C=====
6:      real A(N), V
7:      do 100 I = 1, N
8:          A(I) = V
9:      100 continue
10:     return
11: end
12: C
13:     subroutine PUTVI( IA,      N,      IV )
14:     integer IA(N), IV
15:     do 100 I = 1, N
16:         IA(I) = IV
17:     100 continue
18:     return
19: end
20: C
21:     subroutine PUTVD( DA,      N,      DV )
22:     real*8 DA(N), DV
23:     do 100 I = 1, N
24:         DA(I) = DV
25:     100 continue
26:     return
27: end
```

src/shared/r4prnt.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine R4PRNT( VNAME, R4,      N1,      N2,      HD1, HD2, IPRT )
3:   C
4:   C   JAERI MONTE CARLO CODE UTILITY
5:   C
6:   C=====
7:   C PURPOSE: PRINT OUT TWO- OR ONE-DIMENSIONAL REAL   ARRAY
8:   C CALLS: (NONE)
9:   C=====
10:  real R4(N1,N2)
11:  character VNAME*(*), HD1*(*), HD2*(*)
12:  C ... local ...
13:  character HEAD1*8, HEAD2*8
14:  C
15:  HEAD1 = HD1
16:  HEAD2 = HD2
17:  C
18:  ML      = 10
19:  LVN     = INDEX(VNAME,' ') - 1
20:  if ( LVN.lt.0 ) LVN = LEN(VNAME)
21:  C
22:  if ( N2.gt.1 ) then
23:  C-----
24:  C TWO DIMENSIONAL ARRAY
25:  C-----
26:  write(IPRT,7000) VNAME, N1, N2
27:  7000 format(/1X,'<<< ARRAY ',A,'(I,J) (I=1,',I4,',J=1,',I4,',') >>>')
28:  C
29:  C ..... PRINT UPTO ML PARAMETERS PER LINE .....
30:  C
31:  do 130 K = 1, N1, ML
32:    K2      = MIN(K+ML-1,N1)
33:  C
34:    write(IPRT,7020) ('-----',KK=K,K2)
35:    7020 format(1X,'-----',10A12)
36:  C
37:  C
38:    write(IPRT,7040) HEAD1, HEAD2, (KK,KK=K,K2)
39:    7040 format(1X,8X,'| ',A8/1X,A8,'| ',10(I8,4X))
40:  C
41:  C
42:    write(IPRT,7060) ('-----',KK=K,K2)
43:    7060 format(1X,'-----+',10A12)
44:    write(IPRT,7100) 1, (R4(KK,1),KK=K,K2)
45:  C
46:  C
47:    JJ      = 1
48:    do 120 IJ = 2, N2
49:      do 100 KK = K, K2
50:        if ( R4(KK,IJ).ne.R4(KK,JJ) ) go to 110
51:    100 continue
52:      go to 120
53:    C
54:    110 if ( IJ.gt.JJ+2 ) write(IPRT,7080) HEAD2, JJ + 1, IJ - 1,
55:    &    HEAD2, JJ
56:    if ( IJ.eq.JJ+2 ) write(IPRT,7100) IJ - 1,
57:    &    (R4(KK,IJ-1),KK=K,K2)
58:    write(IPRT,7100) IJ, (R4(KK,IJ),KK=K,K2)
59:    JJ      = IJ
60:  120 continue
61:    if ( JJ.lt.N2-1 ) write(IPRT,7080) HEAD2, JJ + 1, N2,
62:    &    HEAD2, JJ
63:    if ( JJ.eq.N2-1 ) write(IPRT,7100) N2, (R4(KK,N2),KK=K,K2)
64:    write(IPRT,7020) ('-----',KK=K,K2)
65:  130 continue

```

```

66:  7080 format(1X,' ... ',| ( ',A8,I5,' TO ',I5,' ARE SAME AS ',A8,
67:  &    I5,' ')')
68:  7100 format(1X,I7,' | ',1P10E12.5)
69:  C
70:  C-----
71:  C ONE DIMENSIONAL
72:  C-----
73:  C
74:  else
75:  C
76:    do 140 K = 1, N1, ML
77:      K2      = MIN(K+ML-1,N1)
78:  C
79:      write(IPRT,7020) ('-----',KK=K,K2)
80:  C
81:      write(IPRT,7120) HEAD1, (KK,KK=K,K2)
82:      7120 format(1X,A8,'| ',10(I8,4X))
83:  C
84:      write(IPRT,7060) ('-----',KK=K,K2)
85:  C
86:      write(IPRT,7140) VNAME(1:LVN), (R4(KK,1),KK=K,K2)
87:      7140 format(1X,' ',A6,' | ',1P10E12.5)
88:  C
89:      write(IPRT,7020) ('-----',KK=K,K2)
90:    140 continue
91:    end if
92:    return
93:  end
94: c##<2007/03/14:PN3:
95:  C
96:  C
97:  subroutine R4I4PRNT( VNAME, R4,      N1,      N2,      HD0, HD1, HD2,
98:  &    IPRT )
99:  C
100:  C   JAERI MONTE CARLO CODE UTILITY
101:  C
102:  C=====
103:  C PURPOSE: PRINT OUT TWO- OR ONE-DIMENSIONAL INTEGER ARRAY WITH FIRST
104:  C REAL DATA.
105:  C CALLS: (NONE)
106:  C CODED FOR PHOTONUCLEAR on 14 Mar. 2007. (K.Kosako)
107:  C=====
108:  real R4(N1,N2)
109:  character VNAME*(*), HD1*(*), HD2*(*), HD0*(*)
110:  C ... local ...
111:  integer iidata(30)
112:  character HEAD1*8, HEAD2*8, HEAD0*8
113:  C
114:  HEAD1 = HD1
115:  HEAD2 = HD2
116:  HEAD0 = HD0
117:  C
118:  ML      = 22
119:  LVN     = INDEX(VNAME,' ') - 1
120:  if ( LVN.lt.0 ) LVN = LEN(VNAME)
121:  C
122:  if ( N2.gt.1 ) then
123:  C-----
124:  C TWO DIMENSIONAL ARRAY
125:  C-----
126:  write(IPRT,7000) VNAME, N1, N2
127:  7000 format(/1X,'<<< ARRAY ',A,'(I,J) (I=1,',I4,',J=1,',I4,',') >>>')
128:  C
129:  C ..... PRINT UPTO ML PARAMETERS PER LINE .....
130:  C

```

src/shared/r4prnt.f

```

131:      do 130 K = 1, N1, ML
132:         K2      = MIN(K+ML-1,N1)
133: C
134:         if ( K.eq.1 ) then
135:            do KK = 2, K2
136:               IIDATA(KK) = R4(KK,1)
137:            end do
138:            write(IPRT,7020) ('-----',KK=2,K2)
139:            write(IPRT,7041) HEAD1, HEAD0, (KK-1,KK=2,K2)
140:            write(IPRT,7060) ('-----',KK=2,K2)
141:            write(IPRT,7110) 1, R4(1,1), (IIDATA(KK),KK=2,K2)
142:        else
143:            K3      = 0
144:            do KK = K, K2
145:               K3      = K3 + 1
146:               IIDATA(K3) = R4(KK,1)
147:            end do
148:            write(IPRT,7020) ('-----',KK=K+1,K2)
149:            write(IPRT,7040) HEAD1, HEAD2, (KK-1,KK=K,K2)
150:            write(IPRT,7060) ('-----',KK=K+1,K2)
151:            write(IPRT,7120) 1, (IIDATA(KK),KK=1,K3)
152:        end if
153: 7020      format(1X,'-----','-----',21A5)
154: 7040      format(1X,8X,'| ',A8 /1X,A8,'| ',I12,21I5)
155: 7041      format(1X,8X,'| ',12X,A8 /1X,A8,'| ',4X,A8,21I5)
156: 7060      format(1X,'-----','-----',21A5)
157: 7110      format(1X,I7,'| ',1PE12.5,21I5)
158: 7120      format(1X,I7,'| ',I12,21I5)
159: C
160:         JJ      = 1
161:         do 120 IJ = 2, N2
162:            do 100 KK = K, K2
163:               if ( R4(KK,IJ).ne.R4(KK,JJ) ) go to 110
164: 100          continue
165:            go to 120
166: C
167: 110          if ( IJ.gt.JJ+2 ) write(IPRT,7080) HEAD2, JJ + 1, IJ - 1,
168: &            HEAD2, JJ
169:            if ( IJ.eq.JJ+2 ) then
170:               K3      = 0
171:               do KK = K, K2
172:                  K3      = K3 + 1
173:                  IIDATA(K3) = R4(KK,IJ-1)
174:               end do
175:               if ( K.eq.1 ) then
176:                  write(IPRT,7110) IJ-1, R4(1,IJ-1),
177: &                    (IIDATA(KK),KK=2,K3)
178:               else
179:                  write(IPRT,7120) IJ-1, (IIDATA(KK),KK=1,K3)
180:               end if
181:            end if
182:            K3      = 0
183:            do KK = K, K2
184:               K3      = K3 + 1
185:               IIDATA(K3) = R4(KK,IJ)
186:            end do
187:            if ( K.eq.1 ) then
188:               write(IPRT,7110) IJ, R4(1,IJ), (IIDATA(KK),KK=2,K3)
189:            else
190:               write(IPRT,7120) IJ, (IIDATA(KK),KK=1,K3)
191:            end if
192:            JJ      = IJ
193: 120          continue
194:            if ( JJ.lt.N2-1 ) write(IPRT,7080) HEAD2, JJ + 1, N2,
195: &            HEAD2, JJ

```

```

196:         if ( JJ.eq.N2-1 ) then
197:            K3      = 0
198:            do KK = K, K2
199:               K3      = K3 + 1
200:               IIDATA(K3) = R4(KK,N2)
201:            end do
202:            if ( K.eq.1 ) then
203:               write(IPRT,7110) N2, R4(1,N2), (IIDATA(KK),KK=2,K3)
204:            else
205:               write(IPRT,7120) N2, (IIDATA(KK),KK=1,K3)
206:            end if
207:         end if
208:         write(IPRT,7020) ('-----',KK=K+1,K2)
209: 130      continue
210: 7080      format(1X,' ... ',| ( ',A8,I5,' TO ',I5,' ARE SAME AS ',A8,
211: &            I5,' )')
212: 7100      format(1X,I7,'| ',1PE12.5)
213: C
214: C-----
215: C ONE DIMENSIONAL
216: C-----
217: C
218:        else
219: C
220:         do 140 K = 1, N1, ML
221:            K2      = MIN(K+ML-1,N1)
222:            write(IPRT,7020) ('-----',KK=K+1,K2)
223:            if ( K.eq.1 ) then
224:               write(IPRT,7131) HEAD1, HEAD0, (KK-1,KK=K,K2)
225:            else
226:               write(IPRT,7130) HEAD1, (KK-1,KK=K,K2)
227:            end if
228:            write(IPRT,7060) ('-----',KK=K+1,K2)
229:            K3      = 0
230:            do KK = K, K2
231:               K3      = K3 + 1
232:               IIDATA(K3) = R4(KK,1)
233:            end do
234:            if ( K.eq.1 ) then
235:               write(IPRT,7141) VNAME(1:LVN), R4(1,1),
236: &                    (IIDATA(KK),KK=2,K3)
237:            else
238:               write(IPRT,7140) VNAME(1:LVN), (IIDATA(KK),KK=1,K3)
239:            end if
240:            write(IPRT,7020) ('-----',KK=K+1,K2)
241: 7130      format(1X,A8,'| ',I12,21I5)
242: 7131      format(1X,A8,'| ',4X,A8,21I5)
243: 7140      format(1X,' ',A6,'| ',I12,21I5)
244: 7141      format(1X,' ',A6,'| ',1PE12.5,21I5)
245: 140          continue
246:        end if
247:        return
248:      end
249: c##>

```

src/shared/r8prnt.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine R8PRNT( VNAME, R8,      N1,      N2,      HD1, HD2, IPRT )
3:      C
4:      C   JAERI MONTE CARLO CODE UTILITY
5:      C
6:      C=====
7:      C   PURPOSE: PRINT OUT TWO- OR ONE-DIMENSIONAL REAL*8 ARRAY
8:      C   CALLS: (NONE)
9:      C-----
10:     C arguments (i=input. o=output, w=work)
11:     C i VNAME : data name string if any
12:     C i R4(N1,N2) : real*8 array to be printed
13:     C i N1, N2 : dimension size
14:     C i HD1, HD2 : label for 1st and 2nd dimension.
15:     C i IPRT : printout I/O unit.
16:     C=====
17:     real*8 R8(N1,N2)
18:     character VNAME*(*), HD1*(*), HD2*(*)
19:     C ... local ...
20:     character HEAD1*8, HEAD2*8
21:     C
22:     HEAD1 = HD1
23:     HEAD2 = HD2
24:     C
25:     ML      = 10
26:     LVN      = INDEX(VNAME,' ') - 1
27:     if ( LVN.lt.0 ) LVN = LEN(VNAME)
28:     C
29:     if ( N2.gt.1 ) then
30:     C-----
31:     C   TWO DIMENSIONAL ARRAY
32:     C-----
33:     write(IPRT,7000) VNAME, N1, N2
34:     7000    format(/1X,'<<< ARRAY ',A,'(I,J) (I=1,',I4,',J=1,',I4,',>>>')
35:     C
36:     C   .... PRINT UPTO ML PARAMETERS PER LINE ....
37:     C
38:     do 130 K = 1, N1, ML
39:     K2      = MIN(K+ML-1,N1)
40:     C
41:     write(IPRT,7020) ('-----',KK=K,K2)
42:     7020    format(1X,'-----',10A12)
43:     C
44:     C
45:     write(IPRT,7040) HEAD1, HEAD2, (KK,KK=K,K2)
46:     7040    format(1X,8X,'| ',A8/1X,A8,'|',10(I8,4X))
47:     C
48:     C
49:     write(IPRT,7060) ('-----',KK=K,K2)
50:     7060    format(1X,'-----+',10A12)
51:     write(IPRT,7100) 1, (R8(KK,1),KK=K,K2)
52:     C
53:     C
54:     JJ      = 1
55:     do 120 IJ = 2, N2
56:     do 100 KK = K, K2
57:     if ( R8(KK,IJ).ne.R8(KK,JJ) ) go to 110
58:     100      continue
59:     go to 120
60:     C
61:     110      if ( IJ.gt.JJ+2 ) write(IPRT,7080) HEAD2, JJ + 1, IJ - 1,
62:     &          HEAD2, JJ
63:     if ( IJ.eq.JJ+2 ) write(IPRT,7100) IJ - 1,
64:     &          (R8(KK,IJ-1),KK=K,K2)
65:     write(IPRT,7100) IJ, (R8(KK,IJ),KK=K,K2)

```

```

66:     JJ      = IJ
67:     120      continue
68:     if ( JJ.lt.N2-1 ) write(IPRT,7080) HEAD2, JJ + 1, N2,
69:     &          HEAD2, JJ
70:     if ( JJ.eq.N2-1 ) write(IPRT,7100) N2, (R8(KK,N2),KK=K,K2)
71:     write(IPRT,7020) ('-----',KK=K,K2)
72:     130      continue
73:     7080    format(1X,' ... ','| ( ',A8,I5,' TO ',I5,' ARE SAME AS ',A8,
74:     &          I5,' )')
75:     7100    format(1X,I7,' |',1P,10E12.5)
76:     C
77:     C-----
78:     C   ONE DIMENSIONAL
79:     C-----
80:     C
81:     else
82:     C
83:     do 140 K = 1, N1, ML
84:     K2      = MIN(K+ML-1,N1)
85:     C
86:     write(IPRT,7020) ('-----',KK=K,K2)
87:     C
88:     write(IPRT,7120) HEAD1, (KK,KK=K,K2)
89:     7120    format(1X,A8,'|',10(I8,4X))
90:     C
91:     write(IPRT,7060) ('-----',KK=K,K2)
92:     C
93:     write(IPRT,7140) VNAME(1:LVN), (R8(KK,1),KK=K,K2)
94:     7140    format(1X,' ',A6,'|',1P,10E12.5)
95:     C
96:     write(IPRT,7020) ('-----',KK=K,K2)
97:     140      continue
98:     end if
99:     return
100:    end

```

src/shared/rdsubp.f

```
1:      SUBROUTINE RDSUBP( VNAME, SNAME, TYPE, N, A , NA )
2: C
3: C      MVP/GMVP  UTILITY
4: C
5: C=====
6: C      PURPOSE:  INPUT DATA HAVING NAMED SUBPARAMETERS
7: C
8: C      VNAME :  DATA NAME  (ALREADY DETERMINED)
9: C      SNAME(N) : SUBPARAMETER NAMES   ( XX   OR  XX(K) )
10: C      TYPE(N) : SUBPARAMETER TYPES   ( 'I','R4','R8','C' )
11: C
12: C      VNAME #(      SNAME1( DATA ... )
13: C                  SNAME2( DATA ... )
14: C                  .....
15: C                  )#
16: C=====
17: C
18: C      CHARACTER*(*)  VNAME, SNAME(N), TYPE(N)
19: C      REAL           A(*)
20: C
21: C
22: C
23: C      RETURN
24: C      END
```

src/shared/realvr.f

```

1:      subroutine REALVR( WHAT, X,      KDIM1, K1,      N1,      N2,      AVG,
2:      &                  SIGA, SIGR,  CA,      CR,      NMXLAGEPS,
3:      &                  NEITER,IPR,  IDEBG, IERR )
4: C=====
5: C Purpose: estimate "real variance" of a sample
6: C
7: C Reference: "Error Estimations and Their Biases in Monte Carlo
8: C             Eigenvalue Calculations"
9: C             by T.Ueki, T.Mori, M.Nakagawa
10: C             Nuclear Science and Engineering: [125],1-11(1997)
11: C
12: C-----
13: C Arguments ( i=input, o=output, w=work )
14: C
15: C i WHAT : a character string showing what is analyzed etc.
16: C
17: C i X(KDIM1,N2) : sample values
18: C i KDIM1 : first dimension size of X
19: C
20: C i K1 : estimate real variance of X(K1,j)
21: C i N1, N2 : estimate real variance for step (cycle) j=N1,N2.
22: C
23: C o AVG : average
24: C o SIGA : apparent variance
25: C o SIGR : estimated real variance
26: C o CA(NMXLAGEPS) : apparent covariance of lag i=1,NMXLAGEPS.
27: C o CR(NMXLAGEPS) : real covariance of lag i=1,NMXLAGEPS.
28: C i NMXLAGEPS : maximum of step(cycle) lag considered in estimation.
29: C i EPS : convergence criterion of iterative estimation
30: C      Positive value : relative deviation
31: C      Negative : absolute deviation (-EPS)
32: C i NEITER : maximum iteration count.
33: C i IPR : message printout I/O unit.
34: C      Do not print any message if negative.
35: C o IDEBG : print out debugging information if non zero.
36: C o IERR : error flag (returns non-zero if error occurred)
37: C=====
38:      implicit real*8(A-H,O-Z)
39: C
40:      character*(*) WHAT
41:      real*8 X(KDIM1,N2)
42:      real*8 SIGA
43:      real*8 SIGR
44:      real*8 CA(NMXLAGEPS)
45:      real*8 CR(NMXLAGEPS)
46:      real*8 EPS
47:      integer NEITER
48: C
49: C-----
50: C
51: CHECK %%%%%%%%%%
52: c write(ipr,*) '%%% n1,n2 ',n1,n2
53: c do i=n1,n2
54: c   aaa = 0.0d0
55: c   do j=i,n2
56: c     aaa = aaa + X(K1,j)
57: c   end do
58: c   aaa = aaa / (n2-i+1)
59: c   write(ipr, '(1x,%% ',i3,%% ',d12.5,d12.5)') I,X(K1,I),aaa
60: c end do
61: C %%%%%%%%%%
62: C
63:      SIGR = 0.0D0
64:      IERR = 0
65: C

```

```

66: C ... number of samples analyzed
67: C
68:      NN = N2 - N1 + 1
69: C
70:      if ( NN.lt.2 ) then
71:      IERR = IERR + 1
72:      if ( IPR.ge.0 ) write(IPR,*) 'XXX(REALVR) In calculation of ',
73:      & ' "real variance" for "', WHAT(:ICLEN2(WHAT)), "','',
74:      & ' number of steps N (=, NN, ') is less than 2'
75:      end if
76: C
77:      if ( NN.lt.NMXLAGEPS+1 ) then
78:      IERR = IERR + 1
79:      if ( IPR.ge.0 ) write(IPR,*) 'XXX(REALVR) In calculation of ',
80:      & ' "real variance" for "', WHAT(:ICLEN2(WHAT)), "','',
81:      & ' number of steps N (=, NN, ') is less than NMXLAGEPS+1 (=,
82:      & NMXLAGEPS + 1, ') '
83:      end if
84: C
85:      if ( K1.lt.1 .or. K1.gt.KDIM1 ) then
86:      IERR = IERR + 1
87:      if ( IPR.ge.0 ) write(IPR,*) 'XXX(REALVR) In calculation of ',
88:      & ' "real variance" for "', WHAT(:ICLEN2(WHAT)), "','',
89:      & ' index of first dimension (K1=, K1, ') is out of range.'
90:      end if
91: C
92:      if ( IERR.gt.0 ) return
93: C
94:      if ( IPR.ge.0 ) then
95:      write(IPR,7000) WHAT(:ICLEN2(WHAT))
96:      7000 format(/1X,' * real variance estimation for <',A,'>')
97:      end if
98: Check %%%%%%%%%%
99: c write(*,*) '%%% EPS ',eps
100: C %%%%%%%%%%
101:      DDEPS = 1.0D-6
102:      if ( EPS.ne.0.0D0 ) DDEPS = EPS
103: C
104:      if ( NEITER.eq.0 ) NEITER = 300
105: C
106: C-----
107: C calculate average, apparent variance and covariance
108: C-----
109: C
110:      AVG = 0.0D0
111:      A2 = 0.0D0
112:      do 100 I = N1, N2
113: Check %%%%%%%%%%
114: C write(*,*) '%%% X ',X(K1,I)
115: C %%%%%%%%%%
116:      AVG = AVG + X(K1,I)
117:      A2 = A2 + X(K1,I)*X(K1,I)
118: 100 continue
119: C
120:      AVG = AVG/DBLE(NN)
121: C
122: C A2/DBLE(NN)-AVG*AVG may be negative if all X(KI,*) have
123: C the same value
124: C
125:      SIGA = A2/DBLE(NN)-AVG*AVG
126:      if ( SIGA.lt.0.0d0 ) SIGA = 0.0D0
127: C
128:      SIGA = SIGA /DBLE(NN-1)
129: C
130:      do 120 LAG = 1, NMXLAGEPS

```

src/shared/realvr.f

```

131:      CR(LAG) = 0.0D0
132:      CA(LAG) = 0.0D0
133:      do 110 I = N1, N2 - LAG
134:          CA(LAG) = CA(LAG) + (X(K1,I)-AVG)*(X(K1,I+LAG)-AVG)
135:      110 continue
136:      CA(LAG) = CA(LAG) /DBLE(NN-LAG-1)
137:      120 continue
138: C
139: C-----
140: C calculate first guess of real variance
141: C-----
142: C
143:      SCOV      = 0.0D0
144:      do 130 LAG = 1, NMXLAG
145:          SCOV      = SCOV + (NN-LAG)*CA(LAG)
146:      130 continue
147:      SIGR      = SIGA + SCOV/DBLE(NN*(NN-1))*2.0D0
148: C
149: C-----
150: C iteration
151: C by the "method 1" in NSE[125],1-11.
152: C-----
153: C
154:      DEV      = 0.0D0
155:      do 160 II = 1, NEITER
156:          SIGR0      = SIGR
157: C
158: C      ... Cov_real(lag) - Cov_app(lag) = sigr
159: C
160:      do 140 LAG = 1, NMXLAG
161:          CR(LAG) = CA(LAG) + SIGR
162:      140 continue
163: C
164: C      ... update sigr ...
165: C
166:      SCOV      = 0.0D0
167:      do 150 LAG = 1, NMXLAG
168:          SCOV      = SCOV + (NN-LAG)*CR(LAG)
169:      150 continue
170:      SIGR      = SIGA + SCOV/DBLE(NN*(NN-1))*2.0D0
171: C
172:      if ( DDEPS.gt.0.0D0 ) then
173:          if ( SIGR0.ne.0.0D0 ) then
174:              DEV      = (SIGR-SIGR0) /SIGR0
175:          else if ( SIGR.eq.0 ) then
176:              DEV      = 0.0D0
177:          else
178:              DEV      = 1.0
179:          end if
180:      else
181:          DEV      = SIGR - SIGR0
182:      end if
183: C
184:      if ( ABS(DEV).lt.ABS(DDEPS) ) go to 170
185: C
186:      160 continue
187: C
188:      170 continue
189:      if ( SIGR.le.0.0D0 ) then
190:          IERR      = IERR + 1
191:          SIGR0      = SIGR
192:          SIGR      = 0.0D0
193:      end if
194: C
195:      if ( IPR.ge.0 ) then

```

```

196:      if ( AVG.eq.0.0D0 ) then
197:          write(IPR,7020)
198:          return
199:      end if
200:      7020 format(1X,'!!! Calculated average is zero.')
201:      if ( SIGR0.le.0.0D0 ) then
202:          write(IPR,7040) SIGR0
203:      end if
204:      7040 format(1X,'!!! Estimated "real variance" is less than zero ',
205:      &          1P,E12.5,', reset to zero.')
206: C
207:      write(IPR,7060) AVG, SQRT(SIGR), SQRT(SIGR) /AVG*100D0,
208:      &          SQRT(SIGA), SQRT(SIGA) /AVG*100D0,
209:      &          SQRT(SIGR/MAX(SIGA,1.0D-20))
210:      7060 format(/3X,1P,E13.7,' +- ',E11.4,' ( ',0P,F7.4,
211:      &          '% ) (apparent dev. ',1P,E12.5,' ( ',0P,F7.4,'%))',/3X,
212:      &          ' deviation ratio(r/a) ',0P,F8.3)
213:      if ( IDEBG.ne.0 ) then
214:          write(IPR,7080) II, DEV
215:      7080 format(3X,' number of iteration ',I4,3X,' last deviation ',
216:      &          E12.5/3X,' Lag      apparent cov.      real cov.')
217:          do 180 LAG = 1, NMXLAG
218:              write(IPR,7100) LAG, CA(LAG), CR(LAG)
219:          180 continue
220:      7100 format(3X,I3,2E15.6)
221:          end if
222:      end if
223: C
224:      return
225:      end

```

src/shared/refzon.f

```

1: *VOCL TOTAL,SCALAR
2:      subroutine REFZON( SDA,  KZDA,  KZAA,  KZMAT, KSREF, NZDA,
3:      &                 NZONE, KKREF, NLTEMP,NREFS, JDEBG )
4: C=====
5: C PURPOSE: PREPARATION FOR REFLECTION
6: C CALLED IN: INTRO
7: C CALLS:
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C
11: C i SDA(*) : surface shape data
12: C i KZDA(2,NZDA), KZAA(*) : zone surface data
13: C i KZMAT(NZONE) : material # of zones
14: C o KSREF(NZDA) : reflective surface # for each surface
15: C o KKREF(2,*) : position in SDA and material # for reflective
16: C   surfaces.
17: C i NLTEMP : temporary memory size for
18: C o NSREF : number of reflective surfaces
19: C i JDEBG(*) : debug option
20: C-----
21:      real*8 SDA(*)
22:      integer KZDA(2,NZDA), KZAA(*), KZMAT(NZONE), KSREF(NZDA),
23:      &        KKREF(2,*)
24:      integer JDEBG(*)
25: C
26:      include 'INC/_IOUNIT'
27: C
28:      include 'INC/_PMLATT'
29:      include 'INC/_SPLATT'
30: C
31: C-----
32: C
33: C KSREF(I) : REFLECTIVE SURFACE # FOR EACH SURFACE
34: C KKREF(2,I) : POSITION IN SDA FOR I'TH REFLECTIVE SURFACE.
35: C   & MATERIAL #
36: C
37: C =====
38:      call HEADER( IPR, 'INFORMATION OF REFLECTOR' )
39: C =====
40: C
41: C
42: C ..... COUNT THE NUMBER OF REFLECTIVE SURFACES .....
43: C
44:      NREFS = 0
45:      do 130 K = 1, NZONE
46: CCCC if ( KZMAT(K).lt.-1000 ) then
47:      if ( KZMAT(K).lt.-1000.and..not.ISLATT(KZMAT(K)) ) then
48:          do 120 M = KZAA(K), KZAA(K+1) - 1
49:              KK = ABS(KZDA(1,M))
50:              do 100 I = 1, NREFS
51:                  if ( KK.eq.KKREF(1,I).and.KZMAT(K).eq.KKREF(2,I) )
52:                      & go to 110
53:              100 continue
54:
55:                  NREFS = NREFS + 1
56:                  KKREF(1,NREFS) = KK
57:                  KKREF(2,NREFS) = KZMAT(K)
58:                  I = NREFS
59:
60:              110 KSREF(M) = I
61:              120 continue
62:          end if
63:      130 continue
64: C
65: Ccccc IF(LIMIT-LAST.LT.4*NREFS) THEN

```

```

66:      if ( 4*NREFS.gt.NLTEMP ) then
67:          write(IMG,*) 'XXX Memory shortage in subroutine REFZON by ',
68:      & 4*NREFS - NLTEMP, ' words.'
69:          call PRSTOP( 1, 'MEMORY OVER.' )
70:          stop 999
71:      end if
72: C
73: C ..... MODIFY THE ORDER OF REFLECTIVE SURFACES SUCH THAT TYPES OF
74: C   SURFACES ARE IN INCREASING ORDER.
75:      do 140 I = 1, NREFS
76:          KKREF(1,NREFS+I) = I
77:      140 continue
78:
79:      do 160 I = 1, NREFS
80:          JFL = 0
81:          do 150 K = 1, NREFS - 1
82:              if ( SDA(KKREF(1,K)).gt.SDA(KKREF(1,K+1)) ) then
83:                  L = KKREF(1,K)
84:                  KKREF(1,K) = KKREF(1,K+1)
85:                  KKREF(1,K+1) = L
86:                  L = KKREF(2,K)
87:                  KKREF(2,K) = KKREF(2,K+1)
88:                  KKREF(2,K+1) = L
89:                  L = KKREF(1,NREFS+K)
90:                  KKREF(1,NREFS+K) = KKREF(1,NREFS+K+1)
91:                  KKREF(1,NREFS+K+1) = L
92:                  JFL = 1
93:              end if
94:          150 continue
95:          if ( JFL.eq.0 ) go to 170
96:      160 continue
97: C
98:      170 continue
99:      do 180 K = 1, NREFS
100:          KKREF(2,NREFS+KKREF(1,NREFS+K)) = K
101:      180 continue
102:      do 190 M = 1, NZDA
103:          if ( KSREF(M).ne.0 ) KSREF(M) = KKREF(2,NREFS+KSREF(M))
104:      190 continue
105: C
106:      write(IPR,7000) NREFS,
107:      & (K,INT(SDA(KKREF(1,K))),KKREF(1,K),KKREF(2,K),K=1,NREFS)
108:      7000 format(/1X,' == NUMBER OF REFLECTION SURFACES : ',I5//1X,
109:      & ' SURFACE# TYPE ( SDA ) MATERIAL '//
110:      & (1X,3X,I5,3X,4X,I2,6X,'('',I6,'')',5X,I6))
111:      if ( JDEBG(1).ne.0 ) then
112:          write(IPR,7020) (KSREF(I),I=1,NZDA)
113:      7020 format(/1X,' KSREF ARRAY '/(1X,20I4))
114:      end if
115: C
116: C .... KKREF(1,1) TO KKREF(2,NREFS) IS EFFECTIVE ....
117: C
118: Ccccc LAST = LAST + 2*NREFS
119:      return
120:      end

```


src/shared/regdat.f

```

1:      subroutine REGDAT(IOR,  VNAME, VNM,  VSUBF, A,    IA,
2:      &                    LIMIT, LAST, JNEUT, JPHOT, JDEBG, NREG,
3:      &                    NGP, JKPAR, KPSYM, KPLIM, KNGP,
4:      &                    NGROUP, NGP1,  NGP2,  LXIMP, LWKIL, LWSRV,
5:      &                    LWGTF, LWGTP, LRVOL, LTRVOL, LPSALP,
6:      &                    JTLLT, KMAT,  KREG,
7:      &                    KREGI, KCELI, NINPZ, ISPNM, ISUSP, KSUZN,
8:      &                    IRGSP, NEST,  NSPACE, NSUZON, NZONE, TNAMS,
9:      &                    NNAMES, ITRNM, NTRREG, IPTRG, LPTRG, KR,   NRRR0,
10: c##<2007/03/14:PN3:
11:      &                    NUCPN, NMTPN, LWGTPN)
12: c##>
13: C=<MVP>=====
14: C PURPOSE: INPUT REGION/TALLY-REGION-DEPENDENT ARRAY DATA
15: C          (single precision real only)
16: C
17: C          (NGROUP,NREG) ARRAYS : WKIL, WSRV, XIMP
18: c##<2007/03/14:PN3:
19: c##C          (NREG) ARRAY : WGTF, WGTP, RVOL
20: C          (NREG) ARRAY : WGTF, WGTP, WGTPN, RVOL
21: c##>
22: C          (NTRREG) ARRAY : TRVOL
23: C
24: C CALLED IN: INTRO
25: C CALLS : CNTERR,KEPV,CHREAD,CBSINC,DMREAD,RGFIND,R4READ,CKVAL4,
26: C          ATRANS
27: C-----
28: C arguments (i=input, o=output, w=work)
29: C
30: C i IOR : message printout I/O unit
31: C w KR(NRRR0) : working area (NRR0 must be max(NREG,NTRREG))
32: C=====
33:      character*(*) VNAME, VNM, VSUBF
34:      integer JDEBG(*)
35:      real A(LIMIT)
36:      integer IA(LIMIT)
37: C
38:      integer NGP(KPLIM)
39:      integer JKPAR(KPLIM)
40:      character*32 KPSYM(2,KPLIM)
41:      integer KNGP(KPLIM+1)
42: C
43: C ..... DATA FOR REGION NAME MATCHING .....
44: C
45:      integer KMAT(NINPZ), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ)
46:      integer ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
47:      &      KSUZN(NINPZ,2), IRGSP(2,NREG)
48:      include 'INC/_LNAM'
49:      character*(LNAM) TNAMS(NNAMES)
50:      integer ITRNM(NTRREG)
51:      integer IPTRG(NTRREG+1), LPTRG(*)
52: C
53: C
54:      integer KR(NRRR0)
55: C
56: C ..... LOCAL VARIABLE ...
57: C
58:      character LINE*72, MESSGE*96
59:      character*4 CHAR
60: C-----
61: C
62: C ..... number of region is not determined YET !!! ...
63: C
64:      if ( NREG.eq.0 ) then
65:          write(IOR,'(/1X,A,A)') 'XXX(REGDAT) Number of "REGION" (NREG)',

```

```

66:      &          ' is not determined before REGION dependent data input. '
67: c##<2007/03/14:PN3:
68: c##      write(IOR,*) '      Your input for < ', VNAME,
69: c##      &          '> must be placed ', 'after $GEOMETRY block.'
70: c##      call PRSTOP( 1, 'PROBLEM IN ORDER OF INPUT DATA.' )
71:      write(IOR,'(/1X,A,A,A)') '      Your input for < ', VNAME,
72:      &          '> must be placed after $GEOMETRY block.'
73:      call PRSTOP( 1, 'PROBLEM IN ORDER OF INPUT DATA.' )
74: c##>
75:      stop 888
76:      end if
77: C
78: C ..... number of energy group is not determined yet !!! ...
79: C
80:      if ( NGROUP.eq.0.and.(INDEX(VNAME,'WKIL').ne.0
81:      &      .or.INDEX(VNAME,'WSRV').ne.0.or.INDEX(VNAME,'XIMP').ne.0 ) )
82:      &      then
83: C
84:          write(IOR,'(/1X,A)')
85:      &          'XXX(REGDAT) Number of "energy bin" is not defined.'
86:          write(IOR,'(/1X,a,a,a,a)') '      Your input for < ', VNAME,
87:      &          '> must be placed after definition of ',
88:      &          ' energy bin number for all particle species.'
89: c##<2007/03/14:PN3:
90: c##      call PRSTOP( 1, 'PROBLEM IN ORDER OF INPUT DATA.' )
91:      call PRSTOP( 1, 'PROBLEM IN ORDER OF INPUT DATA.' )
92: c##>
93:      stop 888
94:      end if
95: c##<2007/03/14:PN3:
96: C
97: C ..... number of photonuclear nuclide is not determined yet !!! ...
98: C
99:      if ( NUCPN.le.0.and.INDEX(VNAME,'WGTPNR').ne.0 ) then
100:          write(IOR,903) VNAME
101:          call PRSTOP( 1, 'PROBLEM IN ORDER OF INPUT DATA.' )
102:          stop 888
103:      end if
104:      903 format(/' XXX(regdat) Number of "photonuclear nuclide" is not',
105:      &      ' determined.'
106:      &/13x,'Your input for < ',a,'> must be placed after $XSEC block.')
107: c##>
108: C
109: C
110: C =====
111: C      TYPE 1:  VNAME( !REGION-NAME( DATA ... ) ..... )
112: C =====
113: C
114: C
115:      if ( VNAME(1:1).ne.'!' ) then
116: C
117: C ..... EXTRACT 'VARIABLE NAME' FROM VNAME .....
118: C
119:          VNM      = ' '
120:          VSUBF     = ' '
121:          IPER      = INDEX(VNAME,'.')
122:          IKPID     = 0
123: C
124:          JSKIP     = 0
125:          JTR       = 0
126: C
127: C ... negative value check if non-zero
128:          JCKVAL    = 1
129: C
130: C ..... VNAME WITH SUBFIELD .....

```

src/shared/regdat.f

```

131: C
132:       if ( IPER.ne.0 ) then
133:         VNM(1:IPER-1) = VNAME(1:IPER-1)
134:         VSUBF = VNAME(IPER+1:ICLEN2(VNAME))
135: C
136: CCC       IPR2 = INDEX('NP',VSUBF(1:1))
137:       call KPSYMB( VSUBF(1:ICLEN2(VSUBF)),'>',IKPID, 0 )
138:       if ( IKPID.eq.0 ) then
139: c##<2007/03/14:PN3:
140: c##       write(IOR,'(1x,a,a,a,a,a)') 'XXX Data name suffix( ',
141:       write(IOR,'(1x,a,a,a,a,a)')
142: &       'XXX(regdat) Data name suffix( ',
143: c##>
144: &       VSUBF(1:ICLEN2(VSUBF)),
145: &       ' ) of ', VNAME(1:ICLEN2(VNAME)), ' is incorrect.'
146: &       write(IOR,*) ' No matching particle species.'
147:       call CNTERR( 'FATAL' )
148:       JSKIP = 1
149: C
150: C       ... supported particle species but not used in this problem
151: C
152:       else if ( JKPARIKPID).eq.0 ) then
153: c##<2007/03/14:PN3:
154: c##       write(IOR,*) '!!! ' VNAME(:iclen2(VNAME)),
155:       write(IOR,'(1x,A,A,A,A,A,A)')
156: &       '!!! (regdat) ', VNAME(:iclen2(VNAME)),
157: c##>
158: &       ' has no meanings because particle species <',
159: &       VSUBF(1:ICLEN2(VSUBF)),'> is not treated',
160: &       ' in current problem.'
161:       call CNTERR( 'WARNING ' )
162:       JSKIP = 1
163:       end if
164: C
165: C       ..... VNAME WITHOUT SUBFIELD .....
166: C
167:       else
168:         VNM = VNAME
169:       end if
170: C
171: C       if ( VSUBF.eq.'P'.and.JPHOT.eq.0 ) then
172: C         write(IOR,*) '!!! (REGDAT) DATA <', VNAME,
173: C &         '> HAS NO MEANING',
174: C &         ' FOR NO-PHOTON PROBLEM. THIS DATA IS IGNORED !!!'
175: C       call CNTERR( 'WARNING ' )
176: C       VSUBF = ' '
177: C       JSKIP = 1
178: C       end if
179: C
180: C       if ( VSUBF.eq.'N'.and.JNEUT.eq.0 ) then
181: C         write(IOR,*) '!!! (REGDAT) DATA <', VNAME,
182: C &         '> HAS NO MEANING',
183: C &         ' FOR NO-NEUTRON PROBLEM. THIS DATA IS IGNORED !!!'
184: C       call CNTERR( 'WARNING ' )
185: C       VSUBF = ' '
186: C       JSKIP = 1
187: C       end if
188: C
189:       JTR = 0
190:       LPPP = 0
191:       NR0 = NREG
192:       if ( JSKIP.eq.0 ) then
193: C
194: C       ... default array length is N1*NREG and for region (JTR=0)
195: C       if data is tally -region dependent, NR0=NTREG & JTR=1

```

```

196: C
197: C
198:       if ( VNM.eq.'XIMP' ) then
199:         if ( LXIMP.eq.0 ) then
200:           call KEPV( A(1), 'XIMP', LXIMP, NGROUP*NREG, 'R4',
201:             LAST, 1.0, JDEBG )
202:         &
203:         end if
204:         LPPP = LXIMP
205:         N1 = NGROUP
206:       else if ( VNM.eq.'WKIL' ) then
207:         if ( LWKIL.eq.0 ) then
208:           call KEPV( A(1), 'WKIL', LWKIL, NGROUP*NREG, 'R4',
209:             LAST, 1.0E-5, JDEBG )
210:         &
211:         end if
212:         LPPP = LWKIL
213:         N1 = NGROUP
214:       else if ( VNM.eq.'WSRV' ) then
215:         if ( LWSRV.eq.0 ) then
216:           call KEPV( A(1), 'WSRV', LWSRV, NGROUP*NREG, 'R4',
217:             LAST, 1.0E-4, JDEBG )
218:         &
219:         end if
220:         LPPP = LWSRV
221:         N1 = NGROUP
222:       else if ( VNM.eq.'WGTF' ) then
223:         if ( LWGTF.eq.0 ) then
224:           c##<2007/03/14:PN3:
225:           call KEPV( A(1), 'WGTF', LWGTF, NREG, 'R4', LAST, 1.0,
226:             JDEBG )
227:           c## &
228:           call KEPV( A(1), 'WGTF', LWGTF, NREG*2, 'R4', LAST,
229:             1.0, JDEBG )
230:         &
231:         end if
232:         LPPP = LWGTF
233:         N1 = 1
234:       else if ( VNM.eq.'WGTP' ) then
235:         if ( LWGTP.eq.0 ) then
236:           call KEPV( A(1), 'WGTP', LWGTP, NREG, 'R4', LAST, 1.0,
237:             JDEBG )
238:         &
239:         end if
240:         LPPP = LWGTP
241:         N1 = 1
242:       c##<2007/03/14:PN3:
243:       else if ( VNM.eq.'WGTPNR' ) then
244:         if ( LWGTPNR.eq.0 ) then
245:           call KEPV( A(1), 'WGTPNR', LWGTPNR, NMTN*NUCPN*NREG,
246:             'R4', LAST, 0.001, JDEBG )
247:         &
248:         end if
249:         LPPP = LWGTPNR
250:         N1 = 1
251:       else if ( VNM.eq.'RVOL' ) then
252:         if ( LRVOL.eq.0 ) then
253:           call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 1.0,
254:             JDEBG )
255:         &
256:         end if
257:         LPPP = LRVOL
258:         N1 = 1
259:       else if ( VNM.eq.'TRVOL' ) then
260:         if ( LTRVOL.eq.0 ) then
261:           call KEPV( A(1), 'TRVOL', LTRVOL, NTREG, 'R4', LAST,
262:             1.0, JDEBG )
263:         &
264:         end if
265:         LPPP = LTRVOL
266:         N1 = 1
267:         NR0 = NTREG

```

src/shared/regdat.f

```

261:          JTR      = 1
262:          else if ( VNM.eq.'PSALP' ) then
263:            if ( LPSALP.eq.0 ) then
264:              call KEPV( A(1), 'PSALP', LPSALP, NREG, 'R4', LAST,
265:                &          0.0, JDEBG )
266:            end if
267:            LPPP      = LPSALP
268:            N1        = 1
269: C          ... no need for negative value check
270:            JCKVAL    = 0
271:          else
272: C##<2007/03/14:PN3:
273: C##      write(IOR,*) 'XXX(REGION DEPENDENT DATA) DATA <',
274:      write(IOR,*) 'XXX(regdat) DATA <',
275: C##>
276:      &          VNM(:ICLEN(VNM)),
277:      &          '> IS NOT REGION DEPENDENT DATA.'
278:      call PRSTOP( 1, 'PROBLEM IN REGION DEPENDENT DATA INPUT.' )
279:      &          )
280:      stop 666
281:    end if
282:  end if
283: C
284: C
285: C ..... OLD TYPE INPUT ( SEQUENCE OF NUMERICAL DATA ) ...
286: C      VNAME( <...> ) OR VNAME( VALU( ... ) ) ...
287: C
288: C READ DATA OF ALL REGION IN ORDER OF REGION #.
289: C
290: C 100 call FPROBE( IPP, ' (', LINE )
291: C
292: Cmay95 if ( IPP.ne.0.and.INDEX('R<+-.0123456789',LINE(1:1)).ne.0 )
293: Ccc & then
294: C      if ( IPP.ne.0
295: C      & .and.
296: C      & (INDEX('<+-.0123456789',LINE(1:1)).ne.0.or.LINE(1:2).eq.'R('
297: C      & ) ) then
298: C
299: C      NB      = N1*NR0
300: C      if ( JSKIP.ne.0 ) then
301: C        call DMREAD( ' ', IDM1, IDM2, IIER )
302: C        if ( IIER.ne.0 ) go to 330
303: C
304: CCCCCC else if ( VSUBF.eq.' ' .or. N1.eq.1 ) then
305: C      else if ( IKPID.eq.0.or.N1.eq.1 ) then
306: C        call R4READ( VNAME, A(LPPP), NA, NB, IERR )
307: C        if ( IERR.ne.0 ) go to 330
308: C
309: C      else if ( IKPID.ne.0 ) then
310: C
311: C        .... input data temporary from A(LTMPRY)
312: C
313: C        call GTLAST( LASTTM )
314: C        call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
315: C      &          LAST, NGP(IKPID)*NR0, ' ' )
316: C
317: C        .... scatter data a(ltmpry) onto a(lpenrg) ...
318: C
319: C        call ASCTR4( VNAME, A(LPPP), A(LTMPRY), NGROUP, NR0,
320: C      &          NGP(IKPID), KNGP(IKPID), NGP(IKPID) )
321: C        call STLAST( VNAME, LASTTM, LAST )
322: C
323: C      else if ( VSUBF .eq. 'N' ) then
324: C
325: C        .... input data temporary from A(LTMPRY)

```

```

326: C
327: C      call GTLAST( LASTTM )
328: C      call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
329: C      &          LAST, NGP1*NR0, ' ' )
330: C
331: C      .... scatter data a(ltmpry) onto a(lpenrg) ...
332: C
333: C      call ASCTR4( VNAME, A(LPPP), A(LTMPRY), NGROUP, NR0,
334: C      &          NGP1, 1, NGP1 )
335: C      call STLAST( VNAME, LASTTM, LAST )
336: C
337: C      else if ( VSUBF.eq.'P' ) then
338: C
339: C        .... input data temporary from A(LTMPRY)
340: C
341: C        call GTLAST( LASTTM )
342: C        call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
343: C      &          LASTTM, NGP2*NR0, ' ' )
344: C
345: C        .... scatter data a(ltmpry) onto a(lpenrg) ...
346: C
347: C        call ASCTR4( VNAME, A(LPPP), A(LTMPRY), NGROUP, NR0,
348: C      &          NGP2, NGP1+1, NGP2 )
349: C        call STLAST( VNAME, LASTTM, LAST )
350: C
351: C      end if
352: C      go to 320
353: C
354: C      else if ( IPP.eq.0 ) then
355: C        call GTLINE( IEND )
356: C        if ( IEND.ne.0 ) then
357: C          write(IOR,*) 'XXX(REGDAT) End of file is encountered',
358: C          &          ' for data <', VNAME, '>'
359: C          call PRSTOP( 1, 'UNEXPECTED END OF INPUT FILE.' )
360: C          stop 888
361: C        end if
362: C        go to 100
363: C
364: C      end if
365: C
366: C
367: C .....
368: C LOOP 110 : READ !REGION-NAME( ..... )
369: C .....
370: C
371: C
372: C      CHAR(1:2) = '()'
373: C
374: C 110 call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
375: C
376: C      .... END OF INPUT
377: C
378: C      if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.'') go to 320
379: C
380: C      if ( NLEN.eq.0 ) then
381: C        if ( IEND.eq.0 ) then
382: C          go to 110
383: C        else
384: C          write(IOR,*) 'XXX(REGDAT) End of file is encountered',
385: C          &          ' during "!REG-NAME( ..." search.'
386: C          call PRSTOP( 1, 'UNEXPECTED END OF INPUT FILE.' )
387: C          stop 888
388: C        end if
389: C      end if
390: C

```

src/shared/regdat.f

```

391: C
392:      IR      = 0
393:      NR      = 0
394:      MMTR    = 0
395: C
396: C      .... IK > 0 : IF REGION-SPECIFICATION BY RE-DEFINED REGION.
397: C
398:      IK      = INDEX(LINE(:NLEN), '@')
399: C
400:      if ( IK.gt.0 ) then
401: C
402: C      ... @NAME without wildcard character ...
403: C
404:      if ( INDEX(LINE(:NLEN), '?').eq.0
405: &      .and.INDEX(LINE(:NLEN), '*').eq.0 ) then
406: C
407:      call CBSINC( TNAMS, NNAME, LINE(IK:NLEN), IPOS )
408: C
409:      if ( IPOS.eq.0 ) then
410:      write(IOR,*) 'XXX(REGDAT) TALLY REGION <',
411: &      LINE(IK:NLEN), '> DOES NOT EXIST.'
412:      write(IOR,*) ' DATA <', VNAME, '> FOR THIS REGION'
413: &      ' ARE MEANINGLESS.'
414:      call CNTERR( 'FATAL' )
415:      call DMREAD( ' ', IDML, N1, IIER )
416:      if ( IIER.ne.0 ) go to 330
417:      go to 110
418:      end if
419: C
420:      do 120 I = 1, NTREG
421:      if ( ITRNM(I).lt.0.and.IPOS.eq.ABS(ITRNM(I)) ) then
422:      go to 130
423:      end if
424: 120 continue
425:      write(IOR,*) 'XXX(REGDAT) FATAL ERROR ITRNM ', ITRNM
426:      call PRSTOP( 1, 'PROBLEM IN REGION DEPENDENT DATA INPUT.'
427: &      )
428:      stop 666
429: C
430: 130      IRD      = I
431:      if ( JSKIP.ne.0 ) then
432:      call DMREAD( ' ', IDML, N1, IIER )
433:      if ( IIER.ne.0 ) go to 330
434:      go to 110
435:      end if
436: C
437:      if ( JTR.eq.0 ) then
438:      do 140 K = IPTRG(IRD), IPTRG(IRD+1) - 1
439:      NR      = NR + 1
440:      KR(NR) = LPTRG(K)
441: 140 continue
442:      else
443:      NR      = 1
444:      KR(NR) = IRD
445:      end if
446: C
447: C      ... @NAME with wildcard character ...
448: C
449:      else
450:      do 170 J = 1, NTREG
451:      if ( ITRNM(J).lt.0 ) then
452:      if ( IMATCH(LINE(IK:NLEN), TNAMS(ABS(ITRNM(J)))) .ne.
453: &      0 ) then
454:      if ( JTR.eq.0 ) then
455:      do 160 I = IPTRG(J), IPTRG(J+1) - 1

```

```

456:      do 150 K = 1, NR
457:      if ( KR(K).eq.LPTRG(I) ) go to 160
458: 150 continue
459:      NR      = NR + 1
460:      KR(NR) = LPTRG(I)
461: 160 continue
462:      else
463:      NR      = NR + 1
464:      KR(NR) = J
465:      end if
466:      end if
467:      end if
468: 170 continue
469:      end if
470:      end if
471: C
472: C      ... get region/tally-region no.'s .....
473: C
474: C
475: C
476: C      ... @NAME type ...
477: C
478: CC      if ( IK.ne.0 ) then
479: CC      if ( JTR.eq.0 ) then
480: CC      do 140 K = IPTRG(IRD), IPTRG(IRD+1) - 1
481: CC      NR      = NR + 1
482: CC      KR(NR) = LPTRG(K)
483: CC140 continue
484: C      else
485: C      NR      = 1
486: C      KR(NR) = IRD
487: CCCC      end if
488: CCCC      else
489: C
490: C      ... not @NAME type ...
491: C
492:      if ( IK.eq.0 ) then
493: C
494:      IBEGIN = 0
495:      JEND   = 0
496: 180 continue
497: C
498:      call RGFIND( IOR, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
499: &      JTLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM,
500: &      ISUSP, KSUZN, IRGSP, NEST, NSPACE, NSUZON, NZONE,
501: &      NREG, TNAMS, NNAME )
502: C
503:      if ( IERR.ne.0 ) then
504:      write(IOR,*) 'XXX(REGDAT) REGION <', LINE(:NLEN),
505: &      '> is not defined or invalid name.'
506:      write(IOR,*) ' Data <', VNAME, '> for this REGION',
507: &      ' has no meaning. '
508:      call CNTERR( 'FATAL' )
509:      call DMREAD( ' ', IDML, N1, IIER )
510:      if ( IIER.ne.0 ) go to 330
511:      go to 110
512:      end if
513: C
514: C
515: C      if ( JSKIP.ne.0 ) then
516: C      call DMREAD( ' ', IDML, N1, IIER )
517: C      if ( IIER.ne.0 ) go to 240
518: C      go to 110
519: C      end if
520: C
521:      if ( IREG.ne.0 ) then

```

src/shared/regdat.f

```

521: C
522: C      ... "region" dependent data
523: C
524: C      if ( JTR.eq.0 ) then
525: C          do 190 K = 1, NR
526: C              if ( KR(K).eq.IREG ) go to 180
527: C          190      continue
528: C              NR      = NR + 1
529: C              KR(NR)  = IREG
530: C
531: C      ... "tally region" dependent data
532: C
533: C      else
534: C          do 230 J = 1, NTREG
535: C              do 220 I = IPTRG(J), IPTRG(J+1) - 1
536: C                  if ( IREG.eq.LPTRG(I) ) then
537: C                      ... MMTR : count of matching regions
538: C
539: C                      do 200 K = 1, NR
540: C                          if ( J.eq.KR(K) ) then
541: C                              MMTR = MMTR + 1
542: C                              go to 210
543: C                          end if
544: C                      200      continue
545: C                          NR      = NR + 1
546: C                          KR(NR)  = J
547: C                      210      continue
548: C                  end if
549: C              220      continue
550: C              230      continue
551: C          end if
552: C      end if
553: C      if ( JEND.eq.0 ) go to 180
554: C      end if
555: C
556: C      ... Region/tally-region list KR(1:NR) is constructed.
557: C      read and store data ...
558: C
559: C      if ( JTR.ne.0.and.MMTR.gt.1 ) then
560: C          CCCCC write(IOR,*) '!!! IN INPUT OF TALLY-REGION DEPENDENT DATA',
561: C              & ' <', VNM, '>', T-REG NAME <', LINE(1:NLEN), 'S',
562: C              & ' INCLUDES MORE THAN ONE TALLY-REGION(' , NR, ')'.',
563: C              & ' Check results of TRVOL input carefully!'
564: C          call CNTERR( 'WARNING' )
565: C      end if
566: C
567: C      if ( NR.gt.0 ) then
568: C          IR0 = KR(1)
569: C          LP0 = LPPP + N1*(IR0-1)
570: C          NN  = N1
571: C          if ( N1.eq.NGROUP ) then
572: C              if ( VSUBF.eq.'N' ) then
573: C                  NN = NGP1
574: C              else if ( VSUBF.eq.'P' ) then
575: C                  NN = NGP2
576: C                  LP0 = LP0 + NGP1
577: C              end if
578: C          if ( IKPID.ne.0 ) then
579: C              NN = NGP(IKPID)
580: C              LP0 = LP0 + KNGP(IKPID)-1
581: C          end if
582: C      end if
583: C
584: C
585: C

```

```

586: C      MESSAGE = VNAME//'( '//LINE(1:NLEN) //' )'
587: C      call R4READ( MESSAGE, A(LP0), NA, NN, IERR )
588: C
589: C      ... check for negative value
590: C      (do not set fatal error flag here, expecting actual
591: C      value range check is performed afterwards)
592: C
593: C      if ( JCKVAL.ne.0 ) then
594: C          call CKVAL4( MESSAGE, A(LP0), NN, ICHK, 0 )
595: C      end if
596: C
597: C      do 240 K = 2, NR
598: C          LP = LPPP + N1*(KR(K)-1)
599: C          CCCC if ( N1.eq.NGROUP.and.VSUBF.eq.'P' ) LP = LP + NGP1
600: C          if ( N1.eq.NGROUP.and.IKPID.ne.0 ) then
601: C              LP = LP + KNGP(IKPID)-1
602: C          end if
603: C
604: C          if ( LP.ne.LP0 ) call ATRANS( A(LP0), A(LP), NN )
605: C      240      continue
606: C      end if
607: C
608: C      go to 110
609: C      end if
610: C
611: C
612: C      =====
613: C      TYPE 2: !REGION-NAME( VNAME( DATA ... ) .... )
614: C      =====
615: C
616: C      if ( VNAME(1:1).eq.'!' ) then
617: C
618: C      .... MAKE REGION # LIST IN 'KR' ...
619: C
620: C      NR      = 0
621: C      JSKIP   = 0
622: C      ... negative value check if non-zero
623: C      JCKVAL  = 1
624: C
625: C      IK      = INDEX(VNAME,'@')
626: C
627: C      ... region name is a tally region name as "@name" ...
628: C
629: C      if ( IK.ne.0 ) then
630: C
631: C          call CBSINC( TNAMS, NNAMES, VNAME(IK:), IPOS )
632: C
633: C          if ( IPOS.eq.0 ) then
634: C              write(IOR,'(//1x,4a)') 'XXX( REGION DEPENDENT DATA ) ',
635: C              & ' TALLY REGION <', VNAME(2:), '> DOES NOT EXIST.'
636: C          &
637: C          JSKIP = 1
638: C          call CNTERR( 'FATAL' )
639: C      else
640: C          do 250 I = 1, NTREG
641: C              if ( ITRNM(I).lt.0.and.IPOS.eq.ABS(ITRNM(I)) ) go to
642: C              & 260
643: C          250      continue
644: C
645: C          write(IOR,*) 'XXX(REGDAT) FATAL ERROR ITRNM ', ITRNM
646: C          call PRSTOP( 1, 'PROBLEM IN REGION DEPENDENT DATA INPUT.'
647: C          & )
648: C          & 'PROBLEM IN REGION DEPENDENT INPUT DATA.' )
649: C      Cccc &
650: C      stop 666

```

src/shared/regdat.f

```

651:
652: 260      IRD      = I
653:          do 270 I = IPTRG(IRD), IPTRG(IRD+1) - 1
654:              NR      = NR + 1
655:              KR(NR)   = LPTRG(I)
656: 270      continue
657:          end if
658:      else
659: C
660:          IBEGIN = 0
661: C
662: 280      call RGFIND( IOR, VNAME, IBEGIN, JEND, IREG, IERR, JTLLT,
663: &                  KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
664: &                  KSUZN, IROSP, NEST, NSPACE, NSUZON, NZONE, NREG,
665: &                  TNAMS, NNAMES )
666: C
667:          if ( IERR.ne.0 ) then
668:              write(IOR, '(//lx,3a)')
669:              &          'XXX(REGION DEPENDENT DATA) REGION <', VNAME,
670:              &          '> DOES NOT EXIST.'
671:              write(IOR,*) ' DATA FOR THIS REGION(S) ARE IGNORED.'
672:              JSKIP = 1
673:              call CNTERR( 'FATAL' )
674:          end if
675: C
676:          if ( IREG.ne.0 ) then
677: C
678:              do 290 K = 1, NR
679:                  if ( KR(K).eq.IREG ) go to 280
680: 290          continue
681: C
682:              NR      = NR + 1
683:              KR(NR)   = IREG
684:          end if
685:          if ( JEND.eq.0 ) go to 280
686:          end if
687:          if ( NR.eq.0 ) JSKIP = 1
688: C
689: C
690: C .....
691: C LOOP 210 : READ  VNAME( ..... )
692: C .....
693: C
694: C
695:          CHAR(1:2) = '()'
696: C
697: 300      call CHREAD( ' ', LINE, CHAR(1:2), NLEN, ITERM, IEND )
698: C
699:          ..... END OF INPUT
700: C
701:          if ( ITERM.ne.0.and.CHAR(ITERM:ITERM).eq.' ' ) go to 320
702: C
703:          if ( NLEN.eq.0 ) then
704:              if ( IEND.eq.0 ) then
705:                  go to 300
706:              else
707:                  write(IOR,*) 'XXX(REGDAT) END OF FILE IS ENCOUNTERED',
708:                  &          ' IN "DATA-NAME( ..." SEARCH.'
709:                  call PRSTOP( 1, 'UNEXPECTED END OF INPUT FILE.' )
710:                  stop 888
711:              end if
712:          end if
713: C
714: C ..... SKIP INPUT IF REGION NAME IS INVALID. ....
715: C

```

```

716:          if ( JSKIP.ne.0 ) then
717:              call DMREAD( ' ', IDM1, IDM2, IIER )
718:              if ( IIER.ne.0 ) go to 330
719:              go to 300
720:          end if
721: C
722: C
723: C ..... EXTRACT 'VARIABLE NAME' FROM VNAME .....
724: C
725:          VNM      = ' '
726:          VSUBF     = ' '
727:          IPER      = INDEX(LINE(:NLEN),'.')
728:          IKPID     = 0
729: C
730: C ..... VNAME WITH SUBFIELD .....
731: C
732:          if ( IPER.ne.0 ) then
733:              VNM(1:IPER-1) = LINE(1:IPER-1)
734:              VSUBF = LINE(IPER+1:ICLEN2(LINE))
735: C
736:              IPR2 = INDEX('NP',VSUBF(1:1))
737:              if ( IPR2.eq.0 ) then
738:                  write(IOR,*) 'XXX SUFFIX( ' , VSUBF, ' ) OF ' , VNAME,
739:                  &          ' IS', ' INCORRECT XXX '
740:                  call CNTERR( 'FATAL' )
741:                  JSKIP = 1
742:              end if
743: C
744:          if ( IPER.ne.0 ) then
745:              VNM(1:IPER-1) = LINE(1:IPER-1)
746:              VSUBF = LINE(IPER+1:ICLEN2(LINE))
747: C
748:              call KPSYMB( VSUBF(1:ICLEN2(VSUBF)), '>', IKPID, 0 )
749: C
750:              if ( IKPID.eq.0 ) then
751:                  write(IOR, '(lx,a,a,a,a,a)') 'XXX Data name suffix( ' ,
752:                  &          VSUBF(1:ICLEN2(VSUBF)),
753:                  &          ' ) of ' , LINE(1:ICLEN2(LINE)), ' is incorrect.'
754:                  write(IOR,*) ' No matching particle species.'
755:                  call CNTERR( 'FATAL' )
756:                  JSKIP = 1
757: C
758:          ... supported particle species but not used in this problem
759: C
760:          else if ( JKPAR(IKPID).eq.0 ) then
761:              write(IOR,*) '!!! ' , LINE(:iclen2(LINE)),
762:              &          ' has no meanings because particle species <',
763:              &          VSUBF(1:ICLEN2(VSUBF)), '> is not treated',
764:              &          ' in current problem.'
765:              call CNTERR( 'WARNING ' )
766:              JSKIP = 1
767:          end if
768: C
769: C ..... VNAME WITHOUT SUBFIELD .....
770: C
771: C
772:          else
773:              VNM = LINE(:NLEN)
774:          end if
775: C
776:          if ( VSUBF.eq.'P'.and.JPHOT.eq.0 ) then
777:              write(IOR,*) '!!!(REGDAT) DATA <', LINE(:NLEN), '> HAS ' ,
778:              &          ' NO MEANING FOR NO-PHOTON PROBLEM. IGNORED !!!'
779:              call CNTERR( 'WARNING ' )
780:              VSUBF = ' '

```

src/shared/regdat.f

```

781: C      JSKIP = 1
782: C      end if
783: C      if ( VSUBF.eq.'N'.and.JNEUT.eq.0 ) then
784: C          write(IOR,*) '!!!(REGDAT) DATA <', LINE(:NLEN), '> HAS ',
785: C      &      ' NO MEANING FOR NO-NEUTRON PROBLEM. IGNORED !!!'
786: C      call CNTERR( 'WARNING ' )
787: C      VSUBF = ' '
788: C      JSKIP = 1
789: C      end if
790: C
791: C      if ( JSKIP.ne.0 ) then
792: C          call DMREAD( ' ', IDM1, IDM2, IIER )
793: C          if ( IIER.ne.0 ) go to 330
794: C          go to 300
795: C      end if
796: C
797: C      NN = 0
798: C      if ( VNM.eq.'XIMP' ) then
799: C          if ( LXIMP.eq.0 ) then
800: C              call KEPV( A(1), 'XIMP', LXIMP, NGROUP*NREG, 'R4', LAST,
801: C      &      1.0, JDEBG )
802: C          end if
803: C          LPPP = LXIMP
804: C          N1 = NGROUP
805: C          NN = N1
806: C      else if ( VNM.eq.'WKIL' ) then
807: C          if ( LWKIL.eq.0 ) then
808: C              call KEPV( A(1), 'WKIL', LWKIL, NGROUP*NREG, 'R4', LAST,
809: C      &      1.0E-5, JDEBG )
810: C          end if
811: C          LPPP = LWKIL
812: C          N1 = NGROUP
813: C          NN = N1
814: C      else if ( VNM.eq.'WSRV' ) then
815: C          if ( LWSRV.eq.0 ) then
816: C              call KEPV( A(1), 'WSRV', LWSRV, NGROUP*NREG, 'R4', LAST,
817: C      &      1.0E-4, JDEBG )
818: C          end if
819: C          LPPP = LWSRV
820: C          N1 = NGROUP
821: C          NN = N1
822: C      else if ( VNM.eq.'WGTF' ) then
823: C          if ( LWGTF.eq.0 ) then
824: C      c##<2007/03/14:PN3:
825: C      c##      call KEPV( A(1), 'WGTF', LWGTF, NREG, 'R4', LAST, 1.0,
826: C      call KEPV( A(1), 'WGTF', LWGTF, NREG*2, 'R4', LAST, 1.0,
827: C      c##>
828: C      &      JDEBG )
829: C          end if
830: C          LPPP = LWGTF
831: C          N1 = 1
832: C      else if ( VNM.eq.'WGTP' ) then
833: C          if ( LWGTP.eq.0 ) then
834: C              call KEPV( A(1), 'WGTP', LWGTP, NREG, 'R4', LAST, 1.0,
835: C      &      JDEBG )
836: C          end if
837: C          LPPP = LWGTP
838: C          N1 = 1
839: C      c##<2007/03/14:PN3:
840: C      else if ( VNM.eq.'WGTPNR' ) then
841: C          if ( LWGTPN.eq.0 ) then
842: C              call KEPV( A(1), 'WGTPNR', LWGTPN, NMTPN*NUCPN*NREG, 'R4',
843: C      &      LAST, 0.001, JDEBG )
844: C          end if
845: C          LPPP = LWGTPN

```

```

846: C      N1 = 1
847: C      c##>
848: C      else if ( VNM.eq.'RVOL' ) then
849: C          if ( LRVOL.eq.0 ) then
850: C              call KEPV( A(1), 'RVOL', LRVOL, NREG, 'R4', LAST, 1.0,
851: C      &      JDEBG )
852: C          end if
853: C          LPPP = LRVOL
854: C          N1 = 1
855: C      else if ( VNM.eq.'PSALP' ) then
856: C          if ( LPSALP.eq.0 ) then
857: C              call KEPV( A(1), 'PSALP', LPSALP, NREG, 'R4', LAST, 0.0,
858: C      &      JDEBG )
859: C          end if
860: C          LPPP = LPSALP
861: C          N1 = 1
862: C          ... PSALP does not need negative value check
863: C          JCKVAL = 0
864: C
865: C      else if ( VNM.eq.'TRVOL' ) then
866: C      c##<2007/03/14:PN3:
867: C      c##      write(IOR,*) 'XXX Input of <TRVOL> ',
868: C      write(IOR,'(1X,A,A,A,A)') 'XXX(regdat) Input of <TRVOL> ',
869: C      c##>
870: C      &      'in the form as "!T-REGION( DATA(...) ... )"',
871: C      &      ' is not allowed. Use "TRVOL( !T-REG(...))"'
872: C      call CNTERR( 'FATAL' )
873: C      call DMREAD( ' ', IDM1, IDM2, IIER )
874: C      go to 300
875: C
876: C      else
877: C      c##<2007/03/14:PN3:
878: C      c##      write(IOR,*) 'XXX Data <', LINE(:NLEN), '> ',
879: C      write(IOR,'(1X,A,A,A,A,A)')
880: C      &      'XXX(regdat) Data <', LINE(:NLEN), '> ',
881: C      c##>
882: C      &      'is not allowed for !REGION( DATA(...) ... )',
883: C      &      ' type of input.'
884: C      &      ( or missing closing parenthesis !! )'
885: C      call prstop(1,'PROBLEMS IN REGION DEPENDENT INPUT DATA.')
886: C      stop 888
887: C      call CNTERR( 'FATAL' )
888: C      call DMREAD( ' ', IDM1, IDM2, IIER )
889: C      if ( IIER.ne.0 ) go to 330
890: C      go to 300
891: C      end if
892: C
893: C      IRO = KR(1)
894: C      LP0 = LPPP + N1*(IRO-1)
895: C      NN = N1
896: C      if ( N1.eq.NGROUP ) then
897: C          if ( VSUBF.eq.'N' ) then
898: C              NN = NGP1
899: C          else if ( VSUBF.eq.'P' ) then
900: C              NN = NGP2
901: C              LP0 = LP0 + NGP1
902: C          end if
903: C          if ( IKPID.ne.0 ) then
904: C              NN = NGP(IKPID)
905: C              LP0 = LP0 + KNGP(IKPID)-1
906: C          end if
907: C      else
908: C          NN = N1
909: C      end if
910: C

```

src/shared/regdat.f

```
911:      MESSGE = LINE(:NLEN) //'('//VNAME//')'
912:      call R4READ( MESSGE, A(LP0), NA, NN, IERR )
913: C
914: C      ... check for negative value
915: C      (do not set fatal error flag here, expecting actual
916: C      value range check is performed afterwards)
917: C
918: C      if ( JCKVAL.ne.0 ) then
919: C          call CKVAL4( MESSGE, A(LP0), NN, ICHK, 0 )
920: C      end if
921: C
922: C      do 310 K = 2, NR
923: C          LP = LPPP + N1*(KR(K)-1)
924: C          if ( N1.eq.NGROUP.and.VSUBP.eq.'P' ) then
925: C              LP = LP + NGP1
926: C          end if
927: C          if ( N1.eq.NGROUP.and.IKPID.ne.0 ) then
928: C              LP = LP + KNKP(IKPID)-1
929: C          end if
930: C
931: C          if ( LP.ne.LP0 ) call ATRANS( A(LP0), A(LP), NN )
932: 310      continue
933: C
934: C          go to 300
935: C      end if
936: C
937: 320      return
938: C
939: C
940: C      ... error in dmread ...
941: C
942: 330      write(IOR,7000)
943: c###<2007/03/14:PN3:
944: c7000 format(1X,'XXX Unexpected end of data or data input error ',
945: 7000 format(' XXX(regdat) Unexpected end of data or data input error ',
946: c##>
947: &          'during skipping unnecessary or invalid data.//1X,
948: &          '(REGION dependent data input)')
949: call PRSTOP( 1, 'ERROR IN REGION DEPENDENT DATA INPUT.' )
950: stop 888
951: C
952: end
```


src/shared/reglst.f

```

1:      subroutine REGLST( NAME, IREGS, NN,      NNREGS,N2,      A,      CHA,
2:      &
3:      &
4:      C purpose: make region# list for a given region (or tally-region) name.
5:      C called in : TALINI
6:      C=====
7:      C arguments (i=input, o=output, w=work)
8:      C
9:      C i NAME : region or tally region name
10:     C      (after 16 Nov 1999) or combination of them jointed by "+".
11:     C      ex.
12:     C
13:     C REGA* : region matching this pattern (REGA REGAl REGAM ...etc.)
14:     C @TREG1 : regions in tally region @TREG1
15:     C      ( tally region name including wildcard character
16:     C      like @TREG* is not supported!!)
17:     C REG1+AREG*+@TREG1 : region (or tally region components)
18:     C      matching "REG1", "AREG*" or "@TREG1".
19:     C
20:     C o IREGS(NN) : array to store region #'s
21:     C      (a region does not appear more than once in the list)
22:     C i NN : size limit of iregs(*)
23:     C o NNREGS : number of region #'s stored in iregs.
24:     C o N2 : number of region #'s that matches the name.
25:     C      (could be greater than nnregs)
26:     C io A : task shared dynamic data area used as "REAL" type.
27:     C io CHA : task shared dynamic data area used as "CHARACTER*4" type.
28:     C o IERR : error code
29:     C      =0 : successful
30:     C      non-zero : no regions found or more than NN matching regions
31:     C      found.
32:     C-----
33:     character*(*) NAME
34:     integer IREGS(NN)
35:     C
36:     real A(*)
37:     character*4 CHA(*)
38:     C
39:     CCCC include 'INC/_ARRAY'
40:     include 'INC/_SIZES'
41:     include 'INC/_PGEOM'
42:     include 'INC/_PTLSP'
43:     C-----
44:     C
45:     C ... actual procedure is in RGLST0
46:     C
47:     C (arguments necessary for RGFIND routine and TALLY-REGION name
48:     C information is passed.)
49:     C
50:     call RGLST0( NAME, IREGS, NN, NNREGS, N2, IERR, IGTFLG('JTLLT'),
51:     &
52:     & A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
53:     & A(LISPNM), A(LISUSP), A(LKSUZN), A(LIRGSP), NEST, NSPACE,
54:     & NSUZON, NZONE, NREG, CHA(LTNAMS), NNAMES, NTREG,
55:     & A(LIPTRG), A(LLPTRG), A(LITRNM) )
56:     C
57:     if ( N2.gt.NNREGS ) IERR = 2
58:     C
59:     return
60:     end
61: C+++++
62: C
63:      subroutine RGLST0( NAME, IREGS, NN,      NNREGS,N2,      IERR,
64:      &
65:      & JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ,
66:      &
67:      & ISPNM, ISUSP, KSUZN, IRGSP, NEST, NSPACE,

```

```

66:      &
67:      & NSUZON,NZONE, NREG, TNAMS, NNAMES,NTREG,
68:      & IPTRG, LPTRG, ITRNM )
69: C=====
70: C purpose: working body of REGLST routine.
71: C (Make region# list for a given region (or tally-region) name.
72: C called in : REGLST
73: C-----
74: C see argument and update description of REGLST routine.
75: C=====
76:     character*(*) NAME
77:     integer IREGS(NN)
78:     C
79:     integer KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ)
80:     integer ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
81:     & KSUZN(NINPZ,2), IRGSP(2,NREG), KMAT(NINPZ)
82:     include 'INC/_LNAM'
83:     character*(LNAM) TNAMS(NNAMES)
84:     C
85:     C
86:     integer IPTRG(NTREG+1), LPTRG(*)
87:     integer ITRNM(NTREG)
88:     include 'INC/_IUNIT'
89:     C-----
90:     C
91:     NNREGS = 0
92:     N2 = 0
93:     IERR = 0
94:     ISEP = INDEX(NAME, '+')
95:     C
96:     C
97:     ISUBNM = 0
98:     ISS = 1
99:     C
100:    100 if ( ISS.gt.LEN(NAME) ) go to 180
101:    NNN0 = NNREGS
102:    C
103:    C ... regionname pattern may include more than one pattern string
104:    C separated by "+".
105:    C
106:    C      pattern1+pattern2+...
107:    C
108:    IKK = INDEX(NAME(ISS:), '+')
109:    IEE = LEN(NAME)
110:    if ( IKK.eq.1 ) then
111:      write(IMG,7000) ISS, NAME, ( ' ',I=1,ISS-1), '*'
112:    7000 format(/lx,' !!!(REGLST) The following region name pattern',
113:    & ' has excessive "+" concatenator./lx,' Name: ',A/lx,
114:    & ' >> : ',128(A:))
115:    call CNTERR( 'WARNING' )
116:    ISS = ISS + 1
117:    go to 100
118:    else if ( IKK.gt.1 ) then
119:      IEE = ISS + IKK - 2
120:    end if
121:    C
122:    ISUBNM = ISUBNM + 1
123:    C
124:    C == tally region ==
125:    C
126:    if ( NAME(ISS:ISS).eq.'@' ) then
127:    Check %%%
128:    C      write(IPR,*) '%%(REGLST)1 ',ISUBNM,' <',NAME(ISS:IEE),'>'
129:    C %%%%%%%%%
130:    ITRG = 0

```

src/shared/reglst.f

```

131:      do 140 I = 1, NTREG
132: C
133: C      ... names beginning with @ has negative index to TNAMS
134: C
135:      if ( ITRNM(I).lt.0 ) then
136:      if ( TNAMS(ABS(ITRNM(I))).eq.NAME(ISS:IEE) ) then
137:      ITRG      = ITRG + IPTRG(I+1) - IPTRG(I)
138:      do 130 J = IPTRG(I), IPTRG(I+1) - 1
139: C
140: C      ... do not register a region more than once
141: C
142:      NN0      = NNREGS
143:      do 110 K = 1, NN0
144:      if ( IREGS(K).eq.LPTRG(J) ) go to 120
145: 110      continue
146: C
147:      N2      = N2 + 1
148:      if ( N2.gt.NN ) then
149:      IERR      = 1
150:      NNREGS    = NN
151:      else
152:      NNREGS    = N2
153:      IREGS(NNREGS) = LPTRG(J)
154:      end if
155: C
156: 120      continue
157: 130      continue
158:      end if
159:      end if
160: 140      continue
161: C
162: C      ... print error messege for non exstent tally region
163: C      if the name pattern includes more than one pattern string
164: C
165:      if ( ITRG.eq.0.and.ISEP.ne.0 ) then
166:      write(IMG,7020) NAME(ISS:IEE), NAME
167: 7020      format(/lx,' XXX(REGLIST) tally region <' ,A, > in the ',
168:      &      'following region name pattern does not defined.'/
169:      &      lx,' Name: ',A)
170:      call CNTERR( 'FATAL' )
171:      IERR      = 1
172:      end if
173: C
174: C      == region name (may include wild card characters)
175: C
176:      else
177: Check %%%
178: C      write(IPR,*) '%%(REGLST)2 ',ISUBNM,' <',NAME(ISS:IEE),'>'
179: C %%%%%%%%%
180: C
181:      IBEGIN = 0
182: C      ... I/O unit passed is changed to MSG from IPR (Jan 2000)
183: 150      call RGFIND( MSG, NAME(ISS:IEE), IBEGIN, JEND, IREG, IERR,
184:      &      JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
185:      &      KSUZN, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
186:      &      NNAMES )
187: C
188:      if ( IREG.ne.0 ) then
189:      do 160 I = 1, NNREGS
190:      if ( IREG.eq.IREGS(I) ) go to 170
191: 160      continue
192:      N2      = N2 + 1
193:      if ( N2.le.NN ) then
194:      IREGS(N2) = IREG
195:      end if
196: 170      continue
197:      end if
198:      if ( N2.gt.NN ) then
199:      IERR      = 1
200:      NNREGS    = NN
201:      else
202:      NNREGS    = N2
203:      end if
204:      if ( JEND.eq.0 ) go to 150
205: C
206:      end if
207: C
208: Check %%%
209: C      write(IPR,*) '%%(REGLST) IREGS ',(IREGS(L),L=NNN0+1,NNREGS)
210: C %%%%%%%%%
211: C
212: C      ... proceed to next sub pattern
213: C
214:      ISS      = IEE + 2
215:      go to 100
216: C
217: 180      continue
218: C
219: Check %%%
220: C      write(IPR,*) '%%(REGLST) IREGS final ',
221: C      &      (IREGS(L),L=1,NNREGS)
222: C %%%%%%%%%
223: C
224:      return
225:      end

```

src/shared/regnam.f

```

1:      subroutine REGNAM( IREG, DIRC, NAME, LNM, JTLLT, KMAT, KREG,
2:      &                  KREGI, KCELI, NINPZ, ISUSP, ISPNM, KSUZN,
3:      &                  IRGSP, NZONE, NSPACE, NSUZON, NEST, NREG,
4:      &                  TNAMS, NNAMES )
5: C=====
6: C PURPOSE : REGION # <=> REGION NAME TRANSLATION.
7: C CALLED IN: RGFIND, TLOREG
8: C-----
9: C arguments ( i=input o=output w=work )
10: C i/o IREG : region number input/output
11: C i DIRC : '>' -- region # to region name
12: C i '<' -- region name to region #
13: C o/i NAME : region number output/input
14: C o LNM : length of region name output (undefined when DIRC='<')
15: C
16: C i all other arguments
17: C
18: C=====
19:      integer IREG
20:      character*1 DIRC
21:      character*(*) NAME
22: C
23:      integer KMAT(NINPZ,2), KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ)
24:      integer ISUSP(NSUZON,NSPACE), ISPNM(2,0:NEST,NSPACE)
25:      integer KSUZN(NINPZ,2), IRGSP(2,NREG)
26:      include 'INC/_LNAM'
27:      character*(LNAM) TNAMS(NNAMES)
28: C
29:      include 'INC/_IUNIT'
30: C
31: C .... LOCAL VARIABLES ....
32: C
33:      parameter( MAXLVL = 12 )
34:      integer IISPNM(2,0:MAXLVL)
35: C
36: C ==== DIRC = '>': REGION # --> NAME
37: C
38:      if ( DIRC.eq.'>' ) then
39:      if ( JTLLT.eq.0 ) then
40:      do 100 I = 1, NINPZ
41:      if ( KREG(I).eq.IREG ) then
42:      NAME = TNAMS(KREGI(I))
43:      LNM = LNAM
44:      go to 230
45:      end if
46: 100 continue
47:      go to 220
48:      else
49:      do 110 I = 1, NINPZ
50:      if ( KMAT(I,1).ge.0.and.KCELI(I).eq.0 ) then
51:      if ( KREG(I).eq.IREG ) then
52:      NAME = TNAMS(KREGI(I))
53:      LNM = LNAM
54:      go to 230
55:      end if
56:      end if
57: 110 continue
58: C
59: C
60: C IS = IRGSP(1,IREG)
61: C
62: C K = ISPNM(1,0,IS)
63: C LNM = 0
64: C do 120 J = 1, K
65: C LF = INDEX(TNAMS(ISPNM(1,J,IS)),' ') - 1

```

```

66:      if ( LF.lt.0 ) LF = LNAM
67:      LS = INDEX(TNAMS(ISPNM(2,J,IS)),' ') - 1
68:      if ( LS.lt.0 ) LS = LNAM
69:      LNM1 = LNM + 1 + LF + 1 + LS
70:      write(NAME(LNM+1:LNM1),'('!'',A,'''',A)')
71:      & TNAMS(ISPNM(1,J,IS)) (:LF),
72:      & TNAMS(ISPNM(2,J,IS)) (:LS)
73:      LNM = LNM1
74: 120 continue
75: C
76:      LR = INDEX(TNAMS(IRGSP(2,IREG)),' ') - 1
77:      if ( LR.lt.0 ) LR = LNAM
78: C
79:      write(NAME(LNM+1:LNM+1+LR),'('!'',A)')
80:      & TNAMS(IRGSP(2,IREG)) (1:LR)
81: C
82:      LNM = LNM + 1 + LR
83:      go to 230
84:      end if
85: C
86: C ==== REGION NAME --> REGION #
87: C
88:      else if ( DIRC.eq.'<' ) then
89:      IERR = 0
90: C
91:      call CCOMP( NAME, LEN(NAME), NAME, IFL )
92:      LP = 1
93:      if ( NAME(1:1).eq.'!' ) LP = LP + 1
94: C
95: C .... For "TALLY-LATTICE" mode
96: C
97:      if ( JTLLT.ne.0 ) then
98:      LVL = 0
99: C
100: 130 K = INDEX(NAME(LP:),'')
101:      if ( K.ne.0 ) then
102:      LVL = LVL + 1
103: C
104: C ..... Frame NAME:
105: C
106:      call CBSINC( TNAMS, NNAMES, NAME(LP:LP+K-2), IPOS )
107: C
108:      if ( IPOS.eq.0 ) then
109:      write(IMG,*) 'XXX(REGNAM) Invalid "FRAME" name ',
110:      & NAME(LP:LP+K-2)
111:      IERR = IERR + 1
112:      call CNTERR( 'FATAL' )
113:      end if
114:      IISPNM(1,LVL) = IPOS
115:      LP = LP + K
116: C
117: C ..... :SUBFRAME-NAME!
118: C
119:      KK = INDEX(NAME(LP:),'!')
120:      call CBSINC( TNAMS, NNAMES, NAME(LP:LP+KK-2), IPOS )
121:      if ( IPOS.eq.0 ) then
122:      write(IMG,*) 'XXX Invalid "SUBFRAME" name ',
123:      & NAME(LP:LP+KK-2)
124:      IERR = IERR + 1
125:      call CNTERR( 'FATAL' )
126:      end if
127:      IISPNM(2,LVL) = IPOS
128:      LP = LP + KK
129:      go to 130
130:      end if

```

src/shared/regnam.f

```

131: C
132:       if ( LVL.gt.0 ) then
133:         do 150 IS = 1, NSPACE
134:           if ( ISPNM(1,0,IS).eq.LVL ) then
135:             do 140 I = 1, LVL
136:               if ( IISPNM(1,I).ne.ISPNM(1,I,IS)
137:                 &
138:                 .or. IISPNM(2,I).ne.ISPNM(2,I,IS) ) go to 150
139:             continue
140:             ISPACE = IS
141:             go to 160
142:           end if
143:         continue
144:       write(IMG, '(//lx,a,a,a//)')
145:       &
146:       &
147:       &
148:       &
149: C
150:       150 KJ = INDEX(NAME(LP:), ' ')
151:       if ( KJ.eq.0 ) KJ = LEN(NAME) - LP + 1
152:       call CBSINC( TNAMS, NNAMES, NAME(LP:LP+KJ-1), IPOS )
153:       if ( IPOS.eq.0 ) then
154:         write(IMG, '(//lx,a,a//)') 'XXX(REGNAM) Invalid name ',
155:         &
156:         NAME(LP:LP+KJ-1)
157:         IERR = IERR + 1
158:         call CNTERR( 'FATAL' )
159:       end if
160:       if ( IERR.gt.0 ) then
161:         IREG = 0
162:         go to 230
163:       end if
164: C
165:       LP2 = LP + KJ - 1
166:       do 170 I = 1, NSUZON
167:         if ( ISUSP(I,ISPACE).ne.0 ) then
168:           if ( TNAMS(KREGI(KSUZON(I,1))).eq.NAME(LP:LP2) )
169:             then
170:               IREG = ISUSP(I,ISPACE)
171:               go to 230
172:             end if
173:           end if
174:         continue
175:       write(IMG,*) 'XXX(REGNAM) Invalid region name : ', NAME
176:       call CNTERR( 'FATAL' )
177:       IREG = 0
178:       go to 230
179:     else
180: C
181:       KJ = INDEX(NAME(LP:), ' ') - 1
182:       if ( KJ.lt.0 ) KJ = LEN(NAME)
183:       call CBSINC( TNAMS, NNAMES, NAME(LP:KJ), IPOS )
184:       if ( IPOS.eq.0 ) then
185:         write(IMG,*) 'XXX(REGNAM) Invalid region name ',
186:         &
187:         NAME(:KJ)
188:         call CNTERR( 'FATAL' )
189:         IREG = 0
190:         go to 230
191:       end if
192: C
193:       do 180 IZ = 1, NINPZ
194:         if ( KMAT(IZ,1).ge.0.and.IPOS.eq.KREGI(IZ) ) go to 190
195:         continue
196:       write(IMG,*) 'XXX(REGNAM) INVALID REGION NAME ',

```

```

196:       &
197:       NAME(:KJ)
198:       call CNTERR( 'FATAL' )
199:       IREG = 0
200:       go to 230
201: C
202:       190 IREG = KREG(IZ)
203:       go to 230
204: C
205:       end if
206: C
207: C
208: C
209: C
210: C
211: C
212: C
213: C
214: C
215: C
216: C
217: C
218: C
219: C
220: C
221: C
222: C
223: C
224: C
225: C
226: C
227: C
228: C
229: C
230: C
231: C
232: C
233: C
234: C
235: C
236: C
237: C
238: C
239: C
240: C
241: C
242: C
243: C
244: C
245: C
246: C
247: C
248: C
249: C
250: C
251: C
252: C
253: C
254: C
255: C
256: C
257: C
258: C
259: C
260: C
261: C
262: C
263: C
264: C
265: C
266: C
267: C
268: C
269: C
270: C
271: C
272: C
273: C
274: C
275: C
276: C
277: C
278: C
279: C
280: C
281: C
282: C
283: C
284: C
285: C
286: C
287: C
288: C
289: C
290: C
291: C
292: C
293: C
294: C
295: C
296: C
297: C
298: C
299: C
300: C
301: C
302: C
303: C
304: C
305: C
306: C
307: C
308: C
309: C
310: C
311: C
312: C
313: C
314: C
315: C
316: C
317: C
318: C
319: C
320: C
321: C
322: C
323: C
324: C
325: C
326: C
327: C
328: C
329: C
330: C
331: C
332: C
333: C
334: C
335: C
336: C
337: C
338: C
339: C
340: C
341: C
342: C
343: C
344: C
345: C
346: C
347: C
348: C
349: C
350: C
351: C
352: C
353: C
354: C
355: C
356: C
357: C
358: C
359: C
360: C
361: C
362: C
363: C
364: C
365: C
366: C
367: C
368: C
369: C
370: C
371: C
372: C
373: C
374: C
375: C
376: C
377: C
378: C
379: C
380: C
381: C
382: C
383: C
384: C
385: C
386: C
387: C
388: C
389: C
390: C
391: C
392: C
393: C
394: C
395: C
396: C
397: C
398: C
399: C
400: C
401: C
402: C
403: C
404: C
405: C
406: C
407: C
408: C
409: C
410: C
411: C
412: C
413: C
414: C
415: C
416: C
417: C
418: C
419: C
420: C
421: C
422: C
423: C
424: C
425: C
426: C
427: C
428: C
429: C
430: C
431: C
432: C
433: C
434: C
435: C
436: C
437: C
438: C
439: C
440: C
441: C
442: C
443: C
444: C
445: C
446: C
447: C
448: C
449: C
450: C
451: C
452: C
453: C
454: C
455: C
456: C
457: C
458: C
459: C
460: C
461: C
462: C
463: C
464: C
465: C
466: C
467: C
468: C
469: C
470: C
471: C
472: C
473: C
474: C
475: C
476: C
477: C
478: C
479: C
480: C
481: C
482: C
483: C
484: C
485: C
486: C
487: C
488: C
489: C
490: C
491: C
492: C
493: C
494: C
495: C
496: C
497: C
498: C
499: C
500: C
501: C
502: C
503: C
504: C
505: C
506: C
507: C
508: C
509: C
510: C
511: C
512: C
513: C
514: C
515: C
516: C
517: C
518: C
519: C
520: C
521: C
522: C
523: C
524: C
525: C
526: C
527: C
528: C
529: C
530: C
531: C
532: C
533: C
534: C
535: C
536: C
537: C
538: C
539: C
540: C
541: C
542: C
543: C
544: C
545: C
546: C
547: C
548: C
549: C
550: C
551: C
552: C
553: C
554: C
555: C
556: C
557: C
558: C
559: C
560: C
561: C
562: C
563: C
564: C
565: C
566: C
567: C
568: C
569: C
570: C
571: C
572: C
573: C
574: C
575: C
576: C
577: C
578: C
579: C
580: C
581: C
582: C
583: C
584: C
585: C
586: C
587: C
588: C
589: C
590: C
591: C
592: C
593: C
594: C
595: C
596: C
597: C
598: C
599: C
600: C
601: C
602: C
603: C
604: C
605: C
606: C
607: C
608: C
609: C
610: C
611: C
612: C
613: C
614: C
615: C
616: C
617: C
618: C
619: C
620: C
621: C
622: C
623: C
624: C
625: C
626: C
627: C
628: C
629: C
630: C
631: C
632: C
633: C
634: C
635: C
636: C
637: C
638: C
639: C
640: C
641: C
642: C
643: C
644: C
645: C
646: C
647: C
648: C
649: C
650: C
651: C
652: C
653: C
654: C
655: C
656: C
657: C
658: C
659: C
660: C
661: C
662: C
663: C
664: C
665: C
666: C
667: C
668: C
669: C
670: C
671: C
672: C
673: C
674: C
675: C
676: C
677: C
678: C
679: C
680: C
681: C
682: C
683: C
684: C
685: C
686: C
687: C
688: C
689: C
690: C
691: C
692: C
693: C
694: C
695: C
696: C
697: C
698: C
699: C
700: C
701: C
702: C
703: C
704: C
705: C
706: C
707: C
708: C
709: C
710: C
711: C
712: C
713: C
714: C
715: C
716: C
717: C
718: C
719: C
720: C
721: C
722: C
723: C
724: C
725: C
726: C
727: C
728: C
729: C
730: C
731: C
732: C
733: C
734: C
735: C
736: C
737: C
738: C
739: C
740: C
741: C
742: C
743: C
744: C
745: C
746: C
747: C
748: C
749: C
750: C
751: C
752: C
753: C
754: C
755: C
756: C
757: C
758: C
759: C
760: C
761: C
762: C
763: C
764: C
765: C
766: C
767: C
768: C
769: C
770: C
771: C
772: C
773: C
774: C
775: C
776: C
777: C
778: C
779: C
780: C
781: C
782: C
783: C
784: C
785: C
786: C
787: C
788: C
789: C
790: C
791: C
792: C
793: C
794: C
795: C
796: C
797: C
798: C
799: C
800: C
801: C
802: C
803: C
804: C
805: C
806: C
807: C
808: C
809: C
810: C
811: C
812: C
813: C
814: C
815: C
816: C
817: C
818: C
819: C
820: C
821: C
822: C
823: C
824: C
825: C
826: C
827: C
828: C
829: C
830: C
831: C
832: C
833: C
834: C
835: C
836: C
837: C
838: C
839: C
840: C
841: C
842: C
843: C
844: C
845: C
846: C
847: C
848: C
849: C
850: C
851: C
852: C
853: C
854: C
855: C
856: C
857: C
858: C
859: C
860: C
861: C
862: C
863: C
864: C
865: C
866: C
867: C
868: C
869: C
870: C
871: C
872: C
873: C
874: C
875: C
876: C
877: C
878: C
879: C
880: C
881: C
882: C
883: C
884: C
885: C
886: C
887: C
888: C
889: C
890: C
891: C
892: C
893: C
894: C
895: C
896: C
897: C
898: C
899: C
900: C
901: C
902: C
903: C
904: C
905: C
906: C
907: C
908: C
909: C
910: C
911: C
912: C
913: C
914: C
915: C
916: C
917: C
918: C
919: C
920: C
921: C
922: C
923: C
924: C
925: C
926: C
927: C
928: C
929: C
930: C
931: C
932: C
933: C
934: C
935: C
936: C
937: C
938: C
939: C
940: C
941: C
942: C
943: C
944: C
945: C
946: C
947: C
948: C
949: C
950: C
951: C
952: C
953: C
954: C
955: C
956: C
957: C
958: C
959: C
960: C
961: C
962: C
963: C
964: C
965: C
966: C
967: C
968: C
969: C
970: C
971: C
972: C
973: C
974: C
975: C
976: C
977: C
978: C
979: C
980: C
981: C
982: C
983: C
984: C
985: C
986: C
987: C
988: C
989: C
990: C
991: C
992: C
993: C
994: C
995: C
996: C
997: C
998: C
999: C
1000: C

```

src/shared/regnm0.f

```
1:      subroutine REGNM0( IREG, DIRC, NAME, LNM, A, CHA )
2: C=====
3: C PURPOSE :   REGION # <==> REGION NAME   TRANSLATION.
4: C           ARGUMENT LIST IS SIMPLIFIED TO MAKE USABLE AT ANY PLACE
5: C           IN THE PROGRAM.
6: C
7: C           IREG : REGION # (GIVE OR GET)
8: C           DIRC : DIRECTION ('>/'<' = # > NAME   or # < NAME )
9: C           NAME : REGION NAME ( GET OR GIVE )
10: C          LNM  : length of name
11: C
12: C CALLED IN: TALLYO  ETC.
13: C-----
14: C arguments ( i=input, o=output, w=work )
15: C
16: C i/o  IREG : region number input/output
17: C i    DIRC : '>' -- region # to region name
18: C       '<' -- region name to region #
19: C o/i  NAME : region number output/input
20: C o    LNM  : length of region name output
21: C
22: C io  A(*) : dynamic memory array (task shared)
23: C io  CHA(*) : dynamic character memory array (task shared)
24: C=====
25:      integer IREG
26:      character*1 DIRC
27:      character*(*) NAME
28: C
29: CCC   include 'INC/_ARRAY'
30:      real A(*)
31:      character*4 CHA(*)
32: C
33:      include 'INC/_SIZES'
34: ccccc include 'INC/_FLAGS'
35:      include 'INC/_PGEOM'
36:      include 'INC/_PTLSP'
37: C
38:      call REGNAM( IREG, DIRC, NAME, LNM, IGTLG('JTLLI'), A(LKMAT),
39: &                A(LKREG), A(LKREGI), A(LKCELI), NINPZ, A(LISUSP),
40: &                A(LISPNM), A(LKSUZN), A(LIRGSP), NZONE, NSPACE, NSUZON,
41: &                NEST, NREG, CHA(LTNAMS), N NAMES )
42:      return
43:      end
```

src/shared/report.f

```
1:      subroutine REPORT( ITASK, NNHIST,TCPU,  TELAP, NLOST )
2: C=====
3: C purpose: report timing etc. after each 'ACTION' task.
4: C called in : action
5: C-----
6: C arguments (i=input, o=output, w=work)
7: C
8: C*i IOW : message output I/O unit (removed Jan 2000)
9: C i ITASK : task ID (meaningless in sigle task mode)
10: C i NNHIST : number of processed particles (histories)
11: C i TCPU : CPU time used in this task
12: C i TELAP : Elapsed time in this task
13: C i NLOST : number of lost particles
14: C=====
15:      include 'INC/_IOUNIT'
16: C-----
17:      write(IOW,7000) ITASK, NNHIST, TCPU, TELAP
18: 7000 format(1x,' == REPORT FROM TASK',I8,' == '/4x,'HISTORIES RUN ',f10
19:      &      /4x,'CPU TIME USED ',f10.3/4x,'ELAPSED TIME ',f10.3)
20: C
21:      if ( NLOST.gt.0 ) then
22:          write(IMG,7020) NLOST, FLOAT(NLOST) /NNHIST*100
23:      end if
24: 7020 format(4x,'XXX LOST particles ',I7,'( ',1PE10.3,' % ) XXXX')
25: C
26:      return
27:      end
```

src/shared/reqhst.f

```
1:      subroutine REQHST( NGENE, JTSKR )
2: C=====
3: C purpose :   In multi tasking mode of non-reproducible run ,
4: C   require histories by accessing task controller process
5: C   or by accessing global data area .
6: C called in : selone, select
7: C=====
8:      include 'INC/_IOUNIT'
9: C/#IF PARA(PVM)
10:     include 'INC/_PVMPARA'
11: C/#ELSEIF PARA(MPI)
12: *      include 'mpif.h'
13: C/#ENDIF
14:     include 'INC/_TASKDT'
15:     include 'INC/_SIZES'
16: C
17: C
18: C/#IF PARA(PVM)
19: C
20:     call PVMFINITSEND( PVMDEFAULT, INFO )
21: C
22:     call PVMFPACK( INTEGER4, IDTASK, 1, 1, INFO )
23:     call PVMFSEND( ICTASK, 98, INFO )
24: C
25: C
26:     call PVMFRCV( ICTASK, -1, IBUFID )
27: C
28:     call PVMFUNPACK( INTEGER4, NGENE, 1, 1, INFO )
29: C/#IF INTEGER8
30: *      call PVMFUNPACK( INTEGER8, NTHIST0, 1, 1, INFO )
31: C/#ELSE
32:     call PVMFUNPACK( INTEGER4, NTHIST0, 1, 1, INFO )
33: C/#ENDIF
34: C
35: C
36: C   ... for shared memry machine ...
37: C
38: C
39: C/#ELSEIF PARA( CRAY* SX* )
40: C
41: C/#IF PARA(SX* CRAY)
42: *      call MVPSYNC_LOCK( 3 )
43: C/#ENDIF
44: C
45:     NHEAP1 = MAX(0,NHHEAP-NHIST)
46: C
47:     NGENE = NHHEAP - NHEAP1
48:     NHHEAP = NHEAP1
49: C
50: C/#IF PARA(SX* CRAY)
51: *      call MVPSYNC_UNLOCK( 3 )
52: C/#ENDIF
53: C
54: C/#ELSE
55:     write(IMG,*) 'XXX Running in "NO-REPRODUCIBLE-RUN" mode,',
56: &      ' but this mode is effective only in multi-tasking mode!!'
57:     write(IMG,*) '      Terminate execution!!'
58: C
59:     stop 666
60: C/#ENDIF
61:     return
62: end
```

src/shared/rgfind.f

```

1:      subroutine RGFINDD( IPRT,  NAME,  IBEGIN,JEND,  IREG,  IERR,
2:      &                    JTLLT, KMAT,  KREG,  KREGI, KCELI, NINPZ,
3:      &                    ISPNM, ISUSP, KSUZN, IRGSP, NEST,  NSPACE,
4:      &                    NSUZON,NZONE, NREG,  TNAMS, NNAMES )
5: C=====
6: C PURPOSE: RETURN REGION NUMBER WITH NAME 'NAME'.
7: C      IBEGIN = 0 : START TRANSLATION FOR 'WILD CARD' SPECIFICATION.
8: C
9: C CALLED IN : TLOREG,REGDAT
10: C CALLS      : CBSINC,CNTERR,MATCH,REGNAM,
11: C=====
12: C arguments (i=input, o=output, w=work)
13: C
14: C i IPRT : I/O unit of message output
15: C i name : region name given (may include wildcard characters)
16: C i o ibegin :
17: C      IBEGIN = 0 : START TRANSLATION FOR 'WILD CARD' SPECIFICATION.
18: C o jend : 0 = regions that match the name remain.
19: C      1 = no more region that matches the name.
20: C o ireg : region # found.
21: C o ierr : error occured if no zero value were returned.
22: C      (invalid name or no matching region found.)
23: C i jtllt : option flag of 'TALLY-REGION' option.
24: C i kmat(ninpz) : material # of each input-zone
25: C i kreg(ninpz) : region # of each input-zone
26: C i kregi(ninpz) : region name table position of each input-zone
27: C i kceli(ninpz) : belonging cell # of each input-zone
28: C i ninpz : number of input zones
29: C i ispm(2,0:NEST,NSPACE) : Subspace name list.
30: C i isusp(NSUZON,NSPACE) : Region #'s for each terminal-input-zone
31: C i irgsp(2,NREG) :
32: C      (1,I) : Subspace # of I'th region.
33: C      (2,I) : "region"-name number in 'TNAMS' table.
34: C i NEST,NSPACE,NSUZON,NZONE,NREG : size parameters (see comments
35: C      for common /SIZES/ and /PGEOM/ )
36: C i TNAMS(NNAMES) : registered region name list.
37: C i NNAMES : number of names.
38: C-----
39: C
40:      character*(*) NAME
41: C
42:      integer KREG(NINPZ), KREGI(NINPZ), KCELI(NINPZ)
43:      integer ISPNM(2,0:NEST,NSPACE), ISUSP(NSUZON,NSPACE),
44:      &          KSUZN(NINPZ,2), IRGSP(2,NREG), KMAT(NINPZ)
45:      include 'INC/_LNAME'
46:      character*(LNAM) TNAMS(NNAMES)
47: C
48: C .... LOCAL STATIC VARIABLES ... ( SAVED BETWEEN CALLS ) ...
49: C
50:      parameter( MAXLVL = 4, MX2 = MAXLVL*2 )
51: C
52:      character*(LNAM) FRAME(MAXLVL), SFRAME(MAXLVL), REGION
53:      integer NAML(2,MAXLVL)
54: C
55:      data MLVL /0/, IZ0 /0/, LNM0 /0/
56:      data KWILD, KLV /0, 0/
57:      data NAML /MX2*0/
58:      data FRAME /MAXLVL*' '/
59:      data SFRAME /MAXLVL*' '/
60:      data REGION /' '/
61:      data JNXTSP, NMATCH, IZSP, IZSU /0, 0, 0, 0/
62: C
63: C
64:      IERR = 0
65:      JEND = 0

```

```

66: C
67:      if ( IBEGIN.eq.0 ) then
68:      LNM0 = INDEX(NAME,' ') - 1
69:      if ( LNM0.lt.0 ) LNM0 = LEN(NAME)
70:      KWILD = INDEX(NAME(:LNM0),'*') + INDEX(NAME(:LNM0),'?')
71:      &      + INDEX(NAME(:LNM0),'!') + INDEX(NAME(:LNM0),'!')
72:      KLV = INDEX(NAME(:LNM0),':')
73: C
74: C
75: C ===== ONLY ONE CORRESPONDING REGION FOR THE NAME. =====
76: C
77: C
78: C
79: C ..... SIMPLE 0 LEVEL REGION .....
80: C
81: CM      IF( KWILD .EQ. 0 .AND. KLV .EQ. 0 ) THEN
82: CM      II = INDEX( NAME(:LNM0) , '!' ) + 1
83: CM      JEND = 1
84: C
85: CM      CALL CBSINC( TNAMS, NNAMES, NAME(II:LNM0), IPOS )
86: CM      IF( IPOS .EQ. 0 ) THEN
87: CM      IREG = 0
88: CM      IERR = 1
89: CM      GOTO 9000
90: CM      ENDIF
91: C
92: CM      IF( JTLLT.EQ.0 ) THEN
93: CM      DO 100 I=1,NINPZ
94: CM      IF( IPOS .EQ. KREGI(I) ) THEN
95: CM      IREG = KREG(I)
96: CM      IF( IREG .EQ. 0 ) IERR = 1
97: CM      GOTO 9000
98: CM      ENDIF
99: CM100      CONTINUE
100: CM
101: CM      ELSE
102: CM      DO 110 I=1,NINPZ
103: CM      IF( KCELI(I).EQ.0 .AND. IPOS.EQ.KREGI(I) ) THEN
104: CM      IREG = KREG(I)
105: CM      IF( IREG .EQ. 0 ) IERR = 1
106: CM      GOTO 9000
107: CM      ENDIF
108: CM      CONTINUE
109: CM      ENDIF
110: CM      IREG = 0
111: CM      IERR = 1
112: CM      GOTO 9000
113: CM      ENDIF
114: C
115: C ..... NO WILDCARD & LEVEL > 0 SPACE .....
116: C
117: C
118: CM      if ( KWILD.eq.0 ) then
119: CM      IF( JTLLT .EQ. 0 ) THEN
120: CM      IREG = 0
121: CM      IERR = 1
122: CM      JEND = 1
123: CM      GOTO 9000
124: CM      ENDIF
125: C
126: C      call REGNAM( IREG, '<', NAME(:LNM0), LNM, JTLLT, KMAT, KREG,
127: C      &          KREGI, KCELI, NINPZ, ISUSP, ISPNM, KSUZN, IRGSP,
128: C      &          NZONE, NSPACE, NSUZON, NEST, NREG, TNAMS, NNAMES )
129: C
130: C      if ( IREG.eq.0 ) IERR = 1
131: C      JEND = 1

```


src/shared/rgfind.f

```

131:          go to 180
132:        end if
133: C
134: C
135: C ===== RGFIND SHOULD BE CALLED MORE THAN ONCE FOR THE NAME. =====
136: C
137: C
138:        IBEGIN = IBEGIN + 1
139: C
140: C ..... WILDCARD SPECIFICATION .....
141: C
142:        NMATCH = 0
143: C
144: C ..... LEVEL 0 .....
145: C
146:        if ( KLV.eq.0 ) then
147:          IZ0 = 0
148: C
149: C ..... NON-0 LEVEL .....
150: C
151:        else
152:          JNXTSP = 1
153:          IZSP = 0
154: C
155: C ... SEPARATE NAME ( CHECK VALIDITY ALSO ) ...
156: C
157:          LP = 1
158:          MLVL = 0
159: C
160: 100      continue
161:          if ( NAME(LP:LP).eq.'!' ) LP = LP + 1
162: C
163:          IK = INDEX(NAME(LP:LNMO),':')
164: C
165:          if ( IK.ge.1 ) then
166:            MLVL = MLVL + 1
167:            if ( MLVL.gt.NEST ) then
168:              write(IPRT,*) 'XXX INVALID REGION SPECIFICATION',
169:                & ' (TOO DEEPLY NESTED)'
170:              write(IPRT,*) ' NAME: <', NAME(:LNMO), '>'
171:              call CNTERR( 'FATAL' )
172:              IREG = 0
173:              JEND = 1
174:              IERR = 1
175:              go to 180
176:            end if
177: C
178:            if ( IK.gt.1 ) then
179:              FRAME(MLVL) = NAME(LP:LP+IK-2)
180:              NAML(1,MLVL) = IK - 1
181:            else
182:              FRAME(MLVL) = '*'
183:              NAML(1,MLVL) = 1
184:            end if
185:            LP = LP + IK
186: C
187:            IKK = INDEX(NAME(LP:LNMO), '!')
188:            if ( IKK.ge.1 ) then
189:              if ( IKK.gt.1 ) then
190:                SFRAME(MLVL) = NAME(LP:LP+IKK-2)
191:                NAML(2,MLVL) = IKK - 1
192:              else
193:                SFRAME(MLVL) = '*'
194:                NAML(2,MLVL) = 1
195:              end if

```

```

196:        else
197:          write(IPRT,*) 'XXX INVALID REGION SPECIFICATION'
198:          write(IPRT,*) ' NAME: <', NAME(:LNMO), '>'
199:          call CNTERR( 'FATAL' )
200:          IREG = 0
201:          JEND = 1
202:          IERR = 1
203:          go to 180
204:        end if
205:        LP = LP + IKK
206:      else
207:        if ( LP.gt.LNMO ) then
208:          write(IPRT,*) 'XXX INVALID REGION SPECIFICATION'
209:          write(IPRT,*) ' NAME: <', NAME(:LNMO), '>'
210:          call CNTERR( 'FATAL' )
211:          IREG = 0
212:          JEND = 1
213:          IERR = 1
214:          go to 180
215:        end if
216:        REGION = NAME(LP:LNMO)
217:        go to 110
218:      end if
219:      go to 100
220: 110      continue
221:    end if
222:  end if
223: C
224: C
225: C .... RETURN MATCHING REGION FOR WILDCARD CASE ...
226: C
227: C -----
228: C
229: C ==== 0 LEVEL REGION ===
230: C
231: C -----
232: C
233:        if ( KLV.eq.0 ) then
234:          IZ0 = IZ0 + 1
235:          II = INDEX(NAME(:LNMO), '!') + 1
236:          do 120 I = IZ0, NINPZ
237:            if ( KMAT(I).ge.0.and.KREG(I).ne.0
238:              & .and.(JTLLT.eq.0.or.(JTLLT.ne.0.and.KCELI(I).eq.0)) )
239:              & then
240:                call MATCH( IM, NAME(II:LNMO), TNAMS(KREGI(I)) )
241: C
242: C ..... MATCHED .....
243: C
244:          if ( IM.ne.0 ) then
245:            IREG = KREG(I)
246:            NMATCH = NMATCH + 1
247:            IZ0 = I
248:            go to 180
249:          end if
250:        end if
251: 120      continue
252:          IREG = 0
253:          JEND = 1
254:          if ( NMATCH.eq.0 ) then
255:            write(IPRT,*) 'XXX NO MATCHING REGION <', NAME(:LNMO),
256:              & '> (RGFIND)'
257:            call CNTERR( 'FATAL' )
258:            IERR = 1
259:          end if
260:          go to 180

```

src/shared/rgfind.f

```

261:      else
262: C-----
263: C
264: C ==== NON-0 LEVEL REGION ====
265: C
266: C JNXTSP = 1/0 : FIND NEXT(OR FIRST) MATCHING SUBSPACE / NO
267: C IZSP       : CURRENT SUBSPACE #
268: C IZSU       : CURENT "terminal"-zone #
269: C             (A "terminal"-zone is an input-zone
270: C             in lattice-cell and for which tallies can be taken
271: C             (mat ID is non-negative) as regions in subspaces.)
272: C NMATCH     : COUNT OF MATCHED REGION
273: C-----
274: C 130 continue
275: C
276: C   if ( JNXTSP.eq.1 ) then
277: C     IZSP = IZSP + 1
278: C     do 150 IS = IZSP, NSPACE
279: C       if ( ISPNM(1,0,IS).eq.MLVL ) then
280: C         do 140 K = 1, MLVL
281: C           call MATCH( IM, FRAME(K)(:NAML(1,K)),
282: C             &         TNAMS(ISPNM(1,K,IS)) )
283: C           if ( IM.eq.0 ) go to 150
284: C           call MATCH( IM2, SFRAME(K)(:NAML(2,K)),
285: C             &         TNAMS(ISPNM(2,K,IS)) )
286: C           if ( IM2.eq.0 ) go to 150
287: C         140 continue
288: C       C
289: C         IZSP = IS
290: C         IZSU = 0
291: C         JNXTSP = 0
292: C         go to 160
293: C       end if
294: C     150 continue
295: C       IREG = 0
296: C       JEND = 1
297: C       if ( NMATCH.eq.0 ) then
298: C         write(IPRT,*)'XXX NO MATCHING REGION <' NAME(:LNMO), '>'
299: C         call CNTERR( 'FATAL' )
300: C         IERR = 1
301: C       end if
302: C       go to 180
303: C     end if
304: C   C
305: C   160 IZSU = IZSU + 1
306: C   do 170 ISU = IZSU, NSUZON
307: C     IR = ISUSP(ISU,IZSP)
308: C     if ( IR.ne.0 ) then
309: C       call MATCH( IM, REGION, TNAMS(KREGI(KSUZN(ISU,1))) )
310: C       if ( IM.ne.0 ) then
311: C         NMATCH = NMATCH + 1
312: C         IREG = IR
313: C         IZSU = ISU
314: C         go to 180
315: C       end if
316: C     end if
317: C   170 continue
318: C   JNXTSP = 1
319: C   go to 130
320: C   end if
321: C
322: C 180 continue
323: C   end

```

src/shared/rskip.f

```
1:      subroutine RSKIP( NRSKIP,IRAND )
2: C=====
3: C PURPOSE: SKIP RANDOM NUMBER.
4: C CALLED IN: ACTION
5: C-----
6: C arguments (i=input, o=output, w=work)
7: C
8: C i NRSKIP : number of random numbers to be skipped
9: C io IRAND : seed of random numbers
10: C=====
11:      parameter( NN      = 256 )
12:      real DUMMY(NN)
13: C
14:      NR      = ABS(NRSKIP)
15:      do 100 N = 1, NR, NN
16:          N2      = MIN(NN,NR-N+1)
17:          call RANU2( IRAND, DUMMY, N2, ICON )
18:      100 continue
19:      return
20:      end
```

src/shared/rstbod.f

```

1:      subroutine RSTBOD( NBODY, IBSDA, GRBOD, NBODI,
2:      &                  IBODI, DBODI, IPSDA, NSURF, SDA,   NSDA )
3:      C=<MVP/GMVP>=====
4:      C PURPOSE: restore and check body information from working file (IUB)
5:      C          and compose surface data pointer array
6:      C CALLED IN: GEOMIN
7:      C-----
8:      C arguments (i=input, o=output, w=work)
9:      C
10:     C*i IUB : I/O unit of working file which contains body input data
11:     C*      stored in GEOMIN routine.
12:     C*i IOG : message output I/O unit
13:     C i NBODY : number of bodies
14:     C o IBSDA(3,NBODY+1) : data to access body data (see below)
15:     C o GRBOD(6,NBODY) : x,y,z coordinate range of each body
16:     C i NBODI : length of array DBODI
17:     C o IBODI(NBODI+1) : pointer to DBODI array for each body
18:     C o DBODI(NBODI) : body geometry data as input
19:     C o IPSDA(3,NSURF+1): data to access body information from surface data
20:     C i SDA(NSDA): surface shape data
21:     C-----
22:     integer IBSDA(3,NBODY+1)
23:     real*8 GRBOD(6,NBODY)
24:     integer IBODI(NBODY+1)
25:     real*8 DBODI(NBODI)
26:     C
27:     integer IPSDA(3,NSURF+1)
28:     real*8 SDA(NSDA)
29:     C
30:     C ... IUB, IOG and IMSG are passed by include
31:     include 'INC/_IOUNIT'
32:     C
33:     C*** IBSDA(1,I): POSITION OF HEAD OF I'TH BODY IN SDA ARRAY
34:     C IBSDA(1,I): surface # of the first surface of the body.
35:     C IBSDA(2,I): body ID of I'th body.
36:     C IBSDA(3,I): body type # of I'th body.
37:     C
38:     C GRBOD(6,I) : x,y,z coordinate range of each body.
39:     C
40:     character*3 BTYPE
41:     C
42:     C-----
43:     C
44:     call RWIND( IUB )
45:     C
46:     IBODI(1) = 1
47:     do 100 I = 1, NBODY
48:       read(IUB) IBSDA(1,I), IBSDA(2,I), IBSDA(3,I),
49:       &          (GRBOD(J,I),J=1,6), NBI
50:       IBODI(I+1) = IBODI(I) + NBI
51:       read(IUB) (DBODI(J),J=IBODI(I),IBODI(I+1)-1)
52:     100 continue
53:     IBSDA(1,NBODY+1) = NSURF + 1
54:     C
55:     C-----
56:     C Check duplicate use of Body ID ( IBSDA(2,i) ) ...
57:     C-----
58:     C
59:     C ... use IMSG for message output
60:     call CHKDUP( IBSDA(2,1), NBODY, 3, NERR, IMSG, 'BODY ID' )
61:     if ( NERR.gt.0 ) call CNTERR( 'FATAL' )
62:     C
63:     C-----
64:     C ..... compose IPSDA(3,*) (this position may not be appropriate for
65:     C the operation...)

```

```

66:     C-----
67:     IP = 1
68:     do 110 I = 1, NSURF
69:       IPSDA(1,I) = IP
70:     C
71:     C ... first SDA(*) element for each surface has
72:     C 10000*<number of data for the surface> + <surface type ID>
73:     C
74:     M = NINT(SDA(IP)/10000.0D0)
75:     IP = IP + M
76:   110 continue
77:     IPSDA(1,NSURF+1) = IP
78:     C
79:     NBBC = 0
80:     C
81:     do 130 IB = 1, NBODY
82:     C
83:       call TYPBOD( BTYPE, '<', IBSDA(3,IB) )
84:       if ( BTYPE.eq.'BBC' ) NBBC = NBBC + 1
85:     C
86:     MM = NSURF
87:     if ( IB.lt.NBODY ) MM = IBSDA(1,IB+1) - 1
88:     do 120 J = IBSDA(1,IB), MM
89:       IPSDA(2,J) = IBSDA(2,IB)
90:       IPSDA(3,J) = IB
91:   120 continue
92:   130 continue
93:     IPSDA(2,NSURF+1) = 0
94:     IPSDA(3,NSURF+1) = 0
95:     C
96:     C-----
97:     C check "BBC" body if any, and DBODI's from body ID to body#
98:     C-----
99:     C
100:    if ( NBBC.gt.0 ) then
101:      do 170 IB = 1, NBODY
102:        call TYPBOD( BTYPE, '<', IBSDA(3,IB) )
103:        if ( BTYPE.eq.'BBC' ) then
104:          do 160 K = IBODI(IB), IBODI(IB+1) - 1
105:            KB = NINT(DBODI(K))
106:            do 140 KK = 1, NBODY
107:              if ( ABS(KB).eq.IBSDA(2,KK) ) go to 150
108:            140 continue
109:          C
110:          write(IMSG,7000) IBSDA(2,IB), ABS(KB)
111:          call CNTERR( 'FATAL' )
112:          DBODI(K) = 1
113:          go to 160
114:        C
115:      150 continue
116:      DBODI(K) = KK*SIGN(1,KB)
117:      GRBOD(1,IB) = MIN(GRBOD(1,IB),GRBOD(1,KB))
118:      GRBOD(2,IB) = MAX(GRBOD(2,IB),GRBOD(2,KB))
119:      GRBOD(3,IB) = MIN(GRBOD(3,IB),GRBOD(3,KB))
120:      GRBOD(4,IB) = MAX(GRBOD(4,IB),GRBOD(4,KB))
121:      GRBOD(5,IB) = MIN(GRBOD(5,IB),GRBOD(5,KB))
122:      GRBOD(6,IB) = MAX(GRBOD(6,IB),GRBOD(6,KB))
123:    C
124:    call TYPBOD( BTYPE, '<', IBSDA(3,KB) )
125:    C
126:    if ( BTYPE.eq.'BBC' ) then
127:      write(IMSG,7020) IBSDA(2,IB), ABS(KB)
128:      call CNTERR( 'FATAL' )
129:    else if ( KB.lt.0.and.BTYPE.ne.'HAF' ) then
130:      write(IMSG,7040) IBSDA(2,IB), ABS(KB), BTYPE

```

src/shared/rstbod.f

```
131:          call CNTERR( 'FATAL' )
132:          else if ( KB.lt.0.and.BTYPE.ne.'ELT' ) then
133:              write(IMG,7060) IBSDA(2,IB), ABS(KB)
134:              call CNTERR( 'FATAL' )
135:          end if
136: C
137: 160          continue
138:          end if
139: 170          continue
140:          end if
141: C
142: 7000 format(1X,'XXX Combined type body (BBC) ',I5,
143:           & ' is using undefined body ID ',I5)
144: 7020 format(1X,'XXX Combined type body (BBC) ',I5,
145:           & ' is defined using other combined type body ',I5)
146: 7040 format(1X,'XXX Combined type body (BBC) ',I5,' is using body ',I5,
147:           & ' of type ',A,' with minus sign, '/1X,
148:           & ' but only a "HAF" type body is allowed for such a use.')
149: 7060 format(1X,'XXX Combined type body (BBC) ',I5,
150:           & ' cannot be defined using "ELT" type body (ID ',I5,')')
151: C
152:          return
153:          end
```

src/shared/rvlatt.f

```
1:      subroutine RVLATT( NX,      NY,      NZ,      KLATT )
2: C=====
3: C PURPOSE:  REVERSE ORDER IN ARRAY KLATT IN Y & Z INDEX
4: C CALLED IN:  PRLAT0          CALLS:  NONE
5: C===== 3/13/1991 =====
6:      integer KLATT(NX,NY,NZ)
7: C
8:      if ( NY.gt.1.and.NZ.gt.1 ) then
9:          NY1      = NY + 1
10:         NZ1      = NZ + 1
11:         NZH      = NZ / 2
12:         do 120 K = 1, NZ
13:             if ( K.le.NZH ) then
14:                 NYY = NY1 / 2
15:             else
16:                 NYY = NY / 2
17:             endif
18:             do 110 J = 1, NYY
19:                 do 100 I = 1, NX
20:                     KL      = KLATT(I,J,K)
21:                     KLATT(I,J,K)      = KLATT(I,NY1-J,NZ1-K)
22:                     KLATT(I,NY1-J,NZ1-K)      = KL
23:                 100      continue
24:             110      continue
25:         120      continue
26:     else if ( NZ.eq.1.and.NY.gt.1 ) then
27:         NY1      = NY + 1
28:         do 140 J = 1, NY/2
29:             do 130 I = 1, NX
30:                 KL      = KLATT(I,J,1)
31:                 KLATT(I,J,1)      = KLATT(I,NY1-J,1)
32:                 KLATT(I,NY1-J,1)      = KL
33:             130      continue
34:         140      continue
35:     else if ( NZ.gt.1.and.NY.eq.1 ) then
36:         NZ1      = NZ + 1
37:         do 160 K = 1, NZ/2
38:             do 150 I = 1, NX
39:                 KL      = KLATT(I,1,K)
40:                 KLATT(I,1,K)      = KLATT(I,1,NZ1-K)
41:                 KLATT(I,1,NZ1-K)      = KL
42:             150      continue
43:         160      continue
44:     end if
45:     return
46: end
```

src/shared/rxpdet.f

```

1:      subroutine RXPDET( A,      JVMNT, JLATT, JSIMP, JHLAT, JEIGN,
2:      &                  TEMPXP,XPDET, NPDET, NPLEN, NEST,
3:      &                  IPDET, IPDT2, JSPDT,
4:      &                  IPCEL, KCELL, KDALT, KZMAT, MLBZZ,
5:      &                  KZREG, ISUSP, KSPSU, KTCSP,
6:      &                  NSDA, NZDA, NZONE, NCELL, NLBZ ,
7:      &                  NSPACE,NSUZON,NUNV, NKTCSF )
8:      C=====
9:      C PURPOSE:  Process XPDET array for use in random walk
10:     C      and calculate zone # & lattice-related parameters if necessary.
11:     C
12:     C Called in:  INTRO
13:     C Called by: PT2ZON
14:     C=====
15:     C
16:     C IPDET(NP,1) : ZONE # OR LATTICE BUFFER ZONE #
17:     C      IN WHICH NP'TH POINT DETECTOR LIES.
18:     C IPDET(NP,2) : LATTICE LEVEL NUMBER OF NP'TH POINT DETECTOR POSITION.
19:     C
20:     C IPDT2(L,1,NP) : SAME MEANINGS AS 'LZZ' IN PARTICLE BANK.
21:     C      IN L'TH LATTICE LEVEL FOR NP'TH POINT DETECTOR.
22:     C IPDT2(L,2,NP) : SAME MEANINGS AS 'LPOS' IN PARTICLE BANK.
23:     C      IN L'TH LATTICE LEVEL FOR NP'TH POINT DETECTOR.
24:     C IPDT2(L,3,NP) : SAME MEANINGS AS 'LCRS' IN PARTICLE BANK.
25:     C      IN L'TH LATTICE LEVEL FOR NP'TH POINT DETECTOR.
26:     C
27:     C=====
28:     real A(*)
29:     C
30:     include 'INC/_LISTOFF'
31:     include 'INC/_PGEOM'
32:     include 'INC/_LISTON'
33:     include 'INC/_IOUNIT'
34:     C
35:     real*8 TEMPXP(5,NPDET), XPDET(NPLEN,NPDET)
36:     integer IPDET(NPDET,2), IPDT2(NEST,3,NPDET), JSPDT(NPDET)
37:     C
38:     integer IPCEL(NCELL), KCELL(NZONE), KDALT(NLBZ), KZMAT(NZONE),
39:     &      MLBZZ(NZONE)
40:     INTEGER KZREG(NZONE),ISUSP(NSUZON,NSPACE),
41:     &      KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSF)
42:     C
43:     C ..... MOVE XPDET TO PERMANENT LOCATION FROM TEMPORARY ARRAY .....
44:     C
45:     do 100 N = 1, NPDET
46:       XPDET(3,N) = TEMPXP(1,N)
47:       XPDET(4,N) = TEMPXP(2,N)
48:       XPDET(5,N) = TEMPXP(3,N)
49:       XPDET(1,N) = TEMPXP(4,N)
50:       XPDET(2,N) = TEMPXP(5,N)
51:     C
52:     C .... ZONE # , CELL COORDINATES & LATTICE PARAMETERS
53:     C
54:     call PT2ZON( A, JVMNT, JLATT, JSIMP, JHLAT,
55:     &      XPDET(3,N), XPDET(4,N), XPDET(5,N),
56:     &      XPDET(6,N), N, IPDET(N,1), IREG, IPDET(N,2),
57:     &      IPDT2(1,1,N), IPDT2(1,2,N), IPDT2(1,3,N),
58:     &      KZREG, IPCEL, KCELL, KDALT, KZMAT, MLBZZ,
59:     &      ISUSP, KSPSU, KTCSP,
60:     &      NSDA, NZDA, NZONE, NEST, NCELL, NLBZ,
61:     &      NSPACE,NSUZON,NUNV, NKTCSF )
62:   100 continue
63:     C
64:     C .... PRINTOUT
65:     C

```

```

66:     C
67:     C *****
68:     call LABEL( IPR, 'POINT DETECTOR INFORMATION' )
69:     C *****
70:     C
71:     write(IPR,'(//''      * NUMBER OF DETECTORS = '' ,I3//)') NPDET
72:     C
73:     do 110 N = 1, NPDET
74:       write(IPR,7000) N, (XPDET(I,N),I=3,5), (XPDET(I,N),I=1,2),
75:       &      IPDET(N,1)
76:     C
77:     C .... PRINT LATTICE RELATED PARAMETERS IF ANY. ...
78:     C
79:     if ( JLATT.ne.0 ) then
80:       if ( IPDET(N,2).ne.0 ) then
81:         write(IPR,7020) IPDET(N,2),
82:         &      (XPDET(2+3*(IPDET(N,2))+I,N),I=1,3)
83:         write(IPR,7040)
84:         &      (L,IPDT2(L,1,N),IPDT2(L,2,N),IPDT2(L,3,N),L=1,
85:         &      IPDET(N,2))
86:         C
87:         if ( IPDT2(IPDET(N,2),2,N).eq.-1 ) then
88:           write(IPR,
89:           &      '(/T10,' '( IN STATISTICAL GEOMETRY REGION )'')')
90:           IPDET(N,1) = MLBZZ(IPDT2(IPDET(N,2),1,N))
91:           end if
92:         else
93:           write(IPR,'(/T10,' '( NOT IN ANY LATTICE REGION )'')')
94:           end if
95:         end if
96:         if ( JSPDT(N).eq.0 ) then
97:           write(IPR,*) '      *** ISOTROPIC SOURCE IS ASSUMED ***'
98:         else if ( JSPDT(N).eq.1 ) then
99:           write(IPR,*)
100:          &      '      *** ANGULAR DISTRIBUTION OF SOURCE IS TAKEN ***'
101:         else
102:           write(IPR,*)
103:          &      '      *** NO CONTRIBUTION FROM SOURCE IS CALCULATED ***'
104:         end if
105:       110 continue
106:       write(IPR,'(//)')
107:     C
108:     C
109:   7000 format(//6X,'==== DETECTOR ',I3,2X,10('====')
110:     &      //6X,' POSITION (',1P,3E13.5,') '
111:     &      /6X,' (IN ABSOLUTE COORDINATES)'
112:     &      //6X,' DISTANCE TO REJECT TOO DISTANT COLLISION POINT',
113:     &      T65,' ': ',E12.5
114:     &      /6X,' DISTANCE TO AVOID 1/R**2 SINGULARITY ',
115:     &      T65,' ': ',E12.5
116:     &      //6X,' * ZONE # IN WHICH THE DETECTOR LIES : ',I5/)
117:     C
118:   7020 format(6X,' * LATTICE PARAMETERS '
119:     &      //6X,' LEVEL IN LATTICE-GEOMETRY : ',I3
120:     &      /6X,' IN CELL COORDINATES (',1P,3E13.5,') '//)
121:   7040 format(/6X,' LEVEL | ZONE# IN | CELL | FRAME '
122:     &      /6X,' # | UPPER LVL. | POSITION | X-ING FLAG'
123:     &      /6X,' -----'
124:     &      /(6X,7X,I3,T18,' | ',T22,I4,T31,' | ',T35,I4,T44,' | ',T48,I4))
125:     C
126:     return
127:     end

```

src/shared/scfinp.f

```
1:      subroutine SCFINP( ISRCSP, SRCSP, DSRCS, PROG,  MODE,  JDEBG,
2:      &                  ISOURL, NSVECT, SFILE, IOSFIL, NUUSE,  NUCID, NUC,
3:      &                  IERR )
4: C=====
5: C purpose: check contents of particle source file and input contents
6: C   on source information data (on I/O unit IOSI if not changed by
7: C   FILE or IOSFIL arguments).
8: C
9: C   Currently, only input from fission source file is possible.
10: C
11: C called from: SMPINF
12: C calls : packpt
13: C=====
14: C
15: C arguments ( o=output, i=input, io=output/input, w=work, c=constant)
16: C
17: C io isrcsp : data packet container for source information (integer)
18: C io srcsp  : data packet container for source information (real)
19: C io dsrscsp : data packet container for source information (real*8)
20: C
21: C (isrcsp,srcsp & dsrscsp start at ther same memory address.)
22: C
23: C i prog : program name ('MVP','GMVP' etc.)
24: C i mode : file mode (currently only 'FISSION' is allowed)
25: C i isour : current source set #
26: C
27: C i nsvect: number of vectors in " @( ... ) = "
28: C
29: C   for "FISSION" mode, if nsvect=7, sampling of energy is done
30: C   by #FISSIONFILE function and source file must contain
31: C   collided nulide and colliding energy fro sources.
32: C
33: C i sfile: if non blank, filename of source file other than
34: C   the default file on I/O unit IOSI.
35: C i iosfil: if non negative, I/O unit of source file other than
36: C   the default I/O unit IOSI.
37: C io nuse : use upto NUUSE sources in the file if positive.
38: C   number of actually used sources are returned.
39: C
40: C i nucid(nuc) : nuclide ID ( prog='MVP' )
41: C i nuc : nuclide of nuclide in probrem ( prog='MVP' )
42: C
43: C o ierr : returns nonzero if error occured.
44: C
45: C=====
46: C
47: C Structure of source file:
48: C
49: C   Data are stored in a binary file:
50: C
51: C * record 1 : file header (32 character) & number of header records
52: C
53: C   '%MVP/GMVP SOURCE FILE%', NHREC
54: C
55: C * record 2 : file version (32 character)
56: C
57: C   'Version 1.0' etc.
58: C
59: C * record 3 : Problem title of calculation which output this file
60: C   (72*2 characters)
61: C
62: C * record 4 : date time code etc. (64 character)
63: C
64: C * record 5-NHREC : dummy records including at least 32 characters
65: C   for future use if NHREC > 4.
```

```
66: C
67: C Order of records after record NHREC is not fixed.
68: C The following combination of records are placed.
69: C
70: C   * record including data name etc. (32 character)
71: C   * record(s) including data
72: C=====
73: C   include 'INC/_IOUNIT'
74: C/#IF PARA(PVM)
75: C   include 'INC/_PVMPARA'
76: C/#ELSEIF PARA(MPI)
77: C   include 'mpif.h'
78: C/#ENDIF
79: C   include 'INC/_TASKDT'
80: C
81: C   integer ISRCSP(*)
82: C   real SRCSP(*)
83: C   real*8 DSRCS(*)
84: C
85: C   integer JDEBG(*)
86: C
87: C   character*(*) PROG
88: C   character*(*) MODE
89: C   character*(*) SFILE
90: C
91: C   integer ISOURL
92: C   integer NSVECT
93: C
94: C   character*16 NUCID(NUC)
95: C
96: C   integer IERR
97: C
98: C   ..... local data ....
99: C
100: C   parameter( NUCMAX = 300 )
101: C   character*16 NUCIDF(NUCMAX)
102: C   integer KNUCID(NUCMAX)
103: C   character*256 FILENM
104: C   character*256 CWRK
105: C   character*72 TITL(2)
106: C   character*4 PTYPR
107: C   character*8 FPROG
108: C   integer IWRK(32)
109: C   logical OPD, NMD
110: C
111: C   parameter( LHEADS = 32 )
112: C   character*(LHEADS) HEADER
113: C   character*(LHEADS) TAG
114: C
115: C-----
116: C
117: C   IERR = 0
118: C
119: C   write(IPR,7000)
120: C   7000 format(/1X,'==== paritcle source file ===/')
121: C
122: C/#IF PARA(PVM MPI)
123: C
124: C   ... only main task inputs from file ....
125: C
126: C   if ( IDTASK.eq.1 ) then
127: C/#ENDIF
128: C
129: C   .... colliding nuclide/energy information is necessary if
130: C   7'th vector is sampled in MVP
```


src/shared/scfinp.f

```

131: C
132: JNUCS = 0
133: if ( MODE.eq.'FISSION'.and.PROG.eq.'MVP'.and.NSVECT.ge.8 ) then
134: JNUCS = 1
135: end if
136: C
137: C ..... change I/O unit or open file if necessary ...
138: C
139: IUNIT = IOSI
140: C
141: if ( IOSFIL.gt.0 ) then
142: IUNIT = IOSFIL
143: else if ( SFILE.ne.' ' ) then
144: C
145: C ... find an unopened I/O unit from unit 31 to 50 ...
146: C
147: call IOUNUD( IUNIT, 31, 50 )
148: if ( IUNIT.lt.0 ) then
149: write(IMG,*) 'XXX(SCFINP) Cannot find unopened I/O unit',
150: & ' to open source file.'
151: go to 290
152: end if
153: C
154: call OPENF( IUNIT, SFILE, 'UNFORMATTED', 'OLD', 'READ', IRET
155: & )
156: if ( IRET.ne.0 ) then
157: write(IMG,*) 'XXX(SCFINP) Cannot open source file.',
158: & ' error code : ', IRET
159: go to 290
160: end if
161: end if
162: C
163: FILENM = ' '
164: inquire(IUNIT,name =FILENM,opened =OPD)
165: if ( .not.OPD ) then
166: write(IMG,*)
167: & 'XXX(SCFINP) source file I/O unit is not opened.'
168: go to 290
169: end if
170: C
171: write(IPR,7020) IUNIT, FILENM(:ICLEN2(FILENM))
172: 7020 format(1X,' I/O unit: ',I3/1X,' File name: <'A,'>'
173: & )
174: C
175: C=====
176: C ... check contents of file
177: C
178: call RWIND( IUNIT )
179: C
180: C >>> Record 1 :
181: C
182: IREC = 1
183: read(IUNIT,end =250,err =260,iostat =IOS) HEADER, NHREC
184: C
185: FPROG = ' '
186: if ( HEADER.eq.'%MVP SOURCE FILE%' ) then
187: FPROG = 'MVP'
188: else if ( HEADER.eq.'%GMVP SOURCE FILE%' ) then
189: FPROG = 'GMVP'
190: end if
191: C
192: if ( FPROG.eq.' ' ) then
193: write(IMG,*) 'XXX(SCFINP) not a MVP/GMVP source file !'
194: write(IMG,*) ' Bytes in the first record : <', HEADER, '>'
195: go to 290

```

[illegible]

src/shared/scfinp.f

```

261: C
262:       NPSIZ  = 16
263:       do 110 I = 1, NPSIZ
264:         IWRK(I) = -999
265:       110 continue
266: C
267:       120 IREC  = IREC + 1
268:       read(IUNIT,end =130,err =260,iostat =IOS) TAG
269: C
270: C
271: C >>> "SIZES" /
272: C      ns : number of sources in this file
273: C
274:       if ( TAG.eq.'SIZES' ) then
275:         read(IUNIT,end =270,err =280,iostat =IOS) NS
276:         write(IPR,7120) NS
277:       7120 format(1X,'      number of sources in file: ',I8)
278: C
279:       if ( NUSE.eq.0 ) then
280:         NUSE  = NS
281:       else if ( NUSE.gt.0 ) then
282:         if ( NUSE.ne.NS ) then
283:           write(IMG,*)
284:             &      '!!! NUSE is not equal to number of sources',
285:             &      'in the file.'
286:         end if
287:         NUSE  = MIN(NS,NUSE)
288:       end if
289: C
290:       write(IPR,7140) NUSE
291:     7140 format(1X,'      number of sources to be used: ',I8)
292: C
293: C      ... make data packets by saving positions
294: C
295:       call PACKLB( SRCSP, 'LABEL FISSION', '-FISSION-FILE-', IRET
296:       &
297:       call PACKND( SRCSP, 'NPSIZ', 'I4', NPSIZ, 1, IRET )
298:       call PACKPT( SRCSP, 'SIZES', 'I4', LPSIZ, NPSIZ, IRET )
299:       call PACKPT( SRCSP, 'X', 'R8', LPX, NUSE, IRET )
300:       call PACKPT( SRCSP, 'Y', 'R8', LPY, NUSE, IRET )
301:       call PACKPT( SRCSP, 'Z', 'R8', LPZ, NUSE, IRET )
302: c##<2007/03/14:PN3:
303: c##
304:       call PACKPT( SRCSP, 'W', 'R4', LPW, NUSE, IRET )
305:       call PACKPT( SRCSP, 'W', 'R4', LPW, NUSE*2, IRET )
306: c##
307:       if ( JNUCS.ne.0 ) then
308:         call PACKPT( SRCSP, 'INUF', 'I4', LPINUF, NUSE, IRET )
309:         call PACKPT( SRCSP, 'EEEE', 'R4', LPEEEE, NUSE, IRET )
310: C
311: C      ... NNUCID & KNUCID will be packed.
312: C
313:       end if
314:       if ( IRET.ne.0 ) then
315:         write(IMG,*)
316:         &      'XXX(SCFINP) Failed to make space in memory ',
317:         &      'to store source information from file.'
318:       go to 290
319:       end if
320: C
321: C >>> "NUCLIDE ID" /
322: C      nucs : number of nuclides whose ID is stored in this file
323: C      nucidf(:nucs) : nuclide ID's (character*16)
324: C
325:       else if ( TAG.eq.'NUCLIDE ID' ) then

```

```

326:       if ( JNUCS.eq.0 ) then
327:         read(IUNIT,end =270,err =280,iostat =IOS)
328:       else
329:         read(IUNIT,end =270,err =280,iostat =IOS) NUCS,
330:         &      (NUCIDF(I),I=1,MIN(NUCS,NUCMAX))
331:         if ( NUCS.gt.NUCMAX ) then
332:           write(IMG,*) 'XXX(SCFINP) number of nuclide ID's ',
333:           &      'in source file (', NUCS,
334:           &      ') exceeds program limit (nucmax=', NUCMAX,
335:           &      ').'
336:         go to 290
337:       end if
338:     end if
339: C
340: C
341: C >>> "X" , "Y", "Z" /
342: C      X,Y or Z coordinates (real*8)
343: C
344:       else if ( TAG.eq.'X' ) then
345:         read(IUNIT,end =270,err =280,iostat =IOS)
346:         &      (DSRCSP(LPX+I),I=0,NUSE-1)
347:         IWRK(3) = LPX
348:       else if ( TAG.eq.'Y' ) then
349:         read(IUNIT,end =270,err =280,iostat =IOS)
350:         &      (DSRCSP(LPY+I),I=0,NUSE-1)
351:         IWRK(4) = LPY
352:       else if ( TAG.eq.'Z' ) then
353:         read(IUNIT,end =270,err =280,iostat =IOS)
354:         &      (DSRCSP(LPZ+I),I=0,NUSE-1)
355:         IWRK(5) = LPZ
356: C
357: C >>> "INUF" /
358: C      fission nuclide #'s
359: C
360:       else if ( TAG.eq.'INUF'.and.JNUCS.ne.0 ) then
361:         read(IUNIT,end =270,err =280,iostat =IOS)
362:         &      (ISRCSP(LPINUF+I),I=0,NUSE-1)
363:         IWRK(6) = LPINUF
364: C
365: C >>> "EEEE" /
366: C      energy to induce fission
367: C
368:       else if ( TAG.eq.'EEEE'.and.JNUCS.ne.0 ) then
369:         read(IUNIT,end =270,err =280,iostat =IOS)
370:         &      (SRCSP(LPEEEE+I),I=0,NUSE-1)
371:         IWRK(7) = LPEEEE
372: C
373: C >>> "W"/  neutron weight
374: C
375:       else if ( TAG.eq.'W' ) then
376:         read(IUNIT,end =270,err =280,iostat =IOS)
377: c##<2007/03/14:PN3:
378: c## &      (SRCSP(LPW+I),I=0,NUSE-1)
379: c## &      (SRCSP(LPW+I),I=0,NUSE*2-1)
380: c##
381:       IWRK(8) = LPW
382: C
383: C >>> skip ...
384: C
385:       else
386:         read(IUNIT,end =270,err =280,iostat =IOS)
387:       end if
388: C
389:       go to 120
390: C

```

src/shared/scfinp.f

```

391: C -----
392: C
393: 130 continue
394: C
395: IWRK(1) = NUSE
396: IWRK(2) = JNUCS
397: C
398: call ATRANS( IWRK, ISRCSP(LPSIZ), NPSIZ )
399: C
400: if ( IWRK(3).lt.0.or.IWRK(4).lt.0.or.IWRK(5).lt.0 ) then
401: write(IMG,*) 'XXX(SCFINP) not all coordinates',
402: & 'are input from source file.'
403: IERR = 1
404: end if
405:
406: if ( JNUCS.ne.0.and.(IWRK(6).lt.0.or.IWRK(7).lt.0) ) then
407: write(IMG,*)
408: & 'XXX(SCFINP) fission nuclides & induced energy '
409: & 'are not in the source file.'
410: go to 290
411: else if ( JNUCS.ne.0 ) then
412: C
413: C ... change INUF those of the current problem if possible ...
414: C
415: do 140 N = 1, NUCS
416: KNUCID(N) = 0
417: 140 continue
418:
419: NN = 0
420: do 160 I = 1, NUSE
421: II = ISRCSP(LPINUF+I-1)
422: if ( II.le.0.or.II.gt.NUCS ) then
423: write(IMG,*) 'XXX(SCFINP) nuclide # in source file',
424: & ' is invalid : ', II, ' (source ', I, ' )'
425: IERR = 1
426: go to 160
427: end if
428: KNUCID(II) = KNUCID(II) + 1
429: do 150 J = 1, NUC
430: if ( NUCIDF(II).eq.NUCID(J) ) then
431: ISRCSP(LPINUF+I-1) = J
432: NN = NN + 1
433: end if
434: 150 continue
435: 160 continue
436:
437: if ( NN.lt.NUSE ) then
438: write(IMG,*) 'XXX(SCFINP) ', NUSE - NN, ' sources ',
439: & ' has no corresponding fission nuclide in this problem.'
440: IERR = 1
441: do 190 N = 1, NUCS
442: if ( KNUCID(N).gt.0 ) then
443: do 170 J = 1, NUC
444: if ( NUCIDF(N).eq.NUCID(J) ) go to 180
445: 170 continue
446: write(IMG,*) ' nuclide <', NUCIDF(N),
447: & '> is not in', ' this problem.'
448: 180 continue
449: end if
450: 190 continue
451: go to 290
452: C
453: C ... KNUCID(*) : list of converted nuclide #'s
454: C
455: else

```

```

456: NNUCID = 0
457: do 220 II = 1, NUCS
458: if ( KNUCID(II).gt.0 ) then
459: NNUCID = NNUCID + 1
460: do 200 J = 1, NUC
461: if ( NUCIDF(II).eq.NUCID(J) ) go to 210
462: 200 continue
463: 210 KNUCID(NNUCID) = J
464: end if
465: 220 continue
466: C
467: C ...
468: C
469: call PACKND( SRCSP, 'NNUCID', 'I4', NNUCID, 1, IRET )
470: call PACKND( SRCSP, 'KNUCID', 'I4', KNUCID, NNUCID, IRET )
471: &
472: if ( IRET.ne.0 ) then
473: write(IMG,*) 'XXX(SCFINP) Failed to store nuclide ID',
474: & ' list information (NNUCID,KNUCID) from file.'
475: & go to 290
476: end if
477: end if
478: C
479: end if
480: C
481: C ... check weight data is exist or not
482: C
483: if ( IWRK(8).lt.0 ) then
484: write(IMG,7160)
485: 7160 format(1X,' !!! This source file does not include',
486: & ' neutron weight data..'/1X,
487: & ' Weight is sampled as 1.0 .')
488: call CNTERR( 'WARNING' )
489: end if
490: C
491: C ... close I/O unit if it is not IOSI
492: C
493: if ( IUNIT.ne.IOSI ) then
494: inquire(IUNIT,opened=OPD)
495: if ( OPD ) close( IUNIT )
496: end if
497: C
498: if ( IERR.ne.0 ) go to 290
499: C
500: C/#IF PARA(PVM MPI)
501: C
502: C ... send data to other task (IDTASK.eq.1)
503: C
504: if ( NTASK.gt.1 ) then
505: ITO = 1
506: call MVPCOM_BCAST_I4( ITO, 11, IERR, 1, INFO )
507: C
508: C ... find label
509: C
510: call PCTLBR( SRCSP, 'LABEL FISSION', '-FISSION-FILE-', 0,
511: & IRET )
512: C
513: if ( IRET.ne.0 ) then
514: write(IMG,*) 'XXX(SCFINP) cannot find "-FISSION-FILE-"',
515: & ' label!!! Program error??'
516: stop 666
517: end if
518: C
519: IIP = 0
520: C

```

src/shared/scfinp.f

```

521: 230      call PCTCHK( SRCSP, 'scfinp', PTYPE, LLPT, NDATA, IRET )
522: C
523:      if ( JDEBG(1).ne.0 ) then
524:          write(IPR,*) ' -- send packet <', PTYPE, '> ndata ',
525:      &          NDATA
526:      end if
527: C
528:      call MVPCOM_BCAST_CH( IT0, 1112+IIP, PTYPE, LEN(PTYPE), INFO
529:      &          )
530:      call MVPCOM_BCAST_I4( IT0, 1112+IIP, NDATA, 1, INFO )
531: C
532:      if ( PTYPE.ne.' ' ) then
533: C
534:          if ( PTYPE.eq.'LB' ) then
535:
536:              call UNPKPT( SRCSP, 'LB', 'LB', LPT, NDATA, IRET )
537:              call MVCTOI( NNLEN, CWRK, SRCSP(LPT) )
538:              call MVPCOM_BCAST_CH( IT0, 13+IIP, CWRK, NDATA, INFO )
539:
540:          else if ( PTYPE.eq.'CH' ) then
541:
542:              call UNPKCS( SRCSP, 'CH', CWRK, NDATA, IRET )
543:              call MVPCOM_BCAST_CH( IT0, 13+IIP, CWRK, NDATA, INFO )
544:
545:          else if ( PTYPE.eq.'I4' ) then
546:
547:              call UNPKPT( SRCSP, 'I4', 'I4', LPT, NDATA, IRET )
548:              call MVPCOM_BCAST_I4( IT0, 13+IIP, ISRCSP(LPT), NDATA,
549:      &          INFO )
550:
551:          else if ( PTYPE.eq.'R4' ) then
552:
553:              call UNPKPT( SRCSP, 'R4', 'R4', LPT, NDATA, IRET )
554:              call MVPCOM_BCAST_R4( IT0, 13+IIP, SRCSP(LPT), NDATA,
555:      &          INFO )
556:
557:          else if ( PTYPE.eq.'R8' ) then
558:
559:              call UNPKPT( SRCSP, 'R8', 'R8', LPT, NDATA, IRET )
560:              call MVPCOM_BCAST_R8( IT0, 13+IIP, DSRCS(LPT), NDATA,
561:      &          INFO )
562:          end if
563: C
564:          IIP      = IIP + 1
565:          go to 230
566:      end if
567:  end if
568: C
569: C ... IDTASK > 1 ..... receive data from task 1
570: C
571:      else if ( IDTASK.gt.1 ) then
572: C
573:          IT0      = 1
574:          call MVPCOM_BCAST_I4( IT0, 11, IERR, 1, INFO )
575:          if ( IERR.ne.0 ) go to 290
576: C
577: C ... put label
578: C
579:          call PACKLB( SRCSP, 'LABEL FISSION', '-FISSION-FILE-', IRET )
580: C
581:          IIP      = 0
582: 240      call MVPCOM_BCAST_CH( IT0, 1112+IIP, PTYPE, LEN(PTYPE), INFO )
583:          call MVPCOM_BCAST_I4( IT0, 1112+IIP, NDATA, 1, INFO )
584: C
585:          if ( JDEBG(1).ne.0 ) then

```

```

586:          write(IPR,*) ' -- recv packet <', PTYPE, '> ndata ', NDATA
587:      end if
588: C
589:      if ( PTYPE.ne.' ' ) then
590: C
591:          if ( PTYPE.eq.'LB' ) then
592:
593:              call MVPCOM_BCAST_CH( IT0, 13+IIP, CWRK, NDATA, INFO )
594:              call PACKLB( SRCSP, 'LB', CWRK(1:NDATA), IRET )
595:
596:          else if ( PTYPE.eq.'CH' ) then
597:              call MVPCOM_BCAST_CH( IT0, 13+IIP, CWRK, NDATA, INFO )
598:              call PACKCS( SRCSP, 'CH', CWRK(1:NDATA), IRET )
599:          else if ( PTYPE.eq.'I4' ) then
600:              call PACKPT( SRCSP, 'I4', 'I4', LPT, NDATA, IRET )
601:              call MVPCOM_BCAST_I4( IT0, 13+IIP, ISRCSP(LPT), NDATA,
602:      &          INFO )
603:          else if ( PTYPE.eq.'R4' ) then
604:              call PACKPT( SRCSP, 'R4', 'R4', LPT, NDATA, IRET )
605:              call MVPCOM_BCAST_R4( IT0, 13+IIP, SRCSP(LPT), NDATA,
606:      &          INFO )
607:          else if ( PTYPE.eq.'R8' ) then
608:              call PACKPT( SRCSP, 'R8', 'R8', LPT, NDATA, IRET )
609:              call MVPCOM_BCAST_R8( IT0, 13+IIP, DSRCS(LPT), NDATA,
610:      &          INFO )
611:          end if
612: C
613:          IIP      = IIP + 1
614:          go to 240
615:      end if
616: C
617:      end if
618: C
619: C/#ENDIF
620: Cc
621: C
622:      return
623: Cc
624: C -----
625: Cc
626: 250 write(IMG,*) 'XXX unexpected end of source file before record ',
627:      &          IREC
628:      go to 290
629: C
630: 260 write(IMG,*) 'XXX I/O error in source file at record ', IREC
631:      write(IMG,*) '      error code : ', IOS
632:      go to 290
633: C
634: 270 write(IMG,*) 'XXX unexpected end of source after tag record <',
635:      &          TAG, '>'
636:      go to 290
637: C
638: 280 write(IMG,*) 'XXX I/O error in source file after tag record <',
639:      &          TAG, '>'
640:      write(IMG,*) '      error code : ', IOS
641:      go to 290
642: C
643: 290 call CNTERR( 'FATAL' )
644:      IERR      = 1
645: C
646:      if ( IUNIT.ne.IOSI ) then
647:          inquire(IUNIT,opened=OPD)
648:          if ( OPD ) close( IUNIT )
649:      end if
650: C

```

src/shared/scfinp.f

```
651: C/#IF PARA(PVM MPI)
652:   if ( NTASK.gt.1.and.IDTASK.eq.1 ) then
653:     IT0      = 1
654:     call MVPCOM_BCAST_I4( IT0, 11, IERR, 1, INFO )
655:   end if
656: C
657: C/#ENDIF
658:   return
659:   end
```

SAFE

src/shared/scfout.f

```

1:      subroutine SCFOUT( PROG,  TITLE, NUSE,  NFBNK0, NEST,
2:      &                  NREG,  WGTf,  NZONE, KZREG, JDEBG,JTLT,
3:      &                  XXXF,  YYF,  ZZZF,  EEf,  IZZF, INUF,
4:      &                  IBRGf, IBSPf,
5:      &                  NUCID, NUC,   IERR )
6: C=====
7: C purpose: output particle source data on I/O unit IOSO
8: C
9: C      Currently, only input from fission source file is possible.
10: C
11: C called from: ACTSGL
12: C calls :
13: C=====
14: C
15: C arguments ( o=output, i=input, io=output/input, w=work, c=constant)
16: C
17: C i prog : program name ('MVP', 'GMVP' etc.)
18: C i TITLE: problem title (72*2 characters)
19: C i NUSE : output upto NUSE sources in the file if
20: C << fission neutron bank >>
21: C i XXXF,YYF,ZZZF : position of sources
22: C i IZZF : zone #
23: C i INUF,EEf : colliding nuclide # and energy (for MVP)
24: C i IREGf,EEf : colliding nuclide # and energy (for MVP)
25: C i IBRGf : region #
26: C i IBSPf : space # (unused)
27: C
28: C i NUCID(NUC) : nucide ID ( prog='MVP' )
29: C i NUC : nucide of nuclide in problem ( prog='MVP' )
30: C
31: C o IERR : returns nonzero if error occurred.
32: C
33: C=====
34: C
35: C Structure of source file:
36: C
37: C Data are stored in a binary file:
38: C
39: C * record 1 : file header (32 character) & number of header records
40: C
41: C '%MVP/GMVP SOURCE FILE%', NHREC
42: C
43: C * record 2 : file version (32 character)
44: C
45: C 'Version 1.0' etc.
46: C
47: C * record 3 : Problem title of calculation which output this file
48: C (72*2 characters)
49: C
50: C * record 4 : date time code etc. (64 character)
51: C
52: C * record 5-NHREC : dummy records including at least 32 characters
53: C for future use if NHREC > 4.
54: C
55: C Order of records after record NHREC is not fixed.
56: C The following combination of records are placed.
57: C
58: C * record including data name etc. (32 character)
59: C * record(s) including data
60: C=====
61: C include 'INC/_IOUNIT'
62: C
63: C character*(*) PROG
64: C character*72 TITLE(2)
65: C

```

```

66:      integer JDEBG(*)
67: C
68: C##<2007/03/14:PN3:
69: C## real WGTf(NREG)
70: C real WGTf(NREG,2)
71: C##>
72:      integer KZREG(NZONE)
73: C
74:      real*8 XXXF(NFBNK0), YYF(NFBNK0), ZZZF(NFBNK0)
75:      real EEf(NFBNK0)
76:      integer IZZF(NFBNK0),INUF(NFBNK0)
77:      integer IBRGf(NFBNK0), IBSPf(NFBNK0,NEST)
78: C
79:      character*16 NUCID(NUC)
80: C
81:      integer IERR
82: C
83: C ..... local data ....
84: C
85:      character*256 CWRK
86:      character*12 DT, TM
87:      integer IWRK(32)
88: C
89:      parameter( LHEADS = 32 )
90:      character*(LHEADS) HEADER
91:      character*(LHEADS) TAG
92: C
93: C-----
94: C
95:      IERR = 0
96: C
97:      call LABEL( IOW, 'particle source file output' )
98: C
99:      call RWIND( IOSO )
100: C
101: C >>> Record 1 :
102: C
103:      NHREC = 4
104: C
105:      IREC = 1
106:      if ( PROG.eq.'MVP' ) then
107:          HEADER = '%MVP SOURCE FILE%'
108:      else
109:          HEADER = '%MVP SOURCE FILE%'
110:      end if
111: C
112: C
113:      write(IOSO) HEADER, NHREC
114: C
115: C >>> Record 2 :
116: C
117:      IREC = IREC + 1
118:      HEADER = 'source file version 1.0'
119:      write(IOSO) HEADER
120: C
121:      write(IOW,7000) HEADER
122:      7000 format(1X,' version: <','A','>')
123: C
124: C >>> Record 3 :
125: C
126:      IREC = IREC + 1
127:      write(IOSO) (TITLE(I),I=1,2)
128: C
129: C >>> Record 4 :
130: C

```

src/shared/scfout.f

```
131:      IREC      = IREC + 1
132:      call HIZUKE( DT )
133:      call JIKAN( TM(1:8) )
134:      CWRK(:64)  = 'DATE: '//DT//' TIME: '//TM(1:8)
135:      write(IOSO) CWRK(:64)
136: C
137:      write(IOW,7020) CWRK(:64)
138:      7020 format(1X,'      date time etc.: <'A,'>')
139: C
140: C=====
141: C
142: C >>> "SIZES" /
143: C      ns : number of sources in this file
144: C
145:      TAG      = 'SIZES'
146:      write(IOSO) TAG
147:      write(IOSO) NUSE
148: C
149:      write(IOW,7040) NUSE
150:      7040 format(1X,'      number of sources : ',I8)
151: C
152: C >>> "X" , "Y" , "Z" /
153: C      X,Y or Z coordinates (real*8)
154: C
155:      TAG      = 'X'
156:      write(IOSO) TAG
157:      write(IOSO) (XXXF(I),I=1,NUSE)
158: C
159:      TAG      = 'Y'
160:      write(IOSO) TAG
161:      write(IOSO) (YYF(I),I=1,NUSE)
162: C
163:      TAG      = 'Z'
164:      write(IOSO) TAG
165:      write(IOSO) (ZZF(I),I=1,NUSE)
166: C
167:      if( JDEBG(1).ne.0 ) then
168:      write(IOW,1000) (XXXF(i),YYF(i),ZZF(i),i=1,NUSE)
169:      1000 format(1X,' == coordinates '//(1X,3E14.5))
170:      end if
171: C
172: C >>> "W" : fission neutron weight
173: C
174:      TAG      = 'W'
175:      write(IOSO) TAG
176:      if( JTLLT.eq.0 ) then
177: c##<2007/03/14:PN3:
178: c##      write(IOSO) (WGTF(KZREG(IZZF(I))),I=1,NUSE)
179: c##      write(IOSO) ((WGTF(KZREG(IZZF(I))),J),I=1,NUSE),J=1,2)
180: c##>
181:      else
182: c##<2007/03/14:PN3:
183: c##      write(IOSO) (WGTF(IBRGF(I)),I=1,NUSE)
184: c##      write(IOSO) ((WGTF(IBRGF(I),J),I=1,NUSE),J=1,2)
185: c##>
186:      end if
187: C
188:      if ( PROG.eq.'MVP' ) then
189: C
190: C >>> "NUCLIDE ID" /
191: C      nuc : number of nuclides whose ID is stored in this file
192: C      nucid(:nuc) : nuclide ID's (character*16)
193: C
194:      TAG      = 'NUCLIDE ID'
195:      write(IOSO) TAG
196:      write(IOSO) NUC, (NUCID(I),I=1,NUC)
197: C
198: C >>> "INUF" /
199: C      fission nuclide #'s
200: C
201:      TAG      = 'INUF'
202:      write(IOSO) TAG
203:      write(IOSO) (INUF(I),I=1,NUSE)
204: C
205: C >>> "EEEE" /
206: C      energy to induce fission
207: C
208:      TAG      = 'EEEE'
209:      write(IOSO) TAG
210:      write(IOSO) (EEEE(I),I=1,NUSE)
211: C
212:      end if
213: C
214:      call RWIND( IOSO )
215: C
216:      return
217:      end
```

src/shared/seqstr.f

```
1: C=<MVP-GMVP>=====
2: C Functions to check occurrence of specified string pattern sequence
3: C separated by more than one blanks.
4: C
5: C   isqstN( string, pattern1, pattern2, ..., patternN )
6: C
7: C   Returns 1 if "string" includes blank separated pattern1 ...
8: C   patternN in this order, else returns 0.
9: C   Header blanks and strings after patternN are ignored.
10: C
11: C   N = "n" : patterns are in a character string array and starting
12: C             positions are given as integer array.
13: C   N = 2 : two patterns are given as separate arguments.
14: C   N = 3 : three patterns are given as separate arguments.
15: C   N = 4 : four patterns are given as separate arguments.
16: C   .
17: C
18: C   Only "isqst2" is available ( Aug 1995 ).
19: C
20: C=====
21: C
22: C ----- number of patterns = 1 -----
23: C
24: C   function ISQST1(STR,PAT1)
25: C   character*(*) STR, PAT1
26: C   integer ISQST1
27: C
28: C   ISQST1 = 0
29: C   NL     = LEN(STR)
30: C   IP     = 1
31: C
32: C   ... repeat this sequence for each pattern
33: C
34: C   call NOBLNK( STR, IP, IP2, NL )
35: C   if ( IP.gt.NL ) return
36: C   if ( IMATCH(PAT1,STR(IP:IP2)).eq.0 ) return
37: C
38: C   ISQST1 = 1
39: C   return
40: C   end
41: C
42: C ----- number of patterns = 2 -----
43: C
44: C   function ISQST2(STR,PAT1,PAT2)
45: C   character*(*) STR, PAT1, PAT2
46: C   integer ISQST2
47: C
48: C   ISQST2 = 0
49: C   NL     = LEN(STR)
50: C   IP     = 1
51: C
52: C   ... repeat this sequence for each pattern
53: C
54: C   call NOBLNK( STR, IP, IP2, NL )
55: C   if ( IP.gt.NL ) return
56: C   if ( IMATCH(PAT1,STR(IP:IP2)).eq.0 ) return
57: C   IP     = IP2 + 1
58: C
59: C   call NOBLNK( STR, IP, IP2, NL )
60: C   if ( IP.gt.NL ) return
61: C   if ( IMATCH(PAT2,STR(IP:IP2)).eq.0 ) return
62: C
63: C   ISQST2 = 1
64: C   return
65: C   end
```

```
66: C
67: C ----- number of patterns = 3 -----
68: C
69: C   function ISQST3(STR,PAT1,PAT2,PAT3)
70: C   character*(*) STR, PAT1, PAT2, PAT3
71: C   integer ISQST3
72: C
73: C   ISQST3 = 0
74: C   NL     = LEN(STR)
75: C   IP     = 1
76: C
77: C   ... repeat this sequence for each pattern
78: C
79: C   call NOBLNK( STR, IP, IP2, NL )
80: C   if ( IP.gt.NL ) return
81: C   if ( IMATCH(PAT1,STR(IP:IP2)).eq.0 ) return
82: C   IP     = IP2 + 1
83: C
84: C   call NOBLNK( STR, IP, IP2, NL )
85: C   if ( IP.gt.NL ) return
86: C   if ( IMATCH(PAT2,STR(IP:IP2)).eq.0 ) return
87: C   IP     = IP2 + 1
88: C
89: C   call NOBLNK( STR, IP, IP2, NL )
90: C   if ( IP.gt.NL ) return
91: C   if ( IMATCH(PAT3,STR(IP:IP2)).eq.0 ) return
92: C   IP     = IP2 + 1
93: C
94: C
95: C   ISQST3 = 1
96: C   return
97: C   end
```


src/shared/sinklt.f

```

1:      subroutine SINKLT( NPT,  MLT,  M,  IZON,  JHLAT, X,  Y,
2:      &                  Z,  AI,  BI,  CI,  IWK,  IZZ,  XXX,
3:      &                  YYY,  ZZZ,  AAA,  BBB,  CCC,  IPC,  LEVL,
4:      &                  LZZ,  LPOS,  LCRS,  NBANK,  DALT,  LTYP,
5:      &                  NVLAT,  SZLAT,  IPLAT,  KLATT,  KSLAT,  KHLAT,
6:      &                  IPCEL,  CELSZ,  DEPS )
7: C=====
8: C  PURPOSE: ENETER LATTICE REGION.
9: C  CALLED IN;          CALLS: (NONE)
10: C=====
11: C == INPUT PARAMETERS ==
12: C
13: C  NPT : NUMBER OF PARTICLES (Present version allows only 1 particle.)
14: C  MLT : LATTICE #,  M : POINTER TO DALT
15: C  IZON : ZONE # IN UPPER LEVEL
16: C  X,Y,Z, AI,BI,CI : COORDINATES & DIRECTION IN UPPER LEVEL
17: C  INN : NUMBER OF PARTICLES IN THE SEARCH STACK.
18: C
19: C == OUTPUT ==
20: C
21: C  IPC : NEW ZONE # ( RETURN )
22: C
23: C  === INTER-STACK DATA FLOW ===
24: C (NONE)
25: C
26: C  === BANK DATA TO BE UPDATED ===
27: C
28: C  (XXX,YYY,ZZZ),(AAA,BBB,CCC),IZZ : POSITION,DIRECTION & ZONE #
29: C  LEVL,LZZ,LPOS,LCRS          : LATTICE PARAMETER BANK
30: C
31: C=====
32:      implicit real*8(D)
33:      include 'INC/_IOUNIT'
34: C
35: C .... PARTICLE BANK .....
36: C
37:      real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK), AAA(NBANK), BBB(NBANK),
38:      &      CCC(NBANK)
39:      integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,*), LPOS(NBANK,*),
40:      &      LCRS(NBANK,*)
41: C
42: C .... GEOMETRY ARRAY DATA .....
43: C
44:      integer IPLAT(*), KLATT(*), KSLAT(*), NVLAT(4,*), IPCEL(*),
45:      &      LTYP(*), KHLAT(*)
46:      real*8 CELSZ(6,*), SZLAT(8,*), DALT(*)
47: C
48: C .... WORKING ARRAYS .....
49: C
50: C  KEEP IWK(I) UNCHANGED, RETURN IPC(I) AS NEW ZONE #
51: C
52:      integer IWK(NBANK)
53:      real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
54:      &      CI(NBANK)
55: C
56:      parameter( DROOT3 = 1.73205080756887719D0, DHRT3 = DROOT3*0.5D0
57:      &      )
58: C
59: C .... ROTATION VECTOR: ROT(N,1) = COS(PI/3 *N)
60: C      ROT(N,2) = SIN(PI/3 *N)
61: C      common /HEXROT/ DROT
62: C      real*8 DROT(0:5,2)
63: C
64: C
65: C -----

```

```

66: C      RECTANGULAR LATTICE (LTYP(MLT) = 1)
67: C -----
68: C
69:      if ( LTYP(MLT).eq.1 ) then
70: *VOCL LOOP,NOVREC
71:      do 100 I = 1, NPT
72: C
73: C ..... TRANSFORMATION TO IN-LATTICE COORDINATES & DIRECTION .....
74: C
75: C
76:      DX = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*Z(I)
77:      &      - DALT(M+9)
78:      DY = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*Z(I)
79:      &      - DALT(M+10)
80:      DZ = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*Z(I)
81:      &      - DALT(M+11)
82:      DA = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*CI(I)
83:      DB = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) + DALT(M+5)*CI(I)
84:      DC = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) + DALT(M+8)*CI(I)
85: C
86: C ....IP: POSITION IN THE LATTICE .....
87:      DXX = DX + DA*DEPS
88:      DYY = DY + DB*DEPS
89:      DZZ = DZ + DC*DEPS
90:      IX = MAX(0,MIN(NVLAT(1,MLT)-1,INT(DXX/SZLAT(1,MLT))))
91:      IY = MAX(0,MIN(NVLAT(2,MLT)-1,INT(DYY/SZLAT(2,MLT))))
92:      IZ = MAX(0,MIN(NVLAT(3,MLT)-1,INT(DZZ/SZLAT(3,MLT))))
93:      IP = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
94: C
95: C ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
96: C      IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
97: C
98:      ICEL = KLATT(IPLAT(MLT)+IP)
99:      ISS = KSLAT(IPLAT(MLT)+IP)
100:      IDRX = ISS/100
101:      IDRY = (ISS-IDRX*100) /10
102:      IDRZ = MOD(ISS,10)
103:      JSX = 1 - 2*IDRX
104:      JSY = 1 - 2*IDRY
105:      JSZ = 1 - 2*IDRZ
106: *VOCL STMT,IF(100)
107:      if ( ICEL.ne.0 ) then
108: C
109: C ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL
110: C
111:      IPC = IPCEL(ICEL)
112:      IZZ(IWK(I)) = IPC
113:      LEVL(IWK(I)) = LEVL(IWK(I)) + 1
114:      LZZ(IWK(I),LEVL(IWK(I))) = IZON
115:      LPOS(IWK(I),LEVL(IWK(I))) = IP
116:      if ( JHLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
117: C
118: C ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
119: C
120:      XXX(IWK(I)) = CELSZ(4,ICEL)
121:      &      + JSX*(DX-(IX+0.5)*SZLAT(1,MLT))
122:      YYY(IWK(I)) = CELSZ(5,ICEL)
123:      &      + JSY*(DY-(IY+0.5)*SZLAT(2,MLT))
124:      ZZZ(IWK(I)) = CELSZ(6,ICEL)
125:      &      + JSY*(DZ-(IZ+0.5)*SZLAT(3,MLT))
126:      AAA(IWK(I)) = JSX*DA
127:      BBB(IWK(I)) = JSY*DB
128:      CCC(IWK(I)) = JSZ*DC
129: C
130: C ..... CELL # = 0 : OUT OF LATTICE DEFINITION

```

src/shared/sinklt.f

```

131: C
132:       else
133:         IPC      = IZON
134:       end if
135: 100    continue
136: C
137: C ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
138: C       LSSRC(I + INN) = IWK(I)
139: C       IZNXT(I + INN) = IPC
140: C       NNXT(IPC)      = NNXT(IPC) + 1
141: C
142: C -----
143: C       HEXAGONAL LATTICE (LTYP(MLT) = 2,3,4 )
144: C -----
145: C
146:       else if ( LTYP(MLT).ge.2.and.LTYP(MLT).le.4 ) then
147: *VOCL LOOP,NOVREC
148:       do 200 I = 1, NPT
149: C
150: C ..... TRANSFORMATION TO IN-LATTICE COORDINATES & DIRECTION .....
151: C
152: C
153:       DX = DALT(M)*X(I) + DALT(M+1)*Y(I) + DALT(M+2)*Z(I)
154: &       - DALT(M+9)
155:       DY = DALT(M+3)*X(I) + DALT(M+4)*Y(I) + DALT(M+5)*Z(I)
156: &       - DALT(M+10)
157:       DZ = DALT(M+6)*X(I) + DALT(M+7)*Y(I) + DALT(M+8)*Z(I)
158: &       - DALT(M+11)
159:       DA = DALT(M)*AI(I) + DALT(M+1)*BI(I) + DALT(M+2)*CI(I)
160:       DB = DALT(M+3)*AI(I) + DALT(M+4)*BI(I) + DALT(M+5)*CI(I)
161:       DC = DALT(M+6)*AI(I) + DALT(M+7)*BI(I) + DALT(M+8)*CI(I)
162: C
163: C ..... CELL-POSITION ....
164: C
165:       DX = DX + DA*DEPS
166:       DY = DY + DB*DEPS
167:       DZ = DZ + DC*DEPS
168:       N2 = 2*NVLAT(1,MLT)
169:       IN1 = INT(2.0*DX/SZLAT(1,MLT)+1.0+N2) - N2
170:       IN2 = INT((DX+DY*DROOT3)/SZLAT(1,MLT)+1.0+N2) - N2
171:       IN3 = INT((-DX+DY*DROOT3)/SZLAT(1,MLT)+1.0+N2) - N2
172:       IX = (IN1-IN3+1) / 3
173:       IY = (IN2-IN3) / 3
174:       IZ = MAX(0,MIN(NVLAT(3,MLT)-1,INT(DZZ/SZLAT(3,MLT))))
175:       IP = IX + NVLAT(1,MLT)*(IY+NVLAT(2,MLT)*IZ)
176: C
177: C ..... CELL # (KLATT) & DIRECTION INDEXES (KSLAT) .....
178: C       IPLAT(MLT) : STARTING POSITION OF MLT'TH LATTICE
179: C
180:       IC = IPLAT(MLT) + IP
181:       ICEL = KLATT(IC)
182:       ISS = KSLAT(IC)
183:       IDRX = ISS/100
184:       JSX = 1 - 2*IDRX
185:       IDRY = (ISS-IDRX*100) / 10
186:       JSZ = 1 - 2*MOD(ISS,10)
187: *VOCL STMT,IF(100)
188:       if ( ICEL.ne.0 ) then
189: C
190: C ..... IPCEL(ICEL) : ZONE # OF ICEL'TH LATTICE-CELL BUFFER-ZONE
191: C
192:       IPC = IPCEL(ICEL)
193:       IZZ(IWK(I)) = IPC
194:       LEVL(IWK(I)) = LEVL(IWK(I)) + 1
195:       LZZ(IWK(I),LEVL(IWK(I))) = IZON
196:
197:       LPOS(IWK(I),LEVL(IWK(I))) = IP
198:       LCRS(IWK(I),LEVL(IWK(I))) = KHLAT(IPLAT(MLT)+IP)
199: C
200: C ..... IN-CELL COORDINATES (TAKING ACCOUNT OF CELL DIRECTIONS)
201: C
202:       DX = JSX*(DX-(IX+0.5*IY)*SZLAT(1,MLT))
203:       DY = DY - IY*SZLAT(1,MLT)*DHRT3
204:       &       XXX(IWK(I)) = DROT(IDRY,1)*DX - DROT(IDRY,2)*DY
205:       &       + CELSZ(3,ICEL)
206:       &       YYY(IWK(I)) = DROT(IDRY,2)*DX + DROT(IDRY,1)*DY
207:       &       + CELSZ(4,ICEL)
208:       &       ZZZ(IWK(I)) = JSZ*(DZ-(IZ+0.5)*SZLAT(3,MLT))
209:       &       + CELSZ(5,ICEL)
210:       DA = JSX*DA
211:       AAA(IWK(I)) = DROT(IDRY,1)*DA - DROT(IDRY,2)*DB
212:       BBB(IWK(I)) = DROT(IDRY,2)*DA + DROT(IDRY,1)*DB
213:       CCC(IWK(I)) = JSZ*DC
214: C
215: C ..... CELL # = 0 : OUT OF LATTICE DEFINITION
216: C
217:       else
218:         IPC      = IZON
219:       end if
220: 200    continue
221: C
222: C ..... SEND TO THE NEXT-ZONE-SEARCH STACK .....
223: C       LSSRC(I + INN) = IWK(I)
224: C       IZNXT(I + INN) = IPC
225: C       NNXT(IPC)      = NNXT(IPC) + 1
226: C
227: C -----
228: C       STG region (LTYP(MLT) = 10)
229: C -----
230: C
231:       else if ( LTYP(MLT).eq.10 ) then
232: *VOCL LOOP,NOVREC
233:       do 300 I = 1, NPT
234:         IPC = 0
235:         IZZ(IWK(I)) = IPC
236:         LEVL(IWK(I)) = LEVL(IWK(I)) + 1
237:         LZZ(IWK(I),LEVL(IWK(I))) = IZON
238:         LPOS(IWK(I),LEVL(IWK(I))) = -1
239:         if ( JHLAT.ne.0 ) LCRS(IWK(I),LEVL(IWK(I))) = 0
240: 300    continue
241: C
242:       end if
243: C
244: CCCCC NNXT(NZONE+1) = INN + NPT
245:       return
246:       end

```

src/shared/sizchk.f

```
1:      subroutine SIZCHK( VNAME, NXNAME,NX,      IINP,  ICK )
2:  C
3:  C   JAERI MONTE CARLO CODE UTILITY
4:  C
5:  C=====
6:  C   PURPOSE: ASSIGNS VALUE 'IINP' TO AN SIZE-DATA 'NX' IF NOT-YET DEFINED
7:  C             OR ISSUE ERROR MESSAGE IF 'IINP' IS NOT EQUAL TO 'NX'.
8:  C   CALLED IN: INTRO
9:  C-----
10: C   arguments (i=input, o=output, w=work)
11: C
12: C i  VNAME: Data name of 'NX'. for message.
13: C i  NXNAME: variable name of 'NX'. for message.
14: C io NX : size variable whose value is checked or set here if necessary.
15: C i  IINP : size to be checked against NX.
16: C io ICK : counter of size unmatched error
17: C=====
18: C
19:      character VNAME*(*), NXNAME*(*)
20:  C
21:      include 'INC/_IUNIT'
22:  C
23:      if ( NX.eq.0 ) then
24:          NX      = IINP
25:  C
26:      else if ( NX.ne.IINP ) then
27:          LVN      = INDEX(VNAME,' ') - 1
28:          if ( LVN.lt.0 ) LVN = LEN(VNAME)
29:  C
30:          write(IMSG,7000) VNAME(:LVN), NXNAME, NX
31: 7000      format(1X,'XXX THE NUMBER OF DATA GIVEN AS <',A,'>',
32:      &          ' DOES NOT MATCH A PREVIOUSLY DETERMINED SIZE. '//
33:      &          ' ( ',A,' = ',I8,' ) ')
34:          call CNTERR( 'FATAL' )
35:          ICK      = ICK + 1
36:      end if
37:  C
38:      return
39:      end
```

src/shared/smpfyn.f

```

1:      subroutine SMPFYN( PROG, NS,   X,      FNAME, DSRC, NDSRC,
2:      &                  IRAND, RWK,   IWK,    NRWK, IERROR,NERROR )
3: C==<MVP/GMVP>=====
4: C purpose:  source parameter sampling ( FEYNMAN )
5: C           of time from Cf spontaneous fission source
6: C
7: C called in : SYSSRC (source generation routine.)
8: C calls:   RANU2
9: C-----
10: C
11: C arguments (i =input, o=output, w=work, c=constant )
12: C
13: CLASS arg:  meanings
14: C
15: C i prog : program name ('MVP'/'GMVP')
16: C i ns  : number of source to be generated by this source set (input)
17: C i o X(*) : variable to be sampled
18: C i fname : sampling "function" name.
19: C i dsrc : data for sampling.
20: C i ndsrc : number of elements on data array.
21: C i irand : seed of random numbers
22: C w rwk(ns,nrwk): working array mainly used to save random numbers.
23: C i nrwk : second dimension of rwk array.
24: C o ierror : error flag ( 0 = no error, 1 = error )
25: C i o nerror : number of errors
26: C
27: C =====
28:      implicit real*8(A-H,O-Q,S,Z)
29: C
30:      character*(*) PROG, FNAME
31: C
32:      real*8 X(NS), DSRC(NDSRC)
33: C ... RWK and IWK use the same area
34:      real RWK(NS,NRWK)
35:      integer IWK(NS,NRWK)
36: C
37:      real*8 PAI, DPAI, HPAI
38:      real BX(50)
39:      parameter( PAI = 3.1415926535897932D0, DPAI = 2.0D0*PAI,
40:      &          HPAI = 0.5D0*PAI )
41: C
42:      include 'INC/_IOUNIT'
43: C
44:      data BX /
45:      & 5.00000e-01,4.60170e-01,4.20740e-01,3.82090e-01,
46:      & 3.44580e-01,3.08540e-01,2.74250e-01,2.41960e-01,
47:      & 2.11860e-01,1.84060e-01,1.58660e-01,1.35670e-01,
48:      & 1.15070e-01,9.68000e-02,8.07570e-02,6.68070e-02,
49:      & 5.47990e-02,4.45650e-02,3.59300e-02,2.87170e-02,
50:      & 2.27500e-02,1.78640e-02,1.39030e-02,1.07240e-02,
51:      & 8.19750e-03,6.20970e-03,4.66120e-03,3.46700e-03,
52:      & 2.55510e-03,1.86580e-03,1.34990e-03,9.67600e-04,
53:      & 6.87140e-04,4.83420e-04,3.36930e-04,2.32630e-04,
54:      & 1.59110e-04,1.07800e-04,7.23480e-05,4.80960e-05,
55:      & 3.16170e-05,2.06580e-05,1.33460e-05,8.53990e-06,
56:      & 5.41250e-06,3.39770e-06,2.11250e-06,1.30080e-06,
57:      & 7.93330e-07,4.79180e-07 /
58: C
59:      IERROR = 0
60: C
61:      NF = DBLE(NS)/DSRC(3) + 5
62:      if ( NF.gt.NS ) NF = NS
63:      NNF = 0
64:      NNS = NS
65:      NN = 0

```

```

66: C
67:      if ( DSRC(4).gt.0.0 ) then
68: C
69: C == Select the number of coincident particles from
70: C Terrell-Gauss distribution
71: C
72: C      exp(-(x-x0+b)**2/(2*sigma**2))
73: C
74: C      x0 = dsrc(3)
75: C      sigma = dsrc(4)
76: C      b = correction factor to keep x-ave to be x0
77: C          when x>=0 is adopted.
78: C
79: C ... correction factor b
80: C
81:      XX = (DSRC(3)+0.5D0) / DSRC(4)
82:      IX = XX / 0.1D0
83:      if ( IX.lt.49 ) then
84:          XR = (XX-0.1D0*IX) / 0.1D0
85:          B = BX(IX+1) + XR*(BX(IX+2)-BX(IX+1))
86:      else
87:          B = 0.0
88:      end if
89: C
90: 100 continue
91: C
92:      call RANU2( IRAND, RWK, 2*NF, ICOD )
93: C
94:      NF0 = 0
95: *VOCL LOOP,NOVREC
96:      do 110 I = 1, NF
97: C
98: C ... sampling from Terrell-Gauss distribution
99: C
100:          XF = DSRC(3) - B + DSRC(4)*SQRT(-2.D0*LOG(RWK(I,1)))*
101:          & COS(DPAI*RWK(NF+I,1))
102: C
103:          if ( XF.ge.0.5D0 ) then
104:              NF0 = NF0 + 1
105:              IWK(NNF+NF0,3) = XF + 0.5D0
106:          end if
107: 110 continue
108:      NF = NF0
109: C
110:      NN1 = 0
111:      do 120 I = 1, NF
112:          NN1 = NN1 + IWK(I+NNF,3)
113:          if ( NN1.ge.NNS ) then
114:              NE = I
115:              NN2 = NN1
116:              go to 140
117:          end if
118: 120 continue
119:          NNF = NNF + NF
120:          NN = NN + NN1
121:          NNS = NS - NN
122:          NF = DBLE(NNS)/DSRC(3) + 1
123:          if ( NF.gt.NS-NNF ) NF = NS-NNF
124:          go to 100
125: 140 continue
126:          NNF = NNF + NE
127:          NMI = NNS - NN2
128:          IWK(NNF,3) = IWK(NNF,3) + NMI
129: C
130:      else

```

src/shared/smpfyn.f

```
131: C
132: 200    continue
133: C
134:      call RANU2( IRAND, RWK, NF, ICOD )
135: C
136:      NF0 = 0
137:      do 210 I = 1, NF
138:          IWKI = DSRC(3) + RWK(I,1)
139:          if ( IWKI.gt.0 ) then
140:              NF0 = NF0 + 1
141:              IWK(NF0+NNF,3) = IWKI
142:          end if
143: 210    continue
144:      NF = NF0
145: C
146:      NN1 = 0
147:      do 220 I = 1, NF
148:          NN1 = NN1 + IWK(I+NNF,3)
149:          if ( NN1.ge.NNS ) then
150:              NE = I
151:              NN2 = NN1
152:              go to 230
153:          end if
154: 220    continue
155:      NNF = NNF + NF
156:      NN = NN + NN1
157:      NNS = NS - NN
158:      NF = DBLE(NNS)/DSRC(3) + 1
159:      if ( NF.gt.NS-NNF ) NF = NS-NNF
160:      go to 200
161: 230    continue
162:      NNF = NNF + NE
163:      NMI = NNS - NN2
164:      IWK(NNF,3) = IWK(NNF,3) + NMI
165:  end if
166: C
167:      NF = NNF
168:      NN1 = 0
169:      NMAX = 0
170: *VOCL LOOP,NOVREC
171:      do 300 I = 1 , NF
172:          NN1 = NN1 + IWK(I,3)
173:          if ( IWK(I,3).gt.NMAX ) NMAX = IWK(I,3)
174: 300    continue
175: C
176:      if ( NN1.ne.NS ) then
177:          write(IPR,*) ' XXX SMPFYN, selected no of particles(',
178:      &   nn1,') is different from required(',ns,')'
179:          stop
180:      end if
181: C
182: C ... sampling from uniform didtribution ...
183: C
184:      call RANU2( IRAND, RWK, NF, ICOD )
185: C
186:      D1      = DSRC(2) - DSRC(1)
187: *VOCL LOOP,NOVREC
188:      do 310 I = 1, NF
189:          RWK(I,2) = DSRC(1) + D1*RWK(I,1)
190: 310    continue
191: C
192:      NN = 0
193:      do 320 j = 1, NMAX
194: *VOCL LOOP,NOVREC
195:          do 330 I = 1, NF
196:              if ( IWK(I,3).ge.J ) then
197:                  NN = NN + 1
198:                  IWK(NN,4) = I
199:              end if
200: 330          continue
201: 320      continue
202: C
203:          do 340 I = 1, NS
204:              X(I) = RWK(IWK(I,4),2)
205: 340          continue
206: C
207:          return
208:      end
```

src/shared/smpinf.f

```

1:      subroutine SMPINF( SRCSP, PROG,JUNDG,JFISS,JDEBG, NSOUR, SVECNM,
2:      &                   JRVEC, IVREF, MXVEC, NVECS, MVSTK, VNAME, TC,
3:      &                   NERR, IDMAT, NMAT, NUCID, NUC,  NTGX,
4:      &                   LFKAI, FKAI, IBSDA, NBODY, IPSDA, IZNAM,
5:      &                   NINPZ, KINPZ, NZONE, CBUFF, IVLF,  IVRG,  IWRK,
6:      &                   SWRK,  DWRK,  NWORK )
7: C==<MVP/GMVP>=====
8: C purpose: input source sampling information in $SOURCE data block
9: C
10: C      *      ( * : current file pointer in input file )
11: C      @vname = .... ;
12: C
13: C      *
14: C      @(vector-list) = .... ;
15: C
16: C      or
17: C
18: C      * *
19: C      WHEN<when#> [=] .....
20: C
21: C called from: SRCINF
22: C calls :
23: C      CATSTR CCOMP CDENTK CHREAD CKVAL4 CNTERR DENTAK DMREAD GETSTR
24: C      I4READ INTPLC PACKCS PACKND R4READ R8READ TBLXX WALIAS CKODR4
25: C=====
26: C
27: C arguments ( o=output, i=input, io=output/input, w=work, c=constant)
28: C
29: C io srcsp : data packet container for source information
30: C i prog : program name ('MVP','GMVP' etc.)
31: C o jundg : flag to use "underground" modules to process rejection by
32: C      zone or body.
33: C i jfiss : fission neutron treatment is possible if non zero.
34: C i nsour : current source set #
35: C
36: C io svecnm : names of vector variables appereed in current source set
37: C io jrvec : vector connection flags.
38: C      (jrvec(i,i) : sampling count of vector i )
39: C io ivref : vector reffrence count.
40: C c mxvec : allowable maximum of number of vector variables
41: C io nvecs : number of vector variables appeared in current source set
42: C
43: C io mvstk : max of stack depth for vector calculation (for overall
44: C      sampling information)
45: C
46: C i vname : vector variable name whose sampling information is
47: C      input in this routine, or '@' only for @(v1 v2 ...) = ...
48: C i tc : termination character in 'vname' input ('=' or '(')
49: C
50: C io nerr : total number of input data error
51: C
52: C i idmat(nmat) : material ID
53: C i nmat : number of material in problem ( prog='GMVP' )
54: C i nucid(nuc) : nucide ID ( prog='MVP' )
55: C i nuc : nucide of nuclide in problem ( prog='MVP' )
56: C
57: C i ntgx : number of energy groups in cross section library
58: C i fkai(ntgx,nmat) : fission spectrum data (prog='GMVP')
59: C
60: C i ibsda(3,nbody) : first surface #'s & ID's of each body
61: C i nbody : number of bodies
62: C i iznam(ninpz) : names of input zones
63: C i ninpz : number of input zones
64: C i kinpz(nzone) : input zone # of each zone
65: C i nzone : number of each zones

```

```

66: C
67: C
68: C w cbuff : character working buffer.
69: C w ivlf : to save vector # appeared on lefthand side.
70: C w ivrg : to save vector # appeared on righthand side.
71: C w iwrk(nwork),swrk(nwork):
72: C      working area for input & preprocessing (start on the
73: C      same address)
74: C w dwrk(nwork) : double word working area
75: C
76: C-----
77: C
78:      integer SRCSP(*)
79: C
80:      integer JDEBG(*)
81:      character*(*) PROG
82:      character*(*) VNAME
83:      character*1 TC
84: C
85:      character*16 SVECNM(MXVEC)
86:      integer JRVEC(MXVEC,MXVEC)
87:      integer IVREF(MXVEC)
88: C
89:      integer IDMAT(NMAT)
90:      character*16 NUCID(NUC)
91: C
92:      real FKAI(NTGX,NMAT)
93: C
94:      integer IBSDA(3,NBODY)
95:      integer IPSDA(3,*)
96:      character*12 IZNAM(NINPZ)
97:      integer KINPZ(NZONE)
98: C
99:      character*(*) CBUFF
100:      integer IVLF(MXVEC)
101:      integer IVRG(MXVEC)
102: C
103: C ... iwrk & swrk has same starting address !!
104: C      (dwrk is not)
105: C
106:      integer IWRK(NWORK)
107:      real SWRK(NWORK)
108:      real*8 DWRK(NWORK)
109: C
110: C === external function ===
111: C
112:      real*8 DENTAK
113: C
114: Cccc real*8      cdentk
115: C
116: C ==== local data ====
117: C
118:      real*8 DTEMP, DTEMP1, DTEMP2
119:      character*256 CWRK
120:      character*256 CWRK2
121:      character*8 TERM
122:      character*1 TCS
123:      character*1 XYZ
124:      character*16 INTSYM
125:      character*16 FORMT
126:      character*256 SFILE
127: C
128: C ==== constants ====
129: C
130:      character*26 ALP

```

src/shared/smpinf.f

```

131: C
132:     real*8 PAI
133:     parameter( PAI = 3.1415926535897932D0 )
134: C
135:     include 'INC/_IOUNIT'
136: C
137: C ... data statement
138: C
139:     data ALP /'ABCDEFGHIJKLMNPOQRSTUVWXYZ'/
140: C
141: C -----
142: C execution starts after here -----
143: C -----
144: C
145: 7000 format(1X,' SAMPLE >> ',A/(1X,16X,A*))
146: C
147: 7020 format(1X,'XXX <SAMPLING INFORMATION ERROR> ',A,' <',A,'>')
148: 7040 format(1X,'XXX <SAMPLING INFORMATION ERROR> ',A/(20X,A))
149: 7060 format(1X,'XXX <SAMPLING INFORMATION ERROR> ',A/(20X,A,A))
150: 7080 format(1X,'XXX <SAMPLING INFORMATION ERROR> Nuclide-id<',A,
151: & '> is not found in nuclide list.',10X,
152: & '<NUCLIDES AVAILABLE>',15/(10X,5(:A,2X)))
153: C
154:     CBUFF = ' '
155:     LCBUF = 0
156: C
157: C===== @[vector] = ...
158: C
159: C
160: C NSVECT : number of sampled variables (on lefthand side of '=')
161: C NRVECT : number of referenced variables (on righthand side of '=')
162: C
163:     NSVECT = 0
164:     NRVECT = 0
165: C
166:     if ( TC.eq.'=' .or. VNAME(1:4).eq.'WHEN' ) then
167:         NSVECT = NSVECT + 1
168:         LLV = INDEX(VNAME,' ') - 1
169:         if ( LLV.lt.0 ) LLV = LEN(VNAME)
170:         if ( VNAME(1:1).ne.'@' ) then
171:             LV = 1
172:         else
173:             LV = 2
174:         end if
175: C
176:         if ( INDEX(ALP,VNAME(LV:LV)).eq.0 ) then
177:             write(MSG,7020) 'Invalid variable name', VNAME(LV:LLV)
178:             NERR = NERR + 1
179:         end if
180: C
181:         NLEN = LLV - LV + 1
182:         if ( NLEN.gt.LEN(SVECNM(1)) ) then
183:             write(MSG,7020) 'Too long variable name', VNAME(LV:LLV)
184:             NERR = NERR + 1
185:             NLEN = LEN(SVECNM(1))
186:         end if
187: C
188:         call CATSTR( CBUFF, LCBUF, '@', IERR )
189:         call CATSTR( CBUFF, LCBUF, VNAME(LV:LLV), IERR )
190:         call CATSTR( CBUFF, LCBUF, ' = ', IERR )
191:         if ( IERR.ne.0 ) go to 530
192: C
193:         do 100 IVEC = 1, NVECS
194:             if ( SVECNM(IVEC).eq.VNAME(LV:LLV) ) go to 110
195: 100     continue

```

```

196: C
197:     NVECS = NVECS + 1
198:     if ( NVECS.gt.MXVEC ) go to 540
199:     SVECNM(NVECS) = VNAME(LV:LLV)
200:     IVEC = NVECS
201: C
202: 110     continue
203: C
204:     JRVEC(IVEC,IVEC) = JRVEC(IVEC,IVEC) + 1
205:     IVLF(NSVECT) = IVEC
206: C
207: C
208: C===== @(vector ...) = ...
209: C
210: C
211:     else if ( VNAME(1:1).eq.'@' ) then
212: C
213:         call CATSTR( CBUFF, LCBUF, '@( ', IERR )
214: C
215: 120     call CHREAD( ' ', CWRK, ' ', NLEN, IT, IEND )
216:         if ( IEND.ne.0 ) then
217:             write(MSG,*) 'XXX(SMPINF) No data after "@('
218:             go to 550
219:         end if
220: C
221:         if ( NLEN.gt.0 ) then
222: C
223:             NSVECT = NSVECT + 1
224: C
225:             if ( INDEX(ALP,CWRK(1:1)).eq.0 ) then
226:                 write(MSG,7020) 'Invalid variable name', CWRK(:NLEN)
227:                 NERR = NERR + 1
228:             end if
229: C
230:             if ( NLEN.gt.LEN(SVECNM(1)) ) then
231:                 write(MSG,7020) 'Too long variable name', CWRK(:NLEN)
232:                 NERR = NERR + 1
233:                 NLEN = LEN(SVECNM(1))
234:             end if
235: C
236:             call CATSTR( CBUFF, LCBUF, CWRK(:NLEN), IERR )
237:             call CATSTR( CBUFF, LCBUF, ' ', IERR )
238:             if ( IERR.ne.0 ) go to 530
239: C
240:             do 130 IVEC = 1, NVECS
241:                 if ( SVECNM(IVEC).eq.CWRK(:NLEN) ) go to 140
242: 130         continue
243:                 NVECS = NVECS + 1
244:                 if ( NVECS.gt.MXVEC ) go to 540
245:                 SVECNM(NVECS) = CWRK(1:NLEN)
246:                 IVEC = NVECS
247: C
248: 140         JRVEC(IVEC,IVEC) = JRVEC(IVEC,IVEC) + 1
249:                 IVLF(NSVECT) = IVEC
250: C
251:                 if ( IT.eq.0 ) go to 120
252:             end if
253: C
254: C .... skip '=' ...
255: C
256: 150     call CHREAD( ' ', CWRK, '=', NLEN, IT, IEND )
257:         if ( IEND.ne.0 ) then
258:             write(MSG,*) 'XXX(SMPINF) No data after parameter name'
259:             go to 550
260:         end if

```

src/shared/smpinf.f

```

261: C
262:       if ( NLEN.gt.0 .or. IT.eq.0 ) then
263:         write(IMG,7020) 'Invalid data', CWRK(:NLEN)
264:         NERR      = NERR + 1
265:
266: Ccccccccccc if( jret.eq.1 ) return
267:
268:         go to 150
269:       end if
270: C
271:       call CATSTR( CBUFF, LCBUF, ' ) = ' , IERR )
272:       if ( IERR.ne.0 ) go to 520
273:     end if
274: C
275: C
276: C
277: C === input from current input-file pointer =====
278: C
279: C
280: C      #function-name [( ) parameters [( ) ] ] [ ; ]
281: C
282: C      or
283: C
284: C      formula ;
285: C
286: C
287: C
288: C      TERM      = '#( ; '
289: C
290: C      call CHREAD( ' ', CWRK, TERM(:3), NLEN, IT, IEND )
291: C      if ( IEND.ne.0 ) then
292: C        write(IMG,*) ' XXX(SMPINF) Unexpected end of input'
293: C        go to 550
294: C      end if
295: C ... TCS: termination character
296: C       TCS      = ' '
297: C       if ( IT.ne.0 ) TCS = TERM(IT:IT)
298: C
299: C       if( jret.eq.1 ) then
300: C         write(ipr,7910) 'UNEXPECTED END OF DATA.'
301: C         nerr = nerr + 1
302: C         return
303: C       endif
304: C
305: C
306: C === special 'WHEN' condition other than logical expression =====
307: C
308: C       [NOT-]IN-BODY( body ID # )
309: C       [NOT-]IN-ZONE( zone name )
310: C       [NOT-]IN-REGION( region name )
311: C
312: C
313: C       K1      = INDEX(CWRK(:NLEN), 'IN-BODY')
314: C       K2      = INDEX(CWRK(:NLEN), 'IN-ZONE')
315: C       K3      = INDEX(CWRK(:NLEN), 'IN-REGION')
316: C
317: C       if ( VNAME(1:4).eq.'WHEN'.and.TCS.eq.'('
318: C         & .and.(K1.ne.0.or.K2.ne.0.or.K3.ne.0) ) then
319: C
320: C ... IN-BODY/IN-ZONE are treated like function
321: C
322: C       call PACKCS( SRCSP, 'FUNCTION', 'FUNCTION', IRET )
323: C       call PACKCS( SRCSP, 'VECTOR NAME', CBUFF(:LCBUF), IRET )
324: C       call PACKCS( SRCSP, 'IN-WHAT', CWRK(:NLEN), IRET )
325: C

```

```

326: C ... mark that @X,@Y,@Z is reffered
327: C
328:       do 180 I = 1, 3
329:         if ( I.eq.1 ) XYZ = 'X'
330:         if ( I.eq.2 ) XYZ = 'Y'
331:         if ( I.eq.3 ) XYZ = 'Z'
332:         do 160 IVEC = 1, NVECS
333:           if ( SVECNM(IVEC).eq.XYZ ) go to 170
334:         160 continue
335: C
336:         write(IMG,7100) XYZ
337:         7100 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',
338:           & ' "WHEN [NOT-]IN-BODY/ZONE(...)" is used, but <',Al,
339:           & '> is not sampled yet. ')
340:         NERR      = NERR + 1
341:         call CNTERR( 'FATAL' )
342:         go to 180
343: C
344:         170 IVREF(IVEC) = IVREF(IVEC) + 1
345:         NRVECT = NRVECT + 1
346:         IVRG(NRVECT) = IVEC
347:         180 continue
348: C
349:         write(IPR,7000) CWRK(:NLEN)
350: C
351: C ----- [NOT-]IN-BODY
352: C
353:         if ( CWRK(:NLEN).eq.'IN-BODY' .or. CWRK(:NLEN).eq.'NOT-IN-BODY'
354:           & ) then
355: C
356: C         NB      = -1
357: C         call I4READ( 'BODY', IBDID, NA, -1, JRET )
358: C         if ( JRET.ne.0 ) NERR = NERR + 1
359: C         if ( NA.ne.1 ) then
360: C           write(IMG,7040) '[NOT-]IN-BODY must have one body ID.'
361: C           NERR      = NERR + 1
362: C           call CNTERR( 'FATAL' )
363: C           go to 430
364: C         end if
365: C
366: C         write(IPR,7200) 'BODY ID# :', IBDID
367: C
368: C         do 190 I = 1, NBODY
369: C           if ( IBSDA(2,I).eq.IBDID ) then
370: C             IBDN      = I
371: C             go to 200
372: C           end if
373: C         190 continue
374: C         write(IMG,7120) IBDN
375: C         7120 format(/1X,'XXX <SAMPLING INFORMATION ERROR> BODY-ID ',I7,
376:           & 'is not found in BODY-ID list. ')
377: C         call CNTERR( 'FATAL' )
378: C         go to 430
379: C
380: C
381: C ... construct zone-body specification like KZDA &
382: C
383: C         200 if ( IBDN.lt.NBODY ) then
384: C           IBl      = IBSDA(1,IBDN+1) - 1
385: C         else
386: C           IBl      = NBODY
387: C         end if
388: C         NSFN      = IBl - IBSDA(1,IBDN) + 1
389: C
390: C .. pack body ID & number of surfaces

```


src/shared/smpinf.f

```

391: C
392:       IWRK(1) = IBDID
393:       IWRK(2) = NSFN
394:       call PACKND( SRCSP, 'ID NSFN', 'I4', IWRK, 2, IRET2 )
395: C
396:       .. pack "KZAA" data ( "zone #1" has NSFN surfaces )
397: C
398:       IWRK(1) = 1
399:       IWRK(2) = 1 + NSFN
400:       call PACKND( SRCSP, '-KZAA-', 'I4', IWRK, 2, IRET2 )
401: C
402:       .. make "KZDA" data & pack it
403: C
404:       do 210 I = 1, NSFN
405:           IWRK(2*I-1) = IPSDA(1,IBSDA(1,IBDN)+I-1)
406:           IWRK(2*I)   = 0
407:       210 continue
408:       call PACKND( SRCSP, '-KZDA-', 'I4', IWRK, 2*NSFN, IRET2 )
409: C
410: C ----- [NOT-]IN-ZONE( zone-name-matching-patterns )
411: C
412:       else if ( CWRK(:NLEN).eq.'IN-ZONE'
413:       &         .or. CWRK(:NLEN).eq.'NOT-IN-ZONE' ) then
414: C
415:           NNZN = 0
416: C
417:       220 call CHREAD( 'ZONE#NAME', CWRK2, ' ', NL2, IT2, IEND2 )
418:       if ( IEND2.ne.0 ) then
419:           write(IMG,*) ' XXX(SMPINF) Unexpected end of data'
420:           go to 550
421:       end if
422: C
423: C ... construct zone # list in IWRK
424: C
425: cccc if ( IT2.eq.0 ) then
426:       if ( NL2.gt.0 ) then
427:           write(IPR,7240) 'INPUT-ZONE :', CWRK2(:NL2)
428:           do 250 IZ = 1, NZONE
429:               if ( IMATCH(CWRK2(:NL2),IZNAM(KINP2,IZ)).eq.1 ) then
430:                   do 230 J = 1, NNZN
431:                       if ( IWRK(J).eq.IZ ) go to 240
432:                   230 continue
433: C
434:                   NNZN = NNZN + 1
435:                   if ( NNZN.gt.NWORK ) then
436:                       write(IMG,7300) NWORK
437:                       NERR = NERR + 1
438:                       return
439:                   else
440:                       IWRK(NNZN) = IZ
441:                   end if
442:                   240 continue
443:               end if
444:           250 continue
445:           if ( IT2.eq.0 ) go to 220
446:       end if
447: C
448:       write(IPR,7180) 'ZONE # :', (IWRK(I),I=1,NNZN)
449: C
450:       call PACKND( SRCSP, '-NNZN-', 'I4', NNZN, 1, IRET2 )
451:       call PACKND( SRCSP, '-ZONES-', 'I4', IWRK, NNZN, IRET2 )
452: C
453: C ... set JUNDG flag (use ABS2ZN)
454: C
455:       if ( MOD(JUNDG/2,2).eq.0 ) JUNDG = JUNDG + 2

```

```

456: C
457: C ----- [NOT-]IN-REGION
458: C
459: cc else if( k3.ne.0 .and. cwrk(nlen-7:nlen).eq.'IN-REGION') then
460: C
461:       else
462:           write(IMG,7020) 'Unsupported WHEN condition', CWRK(:NLEN)
463:           NERR = NERR + 1
464:           call CNTERR( 'FATAL' )
465:       end if
466:       go to 430
467:   end if
468: C
469: C
470: C === simple value substitution or simple formula =====
471: C
472: C
473: CCC if ( TCS.ne.'#' ) then
474:       if ( TCS.ne.'#' ) then
475:           if ( TCS.eq.'(' ) then
476:               NLEN = NLEN + 1
477:               CWRK(NLEN:NLEN) = TCS
478:           end if
479: C
480:           if ( NSVECT.ne.1 ) then
481:               write(IMG,7040) ' Multi vector sampling is required !!'
482:               NERR = NERR + 1
483:           end if
484: C
485: C
486: C .... input until ';' appears .....
487: C
488: C
489:       if ( TCS.ne.';' ) then
490:           call GETSTR( ' ', CWRK(NLEN+1:), ';', NLLL, 1, IT, JERR,
491:           &             JRET )
492:           NLEN = NLEN + NLLL
493: C
494:           if ( JERR.eq.0.and.JRET.eq.0 ) then
495: C
496:               call CATSTR( CBUFF, LCBUF, CWRK(:NLEN), IERR )
497:               if ( IERR.ne.0 ) NERR = NERR + 1
498: C
499:               else if ( IT.eq.0 ) then
500: C
501:                   write(IMG,7040) '";" is missing or too long statement.'
502:                   NERR = NERR + 1
503:                   return
504: C
505:               end if
506:           end if
507: C
508: C .... @xxx = simple-value ;
509: C
510:       if ( TCS.eq.';' ) then
511:           call CATSTR( CBUFF, LCBUF, CWRK(:NLEN), IERR )
512:           if ( IERR.ne.0 ) NERR = NERR + 1
513:       end if
514: C
515: C
516: C .... check calculation formula ...
517: C
518: C
519:       call CDENTK( CBUFF(:LCBUF), IPR, IERR, CWRK, NSTMT, NPFL,
520:       &             NSTKM, IVECN )

```

src/shared/smpinf.f

```

521: C
522: C
523:       if ( IERR.ne.0 ) then
524:         NERR      = NERR + 1
525:       else
526:         MVSTK      = MAX(MVSTK,NSTKM)
527: C
528: C     ... check vector variable reference ...
529: C
530:       if ( IVECN.ne.0 ) then
531: C
532:         K          = INDEX(CWRK,'=') + 1
533:       260         K0         = K
534:         K          = INDEX(CWRK(K0),'@')
535: C
536:       if ( K.ne.0 ) then
537:         K          = K0 + K - 1
538:         do 270 I = K + 1, LEN(CWRK)
539:           if ( INDEX(ALP,CWRK(I:I)).eq.0
540:             & and.INDEX('0123456789',CWRK(I:I)).eq.0 ) go to
541:             & 280
542:       270         continue
543: C
544:       280         I          = I - 1
545:         do 290 IVEC = 1, NVECS
546:           if ( SVECNM(IVEC).eq.CWRK(K+1:I) ) go to 300
547:       290         continue
548: C
549:         write(MSG,7140) CWRK(K+1:I)
550:       7140         format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',
551:             & 'Reference to unsampled variable <A,A,>')
552:         NERR      = NERR + 1
553: C
554:         K          = I + 1
555:         go to 260
556: C
557:       300         IVREF(IVEC) = IVREF(IVEC) + 1
558:         NRVECT      = NRVECT + 1
559:         IVRG(NRVECT) = IVEC
560:         K          = I + 1
561: C
562:         go to 260
563:       end if
564: C
565: C     .... constant expression .... (no @variable reference )
566: C
567:       else
568:         K          = INDEX(CWRK,'=') + 1
569:         DWRK(1)    = DENTAK(CWRK(K:),IPR,IERR)
570:         if ( IERR.ne.0 ) then
571:           write(MSG,7020) 'Invalid data', CWRK(:NLEN)
572:           NERR      = NERR + 1
573:         end if
574:       end if
575: C
576:       K          = INDEX(CWRK,'=') + 1
577:       LCWRK      = ICLEN(CWRK)
578:       LCBUF      = INDEX(CBUFF(:LCBUF),'=')
579: C
580:       call CATSTR( CBUFF, LCBUF, CWRK(K:LCWRK), IERR )
581: C
582:       if ( IERR.ne.0 ) then
583:         NERR      = NERR + 1
584:       end if
585: C

```

```

586:       NL          = LCBUF/64
587: C
588:       if ( MOD(LCBUF,64).ne.0 ) then
589:         write(IPR,7000) (CBUFF(I:I+63),I=1,NL*64,64),
590:             & CBUFF(NL*64+1:LCBUF)
591:       else
592:         write(IPR,7000) (CBUFF(I:I+63),I=1,NL*64,64)
593:       end if
594: C
595: C
596: C     @vec =formula
597: C
598: C---<packing>-----
599: C
600:       call PACKCS( SRCSP, 'SAMPLING INFORMATION', CBUFF(:LCBUF),
601:             & IRET )
602: C-----
603: C
604: C
605:       if ( IRET.ne.0 ) then
606:         NERR      = NERR + 1
607:       end if
608: C
609:       end if
610: C
611:       go to 430
612:     end if
613: C
614: C
615: C
616: C
617: C     ===== sampling function === #function-name data ;
618: C
619: C     pack 'FUNCTION' & '@vec ='
620: C
621: C     & function name separately
622: C
623: C
624: C
625: C
626: C---<packing>-----
627: C     Csasa call packcs(srcsp,'SAMPLING INFORMATION',
628: C     Csasa& 'FUNCTION '//cbuff(:lcbuf),iret)
629: C
630: C     'FUNCTION'
631: C
632: C     call PACKCS( SRCSP, 'FUNCTION', 'FUNCTION', IRET )
633: C
634: C     '@VEC = '
635: C
636: C     call PACKCS( SRCSP, 'VECTOR NAME', CBUFF(:LCBUF), IRET )
637: C-----
638: C
639: C
640: C     function name
641: C
642: C     TERM      = '(;'
643: C     call CHREAD( ' ', CWRK, TERM, NLEN, IT, IEND )
644: C     if ( IEND.ne.0 ) then
645: C       write(MSG,*) ' XXX(SMPINF) Unexpected end of data'
646: C       go to 550
647: C     end if
648: C
649: C     TCS      = ' '
650: C     if ( IT.gt.0 ) TCS = TERM(IT:IT)

```

src/shared/smpinf.f

```

651: C
652: C--<packing>-----
653: C
654: CM   if( prog.eq.'GMVP' .and. CWRK(:NLEN).eq.'FISSION' ) then
655: CM       call PACKCS( SRCSP, 'SAMPLING FUNCTION', 'TABLE', IRET )
656: CM   else
657: CM       call PACKCS( SRCSP, 'SAMPLING FUNCTION', CWRK(:NLEN), IRET )
658: CM   end if
659: C
660: C   if ( IRET.ne.0 ) then
661: C       NERR = NERR + 1
662: C   end if
663: C
664: C-----
665: C
666: C   call CATSTR( CBUFF, LCBUF, '#', IERR )
667: C   call CATSTR( CBUFF, LCBUF, CWRK(:NLEN), IERR )
668: C   write(IPR,7000) CBUFF(:LCBUF)
669: C   if ( IERR.ne.0 ) then
670: C       NERR = NERR + 1
671: C   end if
672: C
673: C
674: C   7160 format(1X,'      >>  <'A,>'/(:(15X,1P,10(E11.4:)))
675: C   7180 format(1X,'      >>  <'A,>'/(:(15X,10(I5:)))
676: C   7200 format(1X,'      >>  <'A,I5>'
677: C   7220 format(1X,'      >>  <'A,>'/(:(15X,1P,5(E12.5:)))
678: C   7240 format(1X,'      >>  <'A,>' ,A)
679: C   7260 format(1X,'      >>  <'A,>' ,:1P,E12.5,: <'A,>' ,:1P,
680: C       &      E12.5,: <'A,>' ,:1P,E12.5)
681: C   7280 format(1X,'      >>  <'A,>' ,:1P,E12.5,: <'A,>' ,:1P,
682: C       &      E12.5,: <'A,>' ,:1P,E12.5,: <'A,>' ,:1P,E12.5)
683: C
684: C   7300 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',
685: C       &      ' WORKING AREA FOR INPUT IS INSUFFICIENT.'/5X,
686: C       &      ' INCREASE WORKING AREA SIZE BY SPECIFYING OR MODIFYING',
687: C       &      ' "NWORK" AFTER "$SOURCE". (CURRENT NWORK =',I6,')')
688: C   7320 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ', 'NUMBER OF DATA ',A,
689: C       &      ' IS NOT VALID')
690: C
691: C
692: C >>>>> for old-type system source or user source <<<<
693: C
694: C   if ( 'TYPE'.eq.CWRK(1:4) .or. 'USER'.eq.CWRK(1:4) ) then
695: C
696: C       write(FORMT,('( 'I',i1,' '))') NLEN - 4
697: C       read(CWRK(5:NLEN),fmt =FORMT) ISNUM
698: C
699: C   if ( 'TYPE'.eq.CWRK(1:4) ) then
700: C       write(IPR,7200) 'PRE-DEFINED SYSTEM SOURCE   TYPE: ', ISNUM
701: C   else
702: C       write(IPR,7200) 'USER DEFINED SOURCE   TYPE: ', ISNUM
703: C   end if
704: C
705: C   call R8READ( ' ', DWRK(1), NA, -NWORK, JRET )
706: C
707: C   if ( JRET.ne.0 ) NERR = NERR + 1
708: C
709: C   if ( 'USER'.eq.CWRK(1:4) ) then
710: C       write(IPR,7220) 'DATA GIVEN', (DWRK(I),I=1,NA)
711: C
712: C   else if ( CWRK(1:NLEN).eq.'TYPE1' ) then
713: C       write(IPR,7260) 'Point, isotropic source'
714: C       write(IPR,7260) 'X', DWRK(1), 'Y', DWRK(2), 'Z', DWRK(3)
715: C       if ( DWRK(10).eq.0.0.and.DWRK(9).eq.0.0 ) then

```

```

716: C       DWRK(9) = -1.D0
717: C       DWRK(10) = 1.D0
718: C   end if
719: C   write(IPR,7260) 'MU-MIN', DWRK(9), 'MU-MAX', DWRK(10)
720: C
721: C   else if ( CWRK(1:NLEN).eq.'TYPE2' ) then
722: C       write(IPR,7260) 'Point, mono-directional source'
723: C       write(IPR,7260) 'X', DWRK(1), 'Y', DWRK(2), 'Z', DWRK(3),
724: C       &      'A', DWRK(4), 'B', DWRK(5), 'C', DWRK(6)
725: C
726: C   else if ( CWRK(1:NLEN).eq.'TYPE3' ) then
727: C       write(IPR,7260) 'Uniform in RPP, isotropic source'
728: C       write(IPR,7260) 'XMIN', DWRK(1), 'XMAX', DWRK(2)
729: C       write(IPR,7260) 'YMIN', DWRK(3), 'YMAX', DWRK(4)
730: C       write(IPR,7260) 'ZMIN', DWRK(5), 'ZMAX', DWRK(6)
731: C       if ( DWRK(10).eq.0.0.and.DWRK(9).eq.0.0 ) then
732: C           DWRK(9) = -1.D0
733: C           DWRK(10) = 1.D0
734: C       end if
735: C       write(IPR,7260) 'MU-MIN', DWRK(9), 'MU-MAX', DWRK(10)
736: C
737: C   else if ( CWRK(1:NLEN).eq.'TYPE4' ) then
738: C       write(IPR,7260) 'Uniform in sphere, isotropic source'
739: C       write(IPR,7260) 'CX', DWRK(1), 'CY', DWRK(2), 'CZ', DWRK(3),
740: C       &      'RADIUS', DWRK(4)
741: C       if ( DWRK(10).eq.0.0.and.DWRK(9).eq.0.0 ) then
742: C           DWRK(9) = -1.D0
743: C           DWRK(10) = 1.D0
744: C       end if
745: C       write(IPR,7260) 'MU-MIN', DWRK(9), 'MU-MAX', DWRK(10)
746: C
747: C   else if ( CWRK(1:NLEN).eq.'TYPE5' ) then
748: C       write(IPR,7260) 'Uniform on disc, mono-directional source'
749: C       write(IPR,7260) 'CX', DWRK(1), 'CY', DWRK(2), 'CZ', DWRK(3),
750: C       &      'RADIUS', DWRK(4), 'A', DWRK(5), 'B', DWRK(6), 'C',
751: C       &      DWRK(7)
752: C
753: C   else if ( CWRK(1:NLEN).eq.'TYPE6' ) then
754: C       write(IPR,7260) 'Uniform on rectangle, mono-directional'
755: C       write(IPR,7260) 'CX', DWRK(1), 'CY', DWRK(2), 'CZ', DWRK(3),
756: C       &      'LINE-X', DWRK(4), 'LINE-Y', DWRK(5), 'LINE-Z',
757: C       &      DWRK(6), 'LINE-LENGTH', DWRK(7), 'A', DWRK(8), 'B',
758: C       &      DWRK(9), 'C', DWRK(10)
759: C
760: C   else if ( CWRK(1:NLEN).eq.'TYPE9' ) then
761: C       write(IPR,7260) 'FNS (D,T) neutron source'
762: C       write(IPR,7260) 'X0', DWRK(1), 'Y0', DWRK(2), 'Z0', DWRK(3),
763: C       &      'XR', DWRK(4), 'XDIREC', DWRK(5), 'TH1', DWRK(6),
764: C       &      'FA1', DWRK(7), 'PU1', DWRK(8)
765: C
766: C   else
767: C       write(IMG,*) 'XXX(SMPINF) Unsupported source type <',
768: C       &      CWRK(:NLEN),>'
769: C       NERR = NERR + 1
770: C   end if
771: C
772: C--<packing>-----
773: C
774: C   call PACKND( SRCSP, 'DATA', 'R8', DWRK(1), NA, IRET2 )
775: C   if ( IRET2.ne.0 ) NERR = NERR + 1
776: C
777: C-----
778: C
779: C >>>>> Fission source using position/nuclide information in a file <<<
780: C

```

src/shared/smpinf.f

```

781: C   @( X Y Z A B C W [E]) = #FISSIONFILE (
782: C**** nouse ** [ENERGY(EXACT|NONE)]
783: C*****          /** EXACT: sample using colliding nuclide
784: C*****          /**      and energy (for MVP)
785: C*****          /** NONE : do not sample energy
786: C*****          /**      ( "E" should not appear in @VEC
787: C*****          /**      and default for GMVP )
788: C       [FILE( "filename" )]
789: C       [IOUNIT( iounit )]
790: C       /** use file other than default
791: C       /** one assigned on I/O unit IOSI
792: C       /** (IOUNIT is not recommended)
793: C       [ NUSE(nouse) ] /** use upto nouse particles in file.
794: C       ;
795: C
796: C
797: C   else if ( '#FISSIONFILE'.eq.CWRK(1:NL2N).and.TCS.eq.'(' ) then
798: C
799: C       if ( NSVCT.lt.7 ) then
800: C           write(IMG,*)
801: C       &       'XXX(SMPINF) Number of vectors in @(...)=#FISSIONFILE',
802: C       &       ' must not be less than 7.'
803: C       call CNTERR( 'FATAL' )
804: C       else
805: C           do 310 I = 1, 3
806: C               if ( SVECNM(IVLF(I)).ne.'X'.and.SVECNM(IVLF(I)).ne.'Y'
807: C                   .and.SVECNM(IVLF(I)).ne.'Z' ) then
808: C                   &       write(IMG, '(lx,a,il,a,a,a,a)' ) '!!!(SMPINF) ', I,
809: C                   &       ' 'th vector <', SVECNM(IVLF(I)),
810: C                   &       ' is not <X>, <Y> or <Z>.',
811: C                   &       ' I hope it be intentional.'
812: C               call CNTERR( 'WARNING' )
813: C           end if
814: C       310 continue
815: C       do 320 I = 4, 6
816: C           if ( SVECNM(IVLF(I)).ne.'A'.and.SVECNM(IVLF(I)).ne.'B'
817: C               .and.SVECNM(IVLF(I)).ne.'C' ) then
818: C               &       write(IMG, '(lx,a,il,a,a,a,a)' ) '!!!(SMPINF) ', I,
819: C               &       ' 'th vector <', SVECNM(IVLF(I)),
820: C               &       ' is not <A>, <B> or <C>.',
821: C               &       ' I hope it be intentional.'
822: C           call CNTERR( 'WARNING' )
823: C       end if
824: C       320 continue
825: C       if ( SVECNM(IVLF(7)).ne.'W' ) then
826: C           &       write(IMG, '(lx,a,il,a,a,a,a)' )
827: C           &       '!!!(SMPINF) 7''th vector <',
828: C           &       SVECNM(IVLF(7)), ' is not <W>.',
829: C           &       ' I hope it be intentional.'
830: C       call CNTERR( 'WARNING' )
831: C       end if
832: C       if ( NSVCT.gt.7.and.SVECNM(IVLF(8)).ne.'E' ) then
833: C           &       write(IMG, '(lx,a,il,a,a,a,a)' )
834: C           &       '!!!(SMPINF) 8''th vector <',
835: C           &       SVECNM(IVLF(8)), ' is not <E>.',
836: C           &       ' I hope it be intentional.'
837: C       call CNTERR( 'WARNING' )
838: C       end if
839: C       end if
840: C
841: C       JEXACT = 0
842: C       NUSE = 0
843: C       SFILE = ' '
844: C       LSFILE = 1
845: C       IOSFIL = -1

```

```

846: C
847: C   330 call CHREAD( ' ', CWRK, '()', NL2, IT, IEND )
848: C       if ( IEND.ne.0 ) then
849: C           write(IMG,*) ' XXX(SMPINF) End of data during input for ',
850: C       &       '#FISSIONFILE'
851: C       go to 550
852: C       end if
853: C
854: C       .... "name(" ..
855: C       if ( IT.eq.1 ) then
856: C
857: C           if( CWRK(:NL2).eq.'ENERGY') then
858: C
859: C               call CHREAD( ' ', CWRK2, '()', NL3, IT2, IEND2 )
860: C               if ( IEND2.ne.0 ) then
861: C                   &       write(IPR,*) ' XXX No )" for "ENERGY(...)"',
862: C                   &       'of "#FISSIONFILE"'
863: C               go to 420
864: C               end if
865: C
866: C               write(IPR,7230) 'ENERGY',cwrk2(:nl3)
867: C
868: C               if( Cwrk2(:NL3).eq.'EXACT' ) then
869: C                   jexact = 1
870: C               else if( Cwrk2(:NL3).eq.'NONE' ) then
871: C                   jexact = 0
872: C               else
873: C                   &       write(IPR,7900)
874: C                   &       'Invalid Keyword for ENERGY(...) of #FISSIONFILE ',
875: C                   &       CWRK2(:NL3)
876: C                   NERR = NERR + 1
877: C               end if
878: C
879: C       else if( CWRK(:NL2).eq.'FILE') then
880: C
881: C       if ( CWRK(:NL2).eq.'FILE' ) then
882: C
883: C           call CHREAD( ' ', CWRK2, '()', NL3, IT2, IEND2 )
884: C           if ( IEND2.ne.0.or. IT2.eq.0 ) then
885: C               &       write(IMG,*) ' XXX(SMPINF) No )" for "FILE(...)"',
886: C               &       'of "#FISSIONFILE"'
887: C           go to 550
888: C           end if
889: C
890: C           write(IPR,7240) 'FILE', CWRK2(:NL3)
891: C
892: C           if ( IOSFIL.ge.0 ) then
893: C               &       write(IMG,*)
894: C               &       ' XXX(SMPINF) giving both "FILE(...)" and "IOUNIT(...)"',
895: C               &       ' in "#FISSIONFILE"'
896: C               NERR = NERR + 1
897: C           end if
898: C           SFILE = CWRK2(:NL3)
899: C           LSFILE = NL3
900: C
901: C       else if ( CWRK(:NL2).eq.'IOUNIT' ) then
902: C           call I4READ( 'IOUNIT', IOSFIL, NA, -1, JRET )
903: C           if ( JRET.ne.0 ) then
904: C               NERR = NERR + 1
905: C           else if ( SFILE.ne.' ' ) then
906: C               &       write(IMG,*)
907: C               &       ' XXX(SMPINF) giving both "FILE(...)" and "IOUNIT(...)"',
908: C               &       ' in "#FISSIONFILE"'
909: C               NERR = NERR + 1
910: C           end if

```

src/shared/smpinf.f

```

911:         write(IPR,7200) 'IOUNIT', IOSFIL
912: C
913:         else if ( CWRK(:NL2).eq.'NUSE' ) then
914:             call I4READ( 'NUSE', NUSE, NA, -1, JRET )
915:             if ( JRET.ne.0 ) then
916:                 NERR = NERR + 1
917:             end if
918:             write(IPR,7200) 'NUSE', NUSE
919:         else
920:             write(IMG,7020) 'Invalid data', CWRK2(:NL3)
921:             NERR = NERR + 1
922:         end if
923:         go to 330
924:     end if
925: C
926:     call SCFINP( SRCSP, SRCSP, SRCSP, PROG, 'FISSION', JDEBG,
927: & NSOUR, NSVECT, SFILE(:LSFILE), IOSFIL, NUSE, NUCID,
928: & NUC, IERR )
929: C
930: C
931: C >>>>> for 1 variable
932: C
933: C
934:     else if ( NSVECT.eq.1 ) then
935: C
936: C
937: C >>>> TABLE X( point ) P(probability) INT(interpolation) ;
938: C
939: C
940:     if ( 'TABLE'.eq.CWRK(:NLEN).and.TCS.eq.' ' ) then
941: C
942:         INTP = 0
943:         NX = 0
944:         NPX = 0
945:         NWK = NWORK
946:         LP = 1
947:         LX = 0
948:         LPX = 0
949: C
950:         IPDX = -1
951: C
952: C 340 call CHREAD( 'TABLE', CWRK, '(;', NLEN, IT, IEND )
953:     if ( IEND.ne.0 ) then
954:         write(IMG,*) 'XXX(SMPINF) No data after "#TABLE"'
955:         go to 550
956:     end if
957:
958:     if ( IT.eq.1 ) then
959:         NA = 0
960:         if ( CWRK(:NLEN).eq.'X' ) then
961:             LX = LP
962:             call R4READ( ' ', SWRK(LP), NA, -NWK, JRET )
963:             if ( JRET.ne.0 ) NERR = NERR + 1
964:             NX = NA
965:         else if ( CWRK(:NLEN).eq.'PX' ) then
966:             LPX = LP
967:             call R4READ( ' ', SWRK(LP), NA, -NWK, JRET )
968:             if ( JRET.ne.0 ) NERR = NERR + 1
969:             NPX = NA
970:             IPDX = 0
971:         else if ( CWRK(:NLEN).eq.'PDX' ) then
972:             LPX = LP
973:             call R4READ( ' ', SWRK(LP), NA, -NWK, JRET )
974:             if ( JRET.ne.0 ) NERR = NERR + 1
975:             NPX = NA

```

```

976:
977:         IPDX = 1
978:     else if ( CWRK(:NLEN).eq.'INT' ) then
979:         call CHREAD( 'INT', INTSYM, ' ', NL, IT, IEND )
980:         if ( IEND.ne.0 ) then
981:             write(IMG,*) 'XXX(SMPINF) No data after "INT"'
982:             go to 550
983:         end if
984:
985:         call INTPLC( INTSYM(:NL), INTP, IDIFF )
986:         if ( INTP.lt.0 ) then
987:             write(IMG,7040) 'Invalid interpolation ',
988: & INTSYM(:NL)
989:             NERR = NERR + 1
990:         end if
991:     else
992:         write(IMG,7020) 'Invalid data', CWRK(:NLEN)
993:         NERR = NERR + 1
994:     end if
995:     NWK = MAX(0,NWK-NA)
996:     LP = LP + NA
997:     go to 340
998: end if
999:
1000: if ( LP.gt.NWORK ) then
1001:     write(IMG,7300) NWORK
1002:     NERR = NERR + 1
1003:     return
1004: end if
1005:
1006: if ( NX.eq.0 .or. NPX.eq.0 ) then
1007:     write(IMG,7040) 'No input for "X", "PX" or "PDX"'
1008:     NERR = NERR + 1
1009: else
1010:     write(IPR,7160) 'X', (SWRK(I),I=LX,LX+NX-1)
1011:     if ( IPDX.eq.0 ) then
1012:         write(IPR,7160) 'PX', (SWRK(I),I=LPX,LPX+NPX-1)
1013:     else if ( IPDX.eq.1 ) then
1014:         write(IPR,7160) 'PDX', (SWRK(I),I=LPX,LPX+NPX-1)
1015:     end if
1016: C
1017: C
1018: C .... check order of X ...
1019:
1020:     ICMOD = -1
1021:     call CKODR4( SWRK(LX), NX, ICMOD, ITEST )
1022:     if ( ITEST.eq.1 ) then
1023:         ICMOD = -2
1024:         call CKODR4( SWRK(LX), NX, ICMOD, ITEST )
1025:         if ( ITEST.eq.1 ) then
1026:             NERR = NERR + 1
1027:             write(IMG,*) ' '
1028:             write(IMG,*) ' XXXX *****'
1029:             write(IMG,*) ' * X is out of order not allowed *'
1030:             write(IMG,*) ' *****'
1031:             write(IMG,*) ' '
1032:         end if
1033:     end if
1034: C
1035: C
1036: C .... check negative probability ...
1037:
1038:     call CKVAL4( 'PX', SWRK(LPX), NPX, ITEST, 0 )
1039:     if ( ITEST.eq.1 ) NERR = NERR + 1
1040:
1041:     if ( INTP.eq.0 ) then
1042:         if ( NPX.eq.NX ) then

```

src/shared/smpinf.f

```

1041:          INTSYM = 'DISCRETE'
1042:        else if ( NPX.eq.NX-1 ) then
1043:          INTSYM = 'STEP'
1044:        else
1045:          INTSYM = 'UNKNOWN'
1046:        end if
1047:        NL      = ICLEN(INTSYM)
1048:        call INTPLC( INTSYM(:NL), INTP, IDIFF )
1049:      end if
1050: C
1051:      write(IPR,7240) 'INTERPOLATION', INTSYM(:NL)
1052: C
1053:      if ( INTSYM(1:3).eq.'LOG' ) then
1054:        call CKVAL4( 'X', SWRK(LX), NX, ITEST, 1 )
1055:        if ( ITEST.eq.1 ) NERR = NERR + 1
1056:      end if
1057: C
1058:      if ( INTP.gt.0 ) then
1059:        if ( NX-NPX.ne.IDIFF ) then
1060:          write(IMG,7320) 'X' or "PX"
1061:          NERR = NERR + 1
1062:        else
1063: C
1064: C      ... probability data --> sampling form ...
1065: C
1066: C      ... sample intervals by PX(i)+PX(i+1) for linear interpolation
1067: C
1068: C      if ( INTSYM(:NL).eq.'LINEAR'
1069: &      .or. INTSYM(:NL).eq.'LOG-LINEAR' ) then
1070: C
1071: C      if ( LP+NPX-1.gt.NWORK ) then
1072: C        write(IMG,7300) NWORK
1073: C        NERR = NERR + 1
1074: C        return
1075: C      end if
1076: C      NPXP = NPX - 1
1077: C      LPXP = LP
1078: C      LP = LPXP + NPXP
1079: C
1080: C      if ( IPDX.eq.0 ) then
1081: C        do 350 I = 0, NPXP - 1
1082: C          SWRK(LPXP+I) = SWRK(LPX+I) + SWRK(LPX+I+1)
1083: 350 C        continue
1084: C      else if ( IPDX.eq.1 ) then
1085: C        do I = 0, NPXP - 1
1086: C          if ( INTSYM(:NL).eq.'LINEAR' ) then
1087: C            DEL = SWRK(LX+I) - SWRK(LX+I+1)
1088: C          else
1089: C            DEL = LOG( SWRK(LX+I) / SWRK(LX+I+1) )
1090: C          end if
1091: C          DEL = ABS( DEL )
1092: C
1093: C          SWRK(LPXP+I) = (SWRK(LPX+I)+SWRK(LPX+I+1))
1094: &          * DEL
1095: C        end do
1096: C      end if
1097: C
1098: C      ... interval/point sampling probability is as given for
1099: C      STEP & LOG-STEP interpolation
1100: C
1101: C      else
1102: C        NPXP = NPX
1103: C        LPXP = LPX
1104: C
1105: C        if ( IPDX.eq.1 .and.

```

```

1106: &          INTSYM(:NL).ne.'DISCRETE' ) then
1107: C        do I = 0, NPXP-1
1108: C          if ( INTSYM(:NL).eq.'STEP' ) then
1109: C            DEL = SWRK(LX+I) - SWRK(LX+I+1)
1110: C          else
1111: C            DEL = LOG( SWRK(LX+I) / SWRK(LX+I+1) )
1112: C          end if
1113: C          DEL = ABS( DEL )
1114: C
1115: C          SWRK(LPXP+I) = SWRK(LPXP+I) * DEL
1116: C        end do
1117: C      end if
1118: C
1119: C      if ( LP+2*NPXP.gt.NWORK ) then
1120: C        write(IMG,7300) NWORK
1121: C        NERR = NERR + 1
1122: C        return
1123: C      else
1124: C        LPAA = LP + NPXP
1125: C        call WALIAS( SWRK(LPXP), SWRK(LP), SWRK(LPAA),
1126: &        NPXP )
1127: C
1128: C      &
1129: C      C--<packing>-----
1130: C
1131: C      call PACKCS( SRCSP, 'INTERPOLATION',
1132: &      INTSYM(:NL), IRET1 )
1133: C      call PACKND( SRCSP, 'X-points', 'R4', SWRK(LX),
1134: &      NX, IRET2 )
1135: C      IRET3 = 0
1136: C      if ( INTSYM(:NL).eq.'LINEAR'
1137: &      .or. INTSYM(:NL).eq.'LOG-LINEAR' )
1138: C        call PACKND( SRCSP, 'PROBABILITY as given',
1139: &      'R4', SWRK(LPX), NPX, IRET3 )
1140: C
1141: C      call PACKND( SRCSP, 'PROBABILITY', 'R4',
1142: &      SWRK(LP), NPXP, IRET4 )
1143: C      call PACKND( SRCSP, 'ALIAS', 'I4', SWRK(LPAA),
1144: &      NPXP, IRET5 )
1145: C
1146: C      if ( ABS(IRET1)+ABS(IRET2)+ABS(IRET3)+ABS(IRET4)
1147: &      +ABS(IRET5).gt.0 ) NERR = NERR + 1
1148: C
1149: C      C-----
1150: C
1151: C      end if
1152: C      end if
1153: C      end if
1154: C      end if
1155: C
1156: C
1157: C
1158: C >>>> UNIFORM( bound1 bound2 ) [;]
1159: C
1160: C >>>> COSINE( bound1 bound2 ) [;]
1161: C
1162: C >>>> WATT( T ER ) [;]
1163: C      P(x) = exp(-x/T)*sinh(sqrt(ER*x))
1164: C
1165: C >>>> GAUSS( x0 sigma ) [;]
1166: C      P(x) = exp( -(x-x0)**2/(2*sigma**2) ) * (2*pai/sigma)**(1/2)
1167: C
1168: C >>>> MAXWELL( T ) [;]
1169: C
1170: C >>>> EVAPORATION( T ) [;]

```

```

1171: C
1172: C >>>>    POWER( a bound1 bound2) [;]
1173: C
1174:         else if ( ( 'UNIFORM'.eq.CWRK(:NLEN).or.'COSINE'.eq.CWRK(:NLEN)
1175:         &        .or.'WATT'.eq.CWRK(:NLEN).or.'GAUSS'.eq.CWRK(:NLEN)
1176:         &        .or.'MAXWELL'.eq.CWRK(:NLEN)
1177:         &        .or.'EVAPORATION'.eq.CWRK(:NLEN)
1178:         &        .or.'POWER'.eq.CWRK(:NLEN)).and.TCS.eq.'(' ) then
1179: C
1180:         NP          = 2
1181: C
1182:         if ( 'MAXWELL'.eq.CWRK(:NLEN)
1183:         &        .or.'EVAPORATION'.eq.CWRK(:NLEN) ) NP      = 1
1184:         if ( 'POWER'.eq.CWRK(:NLEN) ) NP      = 3
1185: C
1186:         call R8READ( ' ', DWRK, NA, NP, JRET )
1187:         if ( JRET.ne.0 ) NERR      = NERR + 1
1188: C
1189:         if ( NA.ne.NP ) then
1190:             NERR      = NERR + 1
1191:             write(1MSG,7320) CWRK(:NLEN)
1192:         else
1193: C
1194: C---<packing>-----
1195: C
1196:         call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK, NP, IRET )
1197:         if ( IRET.ne.0 ) NERR      = NERR + 1
1198: C
1199: C-----
1200: C
1201:         end if
1202: C
1203:         if ( 'UNIFORM'.eq.CWRK(:NLEN) .or. 'COSINE'.eq.CWRK(:NLEN)
1204:         &        then
1205:             write(IPR,7260) 'BOUND1', DWRK(1), 'BOUND2', DWRK(2)
1206:         else if ( 'WATT'.eq.CWRK(:NLEN) ) then
1207:             write(IPR,7260) 'T', DWRK(1), 'ER', DWRK(2)
1208:         else if ( 'GAUSS'.eq.CWRK(:NLEN) ) then
1209:             write(IPR,7260) 'X0', DWRK(1), 'SIGMA', DWRK(2)
1210:         else if ( 'MAXWELL'.eq.CWRK(:NLEN) ) then
1211:             write(IPR,7260) 'T', DWRK(1)
1212:         else if ( 'EVAPORATION'.eq.CWRK(:NLEN) ) then
1213:             write(IPR,7260) 'T', DWRK(1)
1214:         else if ( 'POWER'.eq.CWRK(:NLEN) ) then
1215:             write(IPR,7260) 'POWER', DWRK(1), 'BOUND1', DWRK(2),
1216:             &        'BOUND2', DWRK(3)
1217:         end if
1218: C
1219: C >>>>    NORMALIZE( a ) [;]
1220: C
1221:         else if ( 'NORMALIZE'.eq.CWRK(:NLEN).and.TCS.eq.'(' ) then
1222: C
1223:             NRVECT      = NSVECT
1224:             do 360 I = 1, NSVECT
1225:                 IVRG(I) = IVLFI(I)
1226:             360         continue
1227: C
1228:             NP          = -1
1229:             call R8READ( ' ', DWRK, NA, NP, JRET )
1230:             if ( JRET.ne.0 ) NERR      = NERR + 1
1231:             if ( NA.eq.0 ) DWRK(1)      = 0.0
1232:             write(IPR,7260) 'SUM', DWRK(1)
1233: C
1234: C---<packing>-----
1235: C

```

```

1236:      call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK, 1, IRET )
1237:      if ( IRET.ne.0 ) NERR = NERR + 1
1238:
1239: C-----
1240: C
1241: C
1242: C
1243: C >>>> <MVP>      FFISSION( nuclide-ID incident-energy ) [:]
1244: C >>>> <GMVP>      FFISSION( Mat-# ) [:]
1245: C
1246: C      prog='MVP' : sample from fission spectrum of specified nuclide
1247: C
1248: C      prog='GMVP': sample from fission spectrum of specified material
1249: C
1250: C
1251:      else if ( 'FFISSION'.eq.CWRK(:NLN).and.TCS.eq.(' ' ) then
1252:          if ( PROG.eq.'MVP' ) then
1253:              call CHREAD( ' ', CWRK, ' ' ), NK, IT, IEND )
1254:              if ( IEND.ne.0 ) then
1255:                  write(IMSG,*) 'XXX(SMPINF) No data after "FFISSION"'
1256:                  go to 550
1257:              end if
1258:
1259: CM      if ( NK.lt.0 .or. IT.eq.0 ) then
1260:          if ( NK.lt.0 ) then
1261:              write(IMSG,7040)
1262:          &      'Nuclide-id specification is invalid.'
1263:          NERR = NERR + 1
1264:          else
1265:              IM = 0
1266:              do 370 IIM = 1, NUC
1267: Ccccccccccccccccccc if ( NUCID(IM).eq.CWRK(:NK) ) go to 250
1268:              if ( IMATCH(CWRK(:NK),NUCID(IIM)).eq.1 ) then
1269:                  if ( IM.eq.0 ) then
1270:                      IM = IIM
1271:                  else
1272:                      L1 = ICLEN(NUCID(IM))
1273:                      L2 = ICLEN(NUCID(IIM))
1274:                      write(IMSG, '(//lx,5a)')
1275:          &      '!!!(SMPINF) "IFISM" matches more than one nuclides. <',
1276:          &      NUCID(IM) (:L1),
1277:          &      '>' (selected) and <',
1278:          &      NUCID(IIM) (:L2), '>'
1279:                      call CNTERR( 'WARNING' )
1280:                      end if
1281:                  end if
1282:          370      continue
1283: C
1284:      EINCD = 0.0253
1285:      if ( IT.eq.0 ) then
1286:          NP = -1
1287:          call R4READ( ' ', SWRK, NA, NP, JRET )
1288:          if ( JRET.ne.0 ) NERR = NERR + 1
1289:          if ( NA.eq.0 ) SWRK(1) = EINCD
1290:      else
1291:          SWRK(1) = EINCD
1292:      end if
1293:      if ( IM.eq.0 ) then
1294:          write(IMSG,7080) CWRK(:NK), NUC, (NUCID(I),I=1,NUC)
1295:          NERR = NERR + 1
1296:      end if
1297: Cc250      write(IPR,7230) 'nuclide', CWRK(:NK)
1298:      if ( IM.gt.0 ) then
1299:          write(IPR,7240) 'nuclide',
1300:          &      NUCID(IM) (:ICLEN(NUCID(IM)))

```

src/shared/smpinf.f

```

1301:      else
1302:        write(IPR,7240) 'nuclide', CWRK(:NK)
1303:      end if
1304:      write(IPR,'(1X,A,1PE12.5,A)')
1305:      &      INCIDENT ENERGY :', SWRK(1),
1306:      &      ' EV'
1307: C
1308: C---<packing>-----
1309:
1310:      call PACKND( SRCSP, 'FISSION', 'I4', IM, 1, IRET )
1311:      if ( IRET.ne.0 ) NERR = NERR + 1
1312:      call PACKND( SRCSP, 'FISSION', 'R4', SWRK(1), 1, IRET
1313:      &      )
1314:      if ( IRET.ne.0 ) NERR = NERR + 1
1315:
1316: C-----
1317:
1318:      end if
1319: C
1320: C
1321: C      .... for GMVP, translate to #TABLE function
1322: C
1323: C
1324:      else if ( PROG.eq.'GMVP' ) then
1325:        call I4READ( ' ', IM, NA, 1, JRET )
1326:        if ( JRET.ne.0 ) NERR = NERR + 1
1327:        write(IPR,7200) 'FISSION SPECTRUM OF MATERIAL', IM
1328: C
1329:        if ( JFISS.eq.0 ) then
1330:          write(IMG,*) 'XXX(SMPINF) #FISSION is specified ',
1331:          &          'but it needs option "FISSION".'
1332:          NERR = NERR + 1
1333:          call CNTERR( 'FATAL' )
1334:        end if
1335: C
1336: C
1337: C      if ( IM.lt.1 .or. IM.gt.NMAT ) then
1338: C        write(IPR,7910) 'INVALID MATERIAL #.'
1339: C        NERR = NERR + 1
1340: C      end if
1341: C
1342: C      .... convert Material ID to sequential #
1343: C
1344: C      do 380 IMI = 1, NMAT
1345: C        if ( IDMAT(IMI).eq.IM ) go to 390
1346: C        continue
1347: C        write(IMG,*) 'XXX(SMPINF) Material ID in #FISSION ',
1348: C        &        'does not match any material.'
1349: C        NERR = NERR + 1
1350: C        call CNTERR( 'FATAL' )
1351: C        return
1352: C        IMI = 1
1353: C      390 IM = IMI
1354: C      if ( LFKAI.eq.0 ) then
1355: C        write(IPR,*) ' XXX(SMPINF) No fission spectra ',
1356: C        &        'are given so far.'
1357: C        write(IPR,*) ' Please input FKAI ',
1358: C        &        'before $SOURCE data block.'
1359: C        NERR = NERR + 1
1360: C        call CNTERR( 'FATAL' )
1361: C        return
1362: C      end if
1363: C
1364: C      LP = 1
1365: C

```

```

1366: CM      NX = NTGX
1367: CM      LX = LP
1368: CM      LPAA = LX + NX
1369: CM      LPFK = LPAA + NX
1370: CM      LP = LPFK + NX
1371: CM      if ( LP.gt.NWORK ) then
1372: CM        write(IPR,7260) NWORK
1373: CM        NERR = NERR + 1
1374: CM        return
1375: CM      else
1376: C
1377: CM        do 392 I=1,NX
1378: CM          SWRK(LPFK+I-1) = FKAI(I,IM)
1379: CM          continue
1380: C
1381: CM          call WALIAS( FKAI(1,IM), SWRK(LP), IWRK(LPAA), NX )
1382: CM          call WALIAS( SWRK(LPFK), SWRK(LP), IWRK(LPAA), NX )
1383: C
1384: CM          do 400 I = 1, NX
1385: CM            SWRK(LX+I-1) = REAL(I)
1386: CM            continue
1387: C
1388: C---<packing>-----
1389:
1390: CM      call PACKCS( SRCSP, 'INTERPOLATION', 'DISCRETE', IRET1
1391: CM      &      )
1392: CM      call PACKND( SRCSP, 'X-points', 'R4', SWRK(LX), NX,
1393: CM      &      IRET2 )
1394: CM      call PACKND( SRCSP, 'PROBABILITY', 'R4', SWRK(LP), NX,
1395: CM      &      IRET3 )
1396: CM      call PACKND( SRCSP, 'ALIAS', 'I4', IWRK(LPAA), NX,
1397: CM      &      IRET4 )
1398: CM
1399: CM      if ( ABS(IRET1)+ABS(IRET2)+ABS(IRET3)+ABS(IRET4).gt.0
1400: CM      &      ) NERR = NERR + 1
1401: C
1402: CM      call PACKND( SRCSP, 'FISSION', 'I4', IM, 1, IRET )
1403: CM      if ( IRET.ne.0 ) NERR = NERR + 1
1404: C
1405: C-----
1406: C
1407: C
1408: CM      end if
1409: CM      end if
1410: C
1411: C >>>> FEYNMAN( bound1 bound2 n-average n-variance ) [:]
1412: C
1413: C      else if ( 'FEYNMAN'.eq.CWRK(:NLEN).and.TCS.eq.'(' ) then
1414: C
1415: C        NP = 4
1416: C
1417: C        call R8READ( ' ', DWRK, NA, NP, JRET )
1418: C        if ( JRET.ne.0 ) NERR = NERR + 1
1419: C
1420: C        if ( NA.ne.NP ) then
1421: C          NERR = NERR + 1
1422: C          write(IMG,7320) CWRK(:NLEN)
1423: C        else
1424: C
1425: C          if ( DWRK(3).lt.1. ) then
1426: C            write(IMG,*)
1427: C            &            '!!!(SMPINF) N-AVE in #FEYNMAN() is',
1428: C            &            ' less than unity. ',
1429: C            &            ' It is changed to unity.'
1430: C            DWRK(3) = 1.D0

```


src/shared/smpinf.f

```

1431:          call CNTERR('WARNING')
1432:        end if
1433:        write(IPR,7280) 'BOUND1', DWRK(1), 'BOUND2', DWRK(2),
1434:          &          'N-AVE', DWRK(3), 'N-VAR', DWRK(4)
1435: C
1436: C---<packing>-----
1437:
1438:          call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK, NP, IRET )
1439:          if ( IRET.ne.0 ) NERR = NERR + 1
1440:
1441: C-----
1442: C
1443:        end if
1444: C
1445: C
1446: C >>>> error <<<<
1447: C
1448:        else
1449:          write(IMG,7060)
1450:          &          'Invalid sampling function or sampling dimension mismatch'
1451:          &          CWRK(:NLEN), TCS
1452:          NERR = NERR + 1
1453:        end if
1454: C
1455: C
1456: C >>>>> for 2 variable
1457: C
1458: C
1459:        else if ( NSVECT.eq.2 ) then
1460: C
1461: C >>>> TABLEX Y(...) PX(...) INTX(interpolation)
1462: C          Y(...) PY(...) INTY(interpolation)
1463: C          Y(...) PY(...) INTY(interpolation)
1464: C          Y(...) PY(...) INTY(interpolation)
1465: C          .....
1466: C
1467:        if ( 'TABLEXY'.eq.CWRK(:NLEN) ) then
1468: C
1469:          call TBLXY( SRCSP, NSOUR, IERR, NERR, IWRK, SWRK, DWORK,
1470:          &          NWORK )
1471: C
1472: C
1473: C >>>> DISC( r1 r2 [ang1 ang2]) [;]
1474: C
1475: C >>>> CIRCLE( r [ang1 ang2]) [;]
1476: C
1477:        else if ( ( 'DISC'.eq.CWRK(:NLEN).or.'CIRCLE'.eq.CWRK(:NLEN) )
1478:          &          .and.TCS.eq.'(' ) then
1479:          NP = 4
1480:          if ( 'CIRCLE'.eq.CWRK(:NLEN) ) NP = 3
1481:          call R8READ( ' ', DWRK, NA, -NP, JRET )
1482: C
1483:          if ( JRET.ne.0 ) NERR = NERR + 1
1484: C
1485:          if ( 'DISC'.eq.CWRK(:NLEN) ) then
1486:            if ( NA.lt.2 .or. NA.gt.4 ) then
1487:              write(IMG,7040) 'Number of data is invalid:',
1488:              &              'Please input as #DISC( r1 r2 ) ;'
1489:              NERR = NERR + 1
1490:            else if ( NA.eq.3 ) then
1491:              write(IMG,7040) 'Angle bound is incomplete',
1492:              &              'Please input as #DISC( r1 r2 angle1 angle2 ) ;'
1493:              NERR = NERR + 1
1494:            else if ( NA.eq.2 ) then
1495:              DWRK(3) = 0.0D0

```

```

1496:          DWRK(4) = 360.0D0
1497:        end if
1498:        write(IPR,7260) 'RADIUS1', DWRK(1), 'RADIUS2', DWRK(2)
1499:        write(IPR,7260) 'ANGLE1', DWRK(3), 'ANGLE2', DWRK(4)
1500:        DWRK(3) = DWRK(3) /180.0*PAI
1501:        DWRK(4) = DWRK(4) /180.0*PAI
1502:        NP = 4
1503:
1504:        else if ( 'CIRCLE'.eq.CWRK(:NLEN) ) then
1505:          if ( NA.lt.1 .or. NA.gt.3 ) then
1506:            write(IMG,7040) 'Number of data is invalid:',
1507:            &            'Please input as #CIRCLE( r ) ;'
1508:            NERR = NERR + 1
1509:          else if ( NA.eq.2 ) then
1510:            write(IMG,7040) 'Angle bound is incomplete',
1511:            &            'Please input as #CIRCLE( r angle1 angle2 ) ;'
1512:            NERR = NERR + 1
1513:          else if ( NA.eq.1 ) then
1514:            DWRK(2) = 0.0D0
1515:            DWRK(3) = 360.0D0
1516:          end if
1517:          write(IPR,7260) 'RADIUS', DWRK(1)
1518:          write(IPR,7260) 'ANGLE1', DWRK(2), 'ANGLE2', DWRK(3)
1519:          DWRK(2) = DWRK(2) /180.0*PAI
1520:          DWRK(3) = DWRK(3) /180.0*PAI
1521:          NP = 3
1522:        end if
1523: C
1524: C---<packing>-----
1525:
1526:          call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK, NP, IRET )
1527:          if ( IRET.ne.0 ) NERR = NERR + 1
1528:
1529: C-----
1530: C
1531: C >>>> SWAP( ) [;]
1532: C
1533:        else if ( 'SWAP'.eq.CWRK(:NLEN).and.TCS.eq.'(' ) then
1534:          call DMREAD( 'SWAP', NA, NB, JRET )
1535: C
1536:          NRVECT = NSVECT
1537:          do I = 1, NSVECT
1538:            IVRG(I) = IVLF(I)
1539:          end do
1540: C
1541: C >>>> ROTATE2D( theta ) [;]
1542: C
1543:        else if ( 'ROTATE2D'.eq.CWRK(:NLEN).and.TCS.eq.'(' ) then
1544:          call R8READ( ' ', DWRK, NA, -1, JRET )
1545:          if ( JRET.ne.0 ) NERR = NERR + 1
1546:          if ( NA.eq.0 ) then
1547:            write(IMG,7040)
1548:            &            'No input for rotation angle for #ROTATE2D'
1549:            NERR = NERR + 1
1550:          else
1551:            write(IPR,7260) 'ROTATION ANGLE (DEGREE)', DWRK(1)
1552: C
1553:            DWRK(1) = PAI/180.0D0*DWRK(1)
1554:            DWRK(2) = COS(DWRK(1))
1555:            DWRK(3) = SIN(DWRK(1))
1556: C
1557:            NRVECT = NSVECT
1558:            do I = 1, NSVECT
1559:              IVRG(I) = IVLF(I)
1560:            end do

```

src/shared/smpinf.f

```

1561:
1562: C--<packing>-----
1563:
1564:          call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK(2), 2, IRET )
1565:          if ( IRET.ne.0 ) NERR = NERR + 1
1566:
1567: C-----
1568:
1569:          end if
1570: C
1571: C >>>> error <<<<
1572:
1573:          else
1574:            write(IMG,7060)
1575:            & 'Invalid sampling function or sampling dimension mismatch',
1576:            & CWRK(:NLEN), TCS
1577:            NERR = NERR + 1
1578:          end if
1579: C
1580: C
1581: C >>>>> for 3 variable
1582: C
1583: C
1584:          else if ( NSVECT.eq.3 ) then
1585: C
1586: C >>>> SPHERE( r1 r2 [mu1 mu2] ) [;]
1587: C >>>> ISOTROPIC( [mu1 [mu2]] ) [;]
1588: C >>>> RESIZE( norm ) [;]
1589: C
1590:          if ( ('SPHERE'.eq.CWRK(:NLEN).or.'ISOTROPIC'.eq.CWRK(:NLEN))
1591:            & .or.'RESIZE'.eq.CWRK(:NLEN)).and.TCS.eq.'(') then
1592: C
1593: C
1594:          if ( 'SPHERE'.eq.CWRK(:NLEN) ) then
1595:            NP = 4
1596:            call R8READ( ' ', DWRK, NA, -NP, JRET )
1597: C
1598:            if ( JRET.ne.0 ) NERR = NERR + 1
1599: C
1600:            if ( NA.lt.2 ) then
1601:              NERR = NERR + 1
1602:              write(IMG,7040)
1603:            & 'Radius range data is incomplete for #SPHERE'
1604:            end if
1605:            if ( NA.le.2 ) DWRK(3) = -1.0D0
1606:            if ( NA.le.3 ) DWRK(4) = 1.0D0
1607:            write(IPR,7260) 'RADIUS1', DWRK(1), 'RADIUS2', DWRK(2)
1608:            write(IPR,7260) 'MU1', DWRK(3), 'MU2', DWRK(4)
1609: C
1610:          else if ( 'ISOTROPIC'.eq.CWRK(:NLEN) ) then
1611:            NP = 2
1612:            call R8READ( ' ', DWRK, NA, -NP, JRET )
1613: C
1614:            if ( JRET.ne.0 ) NERR = NERR + 1
1615: C
1616:            if ( NA.le.0 ) DWRK(1) = -1.0D0
1617:            if ( NA.le.1 ) DWRK(2) = 1.0
1618:            write(IPR,7260) 'MU1', DWRK(1), 'MU2', DWRK(2)
1619: C
1620:          else if ( 'RESIZE'.eq.CWRK(:NLEN) ) then
1621:            NP = 1
1622:            call R8READ( ' ', DWRK, NA, -1, JRET )
1623:            if ( JRET.ne.0 ) NERR = NERR + 1
1624:            if ( NA.le.0 ) then
1625:              NERR = NERR + 1

```

```

1626:          write(IMG,7040) 'No input for #RESIZE length'
1627:        else
1628:          write(IPR,7260) 'VECTOR LENGTH RESIZED', DWRK(1)
1629:        end if
1630: C
1631:        NRVECT = NSVECT
1632:        do 400 I = 1, NSVECT
1633:          IVRG(I) = IVLF(I)
1634:        400 continue
1635:      end if
1636: C
1637: C--<packing>-----
1638:
1639:          call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK, NP, IRET )
1640:          if ( IRET.ne.0 ) NERR = NERR + 1
1641:
1642: C-----
1643: C
1644: C >>>> ROTATED3D( theta ax ay az ) [;]
1645: C
1646:          else if ( 'ROTATE3D'.eq.CWRK(:NLEN).and.TCS.eq.'(') then
1647: C
1648:            call R8READ( ' ', DWRK, NA, -4, JRET )
1649:            if ( JRET.ne.0 ) NERR = NERR + 1
1650:            if ( NA.ne.4 ) then
1651:              write(IMG,7040)
1652:              & 'Number of data is invalid for "ROTATE3D" function.',
1653:              & 'Please input as #ROTATED3D( theta ax ay az ) ;'
1654:              NERR = NERR + 1
1655:            end if
1656: C
1657:            write(IPR,7260) 'ROTATION ANGLE (DEGREE)', DWRK(1)
1658:            write(IPR,7260) 'AXIS-X', DWRK(2), 'AXIS-Y', DWRK(3),
1659:            & 'AXIS-Z', DWRK(4)
1660: C
1661:            DWRK(1) = PAI/180.0D0*DWRK(1)
1662:            DWRK(5) = COS(DWRK(1))
1663:            DWRK(6) = SIN(DWRK(1))
1664: C
1665:            DTEMP = SQRT(DWRK(2)**2+DWRK(3)**2+DWRK(4)**2)
1666:            DWRK(7) = DWRK(2) /DTEMP
1667:            DWRK(8) = DWRK(3) /DTEMP
1668:            DWRK(9) = DWRK(4) /DTEMP
1669: C
1670:            NRVECT = NSVECT
1671:            do 410 I = 1, NSVECT
1672:              IVRG(I) = IVLF(I)
1673:            410 continue
1674: C
1675: C--<packing>-----
1676:
1677:          call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK(5), 5, IRET )
1678:          if ( IRET.ne.0 ) NERR = NERR + 1
1679:
1680: C-----
1681: C
1682: C
1683: C >>>> NEWAXIS( ax1 ay1 az1 ax2 ay2 az2 ) [;]
1684: C
1685:          else if ( 'NEWAXIS'.eq.CWRK(:NLEN).and.TCS.eq.'(') then
1686: C
1687:            call R8READ( ' ', DWRK, NA, -6, JRET )
1688:            if ( JRET.ne.0 ) NERR = NERR + 1
1689:            if ( NA.ne.6 ) then
1690:              write(IMG,7040)

```

src/shared/smpinf.f

```

1691:      &      'Number of data is invalid for "NEWAXIS" function.',
1692:      &      'Please input as #NEWAXIS( ax1 ay1 az1 ax2 ay2 az2 ) ;'
1693:      NERR      = NERR + 1
1694:      end if
1695: C
1696:      write(IPR,7260) 'AXIS1-X', DWRK(1), 'AXIS1-Y', DWRK(2),
1697:      &      'AXIS1-Z', DWRK(3)
1698:      write(IPR,7260) 'AXIS2-X', DWRK(4), 'AXIS2-Y', DWRK(5),
1699:      &      'AXIS2-Z', DWRK(6)
1700: C
1701:      DTEMP1 = SQRT(DWRK(1)**2+DWRK(2)**2+DWRK(3)**2)
1702:      if ( ABS(DTEMP1-1.0D0).ge.1.0D-5 ) then
1703:      write(IMG,*)
1704:      &      '!!!(SMPINF) Axis vector-1 of #NEWAXIS does not seem to be a ',
1705:      &      'unit vector (length = ', DTEMP1, ' )'
1706:      call CNTERR( 'WARNING' )
1707:      end if
1708: C
1709:      DTEMP2 = SQRT(DWRK(4)**2+DWRK(5)**2+DWRK(6)**2)
1710:      if ( ABS(DTEMP2-1.0D0).ge.1.0D-5 ) then
1711:      write(IMG,*)
1712:      &      '!!!(SMPINF) axis vector-2 of #NEWAXIS does not seem to be a ',
1713:      &      'unit vector (length = ', DTEMP2, ' )'
1714:      call CNTERR( 'WARNING' )
1715:      end if
1716: C
1717:      DTEMP = DWRK(1)*DWRK(4) + DWRK(2)*DWRK(5) + DWRK(3)*
1718:      &      DWRK(6)
1719:      if ( ABS(DTEMP/DTEMP1/DTEMP2).ge.1.0D-5 ) then
1720:      write(IMG,*)
1721:      &      '!!!(SMPINF) Axis vector-1 and 2 of #NEWAXIS does not compose',
1722:      &      'right angle (cosine = ', DTEMP, ' )'
1723:      call CNTERR( 'WARNING' )
1724:      end if
1725:
1726: C      .. third axis vector ( normalize )
1727:
1728:      DWRK(7) = DWRK(2)*DWRK(6) - DWRK(3)*DWRK(5)
1729:      DWRK(8) = DWRK(3)*DWRK(4) - DWRK(1)*DWRK(6)
1730:      DWRK(9) = DWRK(1)*DWRK(5) - DWRK(2)*DWRK(4)
1731:      DTEMP = SQRT(DWRK(7)**2+DWRK(8)**2+DWRK(9)**2)
1732:      DWRK(7) = DWRK(7) /DTEMP
1733:      DWRK(8) = DWRK(8) /DTEMP
1734:      DWRK(9) = DWRK(9) /DTEMP
1735: C
1736:      NRVECT = NSVECT
1737:      do 420 I = 1, NSVECT
1738:      IVRG(I) = IVLF(I)
1739:      420 continue
1740: C
1741: C---<packing>-----
1742:
1743:      call PACKND( SRCSP, 'PARAMETER', 'R8', DWRK(1), 9, IRET )
1744:      if ( IRET.ne.0 ) NERR = NERR + 1
1745:
1746: C-----
1747: C
1748: C
1749: C >>>> error <<<<
1750: C
1751:      else
1752:      write(IMG,7060)
1753:      &      'Invalid sampling function or sampling dimension mismatch',
1754:      &      CWRK(:NLEN), TCS
1755:      NERR = NERR + 1
1756:
1757:      end if
1758:
1759: C
1760:      430 continue
1761: C
1762: C ===== set/reset vector connection (relation) flags =====
1763: C
1764: C
1765: C
1766:      if ( NRVECT.eq.0 ) then
1767: C
1768: C      .... reset all connection with other vectors
1769: C      when newly sampled without referencing any other vectors
1770: C
1771:      do 450 I = 1, NSVECT
1772:      do 440 K = 1, NVECS
1773:      if ( IVLF(I).ne.K ) then
1774:      JRVEC(IVLF(I),K) = 0
1775:      JRVEC(K,IVLF(I)) = 0
1776:      end if
1777:      440 continue
1778:      450 continue
1779:      do 470 I = 1, NSVECT
1780:      do 460 J = 1, NSVECT
1781:      if ( IVLF(I).ne.IVLF(J) ) then
1782:      JRVEC(IVLF(I),IVLF(J)) = 1
1783:      JRVEC(IVLF(J),IVLF(I)) = 1
1784:      end if
1785:      460 continue
1786:      470 continue
1787: C
1788:      else
1789:      do 490 I = 1, NSVECT
1790:      do 480 J = 1, NSVECT
1791:      if ( IVLF(I).ne.IVLF(J) ) then
1792:      JRVEC(IVLF(I),IVLF(J)) = 1
1793:      JRVEC(IVLF(J),IVLF(I)) = 1
1794:      end if
1795:      480 continue
1796:      490 continue
1797: C
1798: C      ....
1799: C
1800:      do 520 J = 1, NRVECT
1801:      do 510 I = 1, NSVECT
1802:      do 500 K = 1, NVECS
1803:      if ( JRVEC(IVRG(J),K).ne.0 ) then
1804:      if ( IVLF(I).ne.K ) then
1805:      JRVEC(IVLF(I),K) = 1
1806:      JRVEC(K,IVLF(I)) = 1
1807:      end if
1808:      end if
1809:      500 continue
1810:      510 continue
1811:      520 continue
1812:      end if
1813: C
1814:      return
1815: C
1816: C
1817:      7340 format(1X,'XXX <SAMPLING INFORMATION ERROR> ',
1818:      &      'CHARACTER INPUT BUFFER OVERFLOW.'/(5X,'<','A','>'))
1819:      7360 format(1X,'XXX <SAMPLING INFORMATION ERROR> ',
1820:      &      'TOO MANY VECTOR VARIABLES APPEARED. (MAX='&A','>'))

```

src/shared/smpinf.f

```
1821: C
1822:   530 continue
1823:       write(IPR,7340) (CBUFF(I:MIN(I+63,LCBUF)),I=1,LCBUF,64)
1824:       write(IPR,*) ' <buffer length = ', LEN(CBUFF), '>'
1825:       NERR      = NERR + 1
1826:       return
1827: C
1828:   540 continue
1829:       write(IMG,7360) MXVEC
1830:       NERR      = NERR + 1
1831:       return
1832: C
1833: C ... generic handling of end-of-data ...
1834: C
1835:   550 call PRSTOP( 1, 'UNEXPECTED END OF DATA IN SOURCE SPECIFICATION' )
1836:       stop 888
1837:       end
```

src/shared/smplx1.f

```

1:      subroutine SMPLX1( PROG, NS,   X,   FNAME, DSRC, NDSRC,
2:      &                IRAND, RWK,   NRWK, IERROR,NERROR )
3: C==<MVP/GMVP>=====
4: C purpose:  source parameter sampling ( 1 parameter )
5: C
6: C   Samples from the following type of distributions:
7: C
8: C   UNIFORM
9: C   COSINE
10: C   WATT
11: C   GAUSS
12: C   MAXWELL
13: C   EVAPORATION
14: C   POWER
15: C   NORMALIZE (normalization)
16: C
17: C called in : SYSRC (source generation routine.)
18: C calls:  RANU2
19: C-----
20: C
21: C arguments (i =input, o=output, w=work, c=constant )
22: C
23: CLASS arg:  meanings
24: C
25: C i  prog : program name ('MVP','GMVP')
26: C i  ns  : number of source to be generated by this source set (input)
27: C io X(*) : variable to be sampled
28: C i  fname : sampling "function" name.
29: C i  data : data for sampling.
30: C i  ndata : number of ielement on data array.
31: C i  irand : seed of random numbers
32: C w  rwk(ns,nrwk): working array mainly used to save random numbers.
33: C i  nrwk : second dimension of rwk array.
34: C o  ierror : error flag ( 0 = no-error, 1 = error )
35: C io nerror : number of errors
36: C
37: C =====
38:      implicit real*8(D)
39: C
40:      character*(*) PROG, FNAME
41: C
42:      real*8 X(NS), DSRC(NDSRC)
43: C
44:      real RWK(NS,NRWK)
45: C
46: C
47:      real*8 PAI, DPAI, HPAI
48:      parameter( PAI = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
49:      &          0.5D0*PAI )
50: C
51:      include 'INC/_IOUNIT'
52: C
53:      IERROR = 0
54: C
55: C
56: C
57: C   >>> UNIFORM <<<
58: C
59: C
60:      if ( 'UNIFORM'.eq.FNAME ) then
61: C
62:          call RANU2( IRAND, RWK, NS, ICOD )
63: C
64:          if ( DSRC(1).ne.0.0D0 .or. DSRC(2).ne.1.0D0 ) then
65:              D1

```

```

66:              *VOCL LOOP,NOVREC
67:                  do 100 I = 1, NS
68:                      X(I) = DSRC(1) + D1*RWK(I,1)
69:                  100 continue
70:              else
71:              *VOCL LOOP,NOVREC
72:                  do 110 I = 1, NS
73:                      X(I) = RWK(I,1)
74:                  110 continue
75:              end if
76: C
77: C
78: C   >>> COSINE <<<
79: C
80: C
81:              else if ( 'COSINE'.eq.FNAME ) then
82:                  call RANU2( IRAND, RWK, NS, ICOD )
83:                  D1 = DSRC(2) - DSRC(1)
84:              *VOCL LOOP,NOVREC
85:                  do 120 I = 1, NS
86:                      X(I) = DSRC(1) + D1*(ASIN(2D0*RWK(I,1)-1D0)/PAI+0.5D0)
87:                  120 continue
88: C
89: C
90: C   >>> WATT <<<
91: C
92: C
93: C
94: C
95:              else if ( 'WATT'.eq.FNAME ) then
96:                  call RANU2( IRAND, RWK, 4*NS, ICOD )
97: C
98: C
99: C
100:              *VOCL LOOP,NOVREC
101:                  do 130 I = 1, NS
102:                      DV = -DSRC(1)*
103:                      &      (LOG(RWK(I,1))+LOG(RWK(I,2))*COS(DPAI*RWK(I,3))*2)
104:                      X(I) = DV + DSRC(1)*
105:                      &      ((2*RWK(I,4)-1)*SQRT(DV*DSRC(2))+0.25D0*DSRC(1)*
106:                      &      DSRC(2))
107:                  130 continue
108: C
109: C
110: C
111: C   >>> GAUSS <<<      exp(-(x-x0)**2/(2*sigma**2))
112: C
113: C
114: C
115: C
116: C
117: C
118: C
119: C
120: C
121: C   ... Box-Muller method ...
122: C
123:              *VOCL LOOP,NOVREC
124:                  do 140 I = 1, NS
125:                      X(I) = DSRC(1) + DSRC(2)*SQRT(-2D0*LOG(RWK(I,1)))*
126:                      &      COS(DPAI*RWK(I,2))
127:                  140 continue
128: C
129: C
130: C

```

src/shared/smplx1.f

```

131: C      >>> MAXWELL <<<  sqrt(x)*exp(-x/T)
132: C
133: C
134:       else if ( 'MAXWELL'.eq.FNAME ) then
135:           call RANU2( IRAND, RWK, 3*NS, ICOD )
136: C
137: C           ... p(x)=exp(-x) + square of Gaussian sample
138: C
139: *VOCL LOOP,NOVREC
140:       do 150 I = 1, NS
141:           X(I) = -DSRC(1)*
142: &             (LOG(RWK(I,1))+LOG(RWK(I,2)))*COS(DPAI*RWK(I,3))**2)
143: 150       continue
144: C
145: C
146: C      >>> EVAPORATION <<<  x*exp(-x/T)
147: C
148: C
149:       else if ( 'EVAPORATION'.eq.FNAME ) then
150:           call RANU2( IRAND, RWK, 2*NS, ICOD )
151: C
152: C           ...for gamma distribution  $Pg(x,a) = x^{a-1} \cdot \exp(-x)$ 
153: C
154: C            $Pg(x,a+b) \rightarrow Pg(y,a), Pg(z,b) \rightarrow x = y + z$ 
155: C
156: *VOCL LOOP,NOVREC
157:       do 160 I = 1, NS
158:           X(I) = -DSRC(1)*LOG(RWK(I,1)*RWK(I,2))
159: 160       continue
160: C
161: C
162: C >>>> POWER( a bound1 bound2) [;]
163: C
164: C
165:       else if ( 'POWER'.eq.FNAME ) then
166:           call RANU2( IRAND, RWK, NS, ICOD )
167: C
168: C           ... simple inverse function method ...
169: C
170:       if ( DSRC(1).ne.-1.0 ) then
171:           DAA = DSRC(1) + 1.0D0
172:           DB1 = DSRC(2)**DAA
173:           DB2 = DSRC(3)**DAA - DB1
174: *VOCL LOOP,NOVREC
175:       do 170 I = 1, NS
176:           X(I) = (DB1+RWK(I,1)*DB2)**(1.0D0/DAA)
177: 170       continue
178:       else
179:           DB1 = LOG(DSRC(2))
180:           DB2 = LOG(DSRC(3)) - DB1
181: *VOCL LOOP,NOVREC
182:       do 180 I = 1, NS
183:           X(I) = EXP(DB1+RWK(I,1)*DB2)
184: 180       continue
185:       end if
186: C
187: C >>>> NORMALIZE( a )
188: C
189:       else if ( 'NORMALIZE'.eq.FNAME ) then
190:           DAA = DSRC(1)
191:           if ( DAA.eq.0.0D0 ) DAA = NS
192:           DSUM = 0.0D0
193: *VOCL LOOP,NOVREC
194:       do 190 I = 1, NS
195:           DSUM = DSUM + X(I)
196: 190       continue
197:       if ( DSUM.eq.0.0D0 ) then
198:           IERROR = 1
199:           write(IPR,*) 'SOURCE SAMPLING ERROR: SUM OF DATA IS ZERO',
200: &                     ' IN NORMALIZATION.'
201:           NERROR = NERROR + 1
202:       else
203:           DAA = DAA/DSUM
204: *VOCL LOOP,NOVREC
205:       do 200 I = 1, NS
206:           X(I) = DAA*X(I)
207: 200       continue
208:       end if
209:       end if
210: C
211: C
212:       return
213:       end

```

src/shared/smplx2.f

```

1:      subroutine SMPLX2( PROG, NS,   X,   Y,   FNAME, DSRC,
2:      &                 NDSRC, IRAND, RWK,  NRWK, IERROR, NERROR )
3: C==<MVP/GMVP>=====
4: C purpose:  source parameter sampling ( 2 parameter )
5: C
6: C   Samples from the following type of distributions:
7: C
8: C   DISC
9: C   CIRCLE
10: C  SWAP
11: C  ROTATE2D
12: C
13: C called in:  SYSSRC (source generation routine.)
14: C calls:  RANU2
15: C-----
16: C
17: C arguments (i=input, o=output, w=work, c=constant )
18: C
19: CLASS arg:  meanings
20: C
21: C i  prog : program name ('MVP'/'GMVP')
22: C i  ns  : number of source to be generated by this source set (input)
23: C i o X(*) : variable to be sampled
24: C i o Y(*) : variable to be sampled
25: C i  fname : sampling "function" name.
26: C i  data  : data for sampling.
27: C i  ndata : number of elements on data array.
28: C i  irand : seed of random numbers
29: C w  rwk(ns,nrwk): working array mainly used to save random numbers.
30: C i  nrwk : second dimension of rwk array.
31: C o  ierror : error flag ( 0 = no-error, 1 = error )
32: C i o nerror : number of errors
33: C
34: C =====
35:      implicit real*8(D)
36: C
37:      character*(*) PROG, FNAME
38: C
39:      real*8 X(NS), Y(NS), DSRC(NDSRC)
40: C
41:      real RWK(NS,NRWK)
42: C
43: C
44:      real*8 PAI, DPAI, HPAI
45:      parameter( PAI = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
46:      &          0.5D0*PAI )
47: C
48:      include 'INC/_IOUNIT'
49: C
50:      IERROR = 0
51: C
52: C
53: C
54: C >>>>  DISC( r1 r2 [ang1 ang2]) [:]
55: C
56: C
57:      if ( 'DISC'.eq.FNAME ) then
58:          call RANU2( IRAND, RWK, 2*NS, ICOD )
59: *VOCL LOOP,NOVREC
60:          do 100 I = 1, NS
61:              DR1 = DSRC(1)**2
62:              DR = SQRT(DR1+RWK(I,1)*(DSRC(2)**2-DR1))
63:              DQ = DSRC(3) + RWK(I,2)*(DSRC(4)-DSRC(3))
64:              X(I) = DR*COS(DQ)
65:              Y(I) = DR*SIN(DQ)
66:          100 continue
67: C
68: C
69: C >>>>  CIRCLE( r [ang1 ang2] ) [:]
70: C
71: C
72:      else if ( 'CIRCLE'.eq.FNAME ) then
73:          call RANU2( IRAND, RWK, NS, ICOD )
74:          if ( DSRC(1).ne.1.0D0 ) then
75: *VOCL LOOP,NOVREC
76:              do 110 I = 1, NS
77:                  DQ = DSRC(2) + (DSRC(3)-DSRC(2))*RWK(I,1)
78:                  X(I) = DSRC(1)*COS(DQ)
79:                  Y(I) = DSRC(1)*SIN(DQ)
80:              110 continue
81:          else
82: *VOCL LOOP,NOVREC
83:              do 120 I = 1, NS
84:                  DQ = DSRC(2) + (DSRC(3)-DSRC(2))*RWK(I,1)
85:                  X(I) = COS(DQ)
86:                  Y(I) = SIN(DQ)
87:              120 continue
88:          end if
89: C
90: C
91: C >>>>  SWAP( ) [:]
92: C
93: C
94:      else if ( 'SWAP'.eq.FNAME ) then
95: *VOCL LOOP,NOVREC
96:          do 130 I = 1, NS
97:              DQ = X(I)
98:              X(I) = Y(I)
99:              Y(I) = DQ
100:          130 continue
101: C
102: C
103: C >>>>  ROTATE2D( theta ) [:]
104: C
105: C
106:      else if ( 'ROTATE2D'.eq.FNAME ) then
107: C
108:          cos(theta) = dsrc(1j), sin(theta) = dsrc(1j+1)
109: C
110: *VOCL LOOP,NOVREC
111:          do 140 I = 1, NS
112:              DX = DSRC(1)*X(I) - DSRC(2)*Y(I)
113:              Y(I) = DSRC(2)*X(I) + DSRC(1)*Y(I)
114:              X(I) = DX
115:          140 continue
116:          end if
117: C
118: C
119:      return
120:      end

```

src/shared/smplx3.f

```

1:      subroutine SMPLX3( PROG, NS,   X,   Y,   Z,   FNAME, DSRC,
2:      &                 NDSRC, IRAND, RWK,  NRWK,  IERROR,NERROR )
3:      C==<MVP/GMVP>=====
4:      C purpose:  source parameter sampling ( 3 parameter )
5:      C
6:      C   Samples from the following type of distributions:
7:      C
8:      C   ISOTROPIC
9:      C   RESIZE
10:     C   ROTATE3D
11:     C   NEWAXIS
12:     C
13:     C called in:  SYSSRC (source generation routine.)
14:     C calls:  RANU2
15:     C-----
16:     C
17:     C arguments (i=input, o=output, w=work, c=constant )
18:     C
19:     CLASS arg:  meanings
20:     C
21:     C i  prog : program name ('MVP'/'GMVP')
22:     C i  ns  : number of source to be generated by this source set (input)
23:     C i o X(*) : variable to be sampled
24:     C i o Y(*) : variable to be sampled
25:     C i o Z(*) : variable to be sampled
26:     C i  fname : sampling "function" name.
27:     C i  data  : data for sampling.
28:     C i  ndata : number of ielement on data array.
29:     C i  irand : seed of random numbers
30:     C w  rwk(ns,nrwk): working array mainly used to save random numbers.
31:     C i  nrwk : second dimension of rwk array.
32:     C o  ierror : error flag ( 0 = no-error, 1 = error )
33:     C i o nerror : number of errors
34:     C
35:     C =====
36:     C   implicit real*8(D)
37:     C
38:     C   character*(*) PROG, FNAME
39:     C
40:     C   real*8 X(NS), Y(NS), Z(NS), DSRC(NDSRC)
41:     C
42:     C   real RWK(NS,NRWK)
43:     C
44:     C
45:     C   real*8 PAI, DPAI, HPAI
46:     C   parameter( PAI = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
47:     C   &         0.5D0*PAI )
48:     C
49:     C   include 'INC/_IOUNIT'
50:     C
51:     C   IERROR = 0
52:     C
53:     C
54:     C
55:     C >>>>  SPHERE( r1 r2 [mu1 mu2] ) [;]
56:     C
57:     C
58:     C
59:     C   if ( 'SPHERE'.eq.FNAME ) then
60:     C       if ( DSRC(1).ne.DSRC(2) ) then
61:     C           call RANU2( IRAND, RWK, 3*NS, ICOD )
62:     C       *VOCL LOOP,NOVREC
63:     C           do 100 I = 1, NS
64:     C               DR1 = DSRC(1)**3
65:     C               DR2 = DSRC(2)**3 - DR1

```

```

66:     C               DRR = (DR1+DR2*RWK(I,1))**(1D0/3D0)
67:     C               DMU = DSRC(3) + (DSRC(4)-DSRC(3))*RWK(I,2)
68:     C               X(I) = DRR*DMU
69:     C               DSN = DRR*SQRT(1.0D0-DMU*DMU)
70:     C               DFI = DPAI*RWK(I,3)
71:     C               Y(I) = DSN*COS(DFI)
72:     C               Z(I) = DSN*SIN(DFI)
73:     C       100      continue
74:     C       else
75:     C           call RANU2( IRAND, RWK, 2*NS, ICOD )
76:     C       *VOCL LOOP,NOVREC
77:     C           do 110 I = 1, NS
78:     C               DMU = DSRC(3) + (DSRC(4)-DSRC(3))*RWK(I,1)
79:     C               X(I) = DSRC(1)*DMU
80:     C               DSN = SQRT(1.0D0-DMU*DMU)
81:     C               DFI = DPAI*RWK(I,2)
82:     C               Y(I) = DSRC(1)*DSN*COS(DFI)
83:     C               Z(I) = DSRC(1)*DSN*SIN(DFI)
84:     C       110      continue
85:     C       end if
86:     C
87:     C
88:     C
89:     C >>>>  ISOTROPIC( [mul [mu2]] ) [;]
90:     C
91:     C
92:     C
93:     C   else if ( 'ISOTROPIC'.eq.FNAME ) then
94:     C       call RANU2( IRAND, RWK, 2*NS, ICOD )
95:     C       *VOCL LOOP,NOVREC
96:     C       do 120 I = 1, NS
97:     C           DMU = DSRC(1) + (DSRC(2)-DSRC(1))*RWK(I,1)
98:     C           X(I) = DMU
99:     C           DSN = SQRT(1.0D0-DMU*DMU)
100:     C           DFI = DPAI*RWK(I,2)
101:     C           Y(I) = DSN*COS(DFI)
102:     C           Z(I) = DSN*SIN(DFI)
103:     C       120      continue
104:     C
105:     C
106:     C
107:     C >>>>  RESIZE( norm ) [;]
108:     C
109:     C
110:     C
111:     C   else if ( 'RESIZE'.eq.FNAME ) then
112:     C       *VOCL LOOP,NOVREC
113:     C       do 130 I = 1, NS
114:     C           DSN = SQRT(X(I)**2+Y(I)**2+Z(I)**2)
115:     C           if ( DSN.ne.0.0 ) then
116:     C               DSN = DSRC(1)/DSN
117:     C               X(I) = X(I)*DSN
118:     C               Y(I) = Y(I)*DSN
119:     C               Z(I) = Z(I)*DSN
120:     C           end if
121:     C       130      continue
122:     C
123:     C
124:     C
125:     C >>>>  ROTATED3D( theta ax ay az ) [;]
126:     C
127:     C           cos(theta),sin(theta),ax,ay,az (ax**2+ay**2+az**2=1)
128:     C
129:     C
130:     C   else if ( 'ROTATE3D'.eq.FNAME ) then

```


src/shared/smplx3.f

```

131: *VOCL LOOP,NOVREC
132:   do 140 I = 1, NS
133:     DAX   = DSRC(3)
134:     DAY   = DSRC(4)
135:     DAZ   = DSRC(5)
136: C
137: C     dc : length of projection of (x,y,z) to (ax,ay,az)
138: C
139: C     DC   = DAX*X(I) + DAY*Y(I) + DAZ*Z(I)
140: C
141: C     dp : (x,y,z) - projection vector to (ax,ay,az)
142: C
143: C     DPX   = X(I) - DC*DAX
144: C     DPY   = Y(I) - DC*DAY
145: C     DPZ   = Z(I) - DC*DAZ
146: C
147: C     length of original vector (for size correction)
148: C
149: C     DLV   = SQRT(X(I)**2+Y(I)**2+Z(I)**2)
150: C
151: C     rotate vector dp by theta radian round axis (ax,ay,az)
152: C
153: C     dp' = dp * cos(theta) + ( (ax,ay,az) x dp ) * sin(theta)
154: C
155: C     (x',y',z') = dc*(ax,ay,az) + dp'
156: C
157: C     X(I)   = DC*DAX + DPX*DSRC(1) + (DAY*DPZ-DAZ*DPY)*DSRC(2)
158: C     Y(I)   = DC*DAY + DPY*DSRC(1) + (DAZ*DPX-DAX*DPZ)*DSRC(2)
159: C     Z(I)   = DC*DAZ + DPZ*DSRC(1) + (DAX*DPY-DAY*DPX)*DSRC(2)
160: C
161: C     length of translated vector (for size correction)
162: C
163: C     DLV2   = SQRT(X(I)**2+Y(I)**2+Z(I)**2)
164: C
165: C     vector size correction
166: C
167: C     if ( DLV2.ne.0.0D0 ) then
168: C       X(I)   = X(I)*(DLV/DLV2)
169: C       Y(I)   = Y(I)*(DLV/DLV2)
170: C       Z(I)   = Z(I)*(DLV/DLV2)
171: C     end if
172:   140 continue
173: C
174: C
175: C
176: C >>>> NEWAXIS( ax1 ay1 az1  ax2 ay2 az2 )
177: C
178: C
179: C
180: C     else if ( 'NEWAXIS'.eq.FNAME ) then
181: C
182: C       (x',y',z') = x*(ax1,ay1,az1)+y*(ax2,ay2,az2)+z*(ax3,ay3,az3)
183: C
184: *VOCL LOOP,NOVREC
185:   do 150 I = 1, NS
186: C
187: C     DPX   = X(I)
188: C     DPY   = Y(I)
189: C     DPZ   = Z(I)
190: C
191: C     X(I)   = DPX*DSRC(1) + DPY*DSRC(4) + DPZ*DSRC(7)
192: C     Y(I)   = DPX*DSRC(2) + DPY*DSRC(5) + DPZ*DSRC(8)
193: C     Z(I)   = DPX*DSRC(3) + DPY*DSRC(6) + DPZ*DSRC(9)
194: C
195:   150 continue
196:   end if
197: C
198: C
199:   return
200:   end

```

src/shared/smpstg.f

```

1:      subroutine SMPSTG( IOW,  MLT,  X,    Y,    Z,    AI,    BI,
2:      &                  CI,  DI,  ISTG, IBP,  NN,  PNND, KNND,
3:      &                  NPNND, IPLAT, KLATT, KSLAT, SZLAT, NLATT, CELSZ,
4:      &                  IPCEL, PPPF, KPPF, NPPPF,
5:      &                  IRAND, LEVL, DBNK, KDBNK, MDBNK, IBNK, KIBNK,
6:      &                  MIBNK, NBANK, XSS, YSS, ZSS, IKL, IWK1, DWK1,
7:      &                  R    )
8: C=====
9: C  PURPOSE:  sample STG sphere positions
10: C  CALLED IN:  FLIONE
11: C  CALLS:  RANU2
12: C-----
13: C Arguments (i=input, o=output, w=work)
14: C i IOW: message printout I/O unit
15: C i MLT: lattice # (STG region defined as a lattice region)
16: C i LBZ: cell buffer zone # of STG cell.
17: C i X(NN),Y(NN),Z(NN): particle position
18: C i AI(NN),BI(NN),CI(NN): particle flight direction
19: C i o DI(NN): distance value. When sampled distance is less than this,
20: C      particle may enter STG cell, and STG position is sampled.
21: C o ISTG(NN) : set lattice buffer region of STG cell(LBZ) if sampled
22: C      distance is less than DI(*), else zero.
23: C o IKL(NN) : set cell position in STG lattice if sampled
24: C      distance is less than DI(*), else zero.
25: C i IBP(I) : bank pointer of particles
26: C i NN : number of particles
27: C i PNND(NPNND) : Nearest Neighbour Distribution data (for alias method
28: C      sampling)
29: C i KNND(NPNND) : Nearest Neighbour Distribution alias (for alias method
30: C      sampling)
31: C i NPNND : length of array PNND, KNND
32: C i PPPF(NPPPF) : Partial Packing Fraction data for alias method
33: C i KPPF(NPPPF) : Partial Packing Fraction data for alias method
34: C i NFFFF : length of array PPPF, KPPF
35: C i IPLAT(NLATT) : index to array KLATT & KSLAT for lattices.
36: C i KLATT(*) : STG region attributes (1) STG cell ID, matrix mat ID
37: C      (this is cell-ID's for ordinary lattice)
38: C i KSLAT(*) : STG region attributes (2) pointers to PNND/KNND
39: C      (this is cell-ID's for ordinary lattice)
40: C i SZLAT(8,NLATT) : STG packing fraction, radius, NND intervals
41: C i CELSZ(6,NCELL) : STG cell size (1-3)
42: C i NLATT : number of lattices
43: C i o IRAND : seed of random numbers
44: C i LEVL(NBANK) : current lattice level
45: C i o DBNK(NBANK,*) : optional bank parameters in floating point
46: C i KDBNK(0:MDBNK) : entry #'s of bank parameters in DBNK array.
47: C i o IBNK(NBANK,*) : optional bank parameters in integer
48: C i KIBNK(0:MDBNK) : entry #'s of bank parameters in IBNK array.
49: C
50: C i o XSS(NN),YSS(NN),ZSS(NN) : repalced with translated coordinates
51: C      Y(NN),Y(NN),Z(NN) in coordinate system of STG sphereC
52: C      if calculated distance is less than DI(I).
53: C      ( No need to change distance curtrently!!)
54: C
55: C w IWK1(NN) : work area
56: C w DWK1(NN) : work area (double)
57: C w R(3*NN) : store random number
58: C=====
59:      implicit real*8(D)
60: C
61:      real*8 X(NN), Y(NN), Z(NN)
62:      real*8 AI(NN), BI(NN), CI(NN)
63:      real*8 DI(NN)
64:      integer IBP(NN)
65:      integer ISTG(NN), IKL(NN)

```

```

66: C
67: C .... NND .....
68: C
69:      real PNND(NPNND), PPPF(NPPPF)
70:      integer KNND(NPNND), KPPF(NPPPF)
71: C
72: C .... GEOMETRY .....
73: C
74:      integer IPLAT(NLATT+1), KLATT(*), KSLAT(*), IPCEL(*)
75:      real*8 SZLAT(8,NLATT)
76:      real*8 CELSZ(6,*)
77: C
78: C .... BANK .....
79: C
80:      integer LEVL(NBANK)
81:      real*8 DBNK(NBANK,*)
82:      integer KDBNK(0:MDBNK)
83:      integer IBNK(NBANK,*)
84:      integer KIBNK(0:MIBNK)
85: C
86: C ... translated coordinates
87: C
88:      real*8 XSS(NN), YSS(NN), ZSS(NN)
89: C
90: C .... work .....
91: C
92:      integer IWK1(NN)
93:      real*8 DWK1(NN)
94:      real R(3*NN)
95: C
96: C ... constant
97: C
98:      real*8 PI2
99:      parameter( PI2 = 2*3.14159265358979D0 )
100:      real*8 SMALL
101:      parameter( SMALL = 1.0D-35 )
102: C
103: C-----
104: C
105:      K7 = KIBNK(7)
106: C
107: C ... SZLAT(1,*) : packing ratio
108: C ... SZLAT(2,*) : STG radius
109: C ... SZLAT(3:5,*) : NND mesh rerativel to diameter
110: C
111:      DRP = SZLAT(2,MLT)
112:      DIAM = 2.0*DRP
113:      DFP = SZLAT(1,MLT)
114:      DN1 = SZLAT(3,MLT) * DIAM
115:      DN2 = SZLAT(4,MLT) * DIAM
116:      DN3 = SZLAT(5,MLT) * DIAM
117:      DXFP = -2.0D0/3.0D0*(1.0D0-DFP) /DFP * DIAM
118:      NSTG = KLATT(IPLAT(MLT)+9)
119:      JN1 = KLATT(IPLAT(MLT)+10)
120:      JN2 = KLATT(IPLAT(MLT)+11)
121:      JN3 = KLATT(IPLAT(MLT)+12)
122:      LN1 = KSLAT(IPLAT(MLT)+10)
123:      LN2 = KSLAT(IPLAT(MLT)+11)
124:      LN3 = KSLAT(IPLAT(MLT)+12)
125: C
126:      call RANU2( IRAND, R, 3*NN, ICON )
127: C
128:      KNN = 0
129:      *VOCL LOOP,NOVREC
130:      do 100 I = 1, NN

```

src/shared/smpstg.f

```

131:      IB      = IBNK(IBP(I),K7)
132: C
133: C      ... NND #1
134: C
135: C      if ( IB.eq.1 ) then
136: C
137: C      ... Theoretical distribution ...
138: C      if ( JN1.eq.0 ) then
139: C          DWK1(I) = DXFP*LOG(R(I))+SMALL)
140: C      else
141: C          K      = MIN(JN1-1,INT(JN1*R(I)))
142: C          if ( R(NN+I).lt.PNND(LN1+K) ) K = KNND(LN1+K) - 1
143: C          DWK1(I) = DN1*(REAL(K)+R(2*NN+I))
144: C      end if
145: C
146: C      ... NND #2
147: C
148: C      else if ( IB.eq.2 ) then
149: C
150: C      ... Theoretical distribution ...
151: C      if ( JN2.eq.0 ) then
152: C          DWK1(I) = DXFP*LOG(R(I))+SMALL)
153: C      else
154: C          K      = MIN(JN2-1,INT(JN2*R(I)))
155: C          if ( R(NN+I).lt.PNND(LN2+K) ) K = KNND(LN2+K) - 1
156: C          DWK1(I) = DN2*(REAL(K)+R(2*NN+I))
157: C      end if
158: C
159: C      ... NND #3
160: C
161: C      else if ( IB.eq.3 ) then
162: C
163: C      ... Theoretical distribution ...
164: C      if ( JN3.eq.0 ) then
165: C          DWK1(I) = DXFP*LOG(R(I))+SMALL)
166: C      else
167: C          K      = MIN(JN3-1,INT(JN3*R(I)))
168: C          if ( R(NN+I).lt.PNND(LN3+K) ) K = KNND(LN3+K) - 1
169: C          DWK1(I) = DN3*(REAL(K)+R(2*NN+I))
170: C      end if
171: C
172: C      ... no NND sampling required.
173: C
174: C      else
175: C          KNN      = KNN + 1
176: C      end if
177: C
178: C      ISTG(I) = 0
179: C      IKL(I)  = 0
180: C      100 continue
181: C
182: C      KNN      = NN - KNN
183: C
184: C      ===== Sample center position of STG particle here =====
185: C      (this may be calculated after determined whether partilces
186: C      enter STG particle or not ...)
187: C
188: C      if ( KNN.gt.0 ) then
189: C      check
190: C      call RANU2( IRAND, R, 2*NN, ICON )
191: C      NS      = 0
192: C      do 110 I = 1, NN
193: C          IB      = IBNK(IBP(I),K7)
194: C          if ( IB.ne.0.and.DWK1(I).lt.DI(I) ) then
195: C              NS      = NS + 1

```

```

196: C          IWK1(NS) = I
197: C          DI(I)    = DWK1(I)
198: C      end if
199: C      110 continue
200: C      end if
201: C
202: C      if ( NS.gt.0 ) then
203: C          if ( NSTG.eq.1 ) then
204: C
205: C          *VOCL LOOP,NOVREC
206: C          do 120 J = 1, NS
207: C              IKL(IWK1(J)) = 1
208: C          120 continue
209: C          else
210: C              call RANU2( IRAND, R, 2*NS, ICON )
211: C              LPPF      = KLATT(IPLAT(MLT)+13)
212: C          *VOCL LOOP,NOVREC
213: C          do 130 J = 1, NS
214: C              K      = MIN(NSTG-1, INT(NSTG*R(J)))
215: C              if ( R(NS+J).lt.PPPF(LPPF+K) ) K = KPPF(LPPF+K) - 1
216: C              IKL(IWK1(J)) = K + 1
217: C          130 continue
218: C          end if
219: C
220: C          call RANU2( IRAND, R, 2*NS, ICON )
221: C          K5      = KDBNK(5)
222: C          *VOCL LOOP,NOVREC
223: C          do 140 J = 1, NS
224: C              I      = IWK1(J)
225: C
226: C          ... center position of STG cell.
227: C
228: C          DMU      = SQRT(R(I))
229: C          DETA      = PI2*R(NN+I)
230: C          DMU      = SQRT(R(J))
231: C          DETA      = PI2*R(NS+J)
232: C          DX      = AI(I)
233: C          DY      = BI(I)
234: C          DZ      = CI(I)
235: C          DSINT     = SQRT(1.0D0-DMU*DMU)
236: C          DST      = 1.0D0 - DX*DX
237: C          if ( DST.gt.0.0D0 ) then
238: C              DST = SQRT(DST)
239: C              DCOSFI = DY/DST
240: C              DSINFI = DZ/DST
241: C          else
242: C              DST = 0.0D0
243: C              DCOSFI = 1.0D0
244: C              DSINFI = 0.0D0
245: C          end if
246: C
247: C          DSINE     = SIN(DETA)
248: C          DCOSE     = COS(DETA)
249: C
250: C          DY = DY*DMU + (DX*DCOSFI*DCOSE-DSINFI*DSINE)*DSINT
251: C          DZ = DZ*DMU + (DX*DSINFI*DCOSE+DCOSFI*DSINE)*DSINT
252: C          DX = DX*DMU - DCOSE*DSINT*DST
253: C          LV      = LEVL(IBP(I))
254: C          K1      = K5 + (LV-1)*3
255: C          DXS      = AI(I)*DWK1(I) + DRP*DX
256: C          DYS      = BI(I)*DWK1(I) + DRP*DY
257: C          DZS      = CI(I)*DWK1(I) + DRP*DZ
258: C
259: C          DBNK(IBP(I),K1) = X(I) + DXS
260: C          DBNK(IBP(I),K1+1) = Y(I) + DYS

```

src/shared/smpstg.f

```
261:          DBNK( IBP(I),K1+2)  = Z(I)  + DZS
262: C
263:          ICEL      = KLATT( IPLAT(MLT)+IKL(I) )
264:          XSS(I)    = CELSZ(1,ICEL) - DXS
265:          YSS(I)    = CELSZ(2,ICEL) - DYS
266:          ZSS(I)    = CELSZ(3,ICEL) - DZS
267:          ISTG(I)   = IPCEL(ICEL)
268: 140      continue
269:      end if
270: C
271:      return
272:      end
```

SAFE

src/shared/spcchk.f

```
1:      subroutine SPCCHK( MODE,  NSPACE,ISPACE,ISPNM, NEST,  LV,    ICSP,
2:      &                  NSF,   ISP,   LAST,  LIMIT )
3: C=====
4: C PUROPSE: REGISTER A NEW SUBSPACE TO SUBSPACE-NAME LIST 'ISPNM'
5: C          IF NECESSARY.
6: C          RETURN NEW ( OR ALREADY EXSITING ) SPACE # IN 'ISPACE'.
7: C CALLED IN: SPNUMS
8: C-----
9: C arguments (i=input, o=output, w=work)
10: C
11: C i MODE : operation mode ( 'ADD' : add a newspace, other mode only
12: C          checks existence of space )
13: C io NSPACE : number of space
14: C o ISPACE : space # of added or inquired space.
15: C io ISPNM : pointers to subspace name list
16: C i NEST : lattice geometry nest depth
17: C i LV : current nesting level for added/inquired space
18: C i ICSP : UPPER LEVEL SUBSPACE #
19: C i NSF : FRAME #
20: C i ISP : SUBFRAME #
21: C io LAST : (add mode) last index of dynamic memory array, to check
22: C          memory overflow for ISPNM. (dirty, dirty, dirty !!!)
23: C i LIMIT : memory size limit
24: C=====
25:      character*(*) MODE
26:      integer ISPNM(2,0:NEST,*)
27: C
28:      do 110 I = 1, NSPACE
29:          if ( I.eq.ICSP .or. ISPNM(1,0,I).ne.LV ) go to 110
30: C
31:          if ( LV.gt.1 ) then
32:              do 100 K = 1, LV - 1
33:                  if ( ISPNM(1,K,I).ne.ISPNM(1,K,ICSP)
34:                  &      .or. ISPNM(2,K,I).ne.ISPNM(2,K,ICSP) ) go to 110
35:              100      continue
36:          end if
37: C
38:          if ( ISPNM(1,LV,I).eq.NSF.and.ISPNM(2,LV,I).eq.ISP ) then
39:              ISPACE = I
40:              return
41:          end if
42:      110 continue
43: C
44:      if ( MODE.ne.'ADD' ) then
45:          ISPACE = 0
46:          return
47:      end if
48: C
49: C
50:      NSPACE = NSPACE + 1
51:      ISPACE = NSPACE
52: C
53: C ..... INCREMENT 'LAST' POINTER FOR 'ISPNM(1:2,0:NEST,ISPACE)' ....
54: C
55:      LAST = LAST + 2*(NEST+1)
56: C
57:      if ( LSIZ(LAST).gt.LIMIT ) then
58:          call MEMERR( 'SPADD', ' PREPARATION FOR SUBSPACE NAME LIST. ',
59:          &      LSIZ(LAST), LIMIT )
60:          stop 999
61:      end if
62: C
63: C
64:      ISPNM(1,0,NSPACE) = LV
65:      do 120 K = 1, LV - 1
```

```
66:          ISPNM(1,K,NSPACE) = ISPNM(1,K,ICSP)
67:          ISPNM(2,K,NSPACE) = ISPNM(2,K,ICSP)
68:      120 continue
69:      ISPNM(1,K,NSPACE) = NSF
70:      ISPNM(2,K,NSPACE) = ISP
71:      do 130 K = LV + 1, NEST
72:          ISPNM(1,K,NSPACE) = 0
73:          ISPNM(2,K,NSPACE) = 0
74:      130 continue
75:      return
76:      end
```

src/shared/spear1.f

```

1:      subroutine SPEAR1( MZONE, KZDA,  KZAA,  SDA,  NN,  NBANK, JSS,
2:      &                  X,    Y,    Z,    AI, BI,  CI,  DI,
3:      &                  DLOC1, DLOC2, IKL,  IKL2, DSAVE, DINF, DEPS
4:      &                  )
5: C=====
6: C PURPOSE:  CALCULATION OF DISTANCES TO A ZONE BOUDARY.
7: C
8: C  Accept NN particles whose coordinate & direction data are stored
9: C  in arrays X,Y,Z, AI,BI  and CI sequentially.
10: C  Calculate distances to zone 'MZONE' and store them in array DI(*).
11: C
12: C  Array DI(I) should be initialized to DINF or other default value
13: C  in the calling routine, or it may not be possible to check a valid
14: C  distance is calculated or not.
15: C
16: C CALLED IN:  FLIONE,LATICE
17: C CALLS:     SOSQL1
18: C-----
19: C Arguments (i=input, o=output, w=work)
20: C i MZONE: zone #
21: C i KZDA, KZAA, SDA: zone geometry data (see description attached to
22: C   common /PGEOM/ ety.)
23: C i NN : number of particles
24: C i NBANK : size of arrays X,Y,Z,AI,BI,CI,DI ...
25: C i JSS : when non-zero, IKL2(*) must be set correctly as well for
26: C   bodies other than torus.
27: C i X(NBANK), Y(NBANK), Z(NBANK) : coordinates of particles
28: C i AI(NBANK), BI(NBANK), CI(NBANK) : direction vector of particles
29: C i DI(NBANK) : calculated distance to zone boundary
30: C w DLOC1(NBANK), DLOC2(NBANK) : working array used to keep intermediate
31: C   distance calculation result when "outside of a body" is used
32: C   in the definition of the zone.
33: C w DSAVE(NBANK) : working area used only when JSS.ne.0
34: C
35: C
36: C i IKL(NBANK) : surface pointer (to SDA) on which particle is
37: C   currently placed, if non-zero.
38: C   Used only when geometry definition includes torus.
39: C   Bank data KLSF(*) of each particle should be stored
40: C   in calling routine.
41: C
42: C o IKL2(NBANK) : surface pointer (to SDA) on which particle will be
43: C   placed if the calculated distance to a torus is
44: C   selected as smallest one in body list.
45: C
46: C   When JSS.ne.0, surfaces other than torus is memorized
47: C   as well, but for bodies used with minus sign (outer side)
48: C   the surface index may be for the last surface of the body,
49: C   so IKL2 data for such a case can be used only for
50: C   body detection and not for surface detection.
51: C
52: C i DINF : "infinite" distance. If no valid distance is calculated,
53: C   DI(*) is set to DINF.
54: C i DEPS : "minimum" distance.
55: C-----
56: C=====
57:      implicit real*8(D)
58: C
59: C .... GEOMETRY .....
60: C
61:      real*8 SDA(*)
62:      integer KZDA(2,*), KZAA(*)
63: C
64: C ... coordinates, direction & distance .....
65: C

```

```

66:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
67:      real*8 AI(NBANK), BI(NBANK), CI(NBANK)
68:      real*8 DI(NBANK)
69: C
70: C .... Surface pointers used for torus...
71: C
72:      integer IKL(NBANK), IKL2(NBANK)
73: C
74: C .... Working area .....
75: C
76:      real*8 DLOC1(NBANK), DLOC2(NBANK)
77:      real*8 DSAVE(NBANK)
78: C
79: C
80:      real*8 DSMALL
81:      parameter( DSMALL = 1.0D-7 )
82: C
83: C-----
84: C
85:      DEPS1 = DEPS*1.00001D0
86:      DEPS2 = (DEPS1/2.D0)**2
87: C
88:      do 100 I = 1, NN
89: CCCCC  DI(I) = DINF
90:         DLOC1(I) = -DINF
91:         DLOC2(I) = DINF
92:      100 continue
93: C
94:      if ( JSS.ne.0 ) then
95:         do 110 I = 1, NN
96:            DSAVE(I) = DI(I)
97:            IKL2(I) = 0
98:         110 continue
99:      end if
100: C
101: C .....
102: C
103:      do 220 N = KZAA(MZONE), KZAA(MZONE+1) - 1
104:         LLG = KZDA(2,N)
105:         if ( LLG.lt.0 ) go to 220
106: C
107:         JS = SIGN(1,KZDA(1,N))
108:         LS1 = ABS(KZDA(1,N))
109:         KSF = INT(SDA(LS1))
110: C
111: C ..... JS : + OR - SIGN INDICATES ON WHICH SIDE OF THE SURFACE
112: C           THE PARTICLES EXIST. ( +/- = INSIDE/OUTSIDE )
113: C           LS : HEAD POSITION OF SURFACE DATA IN SDA ARRAY
114: C           KSF: KIND OF SURFACE (1/2/3 = SLAB/SPHERE/CYLINDER)
115: C
116:         LS = LS1 + 1
117: C
118: C===== < INFORMATION > =====
119: C THE FOLLOWING CALCULATIONS USING 'JS' MIGHT LOOK QUEER AT FIRST
120: C SIGHT. THE PURPOSE OF THEM IS TO PICK UP THE LARGER VALUE OF TWO
121: C DISTANCES IF 'JS' IS POSITIVE AND PICK UP SMALLER ONE IF 'JS' IS
122: C NEGATIVE.
123: C=====
124: C
125: C ..... SLAB .....
126: C
127:      if ( KSF.eq.1 ) then
128:         if ( LLG.eq.0 ) then
129:            if ( SDA(LS+1).eq.0.0d0.and.SDA(LS+2).eq.0.0d0 ) then
130:               do 111 I = 1, NN

```

src/shared/spear1.f

```

131: C          DUP      = -X(I)
132: C          DUV      = AI(I)*JS
133: C*VOCL STMT,IF(100)
134: C          if ( DUV.ne.0.0D0 ) then
135: C              D1      = JS*
136: C              &          MAX((SDA(LS+3)+DUP)/DUV,(SDA(LS+4)
137: C              &          +DUP)/DUV)
138: C              if ( D1.ge.0.0D0 ) DI(I)      = MIN(DI(I),D1)
139: C              end if
140: C 111          continue
141: C          else if( SDA(LS+2).eq.0.0d0.and.SDA(LS).eq.0.0d0 ) then
142: C
143: C              do 112 I = 1, NN
144: C                  DUP      = -Y(I)
145: C                  DUV      = BI(I)*JS
146: C*VOCL STMT,IF(100)
147: C          if ( DUV.ne.0.0D0 ) then
148: C              D1      = JS*
149: C              &          MAX((SDA(LS+3)+DUP)/DUV,(SDA(LS+4)
150: C              &          +DUP)/DUV)
151: C              if ( D1.ge.0.0D0 ) DI(I)      = MIN(DI(I),D1)
152: C              end if
153: C 112          continue
154: C          else if( SDA(LS).eq.0.0d0.and.SDA(LS+1).eq.0.0d0 ) then
155: C              do 113 I = 1, NN
156: C                  DUP      = -Z(I)
157: C                  DUV      = CI(I)*JS
158: C*VOCL STMT,IF(100)
159: C          if ( DUV.ne.0.0D0 ) then
160: C              D1      = JS*
161: C              &          MAX((SDA(LS+3)+DUP)/DUV,(SDA(LS+4)
162: C              &          +DUP)/DUV)
163: C              if ( D1.ge.0.0D0 ) DI(I)      = MIN(DI(I),D1)
164: C              end if
165: C 113          continue
166: C          else
167: C              do 120 I = 1, NN
168: C                  DUP      =
169: C                  &          -(SDA(LS)*X(I)+SDA(LS+1)*Y(I)+SDA(LS+2)*Z(I))
170: C                  DUV      =
171: C                  &          (SDA(LS)*AI(I)+SDA(LS+1)*BI(I)+SDA(LS+2)*CI(I)
172: C                  &          )*JS
173: C*VOCL STMT,IF(100)
174: C          if ( DUV.ne.0.0D0 ) then
175: C              D1      = JS*
176: C              &          MAX((SDA(LS+3)+DUP)/DUV,(SDA(LS+4)
177: C              &          +DUP)/DUV)
178: C          CCCCCC          if ( D1.ge.0.0D0 ) DI(I)      = MIN(DI(I),D1)
179: C              if ( JS.gt.0.or.D1.ge.0.0D0 ) DI(I) = MIN(DI(I),D1)
180: C              end if
181: C 120          continue
182: C          end if
183: C          else
184: C              if ( SDA(LS+1).eq.0.0d0.and.SDA(LS+2).eq.0.0d0 ) then
185: C                  do 121 I = 1, NN
186: C                      DUP      = -X(I)
187: C                      DUV      = AI(I)
188: C*VOCL STMT,IF(100)
189: C              if ( DUV.ne.0.0D0 ) then
190: C                  D1      = (SDA(LS+3)+DUP) /DUV
191: C                  D2      = (SDA(LS+4)+DUP) /DUV
192: C                  DLOC1(I) = MAX(DLOC1(I),MIN(D1,D2))
193: C                  DLOC2(I) = MIN(DLOC2(I),MAX(D1,D2))
194: C*VOCL STMT,IF(0)
195: C              else

```

```

196: C              if ( (SDA(LS+3)+DUP)*(SDA(LS+4)+DUP).ge.0.0D0 )
197: C              &          then
198: C                  DLOC1(I) = DINF
199: C                  DLOC2(I) = -DINF
200: C              end if
201: C          end if
202: C          if ( LLG.eq.2 ) then
203: C              if ( DLOC1(I)+DEPS1.lt.DLOC2(I)
204: C              &          .and.DLOC1(I).ge.0.0D0 ) then
205: C                  DI(I) = MIN(DI(I),DLOC1(I))
206: C              end if
207: C              DLOC1(I) = -DINF
208: C              DLOC2(I) = DINF
209: C          end if
210: C 121          continue
211: C          else if( SDA(LS+2).eq.0.0d0.and.SDA(LS).eq.0.0d0 ) then
212: C              do 122 I = 1, NN
213: C                  DUP      = -Y(I)
214: C                  DUV      = BI(I)
215: C*VOCL STMT,IF(100)
216: C              if ( DUV.ne.0.0D0 ) then
217: C                  D1      = (SDA(LS+3)+DUP) /DUV
218: C                  D2      = (SDA(LS+4)+DUP) /DUV
219: C                  DLOC1(I) = MAX(DLOC1(I),MIN(D1,D2))
220: C                  DLOC2(I) = MIN(DLOC2(I),MAX(D1,D2))
221: C*VOCL STMT,IF(0)
222: C              else
223: C                  if ( (SDA(LS+3)+DUP)*(SDA(LS+4)+DUP).ge.0.0D0 )
224: C                  &          then
225: C                      DLOC1(I) = DINF
226: C                      DLOC2(I) = -DINF
227: C                  end if
228: C              end if
229: C              if ( LLG.eq.2 ) then
230: C                  if ( DLOC1(I)+DEPS1.lt.DLOC2(I)
231: C                  &          .and.DLOC1(I).ge.0.0D0 ) then
232: C                      DI(I) = MIN(DI(I),DLOC1(I))
233: C                  end if
234: C                  DLOC1(I) = -DINF
235: C                  DLOC2(I) = DINF
236: C              end if
237: C 122          continue
238: C          else if( SDA(LS).eq.0.0d0.and.SDA(LS+1).eq.0.0d0 ) then
239: C              do 123 I = 1, NN
240: C                  DUP      = -Z(I)
241: C                  DUV      = CI(I)
242: C*VOCL STMT,IF(100)
243: C              if ( DUV.ne.0.0D0 ) then
244: C                  D1      = (SDA(LS+3)+DUP) /DUV
245: C                  D2      = (SDA(LS+4)+DUP) /DUV
246: C                  DLOC1(I) = MAX(DLOC1(I),MIN(D1,D2))
247: C                  DLOC2(I) = MIN(DLOC2(I),MAX(D1,D2))
248: C*VOCL STMT,IF(0)
249: C              else
250: C                  if ( (SDA(LS+3)+DUP)*(SDA(LS+4)+DUP).ge.0.0D0 )
251: C                  &          then
252: C                      DLOC1(I) = DINF
253: C                      DLOC2(I) = -DINF
254: C                  end if
255: C              end if
256: C              if ( LLG.eq.2 ) then
257: C                  if ( DLOC1(I)+DEPS1.lt.DLOC2(I)
258: C                  &          .and.DLOC1(I).ge.0.0D0 ) then
259: C                      DI(I) = MIN(DI(I),DLOC1(I))
260: C                  end if

```

src/shared/spear1.f

```

261: C          DLOC1(I) = -DINF
262: C          DLOC2(I) = DINF
263: C          end if
264: C 123      continue
265: C          else
266:
267:          do 130 I = 1, NN
268:            DUP =
269:            &      -(SDA(LS)*X(I)+SDA(LS+1)*Y(I)+SDA(LS+2)*Z(I))
270:            DUV = SDA(LS)*AI(I) + SDA(LS+1)*BI(I) + SDA(LS+2)*
271:            &      CI(I)
272: *VOCL STMT,IF(100)
273:          if ( DUV.ne.0.0D0 ) then
274:            D1 = (SDA(LS+3)+DUP) /DUV
275:            D2 = (SDA(LS+4)+DUP) /DUV
276:            DLOC1(I) = MAX(DLOC1(I),MIN(D1,D2))
277:            DLOC2(I) = MIN(DLOC2(I),MAX(D1,D2))
278: *VOCL STMT,IF(0)
279:          else
280:            if ( (SDA(LS+3)+DUP)*(SDA(LS+4)+DUP).ge.0.0D0 )
281:            &      then
282:              DLOC1(I) = DINF
283:              DLOC2(I) = -DINF
284:            end if
285:          end if
286:          if ( LLG.eq.2 ) then
287:            if ( DLOC1(I)+DEPS1.lt.DLOC2(I)
288:            &      .and.DLOC1(I).ge.0.0D0 ) then
289:              DI(I) = MIN(DI(I),DLOC1(I))
290:            end if
291:            DLOC1(I) = -DINF
292:            DLOC2(I) = DINF
293:          end if
294: 130      continue
295:
296: C          end if
297: C          end if
298: C
299: C ..... SPHERE .....
300: C
301: C          else if ( KSF.eq.2 ) then
302: C            if ( LLG.eq.0 ) then
303: C              do 140 I = 1, NN
304: C                DPCX = X(I) - SDA(LS)
305: C                DPCY = Y(I) - SDA(LS+1)
306: C                DPCZ = Z(I) - SDA(LS+2)
307: C                DVPC = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
308: C ... if ( LLG.eq.0 ) then
309: C          DD = DVPC*DVPC + SDA(LS+3) - DPCX*DPCX
310: C          &      - DPCY*DPCY - DPCZ*DPCZ
311: C          if ( DD.gt.DEPS2 .or. (DD.ge.0.0.and.JS.gt.0) ) then
312: C            D1 = -DVPC + JS*SQRT(DD)
313: C CCCCCC      if ( D1.ge.0.0 ) DI(I) = MIN(DI(I),D1)
314: C            if ( JS.gt.0.or.D1.ge.0.0D0 ) DI(I) = MIN(DI(I),D1)
315: C          end if
316: C 140      continue
317: C          else
318: C            do 150 I = 1, NN
319: C              DPCX = X(I) - SDA(LS)
320: C              DPCY = Y(I) - SDA(LS+1)
321: C              DPCZ = Z(I) - SDA(LS+2)
322: C              DVPC = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
323: C .... if ( LLG.ne.0 ) then
324: C          DH = SDA(LS+3) - DPCX*DPCX - DPCY*DPCY - DPCZ*
325: C          &      DPCZ

```

```

326:          DD = DVPC*DVPC + DH
327:          if ( DD.gt.0.0D0 ) then
328:            DDD = SQRT(DD)
329:            DLOC1(I) = MAX(DLOC1(I),-DVPC-DD)
330:            DLOC2(I) = MIN(DLOC2(I),-DVPC+DDD)
331:          else
332:            DLOC1(I) = DINF
333:            DLOC2(I) = -DINF
334:          end if
335:          if ( LLG.eq.2 ) then
336:            if ( DLOC1(I)+DEPS1.lt.DLOC2(I).and.DLOC1(I).ge.0.0
337:            &      ) DI(I) = MIN(DI(I),DLOC1(I))
338:            DLOC1(I) = -DINF
339:            DLOC2(I) = DINF
340:          end if
341: 150      continue
342: C          end if
343: C
344: C ..... CYLINDER .....
345: C
346: C          else if ( KSF.eq.3 ) then
347: C            if ( LLG.eq.0 ) then
348: C              do 160 I = 1, NN
349: C                DPCX = X(I) - SDA(LS)
350: C                DPCY = Y(I) - SDA(LS+1)
351: C                DPCZ = Z(I) - SDA(LS+2)
352: C                DUV = SDA(LS+3)*AI(I) + SDA(LS+4)*BI(I) +
353: C                &      SDA(LS+5)*CI(I)
354: C                DA = 1.0D0 - DUV*DUV
355: C                DUPC = SDA(LS+3)*DPCX + SDA(LS+4)*DPCY + SDA(LS+5)*
356: C                &      DPCZ
357: C                DB = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
358: C                &      - DUV*DUPC
359: C
360: C ..... if ( LLG.eq.0 ) then
361: C          DD = DB*DB
362: C          &      - DA*
363: C          &      (DPCX**2+DPCY**2+DPCZ**2-SDA(LS+6)-DUPC**2)
364: C          DEPSDA = DEPS2*DA**2
365: C          if ( DA.gt.0.0
366: C          &      .and.(DD.gt.DEPSDA.or.(DD.ge.0.0.and.JS.gt.0)) )
367: C          &      then
368: C            D1 = (-DB+JS*SQRT(DD)) /DA
369: C CCCCCC      if ( D1.ge.0.0 ) DI(I) = MIN(DI(I),D1)
370: C            if ( JS.gt.0.or.D1.ge.0.0 ) DI(I) = MIN(DI(I),D1)
371: C          end if
372: C 160      continue
373: C
374: C          else
375: C            do 170 I = 1, NN
376: C              DPCX = X(I) - SDA(LS)
377: C              DPCY = Y(I) - SDA(LS+1)
378: C              DPCZ = Z(I) - SDA(LS+2)
379: C              DUV = SDA(LS+3)*AI(I) + SDA(LS+4)*BI(I) +
380: C              &      SDA(LS+5)*CI(I)
381: C              DA = 1.0D0 - DUV*DUV
382: C              DUPC = SDA(LS+3)*DPCX + SDA(LS+4)*DPCY + SDA(LS+5)*
383: C              &      DPCZ
384: C              DB = AI(I)*DPCX + BI(I)*DPCY + CI(I)*DPCZ
385: C              &      - DUV*DUPC
386: C
387: C ..... if ( LLG.ne.0 ) then
388: C          DH = DPCX**2 + DPCY**2 + DPCZ**2 - SDA(LS+6)
389: C          &      - DUPC**2
390: C          DD = DB*DB - DA*DH

```


src/shared/spear1.f

```

391:         if ( DA.gt.0.0.and.DD.gt.0.0 ) then
392:             DD = SQRT(DD)
393:             DLOC1(I) = MAX(DLOC1(I),(-DB-DD)/DA)
394:             DLOC2(I) = MIN(DLOC2(I),(-DB+DD)/DA)
395:         else
396:             if ( DH.ge.0.0 ) then
397:                 DLOC1(I) = DINF
398:                 DLOC2(I) = -DINF
399:             end if
400:         end if
401:         if ( LLG.eq.2 ) then
402:             if ( DLOC1(I)+DEPS1.lt.DLOC2(I).and.DLOC1(I).ge.0.0
403:                 ) DI(I) = MIN(DI(I),DLOC1(I))
404:             DLOC1(I) = -DINF
405:             DLOC2(I) = DINF
406:         end if
407: 170         continue
408:     end if
409: C
410: C ..... QUADRATIC EXPRESSION .....
411: C
412: C S1*X**2 + S2*Y**2 + S3*Z**2 + S4*X*Y + S5*Y*Z + S6*Z*X
413: C S7*X + S8*Y + S9*Z + S10 = 0
414: C
415:     else if ( KSF.eq.4 ) then
416:         do 180 I = 1, NN
417:             & DRV1 = SDA(LS)*AI(I) + SDA(LS+3)*BI(I) + SDA(LS+5)*
418:             & CI(I)
419:             & DRV2 = SDA(LS+1)*BI(I) + SDA(LS+4)*CI(I)
420:             & DRV3 = SDA(LS+2)*CI(I)
421:             & DRW1 = SDA(LS)*X(I) + SDA(LS+3)*Y(I) + SDA(LS+5)*Z(I)
422:             & + SDA(LS+6)
423:             & DRW2 = SDA(LS+1)*Y(I) + SDA(LS+4)*Z(I) + SDA(LS+7)
424:             & DRW3 = SDA(LS+2)*Z(I) + SDA(LS+8)
425:             & DA = AI(I)*DRV1 + BI(I)*DRV2 + CI(I)*DRV3
426:             & DB =
427:             & (AI(I)*DRW1+BI(I)*DRW2+CI(I)*DRW3+X(I)*DRV1+Y(I)*
428:             & DRV2+Z(I)*DRV3)*0.5D0
429:             & DC = X(I)*DRW1 + Y(I)*DRW2 + Z(I)*DRW3 + SDA(LS+9)
430:             D11 = -DINF
431:             D22 = DINF
432:             if ( DA.gt.DSMALL ) then
433:                 DD = DB**2 - DC*DA
434:                 DEPSDA = DEPS2*DA**2
435:                 if ( DD.gt.DEPSDA .or. (DD.ge.0.0.and.JS.gt.0) ) then
436:                     DH = SQRT(DD)
437:                     D11 = (-DB-DH) /DA
438:                     D22 = (-DB+DH) /DA
439: C .....
440:                 else if ( LLG.gt.0 ) then
441:                     D11 = DINF
442:                     D22 = -DINF
443:                 end if
444:             else if ( DA.lt.-DSMALL ) then
445:                 DD = DB**2 - DC*DA
446:                 if ( DD.gt.0.0 ) then
447:                     DH = SQRT(DD)
448:                     D11 = (-DB-DH) /DA
449:                     D22 = (-DB+DH) /DA
450: C ..... CONCAVE SURFACE (DA < 0, D22 < D11 )
451:                 if ( D22.lt.0.0 ) then
452:                     D22 = DINF
453:                 else if ( LLG.eq.0 ) then
454:                     D11 = -DINF
455:                 else

```

```

456:                     D11 = D11
457:                     if ( DLOC1(I).le.D22 ) D11 = -DINF
458:                     if ( DLOC2(I).ge.D11 ) D22 = DINF
459:                 end if
460:             end if
461: C .....
462: C ..... ONLY ONE REAL ROOT IS POSSIBLE (DA = 0) .....
463:         else
464:             if ( DB.gt.0.0 ) then
465:                 D22 = -DC/(DB*2.0D0)
466:             else if ( DB.lt.0.0 ) then
467:                 D11 = -DC/(DB*2.0D0)
468:             else if ( LLG.gt.0.and.DC.ge.0.0 ) then
469:                 D11 = DINF
470:                 D22 = -DINF
471:             end if
472:         end if
473: C .....
474:         if ( LLG.eq.0 ) then
475: CCCCCC
476:             if ( JS.gt.0.and.D22.ge.0.0 ) DI(I) = MIN(DI(I),D22)
477:             if ( JS.gt.0 ) DI(I) = MIN(DI(I),D22)
478:             if ( JS.lt.0.and.D11.ge.0.0 ) DI(I) = MIN(DI(I),D11)
479:         else
480:             DLOC1(I) = MAX(DLOC1(I),D11)
481:             DLOC2(I) = MIN(DLOC2(I),D22)
482:         end if
483:         if ( LLG.eq.2 ) then
484:             & if ( DLOC1(I)+DEPS1.lt.DLOC2(I).and.DLOC1(I).ge.0.0 )
485:             & DI(I) = MIN(DI(I),DLOC1(I))
486:             & DLOC1(I) = -DINF
487:             & DLOC2(I) = DINF
488:         end if
489: 180         continue
490: C ..... PLANE (HALF SPACE) .....
491: C
492:     else if ( KSF.eq.5 ) then
493:         if ( LLG.eq.0 ) then
494:             do 190 I = 1, NN
495:                 & DA = SDA(LS)*AI(I) + SDA(LS+1)*BI(I) + SDA(LS+2)*
496:                 & CI(I)
497:                 & DB = SDA(LS+3)
498:                 & - (SDA(LS)*X(I)+SDA(LS+1)*Y(I)+SDA(LS+2)*Z(I))
499: C ... if ( LLG.eq.0 ) then
500:                 if ( JS*DA.gt.0.0 ) then
501:                     DD = DB/DA
502: CCCCCC
503:                     if ( DD.ge.0.0 ) DI(I) = MIN(DI(I),DD)
504:                     if ( DI(I) = MIN(DI(I),DD)
505:                 end if
506:                 190         continue
507:             else
508: C ... Has been assuming surface sign JS for half-space is always
509: C negative for LLG > 0 (minus sign for body),
510: C but it may not valid when a "HAF" body is used as
511: C a component of a "BBC" body and
512: C the "HAF" body is combined with minus sign!!
513: C
514: C Faking opposite signed "HAF" body when JS>0.
515: C (21 Mar 2000 M.Sasaki)
516: C
517:                 DSDA = -JS*SDA(LS)
518:                 DSDA1 = -JS*SDA(LS+1)
519:                 DSDA2 = -JS*SDA(LS+2)
520:                 DSDA3 = -JS*SDA(LS+3)

```

src/shared/spear1.f

```
521: C
522:       do 200 I = 1, NN
523:         DA      = SDA(LS)*AI(I) + SDA(LS+1)*BI(I) + SDA(LS+2)*
524: C      &         CI(I)
525:         DB      = SDA(LS+3)
526: C      &         - (SDA(LS)*X(I)+SDA(LS+1)*Y(I)+SDA(LS+2)*Z(I))
527:
528:         DA      = DSDA*AI(I) + DSDA1*BI(I) + DSDA2*CI(I)
529:         DB      = DSDA3 - (DSDA*X(I)+DSDA1*Y(I)+DSDA2*Z(I))
530: C      ... if ( LLG.ne.0 ) then
531:         if ( DA.gt.0.0 ) then
532:           DD      = DB/DA
533:           DLOC2(I) = MIN(DLOC2(I),DD)
534:         else if ( DA.lt.0.0 ) then
535:           DD      = DB/DA
536:           DLOC1(I) = MAX(DLOC1(I),DD)
537:         else
538:           if ( DB.le.0.0 ) then
539:             DLOC1(I) = DINF
540:             DLOC2(I) = -DINF
541:           end if
542:         end if
543:         if ( LLG.eq.2 ) then
544:           if ( DLOC1(I)+DEPS1.lt.DLOC2(I).and.DLOC1(I).ge.0.0
545: C      &           ) DI(I) = MIN(DI(I),DLOC1(I))
546:           DLOC1(I) = -DINF
547:           DLOC2(I) = DINF
548:         end if
549:       200 continue
550:       end if
551: C
552: C ..... ELLIPTIC TORUS .....
553: C
554:       else if ( KSF.eq.6 ) then
555:         call SQSOL1( X, Y, Z, IKL, AI, BI, CI, SDA, LS-1, NN, DINF,
556: C      &         DEPS, DI, IKL2 )
557:       end if
558: C
559: C ... set surface pointer to IKL2 if the surface gives smallest
560: C distance.
561: C
562: C currently used in CGVIEW only, and it may give invalid
563: C surface # when any value is not set to DI(I) for current surface.
564: C (21 Mar 2000 M.Sasaki)
565: C
566:       if ( JSS.ne.0.and.LLG.ne.1.and.KSF.ne.6 ) then
567:         do 210 I = 1, NN
568:           if ( DI(I).ne.DINF.and.DI(I).lt.DSAVE(I) ) then
569:             IKL2(I) = LS1
570:           end if
571:           DSAVE(I) = DI(I)
572:       210 continue
573:       end if
574: C
575: C .....
576: C
577:       220 continue
578: C
579:       do 230 I = 1, NN
580: CCCCC if ( DI(I).lt.0.0 ) DI(I) = DINF
581:       if ( DI(I).lt.0.0 ) DI(I) = 0.d0
582:       230 continue
583:       return
584:       end
```

src/shared/spnums.f

```

1:      subroutine SPNUMS( NSPACE, LAST,  ISPNM, KREG,  KUNV,  NUNV,
2:      &                  NSUZON, KSUZN, KZUNV, NNAMES, TNAMS, NLATT,
3:      &                  NCELL, NEST,  NINPZ, NZONE, IDLAT, KLTSP,
4:      &                  ILTSP, KCLSP, KCLST, LIMIT, KREGN, KREGI,
5:      &                  NLTSPC, NSMSPC, KMAT, KCELI, LTYP, KLATT, IPLAT,
6:      &                  IPCLI, KINPZ )
7: C=====
8: C PURPOSE : CHECK SUBSPACE NUMBER AND CREATE SPACE-NAME TABLES.
9: C          SET KREG VALUES FOR LATTICE-FRAME INPUT-ZONES.
10: C
11: C CALLED IN :  GEOMIN
12: C-----
13: C ARGUMENTS :
14: C
15: C      === OUTPUT OR INPUT/OUTPUT VARIABLES ===
16: C
17: C      NSPACE : NUMBER OF SUBSPACES.
18: C      ISPNM(2,0:NEST,NSPACE)
19: C      LAST : STARTING POSITION OF AVAILABLE ARRAY-DATA STORAGE.
20: C
21: C      IT MUST BE THE STARTING POSITION OF TNAMS AT THE BEGINNING
22: C
23: C      OF THIS ROUTINE AND SET TO LISPNM + 2*NEST*NSPACE
24: C      CHARACTERS).
25: C      NUNV : NUMBER OF lattice-frame-zone
26: C      KREG(NINPZ) :
27: C      IF -1 <= MAT < -999 : 0
28: C      IF IN CELL : SEQUENTIAL # IN ZONES IN ALL SUB-SPACES.
29: C      KUNV(NINPZ) : lattice-frame-zone # IF ZONE IS lattice-frame-zone
30: C      NSUZON : NUMBER OF terminal-(INPUT)-ZONES IN SUB-UNIVERSES
31: C      KSUZN(NINPZ,2) :
32: C      (1:NSUZON,1) : INPUT-ZONE # FOR terminal-(INPUT)-ZONES in cell
33: C      (1:NINPZ,1) : terminal-(INPUT)-ZONE # FOR EACH INPUT-ZONE
34: C
35: C      === REFERENCE ONLY ===
36: C
37: C      NNAMES : TOTAL NUMBER OF NAMES (TO BE COUNTED IN THIS ROUTINE)
38: C      TNAMS(NNAMES)
39: C      : NAMES STORED AS CHARACTER*12 DATA.
40: C      BEFORE CALLING THIS ROUTINE,
41: C      STARTING POSITION IN ARRAY-DATA STORAGE ONLY GIVEN,
42: C      LIMIT : LIMIT OF ARRAY-DATA STOREGE IN WORD.
43: C      KREGN : REGION OR FRAME NAMES INPUT IN ZONEIN ROUTINE.
44: C      KREGI : KREGN POSITION IN TNAMS.
45: C      NINPZ : NUMBER OF INPUT-ZONES
46: C      KLATT : LATTICE CELL TABLE POINETERS FOR EACH LATTICE.
47: C      NLATT : NUMBER OF LATTICE
48: C      NLTSPC : NUMBER OF (FRAME , LATTICE) COMBINATION
49: C      (COUNTED IN ZONEIN)
50: C      NSMSPC : TOTAL NUMBER OF SUBFRAME-CELL SPECIFICATIONS.
51: C      (SUMMED IN ZONEIN)
52: C      KLTSP(2,NLTSPC)
53: C      : (FRAME,LATTICE) -> FRAME NAME INDEX(KREGN) &
54: C      LATTICE # (NAME INDEX WILL BE CHANGED TO INDEX TO TNAMS)
55: C      ILTSP(NLTSPC+1)
56: C      : POINTER TO KSPCL ARRAY FOR EACH (FRAME, LATTICE)
57: C      COMBINATION. (MEMORY ALREADY RESEVED)
58: C      KCLSP(NSMSPC)
59: C      : SUBFRAME NAME TABLE (POINTERS TO TNAMS TABLE)
60: C      FOR SUBFRAME-CELL COMBINATIONS (MEMORY ALREADY RESEVED)
61: C=====
62:      integer ISPNM(2,0:NEST,*), KREG(NINPZ), KSUZN(NINPZ,2),
63:      &      KUNV(NINPZ), KZUNV(NZONE)
64:      include 'INC/_LNAM'
65: C

```

```

66:      integer NNAMES
67:      character*(LNAM) TNAMS(NNAMES)
68:      integer KLTSP(2,NLTSPC), ILTSP(NLTSPC), KCLSP(NSMSPC)
69:      integer KCLST(NSMSPC)
70: C
71:      integer KREGI(NINPZ), KINPZ(NZONE)
72:      integer KCELI(NINPZ), IDLAT(NLATT), IPCLI(NCELL+1), IPLAT(*),
73:      &      KLATT(*), KMAT(NINPZ)
74:      integer LTYP(NLATT)
75:      character*(LNAM) KREGN(NINPZ)
76: C
77:      include 'INC/_IOUNIT'
78: C
79: C-----
80: C
81:      parameter( MAXLVL = 4 )
82: C
83: C      .... LOCAL WORKING ARRAYS TO SWEEP NESTING STRUCTURE
84: C
85:      integer IZP(0:MAXLVL), IZL(0:MAXLVL), IZC(0:MAXLVL),
86:      &      IZCP(0:MAXLVL), IZLS(0:MAXLVL), IZSP(0:MAXLVL)
87: C
88:      include 'INC/_PMLATT'
89:      include 'INC/_SFLATT'
90: C
91: C=====
92: C
93: C
94:      NSUZON = 0
95:      NUNV = 0
96: C
97: C-----
98: C      NUNV : Number of "frame"-input-zones which include geometric
99: C      sub-structures such as lattice.
100: C      NSUZON : Number of "terminal"-zones which is defined as input-zones
101: C      in lattice-cell and for which tallies can be taken (mat ID is
102: C      non-negative) as regions in subspaces.
103: C-----
104: C
105:      do 100 I = 1, NINPZ
106:         KREG(I) = 0
107:         KUNV(I) = 0
108:         if ( KCELI(I).gt.0.and.KMAT(I).ge.0 ) then
109:            NSUZON = NSUZON + 1
110:            KREG(I) = NSUZON
111:            KSUZN(NSUZON,1) = I
112:            KSUZN(I,2) = NSUZON
113:         end if
114:         if ( ISLATT(KMAT(I)) ) then
115:            NUNV = NUNV + 1
116:            KUNV(I) = NUNV
117:            KREG(I) = NUNV
118:         end if
119:      100 continue
120:      do 110 IZ = 1, NZONE
121:         KZUNV(IZ) = KUNV(KINPZ(IZ))
122:      110 continue
123: C
124: C      *****
125: C      call LABEL( IOG, 'SUBSPACE STRUCTURE' )
126: C      *****
127: C
128: C      ... SEARCH ZONES AND PRINT SPACE STRUCTURE. ....
129: C
130:      NSPACE = 0

```

src/shared/spnums.f

```

131: C
132:     LV      = 0
133:     IZSP(0) = 0
134:     IZP(0)  = 0
135:     IZC(0)  = 0
136:     IZL(0)  = 0
137: C
138: *   WRITE(IOG,7700) ( ' LATTICE      ', ' CELL      ',I=1,MAXLVL)
139: C   WRITE(IOG,7710) ( ' (TYPE)      ', ' (DIRECTION) ',I=1,MAXLVL)
140: C7700 FORMAT(//4X,5(A12      ,A12: ) )
141: C7710 FORMAT( 4X,5(A12      ,A12:)/ )
142: C7720 FORMAT( 4X,5(A6,'(' ,A4,')' ,12X: ) )
143: C7730 FORMAT( 4X,5(12X      , A6,'(' ,A4,')': ) )
144: C
145: C
146: 120 IZP(LV) = IZP(LV) + 1
147: C
148: C
149: C -----
150: C     .... END OF A CELL
151: C -----
152: C
153:     if ( IZP(LV).gt.NINPZ .or. (LV.gt.0.and.KCELI(IZP(LV)).ne.IZC(LV))
154: & ) then
155: C
156: C
157: 130     IZCP(LV) = IZCP(LV) + 1
158: C
159: C
160: C     .... END OF LATTICE CELL ...
161: C
162: C
163:     if ( LTYP(IZL(LV)).eq.10.and.
164: &       IZCP(LV).gt.KLATT(IPLAT(IZL(LV))+9) .or.
165: &       LTYP(IZL(LV)).ne.10.and.
166: &       IPLAT(IZL(LV))+IZCP(LV).ge.IPLAT(IZL(LV)+1) ) then
167:         LV = LV - 1
168:         go to 120
169: C
170: C     .... ENTER NEXT CELL ....
171: C
172:     else
173:         IZCPT = KCLST(IZLS(LV)+IZCP(LV)) + IPLAT(IZL(LV))
174:         if ( KLATT(IZCPT).eq.0 ) go to 130
175: C
176: C     ... ALREADY CHECKED CELL OR NOT ? ....
177: C
178:         ISP = KCLSP(IZLS(LV)+IZCPT-IPLAT(IZL(LV)))
179:         do 140 K1 = 0, IZCP(LV) - 1
180:             K0 = KCLST(IZLS(LV)+K1) + IZLS(LV)
181:             K   = KCLST(IZLS(LV)+K1) + IPLAT(IZL(LV))
182:             if ( KLATT(K).eq.KLATT(IZCPT).and.KCLSP(K0).eq.ISP )
183: &             go to 130
184: 140         continue
185: C
186: C
187: C     ... NEXT CELL IN CURRENT LATTICE ....
188: C
189:         IZC(LV) = KLATT(IZCPT)
190:         IZP(LV) = IPCLI(IZC(LV))
191: C
192: C     .... ADD SUBSPACE IF NEW SUBSPACE IS ENCOUNTERED, ELSE RETRURN
193: C     SUBSPACE # IN IZSP.
194: C
195:         NSF = KREGI(IZP(LV-1))

```

```

196: C
197:         call SPCCHK( 'ADD', NSPACE, IZSP(LV), ISPNM, NEST, LV,
198: &                   IZSP(LV-1), NSF, ISP, LAST, LIMIT )
199: C
200:         go to 120
201:     end if
202: end if
203: C
204: C
205: C
206: C
207:     IMAT = KMAT(IZP(LV))
208: C
209:     if ( LV.eq.0.and.IMAT.eq.-999 ) go to 180
210: C
211: C
212: C -----
213: C     .... ENTERING A LATTICE (OR SUB-UNIVERSE) ....
214: C -----
215: C
216: C
217:     if ( ISLATT(IMAT) ) then
218:         LV = LV + 1
219: C
220: C
221:     if ( LV.gt.NEST ) then
222:         write(IMG,*) 'XXX(SPNUMS) ',
223: &                   ' Too deep geometry nest ( > NEST = ', NEST, ' )'
224:         call PRSTOP( 1, ' ' )
225:         stop 666
226:     end if
227: C
228: C
229: C     .... LATTICE #
230: C     IZL(LV) = LATTNM(IMAT)
231: C
232: C     ... SEARCH FRAME:LATTICE TABLE ...
233: C
234:         NSF = KREGI(IZP(LV-1))
235:         do 150 K = 1, NLTSPC
236:             if ( KLTSP(1,K).eq.NSF.and.KLTSP(2,K).eq.IZL(LV) ) go to 160
237: 150         continue
238: C
239:         write(IMG,*) 'XXX(SPNUMS) No corresponding ',
240: &                   '(FRAME:LATTICE) table data for input-zone ', IZP(LV-1)
241:         write(IMG,*) ' KREGI ', NSF, ' LATT ', IDLAT(IZL(LV))
242:         call PRSTOP( 1, ' ' )
243:         stop 666
244: 160         continue
245: C
246: C     .... CELL ARRAY POINTER.
247: C
248:         IZCP(LV) = 0
249: C
250: C     .... CELL #
251: C
252: 170         IZCPT = KCLST(ILTSP(K)+IZCP(LV)) + IPLAT(IZL(LV))
253:         IZC(LV) = KLATT(IZCPT)
254:         if ( IZC(LV).eq.0 ) then
255:             IZCP(LV) = IZCP(LV) + 1
256:             go to 170
257:         end if
258: C
259: C     .... CELL STARING ZONE ( KMAT = -999 )
260: C

```

```

261:      IZP(LV) = IPCLI(IZC(LV))
262:      IZLS(LV) = ILTSP(K)
263:      ISP      = KCLSP(IZLS(LV)+IZCPT-IPLAT(IZL(LV)))
264: C
265: C      .... ADD SUBSPACE IF NEW SUBSPACE IS ENCOUNTERED, ELSE RETURN
266: C      SUBSPACE # IN IZSP.
267: C
268: C
269: C      call SPCCHK( 'ADD', NSPACE, IZSP(LV), ISP, NSP, LV,
270: &      IZSP(LV-1), NSF, ISP, LAST, LIMIT )
271: C
272: C
273: C      go to 120
274: C      end if
275: C      go to 120
276: 180 continue
277: C
278: C      .... REORDER ISP
279: C
280: C      write(IOG,'(1X,A,18)') ' * TOTAL NUMBER OF SUBSPACES: ', NSPACE
281: C      write(IOG,
282: &      '(1X,18) * SUBSPACE NAMES (IN ORDER OF APPEARANCE) *')
283: &      '(1X,18) * NAME *')
284: C
285: C      do 190 I = 1, NSPACE
286: C      write(IOG,'(1X,18,1X,5('1',A,18))') I,
287: &      (TNAMS(ISPNM(1,L,I)),TNAMS(ISPNM(2,L,I))),I=1,
288: &      ISPNM(1,0,I))
289: 190 continue
290: C      return
291: C      end

```

```

261:      IZP(LV) = IPCLI(IZC(LV))
262:      IZLS(LV) = ILTSP(K)
263:      ISP      = KCLSP(IZLS(LV)+IZCPT-IPLAT(IZL(LV)))
264: C
265: C      ... ADD SUBSPACE IF NEW SUBSPACE IS ENCOUNTERED, ELSE RETURN
266: C      SUBSPACE # IN IZSP.
267: C
268: C
269: C      call SPCCHK( 'ADD', NSPACE, IZSP(LV), ISP, NSP, LV,
270: C      &            IZSP(LV-1), NSF, ISP, LAST, LIMIT )
271: C
272: C
273: C      go to 120
274: C      end if
275: C      go to 120
276: C 180 continue
277: C
278: C      .... REORDER ISPNM
279: C
280: C      write(IOG, '(1X,A,10)' ) ' * TOTAL NUMBER OF SUBSPACES: ', NSPACE
281: C      write(IOG,
282: C      & ' (/5X, ' * SUBSPACE NAMES (IN ORDER OF APPEARANCE) * ' /
283: C      & ' /5X, ' NO. NAME ' ' ' )
284: C
285: C      do 190 I = 1, NSPACE
286: C      write(IOG, '(5X,I8,1X,5(' ' ' ' ,A ' ' : ' ' ,A:)) ' ) I,
287: C      & (TNAMS(ISPNM(1,L,I)),TNAMS(ISPNM(2,L,I))),I=1,
288: C      & ISPNM(1,0,I))
289: C 190 continue
290: C      return
291: C      end

```

src/shared/spreg.f

```

1:      subroutine SPREG( NREG, KTCSP, NLTEMP,NKTCSP,KSPSU,
2:      O          ISUSP, KREG, KZREG,
3:      R          NUNV, NSUZON,NSPACE,
4:      R          NNames,TNAMS, NZONE, NLATT, NCELL, NEST, KLTSP,
5:      R          ILTSP, KCLSP, KCLST, ISPNM, KUNV, LIMIT, KREGN,
6:      R          KREGI, NINPZ, NLTSPC,NSMSPC,KMAT, KCELI, KINPZ,
7:      R          LTyp, KLATT, IPLAT, IPCLI, IDLAT, IDCEL )
8: C=====
9: C PURPOSE : CREATE SUBSPACE-REGION MAPPING TABLE FROM INPUT-ZONE DATA
10: C AND LATTICE DATA.
11: C CALLED IN : GEOMIN
12: C-----
13: C Arguments (i=input, o=output, w=work)
14: C
15: C === OUTPUT OR INPUT/OUTPUT VARIABLES ===
16: C
17: C o ISUSP(NSUZON,NSPACE) : SUBSPACE # FOR EACH terminal-input-zone.
18: C o KSPSU(NUNV,0:NSPACE) : POINTERS TO SUBSPACE # TABLE (KTCSP)
19: C FOR LATTICE-FRAME-ZONES IN EACH SUBSPACE.
20: C o KTCSP(*) : SUBSPACE # FOR CELL UNDER LATTICE-FRAME-ZONES IN EACH
21: C SUBSPACE
22: C
23: C** LAST : STARTING POSITION OF AVAILABLE ARRAY-DATA STORAGE.
24: C**
25: C** IT MUST BE THE STARTING POSITION OF KTCSP AT THE BEGINNING
26: C**
27: C** OF THIS ROUTINE AND SET TO THE NEXT POSITION OF KTCSP.
28: C i o KREG(NINPZ) :
29: C i o KZREG(NZONE) :
30: C REGION # FOR zones in root-cell (SUBSPACE #0)
31: C IF -1<= MAT < -999 : LATTICE #
32: C IF IN CELL : NEGATED SEQUENTIAL # IN ZONES IN ALL SUB-SPACES.
33: C
34: C === REFERENCE ONLY ===
35: C
36: C i NSPACE : NUMBER OF SUBSPACES.
37: C i NUNV : NUMBER OF lattice-frame-zone (0 > MAT > -999)
38: C i NSUZON : NUMBER OF terminal-input-zone
39: C (TALLIED-ZONES IN SUB-SPACES. ( MAT >= 0 ))
40: C i ISPNM(2,0:NEST,NSPACE) : SUBSPACE NAME LIST.
41: C i NNames : TOTAL NUMBER OF NAMES (TO BE COUNTED IN THIS ROUTINE)
42: C i TNAMS(NNames)
43: C : NAMES STORED AS CHARACTER*12 DATA.
44: C BEFORE CALLING THIS ROUTINE,
45: C STARTING POSITION IN ARRAY-DATA STORAGE ONLY GIVEN,
46: C i LIMIT : LIMIT OF ARRAY-DATA STOREGE IN WORD.
47: C i KREGN : REGION OR FRAME NAMES INPUT IN ZONEIN ROUTINE.
48: C i KREGI : POINTERS TO 'TNAMS' OF 'KREGN' NAMES OF EACH INPUT-ZONE
49: C i NINPZ : NUMBER OF INPUT-ZONES
50: C i NZONE : NUMBER OF ZONES
51: C i KLATT : LATTICE CELL TABLE POINETERS FOR EACH LATTICE.
52: C i NLATT : NUMBER OF LATTICE
53: C i NLTSPC : NUMBER OF (FRAME , LATTICE) COMBINATION
54: C (COUNTED IN ZONEIN)
55: C i NSMSPC : TOTAL NUMBER OF SUBFRAME-CELL SPECIFICATIONS.
56: C (SUMMED IN ZONEIN)
57: C i KLTSP(2,NLTSPC)
58: C : (FRAME,LATTICE) -> FRAME NAME INDEX(KREGN) &
59: C LATTICE ID (NAME INDEX WILL BE CHANGED TO INDEX TO TNAMS)
60: C i ILTSP(NLTSPC+1)
61: C : POINTER TO KSPCL ARRAY FOR EACH (FRAME, LATTICE)
62: C COMBINATION. (MEMORY ALREADY RESEVED)
63: C i KCLSP(NSMSPC)
64: C : SUBFRAME NAME TABLE (POINTERS TO TNAMS TABLE)
65: C FOR SUBFRAME-CELL COMBINATIONS (MEMORY ALREADY RESEVED)

```

```

66: C i IPLAT,IPCLI,IDLAT, IDCEL :
67: C=====
68:      integer KSPSU(NUNV,0:NSPACE), ISUSP(NSUZON,NSPACE), KTCSP(NLTEMP),
69:      &      KREG(NINPZ), KZREG(NZONE)
70: C
71:      include 'INC/_LNAME'
72: C
73:      integer NNames
74:      integer ISPNM(2,0:NEST,NSPACE)
75:      character*(LNAME) TNAMS(NNames)
76:      integer KLTSP(2,NLTSPC), ILTSP(NLTSPC+1), KCLSP(NSMSPC),
77:      &      KCLST(NSMSPC)
78: C
79: C
80:      integer KCELI(NINPZ), IPCLI(NCELL+1), IDLAT(NLATT), IDCEL(NCELL),
81:      &      IPLAT(*), KLATT(*), KMAT(NINPZ), KINPZ(NZONE), KUNV(NINPZ)
82:      integer LTyp(NLATT)
83:      integer KREGI(NINPZ)
84:      character*(LNAME) KREGN(NINPZ)
85: C
86:      include 'INC/_IOUNIT'
87: C
88: C-----
89: C
90:      parameter( MAXLVL = 4 )
91: C
92: C ..... LOCAL WORKING ARRAYS TO SWEEP NESTING STRUCTURE
93: C
94:      integer IZP(0:MAXLVL), IZL(0:MAXLVL), IZC(0:MAXLVL),
95:      &      IZCP(0:MAXLVL), IZLS(0:MAXLVL), IZSP(0:MAXLVL)
96: C
97:      include 'INC/_PMLATT'
98:      include 'INC/_SFLATT'
99: C
100: C=====
101: C
102: C
103: C *****
104: C call LABEL( IOG, 'REGIONS' )
105: C *****
106: C
107: C NREG = 0
108: C
109: C call GTLAST( LAST )
110: C
111: C
112: C LV = 0
113: C IZSP(0) = 0
114: C IZP(0) = 0
115: C IZC(0) = 0
116: C IZL(0) = 0
117: C NKTCSP = 1
118: C
119: C
120: C 100 IZP(LV) = IZP(LV) + 1
121: C
122: C -----
123: C .... END OF A CELL
124: C -----
125: C
126: C if ( IZP(LV).gt.NINPZ .or. (LV.gt.0.and.KCELI(IZP(LV)).ne.IZC(LV))
127: C & ) then
128: C
129: C
130: C 110 IZCP(LV) = IZCP(LV) + 1

```

src/shared/spreg.f

```

131: C
132: C
133: C      .... END OF LATTICE CELL ...
134: C
135: C
136: C      if ( LTyp(IZL(LV)).eq.10.and.
137: C      &      IZCP(LV).gt.KLATT(IPLAT(IZL(LV))+9) .or.
138: C      &      LTyp(IZL(LV)).ne.10.and.
139: C      &      IPLAT(IZL(LV))+IZCP(LV).ge.IPLAT(IZL(LV)+1) ) then
140: C          LV      = LV - 1
141: C          go to 100
142: C
143: C      .... ENTER NEXT CELL ....
144: C
145: C      else
146: C          IZCPT      = KCLSP(IZLS(LV)+IZCP(LV)) + IPLAT(IZL(LV))
147: C          if ( KLATT(IZCPT).eq.0 ) go to 110
148: C
149: C      ... NEXT CELL IN CURRENT LATTICE ...
150: C
151: C          IZC(LV) = KLATT(IZCPT)
152: C          IZP(LV) = IPCLI(IZC(LV))
153: C
154: C      .... CHECK SUBSPACE # & RETURN IN IZSP(LV) ....
155: C
156: C      .... FRAME NAME OF UNIVERSE .....
157: C
158: C          NSF      = KREGI(IZP(LV-1))
159: C
160: C      .... SUBFRAME-NAME OF THE CELL .....
161: C
162: C          K0      = IZCPT - IPLAT(IZL(LV))
163: C          ISP      = KCLSP(IZLS(LV)+K0)
164: C
165: C          call SPCCHK( 'CHK', NSPACE, IZSP(LV), ISPNM, NEST, LV,
166: C      &      IZSP(LV-1), NSF, ISP, LAST, LIMIT )
167: C
168: C
169: C          IUNV      = KUNV(IZP(LV-1))
170: C          KTCSP(KSPSU(IUNV,IZSP(LV-1))+K0)      = IZSP(LV)
171: C          go to 100
172: C      end if
173: C  end if
174: C
175: C
176: C
177: C      IMAT      = KMAT(IZP(LV))
178: C
179: C      if ( LV.eq.0.and.IMAT.eq.-999 ) go to 180
180: C
181: C      .... LEVEL 0 & UNIVERSE = 0 & SPACE = 0 )
182: C
183: C      if ( IMAT.ge.0 ) then
184: C          if ( LV.eq.0 ) then
185: C              KI      = KREGI(IZP(LV))
186: C              do 120 I = 1, IPCLI(1) - 1
187: C                  if ( KMAT(I).ge.0.and.KREG(I).gt.0.and.KI.eq.KREGI(I) )
188: C                      &
189: C                          KREG(IZP(LV))      = KREG(I)
190: C                      go to 100
191: C                  end if
192: C          120      continue
193: C
194: C          NREG      = NREG + 1
195: C          KREG(IZP(LV))      = NREG

```

```

196: C          go to 100
197: C
198: C      .... LEVEL > 0 AND NON UNIVERSE FRAME ....
199: C
200: C      else if ( LV.gt.0 ) then
201: C
202: C          ISUZN      = KREG(IZP(LV))
203: C          if ( ISUZN.eq.0 ) go to 100
204: C
205: C          if ( ISUSP(ISUZN,IZSP(LV)).eq.0 ) then
206: C              KI      = KREGI(IZP(LV))
207: C
208: C              .... CHECK REGION NAMES IN THIS SUBSPACE ....
209: C
210: C              do 130 I = IPCLI(1), NINPZ
211: C                  if ( I.eq.IZP(LV) ) go to 130
212: C                  ISU      = KREG(I)
213: C                  if ( KMAT(I).ge.0.and.KREGI(I).eq.KI
214: C      &                  .and.ISUSP(ISU,IZSP(LV)).ne.0 ) then
215: C                      ISUSP(ISUZN,IZSP(LV))      = ISUSP(ISU,IZSP(LV))
216: C                      go to 100
217: C                  end if
218: C          130      continue
219: C
220: C              .... NEW REGION NAMES IN THIS SUBSPACE ....
221: C
222: C              NREG      = NREG + 1
223: C              ISUSP(ISUZN,IZSP(LV))      = NREG
224: C              end if
225: C              end if
226: C              go to 100
227: C          end if
228: C
229: C      -----
230: C      .... ENTERING A LATTICE ....
231: C      -----
232: C
233: C
234: C
235: C      if ( ISLATT(IMAT) ) then
236: C          LV      = LV + 1
237: C
238: C
239: C      if ( LV.gt.NEST ) then
240: C          write(IMG,*) 'XXX(SPREG) ',
241: C      &          ' Too deep geometry nest ( > NEST = ', NEST, ' )'
242: C          call PRSTOP( 1, 'TOO DEEPLY NESTED GEOMETRY.' )
243: C          stop 666
244: C      end if
245: C
246: C      .... LATTICE #
247: C
248: C      IZL(LV) = LATTNM(IMAT)
249: C
250: C      ... SEARCH FRAME:LATTICE TABLE ...
251: C
252: C          NSF      = KREGI(IZP(LV-1))
253: C          do 140 K = 1, NLTSPC
254: C              if ( KLTSP(1,K).eq.NSF.and.KLTSP(2,K).eq.IZL(LV) ) go to 150
255: C          140      continue
256: C
257: C          write(IMG,*) 'XXX(SPREG) No corresponding ',
258: C      &          '(FRAME:LATTICE) table data for input-zone ', IZP(LV)
259: C          write(IOG,*) ' KREG ', NSF, ' LATT ', IDLAT(IZL(LV))
260: C          call PRSTOP( 1, 'INVALID FRAME SPECIFICATION.' )

```

src/shared/spreg.f

```

261:      stop 666
262: C
263: 150      IZLS(LV)      = ILTSP(K)
264: C
265: ***** IUNV      = KZREG(IZP(LV-1))
266:      IUNV      = KUNV(IZP(LV-1))
267: C
268: C      .... REGISTRATION IN KSPSU ETC. THIS frame-zone OF CURRENT SUBSPACE
269: C      HAS BEEN FINISHED.
270: C
271:      if ( KSPSU(IUNV,IZSP(LV-1)).ne.0 ) then
272:      LV      = LV - 1
273:      go to 100
274:      else
275:      KSPSU(IUNV,IZSP(LV-1)) = NKTCSPP
276:      K0      = IPLAT(IZL(LV)+1) - IPLAT(IZL(LV))
277:      NKTCSPP = NKTCSPP + K0
278: C
279:      if ( NKTCSPP.gt.NKTEMP ) then
280:      call MEMERR( 'SPREG',
281:      &      'PREPARATION FOR CELL-SPACE TABLE (KTCSP).',
282:      &      LSIZ(LAST+NKTCSPP), LIMIT )
283:      stop 999
284:      end if
285: C
286:      do 160 I = NKTCSPP - K0, NKTCSPP - 1
287:      KTCSP(I)      = 0
288: 160      continue
289:      end if
290: C
291: C
292: C      .... CELL ARRAY POINTER.
293: C
294:      IZCP(LV)      = 0
295: C
296: C      .... CELL #
297: C
298: 170      IZCPT      = KCLST(ILTSP(K)+IZCP(LV)) + IPLAT(IZL(LV))
299:      IZC(LV) = KLATT(IZCPT)
300:      if ( IZC(LV).eq.0 ) then
301:      IZCP(LV)      = IZCP(LV) + 1
302:      go to 170
303:      end if
304: C
305: C      .... CHECK SUBSPACE # AND RETURN IN IZSP(LV)
306: C
307:      NSF      = KREGI(IZP(LV-1))
308:      K0      = IZCPT - IPLAT(IZL(LV))
309:      ISP      = KCLSP(IZLS(LV)+K0)
310: C
311:      call SPCCHK( 'CHK', NSPACE, IZSP(LV), ISPNM, NEST, LV,
312:      &      IZSP(LV-1), NSF, ISP, LAST, LIMIT )
313: C
314: C      .... SUBSPACE # TABLE ....
315: C
316:      IUNV      = KUNV(IZP(LV-1))
317:      KTCSP(KSPSU(IUNV,IZSP(LV-1))+K0) = IZSP(LV)
318: C
319: C      .... CELL STARTING ZONE ( KMAT = -999 )
320: C
321:      IZP(LV) = IPCLI(IZC(LV))
322: C
323:      go to 100
324:      end if
325:      go to 100

```

```

326: 180 continue
327: C
328:      NKTCSPP = NKTCSPP - 1
329: C
330:      do 190 I = 1, NZONE
331:      KZREG(I)      = KREG(KINPZ(I))
332: 190 continue
333: C
334:      write(IOG,'(/lx,a,i8)') '      * TOTAL NUMBER OF REGIONS: ', NREG
335: C
336:      write(IOG,
337:      &      '(/6X,*** SUBSPACE NAMES & REGIONS UNDER THEM **//
338:      &      6X,NO.      NAME      '/')')
339: C
340:      write(IOG,'(lx,a)') '      0 !      (ROOT-CELL/FRAME)'
341: C
342:      do 200 K = 1, IPCLI(1) - 1
343:      if ( KMAT(K).ge.0 ) then
344:      write(IOG,7000) KREG(K), TNAMS(KREGI(K)), K, KMAT(K)
345: 7000      format(12X,'REG.',I8,2X,A,3X,'( INPUT ZONE:',I5,', MAT: ',
346:      &      I6,')')
347:      end if
348: 200 continue
349: C
350:      do 220 IS = 1, NSPACE
351:      write(IOG,'(7X,I8,1X,5(''!'',A,':'',A:))') IS, (
352:      &      TNAMS(ISPNM(1,L,IS))(:ICLEN(TNAMS(ISPNM(1,L,IS))))),
353:      &      TNAMS(ISPNM(2,L,IS))(:ICLEN(TNAMS(ISPNM(2,L,IS))))),L=1,
354:      &      ISPNM(1,0,IS))
355:      NR      = 0
356: C
357:      do 210 K = IPCLI(1), NINPZ
358:      if ( .not.ISLATT(KMAT(K)) ) then
359:      ISU      = KREG(K)
360:      if ( ISU.gt.0.and.ISUSP(ISU,IS).ne.0 ) then
361:      write(IOG,7020) ISUSP(ISU,IS), TNAMS(KREGI(K)), K,
362:      &      KMAT(K), IDCEL(KCELI(K))
363: 7020      format(12X,'REG.',I8,2X,A,3X,'( INPUT ZONE:',I5,
364:      &      ', MAT: ',I6,', CELL:',I6,')')
365:      end if
366:      end if
367: 210 continue
368: 220 continue
369: C-----
370:      return
371:      end

```



```
src/shared/sqsol1.f
```

```

1:      subroutine SQSOL1( X,      Y,      Z,      IKL,      AI,      BI,      CI,
2:      &                  SDA,      LS,      NN,      DINF,      DEPS,      DI,      IKL2
3:      &                  )
4: C-----
5: C  PURPOSE : CALCULATION OF DISTANCES TO AN ELLIPTIC-TORUS.
6: C  CALLED IN: FLIONE
7: C  CALLS : (NONE)
8: C
9: C-----
10: C arguments (i=input, o=output, w=work)
11: C
12: C i X(NN),Y(NN),Z(NN) : position of particles
13: C i IKL(NBANK) : surface pointer (to SDA) on which particle is
14: C                 currently placed if non-zero.
15: C                 Bank data KLSF(*) of each particle should be stored
16: C                 in calling routine.
17: C i AI(NN),BI(NN),CI(NN) : direction vector of particles
18: C i SDA(*) : surface data array
19: C i LS : torus geometry data starts from SDA(LS).
20: C i NN : number of particles
21: C i DINF : "infinite" distance.
22: C i DEPS : "minimum" distance.
23: C i o DI(I) : minimum distance to surfaces composing a zone.
24: C o IKL2(NBANK) : surface pointer (to SDA) on which particle will be
25: C                 placed if the calculated distance to the torus is
26: C                 selected as smallest one in body list.
27: C-----
28:      implicit real*8(A-H,O-Z)
29: C
30: C      .... INPUT ....
31: C
32:      real*8 X(NN), Y(NN), Z(NN), AI(NN), BI(NN), CI(NN), SDA(*)
33:      real*8 DINF, DEPS
34:      integer IKL(NN)
35: C
36: C      .... OUTPUT ....
37: C
38:      real*8 DI(NN)
39:      integer IKL2(NN)
40: C
41: C ----- CONSTANTS -----
42: C
43:      parameter( D333 = 0.3333333333333333333D0 )
44: C
45: C      .... PAI * 2 / 3
46: C DATA PAI23 / 2.09439510239319548D0 /
47: C
48: C      .... 2 / 9
49: C parameter( C29 = 0.2222222222222222222D0 )
50: C
51: C
52:      PAI23 = (2.0D0/3.0D0)*ACOS(-1.0D0)
53:      DEPS2 = DEPS**2
54:      DINFN = -DINF
55: C
56:      do 100 I = 1, NN
57: C
58: C-----
59: C      << EQUATION ASSUMED >>
60: C
61: C      X**2/B**2 + (SQRT(Y**2+Z**2)-A)**2/C**2
62: C      + D*X*(SQRT(Y**2+Z**2)-A) - 1 = 0
63: C
64: C      << MEANINGS OF SDA ARRAY >>
65: C

```

```

66: C          SDA(1:3)      : CENTER POSITION
67: C          SDA(4:6)      : ROTATION-AXIS DIRECTION (UNIT VECTOR)
68: C          SDA(7)        =  A
69: C          SDA(8)         =  (C/B)**2
70: C          SDA(9 )        =  D*C**2
71: C          SDA(10)        =  C**2
72: C=====
73: C
74: C          DX              =  X(I) - SDA(LS+1)
75: C          DY              =  Y(I) - SDA(LS+2)
76: C          DZ              =  Z(I) - SDA(LS+3)
77: C          ALP             =  SDA(LS+4)*DX + SDA(LS+5)*DY + SDA(LS+6)*DZ
78: C          BET             =  SDA(LS+4)*AI(I) + SDA(LS+5)*BI(I) + SDA(LS+6)*CI(I)
79: C          E1              =  BET**2
80: C          E2              =  1.0 - E1
81: C          E3              =  1.0/(E2+SDA(LS+8)*E1)
82: C          E5              =  2*(DX*AI(I)+DY*BI(I)+DZ*CI(I)-ALP*BET)
83: C          E6              =  DX**2 + DY**2 + DZ**2 - ALP**2
84: C          T1              =  (E5+2*ALP*BET*SDA(LS+8)-SDA(LS+7)*SDA(LS+9)*BET)*E3
85: C          T2              =  (E6+SDA(LS+8)*ALP**2+SDA(LS+7)**2-SDA(LS+10)
86: C          &              -SDA(LS+7)*SDA(LS+9)*ALP)*E3
87: C          TE1             =  BET*SDA(LS+9)*E3
88: C          TE2             =  (2*SDA(LS+7)-SDA(LS+9)*ALP)*E3
89: C          E4              =  2*TE1*TE2
90: C          ALP             =  TE1**2
91: C          BET             =  TE2**2
92: C-----
93: C          << COEFFICIENTS >>
94: C
95: C          2                                2
96: C          S  + T1*S + T2 = ( -TE1*S + TE2 )*SQRT(E2*S +E5*S+E6 )
97: C
98: C          SQUARE BOTH SIDES
99: C          4      3      2
100: C          A*S + B*S + C*S + D*S + E = 0
101: C
102: C-----
103: C          A              =  1.0/(1.0-ALP*E2)
104: C          B              =  (2.*T1-ALP*E5+E4*E2)*A
105: C          C              =  (T1**2+2.*T2-ALP*E6+E4*E5-BET*E2)*A
106: C          D              =  (2.*T1*T2+E4*E6-BET*E5)*A
107: C          E              =  (T2**2-BET*E6)*A
108: C
109: C-----
110: C          SOLVE A CUBIC EQUATION TO DECOMPOSE THE QUARTIC POLYNOMIAL
111: C          INTO PRODUCT OF TWO SQUARE POLYNOMIALS.
112: C          3      2
113: C          T  + P *T +  Q*T +  R  = 0
114: C
115: C          IF PARTICLES ARE JUST ON THE SURFACE, P=B,Q=C,R=D AND ASSUME E=0.
116: C
117: C-----
118: C          if ( IKL(I).eq.LS ) then
119: C              P          =  B
120: C              Q          =  C
121: C              R          =  D
122: C          else
123: C              P          =  -C
124: C              Q          =  B*D - 4*E
125: C              R          =  -(E*(B*B-4*C)+D*D)
126: C          end if
127: C
128: C          PP              =  (P*P*D333-Q)*D333
129: C          QQ              =  (R+D333*P*(C29*P*P-Q))*0.5
130: C

```

src/shared/sqsol1.f

```

131:      PP3      = PP**3
132:      QQ2      = QQ*QQ
133: C
134: C ... INITIALIZE SOLUTIONS .....
135:      X2      = DINFN
136:      X3      = DINFN
137:      X4      = DINFN
138: C
139:      if ( QQ2.ge.PP3 ) then
140:      S        = SQRT(QQ2-PP3)
141:      A1       = -QQ + S
142:      A2       = -QQ - S
143:      X1       = SIGN(1.0D0,A1)*(ABS(A1)**D333) + SIGN(1.0D0,A2)*
144:      & (ABS(A2)**D333) - P*D333
145: CCCC WRITE(6,*) ' CUBIC : X1 = ',X1, X1*(X1*(X1+P)+Q)+R
146:      else
147:      T        = SQRT(PP)
148:      S        = ACOS(-QQ/T**3)*D333
149:      X1       = 2.0*T*COS(S) - P*D333
150:      if ( IKL(I).eq.LS ) then
151:      X2       = 2.0*T*COS(PI/23-S) - P*D333
152:      X3       = 2.0*T*COS(PI/23+S) - P*D333
153:      end if
154:      end if
155:      if ( IKL(I).ne.LS ) then
156: C
157:      E1       = SQRT(MAX(0.0D0,4*(X1-C)+B*B))
158:      E2       = SQRT(MAX(0.0D0,X1*X1-4*E))*SIGN(1.0D0,E*X1-2*D)
159:      P1       = -(B+E1)*0.5
160:      Q1       = -(B-E1)*0.5
161:      P2       = (X1+E2)*0.5
162:      Q2       = (X1-E2)*0.5
163:      X1       = DINFN
164: C-----
165: C      PAIRS OF SOLUTIONS
166: C
167: C      2
168: C      X - P1*X + P2 = 0
169: C
170: C      2
171: C      X - Q1*X + Q2 = 0
172: C
173: C-----
174:      E1       = P1*P1 - 4.0*P2
175: C
176:      if ( E1.ge.DEPS2 ) then
177:      X3       = (P1+SQRT(E1))*0.5
178:      X4       = P1 - X3
179:      end if
180:      E2       = Q1*Q1 - 4.0*Q2
181: C
182:      if ( E2.ge.DEPS2 ) then
183:      X1       = (Q1+SQRT(E2))*0.5
184:      X2       = Q1 - X1
185:      end if
186:      end if
187: C
188: C-----
189: C ===== IMPROVE PRECISION OF ROOTS BY NEWTON METHOD =====
190: C      (ONLY-ONE-ITERATION)
191: C-----
192: C
193:      if ( X1.ne.DINFN ) then
194:      FD1      = X1*(X1*(X1*(X1+B)+C)+D) + E
195:      TAN      = X1*(X1*(X1*4+3*B)+2*C) + D

```

```

196:      if ( TAN.ne.0 ) X1 = X1 - FD1/TAN
197:      end if
198:      if ( X2.ne.DINFN ) then
199:      FD2      = X2*(X2*(X2*(X2+B)+C)+D) + E
200:      TAN      = X2*(X2*(X2*4+3*B)+2*C) + D
201:      if ( TAN.ne.0 ) X2 = X2 - FD2/TAN
202:      end if
203:      if ( X3.ne.DINFN ) then
204:      FD3      = X3*(X3*(X3*(X3+B)+C)+D) + E
205:      TAN      = X3*(X3*(X3*4+3*B)+2*C) + D
206:      if ( TAN.ne.0 ) X3 = X3 - FD3/TAN
207:      end if
208:      if ( X4.ne.DINFN ) then
209:      FD4      = X4*(X4*(X4*(X4+B)+C)+D) + E
210:      TAN      = X4*(X4*(X4*4+3*B)+2*C) + D
211:      if ( TAN.ne.0 ) X4 = X4 - FD4/TAN
212:      end if
213: C
214: C .... SELECT SMALLEST POSITIVE SOLUTION .....
215: C
216:      DD       = DINFN
217:      if ( X1.gt.0.0 ) DD = MIN(X1,DD)
218:      if ( X2.gt.0.0 ) DD = MIN(X2,DD)
219:      if ( X3.gt.0.0 ) DD = MIN(X3,DD)
220:      if ( X4.gt.0.0 ) DD = MIN(X4,DD)
221:      if ( DD.ne.DINF.and.DD.lt.DI(I) ) then
222:      IKL2(I) = LS
223:      DI(I)   = DD
224:      end if
225: 100 continue
226:      return
227:      end

```

src/shared/srcinf.f

```

1:      subroutine SRCINF( SRCSP, NLTEMP, NSRCSP, NSOUR, PROG, JDEBG,
2:      &                  JKPAR, KPLIM,
3:      &                  JNEUT, JPHOT, JFISS, JEIGN, JTIME, JUNDG,
4:      &                  MWVEC, MVSTK, NERR,
5:      &                  IDMAT, NMAT, NUCID, NUC,
6:      &                  NTGX, LFKAI, FKAI, IBSDA, NBODY, IPSDA,
7:      &                  IZNAM, NINPZ, KINPZ, NZONE, IWRK, SWRK, DWRK,
8:      &                  NWORK )
9:
10: C/#!/IF SOURCE(NEW)
11:
12: C==<MVP/GMVP>=====
13: C purpose: input a source set source information in $SOURCE data block
14: C called from: SRCINP
15: C calls: CATSTR CCOMP CDENTK CHREAD CNTERR DMREAD GETSTR LABEL PACK00
16: C        PACKCS PACKLB PACKND PCTCLS R4READ SMPINF
17: C=====
18: C
19: C arguments ( o=output, i=input, io=output/input, w=work, c=constant)
20: C
21: C io srcsp(*) : source information packed in this array
22: C c nltemp : temporary length of srcsp array
23: C o nsrcsp : real length of srcsp array determined after input
24: C
25: C io nsour : current & total number of source sets
26: C
27: C i prog : program name ( 'GMVP' or 'MVP' )
28: C i jfiss : flag for fission problem
29: C i jeign : flag for eigenvalue problem
30: C i jtime : flag for time dependent problem
31: C
32: C o mwvec: maximum number of working variables in sampling
33: C o mvstk: maximum of vector calculator stack depth
34: C
35: C o nerr: number of error
36: C
37: C i idmat(nmat) : material ID
38: C i nmat : number of material in problem ( prog='GMVP' )
39: C i nucid(nuc) : nuclide ID list ( prog='MVP' )
40: C i nuc : nuclide of nuclide in problem ( prog='MVP' )
41: C
42: C i ntgx : number of energy groups in cross section library
43: C i fkai(ntgx,nmat) : fission spectrum data (prog='GMVP')
44: C
45: C
46: C i ibsda(3,nbody) : first surface #'s & ID's of each body
47: C i nbody : number of bodies
48: C i ipsda(3,nsurf+1) : surface data pointers & belonging body ID's
49: C i iznam(ninpz) : names of input zones
50: C i ninpz : number of input zones
51: C i kinpz(nzone) : input zone # of each zone
52: C i nzone : number of each zones
53: C
54: C w iwrk(nwork),swrk(nwork)
55: C      working area for input & preprocessing (start on the
56: C      same address)
57: C w dwrk(nwork) : double word working area
58: C
59: C-----
60:
61:      integer SRCSP(NLTEMP)
62:      integer JKPAR(KPLIM)
63:      integer JDEBG(*)
64:      character*(*) PROG
65: C

```

```

66:      integer IDMAT(NMAT)
67:      character*16 NUCID(NUC)
68: C
69:      real FKAI(NTGX,NMAT)
70: C
71:      integer IBSDA(3,NBODY)
72:      integer IPSDA(3,*)
73:      character*12 IZNAM(NINPZ)
74:      integer KINPZ(NZONE)
75: C
76: C ... iwrk & swrk has same starting address !!
77: C      (dwrk is not)
78: C
79:      integer IWRK(NWORK)
80:      real SWRK(NWORK)
81:      real*8 DWRK(NWORK)
82: C
83: CCCC include 'INC/_ARRAY'
84: include 'INC/_WORDL'
85: include 'INC/_IUNIT'
86: C
87: Ccc real*8      cdentk
88: C
89: C ... local data ....
90: C
91:      parameter( NVECV = 10 )
92:      character*16 VECNM(NVECV)
93:      integer JSVEC(NVECV)
94: C
95: C ... position of x,y,z, ... in vecnm & jvecs
96: C
97:      parameter( IMX = 1, IMY = 2, IMZ = 3 )
98:      parameter( IMA = 4, IMB = 5, IMC = 6 )
99:      parameter( IME = 7, IMT = 8, IMW = 9 )
100:      parameter( IMG = 10 )
101: C
102: C === local working variables ===
103: C
104:      character*32 VNAME
105: CCCCC character*500 CBUFF
106:      character*2048 CBUFF
107:      character*8 TERM
108:      character*1 TC
109:      real*8 RATIO
110: C
111: C
112: C mxvec : allowable maximum of number of vector in one source set
113: C svecnm(i) : list of vector(variable) name
114: C jrvec(i,j) : relation flags between vector i & vector j
115: C      0 = no connection between vector i & j
116: C      1 = vector i & j has connection
117: C jrvec(i,i) is count of substitution (sampling) for vector i.
118: C ivref(i) : count of reference to vector i.
119: C ivlf(i) : working array used in 'SMPINF' routine..
120: C ivrg(i) : working array used in 'SMPINF' routine..
121: C
122: C
123: CCCC parameter( MXVEC = 24 )
124:      parameter( MXVEC = 96 )
125:      character*16 SVECNM(MXVEC)
126: C
127:      integer JRVEC(MXVEC,MXVEC)
128:      integer IVREF(MXVEC)
129:      integer IVLF(MXVEC)
130:      integer IVRG(MXVEC)

```

src/shared/srcinf.f

```

131: C
132: C
133: C ... 'ACCEPT' block # in i'th 'ACCEPT' nesting level.
134: C
135: C
136: C     parameter( MXACC      = 4 )
137: C     integer LACC(MXACC)
138: C     integer IWHEN(MXACC)
139: C     character*16 WHEN, WHEN2, SAVEN, KPART
140: C
141: C ... to save status ...
142: C
143: C     integer JRVEC0(MXVEC,MXVEC,MXACC)
144: C     integer IVREF0(MXVEC,MXACC)
145: C
146: C
147: C     character*12 GIVEN(0:1)
148: C
149: C ----- Data initialization -----
150: C
151: C
152: C .... name of vector variables sampled ....
153: C
154: C     data VECNM /'X', 'Y', 'Z', 'A', 'B', 'C', 'E', 'T', 'W', 'G'/
155: C
156: C     data CBUFF /' '/
157: C     data GIVEN /'NOT-GIVEN', 'GIVEN' /
158: C
159: C -----
160: C
161: C     NSOUR0 = NSOUR
162: C
163: C     if( nsour0 .gt. 0 ) then
164: C
165: C         NSOUR = 0
166: C
167: C         ... initialize SRCSP array as memory packet ...
168: C
169: C         call PACK00( SRCSP, 'SOURCE DATA(SRCINF)', NLTEMP, IRET )
170: C         if ( IRET.ne.0 ) then
171: C             NERR = NERR + 1
172: C             go to 280
173: C         end if
174: C
175: C         JENDS = 0
176: C
177: C ... mwvec: max number of working array ...
178: C ... mvstk: max of vector calculation stack depth
179: C
180: C     MWVEC = 0
181: C     MVSTK = 0
182: C
183: C     NERR = 0
184: C
185: C
186: C ==== input of each source set starts here ====
187: C
188: C
189: C     100 NSOUR = NSOUR + 1
190: C
191: C     NERR0 = NERR
192: C
193: C     write(VNAME,('SOURCE ',I3,'#')) NSOUR
194: C     K = INDEX(VNAME,'#') - 1
195: C
196: C
197: C     call LABEL( IPR, VNAME(:K) )
198: C
199: C     NDATA = 0
200: C     ISTYP = 0
201: C     IPKIND = 0
202: C     RATIO = -1.0D0
203: C     ISCID = 0
204: C
205: C     NVECS = 0
206: C
207: C ... initialize vector (variable) name list etc. ...
208: C
209: C     do 120 I = 1, MXVEC
210: C         SVECNM(I) = ' '
211: C         IVREF(I) = 0
212: C         do 110 J = 1, MXVEC
213: C             JRVEC(J,I) = 0
214: C         110 continue
215: C     120 continue
216: C
217: C ... iacc : current nesting depth of 'ACCEPT' block
218: C ... nacc : appearance # of 'ACCEPT' block
219: C ... macc : maximum 'ACCEPT' block nesting depth
220: C
221: C     IACC = 0
222: C     NACC = 0
223: C     MACC = 0
224: C     TC = ' '
225: C
226: C
227: C === PUT label to mark sampling information ===
228: C
229: C     write(VNAME,('SAMPLING',I5)) NSOUR
230: C
231: C     call PACKLB( SRCSP, 'SOURCE SET HEADER', VNAME(1:13), IRET )
232: C     if ( IRET.ne.0 ) then
233: C         NERR = NERR + 1
234: C         go to 280
235: C     end if
236: C
237: C     7000 format(1X,' COMMAND>' ,A:/(1X,16X,A:))
238: C     7020 format(1X,' DATA >' ,A:/(1X,16X,A:))
239: C     7040 format(1X,' SAMPLE >' ,A:/(1X,16X,A:))
240: C     7060 format(1X,' LABEL >' ,A,':')
241: C     7080 format(1X,' ACCEPT >' ,A:/(1X,16X,A:))
242: C
243: C     7100 format(2X,'XXX <SOURCE INFORMATION ERROR >' /5X,
244: C         & ' UNRECOGNIZABLE : <',A,'>' /)
245: C     7120 format(2X,'XXX <SOURCE INFORMATION ERROR >' /5X,
246: C         & ' UNRECOGNIZABLE : <',A,A,'>' /)
247: C     7140 format(2X,'XXX <SOURCE INFORMATION ERROR >' /(:5X,' ',A)/)
248: C     7160 format(2X,'XXX <SOURCE INFORMATION ERROR >' /(:5X,' ',A)/)
249: C     7180 format(/2X,'XXX <SOURCE INFORMATION ERROR >' /5X,
250: C         & ' SAMPLING INFORMATION FOR ',A,' IS INCOMPLETE' /
251: C         & (:5X,' VARIABLE <',A,'> ',A))
252: C     7200 format(2X,'XXX <SOURCE INFORMATION ERROR> ',
253: C         & ' CHARACTER BUFFER OVERFLOW')
254: C     7220 format(2X,'XXX <SOURCE INFORMATION ERROR >' /5X,
255: C         & ' TOO DEEP "ACCEPT" BLOCK NEST: ',I3/)
256: C     7240 format(2X,'XXX <SAMPLING INFORMATION ERROR> ',
257: C         & ' TOO MANY VECTOR VARIABLES APPEARED. (MAX=',I3,')')
258: C
259: C
260: C ==== input loop ====

```

src/shared/srcinf.f

```

261: C
262: C
263: C 130 if ( TC.eq.'&' ) go to 230
264: C
265: C      TERM      = '':(=&'
266: C      call CHREAD( 'SOURCE', VNAME, TERM(:5), NLEN, IT, JENDS )
267: C
268: C      TC        = ' '
269: C      if ( IT.ne.0 ) TC      = TERM(IT:IT)
270: C
271: C      ... skip ';' ( end of a statement )
272: C
273: C      if( tc.eq.';' ) goto 1000
274: C
275: C      ... nlen = 0 is '&' or error ....
276: C
277: C      if ( NLEN.eq.0 ) then
278: C        if ( TC.eq.'&' ) then
279: C          go to 230
280: C        else if ( TC.eq.';' ) then
281: C          go to 130
282: C        else
283: C          write(IMG,7100) TC
284: C          NERR      = NERR + 1
285: C          if ( NERR.eq.0 .or. JENDS.eq.0 ) go to 130
286: C          go to 280
287: C        end if
288: C      end if
289: C
290: C      ... end of source data ....
291: C
292: C      if ( VNAME(:NLEN).eq.'$END' ) then
293: C        JENDS      = 1
294: C        go to 230
295: C      end if
296: C
297: C
298: C      == @vector = or @(v1 v2 ... ) =====
299: C
300: C      or WHEN [=] ... ;
301: C
302: C      if ( TC.eq.'=' .or. TC.eq.'(' .and. VNAME(:NLEN).eq.'&'
303: C      & .or. TC.eq.' ' .and. VNAME(:NLEN).eq.'WHEN' ) then
304: C
305: C        IWADD      = 0
306: C        if ( VNAME(:NLEN).eq.'WHEN' ) then
307: C          if ( IACC.eq.0 ) then
308: C            write(IMG,7140) '"WHEN" appeared out of "ACCEPT" block'
309: C            call GETSTR( 'SKIP', CBUF, ';', NLL, 2, IT, IERR, IRET )
310: C            write(IMG,7160) 'IGNORED: ', CBUF(:NLL)
311: C            NERR      = NERR + 1
312: C
313: C          ... 'WHEN' command is replaced by '@WHENi = condition;'
314: C
315: C          else
316: C            IWHEN(IACC) = IWHEN(IACC) + 1
317: C            VNAME(NLEN+1:) = ' '
318: C            write(VNAME(NLEN+1:),'(i2)') IACC
319: C            call CCOMP( VNAME, LEN(VNAME), VNAME, IFFL )
320: C            NLEN      = ICLEN(VNAME)
321: C            if ( IWHEN(IACC).eq.1 ) then
322: C              WHEN      = VNAME(:NLEN)
323: C            else
324: C              write(VNAME(NLEN+1:),'(''Z'',i2)') IWHEN(IACC)
325: C              call CCOMP( VNAME, LEN(VNAME), VNAME, IFFL )

```

```

326: C      NLEN      = ICLEN(VNAME)
327: C      WHEN2     = VNAME(:NLEN)
328: C      IWADD     = 1
329: C      end if
330: C      end if
331: C      end if
332: C
333: C      call SMPINF( SRCSP,PROG,JUNDG,JFISS,JDEBG,NSOUR,SVECNM, JRVEC,
334: C      &      IVPREF, MXVEC, NVECS, MVSTK, VNAME(:NLEN), TC, NERR,
335: C      &      IDMAT, NMAT, NUCID, NUC, NTGX, LFKAI, FKAI, IBSDA,
336: C      &      NBODY, IPSDA, IZNAM, NINPZ, KINPZ, NZONE, CBUF, IVLF,
337: C      &      IVRG, IWRK, SWRK, DWRK, NWORK )
338: C
339: C      .... add @when = AND( @when, @when2 ) if necessary ...
340: C
341: C      if ( IWADD.eq.1 ) then
342: C        LCBUF      = 0
343: C        LL         = ICLEN(WHEN)
344: C        LL2        = ICLEN(WHEN2)
345: C
346: C        call CATST2( CBUF, LCBUF, '@', WHEN(:LL), IER )
347: C        call CATST2( CBUF, LCBUF, '=AND@', WHEN(:LL), IER )
348: C        call CATST2( CBUF, LCBUF, ',@', WHEN2(:LL2), IER )
349: C        call CATSTR( CBUF, LCBUF, ')', IER )
350: C      ccc      call CATST2( CBUF, ICBUF, '=AND@', WHEN(:LL), IER )
351: C      ccc      call CATST2( CBUF, ICBUF, ',@', WHEN2(:LL2), IER )
352: C      ccc      call CATSTR( CBUF, ICBUF, ')', IER )
353: C
354: C      call CDENTK( CBUF(:LCBUF), IPR, IERR, VNAME, NSTMT, NPFL,
355: C      &      NSTKM, IVECN )
356: C
357: C      if ( IERR.ne.0 ) then
358: C        NERR      = NERR + 1
359: C      else
360: C        MVSTK      = MAX(MVSTK,NSTKM)
361: C      end if
362: C      C---<packing>-----
363: C      call PACKCS( SRCSP, 'WHEN AND WHEN', CBUF(:LCBUF), IRET )
364: C      if ( IRET.ne.0 ) NERR      = NERR + 1
365: C      C-----
366: C      end if
367: C
368: C
369: C      == ACCEPT ==
370: C
371: C      ... acceptance condition of sampled source starts ...
372: C
373: C
374: C      else if ( VNAME(:NLEN).eq.'ACCEPT' ) then
375: C
376: C        write(IPR,7080) VNAME(:NLEN)
377: C
378: C        IACC      = IACC + 1
379: C        if ( IACC.gt.MXACC ) then
380: C          write(IMG,7220) IACC
381: C          NERR      = NERR + 1
382: C          go to 280
383: C        else
384: C          MACC      = MAX(IACC,MACC)
385: C          NACC      = NACC + 1
386: C          LACC(IACC) = NACC
387: C          IWHEN(IACC) = 0
388: C
389: C          ... save connection status of vectors at the beginnig of
390: C          'ACCEPT' block.

```

src/shared/srcinf.f

```

391: C
392:       do 150 I = 1, MXVEC
393:           IVREF0(I,IACC) = IVREF(I)
394:           do 140 J = 1, MXVEC
395:               JRVEC0(I,J,IACC) = JRVEC(I,J)
396:           140       continue
397:       150       continue
398: C
399:       write(CBUFF(:16),(''ACCEPT'',2i5')) IACC(IACC), NSOUR
400: C
401: C---<packing>-----
402:       call PACKLB( SRCSP, 'ACCEPT', CBUFF(:16), IRET )
403:       if ( IRET.ne.0 ) NERR = NERR + 1
404:       call PACKCS( SRCSP, 'NOP', 'NOP', IRET )
405:       if ( IRET.ne.0 ) NERR = NERR + 1
406: C-----
407: C
408:       end if
409: C
410: C
411: C === END-ACCEPT ===
412: C
413: C
414: C ... terminate 'ACCEPT' mode
415: C
416:       else if ( VNAME(:NLEN).eq.'END-ACCEPT' ) then
417:
418:           write(IPR,7080) VNAME(:NLEN)
419:
420:           if ( IACC.eq.0 ) then
421:               write(IMG,7140) 'Unexpected "END-ACCEPT"'
422:               NERR = NERR + 1
423:           else if ( IWHEN(IACC).eq.0 ) then
424:               write(IMG,7140) 'No "WHEN" condition in this "ACCEPT" block'
425:               NERR = NERR + 1
426:               IACC = IACC - 1
427:           else
428: C
429: C
430: C ... add commands for vector compression/reordering
431: C
432: C
433: C ... sampling in this block if jrvec(i,i) > jrvec0(i,i,iacc)
434: C
435:       LCBUF = 0
436: C
437:       LL = ICLEN(WHEN)
438:
439:       call CATST2( CBUFF, LCBUF, '@(', WHEN(:LL), IER )
440: C
441:       do 160 I = 1, NVECS
442:           if ( SVECNM(I).eq.WHEN(:LL) ) go to 170
443:       160       continue
444:           NVECS = NVECS + 1
445:           if ( NVECS.gt.MXVEC ) then
446:               write(IMG,7240) MXVEC
447:               NERR = NERR + 1
448:               go to 280
449:           end if
450:           SVECNM(NVECS) = WHEN(:LL)
451:       170       continue
452: C
453:       ICOMP = 0
454:       do 180 I = 1, NVECS
455:

```

```

456:           if ( JRVEC(I,I).gt.JRVEC0(I,I,IACC)
457:               & .and.SVECNM(I)(1:4).ne.'WHEN'
458:               & .and.SVECNM(I)(1:4).ne.'SAVE' ) then
459:
460:               LSVC = ICLEN(SVECNM(I))
461:
462:               call CATST2( CBUFF, LCBUF, ' ', SVECNM(I)(:LSVC), IERR
463:                   & )
464:
465:               if ( IERR.ne.0 ) then
466:                   write(IMG,7200)
467:                   NERR = NERR + 1
468:               end if
469:               ICOMP = ICOMP + 1
470:
471:               end if
472:       180       continue
473:       if ( ICOMP.gt.0 ) then
474:           call CATSTR( CBUFF, LCBUF, ' ) = ', IERR )
475:           if ( IERR.ne.0 ) then
476:               write(IMG,7200)
477:               NERR = NERR + 1
478:           end if
479: C---<packing>-----
480:       call PACKCS( SRCSP, 'COMPRESS', 'FUNCTION', IRET )
481:       if ( IRET.ne.0 ) NERR = NERR + 1
482:       call PACKCS( SRCSP, 'COMPRESS', CBUFF(:LCBUF), IRET )
483:       if ( IRET.ne.0 ) NERR = NERR + 1
484:       call PACKCS( SRCSP, 'COMPRESS', 'COMPRESS', IRET )
485:       if ( IRET.ne.0 ) NERR = NERR + 1
486: C-----
487:       else
488:           write(IMG,7140) 'No sampling in this "ACCEPT" block'
489:           NERR = NERR + 1
490:       end if
491: C
492: C
493: C .... Reorder vector for which connected vectors are sampled or
494: C referenced in 'ACCEPT' block but the vector itself is not sampled.
495: C
496: C
497:       LCBUF = 0
498:       call CATST2( CBUFF, LCBUF, '@(', WHEN(:LL), IER )
499:       if ( IER.ne.0 ) then
500:           write(IMG,7200)
501:           NERR = NERR + 1
502:       end if
503:       VNAME = ' '
504:       write(SAVEN,(''SAVE'',i3')) IACC
505:       call CCOMP( SAVEN, LEN(SAVEN), SAVEN, IFFL )
506:       L2 = ICLEN(SAVEN)
507:       call CATST2( CBUFF, LCBUF, ' ', SAVEN(:L2), IER )
508:       if ( IER.ne.0 ) then
509:           write(IMG,7200)
510:           NERR = NERR + 1
511:       end if
512:
513:       IRORD = 0
514:       do 200 I = 1, NVECS
515:           if ( JRVEC(I,I).gt.JRVEC0(I,I,IACC)
516:               & .or. SVECNM(I)(:4).eq.'WHEN'
517:               & .or. SVECNM(I)(:4).eq.'SAVE' ) go to 200
518:           do 190 J = 1, NVECS
519: C
520:               if ( JRVEC(I,J).ne.0.and.

```

src/shared/srcinf.f

```

521:      &          (JRVEC(J,J).gt.JRVEC0(J,J,IACC)
522:      &          .or.IVREF(J).gt.IVREF0(J,IACC)) ) then
523: C
524:          LSVC      = ICLN(SVECNM(J))
525:          call CATST2( CBUFF, LCBUF, ' ', SVECNM(J):(LSVC),
526:          &          IERR )
527:          if ( IERR.ne.0 ) then
528:              write(IMG,7200)
529:              NERR      = NERR + 1
530:          end if
531:          IRORD      = IRORD + 1
532:          go to 200
533:      end if
534: 190      continue
535: 200      continue
536: C
537: C ... add command for vector reordering
538: C
539:      if ( IRORD.gt.0 ) then
540:          do 210 I = 1, NVECS
541:              if ( SVECNM(I).eq.SAVEN(:L2) ) go to 220
542: 210      continue
543:              NVECS      = NVECS + 1
544:              if ( NVECS.gt.MXVEC ) then
545:                  write(IMG,7240) MXVEC
546:                  NERR      = NERR + 1
547:                  go to 280
548:              end if
549:              SVECNM(NVECS) = SAVEN(:L2)
550:              continue
551: C
552:          call CATSTR( CBUFF, LCBUF, ' ) = ', IERR )
553:          if ( IERR.ne.0 ) then
554:              write(IMG,7200)
555:              NERR      = NERR + 1
556:          end if
557: C-----<packing>-----
558:          call PACKCS( SRCSP, 'REORDER', 'FUNCTION', IRET )
559:          if ( IRET.ne.0 ) NERR      = NERR + 1
560:          call PACKCS( SRCSP, 'REORDER', CBUFF(:LCBUF), IRET )
561:          if ( IRET.ne.0 ) NERR      = NERR + 1
562:          call PACKCS( SRCSP, 'REORDER', 'REORDER', IRET )
563:          if ( IRET.ne.0 ) NERR      = NERR + 1
564: C-----
565:      end if
566: C
567:      write(CBUFF,('END-ACCEPT',2i5)) LACC(IACC), NSOUR
568: C
569: C-----<packing>-----
570:      call PACKLB( SRCSP, 'END-ACCEPT', CBUFF(:20), IRET )
571:      if ( IRET.ne.0 ) NERR      = NERR + 1
572: C-----
573:
574:      IACC      = IACC - 1
575:      end if
576: C
577: C
578: C === label ===
579: C
580: C
581:      else if ( TC.eq.' ' ) then
582:          write(IPR,7060) VNAME(:NLEN)
583: C
584: C-----<packing>-----
585:      call PACKLB( SRCSP, 'LABEL', VNAME(1:NLEN), IRET )

```

```

586:          if ( IRET.ne.0 ) NERR      = NERR + 1
587: C-----
588: C
589:          if ( IRET.ne.0 ) then
590:              NERR      = NERR + 1
591:              go to 280
592:          end if
593: C
594: C
595: C ... PARTICLE(...) ...
596: C
597:      else if ( TC.eq.'(' .and.(VNAME(1:NLEN).eq.'PARTICLE'
598:      &          .or.VNAME(1:NLEN).eq.'PART') ) then
599:          call CHREAD( ' ', KPART, ' )', IL, IT, IEND )
600:          if ( IEND.ne.0 ) then
601:              write(IMG,('1x,a,a,a')) 'XXX(SRCINF) No data after "',
602:              &          VNAME(1:NLEN),' "'
603:              NERR      = NERR + 1
604:              go to 280
605:          end if
606:          call KPSYMB( KPART(:IL), '>', IPKIND, 0)
607: C
608:          if ( IPKIND.eq.0 ) then
609:              write(IMG,*) 'XXX(SRCINF) Unsupported particle species : ',
610:              &          KPART(:IL)
611:              NERR      = NERR + 1
612:              IPKIND = 1
613:          else if ( JKPAR(IPKIND).eq.0 ) then
614:              write(IMG,*) 'XXX(SRCINF) Supported but unused particle',
615:              &          ' species : ', KPART(:IL)
616:              NERR      = NERR + 1
617:              IPKIND = 1
618:          end if
619: C
620: C ... neither vname(...) nor vname = ...
621: C
622: C
623:      else if ( TC.ne.'(' .and.TC.ne.'=' ) then
624: C
625:          if ( NACC.eq.0.and.VNAME(:NLEN).ne.'ACCEPT' ) then
626: C
627: C === KEYWORD before 'ACCEPT' statement ===
628: C
629:              write(IPR,7000) VNAME(:NLEN)
630: C
631: C ... vname = NEUTRON, PHOTON or ELECTORN
632: C
633:          if ( IPKIND.eq.0 ) then
634: C
635:              if ( VNAME(:NLEN).eq.'NEUTRON' ) then
636: C
637:                  IPKIND = 2**0
638: C
639:              else if ( VNAME(:NLEN).eq.'PHOTON' ) then
640: C
641:                  IPKIND = 2**1
642: C
643:              else if ( VNAME(:NLEN).eq.'ELECTRON' ) then
644: C
645:                  IPKIND = 2**2
646: C
647:              else
648:                  NERR      = NERR + 1
649:                  write(IMG,7100) VNAME(:NLEN)
650:                  end if
651:
652:              call KPSYMB( VNAME(:NLEN), '>', IPKIND, 0)
653:              if ( IPKIND.eq.0 ) then
654:                  write(IMG,*)
655:                  &          'XXX(SRCINF) Unsupported or currently unused',
656:                  &          ' particle species : ', KPART(:IL)
657:                  NERR      = NERR + 1

```

src/shared/srcinf.f

```

651:      IPKIND = 1
652:    end if
653:    if ( IPKIND.eq.0 ) then
654:      write(IMG,*)
655:      &      'XXX(SRCINF) Unsupported particle species : ',
656:      &      VNAME(:NLEN)
657:      NERR = NERR + 1
658:      IPKIND = 1
659:    else if ( JKPAR(IPKIND).eq.0 ) then
660:      write(IMG,*)
661:      &      'XXX(SRCINF) Supported but unused particle',
662:      &      ' species : ',VNAME(:NLEN)
663:      NERR = NERR + 1
664:      IPKIND = 1
665:    end if
666:  end if
667: end if
668: C
669: C === DATA( VAL ) ===
670: C
671: C
672:   else if ( TC.eq.'('.and.VNAME(1:1).ne.'@' ) then
673:
674:     if ( VNAME(:NLEN).eq.'RATIO' ) then
675: C
676: C ... source intensity ratio ...
677: C
678: ccccccccc call R4READ( ' ', RATIO, NA, 1, IEND0 )
679: call R8READ( ' ', RATIO, NA, 1, IEND0 )
680: if ( RATIO.lt.0.0D0 ) then
681:   write(IMG,*) 'XXX <SOURCE INFORMATION ERROR> ',
682:   &   'Invalid source generation RATIO ', RATIO
683:   NERR = NERR + 1
684: end if
685: CBUFF(1:32) = ' '
686: c##<2007/03/14:PN3:
687: c## write(CBUFF(1:32),'(''RATIO(''E12.5,'')'') ) RATIO
688: write(CBUFF(1:32),'(''RATIO(''1P,E12.5,'')'') ) RATIO
689: c##>
690: write(IPR,7020) CBUFF(1:32)
691:
692:   else if ( VNAME(:NLEN).eq.'ID' ) then
693: C
694: C ... source set ID (any positive integer) ...
695: C
696:   call I4READ( ' ', ISCID, NA, 1, IEND0 )
697:   if ( ISCID.lt.0 ) then
698:     write(IMG,*) ' <SOURCE INFORMATION ERROR> ',
699:     &   ' Source id must be non negative :', ISCID
700:     NERR = NERR + 1
701:   end if
702:   CBUFF(1:32) = ' '
703:   write(CBUFF(1:32),'(''ID(''I8,'')'') ) ISCID
704:   write(IPR,7020) CBUFF(1:32)
705:
706:   else
707:     call DMREAD( 'INVALID', N1, N2, IRET )
708:     write(IMG,7120) VNAME(:NLEN), '( ... )'
709:   end if
710: C
711: C .... error ...
712: C
713:   else
714:     write(IMG,7120) VNAME(:NLEN), '( ... )'
715:   end if

```

```

716: C
717:   if ( TC.eq.'&' ) then
718:     go to 230
719:   else
720:     go to 130
721:   end if
722: C
723: C
724: C
725: C === post proessing for a source set ===
726: C
727: C
728: C
729:   230 if ( JENDS.ne.0.and.NSOUR.eq.0 ) then
730:     write(IMG,*) 'XXX Unexpected end of input_data in $SOURCE',
731:     &   ' data-block !!'
732:     NERR = NERR + 1
733:     go to 280
734:   end if
735: C
736:   write(IPR,'(/3x,' ' === SAMPLING INFORMATION DIAGNOSTICS ==='//') )
737: C
738: C
739:   if ( IACC.eq.1 ) then
740:     write(IMG,*) 'XXX Unclosed "ACCEPT" block'
741:     NERR = NERR + 1
742:     go to 280
743:   end if
744: C
745: C .... check variables sampled ....
746: C
747:   do 240 I = 1, NVECV
748:     JSVEC(I) = 0
749:   240 continue
750: C
751:   LCBUF1 = 0
752:   LCBUF2 = 0
753:   CBUFF = ' '
754: C
755:   NVECV1 = 0
756: C
757:   do 270 I = 1, NVECS
758:     do 250 J = 1, NVECV
759:       if ( VECNM(J).eq.SVECNM(I) ) then
760:         NVECV1 = NVECV1 + 1
761:         JSVEC(J) = 1
762:         LVN = ICLEN(VECNM(J))
763:         write(IPR,'(5x,' 'Source variable <'',a,' ' is sampled'')' )
764:         &   VECNM(J) (:LVN)
765:         call CATST2( CBUFF(:LEN(CBUFF)/2), LCBUF1,
766:         &   VECNM(J) (:LVN), ' ', IERR )
767:         if ( IERR.ne.0 ) then
768:           write(IMG,7200)
769:           NERR = NERR + 1
770:         end if
771:         go to 260
772:       end if
773:     250 continue
774: C
775:     LVN = ICLEN(SVECNM(I))
776:     write(IPR,'(5x,' 'Working variable <'',A,' ' is used'')' )
777:     &   SVECNM(I) (:LVN)
778:     call CATST2( CBUFF(LEN(CBUFF)/2+1:), LCBUF2, SVECNM(I) (:LVN),
779:     &   ' ', IERR )
780: C

```


src/shared/srcinf.f

```

781: 260 if ( IERR.ne.0 ) then
782:     write(IMG,7200)
783:     NERR = NERR + 1
784: end if
785: 270 continue
786: C
787:     MWVEC = MAX(MWVEC,NVECS-NVECV1)
788: C
789: C .... add default sampling for unsampled variables when possible ....
790: C
791: C
792: C <<< POSITION >>>>
793: C
794:     if ( JSVEC(IMX).eq.0 .or. JSVEC(IMY).eq.0 .or. JSVEC(IMZ).eq.0 )
795:     & then
796:         write(IPR,7180) 'POSITION', 'X', GIVEN(JSVEC(IMX)), 'Y',
797:         & GIVEN(JSVEC(IMY)), 'Z', GIVEN(JSVEC(IMZ))
798:         NERR = NERR + 1
799:     end if
800: C
801: C
802: C <<< DIRECTION >>>>
803: C
804: C
805:     if ( JSVEC(IMA).eq.0 .or. JSVEC(IMB).eq.0 .or. JSVEC(IMC).eq.0 )
806:     & then
807: C
808:         if ( JSVEC(IMA)+JSVEC(IMB)+JSVEC(IMC).gt.0 ) then
809:             write(IMG,7180) 'Direction', 'A', GIVEN(JSVEC(IMA)), 'B',
810:             & GIVEN(JSVEC(IMB)), 'C', GIVEN(JSVEC(IMC))
811:             NERR = NERR + 1
812:         else
813:             write(IPR,'(lx,
814:             & '<<MESSAGE>> NO DIRECTION SAMPLING INFORMATION.'//
815:             & 'SAMPLED ISOTROPICALLY.'//)')
816:             call CNTERR( 'MESSAGE' )
817: C----- pack -----
818: C
819:             call PACKCS( SRCSP, 'DIRECTION', 'FUNCTION', IRET1 )
820:             call PACKCS( SRCSP, 'DIRECTION', '@( A B C ) =', IRET1 )
821:             call PACKCS( SRCSP, 'DIRECTION', 'ISOTROPIC', IRET1 )
822:             DWRK(1) = -1.0D0
823:             DWRK(2) = 1.0D0
824:             call PACKND( SRCSP, 'COSINES', 'R8', DWRK, 2, IRET2 )
825:             if ( IRET1.ne.0 .or. IRET2.ne.0 ) NERR = NERR + 1
826: C-----
827: C
828:             call CATSTR( CBUF(:LEN(CBUF)/2), LCBUF1, ' A B C ', IERR )
829:             if ( IERR.ne.0 ) then
830:                 write(IMG,7200)
831:                 NERR = NERR + 1
832:             end if
833: C
834:         end if
835:     end if
836: C
837: C <<< ENERGY OR ENERGY GROUP >>>>
838: C
839:     if ( JSVEC(IME).eq.0.and.JSVEC(IMG).eq.0 ) then
840:         write(IMG,7180) 'Energy or enrgy group'
841:         NERR = NERR + 1
842:     else if ( JSVEC(IMG).ne.0.and.JSVEC(IME).ne.0 ) then
843:         write(IMG,7180)
844:         & 'Sampling both energy (@E) and enrgy group (@G).'
845:         NERR = NERR + 1

```

```

846:     end if
847: C
848:     if ( PROG.eq.'MVP'.and.JSVEC(IMG).ne.0.and.JSVEC(IME).eq.0 ) then
849:         write(IMG,*)
850:         & ' !!! Sampling of energy groups may not work well',
851:         & ' because "ENERGY GROUP" is not a native attribute',
852:         & ' of continuous enrgy cross seciton libraries of',
853:         & ' <MVP> code, but appears only as "ENERGY BIN" for',
854:         & ' tallies!!'
855:         call CNTERR( 'WARNING' )
856:     end if
857: C
858: C <<< TIME >>>>
859: C
860:     if ( JTIME.ne.0.and.JSVEC(IMT).eq.0 ) then
861:         write(IPR,'(lx,<<MESSAGE>> No time sampling '//
862:         & 'information. Sampled as starting at 0.0 sec'//)')
863:         call CNTERR( 'MESSAGE' )
864: C
865:         call CDENTK( '@T=0.0D0', IPR, IERR, VNAME, NSTMT, NPFL, NSTKM,
866:         & IVECN )
867: C
868:         if ( IERR.ne.0 ) then
869:             NERR = NERR + 1
870:         else
871:             MVSTK = MAX(MVSTK,NSTKM)
872:         end if
873: C
874: C----- pack -----
875: C
876:         call PACKCS( SRCSP, 'TIME', '@T=0.0D0', IRET )
877:         if ( IRET.ne.0 ) NERR = NERR + 1
878: C-----
879: C
880:         call CATSTR( CBUF(:LEN(CBUF)/2), LCBUF1, ' T ', IERR )
881:         if ( IERR.ne.0 ) then
882:             write(IMG,7200)
883:             NERR = NERR + 1
884:         end if
885: C
886: C <<< weight >>>>
887: C
888:     if ( JSVEC(IMW).eq.0 ) then
889:         write(IPR,'(lx,<<MESSAGE>> Weight sampled to be 1.0'//)')
890:         call CNTERR( 'MESSAGE' )
891: C
892:         call CDENTK( '@W=1.0D0', IPR, IERR, VNAME, NSTMT, NPFL, NSTKM,
893:         & IVECN )
894:         if ( IERR.ne.0 ) then
895:             NERR = NERR + 1
896:         else
897:             MVSTK = MAX(MVSTK,NSTKM)
898:         end if
899: C
900: C----- pack -----
901: C
902:         call PACKCS( SRCSP, 'TIME', '@W = 1.0', IRET )
903:         if ( IRET.ne.0 ) NERR = NERR + 1
904: C-----
905: C
906:         call CATSTR( CBUF(:LEN(CBUF)/2), LCBUF1, ' W ', IERR )
907:         if ( IERR.ne.0 ) then
908:             write(IMG,7200)
909:             NERR = NERR + 1
910:         end if

```

src/shared/srcinf.f

```

911:      end if
912: C
913: C----- put label before source specification -----
914:      write(VNAME,(' '&'',I5,' 'SPEC')) NSOUR
915:      call PACKLB( SRCSP, 'SOURCE SET SPEC', VNAME(1:10), IRET )
916:      if ( IRET.ne.0 ) NERR = NERR + 1
917: C-----
918: C
919:      IWRK(1) = ISTYP
920: C
921:      if ( IPKIND.eq.0 ) then
922: CCCC      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
923:      if ( JNEUT.ne.0.and.JPHOT.ne.0 ) then
924:          write(IMG,7140)
925:          &      'Particle type not specified (NEUTRON/PHOTON)'
926:          NERR = NERR + 1
927:      else if ( JNEUT.ne.0 ) then
928: CCCCCCCCC IPKIND = 2**0
929:          call KPSYMB( 'NEUTRON','>',IPKIND,0)
930:      else if ( JPHOT.ne.0 ) then
931: CCCCCCCCC IPKIND = 2**1
932:          call KPSYMB( 'PHOTON','>',IPKIND,0)
933:      end if
934:      end if
935:      IWRK(2) = IPKIND
936: C
937: ccccc SWRK(3) = RATIO
938:      if ( RATIO.lt.0.0D0 ) then
939:          write(IMG,7140) 'No valid source generation RATIO was given.'
940:          NERR = NERR + 1
941:      end if
942: C
943: C      if ( ISCID.eq.0 ) then
944: C          ISCID = NSOUR
945: C          write(IPR,
946: C              end if
947: C
948: C-----
949:      call PACKND( SRCSP, 'SPEC1-2', 'I4', IWRK, 2, IRET )
950:      if ( IRET.ne.0 ) NERR = NERR + 1
951: ccccc call PACKND( SRCSP, 'RATIO', 'R4', SWRK(3), 1, IRET )
952:      call PACKND( SRCSP, 'RATIO', 'R8', RATIO, 1, IRET )
953:      call PACKND( SRCSP, 'ID', 'I4', ISCID, 1, IRET )
954:      if ( IRET.ne.0 ) NERR = NERR + 1
955: C
956: C ... pack sampling variables & working vectors ....
957: C
958:      call PACKCS( SRCSP, 'VECTOR VARIABLE LIST', CBUF(:LCBUF1), IRET )
959:      if ( IRET.ne.0 ) NERR = NERR + 1
960:      call PACKCS( SRCSP, 'WORKING VARIABLE LIST',
961:          &      CBUF(LEN(CBUF)/2+1:LEN(CBUF)/2+LCBUF2), IRET )
962:      if ( IRET.ne.0 ) NERR = NERR + 1
963: C-----
964: C
965:      if ( NERR.gt.NERR0 ) then
966:          write(IMG,(' (/7x,' 'XXX',i4,' errors in this source set. '))
967:          &      NERR - NERR0
968:      end if
969: C
970:      if ( JENDS.eq.0 ) go to 100
971: C
972: C-----
973:      call PCTCLS( SRCSP, 'SOURCE DATA', NLTEMP, NSRCSP, IRET )
974:      if ( IRET.ne.0 ) NERR = NERR + 1
975: C-----
976: C
977:      280 continue
978:      if ( NERR.gt.0 ) then
979:          do 290 I = 1, NERR
980:              call CNTERR( 'FATAL' )
981:          290 continue
982:      end if
983:      return
984: C ... endif for SOURCE(NEW)
985: C/#ENDIF
986: C
987:      end

```

src/shared/srcinp.f

```

1:      subroutine SRCINP( A, H, CHA, LIMIT, LIMITL, LIMITC,
2:      &                  LSRCSP, NSRCSP, LSOUR, LIDSR, PROG,
3:      &                  JKPAR, KPLIM,
4:      &                  JFISS, JEIGN, JTIME,
5:      &                  JNEUT, JPHOT, JUNDG, JDEBG, MWVEC, MVSTK,
6:      &                  MXREJ, NERR,
7:      &                  IDMAT, NUCID, NUC,   NTGX,  LKSOUR, LISZON,
8:      &                  LPSPAC, LPENRG, LIFISM, LFKAI, LENGYB, LENGPB,
9:      &                  EINCD )
10:
11: C/IF SOURCE(NEW)
12:
13: C==<MVP/GMVP>=====
14: C purpose: input $SOURCE data block
15: C
16: C   Called after finding '$SOURCE' header in input data.
17: C
18: C called from: INTRO
19: C calls:  HEADER PRESRC RIUNIT RESIZE CPLIN I4READ REMAIN RWIND CNTERR
20: C         LABEL CHREAD PCIDMP GTSOUR SRCINF KEEP
21: C=====
22: C
23: C arguments ( i=input, o=output, w=work, c=constant )
24: C
25: C io A(*) : dynamic memory array (task shared)
26: C io H(*) : dynamic memory array (task local)
27: C io CHA(*) : character dynamic memory array (task shared)
28: C i LIMIT : effective size of A(*)
29: C i LIMITL : effective size of H(*)
30: C i LIMITC : effective size of CHA(*)
31: C
32: C o LSRCSP : pointer of source information in dynamic memory area
33: C io LSOUR : pointer of relative source intensity data
34: C io LIDSR : pointer of source set ID data
35: C
36: C i PROG : program name ( 'GMVP' or 'MVP' )
37: C
38: C i JKPAR(KPLIM) : particle species flag
39: C i KPLIM : size of JKPAR
40: C i JFISS : flag for fission problem
41: C i JEIGN : flag for eigenvalue problem
42: C i JTIME : flag for time dependent problem
43: C i JNEUT : flag for neutron problem
44: C i JPHOT : flag for photon problem
45: C i JDEBG : flag for debugging information printout
46: C
47: C o MWVEC : maximum number of working vector necessary in sampling
48: C o MVSTK : maximum vector stack depth for vector calculator.
49: C o MXREJ : maximum number of rejection in ACCEPT block.
50: C o NERR : number of errors
51: C
52: C i idmat : material ID's
53: C i nucid : nuclide ID's ( prog='MVP' )
54: C i nuc : number of nuclides ( prog='MVP' )
55: C i ntgx : number of energy groups in cross section library
56: C         ( prog='GMVP' )
57: C
58: C < for compatibility with old-type source input >
59: C
60: C i lksour : pointer to source type data ( ksour(nsour) )
61: C i liszon : pointer to source zone data ( iszon(2,nsour) )
62: C i lpspac : pointer to source specification data ( pspac(10,nsour) )
63: C i lpenrg : pointer to source energy distribution data
64: C         ( penrg(ngroup,nsour) )
65: C i lifism : pointer to initial fission source specification data

```

```

66: C         ( ifism(nsour) : mat # or nuclide # )
67: C i lfkai : pointer to multi group fission spectrum data (prog='GMVP')
68: C i lengyb : pointer to neutron energy group boundary
69: C i lengpb : pointer to photon energy group boundary (MVP)
70: C-----
71: C
72: C common data updated/referenced
73: C
74: C io nsour : number of source generation data sets
75: C i ngroup : number of energy group (GMVP) or energy bin (MVP)
76: C i ngp1 : number of neutron energy group (GMVP) or energy bin (MVP)
77: C i ngp2 : number of photon energy group (GMVP) or energy bin (MVP)
78: C i nmat : number of materials
79: C i libsda : pointer to first surface #'s & ID's of each body
80: C i nbod : number of bodies
81: C i liznam : pointer to names of input zones
82: C i ninpz : number of input zones
83: C i lkinpz : pointer to input zone # of each zone
84: C i nzone : number of each zones
85: C
86: C-----
87: C << Storage configuration of specification data of each source>>
88: C
89: C SRCSP(*) I4 : source specification data is packed in this array
90: C
91: C < Sampling information >
92: C
93: C Compiled in 'SMPINF' routine.
94: C
95: C < general information >
96: C
97: C 1. type of source
98: C
99: C 0 : old type ( might be obsolete )
100: C 1 : general form
101: C 2 : input from a file
102: C
103: C ... obsolete (Apr 2000)
104: CC 2. particle type (digit 1/0 )
105: CC
106: CC 2**0 : neutron
107: CC 2**1 : photon
108: CC 2**2 : electorn
109: C
110: C 2. particle type : parameter constants defined in INC/_KPIDS
111: C
112: C KPNEUT : neutron
113: C KPPHOT : photon
114: C KPELEC : electorn
115: C etc.
116: C
117: C 3. source generation ratio ( un-normalized )
118: C 4. list of sampled variables
119: C 5. list of working variables
120: C
121: C-----
122: C
123: C real A(*)
124: C real H(*)
125: C character*4 CHA(*)
126: C
127: C character*(*) PROG
128: C integer JDEBG(*)
129: C integer JKPAR(KPLIM)
130: C integer IDMAT(NMAT)

```

src/shared/srcinp.f

```

131:      character*16 NUCID(NUC)
132:      real          EINCD
133: C
134: CCCC include 'INC/_ARRAY'
135:      include 'INC/_WORDL'
136:      include 'INC/_SIZES'
137:      include 'INC/_PGEOM'
138:      include 'INC/_IOUNIT'
139: C
140: C === local working variables ===
141: C
142:      character*72 LINE
143:      character*32 VNAME
144:      character*4 TERM
145: C
146:      call GTLAST( LAST )
147: C
148:      call HEADER( IPR, 'SOURCE SAMPLING INFORMATION' )
149: C
150:      NERR = 0
151: C
152: C
153: C ..... store $SOURCE ... $END SOURCE in working file ...
154: C
155: C
156:      if ( NSOUR.eq.0 ) then
157: C
158:          call RWIND( IUG1 )
159:      100 call CPLIN( IOIN, LINE, NCHAR, IEND )
160:          if ( IEND.ne.0 ) then
161:              write(IMG,*) 'XXX UNEXPECTED END OF INPUT DATA IN ',
162:              & '$SOURCE DATA-BLOCK !!'
163:              call CNTERR( 'FATAL' )
164:              return
165:          end if
166:          write(IUG1,'(a)') LINE(:NCHAR)
167:          if ( LINE(1:4).ne.'$END' ) go to 100
168: C
169: C ..... nsour.ne.0 means duplicate source input or old-fashioned
170: C source information is already input.
171: C Prepare source information in current form on unit IUG1.
172: C
173: C else
174: C if ( LSOUR.ne.0 .or. LKSOUR.ne.0 .or. LPSPAC.ne.0
175: C & .or. LPENRG.ne.0 .or. LIFISM.ne.0 ) then
176: C
177: C ... write IUG1: (argumenst IUG1 and IPR removed in Jan 2000)
178: C
179:      call PRESRC( PROG, JNEUT, JPHOT, NSOUR, NGROUP,
180:      & NGP1, NGP2, NTGX, LSOUR, LKSOUR, LPSPAC, LPENRG,
181:      & LIFISM, A(LSOUR), A(LKSOUR), A(LPSPAC), A(LPENRG),
182:      & A(LIFISM), A(LFKAI), A(LENGYB), A(LENGPB), NUCID,
183:      & NUC, IDMAT, NMAT, EINCD, NERR )
184: C
185:      if ( NERR.gt.0 ) then
186:          write(IMG,*) 'XXX Cannot convert an old-type source',
187:          & ' information to a new type.'
188:          call CNTERR( 'FATAL' )
189:          return
190:      else
191:          write(IMG,'(/2x,a)')
192:      & '!!! Your source specification input is in old-fashioned style.',
193:      & ' Authors recommend new style printed as follows: '
194: C
195:      call CNTERR( 'WARNING' )

```

```

196:
197:      call LABEL( IPR,
198:      & 'YOUR SOURCE INPUT CONVERTED TO NEW STYLE' )
199: C
200:      call RWIND( IUG1 )
201:      LINE = ' '
202:      110 call CPLIN( IUG1, LINE, NCHAR, IEND )
203:          if ( IEND.eq.0 ) then
204:              write(IPR,'(5x,a)') LINE(:ICLEN2(LINE))
205:              if ( LINE(1:4).ne.'$END' ) go to 110
206:          end if
207:          call RWIND( IUG1 )
208:      end if
209:      end if
210:      end if
211: C
212:      call RWIND( IUG1 )
213: C
214:      call RIUNIT( IUG1 )
215: C
216: C
217: C ... default size of temporary working area ...
218: C
219: C
220:      MXREJ = 300
221:      NWORK = 4097
222: C
223:      120 TERM = '(&'
224:      call CHREAD( 'SOURCE', VNAME, TERM, NLEN, IT, JRET )
225: C
226: C
227: C
228: C === specifications commonly used in all sources are processed here...
229: C
230: C
231:      if ( NLEN.gt.0.and.VNAME(:NLEN).eq.'NWORK' ) then
232:          call I4READ( VNAME(:NLEN), NWORK, NA, 1, IEND0 )
233:          if ( NWORK.lt.0 ) then
234:              write(IMG,*)
235:              & 'XXX(SRCINP) Input for working are size (NWORK=,'
236:              & NWORK, ') is not positive !!'
237:              NWORK = 4096
238:              call CNTERR( 'FATAL' )
239:          end if
240: C
241:      else if ( NLEN.gt.0.and.VNAME(:NLEN).eq.'MXREJ' ) then
242:          call I4READ( VNAME(:NLEN), MXREJ, NA, 1, IEND0 )
243:          if ( MXREJ.lt.0 ) then
244:              write(IMG,*)
245:              & 'XXX(SRCINP) Input for max. count of rejrcction (MXREJ=,'
246:              & MXREJ, ') is not positive !!'
247:              MXREJ = 300
248:              call CNTERR( 'FATAL' )
249:          end if
250: C
251:      else if ( JRET.eq.1 ) then
252:          write(IMG,*) 'XXX(SRCINP) No source set information !!'
253:          call CNTERR( 'FATAL' )
254: C
255:          call RIUNIT( IOIN )
256:          return
257:      end if
258: C
259:      if ( IT.eq.0 .or. IT.ne.0.and.TERM(IT:IT).ne.'&' ) go to 120
260: C

```

src/shared/srcinp.f

```
261: C
262: C ==== input specification of each source starts or ends ...
263: C
264: C
265: C ... initialization before the first source ....
266: C
267: C
268: C      call KEEP( 'SWRK', LSWRK, NWORK, 'R4D', LAST, JDEBG )
269: C      call KEEP( 'DWRK', LDWRK, NWORK, 'R8', LAST, JDEBG )
270: C
271: C ... allocate all remaining array memory for SRCSP
272: C
273: C      call IMAIN( ' ', NLTEMP, 'R4D', LAST )
274: C      if ( NLTEMP.le.0 ) then
275: C          write(IMG,*) 'XXX No-more memory to get source information!!!'
276: C          call ERSTOP( 1, 'MEMORY OVER.' )
277: C          stop 999
278: C      end if
279: C      NLTEMP = NLTEMP - 1
280: C      call LKEEP( 'SRCSP', LSRCS, NLTEMP, 'R4D', LAST, JDEBG )
281: C
282: C
283: C ===== & ... & : source information set
284: C
285: C
286: C      call SRCINF( H(LSRCSP), NLTEMP, NSRCSP, NSOUR, PROG, JDEBG,
287: C      &          JKP, JKLIM,
288: C      &          JNEUT, JPHOT,
289: C      &          JFISS, JEIGN, JTIME, JUNDG,
290: C      &          MWVEC, MVSTK, NERR, IDMAT, NMAT, NUCID, NUC, NTGX,
291: C      &          LFKAI, A(LFKAI), A(LIBSDA), NBODY, A(LIPSDA),
292: C      &          CHA(LIZNAM), NINPZ, A(LKINPZ),
293: C      &          NZONE, A(LSWRK), A(LSWRK), A(LDWRK), NWORK )
294: C
295: C
296: C      call LRSIZE( 'SRCSP', LSRCS, NSRCSP, 'R4D', LAST )
297: C
298: C      call RIUNIT( IOIN )
299: C
300: C      if ( JDEBG(1).ne.0 ) then
301: C          call LABEL( IPR, 'CONTENTS OF SOURCE INFORMATION DATA PACKETS'
302: C          &
303: C          call PCTDMP( IPR, H(LSRCSP), H(LSRCSP), H(LSRCSP) )
304: C      end if
305: C
306: C
307: C ===== gather source ratio data in an array 'SOUR' ====
308: C
309: C      call KEEP( 'SOUR', LSOUR, NSOUR, 'R8', LAST, JDEBG )
310: C      call KEEP( 'IDSRC', LIDSRC, NSOUR, 'I4', LAST, JDEBG )
311: C
312: C ... argument IPR is removed (Jan 2000)
313: C      call GTSOUR( A(LSOUR), A(LIDSRC), H(LSRCSP), NSOUR )
314: C
315: C ... endif for SOURCE(NEW)
316: C/#ENDIF
317: C      return
318: C      end
```

src/shared/srecin.f

```

1:      subroutine SRECII( IUNIT, ITSK,  NAME, IA,   NA,   IERR )
2:      character*(*) NAME
3:      integer IA(NA)
4:      C
5:      MTYP      = 1
6:      call SRECIN( MTYP, IUNIT, ITSK, NAME, IA, DUMMY, DUMMY, NA, IERR )
7:      return
8:      end
9:      C .....
10:     subroutine SRECIR( IUNIT, ITSK,  NAME, RA,   NA,   IERR )
11:     character*(*) NAME
12:     real RA(NA)
13:     C
14:     MTYP      = 2
15:     call SRECIN( MTYP, IUNIT, ITSK, NAME, DUMMY, RA, DUMMY, NA, IERR )
16:     return
17:     end
18:     C .....
19:     subroutine SRECID( IUNIT, ITSK,  NAME, DA,   NA,   IERR )
20:     character*(*) NAME
21:     real*8 DA(NA)
22:     C
23:     MTYP      = 3
24:     call SRECIN( MTYP, IUNIT, ITSK, NAME, DUMMY, DUMMY, DA, NA, IERR )
25:     return
26:     end
27:     C .....
28:     CC
29:     subroutine SRECIN( MTYP, IUNIT, ITSK,  NAME, IA,   RA,   DA,
30:       &               NA,   IERR )
31:     C=====
32:     C purpose: input records combination including data type, size etc.
33:     C           and long data are splitted into more than one record.
34:     C
35:     C this routine is called from the follwing interface routines:
36:     C
37:     C   SRECII : integer array
38:     C   SRECIR : real array
39:     C   SRECID : double precision real array
40:     C
41:     C called in: RESTRT
42:     C-----
43:     C Arguments ( i=input, o=output, w=work )
44:     C
45:     C i mtyp : data type -- 1/2/3 = integer/real/real*8
46:     C i iunit : I/O unit to input data
47:     C i ITSK  : task ID ( from 1 to NTASK )
48:     C           In message passing multi task mode (PVM,MPI), ITSK may differ
49:     C           from that of running task (IDTASK), and data transfer occur
50:     C           in such a case;
51:     C
52:     C <PVM/MPI mode>
53:     C   ITSK=IDTASK=1 : read & store data on IA/RA/DA
54:     C   ITSK>1,IDTASK=1 : read & send input data to task ITSK.
55:     C   ITSK>1,IDTASK=ITSK : receive from task 1 & store data on IA/RA/DA
56:     C
57:     C           In other modes, ITSK should be equal to IDTASK.
58:     C This routine should be called from a mutex (locked) block in
59:     C parallel mode other than message passing.
60:     C
61:     C i name   : data name labeled on record.
62:     C o IA(NA),RA(NA),DA(NA) : array on which input data are stored.
63:     C i NA     : length of array.
64:     C o IERR   : non zero value is retured if some error has occurred.
65:     C

```

```

66: C Error code are sum of the following value;
67: C   IERR=1 : data type mismatch
68: C   IERR=2 : data length mismatch
69: C   IERR=4 : data name mismatch
70: C   IERR=8 : unexpected EOF
71: C   IERR=16 : I/O error
72: C   IERR=32 : message passing error
73: C=====
74: C   character*(*) NAME
75: C   integer IA(NA)
76: C   real RA(NA)
77: C   real*8 DA(NA)
78: C
79: C   include 'INC/_WORDL'
80: C   include 'INC/_IUNIT'
81: C
82: C/#IF PARA(PVM)
83: C   include 'INC/_PVM PARA'
84: C/#ELSEIF PARA(MPI)
85: C   * include 'mpif.h'
86: C/#ELSEIF PARA(VPP)
87: C   include 'INC/_VPP PARA'
88: C/#ENDIF
89: C   include 'INC/_TASKDT'
90: C
91: C ... data I/O buffer array (length MRBUF is defined in INC/_TASKDT)
92: C
93: C   integer IBUF(MRBUF)
94: C   real RBUF(MRBUF)
95: C   real*8 DBUF(MRBUF)
96: C   equivalence( IBUF(1),RBUF(1),DBUF(1) )
97: C
98: C   character LABEL*16
99: C
100: C-----
101: C
102: C check %%%%%%%%%
103: C   write(iow,*) '== srecin mtyp ',MTYP,' iunit ',IUNIT, ' itsk ',
104: C   & ITSK, ' name ', NAME, ' na ', NA
105: C %%%%%%%%%
106: C   IERR      = 0
107: C
108: C   IERR1      = 0
109: C   IERR2      = 0
110: C   IERR4      = 0
111: C   IERR8      = 0
112: C   IERR16     = 0
113: C   IERR32     = 0
114: C
115: C CCCC/#IF PARA( CRAY SX* VPP ) .OR. .NOT.PARA
116: C
117: C/#IF .NOT.PARA(PVM MPI)
118: C
119: C   read(IUNIT,end =140,err =160,iostat =IOS) LABEL, ITYP, NDATA, LBUF
120: C   if ( ITYP.ne.MTYP ) then
121: C     write(IMG,*) 'XXX(SRECIN) data type mismatch:'
122: C     write(IMG,*) '   on file: ', ITYP, ' expected: ', MTYP
123: C     IERR1      = 1
124: C     go to 180
125: C   end if
126: C   if ( NDATA.ne.NA ) then
127: C     write(IMG,*) 'XXX(SRECIN) data length mismatch:'
128: C     write(IMG,*) '   on file: ', NDATA, ' expected: ', NA
129: C     IERR2      = 2
130: C     go to 180

```

src/shared/srecin.f

```

131:      end if
132:      if ( LABEL.ne.NAME(:MIN(LEN(LABEL),LEN(NAME))) ) then
133:        write(IMG,*) 'XXX(SRECIN) label type mismatch:'
134:        write(IMG,*) '   on file:<', LABEL, '> expected:<', NAME, '>'
135:        IERR4 = 4
136:        go to 180
137:      end if
138: C
139:      do 100 NP = 1, NDATA, LBUF
140:        N1 = NP
141:        N2 = MIN(NDATA,N1+LBUF-1)
142:        if ( MTYP.eq.1 ) then
143:          read(IUNIT,end =150,err =170,iostat =IOS) (IA(I),I=N1,N2)
144:        else if ( MTYP.eq.2 ) then
145:          read(IUNIT,end =150,err =170,iostat =IOS) (RA(I),I=N1,N2)
146:        else if ( MTYP.eq.3 ) then
147:          read(IUNIT,end =150,err =170,iostat =IOS) (DA(I),I=N1,N2)
148:        end if
149:      100 continue
150: C
151: C
152: C/#ELSEIF PARA( PVM MPI )
153: C
154: C
155: C ... only IDTASK=1 can input data from file ...
156: C
157:      if ( NTASK.eq.1 .or. (IDTASK.eq.1.and.ITSK.eq.1) ) then
158: check %%%%%%%%%%
159: c      write(iow,*) '== srecin record 1'
160: c %%%%%%%%%%
161:        read(IUNIT,end =140,err =160,iostat =IOS) LABEL, ITYP, NDATA,
162:        &      LBUF
163:        if ( ITYP.ne.MTYP ) then
164:          write(IMG,*) 'XXX(SRECIN) data type mismatch:'
165:          write(IMG,*) '   on file: ', ITYP, ' expected: ', MTYP
166:          IERR1 = 1
167:          go to 180
168:        end if
169:        if ( NDATA.ne.NA ) then
170:          write(IMG,*) 'XXX(SRECIN) data length mismatch:'
171:          write(IMG,*) '   on file: ', NDATA, ' expected: ', NA
172:          IERR2 = 2
173:          go to 180
174:        end if
175:        if ( LABEL.ne.NAME(:MIN(LEN(LABEL),LEN(NAME))) ) then
176:          write(IMG,*) 'XXX(SRECIN) label type mismatch:'
177:          write(IMG,*) '   on file:<', LABEL, '> expected:<', NAME,
178:          &      '>'
179:          IERR4 = 4
180:          go to 180
181:        end if
182: c
183:      do 110 NP = 1, NDATA, LBUF
184: check %%%%%%%%%%
185: c      write(iow,*) '== srecin record NP 1'
186: c %%%%%%%%%%
187:        N1 = NP
188:        N2 = MIN(NDATA,N1+LBUF-1)
189:        if ( MTYP.eq.1 ) then
190:          read(IUNIT,end =150,err =170,iostat =IOS) (IA(I),I=N1,N2)
191:        else if ( MTYP.eq.2 ) then
192:          read(IUNIT,end =150,err =170,iostat =IOS) (RA(I),I=N1,N2)
193:        else if ( MTYP.eq.3 ) then
194:          read(IUNIT,end =150,err =170,iostat =IOS) (DA(I),I=N1,N2)
195:        end if

```

```

196:      110 continue
197: C
198: C
199:      else if ( IDTASK.eq.1.and.ITSK.gt.1 ) then
200: check %%%%%%%%%%
201: c      write(iow,*) '== ist ',itsk,' srecin record 1'
202: c %%%%%%%%%%
203: C
204:        read(IUNIT,end =140,err =160,iostat =IOS) LABEL, ITYP, NDATA,
205:        &      LBUF
206: C
207: C      ... send number of data & type ...
208: C
209:        call MVPCOM_SEND_CH( ITSK, 233, LABEL, LEN(LABEL), INFO )
210:        IBUF(1) = ITYP
211:        IBUF(2) = NDATA
212:        IBUF(3) = LBUF
213:        call MVPCOM_SEND_I4( ITSK, 234, IBUF, 3, INFO )
214: check %%%%%%%%%%
215: c      write(iow,*) '== ist ',itsk,' srecin record 1 sent'
216: c %%%%%%%%%%
217: C
218: C      ... wait reply from task ITSK to synchronize
219: C
220:        call MVPCOM_RECV_I4( ITSK, 9998, IIERR, 1, INFO )
221:        if ( IIERR.ne.0 ) then
222:          write(IMG,*) 'XXX(SRECIN) data reception error in task ',
223:          &      ITSK, ' code = ', IIERR
224:          IERR32 = 32
225:          go to 180
226:        end if
227: C
228:        K = 0
229:        do 120 NP = 1, NDATA, LBUF
230: check %%%%%%%%%%
231: c      write(iow,*) '== ist ',itsk,' srecin record NP ',NP
232: c %%%%%%%%%%
233:        K = K + 1
234:        N1 = NP
235:        N2 = MIN(NDATA,N1+LBUF-1)
236:        NN = N2 - N1 + 1
237:        if ( MTYP.eq.1 ) then
238:          read(IUNIT,end =150,err =170,iostat =IOS)
239:          &      (IBUF(I),I=1,NN)
240:        else if ( MTYP.eq.2 ) then
241:          read(IUNIT,end =150,err =170,iostat =IOS)
242:          &      (RBUF(I),I=1,NN)
243:        else if ( MTYP.eq.3 ) then
244:          read(IUNIT,end =150,err =170,iostat =IOS)
245:          &      (DBUF(I),I=1,NN)
246:        end if
247:        if ( MTYP.eq.1 ) then
248:
249:          call MVPCOM_SEND_I4( ITSK, 234+K, IBUF, NN, INFO )
250:
251:        else if ( MTYP.eq.2 ) then
252:
253:          call MVPCOM_SEND_R4( ITSK, 234+K, RBUF, NN, INFO )
254:
255:        else if ( MTYP.eq.3 ) then
256:
257:          call MVPCOM_SEND_R8( ITSK, 234+K, DBUF, NN, INFO )
258:
259:        end if
260:      120 continue

```

src/shared/srecin.f

```

261: check %%%%%%%%%%
262: c   write(iow,*) '== ist ',itsk,' srecin record NP ',NP, ' sent'
263: c %%%%%%%%%%%%%%
264: C
265: C ... wait reply from task ITSK to synchronize
266: C
267:       call MVPCOM_RECV_I4( ITSK, 9999, IIERR, 1, INFO )
268:       if ( IIERR.ne.0 ) then
269:         write(IMG,*) 'XXX(SRECIN) data recetion error in task ',
270:         &           ITSK, ' code = ', IIERR
271:         IERR32 = 32
272:         go to 180
273:       end if
274: C
275: C
276:       else if ( ITSK.gt.1 ) then
277: C
278: C ... receive number of data & type ...
279: C
280:       ITYP = -1
281:       NDATA = -1
282:       LBUF = -1
283:       IT0 = 1
284:       call MVPCOM_RECV_CH( IT0, 233, LABEL, LEN(LABEL), INFO )
285:       call MVPCOM_RECV_I4( IT0, 234, IBUF, 3, INFO )
286:       ITYP = IBUF(1)
287:       NDATA = IBUF(2)
288:       LBUF = IBUF(3)
289:       if ( ITYP.ne.MTYP ) then
290:         write(IMG,*) 'XXX(SRECIN) data type mismatch:'
291:         write(IMG,*) ' Record: ', ITYP, ' expected: ', MTYP
292:         IERR1 = 1
293:         IERR32 = 32
294:       end if
295:       if ( NDATA.ne.NA ) then
296:         write(IMG,*) 'XXX(SRECIN) data length mismatch:'
297:         write(IMG,*) ' Record: ', NDATA, ' expected: ', NA
298:         IERR2 = 2
299:         IERR32 = 32
300:       end if
301:       if ( LABEL.ne.NAME(:MIN(LEN(LABEL),LEN(NAME))) ) then
302:         write(IMG,*) 'XXX(SRECIN) label type mismatch:'
303:         write(IMG,*) ' Record:<', LABEL, '> expected:<', NAME, '>'
304:         IERR4 = 4
305:         IERR32 = 32
306:       end if
307: C
308: C ... send synchronization message
309: C
310:       IIERR = IERR1 + IERR2 + IERR4 + IERR32
311:       call MVPCOM_SEND_I4( IT0, 9998, IIERR, 1, INFO )
312: C
313:       if ( IIERR.ne.0 ) go to 180
314: C
315:       K = 0
316:       do 130 NP = 1, NDATA, LBUF
317:         K = K + 1
318:         N1 = NP
319:         N2 = MIN(NDATA,N1+LBUF-1)
320:         NN = N2 - N1 + 1
321:         if ( MTYP.eq.1 ) then
322: C
323:           call MVPCOM_RECV_I4( IT0, 234+K, IA(N1), NN, INFO )
324: C
325:         else if ( MTYP.eq.2 ) then

```

```

326:
327:           call MVPCOM_RECV_R4( IT0, 234+K, RA(N1), NN, INFO )
328:
329:         else if ( MTYP.eq.3 ) then
330: C
331:           call MVPCOM_RECV_R8( IT0, 234+K, DA(N1), NN, INFO )
332: C
333:         end if
334:       130 continue
335: C
336: C ... send synchronization message
337: C
338:       IIERR = 0
339:       call MVPCOM_SEND_I4( IT0, 9999, IIERR, 1, INFO )
340:       end if
341: C/#ENDIF
342: C
343:       return
344: C
345: C ..... error .....
346: C
347: 140 write(IMG,*) 'XXX(SRECIN) Unexpected end of file on reading ',
348:       & 'label record. (UNIT=', IUNIT, ') '
349:       write(IMG,*) ' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
350:       & ' name <', NAME, '> NA ', NA
351:       IERR8 = 8
352:       go to 190
353: C
354: 150 write(IMG,*) 'XXX(SRECIN) Unexpected end of file on reading ', NP,
355:       & 'th splitted data record. (UNIT=', IUNIT, ') '
356:       write(IMG,*) ' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
357:       & ' name <', NAME, '> NA ', NA
358:       IERR16 = 16
359:       go to 190
360: C
361: 160 write(IMG,*) 'XXX(SRECIN) I/O error on reading label record.',
362:       & '(UNIT=', IUNIT, ') '
363:       write(IMG,*) ' I/O Status code:', IOS
364:       write(IMG,*) ' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
365:       & ' name <', NAME, '> NA ', NA
366:       IERR8 = 8
367:       go to 190
368: C
369: 170 write(IMG,*) 'XXX(SRECIN) I/O error on reading ', NP,
370:       & 'th splitted data record. (UNIT=', IUNIT, ') '
371:       write(IMG,*) ' I/O Status code:', IOS
372:       write(IMG,*) ' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
373:       & ' name <', NAME, '> NA ', NA
374:       IERR16 = 16
375:       go to 190
376: C
377: 180 write(IMG,*) ' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
378:       & ' name <', NAME, '> NA ', NA
379:       go to 190
380: C
381: 190 IERR = IERR1 + IERR2 + IERR4 + IERR8 + IERR16 + IERR32
382:       return
383: end

```


src/shared/srecot.f

```

1:      subroutine SRECOI( IUNIT, ITSK, NAME, IA, NA, IERR )
2:      character*(*) NAME
3:      integer IA(NA)
4:      C
5:      MTYP = 1
6:      call SRECOT( MTYP, IUNIT, ITSK, NAME, IA, DUMMY, DUMMY, NA, IERR )
7:      return
8:      end
9:      C .....
10:     subroutine SRECOR( IUNIT, ITSK, NAME, RA, NA, IERR )
11:     character*(*) NAME
12:     real RA(NA)
13:     C
14:     MTYP = 2
15:     call SRECOT( MTYP, IUNIT, ITSK, NAME, DUMMY, RA, DUMMY, NA, IERR )
16:     return
17:     end
18:     C .....
19:     subroutine SRECOD( IUNIT, ITSK, NAME, DA, NA, IERR )
20:     character*(*) NAME
21:     real*8 DA(NA)
22:     C
23:     MTYP = 3
24:     call SRECOT( MTYP, IUNIT, ITSK, NAME, DUMMY, DUMMY, DA, NA, IERR )
25:     return
26:     end
27:     C .....
28:     CC
29:     subroutine SRECOT( MTYP, IUNIT, ITSK, NAME, IA, RA, DA,
30:     & NA, IERR )
31:     C=====
32:     C purpose: output records combination including data type, size etc.
33:     C and long data are splitted into more than one record.
34:     C
35:     C this routine is called from the follwing interface routines:
36:     C
37:     C SRECOI : integer array
38:     C SRECOR : real array
39:     C SRECOD : double precision real array
40:     C
41:     C called in: RESTO
42:     C-----
43:     C Arguments ( i=input, o=output, w=work )
44:     C
45:     C i mtyp : data type -- 1/2/3 = integer/real/real*8
46:     C i iunit : I/O unit to output data
47:     C i ITSK : task ID ( from 1 to NTASK )
48:     C In message passing multi task mode (PVM,MPI), ITSK may differ
49:     C from that of running task (IDTASK), and data transfer occur
50:     C in such a case;
51:     C
52:     C <PVM/MPI mode>
53:     C ITSK=IDTASK=1 : output data on file.
54:     C ITSK>1,IDTASK=1 : receive data from ITSK and output on file.
55:     C ITSK>1,IDTASK=ITSK : send data to task 1.
56:     C
57:     C In other modes, ITSK should be equal to IDTASK.
58:     C This routine should be called from a mutex (locked) block in
59:     C parallel mode other than message passing.
60:     C
61:     C i name : data name labeled on record.
62:     C i IA(NA),RA(NA),DA(NA) : output data array.
63:     C i NA : length of array.
64:     C o IERR : non zero value is returned if some error has occurred.
65:     C

```

```

66:     C Error code are sum of the following value;
67:     C IERR=16 : I/O error
68:     C=====
69:     character*(*) NAME
70:     integer IA(NA)
71:     real RA(NA)
72:     real*8 DA(NA)
73:     C
74:     include 'INC/_WORDL'
75:     include 'INC/_IUNIT'
76:     C
77:     C/#IF PARA(PVM)
78:     include 'INC/_PVMPARA'
79:     C/#ELSEIF PARA(MPI)
80:     * include 'mpif.h'
81:     C/#ELSEIF PARA(VPP)
82:     include 'INC/_VPPPARA'
83:     C/#ENDIF
84:     include 'INC/_TASKDT'
85:     C
86:     C ... data I/O buffer array (length MRBUF is defined in INC/_TASKDT)
87:     C
88:     integer IBUF(MRBUF)
89:     real RBUF(MRBUF)
90:     real*8 DBUF(MRBUF)
91:     equivalence( IBUF(1),RBUF(1),DBUF(1) )
92:     C
93:     character LABEL*16
94:     C
95:     C-----
96:     C
97:     IERR = 0
98:     C
99:     IERR1 = 0
100:    IERR2 = 0
101:    IERR4 = 0
102:    IERR8 = 0
103:    IERR16 = 0
104:    C
105:    LABEL = NAME
106:    C
107:    CCC/#IF PARA( CRAY SX* VPP ) .OR. .NOT.PARA
108:    C/#IF .NOT.PARA(PVM MPI)
109:    C
110:    write(IUNIT,err =140,iostat =IOS) LABEL, MTYP, NA, MRBUF
111:    C
112:    do 100 NP = 1, NA, MRBUF
113:        N1 = NP
114:        N2 = MIN(NA,N1+MRBUF-1)
115:        if ( MTYP.eq.1 ) then
116:            write(IUNIT,err =150,iostat =IOS) (IA(I),I=N1,N2)
117:        else if ( MTYP.eq.2 ) then
118:            write(IUNIT,err =150,iostat =IOS) (RA(I),I=N1,N2)
119:        else if ( MTYP.eq.3 ) then
120:            write(IUNIT,err =150,iostat =IOS) (DA(I),I=N1,N2)
121:        end if
122:    100 continue
123:    C
124:    C/#ELSEIF PARA( PVM MPI )
125:    C
126:    C
127:    C ... only IDTASK=1 can output data on file ...
128:    C
129:    if ( IDTASK.eq.1 ) then
130:        write(IUNIT,err =140,iostat =IOS) LABEL, MTYP, NA, MRBUF

```

src/shared/srecot.f

```

131: c      end if
132: C
133:      if ( NTASK.eq.1 .or. (IDTASK.eq.1.and.ITSK.eq.1) ) then
134:          write(IUNIT,err =140,iostat =IOS) LABEL, MTYP, NA, MRBUF
135:          do 110 NP = 1, NA, MRBUF
136:              N1      = NP
137:              N2      = MIN(NA,N1+MRBUF-1)
138:              if ( MTYP.eq.1 ) then
139:                  write(IUNIT,err =150,iostat =IOS) (IA(I),I=N1,N2)
140:              else if ( MTYP.eq.2 ) then
141:                  write(IUNIT,err =150,iostat =IOS) (RA(I),I=N1,N2)
142:              else if ( MTYP.eq.3 ) then
143:                  write(IUNIT,err =150,iostat =IOS) (DA(I),I=N1,N2)
144:              end if
145: 110      continue
146: C
147: C
148:      else if ( IDTASK.eq.1.and.ITSK.gt.1 ) then
149: C
150: C          ... receive number of data & type ...
151: C
152: C
153:          call MVPCOM_RECV_CH( ITSK, 9233, LABEL, LEN(LABEL), INFO )
154:          call MVPCOM_RECV_I4( ITSK, 9234, IBUF, 3, INFO )
155:          ITYP      = IBUF(1)
156:          NDATA     = IBUF(2)
157:          LBUF      = IBUF(3)
158: C
159:          write(IUNIT,err =140,iostat =IOS) LABEL, ITYP, NDATA, LBUF
160: C
161:          K          = 0
162:          do 120 NP = 1, NDATA, LBUF
163:              K      = K + 1
164:              N1      = NP
165:              N2      = MIN(NDATA,N1+LBUF-1)
166:              NN      = N2 - N1 + 1
167:              if ( MTYP.eq.1 ) then
168:
169:                  call MVPCOM_RECV_I4( ITSK, 9234+K, IBUF, NN, INFO )
170:                  write(IUNIT,err =150,iostat =IOS) (IBUF(I),I=1,NN)
171:
172:              else if ( MTYP.eq.2 ) then
173:
174:                  call MVPCOM_RECV_R4( ITSK, 9234+K, RBUF, NN, INFO )
175:                  write(IUNIT,err =150,iostat =IOS) (RBUF(I),I=1,NN)
176:
177:              else if ( MTYP.eq.3 ) then
178:
179:                  call MVPCOM_RECV_R8( ITSK, 9234+K, DBUF, NN, INFO )
180:                  write(IUNIT,err =150,iostat =IOS) (DBUF(I),I=1,NN)
181:
182:              end if
183: 120      continue
184: C
185: C
186:      else if ( ITSK.gt.1 ) then
187: C
188: C          ... send number of data & type ...
189: C
190:          LABEL      = NAME(:MIN(LEN(NAME),LEN(LABEL)))
191:          IT0         = 1
192:          call MVPCOM_SEND_CH( IT0, 9233, LABEL, LEN(LABEL), INFO )
193:          IBUF(1)     = MTYP
194:          IBUF(2)     = NA
195:          IBUF(3)     = MRBUF

```

```

196:          call MVPCOM_SEND_I4( IT0, 9234, IBUF, 3, INFO )
197: C
198:          K          = 0
199:          do 130 NP = 1, NA, MRBUF
200:              K      = K + 1
201:              N1      = NP
202:              N2      = MIN(NA,N1+MRBUF-1)
203:              NN      = N2 - N1 + 1
204:              if ( MTYP.eq.1 ) then
205:
206:                  call MVPCOM_SEND_I4( IT0, 9234+K, IA(N1), NN, INFO )
207:
208:              else if ( MTYP.eq.2 ) then
209:
210:                  call MVPCOM_SEND_R4( IT0, 9234+K, RA(N1), NN, INFO )
211:
212:              else if ( MTYP.eq.3 ) then
213:
214:                  call MVPCOM_SEND_R8( IT0, 9234+K, DA(N1), NN, INFO )
215:
216:              end if
217: 130      continue
218: C
219:          end if
220: C/#ENDIF
221: C
222:          return
223: C
224: C      .... error .....
225: C
226: 140 write(IMG,*)'XXX(SRECOT) I/O error on writing label record.',
227: &      '(UNIT=', IUNIT, ' )'
228:      write(IMG,*)' I/O Status code:', IOS
229:      write(IMG,*)' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
230: &      ' name <', NAME, '> NA ', NA
231:      IERR8      = 8
232:      go to 160
233: C
234: 150 write(IMG,*)'XXX(SRECOT) I/O error on writing ', NP,
235: &      'th splitted data record. (UNIT=', IUNIT, ' )'
236:      write(IMG,*)' I/O Status code:', IOS
237:      write(IMG,*)' ITSK:', ITSK, ' IDTASK:', IDTASK, ' MTYP:', MTYP,
238: &      ' name <', NAME, '> NA ', NA
239:      IERR16     = 16
240:      go to 160
241: C
242: c 950 write(IOW,*) ' ITSK:',ITSK,' IDTASK:',IDTASK,' MTYP:',MTYP,
243: c & ' name <',name,'> NA ',NA
244: c      goto 9000
245: C
246: 160 IERR      = IERR1 + IERR2 + IERR4 + IERR8 + IERR16
247:      return
248:      end

```

src/shared/stamp.f

```
1:      subroutine STAMP
2: C=====
3: C purpose: stamp target machine name or mode (single/multi-task)
4: C=====
5: C/#IF CUTIL.AND.MS_VISUAL
6: C
7: C ... This interface block is for CUTIL & MS-Visual tools. ...
8: C
9: C      interface
10: C      subroutine GTDATE( CHA1, INT1, CHA2, INT2 )
11: C      character CHA1, CHA2
12: C      integer INT1, INT2
13: *CDECS$ ATTRIBUTES C :: GTDATE
14: *CDECS$ ATTRIBUTES REFERENCE :: CHA1, INT1, CHA2, INT2
15: C      end subroutine
16: C      end interface
17: C/#ENDIF
18: C
19: C      include 'INC/_IUNIT'
20: C
21: C date string
22: C      character*60 D
23: C      character CDATE*11, CTIME*8
24: C
25: C ==== OUTPUT RUNNNIG SYSTEM & MODE =====
26: C
27: 7080 FORMAT(/8X,'RUNNING SYSTEM: ',a)
28: 7090 FORMAT(8X,'RUNNING MODE: ',a)
29: 7100 FORMAT(8X,'COMPILATION DATE: ',a)
30: 7110 format(8x,'COMPILATION DATE: ',a,2x,'TIME: ',a/)
31:
32:      JJ = 0
33:
34: C/#IF SYSTEM(FACOM)
35: C ... FACOM^{TM} : Fujitsu Automatic COMputer ...
36: *      write(ipr,7080) 'FACOM- M or VP series'
37: *      JJ = 1
38: C/#ELSEIF SYSTEM(FACOMVPP)
39: *      write(ipr,7080) 'FACOM VPP'
40: *      JJ = 1
41: C/#ELSEIF SYSTEM(FACOMAP3K)
42: *      write(ipr,7080) 'Fujitsu AP3000'
43: *      JJ = 1
44: C/#ELSEIF SYSTEM(SX*)
45: *      write(ipr,7080) 'NEC SX'
46: *      JJ = 1
47: C/#ELSEIF SYSTEM(HIOSF*)
48: *      write(ipr,7080) 'HI-OSF'
49: *      JJ = 1
50: C/#ELSEIF SYSTEM(CRAY*)
51: *      write(ipr,7080) 'CRAY'
52: *      JJ = 1
53: C/#ENDIF
54:
55: C/#IF SYSTEM(HP*)
56:
57:      write(ipr,7080) 'HP-UX'
58:      JJ = 1
59:
60: C/#ELSEIF SYSTEM(SUN)
61: *
62: *      write(ipr,7080) 'SunOS 4.x'
63: *      JJ = 1
64: *
65: C/#ELSEIF SYSTEM(SUNSV)
```

```
66: *
67: *      write(ipr,7080) 'Sun Solaris2.X'
68: *      JJ = 1
69: *
70: C/#ELSEIF SYSTEM(SUN4C)
71: *
72: *      write(ipr,7080) 'Solaris2.X Ultra Sparc'
73: *      JJ = 1
74: *
75: C/#ELSEIF SYSTEM(MIPS*)
76: *
77: *      write(ipr,7080) 'MIPS'
78: *      JJ = 1
79: *
80: C/#ELSEIF SYSTEM(AIX*)
81: *
82: *      write(ipr,7080) 'IBM AIX'
83: *      JJ = 1
84: *
85: C/#ELSEIF SYSTEM(IBMRS*)
86: *
87: *      write(ipr,7080) 'IBM RS'
88: *      JJ = 1
89: *
90: C/#ELSEIF SYSTEM(DECOSF64)
91: *
92: *      write(ipr,7080) 'DEC OSF/1 64bit'
93: *      JJ = 1
94: *
95: C/#ELSEIF SYSTEM(DECOSF*)
96: *
97: *      write(ipr,7080) 'DEC OSF/1'
98: *      JJ = 1
99: *
100: C/#ELSEIF SYSTEM(NECEWS*)
101: *
102: *      write(ipr,7080) 'NEC EWS'
103: *      JJ = 1
104: *
105: C/#ELSEIF SYSTEM(SGI*)
106: *
107: *      write(ipr,7080) 'SGI'
108: *      JJ = 1
109: *
110: C/#ENDIF
111:
112: C/#IF SYSTEM(SR2K)
113: *
114: *      write(ipr,7080) 'Hitachi SR2000 series'
115: *      JJ = 1
116: *
117: C/#ELSEIF SYSTEM(SR8K)
118: *
119: *      write(ipr,7080) 'Hitachi SR8000'
120: *      JJ = 1
121: *
122: C/#ELSEIF SYSTEM(SP2)
123: *
124: *      write(ipr,7080) 'IBM SP2'
125: *      JJ = 1
126: *
127: C/#ELSEIF SYSTEM(PARAGON)
128: *
129: *      write(ipr,7080) 'Intel PARAGON'
130: *      JJ = 1
```

src/shared/stamp.f

```
131: *
132: C/#ENDIF
133:
134: C/#IF SYSTEM(LINUXALPHA*)
135: *      write(ipr,7080) 'Linux on DEC-ALPHA'
136: *      JJ = 1
137: C/#ELSEIF SYSTEM(LINUXSPARC)
138: *      write(ipr,7080) 'Linux on Sparc'
139: *      JJ = 1
140: C/#ELSEIF SYSTEM(LINUXPPC)
141: *      write(ipr,7080) 'Linux on Power PC'
142: *      JJ = 1
143: C/#ELSEIF SYSTEM(LINUXHPPA)
144: *      write(ipr,7080) 'Linux on PA RISC'
145: *      JJ = 1
146: C/#ELSEIF SYSTEM(LINUXAOUT)
147: *      write(ipr,7080) 'Linux on X86 / a.out binary'
148: *      JJ = 1
149: C/#ELSEIF SYSTEM(LINUXELF)
150: *      write(ipr,7080) 'Linux on X86 / ELF binary'
151: *      JJ = 1
152: C/#ELSEIF SYSTEM(LINUX*)
153: *      write(ipr,7080) 'Linux'
154: *      JJ = 1
155: C/#ENDIF
156:
157: C/#IF SYSTEM(X86SOL*)
158: *
159: *      write(ipr,7080) 'Solaris on X86'
160: *      JJ = 1
161: *
162: C/#ELSEIF SYSTEM(X86BSD*)
163: *
164: *      write(ipr,7080) 'BSD on X86'
165: *      JJ = 1
166: *
167: C/#ELSEIF SYSTEM(FREEBSD*)
168: *
169: *      write(ipr,7080) 'FreeBSD'
170: *      JJ = 1
171: *
172: C/#ELSEIF SYSTEM(NTALPHA)
173: *
174: *      write(ipr,7080) 'Windows-NT on DEC-ALPHA'
175: *      JJ = 1
176: *
177: C/#ELSEIF SYSTEM(DOS)
178: *
179: *      write(ipr,7080) 'MS-DOS'
180: *      JJ = 1
181: *
182: C/#ELSEIF SYSTEM(WIN32)
183: *
184: *      write(ipr,7080) 'Windows 32bit'
185: *      JJ = 1
186: *
187: C/#ELSEIF SYSTEM(CYGWIN*)
188: *
189: *      write(ipr,7080) 'Cygnus/GNU WIN32'
190: *      JJ = 1
191: *
192: C/#ELSEIF SYSTEM(MACOS)
193: *
194: *      write(ipr,7080) 'Mac OS'
195: *      JJ = 1
```

```
196: *
197: C/#ENDIF
198:       if ( JJ.eq.0 ) write(ipr,7080) 'UNKNOWN'
199:
200: C/# IF PARA(PVM)
201: *      write(ipr,7090) 'Multi task on PVM'
202: C/# ELSEIF PARA(MPI)
203: *      write(ipr,7090) 'Multi task on MPI'
204: C/# ELSEIF PARA
205: *      write(ipr,7090) 'Multi task'
206: C/# ELSE
207: *      write(ipr,7090) 'Single task'
208: C/# ENDIF
209:
210: CCC/#EXPORT SYSTEM AS %SYSTEM%
211: C
212: Cc %SYSTEM% is replaced by subparameter strings of FAT parameter SYSTEM
213: C ... for future ...
214: C
215: C/#IF PARA( SX* CRAY* )
216: C      write(ipr,7080)
217: C      & '%SYSTEM%',
218: C      & 'MULTI TASKING AVAILABLE'
219: C/#ELSEIF PARA(PVM)
220: C      write(ipr,7080)
221: C      & '%SYSTEM%',
222: C      & 'MULTI TASKING ON PVM'
223: C/#ELSE
224: C      write(ipr,7080)
225: C      & '%SYSTEM%',
226: C      & 'SINGLE TASK'
227: C/#ENDIF
228: C
229: CC/#UNEXPORT SYSTEM
230:
231: C
232: C this string may be replaced by actual date string on compilation
233: C
234: C/#IF .NOT.NOPRDATE
235: *      call GTDATE( CDATE, len(CDATE), CTIME, len(CTIME) )
236: *      write(IPR,7110) CDATE,CTIME
237: C/#ENDIF
238: C
239:       return
240:       end
```

src/shared/stlout.f

```

1:      subroutine STLOUT( IETAL, NIETAL,NETALY,ETALY, NLETAL,JEIGN,
2:      &                  NBATCH,NSKIP,WSUM,ENGYB,KENGP,KPLIM,
3:      &                  TIMEB,ANGLB,
4:      &                  NETRV, METRV, NBETRV,IETRV, ETRV,  NMMLAG,
5:      &                  NEITER,BIN,RNM,NBINMX,NEDTMX,MXRGBN,CA, CR,
6:      c##<2007/03/14:PN3:
7:      &
8:      &                  NMT,  NMTP,  NMTPN, NUC,  NPATOM,NUCPN,
9:      &                  NUCID, NATMT, NCIDPN,KPNPRD )
10: c##>
11: C ==<MVP/GMVP>=====
12: C what to do: output direct-tally results
13: C called in: cadenz
14: C =====
15: C argument ( i=input, o=output, w=work )
16: C i IOT : printout I/O unit
17: C i IETAL(*) : edited tally information ( data packet container )
18: C i NETALY : number of edited tally
19: C i o ETALY(NLETAL,2) : edited tally ( sum & squared sum )
20: C i NLETAL : length of etaly(*,2)
21: C i NETRV : Number of special (edited) tallies whose "real variance"'s
22: C are estimated.
23: C i METRV : Total length of special (edited) tallies whose
24: C "real variance"'s are estimated. (per batch length)
25: C i IETRV(2,NETRV+1) : Pointers for special (edited) tallies whose
26: C "real variance"'s are estimated.
27: C IETRV(1,*) : edited tally #.
28: C IETRV(2,*) : tally is stored on
29: C ETRV(METRV,*) : special tally storage for "real variance"
30: C calculation.
31: C i NEITER : maximum iteration count for "real variance" estimation.
32: C i NMMLAG : maximum of step(cycle) lag taken in "real variance"
33: C estimation.
34: C -----
35: C
36: C Structure of IETAL(*)
37: C
38: C +----- edited tally loop (NETALY)
39: C | label: TALLY.# ( '#' is edited tally # increasing from 1)
40: C | integer:
41: C | (1) tally ID
42: C | (2) particle type
43: C | (3) method to synthesize direct-tallies
44: C | 0 = no combination
45: C | 1 = weighted sum
46: C | 2 = multiplication / division
47: C | 3 = general expression
48: C | 4 = Feynman-Y values
49: C | (4) number of tally dimensions
50: C | (5) starting address of this tally in etaly(*)
51: C | (6) event #
52: C | (7) weighting function
53: C | (8) nuclide/atom # & reaction #
54: C | (9) <unused>
55: C | (10) custom tally # (non-zero means user customized treatment)
56: C | string: label string
57: C | +----- tally-dimension loop
58: C | | string: "dimension-label" ('ENERGY', 'TIME' etc.)
59: C | +-----
60: C | label: DTALLY.#
61: C | integer: (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[1],
62: C | (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[2],
63: C | ...
64: C | ( data to synthesize d-tallies : not available currently )
65: C

```

```

66: C | label: DIMENSION.#
67: C | +----- tally-dimension loop
68: C | | integer: size of each dimension(number of bins) in ETALY
69: C | | REGION ---> negative, others ---> positive
70: C | integer:
71: C | .....tally-bin loop (in one packet)
72: C | . table length,
73: C | . D-tally -> E-tally translation table.
74: C | . (list of d-tally bins composing this e-tally bin)
75: C | .....
76: C | integer:
77: C | .....E-tally-bin loop (in one packet)
78: C | . table length,
79: C | . Original-tally-bin -> E-tally-bin translation table.
80: C | . (list of original-tally bins composing this e-tally bin)
81: C | .....
82: C | +----- E-tally-bin loop (when this dimension is REGION)
83: C | | string: "region name"
84: C | +-----
85: C | +-----
86: C | +-----
87: C
88: C =====
89: C Special tally output on Result file IORS (from Oct 1998)
90: C
91: C (direct output of ETALY array is obsolete)
92: C
93: C TAGS = 'TALLIES SPECIAL'
94: C write(IORS) TAGS, NETALY, NEDTMX, NBINMX, NETRV
95: C
96: C +----- repeat for tally #1 to NETAL -----
97: C
98: C TAGS = 'TALLY.'//<tally-#>
99: C write(IORS) TAG, tally-#, number-of-dimension, JRV
100: C JRV: flag to show real variance(error) estimation is available
101: C ( 0 / 1 = no/yes )
102: C
103: C TAGS = 'TALLY-LABEL' (optional)
104: C write(IORS) TAG, length-of-label-string, tally-label
105: C
106: C TAGS = 'TALLY-HEADER'
107: C write(IORS) TAG, header-array-length, header-data
108: C ( this is first "integer" data for each tally in IETAL )
109: C
110: C +----- repeat for dimension -----
111: C
112: C TAGS = 'TALLY-DIMENSION'
113: C write(IORS) TAG, dimension-type(16 char), dimension-size
114: C <Dimension type>
115: C REGION ENERGY TIME ANGLE GENERATION
116: C
117: C TAGS = 'TALLY-BIN' (if dimension type is not region)
118: C write(IORS) TAG, ( lower/upper bin combinations )
119: C
120: C TAGS = 'TALLY-REGION' (if dimension type is region)
121: C write(IORS) TAG, string-length-of-region-name, ( region name list )
122: C
123: C +-----
124: C
125: C TAGS = 'TALLY-DATA'
126: C write(IORS) TAG, size, tally-data-array(..., dim3, dim2, dim1)
127: C
128: C TAGS = 'TALLY-ERROR'
129: C write(IORS) TAG, size, tally-error-array(..., dim3, dim2, dim1)
130: C

```

src/shared/stlout.f

```

131: C | TAGS = 'TALLY-ERROR-REAL'
132: C | ("real error(variance)" estimation,if JRV.ne.0)
133: C | write(IORS) TAG, size, tally-error-array(..., dim3, dim2, dim1)
134: C +-----
135: C
136: C Old output records remain for a while...
137: C
138: C TAGS = 'SPECIAL TALLIES'
139: C write(IORS) TAGS, NETALY, NIETAL, NLETAL, NEDTMX, NBINMX
140: C write(IORS) TAGS, (IETAL(I),I=1,NIETAL)
141: C write(IORS) TAGS, (ETALY(I,1),I=1,NLETAL*2)
142: C
143: C=====
144: C implicit real*8(D)
145: C
146: C integer IETAL(*)
147: C real*8 ETALY(NLETAL,2)
148: C real*8 ETRV(METRV,NBETRV+1)
149: C integer IETRV(2,NBETRV+1)
150: C real*8 WSUM
151: C
152: C real ENGYB(*)
153: C integer KENGP(KPLIM+1)
154: C real TIMEB(*)
155: C real*8 ANGLB(*)
156: c##<2007/03/14:PN3:
157: C integer NATMT(NPATOM)
158: C character NUCID(NUC)*16, NCIDPN(NUCPN)*16
159: c##>
160: C
161: C ... working arrays
162: C
163: C real*8 BIN(NBINMX,2,8)
164: C character*128 RNM(MXRGBN)
165: C real*8 CA(NMXLAG)
166: C real*8 CR(NMXLAG)
167: C
168: C
169: C include '../shared/INC/_IOUNIT'
170: C include '../shared/INC/_TASKDT'
171: C
172: C ... local data ...
173: C
174: Ccccc integer INFO(8)
175: C integer INFO(10)
176: C parameter( MAXDIM = 8 )
177: C
178: C character*6 TAG
179: C character*32 PLAB
180: C character*128 TLABEL
181: c##<2007/03/14:PN3:
182: c## character*128 CWRK
183: c##>
184: C character*16 DIMLBL(MAXDIM)
185: c##<2007/03/14:PN3:
186: c## integer NDIME(MAXDIM), NDIMD(MAXDIM), IEDT(MAXDIM)
187: C integer NDIME(MAXDIM), IEDT(MAXDIM)
188: c##>
189: C
190: C character*32 TAGS
191: c##<2007/03/14:PN3:
192: C
193: C external IETPRDN
194: C
195: c##>

```

```

196: C-----
197: C
198: C call HEADER( IOT, 'SPECIAL TALLIES' )
199: c##<2007/03/14:PN3:
200: C
201: C if ( KPNPRD.ne.0 ) write(IOT,7100)
202: C 7100 format(' PNPRODUCE : particle production is restricted to',
203: C & ' photo-nuclear reaction.')
204: c##>
205: C
206: C >>>> output on IORS
207: C TAGS = 'TALLIES SPECIAL'
208: C write(IORS) TAGS, NETALY, NEDTMX, NBINMX, NETRV
209: C >>>>
210: C
211: C
212: C do 250 N = 1, NETALY
213: C
214: C TAG = ' '
215: C write(TAG,(''.',I5)) N
216: C call CCOMP( TAG, LEN(TAG), TAG, IIF )
217: C LTAG = ICLEN(TAG)
218: C
219: C PLAB = 'TALLY'//TAG(:LTAG)
220: C
221: C TAGS = PLAB
222: C
223: C call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
224: C if ( IRET.ne.0 ) go to 260
225: C
226: Ccccc call UNPKND( IETAL, 'INFO', 'I4', INFO, 8, NA, IRET )
227: C call UNPKND( IETAL, 'INFO', 'I4', INFO, 10, NA, IRET )
228: C if ( IRET.ne.0 ) go to 260
229: C
230: C TLABEL = ' '
231: C write(TLABEL,'(A,I3,A,I5,A)') 'Special tally No.', N, ' (ID=',
232: C & INFO(1), ' )'
233: C call LABEL( IOT, TLABEL(:ICLEN2(TLABEL)) )
234: C
235: C TLABEL = ' '
236: C call UNPKCS( IETAL, 'label', TLABEL, NL, IRET )
237: C
238: C write(IOT,7000) TLABEL(:NL)
239: C 7000 format(1X,'LABEL: <',A,'>'//)
240: C
241: C do 100 J = 1, INFO(4)
242: C DIMLBL(J) = ' '
243: C call UNPKCS( IETAL, 'label', DIMLBL(J), NL, IRET )
244: C 100 continue
245: C
246: C
247: C >>>> output on IORS
248: C JRV = 0
249: C do 110 K = 1, NETRV
250: C if ( IETRV(1,K).eq.N ) then
251: C JRV = 1
252: C go to 120
253: C end if
254: C 110 continue
255: C 120 continue
256: C
257: C TAGS = 'TALLY'.N
258: C
259: C write(IORS) TAGS, N, INFO(4), JRV
260: C TAGS = 'TALLY-LABEL'

```

src/shared/stlout.f

```

261:      write(IORS) TAGS, LEN(TLABEL), TLABEL
262:      TAGS = 'TALLY-HEADER'
263:      write(IORS) TAGS, 10, (INFO(J),J=1,10)
264: C >>>>
265: C
266: C
267: C ... combination of more than one d-tally is not supported now ...
268: Ccc if( info(3).ne.0 ) then
269: Ccc endif
270: C
271:      PLAB = 'DTALLY'//TAG(:LTAG)
272:      call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
273:      if( IRET.ne.0 ) go to 260
274: C
275:      call UNPKPT( IETAL, 'D-TALLY', 'I4', LDT, NL, IRET )
276: C
277: C ... pointer to idtal & dtaly not used now
278:      LIDT = IETAL(LDT+1)
279:      LDT = IETAL(LDT+2)
280: C
281: C ... loop for each dimension ...
282: C
283:      PLAB = 'DIMENSION'//TAG(:LTAG)
284:      call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
285:      if( IRET.ne.0 ) go to 260
286: C
287:      NSIZ = 1
288:      do 150 J = 1, INFO(4)
289: C
290: C ... size of dimension (number of edited bins)
291: C
292:      call UNPKND( IETAL, 'INFO', 'I4', NDIME(J), 1, NA, IRET )
293:      if( IRET.ne.0 ) go to 260
294:      JR = 0
295:      if( NDIME(J).lt.0 ) then
296:          NDIME(J) = ABS(NDIME(J))
297:          JR = NDIME(J)
298:      end if
299: C
300:      NSIZ = NSIZ*NDIME(J)
301: C
302: C ... edit information list : dummy not used
303:      call UNPKPT( IETAL, 'EDIT INFO', 'I4', IEDT(J), NL, IRET )
304:      if( IRET.ne.0 ) go to 260
305: C
306: C ... original bin information list
307:      call UNPKPT( IETAL, 'EDIT INFO', 'I4', IEDT(J), NL, IRET )
308:      if( IRET.ne.0 ) go to 260
309: C ... region name when this dimension is REGION
310:      if( JR.gt.0 ) then
311:          do 130 JJ = 1, JR
312:              RNM(JJ) = ' '
313:              call UNPKCS( IETAL, 'region', RNM(JJ), NL, IRET )
314:              if( IRET.ne.0 ) go to 260
315:          130 continue
316:      else
317: C Check %%%%%%%%%
318: C write(*,*) '%STLOUT timeb ',(timeb(mmm),mmm=1,4+1)
319: C %%%%%%%%%
320: C ... prepare bin boundaries
321:      IS = IEDT(J)
322:      do 140 JJ = 1, NDIME(J)
323:          NBINO = IETAL(IS)
324:          if( DIMLBL(J)(1:3).eq.'ENE' ) then
325:              KE = KENGP(INFO(2))-1

```

```

326:              BIN(JJ,1,J) = ENGYB(KE+IETAL(IS+1))
327:              BIN(JJ,2,J) = ENGYB(KE+IETAL(IS+NBINO)+1)
328: C if( INFO(2).eq.1 ) then
329: C     BIN(JJ,1,J) = ENGYB(IETAL(IS+1))
330: C     BIN(JJ,2,J) = ENGYB(IETAL(IS+NBINO)+1)
331: C else if( INFO(2).eq.2 ) then
332: C     BIN(JJ,1,J) = ENGPB(IETAL(IS+1))
333: C     BIN(JJ,2,J) = ENGPB(IETAL(IS+NBINO)+1)
334: C end if
335: C else if( DIMLBL(J)(1:3).eq.'TIM' ) then
336: C Check %%%%%%%%%
337: C write(*,*) '%STLOUT IS NBINO ',is,nbino,' IETAL ',
338: C & IETAL(IS+1), IETAL(IS+NBINO)
339: C %%%%%%%%%
340:      BIN(JJ,1,J) = TIMEB(IETAL(IS+1))
341:      BIN(JJ,2,J) = TIMEB(IETAL(IS+NBINO)+1)
342: C else if( DIMLBL(J)(1:3).eq.'ANG' ) then
343:      BIN(JJ,1,J) = ANGLB(IETAL(IS+1))
344:      BIN(JJ,2,J) = ANGLB(IETAL(IS+NBINO)+1)
345: C##<2007/03/14:PN3:
346: C## else if( DIMLBL(J)(1:3).eq.'GEN'
347: C## & .or. DIMLBL(J)(1:3).eq.'POI' ) then
348: C## else if( DIMLBL(J)(1:3).eq.'GEN' .or.
349: C## & DIMLBL(J)(1:3).eq.'POI' ) then
350: C##>
351:      BIN(JJ,1,J) = IETAL(IS+1)
352:      BIN(JJ,2,J) = IETAL(IS+NBINO)
353: C
354: C##<2007/03/14:PN3:
355: C## else if( DIMLBL(J).eq.'SOURCE'
356: C## & .or. DIMLBL(J).eq.'SOURCE-REGION'
357: C## & .or. DIMLBL(J).eq.'MARKER-REGION' ) then
358: C## else if( DIMLBL(J).eq.'SOURCE' .or.
359: C## & DIMLBL(J).eq.'SOURCE-REGION' .or.
360: C## & DIMLBL(J).eq.'MARKER-REGION' ) then
361: C##>
362:      BIN(JJ,1,J) = IETAL(IS+1)
363:      BIN(JJ,2,J) = IETAL(IS+NBINO)
364: C##<2007/03/14:PN3:
365: C## else if( DIMLBL(J).eq.'PRODUCE-REGION' ) then
366:      BIN(JJ,1,J) = IETAL(IS+1)
367:      BIN(JJ,2,J) = IETAL(IS+NBINO)
368: C## else if( DIMLBL(J).eq.'PRODUCE-NUCLIDE' ) then
369:      call IETPRDN(BIN(JJ,1,J),BIN(JJ,2,J),IETAL(IS+1),
370:      & NBINO,NUC,NPATOM,NUCPN)
371: C## else if( DIMLBL(J).eq.'PRODUCE-REACTION' ) then
372:      BIN(JJ,1,J) = IETAL(IS+1)
373:      BIN(JJ,2,J) = IETAL(IS+NBINO)
374: C##>
375:      end if
376:      IS = IS + NBINO + 1
377:      140 continue
378:      end if
379: C
380: C >>>> output on IORS
381:      TAGS = 'TALLY-DIMENSION'
382:      write(IORS) TAGS, DIMLBL(J), NDIME(J)
383:      if( DIMLBL(J)(1:3).eq.'REG' ) then
384:          TAGS = 'TALLY-REGION'
385:          write(IORS) TAGS, LEN(RNM(1)), (RNM(JJ),JJ=1,NDIME(J))
386:      else
387:          TAGS = 'TALLY-BIN'
388:          write(IORS) TAGS, (BIN(JJ,1,J),BIN(JJ,2,J),JJ=1,NDIME(J))
389:      end if
390: C >>>>

```

src/shared/stlout.f

```

391: C
392: C
393:   150      continue
394: C
395: c##<2007/03/14:PN3:
396: c##      write(IOT,7020) (DIMLBL(J):(8),'',J=1,INFO(4)), ''
397: c7020      format(1X,'  DIMENSION(  ',5(A,A,: ' ')/(1X,13X,5(A,A: ' ')))
398: c##      write(IOT,7040) (NDIME(J),J=1,INFO(4))
399: c7040      format(1X,'  SIZE      : ',5(I8,: ' ')/(1X,13X,5(I8,: ' ')))
400:      if ( INFO(4).le.5 ) then
401:        write(IOT,7020) (DIMLBL(J):(16),'',J=1,INFO(4)), ''
402:      else
403:        write(IOT,7021) (DIMLBL(J):(16),'',J=1,INFO(4)), ''
404:      end if
405:      write(IOT,7040) (NDIME(J),J=1,INFO(4))
406: 7020      format(1X,'  DIMENSION(  ',5(A,A,: ' '),A)
407: 7021      format(1X,'  DIMENSION(  ',5(A,A,: ' ')/(1X,13X,5(A,A: ' ')))
408: 7040      format(1X,'  SIZE      : ',5(I16,: ' ')/(1X,13X,5(I16,: ' ')))
409: c##>
410: C
411: C ... calculate per history value and variance ...
412: C
413:      if ( (NBATCH-NSKIP).lt.2 ) then
414:        write(IOT,'(1x,a,a)')
415: c##<2007/03/14:PN3:
416: c## &      '!!! I CAN'T CALCULATE VARIANCE BECAUSE BATCH ',
417: c## &      'NUMBER (NBATCH-NSKIP) IS LESS THAN 2.'
418: c## &      '!!!(stlout) I CAN'T CALCULATE VARIANCE BECAUSE ',
419: c## &      'BATCH NUMBER (NBATCH-NSKIP) IS LESS THAN 2.'
420: c##>
421:      DNB      = 0.0D0
422:      else
423:        if ( JEIGN.eq.0 ) then
424:          DNB      = 1.0D0/DBLE(NBATCH-1-NSKIP)
425:        else
426:          DNB      = 1.0D0/DBLE(NTASK*(NBATCH-NSKIP)-1)
427:        end if
428:      end if
429: C
430:      DN      = 1.0D0/WSUM
431:      do 160 I = 0, NSIZ - 1
432:        ETALY(INFO(5)+I,2) =
433:          &      SQRT(
434:          &      ABS(ETALY(INFO(5)+I,2)-(ETALY(INFO(5)+I,1)*2)*DN)
435:          &      DN*DNB)
436:        ETALY(INFO(5)+I,1) = ETALY(INFO(5)+I,1)*DN
437:        if ( ETALY(INFO(5)+I,1).ne.0.0 ) then
438:          ETALY(INFO(5)+I,2) = ETALY(INFO(5)+I,2) /
439:          &      ETALY(INFO(5)+I,1)*100.0D0
440:        end if
441: 160      continue
442: C
443: C ... pass dimension size of E-D tally and D -> E table and its size
444: C for each dimension
445: C
446: C ... make short form of dimension label
447: C
448:      do 170 I = 1, INFO(4)
449:        if ( DIMLBL(I).eq.'SOURCE-REGION' ) then
450:          DIMLBL(I) = 'SRC-REG'
451:        else if ( DIMLBL(I).eq.'MARKER-REGION' ) then
452:          DIMLBL(I) = 'MKR-REG'
453: c##<2007/03/14:PN3:
454:      else if ( DIMLBL(I).eq.'PRODUCE-REGION' ) then
455:        DIMLBL(I) = 'PRD-REG'

```

```

456:      else if ( DIMLBL(I).eq.'PRODUCE-NUCLIDE' ) then
457:        DIMLBL(I) = 'PRD-NUCL'
458:      else if ( DIMLBL(I).eq.'PRODUCE-REACTION' ) then
459:        DIMLBL(I) = 'PRD-REACT'
460: c##>
461:      end if
462: 170      continue
463: C
464:      if ( INFO(4).eq.0 ) then
465:        call STLPR0( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2) )
466:      else if ( INFO(4).eq.1 ) then
467:        call STLPR1( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2),
468:          &      NBINMX, BIN, RNM,
469: c##<2007/03/14:PN3:
470: c## &      NDIME(1), DIMLBL(1) )
471: c## &      NDIME(1), DIMLBL(1),
472: c## &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID, NATMT,
473: c## &      NCIDPN )
474: c##>
475:      else if ( INFO(4).eq.2 ) then
476:        call STLPR2( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2),
477:          &      NBINMX, BIN, RNM,
478:          &      NDIME(1), DIMLBL(1),
479: c##<2007/03/14:PN3:
480: c## &      NDIME(2), DIMLBL(2) )
481: c## &      NDIME(2), DIMLBL(2),
482: c## &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID, NATMT,
483: c## &      NCIDPN )
484: c##>
485:      else if ( INFO(4).eq.3 ) then
486:        call STLPR3( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2),
487:          &      NBINMX, BIN, RNM,
488:          &      NDIME(1), DIMLBL(1),
489:          &      NDIME(2), DIMLBL(2),
490: c##<2007/03/14:PN3:
491: c## &      NDIME(3), DIMLBL(3) )
492: c## &      NDIME(3), DIMLBL(3),
493: c## &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID, NATMT,
494: c## &      NCIDPN )
495: c##>
496:      else if ( INFO(4).eq.4 ) then
497:        call STLPR4( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2),
498:          &      NBINMX, BIN, RNM,
499:          &      NDIME(1), DIMLBL(1),
500:          &      NDIME(2), DIMLBL(2),
501:          &      NDIME(3), DIMLBL(3),
502: c##<2007/03/14:PN3:
503: c## &      NDIME(4), DIMLBL(4) )
504: c## &      NDIME(4), DIMLBL(4),
505: c## &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID, NATMT,
506: c## &      NCIDPN )
507: c##>
508:      else if ( INFO(4).eq.5 ) then
509:        call STLPR5( IOT, ETALY(INFO(5),1), ETALY(INFO(5),2),
510:          &      NBINMX, BIN, RNM,
511:          &      NDIME(1), DIMLBL(1),
512:          &      NDIME(2), DIMLBL(2),
513:          &      NDIME(3), DIMLBL(3),
514:          &      NDIME(4), DIMLBL(4),
515: c##<2007/03/14:PN3:
516: c## &      NDIME(5), DIMLBL(5) )
517: c## &      NDIME(5), DIMLBL(5),
518: c## &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID, NATMT,
519: c## &      NCIDPN )
520: c##>

```


src/shared/stlout.f

```

521:         else
522:         write(IOT,*)
523:         &      'XXX(STLOUT) Cannot handle tally of more than 5 dimension.'
524:         write(IOT,*) ' E-tally ', N, ' number of dim. ', INFO(3)
525:         stop 666
526:         end if
527: C
528: C >>>> output on IORS
529:         TAGS = 'TALLY-DATA'
530:         write(IORS) TAGS, NSIZ, (ETALY(INFO(5)+I,1),I=0,NSIZ-1)
531:         TAGS = 'TALLY-ERROR'
532:         write(IORS) TAGS, NSIZ, (ETALY(INFO(5)+I,2),I=0,NSIZ-1)
533: C >>>>
534: C
535: C . . print with real variance if necessary
536: C
537:         if ( NBETRV.gt.0 ) then
538:         do 180 K = 1, NBETRV
539:         if ( IETRV(1,K).eq.N ) go to 190
540:         180 continue
541:         go to 240
542: C
543:         190 DN = 1.0D0/WSUM*(NBATCH-NSKIP)
544:         do 220 J = NBETRV, 1, -1
545:         if ( J.gt.1 ) then
546:         do 200 I = IETRV(2,K) - IETRV(2,K+1) - 1
547:         ETRV(I,J) = (ETRV(I,J)-ETRV(I,J-1))*DN
548:         200 continue
549:         else
550:         do 210 I = IETRV(2,K), IETRV(2,K+1) - 1
551:         ETRV(I,J) = ETRV(I,J)*DN
552:         210 continue
553:         end if
554:         220 continue
555: C
556:         DDEPS = 1.D-9
557: CCCCCCCCCC IPRT= IOT
558:         IPRT = -1
559:         IDEBG = 1
560:
561:         do 230 I = IETRV(2,K), IETRV(2,K+1) - 1
562:         call REALVR( TLABEL(:ICLEN2(TLABEL)), ETRV(1,1), METRV,
563:         &      I, 1, NBETRV, DAVG, DSIGA, DSIGR, CA, CR, NMXLG,
564:         &      DDEPS, NEITER, IPRT, IDEBG, IERR )
565: C
566: C         ... overwrite average
567:         ETRV(I,NBETRV) = DAVG
568:         if ( DAVG.ne.0.0D0 ) then
569:         ETRV(I,NBETRV+1) = SQRT(ABS(DSIGR)) /DAVG*100.0
570:         end if
571:         230 continue
572: C
573:         write(IOT,7060) TLABEL(:ICLEN2(TLABEL))
574:         format(/1X,'* Tally With real variance : <','A','>'/)
575: C
576:         K1 = IETRV(2,K)
577:
578:         if ( INFO(4).eq.0 ) then
579:         call STLPR0( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1) )
580:         else if ( INFO(4).eq.1 ) then
581:         call STLPR1( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1),
582:         &      NBINMX, BIN, RNM,
583: C##<2007/03/14:PN3:
584: C## &      NDIME(1), DIMLBL(1) )
585:         &      NDIME(1), DIMLBL(1),

```

```

586:         &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID,
587:         &      NATMT, NCIDPN )
588: C##>
589:         else if ( INFO(4).eq.2 ) then
590:         call STLPR2( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1),
591:         &      NBINMX, BIN, RNM,
592:         &      NDIME(1), DIMLBL(1),
593: C##<2007/03/14:PN3:
594: C## &      NDIME(2), DIMLBL(2) )
595:         &      NDIME(2), DIMLBL(2),
596:         &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID,
597:         &      NATMT, NCIDPN )
598: C##>
599:         else if ( INFO(4).eq.3 ) then
600:         call STLPR3( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1),
601:         &      NBINMX, BIN, RNM,
602:         &      NDIME(1), DIMLBL(1),
603:         &      NDIME(2), DIMLBL(2),
604: C##<2007/03/14:PN3:
605: C## &      NDIME(3), DIMLBL(3) )
606:         &      NDIME(3), DIMLBL(3),
607:         &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID,
608:         &      NATMT, NCIDPN )
609: C##>
610:         else if ( INFO(4).eq.4 ) then
611:         call STLPR4( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1),
612:         &      NBINMX, BIN, RNM,
613:         &      NDIME(1), DIMLBL(1),
614:         &      NDIME(2), DIMLBL(2),
615:         &      NDIME(3), DIMLBL(3),
616: C##<2007/03/14:PN3:
617: C## &      NDIME(4), DIMLBL(4) )
618:         &      NDIME(4), DIMLBL(4),
619:         &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID,
620:         &      NATMT, NCIDPN )
621: C##>
622:         else if ( INFO(4).eq.5 ) then
623:         call STLPR5( IOT, ETRV(K1,NBETRV), ETRV(K1,NBETRV+1),
624:         &      NBINMX, BIN, RNM,
625:         &      NDIME(1), DIMLBL(1),
626:         &      NDIME(2), DIMLBL(2),
627:         &      NDIME(3), DIMLBL(3),
628:         &      NDIME(4), DIMLBL(4),
629: C##<2007/03/14:PN3:
630: C## &      NDIME(5), DIMLBL(5) )
631:         &      NDIME(5), DIMLBL(5),
632:         &      NMT, NMTP, NMTPN, NUC, NPATOM, NUCPN, NUCID,
633:         &      NATMT, NCIDPN )
634: C##>
635:         end if
636: C
637: C >>>> output on IORS
638:         TAGS = 'TALLY-ERROR-REAL'
639:         write(IORS) TAGS, NSIZ,
640:         &      (ETRV(I,NBETRV+1),I=IETRV(2,K),IETRV(2,K+1)-1)
641: C >>>>
642: C
643:         240 continue
644:         end if
645:         250 continue
646: C
647: C-----
648: C . . . . . OUTPUT TO BINARY FILE (V) EIGEN VALUES
649: C-----
650: C

```

src/shared/stlout.f

```
651: C      0----*-----1-----*-----2-----*-----3--
652: TAGS      = 'SPECIAL TALLIES'
653: C      0----*-----1-----*-----2-----*-----3--
654: write(IORS) TAGS, NETALY, NIETAL, NLETAL, NEDTMX, NBINMX
655: write(IORS) TAGS, (IETAL(I),I=1,NIETAL)
656: write(IORS) TAGS, (ETALY(I,1),I=1,NLETAL*2)
657: C
658: return
659: C
660: 260 write(IOT,*) 'XXX(STLOUT) cannot input edited tally information ',
661: & ' Code = ', IRET
662: stop 666
663: C
664: end
665: c##<2007/03/14:PN3:
666: C
667: subroutine IETPRPN(BIN1,BIN2,IETAL,NBINO,NUC,NPATOM,NUCPN)
668: C==<MVP>=====
669: C purpose: supply the bin number for produce-nuclide in special tally.
670: C called in: STLOUT
671: C=====
672: implicit real*8 (A-H,O-Z)
673: parameter (MAXNDD=200)
674: C
675: integer IETAL(*)
676: integer NDD1(MAXNDD),NDD2(MAXNDD),NDD3(MAXNDD*4)
677: C-----
678: C
679: L1      = NUC                ! neutron
680: L2      = L1 + NPATOM        ! photon
681: L3      = L2 + NUCPN         ! photonuclear
682: N1      = 0
683: N2      = 0
684: N3      = 0
685: C
686: do N = 1, NBINO
687:   if ( IETAL(N).le.L1 ) then
688:     N1      = N1 + 1
689:     NDD1(N1)= IETAL(N)
690:   else if ( IETAL(N).le.L2 ) then
691:     N2      = N2 + 1
692:     NDD2(N2)= IETAL(N) - L1
693:   else if ( IETAL(N).le.L3 ) then
694:     N3      = N3 + 1
695:     NDD3(N3)= IETAL(N) - L2
696:   end if
697: end do
698: if ( N1.gt.0 ) then
699:   BIN1     = NDD1(1)
700:   BIN2     = NDD1(N1)
701: else if ( N3.gt.0 ) then
702:   BIN1     = NDD3(1) + 1000
703:   BIN2     = NDD3(N3) + 1000
704: else if ( N2.gt.0 ) then
705:   BIN1     = NDD2(1) + 10000
706:   BIN2     = NDD2(N2) + 10000
707: end if
708: C
709: return
710: end
711: c##>
```

src/shared/stlprx.f

```

1: C
2: C STLPR0, STLPR1, STLPR2, STLRPT3, STLPR4
3: C=====
4: C Special tally printout routines
5: C
6: C array has dimension such as ETALY(dim_n,dim_n-1,...,dim3,dim2,dim1)
7: C
8: C For the last two dimension dim_n & dim_n-1 printout is done putting
9: C dimension dim_n-1 on columns and dim_n on rows;
10: C
11: C
12: C
13: C
14: C
15: C
16: C
17: C
18: C
19: C
20: C=====
21: C
22: C
23: C subroutine STLPR0( IOT, ETALY, SETALY )
24: C=====
25: C what to do: edit special tally having 0 dimension.
26: C called in : stlout
27: C=====
28: C implicit real*8(A-H,O-Z)
29: C
30: C real*8 ETALY(1)
31: C real*8 SETALY(1)
32: C
33: C=====
34: C
35: C 7000 format(1X,A,1P,E15.5,' (',0P,F6.4,'%')')
36: C
37: C write(IOT,7000) ' RESULT = ', ETALY(1), SETALY(1)
38: C return
39: C end
40: C
41: C=====
42: C
43: C subroutine STLPR1( IOT, ETALY, SETALY,NBINMX,BIN, RNM, NE1,
44: C##<2007/03/14:PN3:
45: C## & LABEL1 )
46: C## & LABEL1,
47: C## & NMT, NMTP, NMTPN, NUC, NPATOM,NUCPN,
48: C## & NUCID, NATMT, NCIDPN )
49: C##
50: C=====
51: C what to do: edit special tally having 1 dimensions.
52: C called in : stlsum
53: C history: programmed by M.Sasaki (Sep 1995)
54: C=====
55: C implicit real*8(A-H,O-Z)
56: C
57: C real*8 ETALY(NE1)
58: C real*8 SETALY(NE1)
59: C
60: C character*(*) LABEL1
61: C real*8 BIN(NBINMX,2,8)
62: C character RNM(NBINMX)*128
63: C##<2007/03/14:PN3:
64: C integer NATMT(NPATOM)
65: C character HSLAB1*5, HSLAB01*10, HSLAB02*10

```

```

66: C character NUCID(NUC)*16, NCIDPN(NUCPN)*16
67: C character HNCIDZA*7, HW1*7, HW2*7, CHSYMB*2
68: C external HNCIDZA, CHSYMB
69: C##
70: C=====
71: C
72: C 7000 format(1X,I8,1P,E15.5,' (',0P,F6.4,'%')')
73: C 7020 format(1X,I4,1X,A12,1P,E15.5,' (',0P,F7.3,'%')')
74: C 7040 format(1X,I4,1X,1P,2E11.4,E15.5,' (',0P,F7.3,'%')')
75: C##<2007/03/14:PN3:
76: C## 7060 format(1X,I4,1X,2I11,E15.5,' (',0P,F7.3,'%')')
77: C## 7080 format(1X,I4,1X,I11,11X,E15.5,' (',0P,F7.3,'%')')
78: C 7060 format(1X,I4,1X,2I11,1P,E15.5,' (',0P,F7.3,'%')')
79: C 7080 format(1X,I4,1X,I11,11X,1P,E15.5,' (',0P,F7.3,'%')')
80: C 7440 format(1X,I4,3X,A5,2I5,1P,E13.5,' (',F7.3,'%')')
81: C 7600 format(1X,I4,3X,2A10,1P,E13.5,' (',F7.3,'%')')
82: C##
83: C
84: C##<2007/03/14:PN3:
85: C NMT1 = NMT + NMTP
86: C NMT2 = NMT1 + NMTPN
87: C
88: C##
89: C write(IOT,'(/1X,a8)') LABEL1
90: C if ( LABEL1(1:3).eq.'REG' ) then
91: C write(IOT,7020) (I,RNM(I)(1:12),ETALY(I),SETALY(I),I=1,NE1)
92: C else if ( LABEL1(1:3).eq.'GEN'.or.
93: C & LABEL1.eq.'SOURCE'.or.
94: C & LABEL1.eq.'MKR-REG'.or.
95: C##<2007/03/14:PN3:
96: C## & LABEL1.eq.'SRC-REG' ) then
97: C & LABEL1.eq.'SRC-REG'.or.
98: C & LABEL1.eq.'PRD-REG' ) then
99: C##
100: C write(IOT,7060)
101: C & (I,(NINT(BIN(I,J,1)),J=1,2),ETALY(I),SETALY(I),I=1,NE1)
102: C else if ( LABEL1(1:3).eq.'POI' ) then
103: C write(IOT,7080)
104: C & (I,NINT(BIN(I,1,1)),ETALY(I),SETALY(I),I=1,NE1)
105: C##<2007/03/14:PN3:
106: C else if ( LABEL1.eq.'PRD-NUCL' ) then
107: C NCBB1 = nint(BIN(I,1,1))
108: C NCBB2 = nint(BIN(I,2,1))
109: C if ( NCBB1.lt.1000 ) then
110: C HW1 = HNCIDZA(NUCID(NCBB1))
111: C HW2 = HNCIDZA(NUCID(NCBB2))
112: C HSLAB01 = 'N: '//HW1
113: C HSLAB02 = 'N: '//HW2
114: C else if ( NCBB1.lt.10000 ) then
115: C HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
116: C HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
117: C HSLAB01 = 'PN: '//HW1(1:6)
118: C HSLAB02 = 'PN: '//HW2(1:6)
119: C else
120: C HW1 = CHSYMB(NATMT(NCBB1-10000))// '
121: C HW2 = CHSYMB(NATMT(NCBB2-10000))// '
122: C HSLAB01 = 'P: '//HW1
123: C HSLAB02 = 'P: '//HW2
124: C end if
125: C write(IOT,7600) (I, HSLAB01, HSLAB02, ETALY(I), SETALY(I),
126: C & I=1,NE1)
127: C else if ( LABEL1.eq.'PRD-REACT' ) then
128: C MTBB1 = nint(BIN(I,1,1))
129: C MTBB2 = nint(BIN(I,2,1))
130: C HSLAB1 = ' MT'

```

src/shared/stlprx.f

```

131:         if ( MTBB1.le.NMT ) then
132:             HSLAB1 = ' N-MT'
133:         else if ( MTBB1.le.NMT1 ) then
134:             HSLAB1 = ' P-MT'
135:             MTBB1 = MTBB1 - NMT
136:             MTBB2 = MTBB2 - NMT
137:         else if ( MTBB1.le.NMT2 ) then
138:             HSLAB1 = 'PN-MT'
139:             MTBB1 = MTBB1 - NMT1
140:             MTBB2 = MTBB2 - NMT1
141:         end if
142:         write(IOT,7440) (I, HSLAB1, MTBB1, MTBB2, ETALY(I), SETALY(I),
143: & I=1,NE1)
144: c##>
145:         else
146:             write(IOT,7040)
147:             & (I,(BIN(I,J,1),J=1,2),ETALY(I),SETALY(I),I=1,NE1)
148:         end if
149:         return
150:     end
151: C
152: C-----
153: C
154:     subroutine STLPR2( IOT, ETALY, SETALY,NBINMX,BIN, RNM, NE1,
155: c##<2007/03/14:PN3:
156: c## & LABEL1,NE2, LABEL2 )
157: & LABEL1,NE2, LABEL2,
158: & NMT, NMTP, NMTPN, NUC, NPATOM,NUCPN,
159: & NUCID, NATMT, NCIDPN )
160: c##>
161: C=====
162: C what to do: edit special tally having 2 dimensions.
163: C called in : stlsum
164: C history: programmed by M.Sasaki (Sep 1995)
165: C update:
166: C 23 Mar 1998: bin output for 1st dimension must be BIN(*,*,1)
167: C but have been output BIN(*,*,2).
168: C Place a blank line between bin-list and results.
169: C=====
170:     implicit real*8(A-H,O-Z)
171:     parameter( NL = 8 )
172: C
173:     real*8 ETALY(NE2,NE1)
174:     real*8 SETALY(NE2,NE1)
175: C
176:     character*(*) LABEL1
177:     character*(*) LABEL2
178:     real*8 BIN(NBINMX,2,8)
179:     character RNM(NBINMX)*128
180: c##<2007/03/14:PN3:
181:     real*8 TOTVAL(NL), TOTERR(NL)
182:     integer NATMT(NPATOM)
183:     character HSLAB1*5, HSLAB0(8,2)*10
184:     character NUCID(NUC)*16, NCIDPN(NUCPN)*16
185:     character HNCIDZA*7, HW1*7, HW2*7, CHSYMB*2
186:     external HNCIDZA, CHSYMB
187: c##>
188: C
189: C-----
190: C
191: 7000 format(/2X,A10,7X,8(1X,A8,I3,1X)/)
192: 7020 format(2X,17X,8(1X,A12)/)
193: 7040 format(2X,17X,1P,8(E12.4,1X))
194: 7060 format(2X,17X,8(I12,1X))
195: 7080 format(

```

```

196: 7100 format(1X,I4,1X,1P,E12.4,8E13.5)
197: c##<2007/03/14:PN3:
198: c## 7120 format(1X,I4,1X,I12,8E13.5)
199: 7120 format(1X,I4,1X,I12,1P,8E13.5)
200: c##>
201: 7140 format(1X,I4,1X,A12,1P,8E13.5)
202: 7160 format(1X,I4,13X,1P,8E13.5)
203: 7180 format(1X,4X,1X,1P,E12.4,0P,8(' (',F7.3,'%') ':))
204: 7200 format(1X,4X,1X,I12,0P,8(' (',F7.3,'%') ':))
205: 7220 format(1X,17X,8(' (',F7.3,'%') ':))
206: c##<2007/03/14:PN3:
207: 7440 format(1X,I4,3X,A5,I5,1P,8E13.5)
208: 7460 format(1X,7X,A5,I5,8(' (',F7.3,'%') ':))
209: 7480 format(2X,17X,8(2X,A10,1X))
210: 7600 format(1X,I4,3X,A10,1P,8E13.5)
211: 7620 format(1X,7X,A10,8(' (',F7.3,'%') ':))
212: 7640 format(1X,' TOTAL',9X,1P,8E13.5)
213: c##>
214: c
215: c##<2007/03/14:PN3:
216: NMT1 = NMT + NMTP
217: NMT2 = NMT1 + NMTPN
218: C
219: c##>
220:     do 110 K = 1, NE1, NL
221:         K1 = MIN(NE1,K+NL-1)
222:         write(IOT,7000) LABEL2, (LABEL1,L,L=K,K1)
223:         if ( LABEL1(1:3).eq.'REG' ) then
224:             write(IOT,7020) (RNM(L)(1:12),L=K,K1)
225:         else if ( LABEL1(1:3).eq.'GEN'.or.
226: & LABEL1.eq.'SOURCE'.or.
227: & LABEL1.eq.'MKR-REG'.or.
228: c##<2007/03/14:PN3:
229: c## & LABEL1.eq.'SRC-REG' ) then
230: & LABEL1.eq.'SRC-REG'.or.
231: & LABEL1.eq.'PRD-REG' ) then
232: c##>
233:             write(IOT,7060) (NINT(BIN(L,1,1)),L=K,K1)
234:             write(IOT,7060) (NINT(BIN(L,2,1)),L=K,K1)
235:             write(IOT,7080)
236:         else if ( LABEL1(1:3).eq.'POI' ) then
237:             write(IOT,7060) (NINT(BIN(L,1,1)),L=K,K1)
238:             write(IOT,7080)
239: c##<2007/03/14:PN3:
240:         else if ( LABEL1.eq.'PRD-NUCL' ) then
241:             LL = 0
242:             do L = K, K1
243:                 NCBB1 = nint(BIN(L,1,1))
244:                 NCBB2 = nint(BIN(L,2,1))
245:                 LL = LL + 1
246:                 if ( NCBB1.lt.1000 ) then
247:                     HW1 = HNCIDZA(NUCID(NCBB1))
248:                     HW2 = HNCIDZA(NUCID(NCBB2))
249:                     HSLAB0(LL,1)='N: '//HW1
250:                     HSLAB0(LL,2)='N: '//HW2
251:                 else if ( NCBB1.lt.10000 ) then
252:                     HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
253:                     HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
254:                     HSLAB0(LL,1)='PN: '//HW1(1:6)
255:                     HSLAB0(LL,2)='PN: '//HW2(1:6)
256:                 else
257:                     HW1 = CHSYMB(NATMT(NCBB1-10000))//
258:                     HW2 = CHSYMB(NATMT(NCBB2-10000))//
259:                     HSLAB0(LL,1)='P: '//HW1
260:                     HSLAB0(LL,2)='P: '//HW2

```

src/shared/stlprx.f

```

261:         end if
262:     end do
263:     write(IOT,7480) (HSLAB0(L,1),L=1,LL)
264:     write(IOT,7480) (HSLAB0(L,2),L=1,LL)
265:     write(IOT,7080)
266:     else if ( LABEL1.eq.'PRD-REACT' ) then
267:         LL = 0
268:         do L = K, K1
269:             MTBB1 = nint(BIN(L,1,1))
270:             MTBB2 = nint(BIN(L,2,1))
271:             HSLAB1 = ' MT'
272:             if ( MTBB1.le.NMT ) then
273:                 HSLAB1 = ' N-MT'
274:             else if ( MTBB1.le.NMT1 ) then
275:                 HSLAB1 = ' P-MT'
276:                 MTBB1 = MTBB1 - NMT
277:                 MTBB2 = MTBB2 - NMT
278:             else if ( MTBB1.le.NMT2 ) then
279:                 HSLAB1 = 'PN-MT'
280:                 MTBB1 = MTBB1 - NMT1
281:                 MTBB2 = MTBB2 - NMT1
282:             end if
283:             LL = LL + 1
284:             write(HSLAB0(LL,1),'(A5,I5)') HSLAB1, MTBB1
285:             write(HSLAB0(LL,2),'(A5,I5)') HSLAB1, MTBB2
286:         end do
287:         write(IOT,7480) (HSLAB0(L,1),L=1,LL)
288:         write(IOT,7480) (HSLAB0(L,2),L=1,LL)
289:         write(IOT,7080)
290: c##>
291:     else
292:         write(IOT,7040) (BIN(L,1,2),L=K,K1)
293:         write(IOT,7040) (BIN(L,2,2),L=K,K1)
294:         write(IOT,7040) (BIN(L,1,1),L=K,K1)
295:         write(IOT,7040) (BIN(L,2,1),L=K,K1)
296:         write(IOT,7080)
297:     end if
298: *VOCL LOOP,SCALAR
299:     do 100 I = 1, NE2
300:         if ( LABEL2(1:3).eq.'REG' ) then
301:             write(IOT,7140) I, RNM(I) (1:12), (ETALY(I,L),L=K,K1)
302:             write(IOT,7220) (SETALY(I,L),L=K,K1)
303:         else if ( LABEL2(1:3).eq.'GEN'.or.
304:             & LABEL2.eq.'SOURCE'.or.
305:             & LABEL2.eq.'MKR-REG'.or.
306: c##<2007/03/14:PN3:
307: c## & LABEL2.eq.'SRC-REG' ) then
308:             & LABEL2.eq.'SRC-REG'.or.
309:             & LABEL2.eq.'PRD-REG' ) then
310: c##>
311:             write(IOT,7120) I, NINT(BIN(I,1,2)), (ETALY(I,L),L=K,K1)
312:             write(IOT,7200) NINT(BIN(I,2,2)), (SETALY(I,L),L=K,K1)
313:         else if ( LABEL2(1:3).eq.'POI' ) then
314:             write(IOT,7120) I, NINT(BIN(I,1,2)), (ETALY(I,L),L=K,K1)
315:             write(IOT,7220) (SETALY(I,L),L=K,K1)
316: c##<2007/03/14:PN3:
317:         else if ( LABEL2.eq.'PRD-NUCL' ) then
318:             NCBB1 = nint(BIN(I,1,2))
319:             NCBB2 = nint(BIN(I,2,2))
320:             LL = 1
321:             if ( NCBB1.lt.1000 ) then
322:                 HW1 = HNCIDZA(NUCID(NCBB1))
323:                 HW2 = HNCIDZA(NUCID(NCBB2))
324:                 HSLAB0(LL,1)='N: '//HW1
325:                 HSLAB0(LL,2)='N: '//HW2

```

```

326:         else if ( NCBB1.lt.10000 ) then
327:             HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
328:             HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
329:             HSLAB0(LL,1)='PN: '//HW1(1:6)
330:             HSLAB0(LL,2)='PN: '//HW2(1:6)
331:         else
332:             HW1 = CHSYMB(NATMT(NCBB1-10000))/' '
333:             HW2 = CHSYMB(NATMT(NCBB2-10000))/' '
334:             HSLAB0(LL,1)='P: '//HW1
335:             HSLAB0(LL,2)='P: '//HW2
336:         end if
337:         write(IOT,7600) I, HSLAB0(1,1), (ETALY(I,L),L=K,K1)
338:         write(IOT,7620) HSLAB0(1,2), (SETALY(I,L),L=K,K1)
339:     else if ( LABEL2.eq.'PRD-REACT' ) then
340:         MTBB1 = nint(BIN(I,1,2))
341:         MTBB2 = nint(BIN(I,2,2))
342:         HSLAB1 = ' MT'
343:         if ( MTBB1.le.NMT ) then
344:             HSLAB1 = ' N-MT'
345:         else if ( MTBB1.le.NMT1 ) then
346:             HSLAB1 = ' P-MT'
347:             MTBB1 = MTBB1 - NMT
348:             MTBB2 = MTBB2 - NMT
349:         else if ( MTBB1.le.NMT2 ) then
350:             HSLAB1 = 'PN-MT'
351:             MTBB1 = MTBB1 - NMT1
352:             MTBB2 = MTBB2 - NMT1
353:         end if
354:         write(IOT,7440) I, HSLAB1, MTBB1, (ETALY(I,L),L=K,K1)
355:         write(IOT,7460) HSLAB1, MTBB2, (SETALY(I,L),L=K,K1)
356: c##>
357:     else
358:         write(IOT,7100) I, BIN(I,1,2), (ETALY(I,L),L=K,K1)
359:         write(IOT,7180) BIN(I,2,2), (SETALY(I,L),L=K,K1)
360:     end if
361:     100 continue
362: c##<2007/03/14:PN3:
363:     do LL = 1, NL
364:         TOTVAL(LL) = 0
365:         TOTERR(LL) = 0
366:     end do
367:     LL = 0
368:     do L = K, K1
369:         LL = LL + 1
370:         do I = 1, NE2
371:             TOTVAL(LL) = TOTVAL(LL) + ETALY(I,L)
372:             TOTERR(LL) = TOTERR(LL) + (ETALY(I,L)*SETALY(I,L))**2
373:         end do
374:         if ( TOTVAL(LL).gt.0.D0 ) then
375:             TOTERR(LL) = sqrt(TOTERR(LL)) / TOTVAL(LL)
376:         else
377:             TOTERR(LL) = 0
378:         end if
379:     end do
380:     write(IOT,7640) (TOTVAL(I),I=1,LL)
381:     write(IOT,7220) (TOTERR(I),I=1,LL)
382: c##>
383:     110 continue
384: c
385:     return
386: end
387: C
388: C-----
389: C
390:     subroutine STLPR3( IOT, ETALY, SETALY,NBINMX,BIN, RNM, NE1,

```

src/shared/stlprx.f

```

391: c##<2007/03/14:PN3:
392: c## & LABEL1,NE2, LABEL2,NE3, LABEL3 )
393: & LABEL1,NE2, LABEL2,NE3, LABEL3,
394: & NMT, NMTP, NMTPN, NUC, NPATOM,NUCPN,
395: & NUCID, NATMT, NCIDPN )
396: c##>
397: C=====
398: C what to do: edit special tally having 3 dimensions.
399: C called in : stlsum
400: C history: programmed by M.Sasaki (Sep 1995)
401: C=====
402: implicit real*8(A-H,O-Z)
403: parameter( NL = 8 )
404: C
405: real*8 ETALY(NE3,NE2,NE1)
406: real*8 SETALY(NE3,NE2,NE1)
407: C
408: character(*) LABEL1
409: character(*) LABEL2
410: character(*) LABEL3
411: real*8 BIN(NBINMX,2,8)
412: character RNM(NBINMX)*128
413: c##<2007/03/14:PN3:
414: real*8 TOTVAL(NL), TOTERR(NL)
415: integer NATMT(NPATOM)
416: character HSLAB1*5, HSLAB0(8,2)*10
417: character NUCID(NUC)*16, NCIDEN(NUCPN)*16
418: character HNCIDZA*7, HW1*7, HW2*7, CHSYMB*2
419: external HNCIDZA, CHSYMB
420: c##>
421: c
422: C-----
423: C
424: 7000 format(/1X,4(:' == ',A8,I4))
425: 7020 format(/1X,' == ',A8,I4,3X,A)
426: 7040 format(/1X,' == ',A8,I4,3X,1P,' FROM ',E11.4,' TO ',E11.4)
427: 7060 format(/1X,' == ',A8,I4,3X,' FROM ',I11,' TO ',I11)
428: c##<2007/03/14:PN3:
429: 7500 format(/1X,1(' == ',A8,I4,3X),2X,A5,' FROM ',I1,' TO ',I5)
430: 7561 format(/1X,1(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
431: c##>
432: c
433: 7080 format(/2X,A10,7X,8(1X,A8,I3,1X)/)
434: 7100 format(2X,17X,8(1X,A12)/)
435: 7120 format(2X,17X,1P,8(E12.4,1X))
436: 7140 format(2X,17X,8(I12,1X))
437: 7160 format()
438: 7180 format(1X,I4,1X,1P,E12.4,8E13.5)
439: c##<2007/03/14:PN3:
440: c## 7200 format(1X,I4,1X,I12,8E13.5)
441: 7200 format(1X,I4,1X,I12,1P,8E13.5)
442: c##>
443: 7220 format(1X,I4,1X,A12,1P,8E13.5)
444: 7240 format(1X,I4,13X,1P,8E13.5)
445: 7260 format(1X,4X,1X,1P,E12.4,0P,8(' (',F7.3,'%') ':))
446: 7280 format(1X,4X,1X,I12,0P,8(' (',F7.3,'%') ':))
447: 7300 format(1X,17X,8(' (',F7.3,'%') ':))
448: c##<2007/03/14:PN3:
449: 7440 format(1X,I4,3X,A5,I5,1P,8E13.5)
450: 7460 format(1X,7X,A5,I5,8(' (',F7.3,'%') ':))
451: 7480 format(2X,17X,8(2X,A10,1X))
452: 7600 format(1X,I4,3X,A10,1P,8E13.5)
453: 7620 format(1X,7X,A10,8(' (',F7.3,'%') ':))
454: 7640 format(1X,' TOTAL',9X,1P,8E13.5)
455: c##>

```

```

456: c
457: c##<2007/03/14:PN3:
458: NMT1 = NMT + NMTP
459: NMT2 = NMT1 + NMTPN
460: C
461: c##>
462: do 120 I1 = 1, NE1
463: if ( LABEL1(1:3).eq.'REG' ) then
464: write(IOT,7020) LABEL1, I1, RNM(I1) (1:ICLEN2(RNM(I1)))
465: else if ( LABEL1(1:3).eq.'GEN' .or.
466: & LABEL1.eq.'SOURCE' .or.
467: & LABEL1.eq.'MKR-REG' .or.
468: c##<2007/03/14:PN3:
469: c## & LABEL1.eq.'SRC-REG' ) then
470: & LABEL1.eq.'SRC-REG' .or.
471: & LABEL1.eq.'PRD-REG' ) then
472: c##>
473: write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
474: else if ( LABEL1(1:3).eq.'POI' ) then
475: write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
476: c##<2007/03/14:PN3:
477: else if ( LABEL1.eq.'PRD-NUCL' ) then
478: NCBB1 = nint(BIN(I1,1,1))
479: NCBB2 = nint(BIN(I1,2,1))
480: LL = 1
481: if ( NCBB1.lt.1000 ) then
482: HW1 = HNCIDZA(NUCID(NCBB1))
483: HW2 = HNCIDZA(NUCID(NCBB2))
484: HSLAB0(LL,1)='N: '//HW1
485: HSLAB0(LL,2)='N: '//HW2
486: else if ( NCBB1.lt.10000 ) then
487: HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
488: HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
489: HSLAB0(LL,1)='PN: '//HW1(1:6)
490: HSLAB0(LL,2)='PN: '//HW2(1:6)
491: else
492: HW1 = CHSYMB(NATMT(NCBB1-10000))/' '
493: HW2 = CHSYMB(NATMT(NCBB2-10000))/' '
494: HSLAB0(LL,1)='P: '//HW1
495: HSLAB0(LL,2)='P: '//HW2
496: end if
497: write(IOT,7561) LABEL1, I1, HSLAB0(1,1), HSLAB0(1,2)
498: else if ( LABEL1.eq.'PRD-REACT' ) then
499: MTBB1 = nint(BIN(I1,1,1))
500: MTBB2 = nint(BIN(I1,2,1))
501: HSLAB1 = ' MT'
502: if ( MTBB1.le.NMT ) then
503: HSLAB1 = ' N-MT'
504: else if ( MTBB1.le.NMT1 ) then
505: HSLAB1 = ' P-MT'
506: MTBB1 = MTBB1 - NMT
507: MTBB2 = MTBB2 - NMT
508: else if ( MTBB1.le.NMT2 ) then
509: HSLAB1 = ' PN-MT'
510: MTBB1 = MTBB1 - NMT1
511: MTBB2 = MTBB2 - NMT1
512: end if
513: write(IOT,7500) LABEL1, I1, HSLAB1, MTBB1, MTBB2
514: c##>
515: else
516: write(IOT,7040) LABEL1, I1, (BIN(I1,JJ,1),JJ=1,2)
517: end if
518: c .....
519: do 110 K = 1, NE2, NL
520: K1 = MIN(NE2,K+NL-1)

```

src/shared/stlprx.f

```

521:      write(IOT,7080) LABEL3, (LABEL2,L,L=K,K1)
522:      if ( LABEL2(1:3).eq.'REG' ) then
523:        write(IOT,7100) (RNM(L)(1:12),L=K,K1)
524:      else if ( LABEL2(1:3).eq.'GEN' .or.
525:        & LABEL2.eq.'SOURCE' .or.
526:        & LABEL2.eq.'MKR-REG' .or.
527: c##<2007/03/14:PN3:
528: c## & LABEL2.eq.'SRC-REG' ) then
529:      & LABEL2.eq.'SRC-REG' .or.
530:      & LABEL2.eq.'PRD-REG' ) then
531: c##>
532:      write(IOT,7140) (NINT(BIN(L,1,2)),L=K,K1)
533:      write(IOT,7140) (NINT(BIN(L,2,2)),L=K,K1)
534:      write(IOT,7160)
535:      else if ( LABEL2(1:3).eq.'POI' ) then
536:      write(IOT,7140) (NINT(BIN(L,1,2)),L=K,K1)
537:      write(IOT,7160)
538: c##<2007/03/14:PN3:
539:      else if ( LABEL2.eq.'PRD-NUCL' ) then
540:      LL = 0
541:      do L = K, K1
542:        NCBB1 = nint(BIN(L,1,2))
543:        NCBB2 = nint(BIN(L,2,2))
544:        LL = LL + 1
545:        if ( NCBB1.lt.1000 ) then
546:          HW1 = HNCIDZA(NUCID(NCBB1))
547:          HW2 = HNCIDZA(NUCID(NCBB2))
548:          HSLAB0(LL,1)='N: '//HW1
549:          HSLAB0(LL,2)='N: '//HW2
550:        else if ( NCBB1.lt.10000 ) then
551:          HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
552:          HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
553:          HSLAB0(LL,1)='PN: '//HW1(1:6)
554:          HSLAB0(LL,2)='PN: '//HW2(1:6)
555:        else
556:          HW1 = CHSYMB(NATMT(NCBB1-10000))// ' '
557:          HW2 = CHSYMB(NATMT(NCBB2-10000))// ' '
558:          HSLAB0(LL,1)='P: '//HW1
559:          HSLAB0(LL,2)='P: '//HW2
560:        end if
561:      end do
562:      write(IOT,7480) (HSLAB0(L,1),L=1,LL)
563:      write(IOT,7480) (HSLAB0(L,2),L=1,LL)
564:      write(IOT,7280)
565:      else if ( LABEL2.eq.'PRD-REACT' ) then
566:      LL = 0
567:      do L = K, K1
568:        MTBB1 = nint(BIN(L,1,2))
569:        MTBB2 = nint(BIN(L,2,2))
570:        HSLAB1 = ' MT'
571:        if ( MTBB1.le.NMT ) then
572:          HSLAB1 = ' N-MT'
573:        else if ( MTBB1.le.NMT1 ) then
574:          HSLAB1 = ' P-MT'
575:        MTBB1 = MTBB1 - NMT
576:        MTBB2 = MTBB2 - NMT
577:        else if ( MTBB1.le.NMT2 ) then
578:          HSLAB1 = ' PN-MT'
579:        MTBB1 = MTBB1 - NMT1
580:        MTBB2 = MTBB2 - NMT1
581:      end if
582:      LL = LL + 1
583:      write(HSLAB0(LL,1),'(A5,I5)') HSLAB1, MTBB1
584:      write(HSLAB0(LL,2),'(A5,I5)') HSLAB1, MTBB2
585:    end do

```

```

586:      write(IOT,7480) (HSLAB0(L,1),L=1,LL)
587:      write(IOT,7480) (HSLAB0(L,2),L=1,LL)
588:      write(IOT,7160)
589: c##>
590:      else
591:      write(IOT,7120) (BIN(L,1,2),L=K,K1)
592:      write(IOT,7120) (BIN(L,2,2),L=K,K1)
593:      write(IOT,7160)
594:      end if
595: *VOCL LOOP,SCALAR
596:      do 100 I = 1, NE3
597:        if ( LABEL3(1:3).eq.'REG' ) then
598:          write(IOT,7220) I, RNM(I) (1:12),
599:            & (ETALY(I,L,I1),L=K,K1)
600:          write(IOT,7300) (SETALY(I,L,I1),L=K,K1)
601:        else if ( LABEL3(1:3).eq.'GEN' .or.
602:          & LABEL3.eq.'SOURCE' .or.
603:          & LABEL3.eq.'MKR-REG' .or.
604: c##<2007/03/14:PN3:
605: c## & LABEL3.eq.'SRC-REG' ) then
606:      & LABEL3.eq.'SRC-REG' .or.
607:      & LABEL3.eq.'PRD-REG' ) then
608: c##>
609:          write(IOT,7200) I, NINT(BIN(I,1,3)),
610:            & (ETALY(I,L,I1),L=K,K1)
611:          write(IOT,7280) NINT(BIN(I,2,3)),
612:            & (SETALY(I,L,I1),L=K,K1)
613:        else if ( LABEL3(1:3).eq.'POI' ) then
614:          write(IOT,7200) I, NINT(BIN(I,1,3)),
615:            & (ETALY(I,L,I1),L=K,K1)
616:          write(IOT,7300)
617:            & (SETALY(I,L,I1),L=K,K1)
618: c##<2007/03/14:PN3:
619:      else if ( LABEL3.eq.'PRD-NUCL' ) then
620:        NCBB1 = nint(BIN(I,1,3))
621:        NCBB2 = nint(BIN(I,2,3))
622:        LL = 1
623:        if ( NCBB1.lt.1000 ) then
624:          HW1 = HNCIDZA(NUCID(NCBB1))
625:          HW2 = HNCIDZA(NUCID(NCBB2))
626:          HSLAB0(LL,1)='N: '//HW1
627:          HSLAB0(LL,2)='N: '//HW2
628:        else if ( NCBB1.lt.10000 ) then
629:          HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
630:          HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
631:          HSLAB0(LL,1)='PN: '//HW1(1:6)
632:          HSLAB0(LL,2)='PN: '//HW2(1:6)
633:        else
634:          HW1 = CHSYMB(NATMT(NCBB1-10000))// ' '
635:          HW2 = CHSYMB(NATMT(NCBB2-10000))// ' '
636:          HSLAB0(LL,1)='P: '//HW1
637:          HSLAB0(LL,2)='P: '//HW2
638:        end if
639:        write(IOT,7600) I, HSLAB0(1,1),
640:          & (ETALY(I,L,I1),L=K,K1)
641:        write(IOT,7620) HSLAB0(1,2), (SETALY(I,L,I1),L=K,K1)
642:      else if ( LABEL3.eq.'PRD-REACT' ) then
643:        MTBB1 = nint(BIN(I,1,3))
644:        MTBB2 = nint(BIN(I,2,3))
645:        HSLAB1 = ' MT'
646:        if ( MTBB1.le.NMT ) then
647:          HSLAB1 = ' N-MT'
648:        else if ( MTBB1.le.NMT1 ) then
649:          HSLAB1 = ' P-MT'
650:        MTBB1 = MTBB1 - NMT

```

src/shared/stlprx.f

```

651:      MTBB2 = MTBB2 - NMT
652:      else if ( MTBB1.le.NMT2 ) then
653:        HSLAB1 = 'PN-MT'
654:        MTBB1 = MTBB1 - NMT1
655:        MTBB2 = MTBB2 - NMT1
656:      end if
657:      write(IOT,7440) I, HSLAB1, MTBB1,
658:      &      (ETALY(I,L,I1),L=K,K1)
659:      write(IOT,7460) HSLAB1, MTBB2, (SETALY(I,L,I1),L=K,K1)
660: c##>
661:      else
662:        write(IOT,7180) I, BIN(I,1,3), (ETALY(I,L,I1),L=K,K1)
663:        write(IOT,7260) BIN(I,2,3), (SETALY(I,L,I1),L=K,K1)
664:      end if
665:      100 continue
666: c##<2007/03/14:PN3:
667:      do LL = 1, NL
668:        TOTVAL(LL) = 0
669:        TOTERR(LL) = 0
670:      end do
671:      LL = 0
672:      do L = K, K1
673:        LL = LL + 1
674:        do I = 1, NE3
675:          TOTVAL(LL) = TOTVAL(LL) + ETALY(I,L,I1)
676:          TOTERR(LL) = TOTERR(LL) +
677:          &      (ETALY(I,L,I1)*SETALY(I,L,I1))**2
678:        end do
679:        if ( TOTVAL(LL).gt.C.D0 ) then
680:          TOTERR(LL) = sqrt(TOTERR(LL)) / TOTVAL(LL)
681:        else
682:          TOTERR(LL) = 0
683:        end if
684:      end do
685:      write(IOT,7640) (TOTVAL(I),I=1,LL)
686:      write(IOT,7300) (TOTERR(I),I=1,LL)
687: c##>
688:      110 continue
689: c .....
690:      120 continue
691: c
692:      return
693:      end
694: C
695: C-----
696: C
697:      subroutine STLPR4( IOT, ETALY, SETALY,NBINMX,BIN, RNM, NE1,
698:      &      LABEL1,NE2, LABEL2,NE3, LABEL3,NE4,
699: c##<2007/03/14:PN3:
700: c## &      LABEL4 )
701:      &      LABEL4,
702:      &      NMT, NMTP, NMTPN, NUC, NPATOM,NUCPN,
703:      &      NUCID, NATMT, NCIDPN )
704: c##>
705: C=====
706: C what to do: edit special tally having 4 dimensions.
707: C called in : stlsum
708: C history: programmed by M.Sasaki (Sep 1995)
709: C update:
710: C 29 Jul 1997: using uninitialized variable "iI2"
711: C=====
712:      implicit real*8(A-H,O-Z)
713:      parameter( NL = 8 )
714: C
715:      real*8 ETALY(NE4,NE3,NE2,NE1)

```

```

716:      real*8 SETALY(NE4,NE3,NE2,NE1)
717: C
718:      character(*) LABEL1
719:      character(*) LABEL2
720:      character(*) LABEL3
721:      character(*) LABEL4
722:      real*8 BIN(NBINMX,2,8)
723:      character RNM(NBINMX)*128
724: c##<2007/03/14:PN3:
725:      real*8 TOTVAL(NL), TOTERR(NL)
726:      integer NATMT(NPATOM)
727:      character HSLAB1*5, HSLAB0(8,2)*10
728:      character NUCID(NUC)*16, NCIDPN(NUCPN)*16
729:      character HNCIDZA*7, HW1*7, HW2*7, CHSYMB*2
730:      external HNCIDZA, CHSYMB
731: c##>
732: c
733: C-----
734: C
735:      7000 format(/1X,4(:' == ',A8,I4))
736:      7020 format(/1X,' == ',A8,I4,3X,A)
737:      7040 format(/1X,' == ',A8,I4,3X,1P,' FROM ',E11.4,' TO ',E11.4)
738:      7060 format(/1X,' == ',A8,I4,3X,' FROM ',I11,' TO ',I11)
739:      7080 format(/1X,2(' == ',A8,I4,3X),A)
740:      7100 format(/1X,2(' == ',A8,I4,3X),1P,' FROM ',E11.4,' TO ',E11.4)
741:      7120 format(/1X,2(' == ',A8,I4,3X),' FROM ',I11,' TO ',I11)
742: c##<2007/03/14:PN3:
743:      7500 format(/1X,1(' == ',A8,I4,3X),2X,A5,' FROM ',I5,' TO ',I5)
744:      7520 format(/1X,2(' == ',A8,I4,3X),2X,A5,' FROM ',I5,' TO ',I5)
745:      7561 format(/1X,1(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
746:      7562 format(/1X,2(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
747: c##>
748: c
749:      7140 format(/2X,A10,7X,8(1X,A8,I3,1X)/)
750:      7160 format(2X,17X,8(1X,A12)/)
751:      7180 format(2X,17X,1P,8(E12.4,1X))
752:      7200 format(2X,17X,8(I12,1X))
753:      7220 format()
754:      7240 format(1X,I4,1X,1P,E12.4,8E13.5)
755: c##<2007/03/14:PN3:
756: c## 7260 format(1X,I4,1X,I12,8E13.5)
757:      7260 format(1X,I4,1X,I12,1P,8E13.5)
758: c##>
759:      7280 format(1X,I4,1X,A12,1P,8E13.5)
760:      7300 format(1X,I4,13X,1P,8E13.5)
761:      7320 format(1X,4X,1X,1P,E12.4,0P,8(' (',F7.3,'%') ':))
762:      7340 format(1X,4X,1X,I12,0P,8(' (',F7.3,'%') ':))
763:      7360 format(1X,17X,8(' (',F7.3,'%') ':))
764: c##<2007/03/14:PN3:
765:      7440 format(1X,I4,3X,A5,I5,1P,8E13.5)
766:      7460 format(1X,7X,A5,I5,8(' (',F7.3,'%') ':))
767:      7480 format(2X,17X,8(2X,A10,1X))
768:      7600 format(1X,I4,3X,A10,1P,8E13.5)
769:      7620 format(1X,7X,A10,8(' (',F7.3,'%') ':))
770:      7640 format(1X,' TOTAL',9X,1P,8E13.5)
771: c##>
772: c
773: c##<2007/03/14:PN3:
774:      NMT1 = NMT + NMTP
775:      NMT2 = NMT1 + NMTPN
776: C
777: c##>
778:      do 130 I1 = 1, NE1
779:        if ( LABEL1(1:3).eq.'REG' ) then
780:          write(IOT,7020) LABEL1, I1, RNM(I1) (1:ICLEN2(RNM(I1)))

```


src/shared/stlprx.f

```

781:         else if ( LABEL1(1:3).eq.'GEN' .or.
782:             & LABEL1.eq.'SOURCE'.or.
783:             & LABEL1.eq.'MKR-REG'.or.
784: c##<2007/03/14:PN3:
785: c## & LABEL1.eq.'SRC-REG' ) then
786:             & LABEL1.eq.'SRC-REG'.or.
787:             & LABEL1.eq.'PRD-REG' ) then
788: c##>
789:             write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
790:         else if ( LABEL1(1:3).eq.'POI' ) then
791:             write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
792: c##<2007/03/14:PN3:
793:         else if ( LABEL1.eq.'PRD-NUCL' ) then
794:             NCBB1 = nint(BIN(I1,1,1))
795:             NCBB2 = nint(BIN(I1,2,1))
796:             LL = 1
797:             if ( NCBB1.lt.1000 ) then
798:                 HW1 = HNCIDZA(NUCID(NCBB1))
799:                 HW2 = HNCIDZA(NUCID(NCBB2))
800:                 HSLAB0(LL,1)='N: '//HW1
801:                 HSLAB0(LL,2)='N: '//HW2
802:             else if ( NCBB1.lt.10000 ) then
803:                 HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
804:                 HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
805:                 HSLAB0(LL,1)='PN: '//HW1(1:6)
806:                 HSLAB0(LL,2)='PN: '//HW2(1:6)
807:             else
808:                 HW1 = CHSYMB(NATMT(NCBB1-10000))//
809:                 HW2 = CHSYMB(NATMT(NCBB2-10000))//
810:                 HSLAB0(LL,1)='P: '//HW1
811:                 HSLAB0(LL,2)='P: '//HW2
812:             end if
813:             write(IOT,7561) LABEL1, I1, HSLAB0(1,1), HSLAB0(1,2)
814:         else if ( LABEL1.eq.'PRD-REACT' ) then
815:             MTBB1 = nint(BIN(I1,1,1))
816:             MTBB2 = nint(BIN(I1,2,1))
817:             HSLAB1 = ' MT'
818:             if ( MTBB1.le.NMT ) then
819:                 HSLAB1 = ' N-MT'
820:             else if ( MTBB1.le.NMT1 ) then
821:                 HSLAB1 = ' P-MT'
822:             MTBB1 = MTBB1 - NMT
823:             MTBB2 = MTBB2 - NMT
824:             else if ( MTBB1.le.NMT2 ) then
825:                 HSLAB1 = 'PN-MT'
826:                 MTBB1 = MTBB1 - NMT1
827:                 MTBB2 = MTBB2 - NMT1
828:             end if
829:             write(IOT,7500) LABEL1, I1, HSLAB1, MTBB1, MTBB2
830: c##>
831:         else
832:             write(IOT,7040) LABEL1, I1, (BIN(I1,JJ,1),JJ=1,2)
833:         end if
834:         do 120 I2 = 1, NE2
835:             if ( LABEL2(1:3).eq.'REG' ) then
836:                 write(IOT,7080) LABEL1, I1, LABEL2, I2,
837:                 & RNM(I2) (1:ICLEN2(RNM(I2)))
838:             else if ( LABEL2(1:3).eq.'GEN'.or.
839:                 & LABEL2.eq.'SOURCE'.or.
840:                 & LABEL2.eq.'MKR-REG'.or.
841: c##<2007/03/14:PN3:
842: c## & LABEL2.eq.'SRC-REG' ) then
843:                 & LABEL2.eq.'SRC-REG'.or.
844:                 & LABEL2.eq.'PRD-REG' ) then
845: c##>

```

```

846:             write(IOT,7120) LABEL1, I1, LABEL2, I2,
847:             & (NINT(BIN(I2,JJ,2)),JJ=1,2)
848:         else if ( LABEL2(1:3).eq.'POI' ) then
849:             write(IOT,7120) LABEL1, I1, LABEL2, I2,
850:             & (NINT(BIN(I2,JJ,2)),JJ=1,2)
851: c##<2007/03/14:PN3:
852:         else if ( LABEL2.eq.'PRD-NUCL' ) then
853:             NCBB1 = nint(BIN(I2,1,2))
854:             NCBB2 = nint(BIN(I2,2,2))
855:             LL = 1
856:             if ( NCBB1.lt.1000 ) then
857:                 HW1 = HNCIDZA(NUCID(NCBB1))
858:                 HW2 = HNCIDZA(NUCID(NCBB2))
859:                 HSLAB0(LL,1)='N: '//HW1
860:                 HSLAB0(LL,2)='N: '//HW2
861:             else if ( NCBB1.lt.10000 ) then
862:                 HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
863:                 HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
864:                 HSLAB0(LL,1)='PN: '//HW1(1:6)
865:                 HSLAB0(LL,2)='PN: '//HW2(1:6)
866:             else
867:                 HW1 = CHSYMB(NATMT(NCBB1-10000))//
868:                 HW2 = CHSYMB(NATMT(NCBB2-10000))//
869:                 HSLAB0(LL,1)='P: '//HW1
870:                 HSLAB0(LL,2)='P: '//HW2
871:             end if
872:             write(IOT,7562) LABEL1, I1, LABEL2, I2, HSLAB0(1,1),
873:             & HSLAB0(1,2)
874:         else if ( LABEL2.eq.'PRD-REACT' ) then
875:             MTBB1 = nint(BIN(I2,1,2))
876:             MTBB2 = nint(BIN(I2,2,2))
877:             HSLAB1 = ' MT'
878:             if ( MTBB1.le.NMT ) then
879:                 HSLAB1 = ' N-MT'
880:             else if ( MTBB1.le.NMT1 ) then
881:                 HSLAB1 = ' P-MT'
882:             MTBB1 = MTBB1 - NMT
883:             MTBB2 = MTBB2 - NMT
884:             else if ( MTBB1.le.NMT2 ) then
885:                 HSLAB1 = 'PN-MT'
886:                 MTBB1 = MTBB1 - NMT1
887:                 MTBB2 = MTBB2 - NMT1
888:             end if
889:             write(IOT,7520) LABEL1, I1, LABEL2, I2,
890:             & HSLAB1, MTBB1, MTBB2
891: c##>
892:         else
893:             write(IOT,7100) LABEL1, I1, LABEL2, I2,
894:             & (BIN(I2,JJ,2),JJ=1,2)
895:         end if
896:
897: c .....
898:         do 110 K = 1, NE3, NL
899:             K1 = MIN(NE3,K+NL-1)
900:             write(IOT,7140) LABEL4, (LABEL3,L,L=K,K1)
901:             if ( LABEL3(1:3).eq.'REG' ) then
902:                 write(IOT,7160) (RNM(L)(1:12),L=K,K1)
903:             else if ( LABEL3(1:3).eq.'GEN'.or.
904:                 & LABEL3.eq.'SOURCE'.or.
905:                 & LABEL3.eq.'MKR-REG'.or.
906: c##<2007/03/14:PN3:
907: c## & LABEL3.eq.'SRC-REG' ) then
908:                 & LABEL3.eq.'SRC-REG'.or.
909:                 & LABEL3.eq.'PRD-REG' ) then
910: c##>

```

src/shared/stlprx.f

```

911:      write(IOT,7200) (NINT(BIN(L,1,3)),L=K,K1)
912:      write(IOT,7200) (NINT(BIN(L,2,3)),L=K,K1)
913:      write(IOT,7220)
914:      else if ( LABEL3.eq.'POI' ) then
915:        write(IOT,7200) (NINT(BIN(L,1,3)),L=K,K1)
916:        write(IOT,7220)
917: c##<2007/03/14:PN3:
918:      else if ( LABEL3.eq.'PRD-NUCL' ) then
919:        LL = 0
920:        do L = K, K1
921:          NCBB1 = nint(BIN(L,1,3))
922:          NCBB2 = nint(BIN(L,2,3))
923:          LL = LL + 1
924:          if ( NCBB1.lt.1000 ) then
925:            HW1 = HNCIDZA(NUCID(NCBB1))
926:            HW2 = HNCIDZA(NUCID(NCBB2))
927:            HSLAB0(LL,1)='N: '//HW1
928:            HSLAB0(LL,2)='N: '//HW2
929:          else if ( NCBB1.lt.10000 ) then
930:            HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
931:            HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
932:            HSLAB0(LL,1)='PN: '//HW1(1:6)
933:            HSLAB0(LL,2)='PN: '//HW2(1:6)
934:          else
935:            HW1 = CHSYMB(NATMT(NCBB1-10000))//''
936:            HW2 = CHSYMB(NATMT(NCBB2-10000))//''
937:            HSLAB0(LL,1)='P: '//HW1
938:            HSLAB0(LL,2)='P: '//HW2
939:          end if
940:        end do
941:        write(IOT,7480) (HSLAB0(L,1),L=1,LL)
942:        write(IOT,7480) (HSLAB0(L,2),L=1,LL)
943:        write(IOT,7280)
944:      else if ( LABEL3.eq.'PRD-REACT' ) then
945:        LL = 0
946:        do L = K, K1
947:          MTBB1 = nint(BIN(L,1,3))
948:          MTBB2 = nint(BIN(L,2,3))
949:          HSLAB1 = ' MT'
950:          if ( MTBB1.le.NMT ) then
951:            HSLAB1 = ' N-MT'
952:          else if ( MTBB1.le.NMT1 ) then
953:            HSLAB1 = ' P-MT'
954:            MTBB1 = MTBB1 - NMT
955:            MTBB2 = MTBB2 - NMT
956:          else if ( MTBB1.le.NMT2 ) then
957:            HSLAB1 = 'PN-MT'
958:            MTBB1 = MTBB1 - NMT1
959:            MTBB2 = MTBB2 - NMT1
960:          end if
961:          LL = LL + 1
962:          write(HSLAB0(LL,1),'(A5,I5)') HSLAB1, MTBB1
963:          write(HSLAB0(LL,2),'(A5,I5)') HSLAB1, MTBB2
964:        end do
965:        write(IOT,7480) (HSLAB0(L,1),L=1,LL)
966:        write(IOT,7480) (HSLAB0(L,2),L=1,LL)
967:        write(IOT,7220)
968: c##>
969:      else
970:        write(IOT,7180) (BIN(L,1,3),L=K,K1)
971:        write(IOT,7180) (BIN(L,2,3),L=K,K1)
972:        write(IOT,7220)
973:      end if
974: *VOCL LOOP, SCALAR
975:      do 100 I = 1, NE4

```

```

976:      if ( LABEL4(1:3).eq.'REG' ) then
977:        write(IOT,7280) I, RNM(I) (1:12),
978:          & (ETALY(I,L,I2,I1),L=K,K1)
979:        write(IOT,7360) (SETALY(I,L,I2,I1),L=K,K1)
980:      else if ( LABEL4(1:3).eq.'GEN' .or.
981:        & LABEL4.eq.'SOURCE' .or.
982:        & LABEL4.eq.'MKR-REG' .or.
983: c##<2007/03/14:PN3:
984: c## & LABEL4.eq.'SRC-REG' ) then
985: c## & LABEL4.eq.'SRC-REG' .or.
986: c## & LABEL4.eq.'PRD-REG' ) then
987: c##>
988:        write(IOT,7260) I, NINT(BIN(I,1,4)),
989:          & (ETALY(I,L,I2,I1),L=K,K1)
990:        write(IOT,7340) NINT(BIN(I,2,4)),
991:          & (SETALY(I,L,I2,I1),L=K,K1)
992:      else if ( LABEL4(1:3).eq.'POI' ) then
993:        write(IOT,7260) I, NINT(BIN(I,1,4)),
994:          & (ETALY(I,L,I2,I1),L=K,K1)
995:        write(IOT,7360)
996:          & (SETALY(I,L,I2,I1),L=K,K1)
997: c##<2007/03/14:PN3:
998:      else if ( LABEL4.eq.'PRD-NUCL' ) then
999:        NCBB1 = nint(BIN(I,1,4))
1000:        NCBB2 = nint(BIN(I,2,4))
1001:        LL = 1
1002:        if ( NCBB1.lt.1000 ) then
1003:          HW1 = HNCIDZA(NUCID(NCBB1))
1004:          HW2 = HNCIDZA(NUCID(NCBB2))
1005:          HSLAB0(LL,1)='N: '//HW1
1006:          HSLAB0(LL,2)='N: '//HW2
1007:        else if ( NCBB1.lt.10000 ) then
1008:          HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1009:          HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1010:          HSLAB0(LL,1)='PN: '//HW1(1:6)
1011:          HSLAB0(LL,2)='PN: '//HW2(1:6)
1012:        else
1013:          HW1 = CHSYMB(NATMT(NCBB1-10000))//''
1014:          HW2 = CHSYMB(NATMT(NCBB2-10000))//''
1015:          HSLAB0(LL,1)='P: '//HW1
1016:          HSLAB0(LL,2)='P: '//HW2
1017:        end if
1018:        write(IOT,7600) I, HSLAB0(1,1),
1019:          & (ETALY(I,L,I2,I1),L=K,K1)
1020:        write(IOT,7620) HSLAB0(1,2),
1021:          & (SETALY(I,L,I2,I1),L=K,K1)
1022:      else if ( LABEL4.eq.'PRD-REACT' ) then
1023:        MTBB1 = nint(BIN(I,1,4))
1024:        MTBB2 = nint(BIN(I,2,4))
1025:        HSLAB1 = ' MT'
1026:        if ( MTBB1.le.NMT ) then
1027:          HSLAB1 = ' N-MT'
1028:        else if ( MTBB1.le.NMT1 ) then
1029:          HSLAB1 = ' P-MT'
1030:          MTBB1 = MTBB1 - NMT
1031:          MTBB2 = MTBB2 - NMT
1032:        else if ( MTBB1.le.NMT2 ) then
1033:          HSLAB1 = 'PN-MT'
1034:          MTBB1 = MTBB1 - NMT1
1035:          MTBB2 = MTBB2 - NMT1
1036:        end if
1037:        write(IOT,7440) I, HSLAB1, MTBB1,
1038:          & (ETALY(I,L,I2,I1),L=K,K1)
1039:        write(IOT,7460) HSLAB1, MTBB2,
1040:          & (SETALY(I,L,I2,I1),L=K,K1)

```

src/shared/stlprx.f

```

1041: c##>
1042:         else
1043:             write(IOT,7240) I, BIN(I,1,4),
1044:             &             (ETALY(I,L,I2,I1),L=K,K1)
1045:             write(IOT,7320) BIN(I,2,4),
1046:             &             (SETALY(I,L,I2,I1),L=K,K1)
1047:         end if
1048:     100 continue
1049: c##<2007/03/14:PN3:
1050:     do LL = 1, NL
1051:         TOTVAL(LL) = 0
1052:         TOTERR(LL) = 0
1053:     end do
1054:     LL = 0
1055:     do L = K, K1
1056:         LL = LL + 1
1057:         do I = 1, NE4
1058:             TOTVAL(LL) = TOTVAL(LL) + ETALY(I,L,I2,I1)
1059:             TOTERR(LL) = TOTERR(LL) +
1060:             &             (ETALY(I,L,I2,I1)*SETALY(I,L,I2,I1))**2
1061:         end do
1062:         if ( TOTVAL(LL).gt.0.D0 ) then
1063:             TOTERR(LL) = sqrt(TOTERR(LL)) * TOTVAL(LL)
1064:         else
1065:             TOTERR(LL) = 0
1066:         end if
1067:     end do
1068:     write(IOT,7640) (TOTVAL(I),I=1,LL)
1069:     write(IOT,7360) (TOTERR(I),I=1,LL)
1070: c##>
1071:     110 continue
1072: c .....
1073:     120 continue
1074:     130 continue
1075: c
1076:     return
1077: end
1078: C
1079: C-----
1080: C
1081:     subroutine STLPR5( IOT, ETALY, SETALY,NBINMX,BIN, RNM, NE1,
1082:     & LABEL1,NE2, LABEL2,NE3, LABEL3,NE4,
1083: c##<2007/03/14:PN3:
1084: c## & LABEL4,NE5, LABEL5 )
1085: & LABEL4,NE5, LABEL5,
1086: & NMT, NMTP, NMTPN, NUC, NPATOM,NUCPN)
1087: & NUCID, NATMT, NCIDPN )
1088: c##>
1089: C=====
1090: C what to do: edit special tally having 5 dimensions.
1091: C called in : stlsum
1092: C history: programmed by M.Sasaki (Sep 1995)
1093: C-----
1094:     implicit real*8(A-H,O-Z)
1095:     parameter( NL = 8 )
1096: C
1097:     real*8 ETALY(NE5,NE4,NE3,NE2,NE1)
1098:     real*8 SETALY(NE5,NE4,NE3,NE2,NE1)
1099: C
1100:     character*(*) LABEL1
1101:     character*(*) LABEL2
1102:     character*(*) LABEL3
1103:     character*(*) LABEL4
1104:     character*(*) LABEL5
1105:     real*8 BIN(NBINMX,2,8)
1106:     character RNM(NBINMX)*128
1107: c##<2007/03/14:PN3:
1108:     real*8 TOTVAL(NL), TOTERR(NL)
1109:     integer NATMT(NPATOM)
1110:     character HSLAB1*5, HSLAB0(8,2)*10
1111:     character NUCID(NUC)*16, NCIDPN(NUCPN)*16
1112:     character HNCIDZA*7, HW1*7, HW2*7, CHSYMB*2
1113:     external HNCIDZA, CHSYMB
1114: c##>
1115: c
1116: C-----
1117: C
1118:     7000 format(/1X,4(:' == ',A8,I4))
1119:     7020 format(/1X,' == ',A8,I4,3X,A)
1120:     7040 format(/1X,' == ',A8,I4,3X,1P,' FROM ',E11.4,' TO ',E11.4)
1121:     7060 format(/1X,' == ',A8,I4,3X,' FROM ',I11,' TO ',I11)
1122:     7080 format(/1X,2(' == ',A8,I4,3X),A)
1123:     7100 format(/1X,2(' == ',A8,I4,3X),1P,' FROM ',E11.4,' TO ',E11.4)
1124:     7120 format(/1X,2(' == ',A8,I4,3X),' FROM ',I11,' TO ',I11)
1125:     7140 format(/1X,3(' == ',A8,I4,3X),A)
1126:     7160 format(/1X,3(' == ',A8,I4,3X),1P,' FROM ',E11.4,' TO ',E11.4)
1127:     7180 format(/1X,3(' == ',A8,I4,3X),' FROM ',I11,' TO ',I11)
1128: c##<2007/03/14:PN3:
1129:     7500 format(/1X,1(' == ',A8,I4,3X),2X,A5,' FROM ',I5,' TO ',I5)
1130:     7520 format(/1X,2(' == ',A8,I4,3X),2X,A5,' FROM ',I5,' TO ',I5)
1131:     7540 format(/1X,3(' == ',A8,I4,3X),2X,A5,' FROM ',I5,' TO ',I5)
1132:     7561 format(/1X,1(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
1133:     7562 format(/1X,2(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
1134:     7563 format(/1X,3(' == ',A8,I4,3X),2X,' FROM ',A10,' TO ',A10)
1135: c##>
1136: c
1137:     7200 format(/2X,A10,7X,8(1X,A8,I3,1X)/)
1138:     7220 format(2X,17X,8(1X,A12)/)
1139:     7240 format(2X,17X,1P,8(E12.4,1X))
1140:     7260 format(2X,17X,8(I12,1X))
1141:     7280 format()
1142:     7300 format(1X,I4,1X,1P,E12.4,8E13.5)
1143: c##<2007/03/14:PN3:
1144: c## 7320 format(1X,I4,1X,I12,8E13.5)
1145: 7320 format(1X,I4,1X,I12,1P,8E13.5)
1146: c##>
1147:     7340 format(1X,I4,1X,A12,1P,8E13.5)
1148:     7360 format(1X,I4,13X,1P,8E13.5)
1149:     7380 format(1X,4X,1X,1P,E12.4,0P,8(' (',F7.3,'%') ':))
1150:     7400 format(1X,4X,1X,I12,0P,8(' (',F7.3,'%') ':))
1151:     7420 format(1X,17X,8(' (',F7.3,'%') ':))
1152: c##<2007/03/14:PN3:
1153:     7440 format(1X,I4,3X,A5,I5,1P,8E13.5)
1154:     7460 format(1X,7X,A5,I5,8(' (',F7.3,'%') ':))
1155:     7480 format(2X,17X,8(2X,A10,1X))
1156:     7600 format(1X,I4,3X,A10,1P,8E13.5)
1157:     7620 format(1X,7X,A10,8(' (',F7.3,'%') ':))
1158:     7640 format(1X,' TOTAL',9X,1P,8E13.5)
1159: c##>
1160: c
1161: c##<2007/03/14:PN3:
1162:     NMT1 = NMT + NMTP
1163:     NMT2 = NMT1 + NMTPN
1164: C
1165: c##>
1166:     do 140 I1 = 1, NE1
1167: c     write(IOT,7000) LABEL1, I1
1168:     if ( LABEL1(1:3).eq.'REG' ) then
1169:         write(IOT,7020) LABEL1, I1, RNM(I1) (1:ICLEN2(RNM(I1)))
1170:     else if ( LABEL1(1:3).eq.'GEN' .or.

```

src/shared/stlprx.f

```

1171:      &          LABEL1.eq.'SOURCE'.or.
1172:      &          LABEL1.eq.'MKR-REG'.or.
1173: c##<2007/03/14:PN3:
1174: c## &          LABEL1.eq.'SRC-REG' ) then
1175:      &          LABEL1.eq.'SRC-REG'.or.
1176:      &          LABEL1.eq.'PRD-REG' ) then
1177: c##>
1178:      write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
1179:      else if ( LABEL1(1:3).eq.'POI' ) then
1180:      write(IOT,7060) LABEL1, I1, (NINT(BIN(I1,JJ,1)),JJ=1,2)
1181: c##<2007/03/14:PN3:
1182:      else if ( LABEL1.eq.'PRD-NUCL' ) then
1183:      NCBB1 = nint(BIN(I1,1,1))
1184:      NCBB2 = nint(BIN(I1,2,1))
1185:      LL = 1
1186:      if ( NCBB1.lt.1000 ) then
1187:      HW1 = HNCIDZA(NUCID(NCBB1))
1188:      HW2 = HNCIDZA(NUCID(NCBB2))
1189:      HSLAB0(LL,1)='N: '//HW1
1190:      HSLAB0(LL,2)='N: '//HW2
1191:      else if ( NCBB1.lt.10000 ) then
1192:      HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1193:      HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1194:      HSLAB0(LL,1)='PN: '//HW1(1:6)
1195:      HSLAB0(LL,2)='PN: '//HW2(1:6)
1196:      else
1197:      HW1 = CHSYMB(NATMT(NCBB1-10000))//
1198:      HW2 = CHSYMB(NATMT(NCBB2-10000))//
1199:      HSLAB0(LL,1)='P: '//HW1
1200:      HSLAB0(LL,2)='P: '//HW2
1201:      end if
1202:      write(IOT,7561) LABEL1, I1, HSLAB0(1,1), HSLAB0(1,2)
1203:      else if ( LABEL1.eq.'PRD-REACT' ) then
1204:      MTBB1 = nint(BIN(I1,1,1))
1205:      MTBB2 = nint(BIN(I1,2,1))
1206:      HSLAB1 = ' MT'
1207:      if ( MTBB1.le.NMT ) then
1208:      HSLAB1 = ' N-MT'
1209:      else if ( MTBB1.le.NMT1 ) then
1210:      HSLAB1 = ' P-MT'
1211:      MTBB1 = MTBB1 - NMT
1212:      MTBB2 = MTBB2 - NMT
1213:      else if ( MTBB1.le.NMT2 ) then
1214:      HSLAB1 = ' PN-MT'
1215:      MTBB1 = MTBB1 - NMT1
1216:      MTBB2 = MTBB2 - NMT1
1217:      end if
1218:      write(IOT,7500) LABEL1, I1, HSLAB1, MTBB1, MTBB2
1219: c##>
1220:      else
1221:      write(IOT,7040) LABEL1, I1, (BIN(I1,JJ,1),JJ=1,2)
1222:      end if
1223:      do 130 I2 = 1, NE2
1224:      if ( LABEL2(1:3).eq.'REG' ) then
1225:      write(IOT,7080) LABEL1, I1, LABEL2, I2,
1226:      &          RNM(I2) (1:ICLEN2(RNM(I2)))
1227:      else if ( LABEL2(1:3).eq.'GEN'.or.
1228:      &          LABEL2.eq.'SOURCE'.or.
1229:      &          LABEL2.eq.'MKR-REG'.or.
1230: c##<2007/03/14:PN3:
1231: c## &          LABEL2.eq.'SRC-REG' ) then
1232:      &          LABEL2.eq.'SRC-REG'.or.
1233:      &          LABEL2.eq.'PRD-REG' ) then
1234: c##>
1235:      write(IOT,7120) LABEL1, I1, LABEL2, I2,

```

```

1236:      &          (NINT(BIN(I2,JJ,2)),JJ=1,2)
1237:      else if ( LABEL2(1:3).eq.'POI' ) then
1238:      write(IOT,7120) LABEL1, I1, LABEL2, I2,
1239:      &          (NINT(BIN(I2,JJ,2)),JJ=1,2)
1240: c##<2007/03/14:PN3:
1241:      else if ( LABEL2.eq.'PRD-NUCL' ) then
1242:      NCBB1 = nint(BIN(I2,1,2))
1243:      NCBB2 = nint(BIN(I2,2,2))
1244:      LL = 1
1245:      if ( NCBB1.lt.1000 ) then
1246:      HW1 = HNCIDZA(NUCID(NCBB1))
1247:      HW2 = HNCIDZA(NUCID(NCBB2))
1248:      HSLAB0(LL,1)='N: '//HW1
1249:      HSLAB0(LL,2)='N: '//HW2
1250:      else if ( NCBB1.lt.10000 ) then
1251:      HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1252:      HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1253:      HSLAB0(LL,1)='PN: '//HW1(1:6)
1254:      HSLAB0(LL,2)='PN: '//HW2(1:6)
1255:      else
1256:      HW1 = CHSYMB(NATMT(NCBB1-10000))//
1257:      HW2 = CHSYMB(NATMT(NCBB2-10000))//
1258:      HSLAB0(LL,1)='P: '//HW1
1259:      HSLAB0(LL,2)='P: '//HW2
1260:      end if
1261:      write(IOT,7562) LABEL1, I1, LABEL2, I2, HSLAB0(1,1),
1262:      &          HSLAB0(1,2)
1263:      else if ( LABEL2.eq.'PRD-REACT' ) then
1264:      MTBB1 = nint(BIN(I2,1,2))
1265:      MTBB2 = nint(BIN(I2,2,2))
1266:      HSLAB1 = ' MT'
1267:      if ( MTBB1.le.NMT ) then
1268:      HSLAB1 = ' N-MT'
1269:      else if ( MTBB1.le.NMT1 ) then
1270:      HSLAB1 = ' P-MT'
1271:      MTBB1 = MTBB1 - NMT
1272:      MTBB2 = MTBB2 - NMT
1273:      else if ( MTBB1.le.NMT2 ) then
1274:      HSLAB1 = ' PN-MT'
1275:      MTBB1 = MTBB1 - NMT1
1276:      MTBB2 = MTBB2 - NMT1
1277:      end if
1278:      write(IOT,7520) LABEL1, I1, LABEL2, I2,
1279:      &          HSLAB1, MTBB1, MTBB2
1280: c##>
1281:      else
1282:      write(IOT,7100) LABEL1, I1, LABEL2, I2,
1283:      &          (BIN(I2,JJ,2),JJ=1,2)
1284:      end if
1285:      do 120 I3 = 1, NE3
1286:      if ( LABEL2(1:3).eq.'REG' ) then
1287:      if ( LABEL3(1:3).eq.'REG' ) then
1288:      write(IOT,7140) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1289:      &          RNM(I3) (1:ICLEN2(RNM(I3)))
1290:      else if ( LABEL3(1:3).eq.'GEN'.or.
1291:      &          LABEL3.eq.'SOURCE'.or.
1292:      &          LABEL3.eq.'MKR-REG'.or.
1293: c##<2007/03/14:PN3:
1294: c## &          LABEL3.eq.'SRC-REG' ) then
1295:      &          LABEL3.eq.'SRC-REG'.or.
1296:      &          LABEL3.eq.'PRD-REG' ) then
1297: c##>
1298:      write(IOT,7180) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1299: c##<2007/03/14:PN3:
1300: c## &          (NINT(BIN(I3,JJ,2)),JJ=1,2)

```

src/shared/stlprx.f

```

1301:      &                (NINT(BIN(I3,JJ,3)),JJ=1,2)
1302: c##>
1303:      else if ( LABEL3(1:3).eq.'POI' ) then
1304:      write(IOT,7180) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1305: c##<2007/03/14:PN3:
1306: c## &                (NINT(BIN(I3,JJ,2)),JJ=1,2)
1307:      &                (NINT(BIN(I3,JJ,3)),JJ=1,2)
1308: c##>
1309: c##<2007/03/14:PN3:
1310:      else if ( LABEL3.eq.'PRD-NUCL' ) then
1311:      NCBB1 = nint(BIN(I3,1,3))
1312:      NCBB2 = nint(BIN(I3,2,3))
1313:      LL = 1
1314:      if ( NCBB1.lt.1000 ) then
1315:      HW1 = HNCIDZA(NUCID(NCBB1))
1316:      HW2 = HNCIDZA(NUCID(NCBB2))
1317:      HSLAB0(LL,1)='N: '//HW1
1318:      HSLAB0(LL,2)='N: '//HW2
1319:      else if ( NCBB1.lt.10000 ) then
1320:      HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1321:      HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1322:      HSLAB0(LL,1)='PN: '//HW1(1:6)
1323:      HSLAB0(LL,2)='PN: '//HW2(1:6)
1324:      else
1325:      HW1 = CHSYMB(NATMT(NCBB1-10000))//''
1326:      HW2 = CHSYMB(NATMT(NCBB2-10000))//''
1327:      HSLAB0(LL,1)='P: '//HW1
1328:      HSLAB0(LL,2)='P: '//HW2
1329:      end if
1330:      write(IOT,7563) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1331:      &                HSLAB0(1,1), HSLAB0(1,2)
1332:      else if ( LABEL3.eq.'PRD-REACT' ) then
1333:      MTBB1 = nint(BIN(I3,1,3))
1334:      MTBB2 = nint(BIN(I3,2,3))
1335:      HSLAB1 = ' MT'
1336:      if ( MTBB1.le.NMT ) then
1337:      HSLAB1 = ' N-MT'
1338:      else if ( MTBB1.le.NMT1 ) then
1339:      HSLAB1 = ' P-MT'
1340:      MTBB1 = MTBB1 - NMT
1341:      MTBB2 = MTBB2 - NMT
1342:      else if ( MTBB1.le.NMT2 ) then
1343:      HSLAB1 = 'PN-MT'
1344:      MTBB1 = MTBB1 - NMT1
1345:      MTBB2 = MTBB2 - NMT1
1346:      end if
1347:      write(IOT,7540) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1348:      &                HSLAB1, MTBB1, MTBB2
1349: c##>
1350:      else
1351:      write(IOT,7160) LABEL1, I1, LABEL2, I2, LABEL3, I3,
1352: c##<2007/03/14:PN3:
1353: c## &                (BIN(I3,JJ,2),JJ=1,2)
1354:      &                (BIN(I3,JJ,3),JJ=1,2)
1355: c##>
1356:      end if
1357: c .....
1358:      do 110 K = 1, NE4, NL
1359:      K1 = MIN(NE4,K+NL-1)
1360:      write(IOT,7200) LABEL5, (LABEL4,L,L=K,K1)
1361:      if ( LABEL4(1:3).eq.'REG' ) then
1362:      write(IOT,7220) (RNM(L)(1:12),L=K,K1)
1363:      else if ( LABEL4(1:3).eq.'GEN' .or.
1364:      & LABEL4.eq.'SOURCE'.or.
1365:      & LABEL4.eq.'MKR-REG'.or.

```

```

1366: c##<2007/03/14:PN3:
1367: c## & LABEL4.eq.'SRC-REG' ) then
1368:      & LABEL4.eq.'SRC-REG'.or.
1369:      & LABEL4.eq.'PRD-REG' ) then
1370: c##>
1371:      write(IOT,7260) (NINT(BIN(L,1,4)),L=K,K1)
1372:      write(IOT,7260) (NINT(BIN(L,2,4)),L=K,K1)
1373:      write(IOT,7280)
1374:      else if ( LABEL4(1:3).eq.'POI' ) then
1375:      write(IOT,7260) (NINT(BIN(L,1,4)),L=K,K1)
1376:      write(IOT,7280)
1377: c##<2007/03/14:PN3:
1378:      else if ( LABEL4.eq.'PRD-NUCL' ) then
1379:      LL = 0
1380:      do L = K, K1
1381:      NCBB1 = nint(BIN(L,1,4))
1382:      NCBB2 = nint(BIN(L,2,4))
1383:      LL = LL + 1
1384:      if ( NCBB1.lt.1000 ) then
1385:      HW1 = HNCIDZA(NUCID(NCBB1))
1386:      HW2 = HNCIDZA(NUCID(NCBB2))
1387:      HSLAB0(LL,1)='N: '//HW1
1388:      HSLAB0(LL,2)='N: '//HW2
1389:      else if ( NCBB1.lt.10000 ) then
1390:      HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1391:      HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1392:      HSLAB0(LL,1)='PN: '//HW1(1:6)
1393:      HSLAB0(LL,2)='PN: '//HW2(1:6)
1394:      else
1395:      HW1 = CHSYMB(NATMT(NCBB1-10000))//''
1396:      HW2 = CHSYMB(NATMT(NCBB2-10000))//''
1397:      HSLAB0(LL,1)='P: '//HW1
1398:      HSLAB0(LL,2)='P: '//HW2
1399:      end if
1400:      end do
1401:      write(IOT,7480) (HSLAB0(L,1),L=1,LL)
1402:      write(IOT,7480) (HSLAB0(L,2),L=1,LL)
1403:      write(IOT,7280)
1404:      else if ( LABEL4.eq.'PRD-REACT' ) then
1405:      LL = 0
1406:      do L = K, K1
1407:      MTBB1 = nint(BIN(L,1,4))
1408:      MTBB2 = nint(BIN(L,2,4))
1409:      HSLAB1 = ' MT'
1410:      if ( MTBB1.le.NMT ) then
1411:      HSLAB1 = ' N-MT'
1412:      else if ( MTBB1.le.NMT1 ) then
1413:      HSLAB1 = ' P-MT'
1414:      MTBB1 = MTBB1 - NMT
1415:      MTBB2 = MTBB2 - NMT
1416:      else if ( MTBB1.le.NMT2 ) then
1417:      HSLAB1 = 'PN-MT'
1418:      MTBB1 = MTBB1 - NMT1
1419:      MTBB2 = MTBB2 - NMT1
1420:      end if
1421:      LL = LL + 1
1422:      write(HSLAB0(LL,1),'(A5,I5)') HSLAB1, MTBB1
1423:      write(HSLAB0(LL,2),'(A5,I5)') HSLAB1, MTBB2
1424:      end do
1425:      write(IOT,7480) (HSLAB0(L,1),L=1,LL)
1426:      write(IOT,7480) (HSLAB0(L,2),L=1,LL)
1427:      write(IOT,7280)
1428: c##>
1429:      else
1430:      write(IOT,7240) (BIN(L,1,4),L=K,K1)

```

src/shared/stlprx.f

```

1431:      write(IOT,7240) (BIN(L,2,4),L=K,K1)
1432:      write(IOT,7280)
1433:    end if
1434: *VOCL LOOP, SCALAR
1435:    do 100 I = 1, NE5
1436:      if ( LABEL5(1:3).eq.'REG' ) then
1437:        write(IOT,7340) I, RNM(I) (1:12),
1438:          & (ETALY(I,L,I3,I2,I1),L=K,K1)
1439:        write(IOT,7420) (SETALY(I,L,I3,I2,I1),L=K,K1)
1440:      else if ( LABEL5(1:3).eq.'GEN' .or.
1441:        & LABEL5.eq.'SOURCE' .or.
1442:        & LABEL5.eq.'MKR-REG' .or.
1443: c##<2007/03/14:PN3:
1444: c## & LABEL5.eq.'SRC-REG' ) then
1445:        & LABEL5.eq.'SRC-REG' .or.
1446:        & LABEL5.eq.'PRD-REG' ) then
1447: c##>
1448:        write(IOT,7320) I, NINT(BIN(I,1,5)),
1449:          & (ETALY(I,L,I3,I2,I1),L=K,K1)
1450:        write(IOT,7400) NINT(BIN(I,2,5)),
1451:          & (SETALY(I,L,I3,I2,I1),L=K,K1)
1452:      else if ( LABEL5(1:3).eq.'POI' ) then
1453:        write(IOT,7320) I, NINT(BIN(I,1,5)),
1454:          & (ETALY(I,L,I3,I2,I1),L=K,K1)
1455:        write(IOT,7420)
1456:          & (SETALY(I,L,I3,I2,I1),L=K,K1)
1457: c##<2007/03/14:PN3:
1458:      else if ( LABEL5.eq.'PRD-NUCL' ) then
1459:        NCBB1 = nint(BIN(I,1,5))
1460:        NCBB2 = nint(BIN(I,2,5))
1461:        LL = 1
1462:        if ( NCBB1.lt.1000 ) then
1463:          HW1 = HNCIDZA(NUCID(NCBB1))
1464:          HW2 = HNCIDZA(NUCID(NCBB2))
1465:          HSLAB0(LL,1)='N: '//HW1
1466:          HSLAB0(LL,2)='N: '//HW2
1467:        else if ( NCBB1.lt.10000 ) then
1468:          HW1 = HNCIDZA(NCIDPN(NCBB1-1000))
1469:          HW2 = HNCIDZA(NCIDPN(NCBB2-1000))
1470:          HSLAB0(LL,1)='PN: '//HW1(1:6)
1471:          HSLAB0(LL,2)='PN: '//HW2(1:6)
1472:        else
1473:          HW1 = CHSYMB(NATMT(NCBB1-10000))//
1474:          HW2 = CHSYMB(NATMT(NCBB2-10000))//
1475:          HSLAB0(LL,1)='P: '//HW1
1476:          HSLAB0(LL,2)='P: '//HW2
1477:        end if
1478:        write(IOT,7600) I, HSLAB0(1,1),
1479:          & (ETALY(I,L,I3,I2,I1),L=K,K1)
1480:        write(IOT,7620) HSLAB0(1,2),
1481:          & (SETALY(I,L,I3,I2,I1),L=K,K1)
1482:      else if ( LABEL5.eq.'PRD-REACT' ) then
1483:        MTBB1 = nint(BIN(I,1,5))
1484:        MTBB2 = nint(BIN(I,2,5))
1485:        HSLAB1 = ' MT'
1486:        if ( MTBB1.le.NMT ) then
1487:          HSLAB1 = ' N-MT'
1488:        else if ( MTBB1.le.NMT1 ) then
1489:          HSLAB1 = ' P-MT'
1490:        MTBB1 = MTBB1 - NMT
1491:        MTBB2 = MTBB2 - NMT
1492:      else if ( MTBB1.le.NMT2 ) then
1493:        HSLAB1 = 'PN-MT'
1494:        MTBB1 = MTBB1 - NMT1
1495:        MTBB2 = MTBB2 - NMT1

```

```

1496:      end if
1497:      write(IOT,7440) I, HSLAB1, MTBB1,
1498:        & (ETALY(I,L,I3,I2,I1),L=K,K1)
1499:      write(IOT,7460) HSLAB1, MTBB2,
1500:        & (SETALY(I,L,I3,I2,I1),L=K,K1)
1501: c##>
1502:    else
1503:      write(IOT,7300) I, BIN(I,1,5),
1504:        & (ETALY(I,L,I3,I2,I1),L=K,K1)
1505:      write(IOT,7380) BIN(I,2,5),
1506:        & (SETALY(I,L,I3,I2,I1),L=K,K1)
1507:    end if
1508:    100 continue
1509: c##<2007/03/14:PN3:
1510:    do LL = 1, NL
1511:      TOTVAL(LL) = 0
1512:      TOTERR(LL) = 0
1513:    end do
1514:    LL = 0
1515:    do L = K, K1
1516:      LL = LL + 1
1517:      do I = 1, NE5
1518:        TOTVAL(LL) = TOTVAL(LL) + ETALY(I,L,I3,I2,I1)
1519:        TOTERR(LL) = TOTERR(LL) +
1520:          & (ETALY(I,L,I3,I2,I1)*SETALY(I,L,I3,I2,I1))**2
1521:      end do
1522:      if ( TOTVAL(LL).gt.0.D0 ) then
1523:        TOTERR(LL) = sqrt(TOTERR(LL)) / TOTVAL(LL)
1524:      else
1525:        TOTERR(LL) = 0
1526:      end if
1527:    end do
1528:    write(IOT,7640) (TOTVAL(I),I=1,LL)
1529:    write(IOT,7420) (TOTERR(I),I=1,LL)
1530: c##>
1531:    110 continue
1532: c .....
1533:    120 continue
1534:    130 continue
1535:    140 continue
1536: c
1537:    return
1538:  end
1539: c##<2007/03/14:PN3:
1540: c
1541:   character*7 function HNCIDZA( HNCID )
1542: c=====
1543: c   the effective name from nuclide identification is returned.
1544: c=====
1545: c
1546:   character HNCID*(*)
1547:   external ICLEN
1548: c
1549:   NW = ICLEN( HNCID )
1550:   if ( NW.ge.9 ) then
1551:     HNCIDZA = HNCID(1:5)//' '
1552:   else
1553:     HNCIDZA = HNCID(1:3)//' '
1554:   end if
1555: c
1556:   return
1557: end
1558: c##>

```

src/shared/stlsum.f

```

1:      subroutine STLsum( IOW, IETAL, NETALY, IDTAL, ETALY, NLETAL,
2:      & DTALY, NGENE, WSUMB, IBSUM, NETRV, METRV,
3:      & IETRV, ETRV, DWRK )
4: C==<MVP/GMVP>=====
5: C what to do: combine direct-tally of each batch to edited-tally.
6: C called in: talsum
7: C=====
8: C argument
9: C i iow : I/O unit for message output
10: C i ietal(*) : edited tally information ( data packet container )
11: C i netaly : number of edited tally
12: C i idtal(*) : direct tally information
13: C o etaly(netal,2) : edited tally ( sum & squared sum )
14: C i nletal : length of etaly(1,2)
15: C i dtaly : direct tally array
16: C i ngene : number of particle in a batch
17: C
18: C i IBSUM : number of batchs whose tally is taken
19: C      (NBATCH-NSKIP)
20: C i NETRV : Number of special (edited) tallies whose "real variance"'s
21: C      are estimated.
22: C i METRV : Total length of special (edited) tallies whose
23: C      "real variance"'s are estimated. (per batch length)
24: C i IETRV(2,NETRV+1) : Pointers for special (edited) tallies whose
25: C      "real variance"'s are estimated.
26: C      IETRV(1,*) : edited tally #.
27: C      IETRV(2,*) : tally is stored on
28: C      ETRV(IETRV(2,i):IETRV(2,i+1)-1,batch).
29: C o ETRV(METRV,*) : special tally storage for "real variance"
30: C      calculation.
31: C w dwrk : working array (length is the maximum length of e-tally)
32: C-----
33:      integer IETAL(*)
34:      integer IDTAL(*)
35:      real*8 ETALY(NLETAL,2)
36:      real*8 DTALY(*)
37:      integer IETRV(2,NETRV+1)
38:      real*8 ETRV(METRV,*)
39:      real*8 DWRK(*)
40:      real*8 WSUMB
41: C
42: C ... offset of direct-tally data in IDTAL(*)
43: C
44: CC      parameter( IDTOFF = 12 )
45: C
46: C ... offset of tally dimension dependent data in IDTAL(*)
47: C
48: CC      parameter( IDMOFF = 3 )
49: C
50:      include 'INC/_IDXOFF'
51: C
52: C ... local data ...
53: C
54: CCCC integer INFO(8)
55:      integer INFO(10)
56:      parameter( MAXDIM = 8 )
57: C
58:      character*6 TAG
59:      character PLAB*32
60:      integer NDIME(MAXDIM), NDIMD(MAXDIM), IEDT(MAXDIM)
61: C-----
62: C
63:      do 160 N = 1, NETALY
64: C
65:      TAG = ' '

```

```

66:      write(TAG,'(''.',i5)') N
67:      call CCOMP( TAG, LEN(TAG), TAG, IIF )
68:      LTAG = ICLEN(TAG)
69: C
70:      PLAB = 'TALLY'//TAG(:LTAG)
71:      call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
72:      if ( IRET.ne.0 ) go to 170
73: C
74: CCCCC call UNPKND( IETAL, 'INFO', 'I4', INFO, 8, NA, IRET )
75:      call UNPKND( IETAL, 'INFO', 'I4', INFO, 10, NA, IRET )
76:      if ( IRET.ne.0 ) go to 170
77: Check *****
78: C      write(iow,*) 'stlsum : info ',(info(iiii),iiii=1,8)
79: C *****
80: C
81: C
82: C ... combination of more than one d-tally is not supported now ...
83: Ccc      if( info(3).ne.0 ) then
84: Ccc      endif
85: C
86:      PLAB = 'DTALLY'//TAG(:LTAG)
87:      call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
88:      if ( IRET.ne.0 ) go to 170
89: C
90:      call UNPKPT( IETAL, 'D-TALLY', 'I4', LDT, NL, IRET )
91: C
92: C ... pointer to idtal & dtaly
93:      LIDT = IETAL(LDT+1)
94:      LDT = IETAL(LDT+2)
95: Check *****
96: C      write(iow,*) 'stlsum : lidt ',lidt,' ldt ',ldt
97: C *****
98: C
99: C ... loop for each dimension ...
100: C
101:      PLAB = 'DIMENSION'//TAG(:LTAG)
102:      call PCTLB( IETAL, PLAB, PLAB(:ICLEN2(PLAB)), IRET )
103:      if ( IRET.ne.0 ) go to 170
104: C
105:      do 110 J = 1, INFO(4)
106: C
107: C ... size of dimension (number of edited bins)
108:      call UNPKND( IETAL, 'INFO', 'I4', NDIME(J), 1, NA, IRET )
109:      if ( IRET.ne.0 ) go to 170
110: C
111: C ... edit information list
112:      call UNPKPT( IETAL, 'EDIT INFO', 'I4', IEDT(J), NL, IRET )
113:      if ( IRET.ne.0 ) go to 170
114: C
115: C ... original bin information list : dummy not used
116:      call UNPKPT( IETAL, 'EDIT INFO', 'I4', IDUM, NL, IRET )
117:      if ( IRET.ne.0 ) go to 170
118: C
119: C ... skip region name when this dimension is REGION
120:      if ( NDIME(J).lt.0 ) then
121:          NDIME(J) = -NDIME(J)
122:          do 100 JJ = 1, NDIME(J)
123:              call UNPKPT( IETAL, 'EDIT INFO', 'CH', IDUM, NL, IRET
124:              &
125:              if ( IRET.ne.0 ) go to 170
126:          100      continue
127:      end if
128: C
129:      110      continue
130: C

```

src/shared/stlsum.f

```

131:      IDT      = LIDT - 1
132:      if ( IDTAL(IDT+6).ne.INFO(4) ) then
133:        write(IOW,*) 'XXX(STLSUM) number of dimensions of ',
134:          &          ' edited-tally and direct-tally does not match.',
135:          &          ' tally : ', N, ' dim E ', INFO(3), ' D ',
136:          &          IDTAL(IDT+6)
137:        write(IOW,*) (IDTAL(IDT+J),J=1,IDTOFF)
138:        stop 666
139:      end if
140: C
141:      IDT1      = IDT + IDTOFF
142:      IDSIZ      = 1
143:      do 120 J = 1, IDTAL(IDT+6)
144:        NDIMD(J) = IDTAL(IDT1+2)
145:        IDT1      = IDT1 + IDMOFF + IDTAL(IDT1+3)
146:        IDSIZ      = IDSIZ * NDIMD(J)
147: 120 continue
148: C
149:      if ( INFO(3).eq.4 ) then
150: C ... special treatment for Feynman-alpha
151:        if ( INFO(4).eq.0 ) then
152:          call FEYNM0(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
153:            &          NGENE, DWRK(1), DWRK(IDSIZ+1) )
154:        else if ( INFO(4).eq.1 ) then
155:          call FEYNM1(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
156:            &          NGENE, DWRK(1), DWRK(IDSIZ+1),
157:            &          NDIME(1), NDIMD(1), IETAL(IEDT(1)) )
158:        else if ( INFO(4).eq.2 ) then
159:          call FEYNM2(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
160:            &          NGENE, DWRK(1), DWRK(IDSIZ+1),
161:            &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
162:            &          NDIME(2), NDIMD(2), IETAL(IEDT(2)) )
163:        else if ( INFO(4).eq.3 ) then
164:          call FEYNM3(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
165:            &          NGENE, DWRK(1), DWRK(IDSIZ+1),
166:            &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
167:            &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
168:            &          NDIME(3), NDIMD(3), IETAL(IEDT(3)) )
169:        else if ( INFO(4).eq.4 ) then
170:          call FEYNM4(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
171:            &          NGENE, DWRK(1), DWRK(IDSIZ+1),
172:            &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
173:            &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
174:            &          NDIME(3), NDIMD(3), IETAL(IEDT(3)),
175:            &          NDIME(4), NDIMD(4), IETAL(IEDT(4)) )
176:        else if ( INFO(4).eq.5 ) then
177:          call FEYNM5(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
178:            &          NGENE, DWRK(1), DWRK(IDSIZ+1),
179:            &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
180:            &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
181:            &          NDIME(3), NDIMD(3), IETAL(IEDT(3)),
182:            &          NDIME(4), NDIMD(4), IETAL(IEDT(4)),
183:            &          NDIME(5), NDIMD(5), IETAL(IEDT(5)) )
184:        else
185:          write(IOW,*)
186:          &          'XXX(STLSUM) Cannot handle tally of more than 5 dimension.'
187:          write(IOW,*) ' E-tally ', N, ' number of dim. ', INFO(4)
188:          stop 666
189:        end if
190: C
191:      else
192: C
193: C
194: C ... pass dimension size of E-D tally and D -> E table and its size
195: C for each dimension

```

```

196: C
197:      if ( INFO(4).eq.0 ) then
198:        call SUMUP0(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
199:          &          NGENE, WSUMB, DWRK )
200:      else if ( INFO(4).eq.1 ) then
201:        call SUMUP1(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
202:          &          NGENE, WSUMB, DWRK,
203:          &          NDIME(1), NDIMD(1), IETAL(IEDT(1)) )
204:      else if ( INFO(4).eq.2 ) then
205:        call SUMUP2(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
206:          &          NGENE, WSUMB, DWRK,
207:          &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
208:          &          NDIME(2), NDIMD(2), IETAL(IEDT(2)) )
209:      else if ( INFO(4).eq.3 ) then
210:        call SUMUP3(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
211:          &          NGENE, WSUMB, DWRK,
212:          &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
213:          &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
214:          &          NDIME(3), NDIMD(3), IETAL(IEDT(3)) )
215:      else if ( INFO(4).eq.4 ) then
216:        call SUMUP4(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
217:          &          NGENE, WSUMB, DWRK,
218:          &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
219:          &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
220:          &          NDIME(3), NDIMD(3), IETAL(IEDT(3)),
221:          &          NDIME(4), NDIMD(4), IETAL(IEDT(4)) )
222:      else if ( INFO(4).eq.5 ) then
223:        call SUMUP5(ETALY(INFO(5),1),ETALY(INFO(5),2),DTALY(LDT),
224:          &          NGENE, WSUMB, DWRK,
225:          &          NDIME(1), NDIMD(1), IETAL(IEDT(1)),
226:          &          NDIME(2), NDIMD(2), IETAL(IEDT(2)),
227:          &          NDIME(3), NDIMD(3), IETAL(IEDT(3)),
228:          &          NDIME(4), NDIMD(4), IETAL(IEDT(4)),
229:          &          NDIME(5), NDIMD(5), IETAL(IEDT(5)) )
230:      else
231:        write(IOW,*)
232:        &          'XXX(STLSUM) Cannot handle tally of more than 5 dimension.'
233:        write(IOW,*) ' E-tally ', N, ' number of dim. ', INFO(4)
234:        stop 666
235:      end if
236:    end if
237: C
238: C ... store batch result (cumulative sum to each batch)
239: C for "real variance" estimation ...
240: C
241:      if ( NETRV.gt.0 ) then
242:        do 150 K = 1, NETRV
243:          if ( N.eq.IETRV(1,K) ) then
244: C
245:          do 130 J = IETRV(2,K), IETRV(2,K+1) - 1
246:            ETRV(J,IBSUM) = ETALY(INFO(5)+J-IETRV(2,K),1)
247: 130          continue
248:          end if
249: 150          continue
250:        end if
251: 160 continue
252:
253:      return
254: C
255: C
256: 170 write(IOW,*) 'XXX(STLSUM) cannot input edited tally information ',
257:      &          ' Code = ', IRET
258:      stop 666
259: C
260:      end

```


src/shared/stprn1.f

```
1:      subroutine STPRN1( MODE, JSTPRN, IDTASK )
2: C=====
3: C purpose: manage I/O units of printout in subtask of multi-task
4: C      calculation using option flag JSTPRN(1).
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C i MODE : 'ON' - manage I/O units by JSTPRN(1)
8: C      'OFF' - restore I/O units previously managed by JSTPRN(1)
9: C i IDTASK : MVP task ID (from 1 to NTASK) of current process
10: C=====
11:      character*(*) MODE
12:      integer JSTPRN(2)
13: C
14:      include 'INC/_IOUNIT'
15: C
16: C ... local variables
17: C
18:      logical OPENED
19:      character*256 CWRK
20: C
21: C-----
22: C
23: C      ... main task is not under control of JSTPRN(1)
24: C
25: C      if ( IDTASK.eq.1 ) return
26: C
27: C      if ( MODE.eq.'ON' ) then
28: C
29: C      ... printout nothing ...
30: C
31: C      if ( JSTPRN(1).eq.0 ) then
32: C          IPR      = IONULL
33: C          MSG      = IONULL
34: C          IOG      = IONULL
35: C
36: C      ... printout warning/error message (MSG) ...
37: C
38: C      else if ( JSTPRN(1).eq.1 ) then
39: C          IPR      = IONULL
40: C          MSG      = IOSTDO
41: C          IOG      = IONULL
42: C
43: C      ... printout everything ...
44: C
45: C      else if ( JSTPRN(1).eq.2 ) then
46: C          IPR      = IOSTDO
47: C          MSG      = IOSTDO
48: C          IOG      = IOSTDO
49: C      end if
50: C
51: C      ... open null device if not opened
52: C
53: C      if ( IPR.eq.IONULL .or. MSG.eq.IONULL .or. IOG.eq.IONULL )
54: C      &      then
55: C          inquire(unit =IONULL,opened =OPENED,iostat =IOS)
56: C          if ( .not.OPENED ) then
57: C              IOS      = 0
58: C /#IF UNIX
59: C          open( unit =IONULL, file ='/dev/null', iostat =IOS )
60: C /#ELSEIF FILENAME(DOS)
61: C          open( unit =IONULL, file ='NULL', iostat =IOS )
62: C /#ELSE
63: C /#ENDIF
64: C          if ( IOS.ne.0 ) then
65: C              write(IOSTDO,7000) IDTASK, IONULL, IOS
```

```
66: 7000      format(/lx,'XXX(STPRN1) Task ',I5,': failed to open',
67:      &      ' null device on I/O unit ',I2,' (code=',I7,
68:      &      ')')
69:      call PRSTOP( 1, 'Failed to open null device.' )
70:      end if
71:      end if
72:      end if
73: C
74: C === restore I/O units
75: C
76: C      else
77: C          IPR      = IOSTDO
78: C          MSG      = IOSTDO
79: C          IOG      = IOSTDO
80: C      end if
81: C
82: C      return
83: C      end
```

src/shared/stprn2.f

```
1:      subroutine STPRN2( MODE, IDTASK, STFN )
2: C=====
3: C purpose: Print out output from each subtask in a multi-task
4: C      calculation.
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C i MODE : 'ON' - manage I/O units by JSTPRN(1)
8: C      'OFF' - restore I/O units previously managed by JSTPRN(1)
9: C i IDTASK : MVP task ID (from 1 to NTASK) of current process
10: C-----
11:      character*(*) MODE
12: C
13:      include 'INC/_IOUNIT'
14: C
15: C ... local variables
16: C
17:      logical OPENED, NMD
18:      character*512 FILE, CWRK
19:      character*4 I2C
20:      character*(*) STFN
21: C
22: C-----
23: C
24: C ... Search unopened I/O unit ...
25: C
26:      do 110 IOU = 8, 99
27:          inquire(IOU, opened =OPENED)
28:          if ( .not.OPENED ) then
29:              ISPR = IOU
30:              go to 9
31:          end if
32:      110 continue
33:      write(IMG,7010)
34: 7010 format(/1x,'!!!(STPRN2) Cannot find unopened I/O unit ')
35:      return
36:      9 continue
37: C
38: C ... determine file names for subtasks ...
39: C
40:      do 120 I = len(STFN),1,-1
41:          IE = I
42:          if(STFN(I:I).ne.' ') goto 99
43:      120 continue
44:      99 continue
45:      FILE = STFN(1:IE)//'.'//I2C(IDTASK-1)
46: C
47: C ... main task leaves here ...
48: C
49:      if ( IDTASK.eq.1 ) return
50: C
51:      if ( MODE.eq.'ON' ) then
52: C
53: C ... change I/O units ...
54: C
55:          IPR      = ISPR
56:          IMG      = ISPR
57:          IOG      = ISPR
58:          IOW      = ISPR
59: C
60: C ... open null device if not opened
61: C
62:          IOS      = 0
63:          open( unit =ISPR, file =FILE, iostat =IOS )
64:          if ( IOS.ne.0 ) then
65:              write(IOSTDO,7030) IDTASK, ISPR, IOS
```

```
66: 7030      format(/1x,'XXX(STPRN2) Task ',I5,' failed to open',
67:      &          ' I/O unit ',I2,' (code=',I7,')')
68: C
69: C === comments ===
70: C      Calling PRSTOP here may cause dead lock.
71: C      The code should be modified in the future.
72: C
73:          call PRSTOP( 1, 'Failed to open file.' )
74:          end if
75: C
76: C === restore I/O units
77: C
78:      else
79:          IPR      = IOSTDO
80:          IMG      = IOSTDO
81:          IOG      = IOSTDO
82:      end if
83: C
84:      return
85:      end
86: C
87:      character*4 function I2C(III)
88:      character*4 CCC
89:      write(CCC,'(i4.4)') III
90:      I2C = CCC
91:      return
92:      end
```

src/shared/ststfn.f

```
1:      subroutine STSTFN( FNAME )
2: C=====
3: C purpose: set a file name for subtask output.
4: C-----
5: C arguments (i=input,o=output,w=work)
6: C i FNAME : input file name for subtask output
7: C=====
8:      character*(*) FNAME
9:
10:     include 'INC/_TASKDT'
11:
12: C/ #IF PARA( EVM MPI )
13:     STFN = FNAME
14: C/ #ENDIF
15:
16:     return
17: end
```

src/shared/sumon.f

```
1:      subroutine SUMONI( IA,   IB,   N )
2: C=====
3: C purpose: elementwise summation of two integer arrays.
4: C       a(i) = a(i) + b(i)  (i = 1, n )
5: C=====
6:      integer IA(N), IB(N)
7: *VOCL LOOP,NOVREC
8:      do 100 I = 1, N
9:          IA(I)  = IA(I) + IB(I)
10:      100 continue
11:      return
12:      end
13: C
14:      subroutine SUMONR( RA,   RB,   N )
15: C=====
16: C purpose: elementwise summation of two real arrays.
17: C       a(i) = a(i) + b(i)  (i = 1, n )
18: C=====
19:      real RA(N), RB(N)
20: *VOCL LOOP,NOVREC
21:      do 100 I = 1, N
22:          RA(I)  = RA(I) + RB(I)
23:      100 continue
24:      return
25:      end
26: C
27:      subroutine SUMOND( DA,   DB,   N )
28: C=====
29: C purpose: elementwise summation of two real*8 arrays.
30: C       a(i) = a(i) + b(i)  (i = 1, n )
31: C history: programmed by M.Sasaki ( Oct 1993 )
32: C=====
33:      real*8 DA(N), DB(N)
34: *VOCL LOOP,NOVREC
35:      do 100 I = 1, N
36:          DA(I)  = DA(I) + DB(I)
37:      100 continue
38:      return
39:      end
40: C
41:      subroutine SUMONI8( I8A, I8B, N )
42: C=====
43: C purpose: elementwise summation of two integer arrays.
44: C       a(i) = a(i) + b(i)  (i = 1, n )
45: C=====
46: C/#IF INTEGER8
47: *      integer*8 I8A(N), I8B(N)
48: C/#ELSE
49:      integer I8A(N), I8B(N)
50: C/#ENDIF
51: *VOCL LOOP,NOVREC
52:      do I = 1, N
53:          I8A(I)  = I8A(I) + I8B(I)
54:      end do
55:      return
56:      end
```

src/shared/sumupx.f

```

1:      subroutine SUMUP0( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK )
2: C=====
3: C what to do: edit special tally having 0 dimensions.
4: C called in : stlsum
5: C=====
6:      implicit real*8(A-H,O-Z)
7: C
8:      real*8 DWRK(1), WSUMB
9: C
10:     real*8 ETALY(1)
11:     real*8 SETALY(1)
12:     real*8 DTALY(1)
13: C
14: C-----
15: C
16: C
17: C     DNG      = 1.0D0/DBLE(NGENE)
18: C     DNG      = 1.0D0/WSUMB
19: C     ETALY(1)  = ETALY(1) + DTALY(1)
20: C     SETALY(1) = SETALY(1) + DTALY(1)**2*DNG
21: C
22:     return
23:     end
24: C
25: C-----
26: C
27:      subroutine SUMUP1( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK,
28: &      NE1, ND1, IEDT1 )
29: C=====
30: C what to do: edit special tally having 1 dimensions.
31: C called in : stlsum
32: C=====
33:      implicit real*8(A-H,O-Z)
34: C
35:      real*8 DWRK(NE1), WSUMB
36: C
37:      real*8 ETALY(NE1)
38:      real*8 SETALY(NE1)
39:      real*8 DTALY(ND1)
40: C
41:      integer IEDT1(*)
42: C-----
43:      do 100 N1 = 1, NE1
44:         DWRK(N1) = 0.0D0
45: 100 continue
46: C
47: C
48:      L1      = 1
49:      do 120 N1 = 1, NE1
50:         do 110 I1 = 1, IEDT1(L1)
51:            M1      = IEDT1(L1+I1)
52: C
53:            DWRK(N1) = DWRK(N1) + DTALY(M1)
54: C
55: 110 continue
56:         L1      = L1 + IEDT1(L1) + 1
57: 120 continue
58: C
59: C
60: C     DNG      = 1.0D0/DBLE(NGENE)
61: C     DNG      = 1.0D0/WSUMB
62: C     do 130 N1 = 1, NE1
63: C         ETALY(N1) = ETALY(N1) + DWRK(N1)
64: C         SETALY(N1) = SETALY(N1) + DWRK(N1)**2*DNG
65: C     130 continue

```

```

66: C
67:     return
68:     end
69: C
70: C-----
71: C
72:      subroutine SUMUP2( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK,
73: &      NE1, ND1, IEDT1,
74: &      NE2, ND2, IEDT2 )
75: C=====
76: C what to do: edit special tally having 2 dimensions.
77: C called in : stlsum
78: C history: programmed by M.Sasaki (Sep 1995)
79: C=====
80:      implicit real*8(A-H,O-Z)
81: C
82:      real*8 DWRK(NE2,NE1), WSUMB
83: C
84:      real*8 ETALY(NE2,NE1)
85:      real*8 SETALY(NE2,NE1)
86:      real*8 DTALY(ND2,ND1)
87: C
88:      integer IEDT1(*)
89:      integer IEDT2(*)
90: C-----
91:      do 110 N1 = 1, NE1
92:         do 100 N2 = 1, NE2
93:            DWRK(N2,N1) = 0.0D0
94: 100 continue
95: 110 continue
96: C
97: C
98: C
99:      L1      = 1
100:      do 150 N1 = 1, NE1
101:         do 140 I1 = 1, IEDT1(L1)
102:            M1      = IEDT1(L1+I1)
103: C
104:            L2      = 1
105:            do 130 N2 = 1, NE2
106:               do 120 I2 = 1, IEDT2(L2)
107:                  M2      = IEDT2(L2+I2)
108: C-----
109:                  DWRK(N2,N1) = DWRK(N2,N1) + DTALY(M2,M1)
110: C-----
111:            120 continue
112:            L2      = L2 + IEDT2(L2) + 1
113: 130 continue
114: C
115: 140 continue
116:         L1      = L1 + IEDT1(L1) + 1
117: 150 continue
118: C
119: C
120: C     DNG      = 1.0D0/DBLE(NGENE)
121: C     DNG      = 1.0D0/WSUMB
122: C     do 170 N1 = 1, NE1
123: C         do 160 N2 = 1, NE2
124: C             ETALY(N2,N1) = ETALY(N2,N1) + DWRK(N2,N1)
125: C             SETALY(N2,N1) = SETALY(N2,N1) + DWRK(N2,N1)**2*DNG
126: C         160 continue
127: C     170 continue
128: C
129:     return
130:     end

```

src/shared/sumupx.f

```

131: C
132: C-----
133: C
134:     subroutine SUMUP3( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK,
135: &          NE1,   ND1,   IEDT1,
136: &          NE2,   ND2,   IEDT2,
137: &          NE3,   ND3,   IEDT3
138: &          )
139: C=====
140: C what to do: edit special tally having 3 dimensions.
141: C called in : stlsum
142: C history: programmed by M.Sasaki (Sep 1995)
143: C-----
144:     implicit real*8(A-H,O-Z)
145: C
146:     real*8 DWRK(NE3,NE2,NE1), WSUMB
147: C
148:     real*8 ETALY(NE3,NE2,NE1)
149:     real*8 SETALY(NE3,NE2,NE1)
150:     real*8 DTALY(ND3,ND2,ND1)
151: C
152:     integer IEDT1(*)
153:     integer IEDT2(*)
154:     integer IEDT3(*)
155: C-----
156:     do 120 N1 = 1, NE1
157:       do 110 N2 = 1, NE2
158:         do 100 N3 = 1, NE3
159:           DWRK(N3,N2,N1) = 0.0D0
160:         continue
161:       continue
162:     continue
163: C
164: C
165: C
166:     L1 = 1
167:     do 180 N1 = 1, NE1
168:       do 170 I1 = 1, IEDT1(L1)
169:         M1 = IEDT1(L1+I1)
170: C
171:       L2 = 1
172:       do 160 N2 = 1, NE2
173:         do 150 I2 = 1, IEDT2(L2)
174:           M2 = IEDT2(L2+I2)
175: C
176:         L3 = 1
177:         do 140 N3 = 1, NE3
178:           do 130 I3 = 1, IEDT3(L3)
179:             M3 = IEDT3(L3+I3)
180: C-----
181:             DWRK(N3,N2,N1) = DWRK(N3,N2,N1)
182:             &          + DTALY(M3,M2,M1)
183: C-----
184:           continue
185:           L3 = L3 + IEDT3(L3) + 1
186:         continue
187: C
188:         continue
189:         L2 = L2 + IEDT2(L2) + 1
190:       continue
191: C
192:       continue
193:       L1 = L1 + IEDT1(L1) + 1
194:     continue
195: C

```

```

196: C
197: C     DNG = 1.0D0/DBLE(NGENE)
198: C     DNG = 1.0D0/WSUMB
199: C     do 210 N1 = 1, NE1
200: C       do 200 N2 = 1, NE2
201: C         do 190 N3 = 1, NE3
202: C           ETALY(N3,N2,N1) = ETALY(N3,N2,N1) + DWRK(N3,N2,N1)
203: C           SETALY(N3,N2,N1) = SETALY(N3,N2,N1) + DWRK(N3,N2,N1)**
204: C             &          2*DNG
205: C         190 continue
206: C       200 continue
207: C     210 continue
208: C
209: C     return
210: C     end
211: C
212: C-----
213: C
214:     subroutine SUMUP4( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK,
215: &          NE1,   ND1,   IEDT1,
216: &          NE2,   ND2,   IEDT2,
217: &          NE3,   ND3,   IEDT3,
218: &          NE4,   ND4,   IEDT4 )
219: C=====
220: C what to do: edit special tally having 4 dimensions.
221: C called in : stlsum
222: C history: programmed by M.Sasaki (Sep 1995)
223: C-----
224:     implicit real*8(A-H,O-Z)
225: C
226:     real*8 DWRK(NE4,NE3,NE2,NE1), WSUMB
227: C
228:     real*8 ETALY(NE4,NE3,NE2,NE1)
229:     real*8 SETALY(NE4,NE3,NE2,NE1)
230:     real*8 DTALY(ND4,ND3,ND2,ND1)
231: C
232:     integer IEDT1(*)
233:     integer IEDT2(*)
234:     integer IEDT3(*)
235:     integer IEDT4(*)
236: C-----
237:     do 130 N1 = 1, NE1
238:       do 120 N2 = 1, NE2
239:         do 110 N3 = 1, NE3
240:           do 100 N4 = 1, NE4
241:             DWRK(N4,N3,N2,N1) = 0.0D0
242:           continue
243:         continue
244:       continue
245:     continue
246: C
247: C
248: C
249:     L1 = 1
250:     do 210 N1 = 1, NE1
251:       do 200 I1 = 1, IEDT1(L1)
252:         M1 = IEDT1(L1+I1)
253: C
254:       L2 = 1
255:       do 190 N2 = 1, NE2
256:         do 180 I2 = 1, IEDT2(L2)
257:           M2 = IEDT2(L2+I2)
258: C
259:         L3 = 1
260:         do 170 N3 = 1, NE3

```

src/shared/sumupx.f

```

261:      do 160 I3 = 1, IEDT3(L3)
262:         M3      = IEDT3(L3+I3)
263: C
264:         L4      = 1
265:         do 150 N4 = 1, NE4
266:            do 140 I4 = 1, IEDT4(L4)
267:               M4      = IEDT4(L4+I4)
268: C-----
269:            DWRK(N4,N3,N2,N1) = DWRK(N4,N3,N2,N1)
270:            &
271:            + DTALY(M4,M3,M2,M1)
272: C-----
272:      140      continue
273:      140      L4      = L4 + IEDT4(L4) + 1
274:      150      continue
275: C
276:      160      continue
277:      160      L3      = L3 + IEDT3(L3) + 1
278:      170      continue
279: C
280:      180      continue
281:      180      L2      = L2 + IEDT2(L2) + 1
282:      190      continue
283: C
284:      200      continue
285:      200      L1      = L1 + IEDT1(L1) + 1
286:      210      continue
287: C
288: C
289: C      DNG      = 1.0D0/DBLE(NGENE)
290: C      DNG      = 1.0D0/WSUMB
291:      do 250 N1 = 1, NE1
292:         do 240 N2 = 1, NE2
293:            do 230 N3 = 1, NE3
294:               do 220 N4 = 1, NE4
295:                  ETALY(N4,N3,N2,N1) = ETALY(N4,N3,N2,N1)
296:                  + DWRK(N4,N3,N2,N1)
297:                  &
298:                  SETALY(N4,N3,N2,N1) = SETALY(N4,N3,N2,N1)
299:                  + DWRK(N4,N3,N2,N1)**2*DNG
300:                  &
301:                  continue
302:                  230      continue
303:                  240      continue
304:                  250      continue
305:                  return
306:                  end
307: C-----
308: C
309:      subroutine SUMUP5( ETALY, SETALY,DTALY, NGENE, WSUMB, DWRK,
310:      &      NE1, ND1, IEDT1,
311:      &      NE2, ND2, IEDT2,
312:      &      NE3, ND3, IEDT3,
313:      &      NE4, ND4, IEDT4,
314:      &      NE5, ND5, IEDT5
315:      &      )
316: C=====
317: C what to do: edit special tally having 5 dimensions.
318: C called in : stlsum
319: C history: programmed by M.Sasaki (Sep 1995)
320: C=====
321:      implicit real*8(A-H,O-Z)
322: C
323:      real*8 DWRK(NE5,NE4,NE3,NE2,NE1), WSUMB
324: C
325:      real*8 ETALY(NE5,NE4,NE3,NE2,NE1)

```

```

326:      real*8 SETALY(NE5,NE4,NE3,NE2,NE1)
327:      real*8 DTALY(ND5,ND4,ND3,ND2,ND1)
328: C
329:      integer IEDT1(*)
330:      integer IEDT2(*)
331:      integer IEDT3(*)
332:      integer IEDT4(*)
333:      integer IEDT5(*)
334: C-----
335:      do 140 N1 = 1, NE1
336:         do 130 N2 = 1, NE2
337:            do 120 N3 = 1, NE3
338:               do 110 N4 = 1, NE4
339:                  do 100 N5 = 1, NE5
340:                     DWRK(N5,N4,N3,N2,N1) = 0.0D0
341:                  100      continue
342:                  110      continue
343:                  120      continue
344:                  130      continue
345:                  140      continue
346: C
347: C
348: C
349:      L1      = 1
350:      do 240 N1 = 1, NE1
351:         do 230 I1 = 1, IEDT1(L1)
352:            M1      = IEDT1(L1+I1)
353: C
354:      L2      = 1
355:      do 220 N2 = 1, NE2
356:         do 210 I2 = 1, IEDT2(L2)
357:            M2      = IEDT2(L2+I2)
358: C
359:      L3      = 1
360:      do 200 N3 = 1, NE3
361:         do 190 I3 = 1, IEDT3(L3)
362:            M3      = IEDT3(L3+I3)
363: C
364:      L4      = 1
365:      do 180 N4 = 1, NE4
366:         do 170 I4 = 1, IEDT4(L4)
367:            M4      = IEDT4(L4+I4)
368: C
369:      L5      = 1
370:      do 160 N5 = 1, NE5
371:         do 150 I5 = 1, IEDT5(L5)
372:            M5      = IEDT5(L5+I5)
373: C-----
374:      DWRK(N5,N4,N3,N2,N1) =
375:      &
376:      &
377:      + DTALY(M5,M4,M3,M2,M1)
378: C-----
379:      150      continue
380:      150      L5      = L5 + IEDT5(L5) + 1
381:      160      continue
382: C
383:      170      continue
384:      170      L4      = L4 + IEDT4(L4) + 1
385:      180      continue
386: C
387:      190      continue
388:      190      L3      = L3 + IEDT3(L3) + 1
389:      200      continue
390:      210      continue

```

src/shared/sumupx.f

```
391:          L2      = L2 + IEDT2(L2) + 1
392:      220      continue
393: C
394:      230      continue
395:          L1      = L1 + IEDT1(L1) + 1
396:      240      continue
397: C
398: C
399: C      DNG      = 1.0D0/DBLE(NGENE)
400:      DNG      = 1.0D0/WSUMB
401:      do 290 N1 = 1, NE1
402:          do 280 N2 = 1, NE2
403:              do 270 N3 = 1, NE3
404:                  do 260 N4 = 1, NE4
405:                      do 250 N5 = 1, NE5
406:                          ETALY(N5,N4,N3,N2,N1) = ETALY(N5,N4,N3,N2,N1)
407:                          & + DWRK(N5,N4,N3,N2,N1)
408:                          SETALY(N5,N4,N3,N2,N1) = SETALY(N5,N4,N3,N2,N1)
409:                          & + DWRK(N5,N4,N3,N2,N1)**2*DNG
410:                      250      continue
411:                  260      continue
412:              270      continue
413:          280      continue
414:      290      continue
415: C
416:      return
417:      end
```


src/shared/syssrc.f

```

1:      subroutine SYSSRC( PROG, A, N,      NS,      NPK,
2:      &                  JDEBG, JLATT, JTLLT, JHLAT,
3:      &                  SRCSP, RSRC,      DSRC,
4:      &                  MXREJ, H,      IH,      DH,      IRAND, SDA, NSDA,
5:      &                  LX,      LY,      LZ,      LA,      LB,      LC,
6:      &                  LE,      LG,      LT,
7:      &                  LW,      JVSMP, MWVEC, MVSTK, LPWRK, LVSTK,
8:      &                  RWK,      IWK,      NRWK, NERROR )
9:
10: C/#!/IF SOURCE(NEW)
11:
12: C==<MVP/GMVP>=====
13: C purpose: generation of source particle from a source set
14: C called in: source generation routine.
15: C calls: MVITOC NOBLNK NUCEIS OLDSRC PCTCHK PCTLB RANU2 UNPKCS
16: C          UNPKND UNPKPT USRSRC VDENTK VDRSET VDVARS VDWOR
17: C-----
18: C
19: C arguments (i =input, o=output, w=work, c=constant )
20: C
21: CLASS arg:  meanings
22: C
23: C i  PROG : program name ('MVP'/'GMVP')
24: C i  N : source set number (input)
25: C i  NS : number of source to be generated from this source set (input)
26: C
27: C o  NPK : type of generated particles
28: C
29: C i  SRCSP : data packet container including source specification data.
30: C          (input)
31: C i  RSRC, DSRC : real & real*8 equivalent to 'srcsp'
32: C
33: C c  MXREJ : limit of iteration count in rejection loop.
34: C
35: C io H      : variable array memory area ( H in COMMON /ARRAY/ )
36: C io IH     : integer equivalent to 'H'
37: C io DH     : real*8 equivalent to 'H'
38: C
39: C io IRAND : seed of random number
40: C
41: C i  LX,LY,LZ,LA,LB,LC,LE,LG,LT,LW :
42: C          pointers of variables to be sampled
43: C          given as indices in real type array 'H' but samled
44: C          values are given as real*8 data on 'DMEM' array.
45: C
46: C o  JVSMP(*) : flag of sampled/not-sampled for the variables above.
47: C
48: C c  MWVEC: max number of working variables in sampling
49: C c  MVSTK: max of vector calculation stack depth
50: C i  LPWK(MWVEC) : pointers for explicit working variables in sampling.
51: C i  LVSTK(MVSTK) : pointers for implicit working variables (stack) in
52: C          'VDENTK' routine.
53: C
54: C w  RWK(NS,NRWK): working array mainly used to save random numbers.
55: C w  IWK(NS,NRWK): working array (may overlap with RWK)
56: C c  NRWK :
57: C o  NERROR : number of error
58: C
59: C =====
60:      implicit real*8(D)
61: C
62:      character*(*) PROG
63:      real A(*)
64: C
65:      integer JDEBG(*)

```

```

66: C
67:      integer SRCSP(*)
68:      real RSRC(*)
69:      real*8 DSRC(*)
70: C
71:      real*8 SDA(NSDA)
72: C
73: C ... H,IH & dH start at the same address ...
74: C
75:      real H(*)
76:      integer IH(*)
77:      real*8 DH(*)
78: C
79:      parameter( NVECV      = 10 )
80: C
81: C          x,y,z : source position (unit: cm)
82: C          a,b,c : direction cosines (a**2 + b**2 + c**2 = 1.0)
83: C          e      : energy (unit: eV)
84: C          g      : energy group
85: C          t      : time (unit: second)
86: C          w      : particle weight (ordinary = 1.0 without biasing)
87: C
88:      integer LX, LY, LZ, LA, LB, LC, LE, LG, LT, LW
89:      integer JVSMP(NVECV)
90: C
91:      integer LPWRK(MWVEC), LVSTK(MVSTK)
92: C
93: C ... rwk & iwk should start from the same address.
94: C
95:      real RWK(NS*NRWK)
96:      integer IWK(NS*NRWK)
97: C
98: C          nrwk >= 7 for JNUCS > 0
99: C
100: C === local data ===
101: C
102: C          character*512 CWRK
103: C          character*256 CWRK2
104: C          parameter( MAXVAR      = 32 )
105: C          character*2048 CWRK
106: C          character*1024 CWRK2
107: C          parameter( MAXVAR      = 128 )
108: C          parameter( MXACC       = 4 )
109: C          character*16 VARNM(MAXVAR)
110: C          character*4 TYPE
111: C          integer LPVAR(MAXVAR)
112: C          integer LPSV(MAXVAR)
113: C          integer IVMASK(MXACC)
114: C          integer INDX(2,0:MXACC)
115: C          integer NREJ(MXACC)
116: C          integer ITEMP(32)
117: C          real RTEMP(32)
118: C
119: C          real*8 PAI, DPAI, HPAI
120: C          parameter( PAI      = 3.1415926535897932D0, DPAI = 2D0*PAI, HPAI =
121: C          &          0.5D0*PAI )
122: C
123: C          include 'INC/_IOUNIT'
124: C          include 'INC/_WORDL'
125: C-----
126: C ... statement function to translate index to real array to real*8
127: C          array
128: C
129: C/#!/IF WORD(64)
130: *      LDBLE( L ) = L

```

src/shared/syssrc.f

```

131: C/#ELSE
132:     LDBLE(L)      = L/2 + 1
133: C/#ENDIF
134: C
135: C-----
136: C
137:     if ( MOD(JDEBG(3),2).ne.0 ) then
138:         write(IPR,*) '(SYSSRC) Source ', N, ' sample ', NS
139:     end if
140: C
141:     NERROR = 0
142: C
143: C ... get general source specification ...
144: C
145:     write(CWRK, '('&'',I5,'SPEC'')' ) N
146:     call PCTLB( SRCSP, 'SPEC0', CWRK(:10), IRET )
147:     if ( IRET.ne.0 ) then
148:         IPOS = 1
149:         go to 380
150:     end if
151: C
152:     call UNPKND( SRCSP, 'SPEC1-2', 'I4', ITEMP, 2, ND, IRET )
153: C
154:     NPK = ITEMP(2)
155: C
156: CCC call UNPKND( SRCSP, 'RATIO', 'R4', RTEMP, 1, ND, IRET )
157: call UNPKND( SRCSP, 'RATIO', 'R8', RTEMP, 1, ND, IRET )
158: call UNPKND( SRCSP, 'ID', 'I4', ISCID, 1, ND, IRET )
159: C
160: C
161:     NVAR = 0
162: C
163: C ... get sampling vector list ....
164: C
165:     call UNPKCS( SRCSP, 'VECTOR', CWRK, LL, IRET )
166:     if ( IRET.ne.0 ) then
167:         IPOS = 2
168:         go to 380
169:     end if
170: C
171:     do 100 I = 1, NVECV
172:         JVSMP(I) = 0
173: 100 continue
174: C
175:     call VDRSET( IERR )
176: C
177:     K1 = 1
178: 110 continue
179:     call NOBLNK( CWRK, K1, K2, LL )
180:     if ( K1.le.LL ) then
181:         if ( CWRK(K1:K2).eq.'X' ) then
182:             JVSMP(1) = 1
183:             LPPP = LX
184:         else if ( CWRK(K1:K2).eq.'Y' ) then
185:             JVSMP(2) = 2
186:             LPPP = LY
187:         else if ( CWRK(K1:K2).eq.'Z' ) then
188:             JVSMP(3) = 3
189:             LPPP = LZ
190:         else if ( CWRK(K1:K2).eq.'A' ) then
191:             JVSMP(4) = 4
192:             LPPP = LA
193:         else if ( CWRK(K1:K2).eq.'B' ) then
194:             JVSMP(5) = 5
195:             LPPP = LB
196:         else if ( CWRK(K1:K2).eq.'C' ) then
197:             JVSMP(6) = 6
198:             LPPP = LC
199:         else if ( CWRK(K1:K2).eq.'E' ) then
200:             JVSMP(7) = 7
201:             LPPP = LE
202:         else if ( CWRK(K1:K2).eq.'G' ) then
203:             JVSMP(8) = 8
204:             LPPP = LG
205:         else if ( CWRK(K1:K2).eq.'T' ) then
206:             JVSMP(9) = 9
207:             LPPP = LT
208:         else if ( CWRK(K1:K2).eq.'W' ) then
209:             JVSMP(10) = 10
210:             LPPP = LW
211:         else
212:             IPOS = 3
213:             go to 380
214:         end if
215:
216:         CWRK2 = '@'//CWRK(K1:K2)
217:         call VDVAR( CWRK2, LPPP, IERR )
218:         NVAR = NVAR + 1
219:         VARNM(NVAR) = CWRK(K1:K2)
220:         LPVAR(NVAR) = LDBLE(LPPP)
221:
222:         K1 = K2 + 1
223:         go to 110
224:     end if
225: C
226: C ... get working vector list ....
227: C
228:     call UNPKCS( SRCSP, 'WORK', CWRK, LL, IRET )
229:     if ( IRET.ne.0 ) then
230:         IPOS = 4
231:         go to 380
232:     end if
233: C
234:     if ( LL.gt.0 ) then
235:         IV = 0
236:         K1 = 1
237: 120 continue
238:         call NOBLNK( CWRK, K1, K2, LL )
239:         if ( K1.le.LL ) then
240:             IV = IV + 1
241:             CWRK2 = '@'//CWRK(K1:K2)
242:             call VDVAR( CWRK2, LPWRK(IV), IERR )
243:             NVAR = NVAR + 1
244:             VARNM(NVAR) = CWRK(K1:K2)
245:             LPVAR(NVAR) = LDBLE(LPWRK(IV))
246:             K1 = K2 + 1
247:             go to 120
248:         end if
249:     end if
250: C
251: C .... send vector stack address for vector-calculator
252: C
253:     do 130 I = 1, MVSTK
254:         call VDWORK( LVSTK(I), IERR )
255: 130 continue
256: C
257: C .... locate position of sampling information ....
258: C
259:     write(CWRK(:13), '('&'SAMPLING'&',I5)') N
260:     call PCTLB( SRCSP, 'SAMPLING', CWRK(:13), IRET )

```

src/shared/syssrc.f

```

261:      if ( IRET.ne.0 ) then
262:        IPOS = 5
263:        go to 380
264:      end if
265: C
266: C ... set index range for vectors (may be changed in rejection sampling)
267: C
268:      I1 = 1
269:      I2 = NS
270: C
271:      IACC = 0
272:      INDX(1,IACC) = I1
273:      INDX(2,IACC) = I2
274: C
275: C ... ivmask(iacc) : vector # of mask vector for 'ACCEPT' block ...
276: C
277: C
278: 140 call PCTCHK( SRCSP, ' ', TYPE, LPP, ND, IRET )
279:      if ( IRET.ne.0 ) then
280:        IPOS = 6
281:        go to 380
282:      end if
283: C
284: C .... label ....
285: C
286:      if ( TYPE.eq.'LB' ) then
287:        call UNPKPT( SRCSP, 'LABEL', TYPE, LPT, NL, IRET )
288: C
289:        call MVITOC( NL, CWRK, SRCSP(LPT) )
290: C
291: C ==== accept block starts ====
292: C
293:      if ( CWRK(:6).eq.'ACCEPT' ) then
294:        IACC = IACC + 1
295:        INDX(1,IACC) = I1
296:        INDX(2,IACC) = I2
297:        NREJ(IACC) = 0
298: C
299:      else if ( CWRK(:10).eq.'END-ACCEPT' ) then
300: C
301: C ==== end of accept block ====
302: C
303: C .... check number of rejected sampling ....
304: C
305:        NREJCT = 0
306: *VOCL LOOP,NOVREC
307:        do 150 I = I1, I2
308:          if ( DH(IVMASK(IACC)+I).eq.0.0 ) NREJCT = NREJCT + 1
309: 150        continue
310: C
311:      if ( MOD(JDEBG(3),2).ne.0 ) then
312:        write(IPR,*) '(SYSSRC) nrejt ', NREJCT, ' iacc ', IACC,
313:      &      ' i1 i2 ', I1, I2, ' mxrej ', MXREJ, ' nrej ',
314:      &      NREJ(IACC)
315:      end if
316: C
317:      if ( NREJCT.gt.0 ) then
318: C
319: C
320:        if ( NREJ(IACC).gt.MXREJ ) then
321:          write(IPR,*) 'XXX TOO MANY REJECTION : SOURCE ', N,
322:      &      ' INDEX ', I1, I2, ' SAMPLING ', NS
323:          NERROR = NERROR + 1
324:          IACC = IACC - 1
325:          I1 = INDX(1,IACC)

```

```

326:          I2 = INDX(2,IACC)
327:        else
328:          I1 = I2 - NREJCT + 1
329:          INDX(1,IACC) = I1
330:          INDX(2,IACC) = I2
331:          NREJ(IACC) = NREJ(IACC) + 1
332: C
333: C ... return to the head of 'ACCEPT' block ...
334: C
335:          CWRK2 = 'ACCEPT'//CWRK(LEN('END-ACCEPT')+1:NL)
336:          NLL2 = LEN('ACCEPT') + (NL-LEN('END-ACCEPT'))
337:          call PCTLB( SRCSP, 'ACCEPT', CWRK2(:NLL2), IRET )
338:        end if
339:      else
340: C
341: C ... leave 'ACCEPT' block ...
342: C
343:          IACC = IACC - 1
344:          I1 = INDX(1,IACC)
345:          I2 = INDX(2,IACC)
346:        end if
347: C
348:      end if
349:      go to 140
350: C
351: C .... formula or function or accept ....
352: C
353:      else if ( TYPE.eq.'CH' ) then
354: C
355:        CWRK = ' '
356:        call UNPKCS( SRCSP, 'CH', CWRK, NL, IRET )
357: C
358:      if ( MOD(JDEBG(3),2).ne.0 ) then
359:        write(IPR,*) '(SYSSRC) sample <', CWRK(:NL), '>'
360:      end if
361: C
362:      if ( CWRK(:NL).eq.'NOP' ) go to 140
363: C
364: C
365: C ===== formula =====
366: C
367: C
368:      if ( CWRK(1:1).eq.'@' ) then
369: C
370:        call VDENTK( CWRK(:NL), DH, IERR, I1, I2 )
371:        if ( IERR.ne.0 ) NERROR = NERROR + 1
372:        go to 140
373: C
374:      end if
375: C
376: C ===== function =====
377: C
378: C
379:      if ( CWRK(1:8).eq.'FUNCTION' ) then
380:        CWRK = ' '
381:        call UNPKCS( SRCSP, 'VECTOR', CWRK, NL, IRET )
382: C
383: C .... get variable #'s in lefthand side ...
384: C
385:        K = INDEX(CWRK(:NL), '@') + 1
386: C
387:      if ( CWRK(K:K).ne.'(' ) then
388:        K2 = INDEX(CWRK(:NL), '=') - 1
389:      else
390:        K = K + 1

```

src/shared/syssrc.f

```

391:          K2      = INDEX(CWRK(:NL),'') - 1
392:          end if
393: C
394: C   ... save pointer (to 'DH') of sampled variable in 'lpsv' array...
395: C
396:          NSVECT  = 0
397:          IS      = K
398: C
399: 160      call NOBLNK( CWRK, IS, IE, K2 )
400: C
401:          if ( IS.le.K2 ) then
402:              NSVECT = NSVECT + 1
403:              do 170 I = 1, Nvars
404:                  if ( VARNM(I).eq.CWRK(IS:IE) ) then
405:                      LPSV(NSVECT) = LPVAR(I) - 1
406:                  end if
407: 170      continue
408:              IS      = IE + 1
409:              go to 160
410:          end if
411: C
412: C   .... get function name ....
413: C
414:          call UNPKCS( SRCSP, 'FNAME', CWRK, NL, IRET )
415: C
416:          if ( MOD(JDEBG(2),2).ne.0 ) then
417:              write(IPR,*) '(SYSSRC) function <', CWRK(:NL), '>'
418:          end if
419: C
420: C
421: C >>>> 1 parameter sampling <<<<
422: C
423: C
424:          if ( 'UNIFORM'.eq.CWRK(:NL) .or. 'COSINE'.eq.CWRK(:NL)
425:              & .or. 'WATT'.eq.CWRK(:NL) .or. 'GAUSS'.eq.CWRK(:NL)
426:              & .or. 'MAXWELL'.eq.CWRK(:NL)
427:              & .or. 'EVAPORATION'.eq.CWRK(:NL) .or. 'POWER'.eq.CWRK(:NL)
428:              & .or. 'NORMALIZE'.eq.CWRK(:NL) ) then
429: C
430:              call UNPKPT( SRCSP, CWRK(:NL), 'R8', LJ, ND, IRET )
431: C
432:              call SMPLEX1( PROG, I2-I1+1, DH(LPSV(1)+I1), CWRK(:NL),
433:              &          DSRC(LJ), ND, IRAND, RWK, NRWK, IERRR, NERROR )
434: C
435:          else if ( 'FEYNMAN'.eq.CWRK(:NL) ) then
436: C
437:              call UNPKPT( SRCSP, CWRK(:NL), 'R8', LJ, ND, IRET )
438: C
439:              call SMPFYN( PROG, I2-I1+1, DH(LPSV(1)+I1), CWRK(:NL),
440:              &          DSRC(LJ), ND, IRAND, RWK, NRWK, IERRR,
441:              &          NERROR )
442: C
443: C
444: C >>>> 2 parameter sampling <<<<
445: C
446: C
447:          else if ( 'DISC'.eq.CWRK(:NL) .or. 'CIRCLE'.eq.CWRK(:NL)
448:              & .or. 'SWAP'.eq.CWRK(:NL) .or. 'ROTATE2D'.eq.
449:              &          CWRK(:NL) ) then
450: C
451: C
452:              if ( 'SWAP'.ne.CWRK(:NL) ) then
453:                  call UNPKPT( SRCSP, CWRK(:NL), 'R8', LJ, ND, IRET )
454:              end if
455: C

```

```

456:          call SMPLEX2( PROG, I2-I1+1, DH(LPSV(1)+I1),
457:              &          DH(LPSV(2)+I1), CWRK(:NL), DSRC(LJ), ND, IRAND,
458:              &          RWK, NRWK, IERRR, NERROR )
459: C
460: C
461: C >>>> 3 parameter sampling <<<<
462: C
463: C
464:          else if ( 'SPHERE'.eq.CWRK(:NL) .or. 'ISOTROPIC'.eq.
465:              &          CWRK(:NL) .or. 'RESIZE'.eq.CWRK(:NL)
466:              &          .or. 'ROTATE3D'.eq.CWRK(:NL)
467:              &          .or. 'NEWAXIS'.eq.CWRK(:NL) ) then
468: C
469: C
470:              call UNPKPT( SRCSP, CWRK(:NL), 'R8', LJ, ND, IRET )
471: C
472:              call SMPLEX3( PROG, I2-I1+1, DH(LPSV(1)+I1),
473:              &          DH(LPSV(2)+I1), DH(LPSV(3)+I1), CWRK(:NL),
474:              &          DSRC(LJ), ND, IRAND, RWK, NRWK, IERRR, NERROR )
475: C
476: C
477: C >>>> TABLE
478: C
479: C
480:          else if ( 'TABLE'.eq.CWRK(:NL) ) then
481:              call UNPKCS( SRCSP, 'INTERPOLATION', CWRK, LL, IRET1 )
482: C
483:              INTX      = 0
484:              if ( CWRK(:LL).eq.'STEP' ) then
485:                  INTX      = 1
486:              else if ( CWRK(:LL).eq.'DISCRETE' ) then
487:                  INTX      = 2
488:              else if ( CWRK(:LL).eq.'LINEAR' ) then
489:                  INTX      = 3
490:              else if ( CWRK(:LL).eq.'LOG-LINEAR' ) then
491:                  INTX      = 4
492:              else if ( CWRK(:LL).eq.'LOG-STEP' ) then
493:                  INTX      = 5
494:              end if
495: C
496:              call UNPKPT( SRCSP, 'X POINTS', 'R4', KX, NX, IRET2 )
497:              IRET3      = 0
498:              if ( INDEX(CWRK(:LL),'LINEAR').ne.0 ) then
499:                  call UNPKPT( SRCSP, 'PRB.AS GIVEN', 'R4', KPX, NPX,
500:                  &          IRET3 )
501:              end if
502:              call UNPKPT( SRCSP, 'PROBABILITY', 'R4', KP, NPXP, IRET4
503:              &          )
504:              call UNPKPT( SRCSP, 'ALIAS', 'I4', KPA, NPA, IRET5 )
505: C
506:              NP          = I2 - I1 + 1
507:              call RANU2( IRAND, RWK(I1), 3*NP, ICOD )
508: C
509: C
510: C   ... sampling of interval/point # by alias method ....
511: C
512: C   ... sample ...
513: C
514: C
515: C ***** if( cwrk(:ll).eq.'STEP' ) then
516: C           if ( INTX.eq.1 ) then
517: C               do 180 I = I1, I2
518: C /#IF ROUND OFF(NEAREST)
519: C               KK          = MIN(NPXP-1,INT(RWK(I)*NPXP))
520: C /#ELSE

```

src/shared/syssrc.f

```

521: *                KK = RWK(I)*NPXP
522: C/#ENDIF
523:                if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
524:                    KK      = SRCSP(KPA+KK) - 1
525:                end if
526:
527:                DH(LPSV(1)+I) = RSRC(KX+KK) + RWK(2*NP+I)*
528:                    & (RSRC(KX+KK+1)-RSRC(KX+KK))
529:                180      continue
530: C
531: C ***** else if( cwrk(:ll).eq.'DISCRETE' ) then
532: C                else if ( INTX.eq.2 ) then
533: C                    do 190 I = I1, I2
534: C/#IF ROUNDOFF(NEAREST)
535: C                    KK      = MIN(NPXP-1,INT(RWK(I)*NPXP))
536: C/#ELSE
537: C                    KK = RWK(I)*NPXP
538: C/#ENDIF
539: C                if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
540: C                    KK      = SRCSP(KPA+KK) - 1
541: C                end if
542:
543:                DH(LPSV(1)+I) = RSRC(KX+KK)
544:                190      continue
545: C
546: C
547: C <linear distribution sampling>
548: C
549: C     x1 < x < x2 ;
550: C     p(x) = p1 + (x-x1)/(x2-x1)*(p2-p1)
551: C
552: C     ( p(x1) = p1, p(x2) = p2 )
553: C
554: C     x = x1 + (x2-x1)/(p2-p1)*( sqrt(p1**2+r*(p2**2-p1**2))-p1)
555: C
556: C     = x1 + (x2-x1)*(p1+p2)*r/(sqrt(p1**2+r*(p2**2-p1**2))+p1)
557: C
558: C     The latter form can treat both of the case p1.ne.p2
559: C     and p1.eq.p2 !!
560: C
561: C ***** else if( cwrk(:ll).eq.'LINEAR' ) then
562: C                else if ( INTX.eq.3 ) then
563: C                    do 200 I = I1, I2
564: C/#IF ROUNDOFF(NEAREST)
565: C                    KK      = MIN(NPXP-1,INT(RWK(I)*NPXP))
566: C/#ELSE
567: C                    KK = RWK(I)*NPXP
568: C/#ENDIF
569: C                if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
570: C                    KK      = SRCSP(KPA+KK) - 1
571: C                end if
572:
573:                P1      = RSRC(KPX+KK)
574:                P2      = RSRC(KPX+KK+1)
575:                RR      = (P1+P2)*RWK(2*NP+I)
576:                DH(LPSV(1)+I) = RSRC(KX+KK)
577:                & + (RSRC(KX+KK+1)-RSRC(KX+KK))*RR/
578:                & (SQRT(P1**2+RR*(P2-P1))+P1)
579:                200      continue
580: C
581: C ***** else if( cwrk(:ll).eq.'LOG-LINEAR' ) then
582: C                else if ( INTX.eq.4 ) then
583: C                    do 210 I = I1, I2
584: C/#IF ROUNDOFF(NEAREST)
585: C                    KK      = MIN(NPXP-1,INT(RWK(I)*NPXP))

```

```

586: C/#ELSE
587: *                KK = RWK(I)*NPXP
588: C/#ENDIF
589:                if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
590:                    KK      = SRCSP(KPA+KK) - 1
591:                end if
592:
593:                P1      = RSRC(KPX+KK)
594:                P2      = RSRC(KPX+KK+1)
595:                XL1      = LOG(RSRC(KX+KK))
596:                XL2      = LOG(RSRC(KX+KK+1)) - XL1
597:                RR      = (P1+P2)*RWK(2*NP+I)
598:                DH(LPSV(1)+I) =
599:                & EXP(XL1+XL2*RR/(SQRT(P1**2+RR*(P2-P1))+P1))
600:                210      continue
601: C
602: C ***** else if( cwrk(:ll).eq.'LOG-STEP' ) then
603: C                else if ( INTX.eq.5 ) then
604: C                    do 220 I = I1, I2
605: C/#IF ROUNDOFF(NEAREST)
606: C                    KK      = MIN(NPXP-1,INT(RWK(I)*NPXP))
607: C/#ELSE
608: C                    KK = RWK(I)*NPXP
609: C/#ENDIF
610: C                if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
611: C                    KK      = SRCSP(KPA+KK) - 1
612: C                end if
613:
614:                XL1      = LOG(RSRC(KX+KK))
615:                XL2      = LOG(RSRC(KX+KK+1)) - XL1
616:                DH(LPSV(1)+I) = EXP(XL1+RWK(2*NP+I)*XL2)
617:                220      continue
618: C                end if
619: C
620: C
621: C >>>> TABLEXY
622: C
623: C
624: C                else if ( 'TABLEXY'.eq.CWRK(:NL) ) then
625: C ... x-data ...
626: C
627: C                call UNPKCS( SRCSP, 'INTERPOLATION', CWRK, LL, IRET1 )
628: C                INTX      = 0
629: C                if ( CWRK(:LL).eq.'STEP' ) then
630: C                    INTX      = 1
631: C                else if ( CWRK(:LL).eq.'DISCRETE' ) then
632: C                    INTX      = 2
633: C                else if ( CWRK(:LL).eq.'LINEAR' ) then
634: C                    INTX      = 3
635: C                else if ( CWRK(:LL).eq.'LOG-LINEAR' ) then
636: C                    INTX      = 4
637: C                else if ( CWRK(:LL).eq.'LOG-STEP' ) then
638: C                    INTX      = 5
639: C                end if
640: C
641: C                call UNPKPT( SRCSP, 'X POINTS', 'R4', KX, NX, IRET2 )
642: C                IRET3      = 0
643: C                if ( INDEX(CWRK(:LL),'LINEAR').ne.0 ) then
644: C                    call UNPKPT( SRCSP, 'PRB.AS GIVEN', 'R4', KPX, NPX,
645: C                                & IRET3 )
646: C                end if
647: C                call UNPKPT( SRCSP, 'PROBABILITY', 'R4', KP, NPXP, IRET4
648: C                                & )
649: C                call UNPKPT( SRCSP, 'ALIAS', 'I4', KPA, NPA, IRET5 )
650: C

```

src/shared/syssrc.f

```

651: C ... y-data ...
652:       call UNPKPT( SRCSP, 'Y NUMBER', 'I4', KNYI, NPX, IRET2 )
653:       call UNPKPT( SRCSP, 'PY NUMBER', 'I4', KNPYI, NPX, IRET2
654:       &
655:       call UNPKPT( SRCSP, 'PYP NUMBER', 'I4', KNPYPI, NPX,
656:       & IRET2 )
657:       call UNPKPT( SRCSP, 'INDEX YI', 'I4', KYI, NPX, IRET2 )
658:       call UNPKPT( SRCSP, 'INDEX PYI', 'I4', KPYI, NPX, IRET2 )
659:       call UNPKPT( SRCSP, 'INDEX PYPI', 'I4', KPYPI, NPX, IRET2
660:       &
661:       call UNPKPT( SRCSP, 'ALIAS', 'I4', KPAAYI, NPX, IRET2 )
662:       call UNPKPT( SRCSP, 'Y INT', 'I4', KINTYI, NPX, IRET2 )
663: C
664:       call UNPKPT( SRCSP, 'Y DATA', 'R4', KDY, NDY, IRET2 )
665:       KDY = KDY - 1
666: C
667:       NP = I2 - I1 + 1
668:       call RANU2( IRAND, RWK(I1), 5*NP, ICOD )
669: C
670: *VOCL LOOP,NOVREC
671:       do 230 I = I1, I2
672: C
673:       ... sampling of interval/point # by alias method ....
674: C
675: C/#IF ROUNDOFF(NEAREST)
676:       KK = MIN(NPXP-1,INT(RWK(I)*NPXP))
677: C/#ELSE
678:       *       KK = RWK(I)*NPXP
679: C/#ENDIF
680:       if ( RWK(NP+I).lt.RSRC(KP+KK) ) then
681:       KK = SRCSP(KPA+KK) - 1
682:       end if
683: C
684:       ... sample ...
685: C
686: C ***** if( cwrk(:ll).eq.'STEP' ) then
687:       if ( INTX.eq.1 ) then
688:       DH(LPSV(1)+I) = RSRC(KX+KK) + RWK(2*NP+I)*
689:       & (RSRC(KX+KK+1)-RSRC(KX+KK))
690: C
691: C ***** else if( cwrk(:ll).eq.'DISCRETE' ) then
692:       else if ( INTX.eq.2 ) then
693:       DH(LPSV(1)+I) = RSRC(KX+KK)
694: C
695: C ***** else if( cwrk(:ll).eq.'LINEAR' ) then
696:       else if ( INTX.eq.3 ) then
697:       P1 = RSRC(KPX+KK)
698:       P2 = RSRC(KPX+KK+1)
699:       RR = (P1+P2)*RWK(2*NP+I)
700:       DH(LPSV(1)+I) = RSRC(KX+KK)
701:       & + (RSRC(KX+KK+1)-RSRC(KX+KK))*RR/
702:       & (SQRT(P1**2+RR*(P2-P1))+P1)
703: C
704: C ***** else if( cwrk(:ll).eq.'LOG-LINEAR' ) then
705:       else if ( INTX.eq.4 ) then
706:       P1 = RSRC(KPX+KK)
707:       P2 = RSRC(KPX+KK+1)
708:       XL1 = LOG(RSRC(KX+KK))
709:       XL2 = LOG(RSRC(KX+KK+1)) - XL1
710:       RR = (P1+P2)*RWK(2*NP+I)
711:       DH(LPSV(1)+I) =
712:       & EXP(XL1+XL2*RR/(SQRT(P1**2+RR*(P2-P1))+P1))
713: C
714: C ***** else if( cwrk(:ll).eq.'LOG-STEP' ) then
715:       else if ( INTX.eq.5 ) then

```

```

716:       XL1 = LOG(RSRC(KX+KK))
717:       XL2 = LOG(RSRC(KX+KK+1)) - XL1
718:       DH(LPSV(1)+I) = EXP(XL1+RWK(2*NP+I)*XL2)
719:       end if
720: C
721: C
722: C ... sampling of interval/point # by alias method ....
723: C
724:       NPYP = SRCSP(KNPYPI+KK)
725:       RKY = RWK(3*NP+I)*NPYP
726: C/#IF ROUNDOFF(NEAREST)
727:       KKY = MIN(NPYP-1,INT(RKY))
728: C/#ELSE
729:       *       KKY = RKY
730: C/#ENDIF
731:       if ( RKY-KKY.lt.RSRC(KDY+SRCSP(KPYPI+KK)+KKY) ) then
732:       KKY = SRCSP(KDY+SRCSP(KPAAYI+KK)+KKY) - 1
733:       end if
734: C
735: C
736: C ... sample ...
737:       INTY = SRCSP(KINTYI+KK)
738: C
739: C
740: C 'STEP'
741:       if ( INTY.eq.1 ) then
742:       Y1 = RSRC(KDY+SRCSP(KYI+KK)+KKY)
743:       Y2 = RSRC(KDY+SRCSP(KYI+KK)+KKY+1) - Y1
744:       DH(LPSV(2)+I) = Y1 + RWK(4*NP+I)*Y2
745: C
746: C 'DISCRETE'
747: C
748:       else if ( INTY.eq.2 ) then
749:       DH(LPSV(2)+I) = RSRC(KDY+SRCSP(KYI+KK)+KKY)
750: C
751: C 'LINEAR'
752: C
753:       else if ( INTY.eq.3 ) then
754:       P1 = RSRC(KDY+SRCSP(KPYI+KK)+KKY)
755:       P2 = RSRC(KDY+SRCSP(KPYI+KK)+KKY+1)
756:       Y1 = RSRC(KDY+SRCSP(KYI+KK)+KKY)
757:       Y2 = RSRC(KDY+SRCSP(KYI+KK)+KKY+1) - Y1
758:       RR = (P1+P2)*RWK(4*NP+I)
759:       DH(LPSV(2)+I) = Y1
760:       & + Y2*RR/(SQRT(P1**2+RR*(P2-P1))+P1)
761: C
762: C 'LOG-LINEAR'
763: C
764:       else if ( INTY.eq.4 ) then
765:       P1 = RSRC(KDY+SRCSP(KPYI+KK)+KKY)
766:       P2 = RSRC(KDY+SRCSP(KPYI+KK)+KKY+1)
767:       YL1 = LOG(RSRC(KDY+SRCSP(KYI+KK)+KKY))
768:       YL2 = LOG(RSRC(KDY+SRCSP(KYI+KK)+KKY+1)) - YL1
769:       RR = (P1+P2)*RWK(4*NP+I)
770:       DH(LPSV(2)+I) =
771:       & EXP(YL1+YL2*RR/(SQRT(P1**2+RR*(P2-P1))+P1))
772: C
773: C
774: C 'LOG-STEP'
775: C
776:       else if ( INTY.eq.5 ) then
777:       YL1 = LOG(RSRC(KDY+SRCSP(KYI+KK)+KKY))
778:       YL2 = LOG(RSRC(KDY+SRCSP(KYI+KK)+KKY+1)) - YL1
779:       DH(LPSV(2)+I) = EXP(YL1+RWK(4*NP+I)*YL2)
780:       end if

```

src/shared/syssrc.f

```

781: 230      continue
782:
783: C
784: C
785: C >>>> fission spectrum of a nuclide or a material : TYPEn
786: C
787: C
788: C      else if ( 'FISSION'.eq.CWRK(:NL) ) then
789: C          call UNPKND( SRCSP, 'NUC', 'I4', ITEMP, 1, ND, IRET )
790: C          if ( PROG.eq.'MVP' ) then
791: C              call UNPKND( SRCSP, 'NUC', 'R4', RTEMP, 1, ND, IRET )
792: C          end if
793: C
794: C          call NUCFIS( A, H, DH(LPSV(1)+I1), I2-I1+1, ITEMP(1),
795: C              &      RTEMP(1), IRAND, IERR, RWK )
796: C
797: C
798: C >>>> sampling from fission source file : "#FISSIONFILE"
799: C
800: C
801: C      else if ( 'FISSIONFILE'.eq.CWRK(:NL) ) then
802: C          call UNPKND( SRCSP, 'NPSIZ', 'I4', NPSIZ, 1, ND, IRET )
803: C          call UNPKND( SRCSP, 'SIZES', 'I4', ITEMP, NPSIZ, ND, IRET
804: C              &      )
805: C          NUSE = ITEMP(1)
806: C          JNUCS = ITEMP(2)
807: C      Check %%%%%%%%%%
808: C      write(6,*) '%%(syssrc) nuse jnucs ', nuse, jnucs
809: C      %%%%%%%%%%
810: C          call UNPKPT( SRCSP, 'X', 'R8', KLX, ND, IRET )
811: C          call UNPKPT( SRCSP, 'Y', 'R8', KLY, ND, IRET )
812: C          call UNPKPT( SRCSP, 'Z', 'R8', KLZ, ND, IRET )
813: C          if ( ITEMP(8).gt.0 ) then
814: C              call UNPKPT( SRCSP, 'W', 'R4', KLW, ND, IRET )
815: C              JWGT = 1
816: C          else
817: C              KLW = 1
818: C              JWGT = 0
819: C          end if
820: C      Check %%%%%%%%%%
821: C      write(6,*) '%%(syssrc) klx kly klz ', klx, kly, klz
822: C      %%%%%%%%%%
823: C          if ( JNUCS.ne.0 ) then
824: C              call UNPKPT( SRCSP, 'INUF', 'I4', KLNUCF, ND, IRET )
825: C              call UNPKPT( SRCSP, 'EEEE', 'R4', KLEEEF, ND, IRET )
826: C              call UNPKND( SRCSP, 'NNUCID', 'I4', NNUCID, 1, ND,
827: C                  &      IRET )
828: C              call UNPKPT( SRCSP, 'KNUCID', 'I4', KLKNUC, ND, IRET )
829: C      Check %%%%%%%%%%
830: C      write(6,*) '%%(syssrc) nnucid ', nnucid
831: C      write(6,*) '%%(syssrc) knucid ', (srcsp(klknuc+i-1), i=1, nnucid)
832: C      %%%%%%%%%%
833: C      end if
834: C
835: C          call FSSAMP( A, H, DH(LPSV(1)+I1), DH(LPSV(2)+I1),
836: C              &      DH(LPSV(3)+I1), DH(LPSV(4)+I1), DH(LPSV(5)
837: C              &      +I1), DH(LPSV(6)+I1), DH(LPSV(7)+I1),
838: C              &      DH(LPSV(8)+I1), I2-I1+1, NUSE, JNUCS, JWGT,
839: C              &      DSRC(KLX), DSRC(KLY), DSRC(KLZ), SRCSP(KLNUCF),
840: C              &      RSRC(KLEEEF), RSRC(KLW), NNUCID, SRCSP(KLKNUC),
841: C              &      IRAND, RWK(1), RWK(NS+1), RWK(2*NS+1), IERR )
842: C
843: C      --- in the call above, array RWK must have at least (1+1+5)*NS
844: C      elements.
845: C

```

```

846: C
847: C >>>> old-fashioned source TYPEn
848: C
849: C
850: C      else if ( 'TYPE'.eq.CWRK(:4) ) then
851: C          call UNPKPT( SRCSP, ' ', 'R8', LJ, ND, IRET )
852: C
853: C          call OLDSRC( CWRK(:NL), NS, I1, I2, DSRC(LJ), ND, RWK,
854: C              &      IRAND, H(LX), H(LY), H(LZ), H(LA), H(LB), H(LC),
855: C              &      H(LE), IERS )
856: C
857: C          if ( IERS.ne.0 ) then
858: C              write(IPR,*) 'XXX SAMPLING ERROR IN OLD TYPE SOURCE'
859: C              IPOS = 7
860: C              go to 380
861: C          end if
862: C
863: C
864: C >>>> user-defined source USERn
865: C
866: C
867: C      else if ( 'USER'.eq.CWRK(:4) ) then
868: C          LXX = LDBLE(LX) + I1 - 1
869: C          LYY = LDBLE(LY) + I1 - 1
870: C          LZZ = LDBLE(LZ) + I1 - 1
871: C          LAA = LDBLE(LA) + I1 - 1
872: C          LBB = LDBLE(LB) + I1 - 1
873: C          LCC = LDBLE(LC) + I1 - 1
874: C          LEE = LDBLE(LE) + I1 - 1
875: C          LTT = LDBLE(LT) + I1 - 1
876: C          LWW = LDBLE(LW) + I1 - 1
877: C          NS1 = I2 - I1 + 1
878: C
879: C          call UNPKPT( SRCSP, ' ', 'R8', LJ, ND, IRET )
880: C
881: C          call USRSRC( CWRK(:NL), NS1, IRAND, DSRC(LJ), ND, RWK,
882: C              &      RWK, DH(LXX), DH(LYY), DH(LZZ), DH(LAA), DH(LBB),
883: C              &      DH(LCC), DH(LEE), DH(LTT), DH(LWW), IERS )
884: C
885: C          if ( IERS.ne.0 ) then
886: C              write(IPR,*) 'XXX SAMPLING ERROR IN USER SOURCE'
887: C              IPOS = 8
888: C              go to 380
889: C          end if
890: C
891: C
892: C >>>> vector compression <<<<
893: C
894: C      @( mask vector1 vector2 ... ) = COMPRESS()
895: C
896: C      else if ( 'COMPRESS'.eq.CWRK(:8) ) then
897: C          IVMASK(IACC) = LPSV(1)
898: C          do 250 NN = 2, NSVECT
899: C              ICMP = I1 - 1
900: C          *VOCL LOOP,NOVREC
901: C          do 240 I = I1, I2
902: C              if ( DH(LPSV(1)+I).ne.0.0 ) then
903: C                  ICMP = ICMP + 1
904: C                  DH(LPSV(NN)+ICMP) = DH(LPSV(NN)+I)
905: C              end if
906: C              continue
907: C              if ( ICMP.eq.I2 ) go to 260
908: C          continue
909: C          continue
910: C

```

src/shared/syssrc.f

```

911: C
912: C >>>> vector reordering <<<<
913: C
914: C @( mask work vector1 vector2 ... ) = REORDER(
915: C
916: C vector --> <mask is true> before <mask is false>
917: C
918: C else if ( 'REORDER'.eq.CWRK(:7) ) then
919: C IVMASK(IACC) = LPSV(1)
920: C do 290 NN = 3, NSVECT
921: C ICMP = I1 - 1
922: C ICMP2 = 0
923: C
924: C .... gather false element ....
925: C
926: C *VOCL LOOP,NOVREC
927: C do 270 I = I1, I2
928: C if ( DH(LPSV(1)+I).eq.0.0 ) then
929: C ICMP2 = ICMP2 + 1
930: C DH(LPSV(2)+ICMP2) = DH(LPSV(NN)+I)
931: C else
932: C ICMP = ICMP + 1
933: C DH(LPSV(NN)+ICMP) = DH(LPSV(NN)+I)
934: C end if
935: C 270 continue
936: C
937: C if ( ICMP2.gt.0 ) then
938: C do 280 J = 1, ICMP2
939: C DH(LPSV(NN)+ICMP+J) = DH(LPSV(2)+J)
940: C 280 continue
941: C else
942: C go to 300
943: C end if
944: C 290 continue
945: C 300 continue
946: C
947: C
948: C >>>> [NOT-]IN-BODY
949: C
950: C
951: C else if ( 'IN-BODY'.eq.CWRK(:NL)
952: C & .or. 'NOT-IN-BODY'.eq.CWRK(:NL) ) then
953: C
954: C call UNPKND( SRCSP, 'ID NSFN', 'I4', ITEMP, 2, ND, IRET )
955: C IBDID = ITEMP(1)
956: C NSFN = ITEMP(2)
957: C
958: C if ( MOD(JDEBG(3),2).ne.0 ) then
959: C write(IPR,*) '(SYSSRC) Body ', IBDID, ' nsfn ', NSFN
960: C end if
961: C
962: C call UNPKPT( SRCSP, '-KZAA-', 'I4', KKZAA, ND, IRET )
963: C call UNPKPT( SRCSP, '-KZDA-', 'I4', KKZDA, ND, IRET )
964: C
965: C
966: C ... check X,Y,Z is in a "zone" defined with "KZDA" data in SRCSP.
967: C
968: C IZ = 1
969: C KNZDA = 2
970: C NZN = 1
971: C JJSIMP = 1
972: C JVMNT0 = 0
973: C NP = I2 - I1 + 1
974: C LXX = LDBLE(LX) + I1 - 1
975: C LYY = LDBLE(LY) + I1 - 1

```

```

976: C LZZ = LDBLE(LZ) + I1 - 1
977: C Check %%%
978: C if( np.lt.10) then
979: C do i=0,np-1
980: C write(ipr,'(1x,3d15.5)') DH(lxx+i),DH(lyy+i),DH(lzz+i)
981: C end do
982: C end if
983: C %%%%%%%%%
984: C
985: C call JUDGE( 'SYSSRC', IZ, NP, DH(LXX), DH(LYY), DH(LZZ),
986: C & IWK(1), SDA, SRCSP(KKZDA), SRCSP(KKZAA), NSDA,
987: C & KNZDA, NZN+1, JVMNT0, IWK(NP+1), JJSIMP, .false.,
988: C & DUMMY, DUMMY, DUMMY )
989: C
990: C if ( CWRK(1:4).eq.'NOT-' ) then
991: C do 310 I = I1, I2
992: C IJ = 0
993: C if ( IWK(I-I1+1).eq.0 ) IJ = 1
994: C DH(LPSV(1)+I) = IJ
995: C 310 continue
996: C else
997: C do 320 I = I1, I2
998: C IJ = 0
999: C if ( IWK(I-I1+1).ne.0 ) IJ = 1
1000: C DH(LPSV(1)+I) = IJ
1001: C 320 continue
1002: C end if
1003: C
1004: C
1005: C >>>> [NOT-]IN-ZONE
1006: C
1007: C
1008: C else if ( 'IN-ZONE'.eq.CWRK(:NL)
1009: C & .or. 'NOT-IN-ZONE'.eq.CWRK(:NL) ) then
1010: C
1011: C call UNPKND( SRCSP, '-NNZN-', 'I4', NNZN, 1, ND, IRET )
1012: C call UNPKPT( SRCSP, '-ZONES-', 'I4', KKZON, NNZN, IRET )
1013: C
1014: C if ( MOD(JDEBG(3),2).ne.0 ) then
1015: C write(IPR,*) '(SYSSRC) zone list ',
1016: C & (SRCSP(KKZON+I-1),I=1,NNZN)
1017: C end if
1018: C
1019: C ... only get zone # ...
1020: C JMODE = 0
1021: C JMEMZN = 0
1022: C NP = I2 - I1 + 1
1023: C LXX = LDBLE(LX) + I1 - 1
1024: C LYY = LDBLE(LY) + I1 - 1
1025: C LZZ = LDBLE(LZ) + I1 - 1
1026: C do 330 I = 1, NP
1027: C IWK(I) = 0
1028: C 330 continue
1029: C
1030: C ... get zone #'s on IWK
1031: C
1032: C call ABS2ZN( IPR, A, H, JDEBG, JLATT, JTLLT, JHLAT,
1033: C & NNLOST, JMODE, IERR, NP, DH(LXX), DH(LYY),
1034: C & DH(LZZ), DUMMYA, DUMMYB, DUMMYC, MDUMMY, 0,
1035: C & JMEMZN, IWK(1), IDUMMY, IDUMMY, IDUMMY, IDUMMY,
1036: C & IDUMMY, IDUMMY )
1037: C
1038: C if ( MOD(JDEBG(3),2).ne.0 ) then
1039: C write(IPR,*) '(SYSSRC) zones ', (IWK(I),I=1,NP)
1040: C end if

```


src/shared/syssrc.f

```
1041: C
1042:       do 340 I = I1, I2
1043:           DH(LPSV(1)+I) = 0
1044:       340 continue
1045: C
1046:       do 360 IZ = 1, NNZN
1047:           do 350 I = I1, I2
1048:               if ( IWK(I-I1+1).eq.SRCSP(KKZON+IZ-1) ) then
1049:                   DH(LPSV(1)+I) = 1
1050:               end if
1051:           350 continue
1052:       360 continue
1053: C
1054:       if ( 'NOT-IN-ZONE'.eq.CWRK(:NL) ) then
1055:           do 370 I = I1, I2
1056:               DH(LPSV(1)+I) = 1 - DH(LPSV(1)+I)
1057:           370 continue
1058:       end if
1059: C
1060:       if ( MOD(JDEBG(3),2).ne.0 ) then
1061:           write(IPR,*) '(SYSSRC) flag ', (DH(LPSV(1)+I),I=I1,I2)
1062:       end if
1063: C
1064: C === invalid function name ( may be program error ) ===
1065: C
1066:       else
1067:           NERROR = NERROR + 1
1068:           write(IPR,'(1x,a,a,a)')
1069:           & 'XXX UNSUPPORTED SAMPLING FUNCTION <', CWRK(:NL),
1070:           & '>'
1071:           return
1072:       end if
1073: C
1074: C .... end of function ...
1075: C
1076:       go to 140
1077:   end if
1078: C
1079: end if
1080: C
1081: return
1082: C
1083: C
1084: C XXXXX ERROR XXXXX
1085: C
1086: 380 write(IPR,*) 'XXX SOURCE SAMPLING ERROR IN ', N, ''TH',
1087:   & ' SOURCE. ', IPOS
1088: C
1089:       NERROR = NERROR + 1
1090: C ... endif for SOURCE(NEW)
1091: C/#ENDIF
1092:       return
1093:       end
```

src/shared/talin1.f

```

1:      subroutine TALIN1( CODE, IOWK, NERR, A,      CHA,  IPK,  RPK,
2:      &                  NPS,  ITAL,  IDTAL, NLTEMP,NIDTAL,NDTALY,
3:      &                  NETALY, NGP,  KPLIM,
4:      &                  JNEUT, JPHOT, JTIME, JFISS, JEIGN,
5:      &                  JRESP, JTLLT, JTSRF, JDEBG,
6:      &                  IDSRC, LIDSRC,MKREG,
7:      &                  NUC,   INCST, JJPNT, INCSI, IDSRF,
8:      &                  KE2D,
9:      &                  JPHNU, NPATOM,NUCPN, INCSTPN,MTMPN,   ! ph-nuc-3
10:     &                  IWRK,  RWRK,  DWRK,  NWORK)
11: C=<MVP/GMVP>=====
12: C  purpose: process "special" tally specifications - Stage 1;
13: C      compose IDTAL data etc.
14: C  called in: talin routine of mvp/gmvp.
15: C  calls : iorder,pctchk,iclen2,rwind,ibtile,reglst,cnterr,label,
16: C          unpkcs,unpknd,pctlb,ccomp,imatch,nid2nn,prstop
17: C=====
18: C  arguments ( i=input, o=output, w=work )
19: C i CODE   : name of code using this routine ('MVP'/'GMVP')
20: C i IOWK   : I/O unit number of working file (text) to store temporary
21: C            information for a succeeding process.
22: C o NERR   : number of errors
23: C io  A    : task shared dynamic data area used as "REAL" type.
24: C io  CHA   : task shared dynamic data area used as "CHARACTER*4" type.
25: C i IPK    : data packet container in which $TALLY block is stored.
26: C i RPK    : equivalent to IPK but in REAL type.
27: C i NPS    : size of "IPK" in words.
28: C i ITAL   : number of special tally entry in $TALLY input block.
29: C o IDTAL  : direct tally information.
30: C i NLTEMP : size of memory available to IDTAL ( actual size of IDTAL
31: C            is determined in this routine.)
32: C o NIDTAL : actual size of IDTAL array.
33: C o NDTALY : total number of direct-tally.
34: C o NETALY : total number of edited-tally.
35: C i JNEUT,JPHOT,JTIME,JFISS,JEIGN,JRESP,JTLLT,JDEBG : option flags
36: C i IDSRF(NTSRF) : ID of tally-surfaces
37: C o KE2D : reference of i-th ETALY to DTALY ( used for data processing)
38: C          if negative, tally-ID is stored.
39: C w IWRK : working array (integer)
40: C w RWRK : working array (real)
41: C w DWRK : working array (real*8)
42: C i NWORK : working array size in word.
43: C
44: C (iwr, rwrk & dwrk have the same address)
45: C
46: C-----
47: C
48: C Structure of IDTAL(*)
49: C
50: C +----- direct tally loop ( NDTALY )
51: C |
52: C | (1) ID
53: C | (2) (dummy for debug: -999)
54: C | (3) pointer to DTALY(*)
55: C | (4) event #
56: C | (5) particle type
57: C | (6) number of dimensions
58: C | (7) multiplication factor type
59: C | (8) nuclide/atom # & reaction #
60: C | (9) direct tally #
61: C | (10) custom tally # (non-zero means user customized treatment)
62: C | (11) detector # (for surface event positive tally-surface #
63: C |      means particle current tally, and negative one
64: C |      means flux tally)
65: C | ( IDTOFF data so far )

```

```

66: C |
67: C | +----- tally dimension loop
68: C | | (1) dimension type
69: C | | (2) number of tallied bins
70: C | | (3) size of common bin -> tallied bin conversion table
71: C | | ( IDMOFF data so far )
72: C | | +----- common bin loop
73: C | | | tallied bin # for each common bin
74: C | | +-----
75: C | +-----
76: C +-----
77: C=====
78: C
79: C      character*(*) CODE
80: C
81: C      real A(*)
82: C      character*4 CHA(*)
83: C
84: C      integer IPK(NPS)
85: C      real RPK(NPS)
86: C      integer IDTAL(NLTEMP)
87: C      integer IWRK(NWORK)
88: C      real RWRK(NWORK)
89: C      real*8 DWRK(*)
90: C
91: C      integer NGP(KPLIM)
92: C      integer JDEBG(*)
93: C
94: C      integer KE2D(ITAL)
95: C
96: C      integer IDSRC(NSOUR), LIDSRC
97: C      integer MKREG(NREG,2)
98: C      integer INCST(NUC,KPLIM), INCSI(NUC,KPLIM)
99: C      integer INCSTPN(ITAL,2), MTMPN(NMTMPN)      ! ph-nuc-3
100: C
101: C      integer IDSRF(NTSRF)
102: C
103: C ... offset of direct-tally data in IDTAL(*)
104: C
105: CC      parameter( IDTOFF   = 12 )
106: C
107: C ... offset of tally dimension dependent data in IDTAL(*)
108: C
109: CC      parameter( IDMOFF   = 3 )
110: C
111: C      include 'INC/_IDXOFF'
112: C
113: C      CCCCCinclude 'INC/_ARRAY'
114: C      Ccccc include 'INC/_FLAGS'
115: C      include 'INC/_SIZES'
116: C      include 'INC/_PTALY0'
117: C      include 'INC/_PTLSP'
118: C      CCCCCinclude 'INC/_STALY'
119: C      include 'INC/_IOUNIT'
120: C
121: C ===== local variables =====
122: C
123: C      character*6 TAG
124: C      character*12 THREAD
125: C      integer ID
126: C      character*72 TLABEL
127: C      character*16 EVENTN
128: C      character*8 PARTCL
129: C      character*16 NUCNAM
130: C      character*32 REACT

```

```
src/shared/talin1.f
```

```

131:      parameter( MAXDIM    = 8 )
132:      character*24 TALDIM(MAXDIM)
133: C
134:      character CWRK*128
135:      character PLAB*32
136:      character*2 TYP
137: C
138: -----
139: C
140:      JNP          = JNEUT*JPHOT
141:      call RWIND( IOWK )
142: C
143:      NERR         = 0
144: C
145: C
146:      NTALY        = 0
147:      NEITALY       = 0
148:      IIMIPN       = 0 ! ph-nuc-3
149: C
150:      KKDT         = 1
151: C
152:      IDT          = 0
153: C
154: C
155:      do 660 IT = 1, ITAL
156: C
157:           KE2D(IT)   = 0
158:           JADD       = 0
159: C
160:           CWRK       = ' '
161:           write(CWRK,'('&'&TALLY' ',i5)') IT
162:           *****
163:           call LABEL( IPR, CWRK(:ICLEN2(CWRK)) )
164:           *****
165: C
166: c##<2007/03/14:PN3:
167: c##             if ( IDT+IDTOFF.gt.NLTEMP ) go to 999
168:                 if ( IDT+IDTOFF.gt.NLTEMP ) go to 9002
169: c##>
170: C
171:             do 100 I = 1, IDTOFF
172:                  IDTAL(IDT+I)     = 0
173:            100 continue
174: C
175:             TAG       = ' '
176:             write(TAG,'(i5,''.'''')') IT
177:             call CCOMP( TAG, LEN(TAG), TAG, IIF )
178:             LTAG      = ICLEN(TAG)
179:             PLAB       = TAG(:LTAG) //'&TALLY'
180: C
181: C <<<<<<<<<<<<
182:             call PCTLB( IPK, 'tally header', PLAB(:ICLEN2(PLAB)), IRET )
183: C <<<<<<<<<<<<
184: C
185: c##<2007/03/14:PN3:
186: c##             if ( IRET.ne.0 ) go to 666
187:                 if ( IRET.ne.0 ) go to 9001
188: c##>
189: C
190: C=====
191: C === data for each special tally section ===
192: C=====
193:             IDTAL(IDT+2)     = -999
194: C
195: C ... ID ...
```

```

196: C
197:         PLAB      = TAG(:LTAG) //'ID'
198: C
199: C <<<<<<<<<<
200:         call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
201: C <<<<<<<<<<
202: C
203:         if ( IRET.eq.0 ) then
204: C <<<<<<<<<<
205:             NDATA    = 1
206:             call UNPKND( IPK, 'ID', 'I4', ID, NDATA, NDATA2, IRET )
207: C <<<<<<<<<<
208:             IDTAL(IDT+1)   = ID
209:             else
210:                 IDTAL(IDT+1)   = 0
211:             end if
212: C
213:             write(IPR,'(1X,A,I7)') '* ID # : ', IDTAL(IDT+1)
214: C
215:             IDTAL(IDT+2)     = -999
216: C
217:             IDTAL(IDT+3)     = KKDT
218:             IDSIZ    = 1
219: C
220: C ... D-tally #
221: C
222: CCCC       IDTAL(IDT+9)     = IT
223:           IDTAL(IDT+9)     = NDATALY + 1
224: C
225: C ... LABEL ... skipped in Stage 1
226: C
227: C
228: C ... NEUTRON/PHOTON ...
229: C
230: C <<<<<<<<<<
231:         PLAB      = TAG(:LTAG) //'PARTICLE'
232:         call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
233: c##<2007/03/14:PN3:
234: c##        if ( IRET.ne.0 ) go to 666
235: c##        if ( IRET.ne.0 ) go to 9001
236: c##>
237: C <<<<<<<<<<
238: C
239: C <<<<<<<<<<
240:         call UNPKCS( IPK, 'PARTICLE', PARTCL, NLP, IRET )
241: C <<<<<<<<<<
242: C
243:         call KPSYMB( PARTCL(:NLP), '>', IPART, 0 )
244:         NGRP    = 1
245:         if ( IPART.eq.0 ) then
246:             write(IMG,* ) 'XXX(TALIN1) Unsupported particle type: ',
247:               &          PARTCL(:NLP)
248:             call PRSTOP( 1, 'Unsupported particle in $TALLY' )
249:             stop 888
250:         else
251:             NGRP    = NGP(IPART)
252:         end if
253: C
254:         if ( PARTCL(1:1).eq.'N' ) then
255:             IPART    = 1
256:             PARTCL    = 'NEUTRON'
257:             NGRP      = NGP1
258:         else if ( PARTCL(1:1).eq.'P' ) then
259:             IPART    = 2
260:             PARTCL    = 'PHOTON'

```

```

Event #'s currently defined are as follows:
1 : Particle generation (source or gamma-generation or
    next event tally)
2 : Free flight ( -> default )
3 : Boundary/surface crossing
4 - 6 : (spare for future)
7 : leakage
8 : analog events of neutron for noise analysis
9 : analog events of photon for noise analysis
10 : neutron collision
11 : photon collision
12 : electron collision

    PLAB      = TAG(:LTAG) / 'EVENT'
<<<<<<<<<
    call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
<<<<<<<<<

... default event (detector) is track length ...
if ( IRET.ne.0 ) then
    EVENTN = 'TRACK'
    IDTAL(IDT+4) = 2
else

<<<<<<<<<
    call UNPKCS( IPK, 'EVENT', EVENTN, NLE, IRET )
<<<<<<<<<

if ( EVENTN(1:4).eq.'TRAC' ) then

    IDTAL(IDT+4) = 2
    EVENTN = 'TRACK'

else if ( EVENTN(1:4).eq.'COLL' ) then

    EVENTN = 'COLLISION'
    if ( PARTCL(1:1).eq.'N' ) then
        IDTAL(IDT+4) = 10
    else if ( PARTCL(1:1).eq.'P' ) then
        IDTAL(IDT+4) = 11
    else

```

```

Event #'s currently defined are as follows:
1 : Particle generation (source or gamma-generation or
    next event tally)
2 : Free flight ( -> default )
3 : Boundary/surface crossing
4 - 6 : (spare for future)
7 : leakage
8 : analog events of neutron for noise analysis
9 : analog events of photon for noise analysis
10 : neutron collision
11 : photon collision
12 : electron collision

    PLAB      = TAG(:LTAG) / 'EVENT'
<<<<<<<<<
    call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
<<<<<<<<<

... default event (detector) is track length ...
if ( IRET.ne.0 ) then
    EVENTN = 'TRACK'
    IDTAL(IDT+4) = 2
else

<<<<<<<<<
    call UNPKCS( IPK, 'EVENT', EVENTN, NLE, IRET )
<<<<<<<<<

if ( EVENTN(1:4).eq.'TRAC' ) then

    IDTAL(IDT+4) = 2
    EVENTN = 'TRACK'

else if ( EVENTN(1:4).eq.'COLL' ) then

    EVENTN = 'COLLISION'
    if ( PARTCL(1:1).eq.'N' ) then
        IDTAL(IDT+4) = 10
    else if ( PARTCL(1:1).eq.'P' ) then
        IDTAL(IDT+4) = 11
    else

```

```

Event #'s currently defined are as follows:
1 : Particle generation (source or gamma-generation or
    next event tally)
2 : Free flight ( -> default )
3 : Boundary/surface crossing
4 - 6 : (spare for future)
7 : leakage
8 : analog events of neutron for noise analysis
9 : analog events of photon for noise analysis
10 : neutron collision
11 : photon collision
12 : electron collision

    PLAB      = TAG(:LTAG) / 'EVENT'
<<<<<<<<<
    call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
<<<<<<<<<

... default event (detector) is track length ...
if ( IRET.ne.0 ) then
    EVENTN = 'TRACK'
    IDTAL(IDT+4) = 2
else

<<<<<<<<<
    call UNPKCS( IPK, 'EVENT', EVENTN, NLE, IRET )
<<<<<<<<<

if ( EVENTN(1:4).eq.'TRAC' ) then

    IDTAL(IDT+4) = 2
    EVENTN = 'TRACK'

else if ( EVENTN(1:4).eq.'COLL' ) then

    EVENTN = 'COLLISION'
    if ( PARTCL(1:1).eq.'N' ) then
        IDTAL(IDT+4) = 10
    else if ( PARTCL(1:1).eq.'P' ) then
        IDTAL(IDT+4) = 11
    else

```

[illegible][illegible]

src/shared/talin1.f

```

521: C
522: 190      write(IMG, '(1X,A,A,A,A,A)')
523:      &      'XXX($TALLY) error in description',
524:      &      ' of neutron reaction MT # : <', CWRK(:NL),
525:      &      '>'
526:      call CNTERR( 'FATAL' )
527:      NERR = NERR + 1
528: C
529: C      ... factor may be taken from SGTAL ...
530: C 195      IDTAL(IDT+7) = 2000
531: C
532: C      write(IPR,*) '* WHAT IS TALLIED : ',
533: C      &      'rate of neutron reaction MT #', IMT,
534: C      &      ' of ', NUCNAM
535: C
536: C      ... put reaction MT in idtal(8) (temporary)
537: C
538: C      IDTAL(IDT+8) = 1024*II + IMT
539: C      else
540: C      if ( IMATCH('TOT*',CWRK(:NL)).eq.1 ) then
541: C      IMT = 1
542: C      REACT = 'TOTAL'
543: C      else if ( IMATCH('NUF*',CWRK(:NL)).eq.1 ) then
544: C      IMT = 2
545: C      REACT = 'NU*FISSION'
546: C      else if ( IMATCH('FIS*',CWRK(:NL)).eq.1 ) then
547: C      IMT = 3
548: C      REACT = 'FISSION'
549: C      else if ( IMATCH('ELA*',CWRK(:NL)).eq.1 ) then
550: C      IMT = 4
551: C      REACT = 'ELASTIC SC.'
552: C      else if ( IMATCH('CAP*',CWRK(:NL)).eq.1 ) then
553: C      IMT = 5
554: C      REACT = 'CAPTURE'
555: C      else if ( IMATCH('INE*',CWRK(:NL)).eq.1 ) then
556: C      IMT = 6
557: C      REACT = 'INELASTIC SC.'
558: C      else if ( IMATCH('N2N*',CWRK(:NL)).eq.1 ) then
559: C      IMT = 7
560: C      REACT = '(N,2N)'
561: C      else if ( IMATCH('LOS*',CWRK(:NL)).eq.1 ) then
562: C      IMT = 8
563: C      REACT = 'LOSS'
564: C      end if
565: C      IDTAL(IDT+8) = 1024*II + IMT
566: C
567: C      write(IPR, '(1X,A,A,A,A,A)') '* WHAT IS TALLIED : ',
568: C      &      'rate of neutron reaction <',
569: C      &      REACT(:ICLEN2(REACT)), '> of ', NUCNAM
570: C
571: C      end if
572: C      if ( IMT.eq.-999 ) then
573: C      write(IMG, '(1X,A,A,A,A,A)')
574: C      &      'XXX($TALLY) neutron reaction type',
575: C      &      ' for "MICRO" is invalid : <', CWRK(:NL),
576: C      &      '>'
577: C      call CNTERR( 'FATAL' )
578: C      NERR = NERR + 1
579: C      end if
580: C      MM = MOD(IDTAL(IDT+7),2)
581: C      if ( MM.eq.1 ) then
582: C      write(IPR, '(1X,A,A,A)') '* TALLY MODE : ',
583: C      &      'weighted by number density.'
584: C      else if ( MM.eq.0 ) then
585: C      write(IPR, '(1X,A,A,A)') '* TALLY MODE : ',

```

```

586:      &      'per atom reaction is tallied everywhere.'
587:      &      end if
588: C
589: C      --- photon
590: C
591: C      else if ( IPART.eq.2 ) then
592: C      IMT = -999
593: C
594: C      ... reaction type 1-8 corresponds to that for LMIC/LMAC
595: C
596: C      if ( IMATCH('MT*',CWRK(:NL)).eq.1 ) then
597: C      CWRK(NL+1:) = ' '
598: C      read(CWRK(3:7), '(bn,i5)', err =200) IMT
599: C      go to 205
600: C
601: C 200      write(IMG, '(1X,A,A,A,A,A)')
602: C      &      'XXX($TALLY) error in description',
603: C      &      ' of photon reaction MT # : <', CWRK(:NL),
604: C      &      '>'
605: C      call CNTERR( 'FATAL' )
606: C      NERR = NERR + 1
607: C
608: C 205      IDTAL(IDT+7) = 2000
609: C
610: C      write(IPR,*) '* WHAT IS TALLIED : ',
611: C      &      'rate of photon reaction mt', IMT, ' of ',
612: C      &      NUCNAM
613: C
614: C      ... put reaction MT in idtal(8) (temporary)
615: C
616: C      IDTAL(IDT+8) = 1024*II + IMT
617: C      else
618: C      if ( IMATCH('TOT*',CWRK(:NL)).eq.1 ) then
619: C      IMT = 1
620: C      REACT = 'TOTAL'
621: C      else if ( IMATCH('COH*',CWRK(:NL)).eq.1 ) then
622: C      IMT = 2
623: C      REACT = 'COHERENT SCATTERING'
624: C      else if ( IMATCH('INC*',CWRK(:NL)).eq.1 ) then
625: C      IMT = 3
626: C      REACT = 'INCOHERENT SCATTERING'
627: C      else if ( IMATCH('P*PR*',CWRK(:NL)).eq.1 ) then
628: C      IMT = 4
629: C      REACT = 'ELECTRON PAIR PRODUCTION'
630: C      else if ( IMATCH('P*EL*',CWRK(:NL)).eq.1 ) then
631: C      IMT = 5
632: C      REACT = 'PHOTO-ELECTRON GENERATION'
633: C      end if
634: C
635: C      IDTAL(IDT+8) = 1024*II + IMT
636: C      write(IPR, '(1X,A,A,A,A,A)') '* WHAT IS TALLIED : ',
637: C      &      'rate of photon reaction <',
638: C      &      REACT(:ICLEN2(REACT)), '> of ', NUCNAM
639: C
640: C      end if
641: C
642: C      if ( IMT.eq.-999 ) then
643: C      write(IMG, '(1X,A,A,A,A,A)')
644: C      &      'XXX($TALLY) photon reaction type',
645: C      &      ' for "MICRO" is invalid : <', CWRK(:NL),
646: C      &      '>'
647: C      call CNTERR( 'FATAL' )
648: C      NERR = NERR + 1
649: C      end if
650: C      MM = MOD(IDTAL(IDT+7),2)

```

[illegible]

```

116:         else if ( IMATCH('NUF*',CWRK(:NL)).eq.1 ) then
117:             IMT          = 2
118:             REACT        = 'NU*FISSION'
119:         else if ( IMATCH('FIS*',CWRK(:NL)).eq.1 ) then
120:             IMT          = 3
121:             REACT        = 'FISSION'
122:         else if ( IMATCH('ELA*',CWRK(:NL)).eq.1 ) then
123:             IMT          = 4
124:             REACT        = 'ELASTIC SC.'
125:         else if ( IMATCH('CAP*',CWRK(:NL)).eq.1 ) then
126:             IMT          = 5
127:             REACT        = 'CAPTURE'
128:         else if ( IMATCH('INE*',CWRK(:NL)).eq.1 ) then
129:             IMT          = 6
130:             REACT        = 'INELASTIC SC.'
131:         else if ( IMATCH('N2N*',CWRK(:NL)).eq.1 ) then
132:             IMT          = 7
133:             REACT        = '(N,2N)'
134:         else if ( IMATCH('LOS*',CWRK(:NL)).eq.1 ) then
135:             IMT          = 8
136:             REACT        = 'LOSS'
137:         end if
138:         IDTAL(IDT+8)      = IMT
139:         write(IPR, '(1X,A,A,A,A,A)') ' * WHAT IS TALLIED : ',
140:             & 'rate of macroscopic neutron reaction <',
141:             & REACT(:ICLEN2(REACT)), '>'
142:         end if
143:
144:         if ( IMT.eq.-999 ) then
145:             write(IMG, '(1X,A,A,A,A,A)')
146:             & 'XXX($TALLY) neutron reaction type',
147:             & ' for "MACRO" is invalid : <', CWRK(:NL),
148:             & '>'
149:             call CNTERR( 'FATAL' )
150:             NERR      = NERR + 1
151:         end if
152: C
153: C      --- photon
154: C
155:         else if ( IPART.eq.2 ) then
156:             IMT          = -999
157: C
158: C      ... reaction type 1-8 corresponds to that for LMIC/LMAC
159: C
160:         if ( IMATCH('MT*',CWRK(:NL)).eq.1 ) then
161:             CWRK(NL+1:) = ' '
162:             read(CWRK(3:7), '(bn,i5)', err =220) IMT
163:             go to 225
164: C
165:             220 write(IMG, '(1X,A,A,A,A,A)')
166:                 & 'XXX($TALLY) error in description',
167:                 & ' of photon reaction MT # : <', CWRK(:NL),
168:                 & '>'
169:             call CNTERR( 'FATAL' )
170:             NERR      = NERR + 1
171: C
172:             225 IDTAL(IDT+7)      = 1
173: C
174: C      ... put reaction MT in idtal(8) (temporary)
175: C
176:             IDTAL(IDT+8)      = IMT
177:             write(IPR,*) ' * WHAT IS TALLIED : ',
178:                 & 'rate of macroscopic neutron reaction MT',
179:                 & IMT
180:         else

```

```

      REACT = 'ELECTRON PAIR PRODUCTION'
    else if ( IMATCH('PEL*',CWRK(:NL)).eq.1 ) then
      IMT = 5
      REACT = 'PHOTO-ELECTRON GENERATION'
    end if
    IDTAL(IDT+8) = IMT
    write(IPR, '(1X,A,A,A,A)') '* WHAT IS TALLIED : ',
      'rate of macroscopic neutron reaction <',
      REACT(:ICLEN2(REACT)), '>'
  end if

  if ( IMT.eq.-999 ) then
    write(IMG, '(1X,A,A,A,A)')
      'XXX($TALLY) photon reaction type',
      'for "MACRO" is invalid : <', CWRK(:NL),
      '>'
    call CNTERR( 'FATAL' )
    NERR = NERR + 1
  end if
end if
007/03/14:PN3:

===== MICROPN for photonuclear =====

    else if ( IMATCH('MICROPN:',CWRK(:NL)).eq.1 ) then

      if ( IPART.ne.2 ) then
        write(IMG,9118)
        call CNTERR( 'FATAL' )
        NERR = NERR + 1
      end if

      .... check photonuclear nuclide ID
      K = INDEX(CWRK(:NL),':')
      K2 = INDEX(CWRK(K+1:NL),':') + K
      K1 = K2 - 1
      if ( K.eq.K2 ) K1 = NL
      NUCNAM = CWRK(K+1:K1)
      IPARTPN = 3
      call NID2NN( NUCNAM, IPARTPN, II, A, CHA )
      if ( II.eq.0 ) then
        write(IMG,9119) NUCNAM
        call CNTERR( 'FATAL' )
        NERR = NERR + 1
      end if
    end if
  end if
end if

```

```

      REACT = 'ELECTRON PAIR PRODUCTION'
    else if ( IMATCH('PEL*',CWRK(:NL)).eq.1 ) then
      IMT = 5
      REACT = 'PHOTO-ELECTRON GENERATION'
    end if
    IDTAL(IDT+8) = IMT
    write(IPR, '(1X,A,A,A,A)') '* WHAT IS TALLIED : ',
      'rate of macroscopic neutron reaction <',
      REACT(:ICLEN2(REACT)), '>'
  end if

  if ( IMT.eq.-999 ) then
    write(IMG, '(1X,A,A,A,A)')
      'XXX($TALLY) photon reaction type',
      'for "MACRO" is invalid : <', CWRK(:NL),
      '>'
    call CNTERR( 'FATAL' )
    NERR = NERR + 1
  end if
end if
007/03/14:PN3:

===== MICROPN for photonuclear =====

    else if ( IMATCH('MICROPN:',CWRK(:NL)).eq.1 ) then

      if ( IPART.ne.2 ) then
        write(IMG,9118)
        call CNTERR( 'FATAL' )
        NERR = NERR + 1
      end if

      .... check photonuclear nuclide ID
      K = INDEX(CWRK(:NL),':')
      K2 = INDEX(CWRK(K+1:NL),':') + K
      K1 = K2 - 1
      if ( K.eq.K2 ) K1 = NL
      NUCNAM = CWRK(K+1:K1)
      IPARTPN = 3
      call NID2NN( NUCNAM, IPARTPN, II, A, CHA )
      if ( II.eq.0 ) then
        write(IMG,9119) NUCNAM
        call CNTERR( 'FATAL' )
        NERR = NERR + 1
      end if
    end if
  end if
end if

```

[illegible]

[illegible][illegible]

[illegible][illegible]

```

1171:      write(IOWK,'(A6,I8)') 'ENG ', MX
1172:      do 270 I = 1, MX
1173:         NR       = 0
1174:         do 260 J = 1, NGRP
1175:            if ( IDTAL(IDST+IDMOFF+J).eq.I ) then
1176:               NR     = NR + 1
1177:               IWRK(NR) = J
1178:            end if
1179:        260 continue
1180:           write(IOWK,'(i8)') NR
1181:           write(IOWK,'(l0i8)') (IWRK(J),J=1,NR)
1182: 270 continue
1183: C ... IENERGY2 ...
1184: C
1185: C
1186:      else if ( CWRK(:NL).eq.'IENERGY2' ) then
1187: C <<<<<<<<<<<<
1188:      call UNPKND( IPK, 'ENERGY', 'I4', IWRK, NWORK, ND,
1189:                  & IRET )
1190: C <<<<<<<<<<<<
1191: C
1192:      write(IPR,'(lX,A)') '          ENERGY BIN RANGE'
1193: C
1194:      write(IOWK,'(a6,i8)') 'ENG ', ND/2
1195: C
1196:      do 290 I = 1, ND, 2
1197:         KK   = (I+1)/2
1198:         I1   = IWRK(I)
1199:         I2   = IWRK(I+1)
1200:         write(IPR,'(lX,A,I4,A,I4,A,I4)')
1201:             '          BIN ', KK, ': ', I1, ' to ',
1202:             & I2
1203: C
1204:         if ( I1.lt.1 .or. I1.gt.NGRP .or. I2.lt.1
1205:             & .or. I2.gt.NGRP ) then
1206:            write(IMSG,'(lX,A,A,A)')
1207:                & 'XXX($TALLY) energy group ',
1208:                & ' specification of type "IENERGY2" ',
1209:                & 'is out of range.'
1210:            write(IMSG,'(lX,A,I4,A,I4,A,I4,A,I4,A)')
1211:                & ' range ', KK, ': ', I1,
1212:                & ' to ', I2, ' (should be ',
1213:                & 'within 1 to ', NGRP, ') '
1214:            NERR    = NERR + 1
1215:            call CNTERR('FATAL' )
1216: C
1217:            I1      = MIN(NGRP,MAX(I1,1))
1218:            I2      = MIN(NGRP,MAX(I2,1))
1219:         end if
1220: C
1221: C
1222:         if ( I1.gt.I2 ) then
1223:            III   = I1
1224:            I1    = I2
1225:            I2    = III
1226:         end if
1227: C
1228:         do 280 J = I1, I2
1229:            IWRK(ND+J-I1+1) = J
1230: 280 continue
1231: C
1232:      NR      = I2 - I1 + 1
1233: C
1234:      write(IOWK,'(I8)') NR
1235:      write(IOWK,'(l0I8)') (J,J=I1,I2)

```

[illegible]

```

do 320 I = 1, MX
  NR      = 0
  do 310 J = 1, NTIME
    if ( IDTAL(IDST+IDMOFF+J).eq.I ) then
      NR      = NR + 1
      IWRK(NR) = J
    end if
  continue
  write(IOWK,'(i8)') NR
  write(IOWK,'(10i8)') (IWRK(J),J=1,NR)
  continue

... ITIME2 ...

else if (CWRK(:NL).eq.'ITIME2') then
<<<<<<<<<<
call UNPKND( IPK, 'ITIME2', 'I4', IWRK, NWORK, ND,
             IRET )
<<<<<<<<<<
write(IPR,'(1X,A)') '          TIME BIN RANGE'

write(IOWK,'(a6,i8)') 'TIME ', ND/2

do 340 I = 1, ND, 2
  KK      = (I+1) / 2
  I1      = IWRK(I)
  I2      = IWRK(I+1)
  write(IPR,'(1X,A,I4,A,I5,I5)') '          BIN ',
    KK, ': ', I1, I2

  if ( I1.lt.1 .or. I1.gt.NTIME .or. I2.lt.1
      .or. I2.gt.NTIME ) then
    write(IMG,'(1X,A,A,A)')
      'XXX($TALLY) Time bin',
      ' specification of type "ITIME2" ',
      ' is out of range.'
    write(IMG,'(1X,A,I4,A,I4,A,I4,A,A,I4,A)')
      ' range ', KK, ' : ', I1,
      ' to ', I2, ' (should be ',
      ' within 1 to ', NTIME, ') '
    NERR      = NERR + 1
    call CNTERR( 'FATAL' )

    I1      = MIN(NTIME,MAX(I1,1))

```

[illegible][illegible]

[illegible][illegible]

[illegible]

```

1626: C
1627: C          ... renage_2 < 0 means
1628: C          else if ( I2.lt.0 ) then
1629: C              ICHECK = 1
1630: C          end if
1631: C          430      continue
1632: C          if ( ICHECK.ne.0 ) NGN = NGN + 1
1633: C          c##<2007/03/14:PN3:
1634: C          c##      if ( IDST+IDMOFF+NGN.gt.NLTEMP ) go to 999
1635: C          if ( IDST+IDMOFF+NGN.gt.NLTEMP ) go to 9002
1636: C          c##>
1637: C
1638: C          do 450 I = 1, ND, 2
1639: C              KK = (I+1) /2
1640: C              I1 = IWRK(I)
1641: C              I2 = IWRK(I+1)
1642: C              if ( I2.lt.0 ) I2 = NGN
1643: C              do 440 J = I1, I2
1644: C                  IWRK(ND+J-I1+1) = J
1645: C              440      continue
1646: C
1647: C              NR = I2 - I1 + 1
1648: C              write(IOWK,'(I8)') NR
1649: C              write(IOWK,'(10I8)') (J,J=I1,I2)
1650: C
1651: C              call IBTILE( IDTAL(IDST+IDMOFF+1), NGN, KK,
1652: C              &              IWRK(ND+1), NR )
1653: C              450      continue
1654: C
1655: C              call IORDER( IDTAL(IDST+IDMOFF+1), NGN, MX )
1656: C              IDTAL(IDST+2) = MX
1657: C              IDTAL(IDST+3) = NGN
1658: C          end if
1659: C      end if
1660: C
1661: C      ==== source ===
1662: C
1663: C      else if ( TALDIM(K).eq.'SOURCE' ) then
1664: C
1665: C          write(IPR,'(1X,A)') '          * SOURCE SET'
1666: C
1667: C          IDTAL(IDST+1) = 6
1668: C
1669: C          IDTAL(IDST+2) = 0
1670: C          IDTAL(IDST+3) = NSOUR
1671: C
1672: C          if ( NSOUR.eq.0 .or. LIDSRC.eq.0 ) then
1673: C              if ( NSOUR.eq.0 ) then
1674: C                  write(IMG,'(1X,A)')
1675: C              &          'XXX($TALLY) No source set has been input yet.'
1676: C              end if
1677: C              if ( LIDSRC.eq.0 ) then
1678: C                  write(IMG,'(1X,A)')
1679: C              &          'XXX($TALLY) No source set ID has been input yet.'
1680: C              end if
1681: C              write(IMG,'(1X,A,A)')
1682: C              &          'XXX($TALLY) To use "SOURCE" dimension, input'
1683: C              &          ' , '
1684: C              &          ' for source set ($SOURCE) must precede $TALLY.'
1685: C              call CNTERR( 'FATAL' )
1686: C              NERR = NERR + 2
1687: C          end if
1688: C
1689: C          c##<2007/03/14:PN3:
1690: C          c##      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 999

```

[illegible]

```

1756:      do 520 I = 1, NSOUR
1757:         if ( IDTAL(IDST+IDMOFF+I).eq.J ) then
1758:            NR          = NR + 1
1759:            IWRK(NR)     = I
1760:         end if
1761:      520      continue
1762:      write(IOWK,'(i8)') NR
1763:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)
1764:      530      continue
1765: C
1766: C
1767: C      ==== source region ===
1768: C
1769:      else if ( IMATCH('SOU*-REG*',TALDIM(K)).eq.1 ) then
1770:         write(IPR,'(1X,A)') '          * SOURCE REGION'
1771:         IDTAL(IDST+1) = 7
1772: C
1773:         IDTAL(IDST+2) = 0
1774:         IDTAL(IDST+3) = NREG
1775: C
1776: c##<2007/03/14:PN3:
1777: c##      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 999
1778:      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 9002
1779: c##>
1780: C
1781: C <<<<<<<<<<<<
1782:      PLAB      = TAG(:LTAG) //'SOURCE-REGION'
1783:      call PCTLB2( IPK, 'SOURCE-REGION', PLAB(:ICLEN2(PLAB)),
1784:      &          IRET )
1785: C <<<<<<<<<<<<
1786: C
1787:      if ( IRET.eq.0 ) then
1788: C
1789:      ... number of specified region-sets ...
1790: C
1791: C <<<<<<<<<<<<
1792:      PLAB      = TAG(:LTAG) //'END-REGION'
1793:      call PCTLB( IPK, 'REGION', PLAB(:ICLEN2(PLAB)), IRET )
1794:      call UNPKND( IPK, 'NUM', 'I4', NRG, 1, NDN, IRET )
1795: C <<<<<<<<<<<<
1796: C
1797: C <<<<<<<<<<<<
1798:      PLAB      = TAG(:LTAG) //'SOURCE-REGION'
1799:      call PCTLB2( IPK, 'SOURCE-REGION',
1800:      &          PLAB(:ICLEN2(PLAB)), IRET )
1801: C
1802: c##<2007/03/14:PN3:
1803: c##      if ( IDST+IDMOFF+NREG.gt.NLTEMP ) go to 999
1804:      if ( IDST+IDMOFF+NREG.gt.NLTEMP ) go to 9002
1805: c##>
1806: C
1807:      write(IOWK,'(A6,I8)') 'SRCREG', NRG
1808: C
1809:      IREGB      = 0
1810: C <<<<<<<<<<<<
1811:      540      call PCTCHK( IPK, 'DIM', TYP, LPT, NDATA, IRET )
1812: C <<<<<<<<<<<<
1813:      if ( TYP.ne.'LB' ) then
1814:         IREGB      = IREGB + 1
1815: C <<<<<<<<<<<<
1816:      call UNPKCS( IPK, 'REG', CWRK, NL, IRET )
1817: C <<<<<<<<<<<<
1818:      write(IPR,'(1X,A,I6,A,A)') '          #', IREGB,
1819:      &          ' : ', CWRK(:NL)
1820: C

```

```

1821:      call REGLST( CWRK(:NL), IWRK, NWORK, NR, NR2, A,  

1822:                  &          CHA, IERR )  

1823: C  

1824:      if ( NR.eq.0 ) then  

1825:        write(IMGF,'(1X,A,A,A,A)')  

1826:        &          'XXX($TALLY) no region that',  

1827:        &          ' matches a name <', CWRK(:NL), '>'  

1828:        call CNTERR( 'FATAL' )  

1829:        NERR = NERR + 1  

1830:      else if ( NR2.gt.NR ) then  

1831:        write(IMGF,'(1X,A,A,A,A)')  

1832:        &          'XXX($TALLY) too many regions that',  

1833:        &          ' matches name <', CWRK(:NL), '>'  

1834:        write(IMGF,'(1X,A,A,I6,A,A)')  

1835:        &          ' Please specify "NWORK" data',  

1836:        &          ' at least greater than ', NR2,  

1837:        &          ' ( place it between ',  

1838:        &          '"$TALLY" and first "&"..'  

1839:        call CNTERR( 'FATAL' )  

1840:        NERR = NERR + 1  

1841:      end if  

1842: C  

1843:      write(IOWK,'(i8)') NR  

1844:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)  

1845: C  

1846:      call IRFILE( IDTAL(IDST+IDMOFF+1), NREG, IREGB,  

1847:                  &          INRK, NR )  

1848: C  

1849:      go to 540  

1850:    end if  

1851: C  

1852:      call IORDER( IDTAL(IDST+IDMOFF+1), NREG, MMM )  

1853:      IDTAL(IDST+2) = MMM  

1854:    end if  

1855: C  

1856: C <<<<<<<<<<<<  

1857: C  

1858: *      do 440 I = 1, NREG  

1859: *        IDTAL(IDST+IDMOFF+I) = 0  

1860: * 440 continue  

1861: C  

1862: *      IREGB = 0  

1863: C <<<<<<<<<<<<  

1864: * 450 call PCTCHK( IPK, 'REG', TYP, LPT, NDATA, IRET )  

1865: C <<<<<<<<<<<<  

1866: *      if ( TYP.ne.'LB' ) then  

1867: *        IREGB = IREGB + 1  

1868: C <<<<<<<<<<<<  

1869: *      call UNPKCS( IPK, 'REG', CWRK, NL, IRET )  

1870: C <<<<<<<<<<<<  

1871: *      write(IPR,'(1X,A,A)')  

1872: *      &          ' #', IREGB, ': ', CWRK(:NL)  

1873: C  

1874: *      call REGNM0( IR, '<', CWRK(:NL), NL0, A, CHA )  

1875: C  

1876: *      if ( IR.eq.0 ) then  

1877: *        write(IMGF,*) 'XXX($TALLY) no region that',  

1878: *        &          ' matches a name <', CWRK(:NL), '>'  

1879: *        call CNTERR( 'FATAL' )  

1880: *        NERR = NERR + 1  

1881: *      end if  

1882: C  

1883: *      IDTAL(IDST+IDMOFF+IR) = IREGB  

1884: *      IWRK(IREGB) = IR  

1885: C

```

[illegible]

src/shared/talin1.f

```

1951:      IREGB = IREGB + 1
1952: C <<<<<<<<<<<<
1953:      call UNPKCS( IPK, 'REG', CWRK, NL, IRET )
1954: C <<<<<<<<<<<<
1955:      write(IPR,'(1X,A,I4,A,A)') '      #', IREGB,
1956:      &      ': ', CWRK(:NL)
1957: C
1958:      call REGNM0( IR, '<', CWRK(:NL), NL0, A, CHA )
1959: C
1960:      if ( IR.eq.0 ) then
1961:      write(IMG,'(1X,A,A,A,A)')
1962:      &      'XXX(STALLY) no region that',
1963:      &      ' matches a name <', CWRK(:NL), '>'
1964:      call CNTERR( 'FATAL' )
1965:      NERR = NERR + 1
1966:      end if
1967: C
1968:      IDTAL(IDST+IDMOFF+IR) = 1
1969: C
1970: C ... register marker region
1971: C
1972:      if ( MKREG(IR,1).eq.0 ) then
1973:      NMKREG = NMKREG + 1
1974:      MKREG(IR,1) = NMKREG
1975:      MKREG(NMKREG,2) = IR
1976:      end if
1977: C
1978:      go to 560
1979:      end if
1980: C
1981:      end if
1982: C
1983:      write(IOWK,'(a6,i8)') 'MKRREG', 3
1984: C ... marker region #list
1985:      NR = 0
1986:      do 570 I = 1, NREG
1987:      if ( IDTAL(IDST+IDMOFF+I).eq.1 ) then
1988:      NR = NR + 1
1989:      IWRK(NR) = I
1990:      end if
1991: 570      continue
1992:      write(IOWK,'(i8)') NR
1993:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)
1994: C
1995: C ... non-marker region #list
1996:      NR = 0
1997:      do 580 I = 1, NREG
1998:      if ( IDTAL(IDST+IDMOFF+I).eq.2 ) then
1999:      NR = NR + 1
2000:      IWRK(NR) = I
2001:      end if
2002: 580      continue
2003:      write(IOWK,'(i8)') NR
2004:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)
2005: C
2006:      write(IOWK,'(i8)') NREG
2007:      write(IOWK,'(10i8)') (I,I=1,NREG)
2008: C
2009: C ==== spatial point ===
2010: C
2011:      else if ( TALDIM(K)(1:3).eq.'POI' ) then
2012: C
2013:      write(IPR,*) '      * POINT'
2014: C
2015:      IDTAL(IDST+1) = 9

```

```

2016: C
2017:      IDTAL(IDST+2) = 0
2018:      IDTAL(IDST+3) = NPDET
2019: C
2020: C##<2007/03/14:PN3:
2021: C##      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 999
2022:      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 9002
2023: C##>
2024: C
2025: C <<<<<<<<<<<<
2026:      PLAB = TAG(:LTAG) //'POINT'
2027:      call PCTLB2( IPK, 'POINT', PLAB(:ICLEN2(PLAB)), IRET )
2028: C <<<<<<<<<<<<
2029: C
2030:      if ( IRET.eq.0 ) then
2031:      call UNPKCS( IPK, 'POINT', CWRK, NL, IRET )
2032: C <<<<<<<<<<<<
2033:      call UNPKND( IPK, 'POINT', 'I4', IWRK, NWORK, ND, IRET
2034:      &      )
2035: C <<<<<<<<<<<<
2036: C
2037:      IDTAL(IDST+2) = ND
2038: C
2039: C ... compose point tally bin table ...
2040: C
2041:      do 610 I = 1, NPDET
2042:      IDTAL(IDST+IDMOFF+I) = 0
2043:      do 590 J = 1, ND
2044:      if ( IWRK(J).eq.I ) then
2045:      IDTAL(IDST+IDMOFF+I) = J
2046:      go to 600
2047:      end if
2048: 590      continue
2049: 600      continue
2050: 610      continue
2051: C
2052: C      write(IPR,*) '      POINT DETECTOR -> TALLIED BIN'
2053: C      write(IPR,'(1x,' ' ',10i5)')
2054: C      &      (IDTAL(IDST+IDMOFF+J),J=1,NPDET)
2055: C      write(IPR,*) '      POINT DETECTOR # IN THIS TALLY'
2056: C      write(IPR,'(1x,' ' ',10i5)')
2057: C      &      (IWRK(J),J=1,NPDET)
2058: C      end if
2059: C
2060:      write(IOWK,'(a6,i8)') 'POINT ', ND
2061:      do 620 J = 1, ND
2062:      NR = 1
2063:      write(IOWK,'(i8)') NR
2064:      write(IOWK,'(10i8)') IWRK(J)
2065: 620      continue
2066: C##<2007/03/14:PN3:
2067: C
2068: C ==== produce region ===
2069: C
2070:      else if ( IMATCH('PRO*-REG*',TALDIM(K)).eq.1 ) then
2071:      write(IPR,'(1X,A)') '      * PRODUCE REGION'
2072:      IDTAL(IDST+1) = 10
2073:      IDTAL(IDST+2) = 0
2074:      IDTAL(IDST+3) = NREG
2075:      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 9002
2076:      PLAB = TAG(:LTAG) //'PRODUCE-REGION'
2077:      call PCTLB2( IPK, 'PRODUCE-REGION', PLAB(:ICLEN2(PLAB)),
2078:      &      IRET )
2079:      if ( IRET.eq.0 ) then
2080: C ... number of specified region-sets ...

```

src/shared/talin1.f

```

2081:      PLAB = TAG(:LTAG) //'END-REGION'
2082:      call PCTLB( IPK, 'REGION', PLAB(:ICLEN2(PLAB)), IRET )
2083:      call UNPKND( IPK, 'NUM', 'I4', NRG, 1, NDN, IRET )
2084:      PLAB = TAG(:LTAG) //'PRODUCE-REGION'
2085:      call PCTLB2( IPK, 'PRODUCE-REGION',
2086:      &          PLAB(:ICLEN2(PLAB)), IRET )
2087:      if ( IDST+IDMOFF+NREG.gt.NLTEMP ) go to 9002
2088:      write(IOWK,'(A6,I8)') 'PROREG', NRG
2089: C
2090:      IREGB = 0
2091: 710 call PCTCHK( IPK, 'DIM', TYP, LPT, NDATA, IRET )
2092:      if ( TYP.ne.'LB' ) then
2093:      IREGB = IREGB + 1
2094:      call UNPKCS( IPK, 'REG', CWRK, NL, IRET )
2095:      write(IPR,'(1X,A,I6,A,A)') ' ', IREGB,
2096:      &          ' ', CWRK(:NL)
2097:      call REGLST( CWRK(:NL), IWRK, NWORK, NR, NR2, A,
2098:      &          CHA, IERR )
2099:      if ( NR.eq.0 ) then
2100:      write(IMG,9101) CWRK(:NL)
2101:      call CNTERR( 'FATAL' )
2102:      NERR = NERR + 1
2103:      else if ( NR2.gt.NR ) then
2104:      write(IMG,9102) CWRK(:NL)
2105:      write(IMG,9103) NR2
2106:      call CNTERR( 'FATAL' )
2107:      NERR = NERR + 1
2108:      end if
2109:      write(IOWK,'(i8)') NR
2110:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)
2111:      call IBTILE( IDTAL(IDST+IDMOFF+1), NREG, IREGB,
2112:      &          IWRK, NR )
2113:      go to 710
2114:      end if
2115:      call IORDER( IDTAL(IDST+IDMOFF+1), NREG, MMM
2116:      IDTAL(IDST+2) = MMM
2117:      end if
2118: C
2119: C
2120: C
2121:      else if ( IMATCH('PRO*-NUC*',TALDIM(K)).eq.1 ) then
2122:      write(IPR,'(1X,A)') ' ' * PRODUCE NUCLIDE'
2123:      NC = NUC + NPATOM + NUCPN
2124:      IDTAL(IDST+1) = 11
2125:      IDTAL(IDST+2) = 0
2126:      IDTAL(IDST+3) = NC
2127:      if ( IDST+IDMOFF+IDTAL(IDST+3).gt.NLTEMP ) go to 9002
2128:      PLAB = TAG(:LTAG) //'PRODUCE-NUCLIDE'
2129:      call PCTLB2( IPK, 'PRODUCE-NUCLIDE', PLAB(:ICLEN2(PLAB)),
2130:      &          IRET )
2131:      if ( IRET.eq.0 ) then
2132: C
2133:      ... number of specified nuclide-name sets ...
2134:      PLAB = TAG(:LTAG) //'END-NUCLIDE'
2135:      call PCTLB( IPK, 'NUCLIDE', PLAB(:ICLEN2(PLAB)), IRET )
2136:      call UNPKND( IPK, 'NUM', 'I4', NNC, 1, NDN, IRET )
2137:      PLAB = TAG(:LTAG) //'PRODUCE-NUCLIDE'
2138:      call PCTLB2( IPK, 'PRODUCE-NUCLIDE',
2139:      &          PLAB(:ICLEN2(PLAB)), IRET )
2140:      write(IOWK,'(A6,I8)') 'PRONUC', NNC
2141:      INUCB = 0
2142: 720 call PCTCHK( IPK, 'DIM', TYP, LPT, NDATA, IRET )
2143:      if ( TYP.ne.'LB' ) then
2144:      INUCB = INUCB + 1
2145:      call UNPKCS( IPK, 'NUC', CWRK, NL, IRET )
2146:      write(IPR,'(1X,A,I6,A,A)') ' ', INUCB,
2147:      &          ' ', CWRK(:NL)
2148:      call NUCLST( CWRK(:NL), IWRK, NWORK, NR, A, CHA,
2149:      &          IERR )
2150:      if ( NR.eq.0 ) then
2151:      write(IMG,9104) CWRK(:NL)
2152:      call CNTERR( 'FATAL' )
2153:      NERR = NERR + 1
2154:      else if ( IERR.ne.0 ) then
2155:      write(IMG,9106) CWRK(:NL), NWORK
2156:      call CNTERR( 'FATAL' )
2157:      NERR = NERR + 1
2158:      end if
2159:      write(IOWK,'(i8)') NR
2160:      write(IOWK,'(10i8)') (IWRK(I),I=1,NR)
2161:      call IBTILE( IDTAL(IDST+IDMOFF+1), NC, INUCB,
2162:      &          IWRK, NR )
2163:      go to 720
2164:      end if
2165:      call IORDER( IDTAL(IDST+IDMOFF+1), NC, MMM )
2166:      IDTAL(IDST+2) = MMM
2167: C
2168: C
2169: C
2170:      else if ( IMATCH('PRO*-REA*',TALDIM(K)).eq.1 ) then
2171:      write(IPR,'(1X,A)') ' ' * PRODUCE REACTION'
2172:      NRT = 0
2173:      I1 = 0
2174:      I2 = 0
2175:      I3 = 0
2176:      I4 = 0
2177:      I5 = 0
2178:      I6 = 0
2179:      if ( JNEUT.ne.0 ) then
2180:      NRT = NRT + 120 ! neutron library: NMT=120
2181:      I1 = 1
2182:      I2 = 120
2183:      end if
2184:      if ( JPHOT.ne.0 ) then
2185:      NRT = NRT + 60 ! photon library: NMTP=60
2186:      I3 = I1 + I2
2187:      I4 = 60
2188:      end if
2189:      if ( JPHNU.ne.0 ) then
2190:      NRT = NRT + 135 ! photo-nuclear library: NMTPN
2191:      I5 = I3 + I4
2192:      I6 = 135
2193:      end if
2194:      IDTAL(IDST+1) = 12
2195:      IDTAL(IDST+2) = 0
2196:      IDTAL(IDST+3) = NRT
2197:      if ( IDST+IDMOFF+NRT.gt.NLTEMP ) go to 9002
2198:      PLAB = TAG(:LTAG) //'PRODUCE-REACTION'
2199:      call PCTLB2( IPK, 'PRODUCE-REACTION',
2200:      &          PLAB(:ICLEN2(PLAB)), IRET )
2201:      if ( IRET.eq.0 ) then
2202:      call UNPKCS( IPK, 'PRODUCE-REACTION', CWRK, NL, IRET )
2203:      call UNPKND( IPK, 'PRR', 'I4', IWRK, NRT, ND, IRET )
2204:      MX = 0
2205:      do I = 1, ND
2206:      if ( IWRK(I).lt.0 ) then
2207:      write(IMG,9105) 'reaction', 'PRODUCE-REACTION'
2208:      call CNTERR( 'FATAL' )
2209:      NERR = NERR + 1

```

```

      if ( I1.gt.0 )
        write(IPR,'(1x,'' neutron: ''',10i5)')
          (IDTAL(IDST+IDMOFF+J),J=I1,I1+I2-1)
      if ( I3.gt.0 )
        write(IPR,'(1x,'' photon: ''',10i5)')
          (IDTAL(IDST+IDMOFF+J),J=I3,I3+I4-1)
      if ( I5.gt.0 )
        write(IPR,'(1x,'' photonuclear:''',10i5)')
          (IDTAL(IDST+IDMOFF+J),J=I5,I5+I6-1)
      write(IOWK,'(A6,I8)') 'PROREA',MX
      do I = 1, MX
        NR = 0
        do J = 1, NRT
          if ( IDTAL(IDST+IDMOFF+J).eq.1 ) then
            NR = NR + 1
            IWRK(NR) = J
          end if
        end do
        write(IOWK,'(I8)') NR
        write(IOWK,'(10I8)') (IWRK(J),J=1,NR)
      end do
    end if

==== INVALID DIMENSION ====

      else
        write(IMGF,*) 'XXX($TALLY) DIMENSION is invalid < ',
          TALDIM(K), ' >'
        NERR = NERR + 1
        call CNTERR( 'FATAL' )

      end if

... if no bin is specified, tallied in one bin ...

      if ( IDTAL(IDST+2).eq.0 ) then
        write(IMGF,*) '!!!($TALLY) No bin is specified,',
          ' dimension size is set to 1.'
        call CNTERR( 'WARNING' )
        IDTAL(IDST+2) = 1
        do 630 I = 1, IDTAL(IDST+3)

```

```

2275:                write(IOWK,'(i8)') NR
2276:                end if
2277:                write(IOWK,'(l0i8)' ) (I,I=1,NR)
2278: C
2279:                end if
2280: C
2281: C    ... if no bin is specified, dimension size is set to 1...
2282: C
2283: C            if( IDTAL(IDST+2).eq.0 ) then
2284: C                IDTAL(IDST+2) = 1
2285: C                IDTAL(IDST+3) = 0
2286: C                write(IOWK,'(A6,i8)') 'DUM   ',1
2287: C                write(IOWK,'(i8)') 1
2288: C                write(IOWK,'(l0i8)') 1
2289: C            endif
2290: C
2291:                IDSIZ  = IDSIZ*IDTAL(IDST+2)
2292: C
2293: C        =====
2294: C
2295:                IDST      = IDST + IDMOFF + IDTAL(IDST+3)
2296: C        640          continue
2297: C
2298: C    ... ID of custom (user defined) tally ...
2299: C
2300:                PLAB      = TAG(:LTAG) //'USER'
2301: C
2302: C <<<<<<<<<<<<
2303: C        call PCTLB2( IPK, 'USER', PLAB(:ICLEN2(PLAB)), IRET )
2304: C <<<<<<<<<<<<
2305: C
2306: C            if ( IRET.eq.0 ) then
2307: C <<<<<<<<<<<<
2308: C                NDATA      = 1
2309: C                call UNPKND( IPK, 'IUSER', 'I4', IUSER, NDATA, NDATA2, IRET
2310: C                &
2311: C <<<<<<<<<<<<
2312: C                IDTAL(IDT+10) = IUSER
2313: C                write(IPR,'(lX,A,i6)') '* USER DEFINED TALLY # : ',
2314: C                &
2315: C                    IDTAL(IDT+10)
2316: C            else
2317: C                IDTAL(IDT+10) = 0
2318: C            end if
2319: C
2320: CCCCC in current version, d-tally number & e-tally number is identical
2321: C ... d-tally number & e-tally number is *not* identical after
2322: C v2.0 beta4.11test (modification taken into v2.0beta4.13)
2323: C
2324: C                NDTALY = NDTALY + 1
2325: C                NETALY = NETALY + 1
2326: C                KE2D(IT) = NDTALY
2327: C
2328: C    ... increase d-tallies for this e-tally ...
2329: C
2330: C            if ( JADD.ne.0 ) then
2331: C ... at the present , only flux tally is added
2332: C
2333: C                NDT      = IDST - IDT
2334: C c##<2007/03/14:PN3:
2335: C                if ( IDST+NDT.gt.NLTEMP ) go to 999
2336: C                if ( IDST+NDT.gt.NLTEMP ) go to 9002
2337: C c##>
2338: C
2339: C                do 650 I = 1, NDT
2340: C                    IDTAL(IDST+I) = IDTAL(IDT+I)

```

```

2275:                write(IOWK,'(i8)') NR
2276:            end if
2277:            write(IOWK,'(l0i8)' ) (I,I=1,NR)
2278: C
2279:            end if
2280: C
2281: C    ... if no bin is specified, dimension size is set to 1...
2282: C
2283: C        if( IDTAL(IDST+2).eq.0 ) then
2284: C            IDTAL(IDST+2) = 1
2285: C            IDTAL(IDST+3) = 0
2286: C            write(IOWK,'(A6,i8)') 'DUM   ',1
2287: C            write(IOWK,'(i8)') 1
2288: C            write(IOWK,'(l0i8)') 1
2289: C        endif
2290: C
2291:            IDSIZ  = IDSIZ*IDTAL(IDST+2)
2292: C
2293: C        =====
2294: C
2295:            IDST      = IDST + IDMOFF + IDTAL(IDST+3)
2296: C        640      continue
2297: C
2298: C    ... ID of custom (user defined) tally ...
2299: C
2300:            PLAB      = TAG(:LTAG) //'USER'
2301: C
2302: C <<<<<<<<<<<<
2303: C        call PCTLB2( IPK, 'USER', PLAB(:ICLEN2(PLAB)), IRET )
2304: C <<<<<<<<<<<<
2305: C
2306: C        if ( IRET.eq.0 ) then
2307: C <<<<<<<<<<<<
2308: C            NDATA      = 1
2309: C            call UNPKND( IPK, 'IUSER', 'I4', IUSER, NDATA, NDATA2, IRET
2310: C            &          )
2311: C <<<<<<<<<<<<
2312: C            IDTAL(IDT+10) = IUSER
2313: C            write(IPR,'(lX,A,i6)') '* USER DEFINED TALLY # : ',
2314: C            &          IDTAL(IDT+10)
2315: C        else
2316: C            IDTAL(IDT+10) = 0
2317: C        end if
2318: C
2319: CCCCC in current version, d-tally number & e-tally number is identical
2320: C ... d-tally number & e-tally number is *not* identical after
2321: C v2.0 beta4.11test (modification taken into v2.0beta4.13)
2322: C
2323: C            NDTALY     = NDTALY + 1
2324: C CCCC      NETALY      = NETALY + 1
2325: C            KE2D(IT)    = NDTALY
2326: C
2327: C    ... increase d-tallies for this e-tally ...
2328: C
2329: C        if ( JADD.ne.0 ) then
2330: C    ... at the present , only flux tally is added
2331: C
2332: C            NDT         = IDST - IDT
2333: C c##<2007/03/14:PN3:
2334: C c##           if ( IDST+NDT.gt.NLTEMP ) go to 999
2335: C             if ( IDST+NDT.gt.NLTEMP ) go to 9002
2336: C c##>
2337: C
2338: C        do 650 I = 1, NDT
2339: C            IDTAL(IDST+I) = IDTAL(IDT+I)

```

src/shared/talin1.f

```

2340: 650      continue
2341: C
2342:         IDT      = IDST
2343:         KKDT      = KKDT + IDSIZ
2344:         IDTAL(IDT+3) = KKDT
2345:         IDTAL(IDT+7) = 0
2346:         IDTAL(IDT+9) = NDTALY + 1
2347:         IDTAL(IDT+9) = -IDTAL(IDT+9)
2348: C
2349:         NDTALY = NDTALY + 1
2350:         IDST = IDT + NDT
2351:       end if
2352: C
2353:         IDT      = IDST
2354:         KKDT      = KKDT + IDSIZ
2355: 660 continue
2356: C
2357:         NIDTAL = IDT
2358:         NETALY = ITAL
2359: C
2360: C ... set NNCST and NNCSI ...
2361: C
2362:       do 690 K = 1, NPKIND
2363:         NNM = 0
2364:         do 670 IN = 1, NUC
2365:           if ( INCST(IN,K).ne.0 ) NNM = NNM + 1
2366: 670       continue
2367:         NNCST = MAX(NNCST,NNM)
2368:         if ( JJPNT.ne.0 ) then
2369:           NNMI = 0
2370:           do 680 IN = 1, NUC
2371:             if ( INCSI(IN,K).ne.0 ) NNMI = NNMI + 1
2372: 680       continue
2373:           NNCSI = MAX(NNCSI,NNMI)
2374:         end if
2375: 690 continue
2376: c##<2007/03/14:PN3:
2377:         NNM = 0
2378:         do IT = 1, ITAL
2379:           if ( INCSTPN(IT,1).ne.0 ) NNM = NNM + 1
2380:         enddo
2381:         NNCSTPN = NNM
2382: c##>
2383: C
2384:       if ( NTSRF.gt.0.and.JTSRF.eq.0 ) then
2385:         write(IMG,'(1X,A,A)')
2386:         & '!!!($TALLY) TALLY-SURFACE is input, but no surface-tally',
2387:         & ' is used.'
2388:         call CNTERR( 'WARNING' )
2389:       end if
2390:
2391: C ***** debug *****
2392:       if ( JDEBG(1).ne.0 ) then
2393:         write(IPR,*) 'check IDTAL : ', NIDTAL
2394:         write(IPR,'(1X,10i6)') (IDTAL(I),I=1,NIDTAL)
2395:         write(IPR,*) 'check NMKREG : ', NMKREG
2396:         write(IPR,*) 'MKREG(*,1) : ', (MKREG(I,1),I=1,NREG)
2397:         write(IPR,*) 'MKREG(*,2) : ', (MKREG(I,2),I=1,NMKREG)
2398:         write(IPR,*) 'NNCST : ', NNCST
2399:         write(IPR,*) 'NTSRF : ', NTSRF
2400:         write(IPR,*) 'IDSRF : ', IDSRF
2401:         write(IPR,*) 'NANGLE : ', NANGLE
2402:         do 700 K = 1, NPKIND
2403:           write(IPR,*) ' particle ', K, ' INCST ',
2404:           & (INCST(IN,K),IN=1,NUC)

```

```

2405: 700      continue
2406:       end if
2407: C *****
2408:
2409:       return
2410: C
2411: C ===== error messages ===
2412: C
2413: 666 continue
2414:       write(IMG,7000) PLAB(:ICLEN2(PLAB))
2415: 7000 format(/1X,'XXX($TALLY) program error) CANNOT FIND STORED DATA <',
2416:         & A,'>')
2417:       call PRSTOP( 1, 'program error in $TALLY block processing.' )
2418:       stop 666
2419: c##<2007/03/14:PN3:
2420: 9101 format(/' XXX($TALLY)(TALIN1) No region that matches a name <',
2421:         & A,'>')
2422: 9102 format(/' XXX($TALLY)(TALIN1) Too many regions that matche name',
2423:         & ' <', A,'>')
2424: 9103 format(' Please specify "NWORK" data at least greater than ', I7,
2425:         & ' (place it between "$TALLY" and first "&").')
2426: 9104 format(/' XXX($TALLY)(TALIN1) No nuclide that matches a name <',
2427:         & A,'>')
2428: 9105 format(/' XXX($TALLY)(TALIN1) ', A, 'specification in type "', A,
2429:         & '()' is negative.')
2430: 9106 format(/' XXX(talin1) Too many nuclides that matche name <',A,
2431:         & '>; NWORK=',i8)
2432: 9118 format(/' XXX(talin1) particle type must be photon for "MICROPN".'
2433:         & )
2434: 9119 format(/' XXX(talin1) photonuclear nuclide-ID <',a,'> in ',
2435:         & ' "MICROPN:" is invalid or not specified in the material',
2436:         & ' composition block.')
2437: 9120 format(/' XXX(talin1) tally mode of "MICROPN" is invalid : <',a,
2438:         & '>')
2439: 9121 format(/' XXX(talin1) number of a "MICROPN" reactions <',i5,
2440:         & '> is exceeded 100.')
2441: c##>
2442: C
2443: c##<2007/03/14:PN3:
2444: c#999 write(IMG,*) 'XXX($TALLY) Insufficient memory to compose ',
2445: c## & 'special tally infomation.'
2446: 9001 write(IMG,7001) PLAB(:ICLEN2(PLAB))
2447: 7001 format(/' XXX($TALLY)(TALIN1) Cannot find stored data <',
2448:         & A,'>')
2449:       call PRSTOP( 1, 'program error in $TALLY block processing.' )
2450:       stop 666
2451: 9002 write(IMG,7002)
2452: 7002 format(/' XXX($TALLY)(TALIN1) Insufficient memory to compose',
2453:         & ' special tally information.')
2454: c##>
2455:       call PRSTOP( 1, 'Insufficient memory to process $TALLY block.' )
2456:       stop 999
2457: C
2458:       end

```

src/shared/talin2.f

```

1:      subroutine TALIN2( CODE, IOWK, IOWK2, NERR, IPK, RPK, NPS,ITAL,
2:      &                  IETAL, NLTEMP,NIETAL,KETAL,JDTRG,NREG,
3:      &                  NDTALY,NETALY,NEDTMX,NDTWMX,NBINMX,MKRGBN,
4:      &                  KDTAL, IDTAL, NIDTAL, NLDOTAL, NLETAL,
5:      &                  NETRV, METRV, IETRV, JPUSD,
6:      F                  JNEUT, JPHOT, JTIME, JFISS, JEIGN, JRESP,
7:      F                  JTLLT, JDEBG,
8:      W                  IWRK,   RWRK,   DWRK,   NWORK,
9:      O                  KE2D, KIDT )
10: C=<MVP/GMVP>=====
11: C  purpose: process "special" tally specifications - Stage 2;
12: C           compose IETAL data etc.
13: C  called in: talinp routine of mvp/gmvp
14: C=====
15: C  arguments ( i=input, o=output, w=work )
16: C  i code   : name of code using this routine ('MVP'/'GMVP')
17: C  i iowk   : I/O unit number of working file (text) from which stored
18: C             data in preceding process are input.
19: C  i iowk2  : I/O unit number of working file (text) used in this
20: C             procedure.
21: C  o nerr   : number of errors
22: C  i ipk    : data packet container in which $TALLY block is stored.
23: C  i rpk    : equivalent to IPK but in REAL type.
24: C  i nps    : size of "IPK" in words
25: C  i ital   : number of special tally entry in $TALLY input block.
26: C  o ietal  : edited tally information.
27: C  i nltemp : size of memory available to IETAL ( actual size of IETAL
28: C             is determined in this routine.
29: C  o nietal : actual size of IETAL array. not set in this routine.
30: C  o ketal  : pointer to ETALY(*) array for each edited tally.
31: C  o jdtrg  : flag to show each region needs specified d-tally.
32: C  i nreg   : total number of region
33: C  i ndtaly : total number of direct-tally.
34: C  i netaly : total number of edited-tally.
35: C  o nedtmx : maximum length of edited tally
36: C  o kdatal : pointer to DTALY(*) array for each direct tally.
37: C  i idtal  : direct tally information.
38: C  i nidtal : actual size of IDTAL array.
39: C  o NLDOTAL : size of DTALY array
40: C  o NletAL : size of ETALY array
41: C  i NETRV  : Number of special (edited) tallies whose "real variance"'s
42: C             are estimated.
43: C  o METRV  : Total length of special (edited) tallies whose
44: C             "real variance"'s are estimated. (per batch length)
45: C  o IETRV(2,NETRV+1) : Pointers for special (edited) tallies whose
46: C             "real variance"'s are estimated.
47: C             IETRV(1,*) : edited tally #.
48: C             IETRV(2,*) : tally is stored on
49: C                   ETRV(IETRV(2,i):IETRV(2,i+1)-1,batch).
50: C  o JPUSD  : flag to show point detector really used.
51: C  i JNEUT,JPHOT,JTIME,JFISS,JEIGN,JRESP,JTLLT,JDEBG : option flags
52: C  i KE2D  : reference of i-th ETALY to DTALY ( used for data processing)
53: C             if negative, tally-ID is stored.
54: C  o KIDT  : starting position for each d-tally in IDTAL
55: C  o ndtwmx: maximum size of work area to peocess d-tally to e-tally
56: C             except NEDTMX.
57: C  w iwrk   : working array (integer)
58: C  w rwrk   : working array (real)
59: C  w dwrk   : working array (real*8)
60: C  i nwork  : working array size in word.
61: C
62: C (iwr, rwrk & dwrk have the same address)
63: C
64: C-----
65: C

```

```

66: C Structure of IDTAL(*)
67: C
68: C +----- direct tally loop ( NDTALY )
69: C |
70: C | (1) ID
71: C | (2) (dummy for debug: -999)
72: C | (3) pointer to DTALY(*)
73: C | (4) event #
74: C | (5) particle type
75: C | (6) number of dimensions
76: C | (7) multiplication factor type
77: C | (8) nuclide/atom # & reaction #
78: C | (9) D-tally #
79: C | (10) custom tally # (non-zero means user customized treatment)
80: C | (11) detector # (for surface event positive tally-surface #
81: C |             means particle current tally, and negative one
82: C |             means flux tally)
83: C | ( IDTOFF data so far )
84: C
85: C +----- tally dimension loop
86: C | * dimension type
87: C | * number of tallied bins
88: C | * size of common bin -> tallied bin conversion table
89: C | ( IDMOFF data so far )
90: C | +----- common bin loop
91: C | | tallied bin # for each common bin
92: C | +-----
93: C | +-----
94: C +-----
95: C
96: C
97: C Structure of IETAL(*)
98: C
99: C +----- edited tally loop (NETALY)
100: C | label: TALLY.# ( '#' is edited tally # increasing from 1)
101: C | integer:
102: C | (1) tally ID
103: C | (2) particle type
104: C | (3) methed to synthesize direct-tallies
105: C |         0 = no combination
106: C |         1 = weighted sum
107: C |         2 = multiplication / division
108: C |         3 = general expression
109: C |         4 = Feynman-Y values
110: C | (4) number of tally dimensions
111: C | (5) starting address of this tally in etaly(*)
112: C | (6) event #
113: C | (7) weighting function
114: C | (8) nuclide/atom # & reaction #
115: C | (9) <unused>
116: C | (10) custom tally # (non-zero means user customized treatment)
117: C | string: label string
118: C | +----- tally-dimension loop
119: C | | string: "dimension-label" ('ENERGY', 'TIME' etc.)
120: C | +-----
121: C | label: DTALLY.#
122: C | integer: (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[1],
123: C |         (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[2],
124: C |         ...
125: C | ( data to synthesize d-tallies : not available currently )
126: C
127: C | label: DIMENSION.#
128: C | +----- tally-dimension loop
129: C | | integer: size of each dimension(number of bins) in ETALY
130: C | | REGION ---> negative, others ---> positive

```

src/shared/talin2.f

```

131: C | integer:
132: C | .....E-tally-bin loop (in one packet)
133: C | . table length,
134: C | . D-tally -> E-tally translation table.
135: C | . (list of d-tally bins composing this e-tally bin)
136: C | .....
137: C | integer:
138: C | .....E-tally-bin loop (in one packet)
139: C | . table length,
140: C | . Original-tally-bin -> E-tally-bin translation table.
141: C | . (list of original-tally bins composing this e-tally bin)
142: C | .....
143: C | +----- E-tally-bin loop (when this dimension is REGION)
144: C | | string: "region name"
145: C | +-----
146: C | +-----
147: C | +-----
148: C |-----
149: C
150: C character*(*) CODE
151: C integer IPK(NPS)
152: C real RPK(NPS)
153: C
154: C integer KETAL(NETALY)
155: C integer IETAL(NLTEMP)
156: C
157: C integer JDTRG(NREG,NDTALY)
158: C integer JPUSD(NPDET)
159: C
160: C integer KDTAL(NDTALY+1)
161: C integer IDTAL(NIDTAL)
162: C
163: C integer IETRV(2,NETRV+1)
164: C
165: C integer JDEBG(*)
166: C
167: C integer KE2D(ITAL), KIDT(NDTALY)
168: C
169: C integer IWRK(NWORK)
170: C real RWRK(NWORK)
171: C real*8 DWRK(*)
172: C
173: C ... offset of direct-tally data in IDTAL(*)
174: C
175: CC parameter( IDTOFF = 12 )
176: C
177: C ... offset of tally dimension dependent data in IDTAL(*)
178: C
179: CC parameter( IDMOFF = 3 )
180: C
181: C include 'INC/_IDXOFF'
182: C
183: CCCCCinclude 'INC/_ARRAY'
184: Ccccc include 'INC/_FLAGS'
185: CCCCCinclude 'INC/_SIZES'
186: C include 'INC/_PTALY0'
187: C include 'INC/_PTLSP'
188: CCCCCinclude 'INC/_STALY'
189: C include 'INC/_IOUNIT'
190: C
191: C ===== local variables =====
192: C
193: C character*9 TAG
194: C character*9 TAG2
195: C character*12 THEAD

```

```

196: C character*6 BINTYP
197: C integer ID
198: C
199: C character*128 CWRK
200: C character*32 PLAB
201: C character*32 PLAB2
202: C
203: C-----
204: C
205: C JNP = JNEUT*JPHOT
206: C call RWIND( IOWK )
207: C
208: C ... make KDTAL (a sample of access method to IDTAL array !!)
209: C and calculate total size of DTALY array (NLDTAL)
210: C
211: C NLDTAL = 0
212: C IDT = 0
213: C do 110 IT = 1, NDTALY
214: C
215: C KIDT(IT) = IDT + 1
216: C
217: C if ( IDTAL(IDT+2).ne.-999 ) then
218: C write(IMG*,*) 'XXX(TALIN2) Failed to make direct tally',
219: C & ' pointer. Direct tally information array(IDTAL) is'
220: C & ' corrupt?'
221: C write(IMG*,*) ' D-TALLY ', IT, ' IDTAL ',
222: C & (IDTAL(IDT+1),I=1,IDTOFF)
223: C call PRSTOP( 1, 'Program error in TALIN2 routine (1).')
224: C stop 666
225: C end if
226: C
227: C KDTAL(IT) = IDTAL(IDT+3)
228: C
229: C IDT1 = IDT + IDTOFF
230: C
231: C IDSIZ = 1
232: C do 100 J = 1, IDTAL(IDT+6)
233: C IDSIZ = IDSIZ*IDTAL(IDT1+2)
234: C IDT1 = IDT1 + IDMOFF + IDTAL(IDT1+3)
235: C 100 continue
236: C IDT = IDT1
237: C
238: C NLDTAL = NLDTAL + IDSIZ
239: C 110 continue
240: C
241: C ... preset JDTRG ...
242: C
243: C do 130 J = 1, NDTALY
244: C do 120 I = 1, NREG
245: C JDTRG(I,J) = 0
246: C 120 continue
247: C 130 continue
248: C
249: C-----
250: C start processing of each "& ..." input
251: C-----
252: C
253: C NERR = 0
254: C
255: C NEDTMX = 0
256: C NDTWMX = 0
257: C KKET = 1
258: C IDT = 0
259: C MXRGBN = 0
260: C

```

[illegible][illegible]

```

337:      call PCTLB2( IPK, 'ID', PLAB(:ICLEN2(PLAB)), IRET )
338:      if ( IRET.eq.0 ) then
339:         call UNPKCS( IPK, 'WHAT', CWRK, NL, IRET )
340:         if ( CWRK(:NL).eq.'FEYNMAN-Y' ) then
341:            ISYN = 4
342:            call UNPKND( IPK, 'NGATE', 'I4', IWRK, 2, ND, IRET )
343:            NGATE = IWRK(1)
344:        end if
345:    end if
346: C
347:     IDSIZ = 1
348:     IESIZ = 1
349: C
350: C ... particle,how to compose , number of dimension ...
351: C
352: CCCCC   IWRK(1) = IDTAL(IDT+1)
353: CCCCC   IWRK(1) = ID
354: CCCCC   IWRK(2) = IDTAL(IDT+5)
355: C
356: CCCCCCCCCC currently no multi d-tally feature
357: C ... currently multi d-tally feature
358: CCCCCCC   IWRK(3) = 0
359: CCCCCCC   IWRK(3) = ISYN
360: C
361: CCCCCCC   IWRK(4) = IDTAL(IDT+6)
362: CCCCCCC   IWRK(5) = KKET
363: CCCCCCC   IWRK(6) = IDTAL(IDT+4)
364: CCCCCCC   IWRK(7) = IDTAL(IDT+7)
365: CCCCCCC   IWRK(8) = IDTAL(IDT+8)
366: CCCCCCC   IWRK(9) = 0
367: CCCCCCC   IWRK(10) = IDTAL(IDT+10)
368: C
369: CCCCCCC   KETAL(IT) = KKET
370: C
371: C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
372: Ccccccc call PACKND( IETAL, 'datal-3', 'I4', IWRK, 8, IRET2 )
373: Ccccccc call PACKND( IETAL, 'datal-3', 'I4', IWRK, 10, IRET2 )
374: C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
375: C
376: C ... LABEL ...
377: C
378: CCCCCCC   PLAB = TAG(:LTAG) //'LABEL'
379: CCCCCCC   call PCTLB2( IPK, 'label', PLAB(:ICLEN2(PLAB)), IRET )
380: CCCCCCC   if ( IRET.eq.0 ) then
381: CCCCCCC       call UNPKCS( IPK, 'LABEL', CWRK, NL, IRET )
382: CCCCCCC   else
383: CCCCCCC       CWRK = 'SPECIAL TALLY' //TAG2(:LTAG2)
384: CCCCCCC       NL = ICLEN2(CWRK)
385: CCCCCCC   end if
386: C
387: C ... real variance flag ...
388: C
389: CCCCCCC   JRV = 0
390: CCCCCCC   if ( NETRV.gt.0 ) then

```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
call PACKCS( IETAL, 'LABEL', CWRK(:NL), IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

dimension label ...

IDT1      = IDT + IDTOFF
do 160 I = 1, IDTAL(IDT+6)
    if ( IDTAL(IDT1+1).eq.1 ) then
        CWRK      = 'REGION'
    else if ( IDTAL(IDT1+1).eq.2 ) then
        CWRK      = 'ENERGY'
    else if ( IDTAL(IDT1+1).eq.3 ) then
        CWRK      = 'TIME'
    else if ( IDTAL(IDT1+1).eq.4 ) then
        CWRK      = 'ANGLE'
    else if ( IDTAL(IDT1+1).eq.5 ) then
        CWRK      = 'GENERATION'
    else if ( IDTAL(IDT1+1).eq.6 ) then
        CWRK      = 'SOURCE'
    else if ( IDTAL(IDT1+1).eq.7 ) then
        CWRK      = 'SOURCE-REGION'
    else if ( IDTAL(IDT1+1).eq.8 ) then
        CWRK      = 'MARKER-REGION'
    else if ( IDTAL(IDT1+1).eq.9 ) then
        CWRK      = 'POINT'

007/03/14:PN3:
    else if ( IDTAL(IDT1+1).eq.10 ) then
        CWRK      = 'PRODUCE-REGION'
    else if ( IDTAL(IDT1+1).eq.11 ) then
        CWRK      = 'PRODUCE-NUCLIDE'
    else if ( IDTAL(IDT1+1).eq.12 ) then
        CWRK      = 'PRODUCE-REACTION'

    end if
    NL          = ICLEN2(CWRK)

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
call PACKCS( IETAL, 'LABEL', CWRK(:NL), IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

IDT1      = IDT1 + IDMOFF + IDTAL(IDT1+3)
continue
```

```

566: C
575: C      IWRK(1) = IT
584: C      IWRK(2) = IDT + 1
593: C      IWRK(3) = KDTAL(IT)
602: C      if ( ISYN.eq.0 .or. ISYN.eq.4 ) then
611: C          NDATA = 3
620: C          IWRK(1) = IDTN
629: C          IWRK(2) = KIDT(IDTN)
638: C          IWRK(3) = KDTAL(IDTN)
647: C      else
656: C          write(IMG*,*) 'XXX(TALIN2) not defined type of symthesis'
665: C          call PRSTOP( 1, 'Program error in TALIN2 routine (4).')
674: C          stop 666
683: C      end if
692: C
701: C
710: C      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
720: C      CCCCCC call PACKND( IETAL, 'datal-3', 'I4', IWRK, 3, IRET2 )
729: C      call PACKND( IETAL, 'datal-3', 'I4', IWRK, NDATA, IRET2 )
738: C      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
747: C
756: C      ===== dimension =====
765: C
774: C          CWRK = ' '
783: C          write(CWRK(1:13),'(a,i8)' ) 'TALLY', IDTN
792: C
801: C
810: C          PLAB2 = ' '
819: C
828: C          IRWD = 0
837: C          continue
846: C          read(IOWK,'(a13)',end=180) PLAB2
855: C
864: C          if ( CWRK.ne.PLAB2 ) then
873: C              go to 170
882: C          else
891: C              go to 190
900: C          end if
909: C
918: C
927: C          continue
936: C          if ( IRWD.eq.0 ) then
945: C              IRWD = 1
954: C              call RWIND( IOWK )
963: C              go to 170
972: C          else
981: C              write(IMG*,*) 'XXX(TALIN2) Contents of working file ',
990: C                  & 'is not correct. <', PLAB2, '>'
999: C              call PRSTOP( 1, 'Program error in TALIN2 routine (5).')
1008: C              stop 666
1017: C              end if
1026: C
1035: C
1044: C          continue
1053: C
1062: C      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
1071: C      PLAB2 = 'DIMENSION'//TAG2(:LTAG2)
1080: C      call PACKLB( IETAL, 'IETAL:LABEL', PLAB2(:ICLEN2(PLAB2)), IRET2
1089: C      & )
1098: C      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
1107: C
1116: C      IDT1 = IDT + IDTOFF
1125: C
1134: C      ... JRBIN: "region" bin is used or not ...
1143: C      JRBIN = 0
1152: C      ... ITBIN: "time" bin is used or not (necessary for Feynman)...
1161: C      ITBIN = 0
1170: C

```

456: C
457: C
458: C
459: C
460: C
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471: C
472: C
473: C
474:
475: C
476: C
477: C
478: C
479:
480:
481: C
482:
483: C
484:
485:
486:
487: C
488:
489:
490:
491:
492:
493: C
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505: C
506:
507: C
508: C
509:
510:
511:
512: C
513: C
514:
515: C
516: C
517:
518: C
519:
520:


```

      write(MSG,*)'XXX(STALLY) ',
        'No. of time bins in direct tally is too small.'
      write(MSG,*) '    NGATE=', NGATE, ' NBIN=', NBIN
      call CNTERR( 'FATAL' )
      NERR     = NERR + 1
    end if
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

      do 260 I = 1, NGATE
        IWRK(2*I)   = I
      continue

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    else

      call RWIND( IOWK2 )
      LL         = 0
      do 270 J = 1, NBIN
        read(IOWK2,'(3i8)') IWRK(LL+1), NR, NL
        LL           = LL + 1
        read(IOWK2,'(10i8)') (IDUM,M=1,NR)
          (IWRK(LL+M),M=1,IWRK(LL))
        LL           = LL + IWRK(LL)
        read(IOWK2,'(A)') CWRK(:NL)
      continue

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

      call RWIND( IOWK2 )
      LL         = 0
      do 280 J = 1, NBIN
        read(IOWK2,'(3i8)') IJ, IWRK(LL+1), NL
        LL           = LL + 1
        read(IOWK2,'(10i8)') (IWRK(LL+M),M=1,IWRK(LL)),
          (IDUM,M=1,IJ)
        LL           = LL + IWRK(LL)
        read(IOWK2,'(A)') CWRK(:NL)

```

```

      write(MSG,*)'XXX(STALLY) ',
        'No. of time bins in direct tally is too small.'
      write(MSG,*) '    NGATE=', NGATE, ' NBIN=', NBIN
      call CNTERR( 'FATAL' )
      NERR     = NERR + 1
    end if
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

      do 260 I = 1, NGATE
        IWRK(2*I) = I
      continue

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    else

      call RWIND( IOWK2 )
      LL       = 0
      do 270 J = 1, NBIN
        read(IOWK2,'(3i8)') IWRK(LL+1), NR, NL
        LL       = LL + 1
        read(IOWK2,'(10i8)') (IDUM,M=1,NR)
          (IWRK(LL+M),M=1,IWRK(LL))
        LL       = LL + IWRK(LL)
        read(IOWK2,'(A)') CWRK(:NL)
      continue

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      call PACKND( IETAL, 'BIN', 'I4', IWRK(1), LL, IRET2 )
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

      call RWIND( IOWK2 )
      LL       = 0
      do 280 J = 1, NBIN
        read(IOWK2,'(3i8)') IJ, IWRK(LL+1), NL
        LL       = LL + 1
        read(IOWK2,'(10i8)') (IWRK(LL+M),M=1,IWRK(LL)),
          (IDUM,M=1,IJ)
        LL       = LL + IWRK(LL)
        read(IOWK2,'(A)') CWRK(:NL)

```

```

716:                                do 290 J = 1, NBIN
717:                                    read(IOWK2,'(3i8)') IJ, NR, NL
718:                                    read(IOWK2,'(10i8)') (IDUM,M=1,IJ+NR)
719:                                    read(IOWK2,'(A)') CWRK(:NL)
720: C                                >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
721:                                    call PACKCS( IETAL, 'REGION', CWRK(:NL), IRET2 )
722: C                                >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
723:      290        continue
724:                    end if
725:            end if
726:                IDT1    = IDT1 + IDMOFF + IDTAL(IDT1+3)
727:      300        continue
728: C
729:                if ( JRBIN.eq.0 ) then
730:                    do 310 J = 1, NREG
731:                        JDTRG(J,IT) = 1
732:                        JDTRG(J,IDTN) = 1
733:      310        continue
734:                end if
735: C
736: CCCCC in current version, d-tally number & e-tally number is identical
737: C ... in current version, d-tally number & e-tally number is *not*
738: C identical (modified in Jul, 1999)
739: C
740: Ccccc   IDT       = IDT + IDT1
741: CCC     IDT       = IDT1
742: CCC     KKET      = KKET + IDSIZ
743:         KKET      = KKET + IESIZ
744:         if ( JRV.ne.0 ) then
745: CCC       KRV     = KRV + IDSIZ
746:           KRV     = KRV + IESIZ
747:         end if
748: CCC     NEDTMX    = MAX(NEDTMX,IDSIZ)
749:           NEDTMX  = MAX(NEDTMX,IESIZ)
750:         if ( ISYN.eq.4 ) then
751:           NDTWMX  = MAX(NDTWMX,IDSIZ)
752:         end if
753:      320 continue
754: C
755:           NLETAL  = KKET - 1
756: C
757:           if ( NETRV.gt.0 ) then
758:               IETRV(2,NETRV+1) = KRV
759:               METRV  = KRV - 1
760:           end if
761: Check *****
762: C write(ipr,*) 'check TALIN2 : nletal ',nletal
763: C write(ipr,*) 'check TALIN2 : jdtrg '
764: C write(ipr,'(40i2)') jdtgr
765: C *****
766: C
767:         return
768: C
769: C ===== error messages ===
770: C
771:      666 continue
772:          write(IMSG,7000) PLAB(:ICLEN2(PLAB))
773:      7000 format(/lx,'XXX(TALLY: program error) CANNOT FIND STORED DATA <',
774:              & A,>' )
775:          call PRSTOP( 1, 'program error in $TALLY block processing.' )
776:          stop 666
777: C
778:      999 write(IMSG,*) 'XXX(TALIN2) too small working area.'
779:          write(IMSG,*) ' Please specify "NWORK" data',
780:              & ' at least greater than ', LLWRK, ' ( place it between '

```

src/shared/talin2.f

```
781:      &          '$TALLY' and first "&").'  
782:      call PRSTOP( 1, 'Insufficient memory to process $TALLY block.' )  
783:      stop 999  
784:      end
```

SAFE

src/shared/talin3.f

```

1:      subroutine TALIN3(JTEVE, LTEVE, NTEVE, KTEVE, NLTEMP, NKTEVE,
2:      & IDTAL, NIDTAL, NDTALY, IWRK, RWRK, DWRK, NWORK)
3: C=<MVP/GMVP>=====
4: C purpose: process "special" tally specifications - Stage 3;
5: C called in: talinp routine of mvp/gmvp.
6: C=====
7: C arguments ( i=input, o=output, w=work )
8: C
9: C o JTEVE : number of direct tallies to be taken for each tally event
10: C o LTEVE : pointer to KTEVE array for each tally event
11: C o KTEVE : pointer to IDTAL for each direct tally to be taken every
12: C          tally event
13: C i nltemp : size of memory available for IETAL ( actual size of IETAL
14: C           is determined in this routine.
15: C i idtal  : direct tally information.
16: C i nidtal : actual size of IDTAL array.
17: C i ndtaly : totia number of direct-tally.
18: C w iwrk   : working array (integer)
19: C w rwrk   : working array (real)
20: C w dwrk   : working array (real*8)
21: C i nwork  : working array size in word.
22: C
23: C (iwr, rwrk & dwrk have the same address)
24: C
25: C-----
26: C
27: C Structure of IDTAL(*)
28: C
29: C +----- direct tally loop ( NDTALY )
30: C |
31: C | (1) ID
32: C | (2) (dummy for debug: -999)
33: C | (3) pointer to DTALY(*)
34: C | (4) event #
35: C | (5) particle type
36: C | (6) number of dimensions
37: C | (7) multiplication factor type
38: C | (8) nuclide/atom # & reaction #
39: C | (9) d-tally #
40: C | (10) custom tally # (non-zero means user customized treatment)
41: C | (11) detector # (for surface event positive tally-surface #
42: C |              means particle current tally, and negative one
43: C |              means flux tally)
44: C | ( IDTOFF data so far )
45: C |
46: C | +----- tally dimension loop
47: C | | * dimension type
48: C | | * number of tallied bins
49: C | | * size of common bin -> tallied bin conversion table
50: C | | ( IDMOFF data so far )
51: C | | +----- common bin loop
52: C | | | tallied bin # for each common bin
53: C | | +-----
54: C | +-----
55: C +-----
56: C
57: C=====
58:      integer JTEVE(NTEVE)
59:      integer LTEVE(NTEVE)
60:      integer KTEVE(NLTEMP)
61: C
62:      integer IDTAL(NIDTAL)
63: C
64:      integer IWRK(NWORK)
65:      real RWRK(NWORK)

```

```

66:      real*8 DWRK(*)
67: C
68: C ... offset of direct-tally data in IDTAL(*)
69: C
70: CC parameter( IDTOFF = 12 )
71: C
72: C ... offset of tally dimension dependent data in IDTAL(*)
73: C
74: CC parameter( IDMOFF = 3 )
75: C
76:      include 'INC/_IDXOFF'
77: C
78:      include 'INC/_SIZES'
79:      include 'INC/_PTALY0'
80:      include 'INC/_PTLSP'
81:      include 'INC/_IOUNIT'
82: C
83: C===== local variables =====
84: C
85: C
86: C
87: C-----
88: C
89: C ... make KDTAL (the following procedure is a sample of access
90: C method to IDTAL array !!)
91: C
92:      KP = 0
93:      do 120 N = 1, NTEVE
94:          JTEVE(N) = 0
95:          LTEVE(N) = KP + 1
96:          IDT = 0
97:          do 110 I = 1, NDTALY
98:              if ( IDTAL(IDT+2).ne.-999 ) then
99:                  write(IMG,*) 'XXX(TALIN3) Failed to make direct tally',
100:                     & ' pointer. Direct tally information array(IDTAL) is',
101:                     & ' corrupt?'
102:                  call PRSTOP( 1, 'Program error in TALIN2 routine (1).')
103:                  stop 666
104:              end if
105: C
106:              if ( IDTAL(IDT+4).eq.N ) then
107:                  JTEVE(N) = JTEVE(N) + 1
108:                  KP = KP + 1
109:                  if ( KP.le.NLTEMP ) KTEVE(KP) = IDT + 1
110:              end if
111: C
112:              IDT1 = IDT + IDTOFF
113:              do 100 J = 1, IDTAL(IDT+6)
114:                  IDT1 = IDT1 + IDMOFF + IDTAL(IDT1+3)
115:              100 continue
116:              IDT = IDT1
117:              110 continue
118:              120 continue
119: C
120:          NKTEVE = KP
121:      check *****
122:      c write(ip,*) 'check TALIN3: JTEVE ',JTEVE
123:      c write(ip,*) 'check TALIN3: LTEVE ',LTEVE
124:      c write(ip,*) 'check TALIN3: KTEVE ',(KTEVE(i),i=1,KP)
125:      c *****
126:      if ( KP.gt.NLTEMP ) then
127:          write(IMG,*) 'XXX(TALIN3) Insufficient memory to compose ',
128:             & 'special tally infomation.',
129:             & 'Increase dynamic memory size at least ', KP - NLTEMP,
130:             & ' words.'

```

src/shared/talin3.f

```
131:      call PRSTOP(1, 'Insufficient memory to process $TALLY block.')
132:      stop 999
133:  end if
134:
135:  return
136:  end
```

SAFE

src/shared/talinp.f

```

1:      subroutine TALINP( CODE, ICALL, LICRES,NICRES,LWTIME,A,      IA,
2:      &                  DA, CHA, H, IH, DH, NGP,
3:      &                  JKPAR, KPSYM, KNGP, KENGP, KPLIM, JNEUT,
4:      &                  JPHOT, JTIME, JFISS, JEIGN, JRESP, JTLLT,
5:      &                  JTSRF, JPTDT, JDEBG, IDSRC, LIDSRN,NUC,
6:      &                  KDBNK, MDBNK, KIBNK, MIBNK,
7:      &                  LKR, LIMIT, LXIMP, LWKIL, LWSRV, LWGTF,
8:      &                  LWGTP, LPSALP,
9:      &                  JPHNU, NPATOM, NUCPN, NMTPN, LWGTPN) ! photo-nuc
10: C=<MVP/GMVP>=====
11: C purpose: input "special" tally specifications in $TALLY block.
12: C called in: intro routine of mvp, gmvp.
13: C=====
14: C arguments ( i=input, o=output, w=work )
15: C i CODE : name of code using this routine ('MVP'/'GMVP')
16: C o ICALL : output with value 1.
17: C io A : task shared dynamic data area used as "REAL" type.
18: C io IA : task shared dynamic data area used as "INTEGER" type.
19: C io DA : task shared dynamic data area used as "REAL*8" type.
20: C io CHA : task shared dynamic data area used as "CHARACTER*4" type.
21: C io H : task local dynamic data area used as "REAL" type.
22: C io IH : task local dynamic data area used as "INTEGER" type.
23: C io DH : task local dynamic data area used as "REAL*8" type.
24: C
25: C (A,IA,DA,H,IH & DH must have the same address as A(*) & H(*) of
26: C common /ARRAY/ & /LARRAY/)
27: C
28: C i J... : option flags
29: C i IDSRC(NSOUR) : source set ID's
30: C i LIDSRN : array index pointer to IDSRC (used to check source set ID's
31: C are input or not)
32: C o KDBNK(0:MDBNK) : index (flag) to save specified particle attribute
33: C as optional bank parameter (floating point data).
34: C o KIBNK(0:MIBNK) : index (flag) to save specified particle attribute
35: C as optional bank parameter(integer data).
36: C-----
37: C
38: C $TALLY block inputs:
39: C
40: C [...] means an optional item.
41: C ... | ... | ... : specify one of the items.
42: C
43: C $TALLY /* block header
44: C
45: C [ NGROUP[.N/.P]( number-of-energy-bin/group ) ]
46: C [ ENGYB[.N/.P]( energy boundaries ) ]
47: C
48: C [ NTIME( number-of-time-bin ) ] /** this should be neutron/photon
49: C dependent, but common in current version. ***/
50: C [ TIMEB( time boundaries ) ] /** common time bins
51: C [ WTIME( time weights ) ] /** time weight factor for variance red.
52: C /** particle generation weights
53: C
54: C [ NRESP( nresp ) ] /** number of group wise response functions.
55: C [ RESP[.N/.P]( r_1 r_2 ... r_nresp*ngroup[.N/.P] ) ]
56: C
57: C [ NSTAL( nstal ) ] /** number of point-wise response functions.
58: C [ STAL or CRESP( NUCLIDE-ID( 'MT'+reaction ID ) [ NUCLIDE-ID(...) ]
59: C /** point-wise response functions
60: C
61: C [ NPDET( NPDET ) ] /** number of point detectors.
62: C [ XPDET( specification of point detectors ) ]
63: C [ SPDET( specification of source treatment for point detectors ) ]
64: C
65: C [ TALLY-SURFACE( ID [S=][sign]body1 [S=][sign]body2 ... ) ]

```

```

66: C /** surface definition for surface crossing tally
67: C /** bodies other than those input as "S=+-body" are
68: C /** used only to "clip" surfaces.
69: C /** If no S=... is given, bodies other than the first one
70: C /** are all used for clipping.
71: C /** (currently "S=" is not implemented ... Mar 2000)
72: C
73: C /** arbitrary number of tally-surface can be input
74: C
75: C [ ANGLE( angl ang2 ... ) ] | [ COSINE( cos1 cos2 ... ) ]
76: C /** angle or cosine bin for surface crossing tally.
77: C /** angle between particle flight direction and
78: C /** normal vector of the surface.
79: C
80: C [ NWORK( nwork ) ] /** size of working area for tally input
81: C /** processing (word).
82: C##<2007/03/14:PN3:
83: C
84: C [ PNPRODUCE ] /** restriction of particles by photo-nuclear
85: C /** production.
86: C##>
87: C
88: C & /** delimiter of tally specifications **/
89: C [ID( tally-id-number )] /** optional tally ID
90: C [LABEL( "label string" )] /** tally label (printed)
91: C [REAL-VARIANCE] /** real variance calculation
92: C ---
93: C [EVENT( TRACK | COLLISION | POINT | ANALOG | LEAKAGE |
94: C ---
95: C SURFACE[:FLUX]:CURRENT](Tally-surface-ID) )
96: C ---
97: C /** Estimator (or event tallied)
98: C /** default is TRACK
99: C
100: C /** SURFACE() :
101: C /** ":FLUX" means flux tally and ":CURRENT" means current
102: C /** if ":FLUX" or ":CURRENT" is not given,
103: C /** positive Tally-surface-ID means particle current tally
104: C /** and negative ID means flux tally.
105: C
106: C [PHOTON | NEUTRON] /** particle type (coupled problem)
107: C
108: C or
109: C
110: C /** particle type (new form supported from April 2000)
111: C##<2007/03/14:PN3:
112: C##C [PARTICLE( particle-species-string )]
113: C [PARTICLE( particle-species-string )]
114: C##>
115: C
116: C ---
117: C [DIMENSION( dim1 dim2 dim3 ... )]
118: C ---
119: C /** dim? : dimensions of tallied data matrix.
120: C /** --- spatial variables -----
121: C /** REG[ION] : tally-region
122: C /** POI[NT] : spatial points ( point detector )
123: C /** --- others -----
124: C /** ENE[RGY] : energy
125: C /** TIM[E] : time
126: C /** ANG[LE] : angle
127: C /** SOU[RCE] : particle source set
128: C /** GEN[ERATION] : particle generation
129: C /** SOURCE : source set
130: C /** SOU[RCE]-REG[ION] : source region

```

src/shared/talinp.f

```
131: C      /**      MAR[KER]-REG[ION] : marker region
132: C      c##<2007/03/14:PN3:
133: C      /**      PRO[DUCE]-REG[ION] : produce region
134: C      /**      PRO[DUCE]-NUC[LIDE] : produce nuclide
135: C      /**      PRO[DUCE]-REA[CTION] : produce reaction (MT)
136: C      c##>
137: C      /** -----
138: C      /** The order of dimension is free, and tally is take as a
139: C      /** fortran-like matrix M( dim1, dim2, ... ), and output
140: C      /** as such.
141: C
142: C      /** specification for each tally dimension:
143: C
144: C      [REGION( list of tally-regions )] /** tallies are taken for each
145: C      /** region specified here.
146: C      [TALLY-REGION( list of tally-regions )] /** synonym of the above
147: C
148: C      [TIME( time1 time2 ... )] /** time boundaries tallied (not yet
149: C      /** available in the current version).
150: C      [ITIME( it_1 it_2 ... it_NTIME )]
151: C      /** it_i : When the particle is in i'th common-time bin,
152: C      /** it is tallied as it_i'th bin for this tally.
153: C      /** It_i's must compose integer sequence from 0.
154: C      /** IF it_i is zero, the time bin is excluded from tally.
155: C      [ITIME2( its-1 ite-1 its-2 ite-2 ... )]
156: C      /** When the common time bin # of particle is between its-i and
157: C      /** ite-i (includes its-i and ite-i), it is tallied as i'th
158: C      /** bin for this tally.
159: C
160: C      /** TIME(...), ITIME(...) and ITIME2(...) are mutually
161: C      /** exclusive input items.
162: C
163: C      [ENERGY( e1 e2 ... )] /** energy boundaries.
164: C      /** (not yet available in the current version).
165: C      [IENERGY( ie_1 ie_2 ... ie_NGROUP(.N/.P) )]
166: C      /** ie_i : When the particle is in i'th common-energy bin,
167: C      /** it is tallied as ie_i'th bin for this tally.
168: C      /** ie_i's must compose integer sequence from 0.
169: C      /** IF ie_i is zero, the time bin is excluded from tally.
170: C      [IENERGY2( ies-1 iee-1 ies-2 iee-2 ... )]
171: C      /** When the common energy bin # of particle is between ies-i
172: C      /** and iee-i (includes ies-i and iee-i), it is tallied as i'th
173: C      /** bin for this tally.
174: C
175: C      /** ENERGY(...), IENERGY(...) and IENERGY2(...) are mutually
176: C      /** exclusive input items.
177: C
178: C      [ANGLE( e1 e2 ... )] /** angle boundaries.
179: C      /** (not yet available in the current version).
180: C      [IANGLE( ia_1 ia_2 ... ia_NANGLE )]
181: C      /** ia_i : When the particle is in i'th common-angle bin,
182: C      /** it is tallied as ia_i'th bin for this tally.
183: C      /** Ia_i's must compose integer sequence from 0.
184: C      /** IF ia_i is zero, the angle bin is excluded from tally.
185: C      [IANGLE2( ias-1 iae-1 ias-2 iae-2 ... )]
186: C      /** When the common angle bin # of particle is between ias-i
187: C      /** and iae-i (includes ias-i and iae-i), it is tallied as i'th
188: C      /** bin for this tally.
189: C
190: C      [GENERATION( g1 g2 ... )] /** generation of particles
191: C      /** particles generated from source has the generation number 1
192: C      /** and secondary particles have 2, tertiary ones have 3
193: C      /** and so on
194: C      c##<2007/03/14:PN3:
195: C      c##C      [IGENE2( igs_1 ige_2 igs_2 ige_2 ... )]
```

```
196: C      [IGENE2( igs_1 ige_1 igs_2 ige_2 ... )]
197: C      c##>
198: C      /** particle generation number ranges.
199: C      /** When particle generation # is between ies_i
200: C      /** and iee_i (includes ies_i and iee_i), it is tallied as i'th
201: C      /** bin for this tally.
202: C      /** If igs_i is less than or equal to zero, that means
203: C      /** generation # less than or equal to ige_i,
204: C      /** and ige_i is less than or equal to zero, that means
205: C      /** generation # greater than or equal to igs_i.
206: C
207: C      [SOURCE( s_1 s_2 ... )] /** source set ID's
208: C      /** particles generated from source set having ID s_i are
209: C      /** tallied as i'th tally.
210: C
211: C      [SOURCE-REGION( r_1 r_2 ... )] /** region names
212: C      /** particles generated from regions r_i are
213: C      /** tallied as i'th tally.
214: C
215: C      [MARKER-REGION( r_1 r_2 ... )] /** region names
216: C      /** particles any of passed regions r_i are
217: C      /** marked and tallied separately.
218: C      /** Tally bin 1 is for marked, bin2 is not marked and
219: C      /** bin 3 is total.
220: C      [POINT( p_1 p_2 ..... )] /** point detector #
221: C      c##<2007/03/14:PN3:
222: C
223: C      [PRODUCE-REGION( r_1 r_2 ... )] /** region names
224: C      /** particles produced by collision from regions r_i are
225: C      /** tallied as i'th tally.
226: C
227: C      [PRODUCE-NUCLIDE( n_1 n_2 ... )] /** nuclide names
228: C      /** particles produced by collision with nuclides n_i are
229: C      /** tallied as i'th tally.
230: C
231: C      [PRODUCE-REACTION( mt_1 mt_2 ... )] /** reaction #
232: C      /** particles produced by collision with reactions mt_i are
233: C      /** tallied as i'th tally.
234: C      c##>
235: C
236: C      /** Specification of what is tallied (or factors applied to bare
237: C      /** tallied data):
238: C      /** If nothing is specified, particle flux is taken for volume
239: C      /** and point tallies and current for surface tally.
240: C      /** In general, one of the following is available.
241: C
242: C      [IRESP( n )]
243: C      /** use the multigroup response data RESP(NGROUP,n)
244: C      /** as a weighting function.
245: C      [PRESP( n )]
246: C      /** use the point-wise response data STAL(n)
247: C      /** as a weighting function.
248: C      /** [ERESP( r1 r2 ... )] /** energy-bin-wise response function.
249: C      /** (unsupported incurrent version)
250: C
251: C      /** Macroscopic reaction rate for specified reaction is tallied.
252: C
253: C      /** NEUTRON
254: C      [MACRO( TOTAL | NUFISSION | FISSION | ELASTIC | CAPTURE |
255: C      /** -----
256: C      /** INELASTIC | N2N | LOSS ) ]
257: C      /** -----
258: C      /** PHOTON
259: C      [MACRO( TOTAL | COHERENT | INCOHERENT | PAIRPRODUCTION |
260: C      /** -----
```

src/shared/talinp.f

```

261: C          PHOTELECTRIC ) ]
262: C          - - -
263: C
264: C /** Microscopic reaction rate for specified nuclide and reaction.
265: C /** If ":SPREAD" is after nuclide/atom name, per-atom reaction is
266: C /** tallied for all spatial region (simple spatial average), else
267: C /** reaction rate is multiplied by the number density of the nuclide
268: C /** at particle position (density weighted i.e. actual reaction rate)
269: C /**
270: C
271: C /** NEUTRON
272: C [MICRO:nuclide[:SPREAD](
273: C          TOTAL | NUFISSION | FISSON | ELASTIC | CAPTURE |
274: C          --- | --- | --- | --- | ---
275: C          INELASTIC | N2N | LOSS ) ]
276: C          --- | --- | ---
277: C /** PHOTON
278: C [MICRO:atomid[:SPREAD](
279: C          TOTAL | COHERENT | INCOHERENT | PAIRPRODUCTION |
280: C          --- | --- | --- | ---
281: C c##<2007/03/14:PN3:
282: C c##C          PHOTELECTRIC ) ]
283: C c##C          - - -
284: C          PHOTELECTRIC ) ]
285: C          - - -
286: C /** photonuclear
287: C [MICROPN:pn-nuclide[:SPREAD]( MT# ) ]
288: C c##>
289: C
290: C /** Multiplication factor in matrix form
291: C /**
292: C [MULTIPLY( ... )]
293: C
294: C /** Count the number of analog events for noise analysis
295: C /**
296: C [ANALOG( SCATTERING | ABSORPTION | CAPTURE | FISSON )]
297: C
298: C /** Feynman-alpha analysis
299: C /**
300: C [FEYNMAN-Y( ngate [processed tally ID] )]
301: C
302: C /** Custom (user defined) tally requires this ID (non-zero)
303: C /**
304: C [USER( custom-tally-ID )] or [CUSTOM( custom-tally-ID )]
305: C ---
306: C
307: C & /** delimiter of tally specifications */
308: C ...
309: C
310: C $END [TALLY]
311: C =====
312: C character*(*) CODE
313: C real A(*), H(*)
314: C character*4 CHA(*)
315: C integer IA(*), IH(*)
316: C real*8 DA(*), DH(*)
317: C
318: C integer NGP(KPLIM)
319: C integer JKPAR(KPLIM)
320: C integer KNGP(KPLIM+1), KENGP(KPLIM+1)
321: C character*32 KPSYM(2,KPLIM)
322: C
323: C integer JDEBG(*)
324: C
325: C include 'INC/_SIZES'

```

```

326: C include 'INC/_PTALY0'
327: C include 'INC/_PTLSP'
328: C include 'INC/_STALY'
329: C include 'INC/_PGEOM'
330: C include 'INC/_IOUNIT'
331: C
332: C
333: C integer IDSRC(NSOUR), LIDSR
334: C integer KDBNK(0:MDBNK), KIBNK(0:MIBNK)
335: C
336: C ===== local variables =====
337: C
338: C character VNAME*72, VSUBF*1
339: C character VNM*6
340: C
341: C ... items of E-tally specification ..
342: C
343: C character*6 TAG
344: C character*12 THEAD
345: C integer ID
346: C character*72 TLABEL
347: C character*32 EVENTN
348: C character*8 PARTCL
349: C parameter( MAXDIM = 8 )
350: C character*16 TALDIM
351: C
352: C character CWRK*128
353: C character PLAB*32
354: C
355: C ... working array for TALLY-SURFACE input
356: C parameter( MXBDY = 128 )
357: C integer ISFWK(MXBDY,2)
358: C
359: C -----
360: C
361: C .... search parameters in input ..
362: C
363: C ICALL = 1
364: C call HEADER( IPR, 'SPECIAL TALLY SPECIFICATION DATA' )
365: C
366: C ... zero set for special tally related data pointers
367: C
368: C LIDTAL = 0
369: C LIETAL = 0
370: C LKDTAL = 0
371: C LKETAL = 0
372: C
373: C call GTLAST( LAST )
374: C
375: C JNP = JNEUT*JPHOT
376: C call RWIND( IUG1 )
377: C call RWIND( IUG2 )
378: C NERR = 0
379: C ... temporary flag to show point detectors are used as special tally
380: C JJPNT = 0
381: C
382: C ... size and pointer for working area ...
383: C
384: C NWORK = MAX(NGROUP,NTIME,NREG,NTREG,4096)
385: C LSWRK = 0
386: C
387: C ... pointer to a temporary packet area.
388: C
389: C KLPT = 0
390: C

```


src/shared/talinp.f

```

391:      LASTPP = 0
392: C
393: C ... NETRV : number of tallies whose "real variance" is calculated ...
394: C
395:      NETRV = 0
396: C
397: C ... ital: number of tallies ...
398: C
399:      ITAL = 0
400: c##<2007/03/14:PN3:
401:      LINCSTEN= 0
402:      LMTMPN = 0
403:      NMTMPN = 0
404: c##>
405: C
406:      LTAG = 0
407:      LLABEL = 1
408:      PARTCL = ' '
409:      KJID = 0
410:      TLABEL = ' '
411:      KJREG = 0
412:      KJSOU = 0
413:      KJSRG = 0
414:      KJMRG = 0
415: c##<2007/03/14:PN3:
416:      KJPRG = 0
417:      KJPRN = 0
418:      KJPRR = 0
419:      KJPNP = 0
420: c##>
421: C
422:      EVENTN = ' '
423:      KJENG = 0
424:      NDIM = 0
425:      KJTIME = 0
426:      KJGENE = 0
427: C
428:      KJWHAT = 0
429:      KJRESP = 0
430:      KJMAC = 0
431:      KJMIC = 0
432:      KJMULT = 0
433: c##<2007/03/14:PN3:
434:      KJMICPN = 0
435:      LINCSEI = 0
436: c##>
437: C
438:      NTSRF = 0
439:      NANGLE = 0
440:      LIDSRF = 0
441:      MTSRF = 0
442:      LITSRF = 0
443:      LKTSRF = 0
444:      LKTSFJ = 0
445:      LANGLB = 0
446: C
447:      KJCOS = 0
448:      KJANG = 0
449: C
450: C ... KKPID is the only possible particle species ID if non zero.
451:      KKPID = 0
452: if ( NPKIND.eq.1 ) then
453:   do 100 IK = 1, KPLIM
454:     if ( JKPAR(IK).ne.0 ) then
455:       KKPID = IK

```

```

456:      end if
457: 100 continue
458: end if
459: C ... IUB may be used to save tally-surface information temporary
460: call RWIND( IUB )
461: C
462: 110 call FREADS( VNAME, NLEN, IEND )
463: if ( IEND.ne.0 ) go to 9003
464: C
465: C=====
466: C ===== data common to all tallies =====
467: C=====
468: C
469: if ( ITAL.eq.0.and.VNAME(:NLEN).ne.'&'.and.VNAME(:NLEN).ne.'$END'
470: & ) then
471: c##<2007/03/14:PN3:
472: c## if ( IMATCH('NGROUP*',VNAME(:NLEN)).ne.0
473: c## & .or. 'NTIME'.eq.VNAME(:NLEN) .or. 'NRESP'.eq.VNAME(:NLEN)
474: c## & .or. 'NSTAL'.eq.VNAME(:NLEN) .or. 'NANGLE'.eq.VNAME(:NLEN)
475: c## & .or. 'NPDET'.eq.VNAME(:NLEN) ) then
476: if ( IMATCH('NGROUP*',VNAME(:NLEN)).ne.0 .or.
477: & VNAME(:NLEN).eq.'NTIME' .or. VNAME(:NLEN).eq.'NRESP' .or.
478: & VNAME(:NLEN).eq.'NSTAL' .or. VNAME(:NLEN).eq.'NANGLE' .or.
479: & VNAME(:NLEN).eq.'NPDET' ) then
480: c##>
481: N = -999
482: call VALUE0( VNAME(:NLEN), ICALL, 'I4', N )
483: C
484: if ( N.ne.-999 ) then
485: C
486: C ... "energy group" number
487: C
488: C <<comment on Apr 2000 by M.Sasaki>>
489: C
490: C It might cause some troubles to speciy number of energy group
491: C here especially in calcuations of multiple particle species.
492: C
493: if ( IMATCH('NGROUP*',VNAME(:NLEN)).ne.0 ) then
494: KK = INDEX(VNAME(:NLEN),'.')
495: if ( KK.eq.0 ) then
496: NGROUP = N
497: if ( KKPID.ne.0 ) then
498: NGP(KKPID) = NGROUP
499: else
500: write(IMG,'(1X,A,A,A)')
501: & '!!!(TALINP) Specifying "NGROUP" without ',
502: & 'particle species in multi-particle-species',
503: & ' calculation may cause error...'
504: call CNTERR( 'WARNING' )
505: end if
506: else
507: call KPSYMB( VNAME(KK+1:NLEN), '>', IKPID, 0 )
508: if ( IKPID.eq.0 ) then
509: write(IMG,'(1X,A,A,A,A)')
510: & 'XXX(TALINP) Specifying unsupported particle',
511: & ' type in <', VNAME(1:NLEN), '>'
512: & NERR = NERR + 1
513: call CNTERR( 'FATAL' )
514: else if ( JKPAR(IKPID).eq.0 ) then
515: write(IMG,'(1X,A,A,A,A)')
516: & '!!!(TALINP) Specifying unused particle'
517: & ', type in <', VNAME(1:NLEN),
518: & '>. Ignored'
519: call CNTERR( 'WARNING' )
520: else

```

src/shared/talinp.f

```

521:          NGP(IKPID) = N
522:        end if
523:      end if
524: C
525: C      if ( IMATCH('NGROUP.N',VNAME(:NLEN)).ne.0 ) then
526: C          NGP1 = N
527: C          if ( JPHOT.eq.0 ) NGROUP = NGP1
528: C          if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
529: C      else if ( IMATCH('NGROUP.P',VNAME(:NLEN)).ne.0 ) then
530: C          NGP2 = N
531: C          if ( JNEUT.eq.0 ) NGROUP = NGP2
532: C          if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
533: C      else if ( VNAME(:NLEN).eq.'NGROUP' ) then
534: C          NGROUP = N
535: C          if ( JNEUT.ne.0 ) NGP1 = NGROUP
536: C          if ( JPHOT.ne.0 ) NGP2 = NGROUP
537: C
538: C      ... number of (common) time boundaries.
539: C
540: C      else if ( VNAME(:NLEN).eq.'NTIME' ) then
541: C          NTIME = N
542: C
543: C      ... number of responses.
544: C
545: C      else if ( VNAME(:NLEN).eq.'NRESP' ) then
546: C          NRESP = N
547: C      else if ( VNAME(:NLEN).eq.'NSTAL' ) then
548: C          NSTAL = N
549: C
550: C      ... number of angle.
551: C
552: C      else if ( VNAME(:NLEN).eq.'NANGLE' ) then
553: C          NANGLE = N
554: C
555: C      ... number of point detectors.
556: C
557: C      else if ( VNAME(:NLEN).eq.'NPDET' ) then
558: C          NPDET = N
559: C      end if
560: C      end if
561: C      go to 110
562: C      end if
563: C
564: C      ... energy boundaries
565: C
566: C      if ( IMATCH('ENGYB*',VNAME(:NLEN)).ne.0 ) then
567: C
568: C          call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
569: C      &          NGROUP )
570: C          call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
571: C
572: C          if ( LENGYB.eq.0 ) then
573: C              call KEPV( A(1), 'ENGYB', LENGYB, KENGP(KPLIM+1)-1, 'R4',
574: C      &          LAST, 0.0, JDEBG )
575: C          end if
576: C
577: C          KP = 0
578: C          KK = INDEX(VNAME(:NLEN),'.')
579: C          if ( KK.eq.0 ) then
580: C              if ( KKPID.ne.0 ) KP = KKPID
581: C          else
582: C              call KPSYMB( VNAME(KK+1:NLEN), '>', KP, 0 )
583: C              if ( KP.eq.0 .or. KP.ne.0.and.JKPAR(KP).eq.0 ) then
584: C                  write(IMG,'(1X,A,A,A)')
585: C      &          '!!!(talinp) Energy boundary data <',

```

```

586: C      &          VNAME(:NLEN),
587: C      &          '>' is not effective for current input.'
588: C          KP = 0
589: C      end if
590: C      end if
591: C
592: C      if ( KP.eq.0 ) then
593: C          call DMREAD( VNAME(:NLEN), IDUMMY, IDUMMY, IRET )
594: C          if ( IRET.ne.0 ) go to 9004
595: C      else
596: C          NB = NGP(KP) + 1
597: C          call R4READ( VNAME(:NLEN), A(LENGYB+KENGP(KP)-1), NA,
598: C      &          -NB, IRET )
599: C          if ( NA.ne.NB ) then
600: C              write(IMG,'(1X,A,A,A,A,I4,A,I4,A)')
601: C      &          'XXX(talinp) Number of energy boundary data',
602: C      &          ' <', VNAME(:NLEN), '>' (=, NA,
603: C      &          'i is different from an expected one (=, NB,
604: C      &          ').')
605: C          call CNTERR( 'FATAL' )
606: C          end if
607: C
608: C          call CKVAL4( VNAME(:NLEN), A(LENGYB+KENGP(KP)-1), NGP(KP)
609: C      &          +1, IFL, 1 )
610: C          if ( IFL.ne.0 ) then
611: C      c##<2007/03/14:PN3:
612: C              write(IMG,906) VNAME(1:NLEN), IFL
613: C          906 format(/' XXX(talinp) Zero or negative energy boundary data <',a,
614: C      &          '>' is found. IFL=',i5)
615: C      c##>
616: C          call CNTERR( 'FATAL' )
617: C          end if
618: C          end if
619: C          go to 110
620: C      end if
621: C
622: C      if ( IMATCH('ENGYB.N',VNAME(:NLEN)).ne.0
623: C      &      .or. 'ENGYB'.eq.VNAME(:NLEN) ) then
624: C          if ( CODE.eq.'MVP' ) then
625: C              call ARAYIN( VNAME, A, ICALL, 'R4', LENGYB, ICK,
626: C      &          JDEBG, LAST, NGP1+1, ' ' )
627: C              call SIZCHK( VNAME, 'NGP1', NGP1, NFINP(0)-1, ICK )
628: C
629: C              call CKVAL4( VNAME, A(LENGYB), NGP1+1, ICHK, 1 )
630: C          else if ( CODE.eq.'GMVP' ) then
631: C              NGB = NGROUP + 1
632: C              if ( JNP.ne.0 ) NGB = NGB + 1
633: C              if ( JNP.eq.0 ) then
634: C                  call ARAYIN( VNAME, A, ICALL, 'R4', LENGYB, ICK,
635: C      &          JDEBG, LAST, NGB, ' ' )
636: C                  call SIZCHK( VNAME, 'NGP1', NGB, NFINP(0), ICK )
637: C              else
638: C                  call CKVAL4( VNAME, A(LENGYB), NGB, ICHK, 1 )
639: C              else
640: C                  if ( LENGYB.eq.0 ) then
641: C                      call KEPV( A(1), 'ENGYB', LENGYB, NGB, 'R4',
642: C      &          LAST, 0.0, JDEBG )
643: C                  end if
644: C
645: C          .... input data temporary from A(LTMPRY)
646: C
647: C          call GTLAST( LASTTM )
648: C          call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK,
649: C      &          JDEBG, LAST, NGP1+1, ' ' )
650: C          call SIZCHK( VNAME, 'NGP1', NGP1, NFINP(0)-1, ICK )

```

src/shared/talinp.f

```

651: C          call CKVAL4( VNAME, A(LTMPRY), NGP1+1, ICHK, 1 )
652: C
653: C          .... scatter data a(ltmpry) onto a(lengyb) ...
654: C
655: C          call ASCTR4( VNAME, A(LENGYB), A(LTMPRY), NGB, 1,
656: C &              NGP1+1, 1, NGP1+1 )
657: C
658: C          call STLAST( 'ENGYB', LASTTM, LAST )
659: C          end if
660: C          end if
661: C          go to 100
662: C      else if ( IMATCH('ENGYB.P',VNAME(:NLEN)).ne.0 ) then
663: C          call ARAYIN( VNAME, A, ICALL, 'R4', LENGPB, ICK, JDEBG,
664: C &              LAST, NGP2+1, ' ' )
665: C          call SI2CHK( VNAME, 'NGP2', NGP2, NFIMP(0)-1, ICK )
666: C          call CKVAL4( VNAME, A(LENGPB), NGP2+1, ICHK, 1 )
667: C
668: C          if ( CODE.eq.'GMVP' ) then
669: C              .... scatter data a(lengpb) onto a(lengyb) ...
670: C
671: C              NGB = NGROUP + 1
672: C              if ( JNP.ne.0 ) NGB = NGB + 1
673: C              if ( LENGYB.eq.0 ) then
674: C                  call KEPV( A(1), 'ENGYB', LENGYB, NGB, 'R4', LAST,
675: C &                      0, 0, JDEBG )
676: C              end if
677: C              NGP2S = NGP1 + 2
678: C              if ( NGP1.le.0 ) NGP2S = 1
679: C              call ASCTR4( VNAME, A(LENGYB), A(LENGPB), NGB, 1, NGP2
680: C &                  +1, NGP2S, NGP2+1 )
681: C          end if
682: C          go to 100
683: C
684: C      end if
685: C      end if
686: C
687: C      ... time boundaries
688: C
689: C      if ( VNAME(:NLEN).eq.'TIMEB' ) then
690: C          call ARYIN0( VNAME, A, ICALL, 'R4', LTIMEB, ICK, JDEBG,
691: C &              LAST, NTIME, +1, ' ' )
692: C          call SI2CHK( VNAME, 'NTIME', NTIME, NFIMP(0)-1, ICK )
693: C          go to 110
694: C      end if
695: C
696: C
697: C      ... time weighting factor (here?)
698: C
699: C      if ( VNAME(:NLEN).eq.'WTIME' ) then
700: C          call ARAYIN( VNAME, A, ICALL, 'R4', LWTIME, ICK, JDEBG,
701: C &              LAST, NTIME, ' ' )
702: C          go to 110
703: C      end if
704: C
705: C      ... angle boundaries
706: C
707: C      if ( VNAME(:NLEN).eq.'ANGLE' .or. VNAME(:NLEN).eq.'COSINE' )
708: C &      then
709: C          if ( LANGLEB.ne.0 ) then
710: C              write(IMG,'(1X,A,A)') 'XXX(talinp) Angle tally bin',
711: C &              'are given more than once.'
712: C              call CNTERR( 'FATAL' )
713: C              call DMREAD( VNAME(:NLEN), NA, NB, IEND )
714: C              if ( IEND.ne.0 ) go to 9004
715: C          else

```

```

716: C
717: C          call ARYIN0( VNAME, A, ICALL, 'R8', LANGLEB, ICK, JDEBG,
718: C &              LAST, NANGLE, +1, 'NANGLE' )
719: C
720: C          ... convert to cosine bin if input in degree
721: C
722: C          MODE = 1
723: C          if ( VNAME(:NLEN).eq.'ANGLE' ) MODE = 2
724: C
725: C          call ANGSET( MODE, A(LANGLEB), NANGLE, IPR, MSG )
726: C
727: C          end if
728: C          go to 110
729: C      end if
730: C
731: C      ... tally-surface ==> stored on I/O unit IUB here
732: C
733: C      if ( VNAME(:NLEN).eq.'TALLY-SURFACE' ) then
734: C          NB = -MXBDY
735: C          call I4READ( VNAME(:NLEN), ISFWK(1,1), NA, NB, IEND )
736: C          if ( IEND.ne.0 ) go to 9001
737: C      check %%%%%%%%%%
738: C      write(*,*) '%%%% isfwk ',(isfwk(i,1),i=1,na)
739: C      %%%%%%%%%%
740: C      if ( NA.gt.MXBDY ) then
741: C          write(IMG,'(1X,A,A,I6,A)')
742: C          'XXX(talinp) TALLY-SURFACE() has too many',
743: C          ' data (current program limit is ', MXBDY, ' )'
744: C          call CNTERR( 'FATAL' )
745: C          NA = MXBDY
746: C      end if
747: C      if ( NA.le.1 ) then
748: C          write(IMG,'(1X,A,I5,A)')
749: C          'XXX(talinp) TALLY-SURFACE() has only ',
750: C          ' NA, ' data (at least ID and a body are required)'
751: C          call CNTERR( 'FATAL' )
752: C          ISFWK(1,1) = 999
753: C      end if
754: C      NTSRF = NTSRF + 1
755: C
756: C      if ( ISFWK(1,1).le.0 ) then
757: C          write(IMG,'(1X,A,A,I6)')
758: C          'XXX(talinp) ID of TALLY-SURFACE',
759: C          ' must be a positive integer : given ',
760: C          ISFWK(1,1)
761: C          call CNTERR( 'FATAL' )
762: C      end if
763: C
764: C      ... currently only the first body is actual tally surface
765: C      (effective surface should be specified as S=body in future)
766: C
767: C      ISFWK(2,2) = 1
768: C      do 120 I = 3, NA
769: C          ISFWK(I,2) = 0
770: C      120 continue
771: C
772: C      write(IUB) NA - 1, ISFWK(1,1)
773: C      write(IUB) (ISFWK(I,1),I=2,NA)
774: C      write(IUB) (ISFWK(I,2),I=2,NA)
775: C
776: C      go to 110
777: C      end if
778: C
779: C      ... response
780: C

```

src/shared/talinp.f

```

781: CM      if ( VNAME(:NLEN).eq.'RESP' ) then
782: CM          call ARAYIN( VNAME,A, ICALL, 'R4', LRESP, ICK, JDEBG, LAST,
783: CM      &              NGROUP*NRESP, ' ' )
784: CM          call CKVAL4( VNAME, A(LRESP), NGROUP*NRESP, ICHK, 0 )
785: CM          go to 100
786: CM      end if
787:
788: CM      if ( IMATCH('RESP*',VNAME(:NLEN)).ne.0 ) then
789: CM          call NGPCHK( 'MVP', NGP, JKPAR, KPSYM, KPLIM, NGP1, NGP2,
790: CM      &              NGROUP )
791: CM          call KGPSET( KNGP(1), KENGP(1), NGP(1), JKPAR(1), KPLIM )
792:
793: CM          if ( LRESP.eq.0 ) then
794: CM              call KEPV( A(1), 'RESP', LRESP, NRESP*NGROUP, 'R4', LAST,
795: CM      &              0.0, JDEBG )
796: CM          end if
797:
798: CM          KP      = 0
799: CM          KK      = INDEX(VNAME(:NLEN),'.')
800: CM          if ( KK.eq.0 ) then
801: CM              if ( KKPID.ne.0 ) KP      = KKPID
802: CM          else
803: CM              call KPSYMB( VNAME(KK+1:NLEN), '>', KP, 0 )
804: CM              if ( KP.eq.0 .or. KP.ne.0.and.JKPAR(KP).eq.0 ) then
805: CM                  write(IMG,'(1X,A,A)')
806: CM      &              '!!!(talinp) Response data <', VNAME(:NLEN),
807: CM      &              '> is not effective for current input.'
808: CM              KP      = 0
809: CM          end if
810: CM      end if
811: C
812: CM      if ( KP.eq.0 ) then
813: CM          call DMREAD( VNAME(:NLEN), IDUMMY, IDUMMY, IRET )
814: CM          if ( IRET.ne.0 ) go to 9004
815: CM      else
816: C
817: C          .... input data temporary from A(LTMPRY)
818: C
819: C          NG      = NGP(KP)
820: C          call GTLAST( LASTTM )
821: C          call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
822: C      &              LAST, NG*NRESP, ' ' )
823: C
824: C          .... scatter data a(ltmpry) onto a(lpenrg) ...
825: C
826: C          call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
827: C      &              NGP(KP), KNGP(KP), NGP(KP) )
828: C
829: C          call STLAST( 'RESP', LASTTM, LAST )
830: C      end if
831: C      go to 110
832: C      end if
833:
834: C      if ( JNP.eq.0 .or. VNAME(:NLEN).eq.'RESP' ) then
835: C          call ARAYIN( VNAME, A, ICALL, 'R4', LRESP, ICK, JDEBG,
836: C      &              LAST, NGROUP*NRESP, ' ' )
837: C          call CKVAL4( VNAME, A(LRESP), NGROUP*NRESP, ICHK, 0 )
838: C          go to 100
839: C      else if ( IMATCH('RESP.N',VNAME(:NLEN)).ne.0 ) then
840: C          if ( LRESP.eq.0 ) then
841: C              call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4',
842: C      &              LAST, 0.0, JDEBG )
843: C          end if
844: C
845: C          .... input data temporary from A(LTMPRY)

```

```

846: C
847: C          call GTLAST( LASTTM )
848: C          call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
849: C      &              LAST, NGP1*NRESP, ' ' )
850: C
851: C          .... scatter data a(ltmpry) onto a(lpenrg) ...
852: C
853: C
854: C          call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
855: C      &              NGP1, 1, NGP1 )
856: C
857: C          call STLAST( 'RESP', LASTTM, LAST )
858: C          go to 100
859: C      else if ( IMATCH('RESP.P',VNAME(:NLEN)).ne.0 ) then
860: C          if ( LRESP.eq.0 ) then
861: C              call KEPV( A(1), 'RESP', LRESP, NGROUP*NRESP, 'R4',
862: C      &              LAST, 0.0, JDEBG )
863: C          end if
864: C
865: C          .... input data temporary from A(LTMPRY)
866: C
867: C          call GTLAST( LASTTM )
868: C          call ARAYIN( VNAME, A, ICALL, 'R4', LTMPRY, ICK, JDEBG,
869: C      &              LAST, NGP2*NRESP, ' ' )
870: C
871: C          .... scatter data a(ltmpry) onto a(lpenrg) ...
872: C
873: C          call ASCTR4( VNAME, A(LRESP), A(LTMPRY), NGROUP, NRESP,
874: C      &              NGP2, NGP1+1, NGP2 )
875: C
876: C          call STLAST( 'RESP', LASTTM, LAST )
877: C          go to 100
878: C      end if
879: C      end if
880: C
881: C      ... continuous energy
882: C
883: C      if ( VNAME(:NLEN).eq.'STAL'
884: C      &      .or. IMATCH('CRESP',VNAME(:NLEN)).ne.0 ) then
885: C          if ( CODE.eq.'MVP' ) then
886: C              call CRSINP( VNAME(:NLEN), A, IA, LICRES, NICRES, NSTAL,
887: C      &              JDEBG )
888: C          else
889: C              write(IMG,'(1X,A,A)')
890: C      &              '!!!(talinp) Point-wise response data ',
891: C      &              'are specified in Multi-group code.'
892: C              call CNTERR( 'WARNING' )
893: C              call DMREAD( VNAME(:NLEN), NA, NB, IEND )
894: C              if ( IEND.ne.0 ) go to 9004
895: C          end if
896: C          go to 110
897: C      end if
898: C
899: C      ... specification of point detectors
900: C
901: C      if ( VNAME(:NLEN).eq.'XPDET' ) then
902: C          call ARAYIN( VNAME, A, ICALL, 'R8', LXPDET, ICK, JDEBG,
903: C      &              LAST, NPDET*5, ' ' )
904: C          go to 110
905: C      end if
906: C      if ( VNAME(:NLEN).eq.'SPDET' ) then
907: C          call ARAYIN( VNAME, A, ICALL, 'I4', LJSPDT, ICK, JDEBG,
908: C      &              LAST, NPDET, ' ' )
909: C          go to 110
910: C      end if

```

src/shared/talinp.f

```

911: C
912: C ... nwork
913: C
914:       if ( VNAME(:NLEN).eq.'NWORK' ) then
915:         call I4READ( VNAME(:NLEN), NWORK, NA, -1, IEND )
916:         if ( NWORK.lt.0 ) then
917:           write(IMG,'(1X,A,A,I8,A)')
918:           & 'XXX(talinp) Input for working area size ',
919:           & '(NWORK=', NWORK, ') is not positive !!'
920:
921:           NWORK = MAX(NGROUP,NTIME,NREG,NTREG,4096)
922:
923:           call CNTERR( 'FATAL' )
924:         end if
925:         go to 110
926:       end if
927: C
928: C ... VOL
929: C
930:       if ( VNAME(:NLEN).eq.'VOL' ) then
931:         call ARAYIN( VNAME, A, ICALL, 'R4', LVOL, ICK, JDEBG, LAST,
932:         & NINPZ, 'NINPZ' )
933:         call CKVAL4( VNAME, A(LVOL), NINPZ, IFL, 1 )
934:         if ( IFL.ne.0 ) then
935:           write(IMG,'(/1X,'XXX($TALLY) Data <RVOL> has a value',
936:           & ' with invalid sign.'/))
937:           call CNTERR( 'FATAL' )
938:         end if
939:         go to 110
940:       end if
941: C
942: C ... RVOL, TRVOL
943: C
944:       if ( VNAME(:NLEN).eq.'RVOL' .or. VNAME(:NLEN).eq.'TRVOL' )
945:       & then
946: C
947: C .... TEMPORARY WORKING ARRAY ....
948: C
949:       NRRR0 = max(NREG,NTREG)
950:       if ( LKR.eq.0 ) then
951:         call KEEP( 'KR', LKR, NRRR0, 'I4', LAST, JDEBG )
952:       end if
953: C
954:       call REGDAT( IMG, VNAME(:NLEN), VNM, VSUBF, A, A, LIMIT,
955:       & LAST,
956:       & JNEUT, JPHOT, JDEBG, NREG, NGP(1), JKPAR(1), KPSYM,
957:       & KPLIM, KNGP(1), NGROUP, NGP1, NGP2, LXIMP, LWKIL,
958:       & LWSRV, LWGTF, LWGTP, LRVOL, LTRVOL, LPSALP, JTLIT,
959:       & A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
960:       & A(LISPNM), A(LISUSP), A(LKSUZN), A(LIRGSP), NEST,
961:       & NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAMES, A(LITRNM),
962:       & NTRREG, A(LIPTRG), A(LLPTRG), A(LKR), NRRR0,
963:       & NUCPN, NMTPN, LWGTPN)
964:       go to 110
965:     end if
966: c##<2007/03/14:PN3:
967: C
968: C ... pnproduce
969: C
970:       if ( VNAME(:NLEN).eq.'PNPRODUCE' ) then
971:         KPNPRD = 1
972:         write(IPR,'(1h)')
973:         write(IPR,*) ' PNPRODUCE : ',KPNPRD, ' (photo-nuclear)'
974:         go to 110
975:       end if

```

```

976: c##>
977: C
978: C ---- invalid data name ---
979: C
980:       write(IMG,7000) VNAME(:NLEN)
981:       7000 format(/' XXX(talinp) Input data whose name is <',A,
982:       & '> does not exist or direct input is not allowed !!!')
983:       & call CNTERR( 'FATAL' )
984:       NERR = NERR + 1
985:       call FPROBE( IIP, ' ', CWRK )
986:       if ( IIP.ne.0.and.CWRK(1:1).eq.'(' ) then
987:         call DMREAD( VNAME(:NLEN), NA, NB, IEND )
988:         if ( IEND.ne.0 ) go to 9004
989:       end if
990: C
991:       go to 110
992: C
993:       end if
994: C=====
995: C ... manage tally-surface data
996: C=====
997: check %%%%%%%%%
998: c       write(*,*) '%% lswrk-1 <', lswrk,> '
999: c %%%%%%%%%
1000:       if ( ITAL.eq.0.and.NTSRF.gt.0 ) then
1001:         if ( NANGLE.eq.0 ) then
1002:           NANGLE = 2
1003:           call KEEP( 'ANGLB', LANGLEB, NANGLE+1, 'R8', LAST, JDEBG )
1004:           MODE = -1
1005:           call ANGSET( MODE, A(LANGLEB), NANGLE, IPR, IMG )
1006:           write(IMG,'(1X,A,A)')
1007:           & '!!! (talinp) No angle bin is specified for surface',
1008:           & ' tally. Default bin is used.'
1009:         end if
1010: C
1011:       call KEEP( 'IDSRF', LIDSRF, NTSRF, 'I4', LAST, JDEBG )
1012:       call KEEP( 'ITSRF', LITSRF, (NTSRF+1)*2, 'I4', LAST, JDEBG )
1013: C
1014:       call REMAIN( 'KTSRF', NLTEMP, 'I4', LAST )
1015:       NLTEMP = NLTEMP/2
1016:       if ( NLTEMP.le.0 ) then
1017:         write(IMG,'(1X,A,A)')
1018:         & 'XXX(talinp) No-more memory to store TALLY-SURFACE',
1019:         & ' information !!!'
1020:       call PRSTOP( 1, 'MEMORY OVER IN "$TALLY" BLOCK.' )
1021:       stop 999
1022:     end if
1023:     call KEEP( 'KTSRF', LKTSRF, 2*NLTEMP, 'I4', LAST, JDEBG )
1024: C
1025:     call TALSUF( IUB, JDEBG, NTSRF, A(LIDSRF), A(LITSRF), MTSRF, A,
1026:     & LKTSRF, A(LKTSRF), NLTEMP, LKTSFJ, NBDY, NSURF,
1027:     & A(LIBSDA), A(LIPSDA), MKBDY, ISFWK(1,1) )
1028: C
1029:     end if
1030: C
1031: C=====
1032: C ... keep working area etc. before input of the first tally
1033: C=====
1034: check %%%%%%%%%
1035: c       write(*,*) '%% lswrk <', lswrk,> '
1036: c %%%%%%%%%
1037:       if ( LSWRK.eq.0 ) then
1038:         NWORK = MAX(NGROUP,NTIME,NREG,NTREG,4096,NWORK)
1039:         call KEEP( 'SWRK', LSWRK, NWORK, 'R4D', LAST, JDEBG )
1040:         call GTLAST( LASTPP )

```

src/shared/talinp.f

```

1041: C
1042: C ... create temporary "packet" region.
1043: C
1044:       call REMAIN( 'temp-packet', NLTEMP, 'R4D', LAST )
1045:       if ( NLTEMP.le.0 ) then
1046:         write(IMG,'(1X,A,A)')
1047:         &       'XXX(talinp) No-more memory to store $TALLY block',
1048:         &       ' information !!!'
1049:         call PRSTOP( 1, 'MEMORY OVER IN "$TALLY" BLOCK.' )
1050:         stop 999
1051:       end if
1052: C
1053:       call KEEP( '$TALLY-PACKET', KLPT, NLTEMP, 'R4D', LAST, JDEBG )
1054: C
1055:       call PACK00( A(KLPT), '$TALLY', NLTEMP, IRET )
1056: check %%%%%%%%%
1057: c      write(*,*) '%% klpt <' klpt, '> ', LSIZ(klpt)
1058: c %%%%%%%%%
1059: C
1060:       if ( IRET.ne.0 ) then
1061:         write(IMG,'(1X,A,A)')
1062:         &       'XXX(talinp) Insufficient memory to make data',
1063:         &       ' packet area to process $TALLY block information',
1064:         &       ' !!!'
1065:         call PRSTOP( 1, 'MEMORY SHORTAGE IN "$TALLY" BLOCK.' )
1066:         stop 999
1067:       end if
1068: C
1069:       end if
1070: C=====
1071: C ==== data for each special tally ====
1072: C=====
1073: C
1074: C ... ID ...
1075: C
1076:       if ( VNAME(:NLEN).eq.'ID' ) then
1077:         if ( KJID.ne.0 ) then
1078:           write(IMG,'(1X,A)') 'XXX(talinp) Duplicate "ID" input.'
1079:           call CNTERR( 'FATAL' )
1080:           NERR = NERR + 1
1081:         end if
1082:         call I4READ( VNAME(:NLEN), KJID, NA, -1, IERR )
1083:         PLAB = TAG(:LTAG) //'ID'
1084:         call PACKLB( A(KLPT), ' ', PLAB(:ICLEN2(PLAB)), IRET )
1085:         call PACKND( A(KLPT), 'ID', 'I4', KJID, 1, IRET )
1086: C
1087: C ... LABEL ...
1088: C
1089:       else if ( VNAME(:3).eq.'LAB' ) then
1090:         if ( TLABEL.ne.' ' ) then
1091:           write(IMG,'(1X,A)') 'XXX(talinp) Duplicate "LABEL" input.'
1092:           call CNTERR( 'FATAL' )
1093:           NERR = NERR + 1
1094:         end if
1095:         IL = 0
1096: C
1097:       130 call CHREAD( VNAME(:NLEN), TLABEL(IL+1:), ' ', NL, ITERM, IEND
1098:       &
1099:         if ( IEND.ne.0 ) go to 9001
1100:         IL = MIN(LEN(TLABEL),IL+NL+1)
1101:         if ( ITERM.eq.0 ) go to 130
1102: C
1103:         PLAB = TAG(:LTAG) //'LABEL'
1104:         call PACKLB( A(KLPT), ' ', PLAB(:ICLEN2(PLAB)), IRET )
1105:         call PACKCS( A(KLPT), ' ', TLABEL(:IL), IRET )

```

```

1106: C
1107:       LLABEL = IL
1108: C
1109: C ... NEUTRON/PHOTON ...
1110: C
1111:       else if ( VNAME(:4).eq.'NEUT' .or. VNAME(:4).eq.'PHOT' ) then
1112:         if ( PARTCL.ne.' ' ) then
1113:           write(IMG,'(1X,A)')
1114:           &       'XXX(talinp) Duplicate NEUTRON/PHOTON input.'
1115:           call CNTERR( 'FATAL' )
1116:           NERR = NERR + 1
1117:         end if
1118:         PARTCL = VNAME(1:NLEN)
1119: C
1120: C ... PARTICLE( particle-species-string ) ...
1121: C
1122:       else if ( VNAME(:NLEN).eq.'PARTICLE' .or. VNAME(:NLEN).eq.'PART' )
1123:       &       then
1124:         if ( PARTCL.ne.' ' ) then
1125:           write(IMG,'(1X,A)')
1126:           &       'XXX(talinp) Duplicate "PARTICLE(...)" input.'
1127:           call CNTERR( 'FATAL' )
1128:           NERR = NERR + 1
1129:         end if
1130:       140 call CHREAD( VNAME(:NLEN), PARTCL, '()', NL, ITERM, IEND )
1131:         if ( IEND.ne.0 ) go to 9001
1132: C ... skip "(" of "PARTICLE(" ....
1133:         if ( ITERM.eq.1.and.NL.eq.0 ) go to 140
1134: C
1135: C ... EVENT ...
1136: C
1137:       else if ( VNAME(:3).eq.'EVE' ) then
1138:         if ( EVENTN.ne.' ' ) then
1139:           write(IMG,'(1X,A)') 'XXX(talinp) Duplicate "EVENT" input.'
1140:           call CNTERR( 'FATAL' )
1141:           NERR = NERR + 1
1142:         end if
1143: C
1144:       150 call CHREAD( VNAME(:NLEN), EVENTN, '()', NL, ITERM, IEND )
1145:         if ( IEND.ne.0 ) go to 9001
1146: check %%%%%%%%%
1147: c      write(*,*) '%%event <' EVENTN(:NL), '> item ', item
1148: c %%%%%%%%%
1149: C ... skip "(" of "EVENT(" ....
1150:         if ( ITERM.eq.1.and.NL.eq.0 ) go to 150
1151: C
1152: C
1153:         PLAB = TAG(:LTAG) //'EVENT'
1154:         call PACKLB( A(KLPT), 'EVENT', PLAB(:ICLEN2(PLAB)), IRET )
1155:         call PACKCS( A(KLPT), ' ', EVENTN(:NL), IRET )
1156: C
1157:         ... SURFACE[:CURRENT]:FLUX( surface-ID ) ...
1158: C
1159:       if ( EVENTN(1:4).eq.'SURF' ) then
1160: C
1161:         JKK = 0
1162:         KK = INDEX(EVENTN(:NL),':')
1163:         if ( KK.ne.0 ) then
1164:           if ( EVENTN(KK+1:NLEN).eq.'CURRENT' ) then
1165:             JKK = 1
1166:           else if ( EVENTN(KK+1:NLEN).eq.'FLUX' ) then
1167:             JKK = 2
1168:           else
1169:             write(IMG,'(1X,A,A,A)')
1170:             &       'XXX(talinp) "SURFACE(...)" in EVENT()',

```

src/shared/talinp.f

```

1171:      &          ' has invalid form of "SURFACE:XXX" : ',
1172:      &          EVENTN(:NL)
1173:      NERR      = NERR + 1
1174:      end if
1175:      end if
1176:      call I4READ( 'SURFACE', ISURF, NA, -1, IERR )
1177:      if ( NA.ne.1 ) then
1178:          write(IMG, '(1X,A,A)')
1179:          &          'XXX(talinp) "SURFACE(...)" in EVENT()',
1180:          &          ' has too many items or lacks closing ")"..'
1181:          NERR      = NERR + 1
1182:      end if
1183: C
1184:      if ( JKK.eq.1 ) then
1185:          ISURF      = ABS(ISURF)
1186:      else if ( JKK.eq.2 ) then
1187:          ISURF      = -ABS(ISURF)
1188:      end if
1189: C
1190:      call PACKND( A(KLPT), 'SURFACE ID', 'I4', ISURF, 1, IRET )
1191: C
1192:      ... skip ")" of "EVENT( .... )"
1193:      call CHREAD( VNAME(:NLEN), EVENTN, ' ', NL, ITERM, IEND )
1194:      if ( IEND.ne.0 ) go to 9001
1195: C
1196:      ... events other than SURFACE ...
1197:      else
1198:          if ( ITERM.eq.0 ) then
1199:              NERR      = NERR + 1
1200:              write(IMG, '(1X,A,A)') 'XXX(talinp) "EVENT" data ',
1201:              &          'has too many items or lacks closing ")"..'
1202:              call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1203:              if ( IEND.ne.0 ) go to 9004
1204:          end if
1205:      end if
1206: C
1207: C ... DIMENSION ...
1208: C
1209:      else if ( VNAME(:3).eq.'DIM' ) then
1210:          if ( NDIM.ne.0 ) then
1211:              write(IMG, '(1X,A)')
1212:              &          'XXX(talinp) Duplicate "DIMENSION" input.'
1213:              call CNTERR( 'FATAL' )
1214:              NERR      = NERR + 1
1215:          end if
1216:          NDIM      = 0
1217:          PLAB      = TAG(:LTAG) //'DIMENSION'
1218:          call PACKLB( A(KLPT), 'DIMENSION', PLAB(:ICLEN2(PLAB)), IRET )
1219: C
1220: 160      continue
1221:          TALDIM      = ' '
1222:          call CHREAD( VNAME(:NLEN), TALDIM, ' ', NL, ITERM, IEND )
1223:          if ( IEND.ne.0 ) go to 9003
1224:          NDIM      = NDIM + 1
1225:          call PACKCS( A(KLPT), 'DIM', TALDIM(:NL), IRET )
1226:          if ( ITERM.eq.0 ) go to 160
1227: C
1228:          PLAB      = TAG(:LTAG) //'END-DIMENSION'
1229:          call PACKLB( A(KLPT), 'END DIM', PLAB(:ICLEN2(PLAB)), IRET )
1230: C
1231: C ... REGION ...
1232: C
1233:      else if ( VNAME(:3).eq.'REG'
1234:      &          .or. IMATCH('TAL*-REG*',VNAME(:NLEN)).eq.1 ) then
1235:          if ( KJREG.ne.0 ) then
1236:              write(IMG, '(1X,A,A)') 'XXX(talinp) Duplicate "REGION" ',
1237:              &          'or "TALLY-REGION" input.'
1238:              call CNTERR( 'FATAL' )
1239:              NERR      = NERR + 1
1240:          end if
1241: C
1242:          PLAB      = TAG(:LTAG) //'REGION'
1243:          call PACKLB( A(KLPT), 'REGION', PLAB(:ICLEN2(PLAB)), IRET )
1244:          KJREG      = 0
1245: C
1246: 170      continue
1247:          call CHREAD( VNAME(:NLEN), CWRK, ' ', NL, ITERM, IEND )
1248:          if ( IEND.ne.0 ) go to 9003
1249:          KJREG      = KJREG + 1
1250:          call PACKCS( A(KLPT), 'REG', CWRK(:NL), IRET )
1251:          if ( ITERM.eq.0 ) go to 170
1252: C
1253:          PLAB      = TAG(:LTAG) //'END-REGION'
1254:          call PACKLB( A(KLPT), 'END REG', PLAB(:ICLEN2(PLAB)), IRET )
1255:          call PACKND( A(KLPT), 'REG BIN#', 'I4', KJREG, 1, IRET )
1256: C
1257: C ... TIME ...
1258: C
1259:      else if ( VNAME(:NLEN).eq.'TIME' .or. VNAME(:NLEN).eq.'ITIME'
1260:      &          .or. VNAME(:NLEN).eq.'ITIME2' ) then
1261:          if ( KJTIME.ne.0 ) then
1262:              write(IMG, '(1X,A,A)') 'XXX(talinp) Duplicate time bin input.'
1263:              call CNTERR( 'FATAL' )
1264:              NERR      = NERR + 1
1265:          end if
1266: C
1267: C
1268: C ... direct specification of time bin ...
1269: C
1270: C ---- currently not in use ...
1271:      if ( VNAME(:NLEN).eq.'TIME' ) then
1272: Cccc
1273:          KJTIME      = -1
1274:          write(IMG, '(1X,A,A)')
1275:          &          'XXX(talinp) Sorry, time bin input for each',
1276:          &          ' tally (TIME(...)) is currently not supported !'
1277:          call CNTERR( 'FATAL' )
1278:          NERR      = NERR + 1
1279: C
1280: C ... common time bin -> tallied time bin ...
1281:      else if ( VNAME(:NLEN).eq.'ITIME' ) then
1282:          KJTIME      = 1
1283:          call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NTIME, IERR )
1284:          if ( IERR.ne.0 ) go to 9001
1285:          if ( NA.ne.NTIME ) then
1286:              write(IMG, '(1X,A,A,A,I5,A)')
1287:              &          'XXX(talinp) Data <', VNAME(:NLEN), '> ',
1288:              &          'must have ', NTIME, ' (=NTIME) items.'
1289:              call CNTERR( 'FATAL' )
1290:              NERR      = NERR + 1
1291:          end if
1292:          PLAB      = TAG(:LTAG) //'TIME'
1293:          call PACKLB( A(KLPT), 'TIME BIN', PLAB(:ICLEN2(PLAB)), IRET )
1294:          &
1295:          call PACKCS( A(KLPT), 'TIME', 'ITIME', IRET )
1296:          call PACKND( A(KLPT), 'ITIME', 'I4', IA(LSWRK), NTIME, IRET )
1297:          &
1298: C
1299: C ... ranges in common time bin -> tallied time bin ...
1300:      else if ( VNAME(:NLEN).eq.'ITIME2' ) then

```

src/shared/talinp.f

```

1301:      KJTIME = 2
1302:      call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1303:      if ( NA.gt.NWORK ) go to 9002
1304:      if ( IERR.ne.0 ) go to 9001
1305:      if ( MOD(NA,2).ne.0 ) then
1306:        write(IMG,'(1X,A,A,A,I6,A)')
1307:      &      'XXX(talinp) Number of data <',
1308:      &      VNAME(:NLEN),
1309:      &      '> must be an even integer.(NDATA=', NA, ' )'
1310:      call CNTERR( 'FATAL' )
1311:      NERR = NERR + 1
1312:    end if
1313:    PLAB = TAG(:LTAG) //'TIME'
1314:    call PACKLB( A(KLPT), 'TIME BIN', PLAB(:ICLEN2(PLAB)), IRET
1315:    &      )
1316:    call PACKCS( A(KLPT), 'TIME BIN', 'ITIME2', IRET )
1317:    call PACKND( A(KLPT), 'ITIME2', 'I4', IA(LSWRK), NA, IRET )
1318:  end if
1319: C
1320: C ... ENERGY ...
1321: C
1322:   else if ( VNAME(:NLEN).eq.'ENERGY' .or. VNAME(:NLEN).eq.'IENERGY'
1323:   &      .or. VNAME(:NLEN).eq.'IENERGY2' ) then
1324:     if ( KJENG.ne.0 ) then
1325:       write(IMG,'(1X,A,A,I4,A)')
1326:       &      'XXX(talinp) Duplicate "ENERGY" input ',
1327:       &      'for ', ITAL, ''th tally.'
1328:       call CNTERR( 'FATAL' )
1329:       NERR = NERR + 1
1330:     end if
1331: C
1332: C
1333: C ... direct specification of energy bin ...
1334:   if ( VNAME(:NLEN).eq.'ENERGY' ) then
1335: C ---- currently not in use ...
1336: Cccc      KJENG = -1
1337: C
1338: C ... common energy bin -> tallied energy bin ...
1339:   else if ( VNAME(:NLEN).eq.'IENERGY' ) then
1340:     KJENG = 1
1341: C
1342: C
1343:     NNG = NGROUP
1344:     JNNER = 0
1345:     IKPID = 1
1346:     if ( PARTCL.eq.' ' ) then
1347:       if ( NPKIND.eq.1 ) then
1348:         IKPID = KKPID
1349:       else
1350:         write(IMG,'(1X,A,A,I4,A)')
1351:         &      'XXX(talinp) No particle type is',
1352:         &      ' specified before IENERGY() for ', ITAL,
1353:         &      ''th tally.'
1354:         call CNTERR( 'FATAL' )
1355:         NERR = NERR + 1
1356:         JNNER = 1
1357:       end if
1358:     else
1359:       call KPSYMB( PARTCL, '>', IKPID, 0 )
1360:       if ( IKPID.eq.0 ) then
1361:         write(IMG,'(1X,A,A,A,I4,A)')
1362:         &      'XXX(talinp) Unsupported particle type <',
1363:         &      PARTCL(:ICLEN2(PARTCL)),
1364:         &      '> is specified before IENERGY() for ', ITAL,
1365:         &      ''th tally.'

```

```

1366:         call CNTERR( 'FATAL' )
1367:         NERR = NERR + 1
1368:         JNNER = 1
1369:       else
1370:         if ( JKPAR(IKPID).ne.0 ) then
1371:           NNG = NGP(IKPID)
1372:         else
1373:           write(IMG,'(1X,A,A,A,A,I4,A)')
1374:           &      'XXX(talinp) unused particle type ',
1375:           &      PARTCL(:ICLEN2(PARTCL)), '> is specified ',
1376:           &      'before IENERGY() for ', ITAL,
1377:           &      ''th tally.'
1378:           call CNTERR( 'FATAL' )
1379:           NERR = NERR + 1
1380:           JNNER = 1
1381:         end if
1382:       end if
1383:     end if
1384:   if ( PARTCL(1:4).eq.'NEUT' ) then
1385: C
1386: C     NNG = NGP1
1387: C   else if ( PARTCL(1:4).eq.'PHOT' ) then
1388: C     NNG = NGP2
1389: C   else if ( JNP.ne.0 ) then
1390: C     write(IMG,*) 'XXX($TALLY) No valid particle type is ',
1391: C     &      'specified before IENERGY() ',
1392: C     &      'for ', ITAL, ''th tally.'
1393: C     call CNTERR( 'FATAL' )
1394: C     NERR = NERR + 1
1395: C     JNNER = 1
1396: C   end if
1397: C
1398:   call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NNG, IERR )
1399:   if ( IERR.ne.0 ) go to 9001
1400: C
1401:   if ( NA.ne.NNG.and.JNNER.eq.0 ) then
1402:     write(IMG,'(1X,A,I6,A,I4,A)')
1403:     &      'XXX(talinp) Number of IENERGY() data must be ',
1404:     &      NNG, ' for ', ITAL, ''th tally.'
1405:     call CNTERR( 'FATAL' )
1406:     NERR = NERR + 1
1407:   end if
1408: C
1409:   PLAB = TAG(:LTAG) //'ENERGY'
1410:   call PACKLB( A(KLPT), 'ENERGY BIN', PLAB(:ICLEN2(PLAB)),
1411:   &      IRET )
1412:   call PACKCS( A(KLPT), 'ENERGY BIN', 'IENERGY', IRET )
1413:   call PACKND( A(KLPT), 'IENERGY', 'I4', IA(LSWRK),
1414:   &      MIN(NA,NWORK), IRET )
1415: C
1416: C ... ranges in common energy bin -> tallied energy bin ...
1417:   else if ( VNAME(:NLEN).eq.'IENERGY2' ) then
1418:     KJENG = 2
1419: C
1420:   call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1421:   if ( NA.gt.NWORK ) go to 9002
1422:   if ( IERR.ne.0 ) go to 9001
1423: C
1424:   if ( MOD(NA,2).ne.0 ) then
1425:     write(IMG,'(1X,A,A,A,I6,A)')
1426:     &      'XXX(talinp) Number of data <',
1427:     &      VNAME(:NLEN),
1428:     &      '> must be an even integer.(NDATA=', NA, ' )'
1429:     call CNTERR( 'FATAL' )
1430:     NERR = NERR + 1

```


src/shared/talinp.f

```

1431:         end if
1432:         PLAB = TAG(:LTAG) //'ENERGY'
1433:         call PACKLB( A(KLPT), 'ENERGY BIN', PLAB(:ICLEN2(PLAB))),
1434:         &
1435:         call PACKCS( A(KLPT), 'ENERGY BIN', 'IENERGY2', IRET )
1436:         call PACKND( A(KLPT), 'IENERGY2', 'I4', IA(LSWRK), NA, IRET
1437:         &
1438:         end if
1439: C
1440: C ... ANGLE ...
1441: C
1442:         else if ( VNAME(:NLEN).eq.'ANGLE' .or. VNAME(:NLEN).eq.'IANGLE'
1443:         & .or. VNAME(:NLEN).eq.'IANGLE2' ) then
1444:
1445:         if ( KJANG.ne.0 ) then
1446:         write(IMG, '(1X,A)')
1447:         & 'XXX(talinp) Duplicate angle bin input.'
1448:         call CNTERR( 'FATAL' )
1449:         NERR = NERR + 1
1450:         end if
1451: C
1452: C ... direct specification of time bin ...
1453: C ---- currently not in use ...
1454:         if ( VNAME(:NLEN).eq.'ANGLE' ) then
1455:         Cccc
1456:         KJTIME = -1
1457:         write(IMG, '(1X,A,A)')
1458:         & 'XXX(talinp) Sorry, angle bin input for each',
1459:         & 'tally (ANGLE(...)) is currently not supported !'
1460:         call CNTERR( 'FATAL' )
1461:         NERR = NERR + 1
1462: C
1463:         ... common time bin -> tallied time bin ...
1464:         else if ( VNAME(:NLEN).eq.'IANGLE' ) then
1465:         KJANG = 1
1466:         call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NANGLE, IERR )
1467:         if ( IERR.ne.0 ) go to 9001
1468:         if ( NA.ne.NANGLE ) then
1469:         write(IMG, '(1X,A,A,A,i5,A)')
1470:         & 'XXX(talinp) Data <', VNAME(:NLEN),
1471:         & '> must have ', NANGLE, ' (=NANGLE) items.'
1472:         call CNTERR( 'FATAL' )
1473:         NERR = NERR + 1
1474:         end if
1475: C
1476: C check %%%%%%%%%
1477: C write(*,*) '%%% nangle ', nangle
1478: C write(*,*) '%%% klpt <', klpt, '> ', LSIZ(klpt)
1479: C %%%%%%%%%
1480:         PLAB = TAG(:LTAG) //'ANGLE'
1481:         call PACKLB( A(KLPT), 'ANGLE BIN', PLAB(:ICLEN2(PLAB))), IRET
1482:         &
1483:         call PACKCS( A(KLPT), 'ANGLE', 'IANGLE', IRET )
1484:         call PACKND( A(KLPT), 'IANGLE', 'I4', IA(LSWRK), NANGLE,
1485:         & IRET )
1486: C
1487: C ... ranges in common time bin -> tallied time bin ...
1488:         else if ( VNAME(:NLEN).eq.'IANGLE2' ) then
1489:         KJANG = 2
1490:         call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1491:         if ( NA.gt.NWORK ) go to 9002
1492:         if ( IERR.ne.0 ) go to 9001
1493:         if ( MOD(NA,2).ne.0 ) then
1494:         write(IMG, '(1X,A,A,A,i6,A)')
1495:         & 'XXX(talinp) Number of data <',

```

```

1496:         & VNAME(:NLEN),
1497:         & '> must be an even integer.(NDATA=', NA, ')')
1498:         call CNTERR( 'FATAL' )
1499:         NERR = NERR + 1
1500:         end if
1501:         PLAB = TAG(:LTAG) //'ANGLE'
1502:         call PACKLB( A(KLPT), 'ANGLE BIN', PLAB(:ICLEN2(PLAB))), IRET
1503:         &
1504:         call PACKCS( A(KLPT), 'ANGLE BIN', 'IANGLE2', IRET )
1505:         call PACKND( A(KLPT), 'IANGLE2', 'I4', IA(LSWRK), NA, IRET )
1506:         end if
1507: C
1508: C ... GENERATION ...
1509: C
1510:         else if ( VNAME(:NLEN).eq.'GENERATION'
1511:         & .or. VNAME(:NLEN).eq.'IGENE2' ) then
1512:         if ( KJGENE.ne.0 ) then
1513:         write(IMG, '(1X,A,A,i5,A)')
1514:         & 'XXX(talinp) Duplicate "GENERATION" input ',
1515:         & 'for ', ITAL, "'th tally.'"
1516:         call CNTERR( 'FATAL' )
1517:         NERR = NERR + 1
1518:         end if
1519: C
1520: C ... generation of particle is added to bank parameter
1521: C
1522:         KIBNK(4) = 1
1523: C
1524: C ... particle generation ...
1525:         if ( VNAME(:NLEN).eq.'GENERATION' ) then
1526:         KJGENE = 1
1527:         call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1528:         if ( NA.gt.NWORK ) go to 9002
1529:         if ( IERR.ne.0 ) go to 9001
1530: C
1531:         PLAB = TAG(:LTAG) //'GENERATION'
1532:         call PACKLB( A(KLPT), 'GEN BIN', PLAB(:ICLEN2(PLAB))), IRET )
1533:         call PACKCS( A(KLPT), 'GEN', 'GENERATION', IRET )
1534:         call PACKND( A(KLPT), 'GEN', 'I4', IA(LSWRK), NA, IRET )
1535: C
1536: C ... ranges in common time bin -> tallied time bin ...
1537:         else if ( VNAME(:NLEN).eq.'IGENE2' ) then
1538:         KJGENE = 2
1539:         call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1540:         if ( NA.gt.NWORK ) go to 9002
1541:         if ( IERR.ne.0 ) go to 9001
1542:         if ( MOD(NA,2).ne.0 ) then
1543:         write(IMG, '(1X,A,A,A,i6,A)')
1544:         & 'XXX(talinp) Number of data <',
1545:         & VNAME(:NLEN),
1546:         & '> must be an even integer.(NDATA=', NA, ')')
1547:         call CNTERR( 'FATAL' )
1548:         NERR = NERR + 1
1549:         end if
1550:         PLAB = TAG(:LTAG) //'GENERATION'
1551:         call PACKLB( A(KLPT), 'GEN BIN', PLAB(:ICLEN2(PLAB))), IRET )
1552:         call PACKCS( A(KLPT), 'TGEN BIN', 'IGENE2', IRET )
1553:         call PACKND( A(KLPT), 'IGENE2', 'I4', IA(LSWRK), NA, IRET )
1554:         end if
1555: C
1556: C ... SOURCE ...
1557: C
1558:         else if ( VNAME(:NLEN).eq.'SOURCE' ) then
1559:         if ( KJSOU.ne.0 ) then
1560:         write(IMG, '(1X,A,i5,A)')

```

src/shared/talinp.f

```

1561:      &          'XXX(talinp) Duplicate "SOURCE" input for ',
1562:      &          ITAL, ''th tally.'
1563:      call CNTERR( 'FATAL' )
1564:      NERR      = NERR + 1
1565:      end if
1566: C
1567: C    ... generated source set # is added to bank parameter
1568: C
1569:      KIBNK(1)   = 1
1570: C
1571:      KJSQU      = 1
1572:      call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1573:      if ( NA.gt.NWORK ) go to 9002
1574:      if ( IERR.ne.0 ) go to 9001
1575:      PLAB       = TAG(:LTAG) //'SOURCE'
1576:      call PACKLB( A(KLPT), 'SRC BIN', PLAB(:ICLEN2(PLAB)), IRET )
1577:      call PACKND( A(KLPT), 'SRCID', 'I4', IA(LSWRK), NA, IRET )
1578: C
1579: C    ... SOURCE-REGION ...
1580: C
1581:      else if ( IMATCH('SOU*-REG*',VNAME(:NLEN)).eq.1 ) then
1582:      if ( KJSRG.ne.0 ) then
1583:      write(IMG,'(1X,A,A,I5,A)')
1584:      &          'XXX(talinp) Duplicate "SOURCE-REGION" input ',
1585:      &          'for ', ITAL, ''th tally.'
1586:      call CNTERR( 'FATAL' )
1587:      NERR      = NERR + 1
1588:      end if
1589: C
1590: C    ... generated region # is added to bank parameter
1591: C
1592:      KIBNK(2)   = 1
1593: C
1594: C
1595:      PLAB       = TAG(:LTAG) //'SOURCE-REGION'
1596:      call PACKLB( A(KLPT), 'SOURCE-REGION', PLAB(:ICLEN2(PLAB)),
1597:      &          IRET )
1598:      KJSRG      = 0
1599: C
1600: 180  continue
1601:      call CHREAD( VNAME(:NLEN), CWRK, ' '), NL, ITERM, IEND )
1602:      if ( IEND.ne.0 ) go to 9003
1603:      KJSRG      = KJSRG + 1
1604:      call PACKCS( A(KLPT), 'REG', CWRK(:NL), IRET )
1605:      if ( ITERM.eq.0 ) go to 180
1606: C
1607:      PLAB       = TAG(:LTAG) //'END-REGION'
1608:      call PACKLB( A(KLPT), 'END REG', PLAB(:ICLEN2(PLAB)), IRET )
1609:      call PACKND( A(KLPT), 'REG BIN#', 'I4', KJSRG, 1, IRET )
1610: C
1611: C    ... MARKER-REGION ...
1612: C
1613:      else if ( IMATCH('MAR*-REG*',VNAME(:NLEN)).eq.1 ) then
1614:      if ( KJMRG.ne.0 ) then
1615:      write(IMG,'(1X,A,A,I5,A)')
1616:      &          'XXX(talinp) Duplicate "MARKER-REGION" input for ',
1617:      &          ITAL, ''th tally.'
1618:      call CNTERR( 'FATAL' )
1619:      NERR      = NERR + 1
1620:      end if
1621: C
1622: C    ... passed flag for marker region is added to bank parameter
1623: C
1624:      KIBNK(9)   = 1
1625: C

```

```

1626: C    ... use flight counter to check a particle has "previous region"
1627: C    when entering next zone.
1628: C    If a particle is generated as source in lattice zone,
1629: C    its actual region # is determined in next-zone search
1630: C    phase, and the particle has no "previous region" in the case.
1631: C
1632:      KIBNK(5)   = 1
1633: C
1634:      PLAB       = TAG(:LTAG) //'MARKER-REGION'
1635:      call PACKLB( A(KLPT), 'MARKER-REGION', PLAB(:ICLEN2(PLAB)),
1636:      &          IRET )
1637:      KJMRG      = 0
1638: C
1639: 190  continue
1640:      call CHREAD( VNAME(:NLEN), CWRK, ' '), NL, ITERM, IEND )
1641:      if ( IEND.ne.0 ) go to 9003
1642:      KJMRG      = KJMRG + 1
1643:      call PACKCS( A(KLPT), 'REG', CWRK(:NL), IRET )
1644:      if ( ITERM.eq.0 ) go to 190
1645: C
1646:      PLAB       = TAG(:LTAG) //'END-REGION'
1647:      call PACKLB( A(KLPT), 'END REG', PLAB(:ICLEN2(PLAB)), IRET )
1648:      call PACKND( A(KLPT), 'REG BIN#', 'I4', KJMRG, 1, IRET )
1649: C
1650: C    ... POINT ...
1651: C
1652:      else if ( VNAME(:NLEN).eq.'POINT'
1653:      &          .or. VNAME(:NLEN).eq.'POINTX' ) then
1654: C
1655:      if ( KJREG.ne.0 ) then
1656:      write(IPR,'(1X,A)') 'XXX(talinp) Duplicate spatial input.'
1657:      call CNTERR( 'FATAL' )
1658:      NERR      = NERR + 1
1659:      end if
1660: C
1661: C    ... direct specification of point detector data ...
1662: C    ---- currently not in use ...
1663: C
1664:      if ( VNAME(:NLEN).eq.'POINTX' ) then
1665: Ccccc
1666:      KJPNT      = -1
1667:      write(IPR,'(1X,A,A)')
1668:      &          'XXX(talinp) Sorry, point input for each',
1669:      &          'tally (POINTX(...)) is currently not supported !'
1670:      call CNTERR( 'FATAL' )
1671:      NERR      = NERR + 1
1672: C
1673: C    ... common point detectors -> tallied point detector ...
1674: C
1675:      else if ( VNAME(:NLEN).eq.'POINT' ) then
1676:      KJREG      = 1
1677:      KJPNT      = 1
1678:      JJPNT      = 1
1679:      call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NPDET, IERR )
1680:      if ( IERR.ne.0 ) go to 9001
1681:      if ( NA.gt.NPDET ) then
1682:      write(IPR,'(1X,A,A,A,I6,A)')
1683:      &          'XXX(talinp) Data <', VNAME(:NLEN),
1684:      &          '> must be less than NPDET (',NPDET,')'
1685:      call CNTERR( 'FATAL' )
1686:      NERR      = NERR + 1
1687:      end if
1688: C
1689:      PLAB       = TAG(:LTAG) //'POINT'
1690:      call PACKLB( A(KLPT), 'POINT', PLAB(:ICLEN2(PLAB)), IRET
1691:      &          )
1692:      call PACKCS( A(KLPT), 'POINT', 'POINT', IRET )

```

src/shared/talinp.f

```

1691:      call PACKND( A(KLPT), 'POINT', 'I4', IA(LSWRK), NA, IRET
1692:      &
1693:      end if
1694: c##<2007/03/14:PN3:
1695: C
1696: C ... PRODUCE-REGION ...
1697: C
1698:      else if ( IMATCH('PRO*-REG*',VNAME(:NLEN)).eq.1 ) then
1699:          if ( KJPRG.ne.0 ) then
1700:              write(IMG,9901) '"PRODUCE-REGION"', ITAL
1701:              call CNTERR( 'FATAL' )
1702:              NERR = NERR + 1
1703:          end if
1704: C
1705: C ... generated region # is added to bank parameter
1706:      KIBNK(17) = 1
1707: C
1708:      PLAB = TAG(:LTAG) // 'PRODUCE-REGION'
1709:      call PACKLB( A(KLPT), 'PRODUCE-REGION', PLAB(:ICLEN2(PLAB)),
1710:      &
1711:      IRET )
1712:      KJPRG = 0
1713:      continue
1714:      call CHREAD( VNAME(:NLEN), CWRK, ' ', NL, ITERM, IEND )
1715:      if ( IEND.ne.0 ) go to 9003
1716:      KJPRG = KJPRG + 1
1717:      call PACKCS( A(KLPT), 'REG', CWRK(:NL), IRET )
1718:      if ( ITERM.eq.0 ) go to 210
1719:      PLAB = TAG(:LTAG) // 'END-REGION'
1720:      call PACKLB( A(KLPT), 'END REG', PLAB(:ICLEN2(PLAB)), IRET )
1721:      call PACKND( A(KLPT), 'REG BIN#', 'I4', KJPRG, 1, IRET )
1722: C
1723: C ... PRODUCE-NUCLIDE ...
1724: C
1725:      else if ( IMATCH('PRO*-NUC*',VNAME(:NLEN)).eq.1 ) then
1726:          if ( KJPRN.ne.0 ) then
1727:              write(IMG,9901) '"PRODUCE-NUCLIDE"', ITAL
1728:              call CNTERR( 'FATAL' )
1729:              NERR = NERR + 1
1730:          end if
1731: C
1732: C ... generated nuclide # is added to bank parameter
1733:      KIBNK(18) = 1
1734: C
1735:      PLAB = TAG(:LTAG) // 'PRODUCE-NUCLIDE'
1736:      call PACKLB( A(KLPT), 'PRODUCE-NUCLIDE', PLAB(:ICLEN2(PLAB)),
1737:      &
1738:      IRET )
1739:      KJPRN = 0
1740:      continue
1741:      call CHREAD( VNAME(:NLEN), CWRK, ' ', NL, ITERM, IEND )
1742:      if ( IEND.ne.0 ) go to 9003
1743:      KJPRN = KJPRN + 1
1744:      call PACKCS( A(KLPT), 'NUC', CWRK(:NL), IRET )
1745:      if ( ITERM.eq.0 ) go to 220
1746:      PLAB = TAG(:LTAG) // 'END-NUCLIDE'
1747:      call PACKLB( A(KLPT), 'END NUC', PLAB(:ICLEN2(PLAB)), IRET )
1748:      call PACKND( A(KLPT), 'NUC BIN#', 'I4', KJPRN, 1, IRET )
1749: C
1750: C ... PRODUCE-REACTION ...
1751: C
1752:      else if ( IMATCH('PRO*-REA*',VNAME(:NLEN)).eq.1 ) then
1753:          if ( KJPRR.ne.0 ) then
1754:              write(IMG,9901) '"PRODUCE-REACTION"', ITAL
1755:              call CNTERR( 'FATAL' )
1756:              NERR = NERR + 1
1757:          end if

```

```

1756: C
1757: C ... generated reaction # is added to bank parameter
1758:      KIBNK(19) = 1
1759: C
1760:      KJPRR = 1
1761:      call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -NWORK, IERR )
1762:      if ( NA.gt.NWORK ) go to 9002
1763:      if ( IERR.ne.0 ) go to 9001
1764:      PLAB = TAG(:LTAG) // 'PRODUCE-REACTION'
1765:      call PACKLB( A(KLPT), 'PRR BIN', PLAB(:ICLEN2(PLAB)),
1766:      &
1767:      IRET )
1768:      call PACKCS( A(KLPT), 'PRR', 'PRODUCE-REACTION', IRET )
1769:      call PACKND( A(KLPT), 'PRR', 'I4', IA(LSWRK), NA, IRET )
1770: C
1771: C --- what is tallied ---
1772: C
1773: C ... response ...
1774: C
1775: C
1776:      else if ( VNAME(:NLEN).eq.'IRESP' .or. VNAME(:NLEN).eq.'GRES' )
1777:      &
1778:      then
1779:          KJWHAT = KJWHAT + 1
1780:          call I4READ( VNAME(:NLEN), IR, NA, -1, IERR )
1781:          if ( IERR.ne.0 ) go to 9001
1782:          if ( NRESP.eq.0 .or. IR.lt.0 .or. IR.gt.NRESP ) then
1783:              write(IMG,'(1X,A,I5,A)')
1784:              &
1785:              'XXX(talinp) Specified "IRESP" (=' , IR,
1786:              &
1787:              ') does not match any response.'
1788:              call CNTERR( 'FATAL' )
1789:              NERR = NERR + 1
1790:          end if
1791:          NRESP2 = NRESP2 + 1
1792:          PLAB = TAG(:LTAG) // 'WHAT'
1793:          call PACKLB( A(KLPT), 'RESPONSE', PLAB(:ICLEN2(PLAB)), IRET )
1794:          call PACKCS( A(KLPT), 'RESPONSE #', 'IRESP', IRET )
1795:          call PACKND( A(KLPT), 'IRESP', 'I4', IR, 1, IRET )
1796: C
1797: C ... special response ( from dosimetry file ) ...
1798: C
1799: C
1800:      else if ( VNAME(:NLEN).eq.'PRES' ) then
1801:          KJWHAT = KJWHAT + 1
1802:          call I4READ( VNAME(:NLEN), IR, NA, -1, IERR )
1803:          if ( IERR.ne.0 ) go to 9001
1804:          if ( NSTAL.eq.0 .or. IR.lt.0 .or. IR.gt.NSTAL ) then
1805:              write(IMG,'(1X,A,I6,A)')
1806:              &
1807:              'XXX(talinp) Specified "PRES" (=' , IR,
1808:              &
1809:              ') does not match any response.'
1810:              call CNTERR( 'FATAL' )
1811:              NERR = NERR + 1
1812:          end if
1813:          NSTAL2 = NSTAL2 + 1
1814:          PLAB = TAG(:LTAG) // 'WHAT'
1815:          call PACKLB( A(KLPT), 'RESPONSE', PLAB(:ICLEN2(PLAB)), IRET )
1816:          call PACKCS( A(KLPT), 'RESPONSE #', 'CRES', IRET )
1817:          call PACKND( A(KLPT), 'IRESP', 'I4', IR, 1, IRET )
1818: C
1819: C
1820:      else if ( VNAME(:NLEN).eq.'ERESP' ) then
1821:          KJWHAT = KJWHAT + 1
1822:          NNG = NGROUP
1823:          if ( PARTCL.eq.'NEUT' ) NNG = NGP1
1824:          if ( PARTCL.eq.'PHOT' ) NNG = NGP2
1825:          call R4READ( VNAME(:NLEN), A(LSWRK), NA, -NNG, IERR )
1826:          if ( IERR.ne.0 ) go to 9100

```

src/shared/talinp.f

```

1821: CC      PLAB = tag(:ltag)//'WHAT'
1822: CC      call PACKLB( A(KLPT), 'RESPONSE', PLAB(:iclen2(PLAB)), IRET )
1823: CC      call PACKCS( A(KLPT), 'RESPONSE(G)', 'ERESP', IRET )
1824: CC      call PACKND( A(KLPT), 'ERESP', 'R4', A(LSWRK), MIN(NWORK,NA),
1825: CC      &          IRET )
1826: C
1827: C ... micro reaction rate ...
1828: C
1829:       else if ( IMATCH('MICRO*',VNAME(:NLEN)).eq.1 ) then
1830:           if ( CODE.eq.'MVP' ) then
1831:               KJWHAT = KJWHAT + 1
1832:               KJMIC = 0
1833: C
1834: C
1835:       call CHREAD( VNAME(:NLEN), CWRK, ' )', NL, ITERM, IEND )
1836:       if ( IEND.ne.0 ) go to 9003
1837:       KJMIC = KJMIC + 1
1838: C
1839:       PLAB = TAG(:LTAG) //'WHAT'
1840:       call PACKLB( A(KLPT), 'MICRO', PLAB(:ICLEN2(PLAB)), IRET )
1841:       call PACKCS( A(KLPT), 'MICRO', VNAME(:NLEN), IRET )
1842:       call PACKCS( A(KLPT), 'REACTION', CWRK(:NL), IRET )
1843: C
1844:       if ( ITERM.eq.0 ) then
1845:           NERR = NERR + 1
1846:           write(IMG,'(1X,A,A)') 'XXX(talinp) "MICRO:*" data ',
1847:           &           'has too many items or lacks closing ")"'.
1848:           call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1849:           if ( IEND.ne.0 ) go to 9004
1850:       end if
1851:       else if ( CODE.eq.'GMVP' ) then
1852:           NERR = NERR + 1
1853:           write(IMG,'(1X,A,A)') 'XXX(talinp) "MICRO:*" tally is ',
1854:           &           'not supported in GMVP.")'.
1855:           call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1856:           if ( IEND.ne.0 ) go to 9004
1857:       end if
1858: C
1859: C ... macro reaction rate ...
1860: C
1861:       else if ( IMATCH('MACRO*',VNAME(:NLEN)).eq.1 ) then
1862:           KJWHAT = KJWHAT + 1
1863:           KJMAC = 0
1864: C
1865:       PLAB = TAG(:LTAG) //'WHAT'
1866:       call PACKLB( A(KLPT), 'MACRO', PLAB(:ICLEN2(PLAB)), IRET )
1867:       call PACKCS( A(KLPT), 'MACRO', VNAME(:NLEN), IRET )
1868: C
1869:       call CHREAD( VNAME(:NLEN), CWRK, ' )', NL, ITERM, IEND )
1870:       if ( IEND.ne.0 ) go to 9003
1871:       KJMAC = KJMAC + 1
1872: C
1873:       call PACKCS( A(KLPT), 'REACTION', CWRK(:NL), IRET )
1874: C
1875:       if ( ITERM.eq.0 ) then
1876:           NERR = NERR + 1
1877:           write(IMG,'(1X,A,A)') 'XXX(talinp) "MACRO:*" data ',
1878:           &           'has too many items or lacks closing ")"'.
1879:           call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1880:           if ( IEND.ne.0 ) go to 9004
1881:       end if
1882: c##<2007/03/14:PN3:
1883: C
1884: C ... micro photonuclear reaction rate ...
1885: C

```

```

1886:       else if ( IMATCH('MICROPN*',VNAME(:NLEN)).eq.1 ) then
1887:           if ( CODE.eq.'MVP' ) then
1888:               KJWHAT = KJWHAT + 1
1889:               KJMICPN = 0
1890:               call I4READ( VNAME(:NELN), IA(LSWRK), NA, -NWORK, IERR )
1891:               if ( NA.gt.NWORK ) go to 9002
1892:               if ( IERR.ne.0 ) go to 9001
1893:               KJMICPN = KJMICPN + 1
1894:               PLAB = TAG(:LTAG) //'WHAT'
1895:               call PACKLB( A(KLPT), 'MICROPN', PLAB(:ICLEN2(PLAB)), IRET )
1896:               call PACKCS( A(KLPT), 'MICROPN', VNAME(:NLEN), IRET )
1897:               call PACKND( A(KLPT), 'REACTION', 'I4', IA(LSWRK), NA, IRET)
1898:               NMTMPN = NMTMPN + NA
1899:               if ( ITERM.eq.0 ) then
1900:                   NERR = NERR + 1
1901:                   write(IMG,933) 'MICROPN:*'
1902:                   call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1903:                   if ( IEND.ne.0 ) go to 9004
1904:               end if
1905:           else if ( CODE.eq.'GMVP' ) then
1906:               NERR = NERR + 1
1907:               write(IMG,934) 'MICROPN:*'
1908:               call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1909:               if ( IEND.ne.0 ) go to 9004
1910:           end if
1911: c##>
1912: C
1913: C ... the number of analog events ...
1914: C
1915:       else if ( VNAME(:NLEN).eq.'ANALOG' ) then
1916:           KJWHAT = KJWHAT + 1
1917: C
1918:       PLAB = TAG(:LTAG) //'WHAT'
1919:       call PACKLB( A(KLPT), 'ANALG', PLAB(:ICLEN2(PLAB)), IRET )
1920:       call PACKCS( A(KLPT), 'ANALG', VNAME(:NLEN), IRET )
1921: C
1922:       call CHREAD( VNAME(:NLEN), CWRK, ' )', NL, ITERM, IEND )
1923:       if ( IEND.ne.0 ) go to 9003
1924: C
1925:       call PACKCS( A(KLPT), 'REACTION', CWRK(:NL), IRET )
1926: C
1927:       if ( ITERM.eq.0 ) then
1928:           NERR = NERR + 1
1929:           write(IMG,'(1X,A,A)') 'XXX(talinp) "MACRO:*" data ',
1930:           &           'has too many items or lacks closing ")"'.
1931:           call DMREAD( VNAME(:NLEN), NA, NB, IEND )
1932:           if ( IEND.ne.0 ) go to 9004
1933:       end if
1934: C
1935: C ... Feynman-alpha ...
1936: C
1937:       else if ( VNAME(:NLEN).eq.'FEYNMAN-Y' ) then
1938:           KJWHAT = KJWHAT + 1
1939: C
1940:       call I4READ( VNAME(:NLEN), IA(LSWRK), NA, -2, IERR )
1941:       if ( IERR.ne.0 ) go to 9001
1942:       if ( NA.le.1 ) then
1943:           IA(LSWRK+1) = 0
1944:       end if
1945: C
1946:       PLAB = TAG(:LTAG) //'WHAT'
1947:       call PACKLB( A(KLPT), 'FEYNM', PLAB(:ICLEN2(PLAB)), IRET )
1948:       call PACKCS( A(KLPT), 'FEYNM', VNAME(:NLEN), IRET )
1949:       call PACKND( A(KLPT), 'NGATE', 'I4', IA(LSWRK), 2, IRET )
1950: C

```

src/shared/talinp.f

```

1951: C
1952: C --- multiplication factor ---
1953: C
1954:       else if ( IMATCH('MULT*',VNAME(:NLEN)).eq.1 ) then
1955:           if ( KJMULT.ne.0 ) then
1956:               write(IMG,'(1X,A,A,I5,A)')
1957:               &          'XXX(talinp) Duplicate multiplication factor',
1958:               &          ' input for ', ITAL, 'the tally.'
1959:               call CNTERR( 'FATAL' )
1960:               NERR = NERR + 1
1961:           end if
1962:           KJMULT = 1
1963:           call R4READ( VNAME(:NLEN), A(LSWRK), NA, -NWORK, IERR )
1964:           if ( NA.gt.NWORK ) go to 9002
1965:           if ( IERR.ne.0 ) go to 9001
1966: C
1967:       PLAB = TAG(:LTAG) //'FACTOR'
1968:       call PACKLB( A(KLPT), 'MULT', PLAB(:ICLEN2(PLAB)), IRET )
1969: C
1970:       call PACKCS( A(KLPT), 'MULTIPLICATION', VNAME(:NLEN), IRET )
1971:       call PACKND( A(KLPT), 'MULT', 'R4', A(LSWRK), MIN(NWORK,NA),
1972:       &          IRET )
1973: C
1974: C ... custom (user defined) tally ID (good boys/girls don't use this?)
1975: C
1976:       else if ( VNAME(:NLEN).eq.'USER' .or. VNAME(:NLEN).eq.'CUSTOM' )
1977:       then
1978:           KJWHAT = KJWHAT + 1
1979: CCCCC call I4READ( VNAME(:NLEN), IR, NA, 1, IERR )
1980:           call I4READ( VNAME(:NLEN), IR, NA, -1, IERR )
1981:           if ( IERR.ne.0 ) go to 9001
1982:
1983:           write(IMG,'(1X,A,I5,A,A,A)') '!!!(talinp) "CUSTOM"/"USER(',
1984:           &          IR, ') is effective only in a user customized version',
1985:           &          ' possible to take a non standard tally etc.'
1986:           write(IMG,'(1X,A)')
1987:           &          ' You should know what you are really doing.'
1988:           call CNTERR( 'WARNING' )
1989:
1990:           PLAB = TAG(:LTAG) //'USER'
1991:           call PACKLB( A(KLPT), 'USER', PLAB(:ICLEN2(PLAB)), IRET )
1992:           call PACKND( A(KLPT), 'USER', 'I4', IR, 1, IRET )
1993: C
1994: C .... real variance estimation ....
1995: C
1996:       else if ( VNAME(:8).eq.'REAL-VAR' ) then
1997:           NETRV = NETRV + 1
1998:           PLAB = TAG(:LTAG) //'REAL-VARIANCE'
1999:           call PACKLB( A(KLPT), 'REAL VARIANCE', PLAB(:ICLEN2(PLAB)),
2000:           &          IRET )
2001: C
2002: C=====
2003: C === delimiter of special tally or "$END" ===
2004: C=====
2005: C
2006:       else if ( VNAME(:NLEN).eq.'&' .or. VNAME(:NLEN).eq.'$END' ) then
2007: C
2008: C ... print & check validity of the previous tally ...
2009: C
2010:           if ( ITAL.gt.0 ) then
2011: C
2012: C >>>> ID <<<<
2013:           if ( KJID.ne.0 ) write(IPR,'(3x,' ID : ',i5)') KJID
2014:
2015: C >>>> LABEL <<<<

```

```

2016:           if ( TLABEL.ne.' ' )
2017:           &          write(IPR,'(3x,' LABEL : <'',a,'>')') TLABEL(:LLABEL)
2018:
2019: C >>>> particle <<<<
2020:           IKPID = 0
2021:           if ( PARTCL.eq.' ' ) then
2022:               if ( KKPID.ne.0 ) then
2023:                   IKPID = KKPID
2024: C
2025:                   ... get particle symbol
2026:                   call KPSYMB( PARTCL, '<', IKPID, 0 )
2027:               else
2028:                   write(IMG,'(1X,A)')
2029:                   &          'XXX(talinp) Particle type is not specified.'
2030:                   call CNTERR( 'FATAL' )
2031:                   NERR = NERR + 1
2032:               end if
2033:           else
2034:               call KPSYMB( PARTCL, '>', IKPID, 0 )
2035:               if ( IKPID.eq.0 ) then
2036:                   write(IMG,'(1X,A,A,A)')
2037:                   &          'XXX(talinp) Unsupported particle type <',
2038:                   &          PARTCL(:ICLEN2(PARTCL)), '>'
2039:                   call CNTERR( 'FATAL' )
2040:                   NERR = NERR + 1
2041:               else
2042: C
2043:                   ... convert particle to full name
2044:                   call KPSYMB( PARTCL, '<', IKPID, 0 )
2045:                   if ( JKPAR(IKPID).eq.0 ) then
2046:                       write(IMG,'(1X,A,A,A,A)')
2047:                       &          'XXX(talinp) Particle type <',
2048:                       &          PARTCL(:ICLEN2(PARTCL)),
2049:                       &          '> is specified, but',
2050:                       &          ' this problem does not treat this particle.'
2051:                       call CNTERR( 'FATAL' )
2052:                       NERR = NERR + 1
2053:                   end if
2054:               end if
2055:           end if
2056: C
2057: C
2058: C
2059: C
2060: C
2061: C
2062: C
2063: C
2064: C
2065: C
2066: C
2067: C
2068: C
2069: C
2070: C
2071: C
2072: C
2073: C
2074: C
2075: C
2076: C
2077: C
2078: C
2079: C
2080: C

```

src/shared/talinp.f

```

2081: C      write(IMGMSG,*) 'XXX Invalid particle type <', PARTCL, '>'
2082: C      call CNTERR( 'FATAL' )
2083: C      NERR      = NERR + 1
2084: C      end if
2085:
2086:      write(IPR,'(3x,' ' PARTICLE : <' ,a,'>')') PARTCL
2087: C
2088: C
2089: C      PLAB      = TAG(:LTAG) //'PARTICLE'
2090: C      call PACKLB( A(KLPT), 'PARTICLE', PLAB(:ICLEN2(PLAB)), IRET
2091: C      &
2092: C      call PACKCS( A(KLPT), ' ', PARTCL(:ICLEN(PARTCL)), IRET )
2093: C
2094: C      if ( KJWHAT.gt.1 ) then
2095: C        write(IMGMSG,'(1X,A,A,A)')
2096: C      &      'XXX(talinp) Duplicate specification for what',
2097: C      &      ' to be tallied. You can specify only one of',
2098: C      &      ' IRESP/PRESP/MICRO*/MACRO*.'
2099: C      call CNTERR( 'FATAL' )
2100: C      NERR      = NERR + 1
2101: C      end if
2102: C
2103: C      ... put END label
2104: C
2105: C      THEAD      = TAG(:LTAG) //'&END'
2106: C      call PACKLB( A(KLPT), '&END-TALLY', THEAD, IRET )
2107: C
2108: C >>>> event <<<<
2109: C      if( event.eq.' ' ) then
2110: C        event = 'TRACK'
2111: C        write(ipr,'(3x,' ' (default event <' ,a,'> is used.)')')
2112: C      &      event
2113: C      else if( event(1:4).eq.'TRAC' ) then
2114: C        event = 'TRACK'
2115: C      else if( event(1:4).eq.'COLL' ) then
2116: C        event = 'COLLISION'
2117: C      else if( event(1:4).eq.'SURF' ) then
2118: C        event = 'SURFACE'
2119: C      else if( event(1:4).eq.'NEXT' ) then
2120: C        event = 'NEXTEVENT'
2121: C      else if( event(1:4).eq.'POIN' ) then
2122: C        event = 'POINT'
2123: C      else if( event(1:4).eq.'ANAL' ) then
2124: C        event = 'ANALOG'
2125: C      else if( event(1:4).eq.'LEAK' ) then
2126: C        event = 'LEAKAG'
2127: C      else
2128: C        write(ipr,*) 'XXX invalid tally event type <',event,>'
2129: C        call cnterr('FATAL')
2130: C        nerr = nerr + 1
2131: C      endif
2132: C      write(ipr,'(3x,' ' EVENT : <' ,a,'>')') event
2133: C
2134: C >>>> dimension & related specifications <<<<
2135: C      do 500 k=1,ndim
2136: C      end if
2137: C
2138: C      ... set flags to check certain input item is input or not...
2139: C
2140: C      PARTCL = ' '
2141: C      KJID    = 0
2142: C      TLABEL  = ' '
2143: C      KJREG   = 0
2144: C      KJSOU   = 0
2145: C      KJSRG   = 0

```

```

2146: C      KJMRG   = 0
2147: C      c##<2007/03/14:PN3:
2148: C      KJPRG   = 0
2149: C      KJPRN   = 0
2150: C      KJPRR   = 0
2151: C      KJPNP   = 0
2152: C      c##>
2153: C
2154: C      EVENTN  = ' '
2155: C      KJENG   = 0
2156: C      NDIM    = 0
2157: C      KJTIME  = 0
2158: C      KJGENE  = 0
2159: C      KJANG   = 0
2160: C      KJPNT   = 0
2161: C
2162: C      KJWHAT  = 0
2163: C      KJRESP  = 0
2164: C      KJMAC   = 0
2165: C      KJMIC   = 0
2166: C      KJMULT  = 0
2167: C      c##<2007/03/14:PN3:
2168: C      KJMICPN = 0
2169: C      c##>
2170: C
2171: C      ... terminate input here
2172: C
2173: C      if ( VNAME(:NLEN).eq.'$END' ) go to 200
2174: C
2175: C      ... increment tally # and put header label in temporary packet.
2176: C
2177: C      ITAL    = ITAL + 1
2178: C      write(IPR,7020) ITAL
2179: C      7020    format(/3X,'=== Input special tally No. ',I3,' ===/')
2180: C
2181: C      TAG     = ' '
2182: C      write(TAG,'(i5,' ' .')') ITAL
2183: C      call CCOMP( TAG, LEN(TAG), TAG, IIF )
2184: C      LTAG    = ICLEN(TAG)
2185: C      THEAD   = TAG(:LTAG) //'&TALLY'
2186: C      call PACKLB( A(KLPT), '&TALLY', THEAD(:ICLEN2(THEAD)), IRET )
2187: C
2188: C      === invalid data ===
2189: C
2190: C      else
2191: C        write(IMGMSG,'(1X,A,A,A,A)') 'XXX(talinp) Data <', VNAME(:NLEN),
2192: C      &      '> is not acceptable as special tally data.'
2193: C        write(IMGMSG,'(1X,A,I5,A,I5,A)')
2194: C      &      ' Tally No. ', ITAL, ' ID(', KJID, ') '
2195: C        write(IMGMSG,'(1X,A,A,A)') ' Label <', TLABEL, '>'
2196: C        call CNTERR( 'FATAL' )
2197: C        NERR      = NERR + 1
2198: C        call FPROBE( IIP, ' ', CWRK )
2199: C        if ( IIP.ne.0.and.CWRK(1:1).eq.'(' ) then
2200: C          call DMREAD( VNAME(:NLEN), NA, NB, IEND )
2201: C          if ( IEND.ne.0 ) go to 9004
2202: C        end if
2203: C      end if
2204: C      go to 110
2205: C
2206: C      C=====
2207: C      C===== end of $TALLY block
2208: C      C=====
2209: C
2210: C      200 continue

```

src/shared/talinp.f

```

2211: C
2212:   call RESET
2213: C
2214:   call PCTCLS( A(KLPT), 'CLOSE $TALLY', NPSIZE, NPS, IRET )
2215: C
2216:   if ( JDEBG(1).ne.0 ) then
2217:     call PCTDMP( IPR, A(KLPT), A(KLPT), A(KLPT) )
2218:   end if
2219: C
2220:   call PCTERR( A(KLPT), ' ', NPKERR, IRET )
2221: C
2222:   ... free memory ...
2223:   call RESIZE( 'temporary-packet', KLPT, NPS, 'R4D', LAST )
2224: C
2225:   if ( NERR.gt.0 ) then
2226:     write(IMG, '(1X,A,A)') 'XXX(talinp) $TALLY block contains',
2227:   &   ' error. Skip processing.'
2228:     call CNTERR( 'FATAL' )
2229:     return
2230:   end if
2231: C
2232:   if ( NPKERR.gt.0 ) then
2233:     write(IMG, '(1X,A,A)')
2234:   &   'XXX(talinp) Could not store all special tally data',
2235:   &   ' correctly in memory. Skip processing.'
2236:     call CNTERR( 'FATAL' )
2237:     return
2238:   end if
2239: C
2240:   if ( ITAL.eq.0 ) then
2241:     write(IPR, '(1X,A,A)') '<<MESSAGE from talinp>> No special',
2242:   &   ' tally set is specified in $TALLY block.'
2243:     call CNTERR( 'MESSAGE' )
2244:     NDTALY = 0
2245:     NETALY = 0
2246:     return
2247:   end if
2248: C
2249:   if ( JEIGN.eq.0.and.NETRV.gt.0 ) then
2250:     write(IMG, '(1X,A,A)')
2251:   &   '!!!(talinp) "Real variance" calculation is specified',
2252:   &   ' for some special tallies in non-eigenvalue problem.'
2253:     call CNTERR( 'WARNING' )
2254:   end if
2255: C
2256: C ===== post processing of input data =====
2257: C
2258:   NSTALY = ITAL
2259: C
2260: C ... create marker region index MKREG if necessary ...
2261: C
2262:   NMKREG = 0
2263:   if ( KIBNK(9).ne.0 ) then
2264:     call KEPV( A(1), 'MKREG', LMKREG, NREG*2, 'I4', LAST, 0, JDEBG
2265:   &   )
2266:   end if
2267: C
2268: CCC call KEPV( A(1), 'INCST', LINCST, NUC*NPKIND, 'I4', LAST, 0, JDEBG
2269: call KEPV( A(1), 'INCST', LINCST, NUC*KPLIM, 'I4', LAST, 0, JDEBG
2270: &   )
2271:   if ( JJPNT.ne.0 )
2272: CCC &call KEPV( A(1), 'INCSI', LINCSI, NUC*NPKIND, 'I4', LAST, 0, JDEBG
2273: &call KEPV( A(1), 'INCSI', LINCSI, NUC*KPLIM, 'I4', LAST, 0, JDEBG
2274: &   )
2275: c##<2007/03/14:PN3:

```

```

2276:   ND      = MAX(1,NMTMPN)
2277:   call KEPV( A(1), 'INCSTPN', LINCSTPN, NSTALY*2, 'I4', LAST, 0, JDEBG )
2278:   call KEPV( A(1), 'MTMPN', LMTMPN, ND, 'I4', LAST, 0, JDEBG )
2279: c##>
2280: C
2281: C ... compose direct tally information ...
2282: C
2283: C ... work area
2284:   call KEEP( 'KE2D', LKE2D, ITAL, 'I4', LAST, JDEBG )
2285: C
2286:   call REMAIN( 'IDTAL', NLTEMP, 'I4D', LAST )
2287:   call KEEP( 'IDTAL', LIDTAL, NLTEMP, 'I4D', LAST, JDEBG )
2288: C
2289: C -----
2290:   call TALIN1( CODE, IUG1, NERR, A, CHA, A(KLPT), A(KLPT), NPS,
2291: &   ITAL, A(LIDTAL), NLTEMP, NIDTAL, NDTALY, NETALY,
2292: &   NGP, KPLIM,
2293: &   JNEUT, JPHOT, JTIME, JFISS, JEIGN, JRESP, JTLT, JTSRF,
2294: &   JDEBG,
2295: &   IDSRC, LIDSRC, A(LMKREG), NUC, A(LINCST),
2296: &   JJPNT, A(LINCSI), A(LIDSRF),
2297: c##<2007/03/14:PN3:
2298: c## &   A(LKE2D), A(LSWRK), A(LSWRK), A(LSWRK), NWORK )
2299: &   A(LKE2D), JPHNU, NPATOM, NUCPN, A(LINCSTPN), A(LMTMPN),
2300: &   A(LSWRK), A(LSWRK), A(LSWRK), NWORK )
2301: c##>
2302: C
2303: C -----
2304:   call RESIZE( 'IDTAL', LIDTAL, NIDTAL, 'I4D', LAST )
2305: C
2306:   if ( NERR.gt.0 ) then
2307:     write(IMG, '(1X,A,A)') 'XXX(talinp) $TALLY block contains',
2308:   &   ' error. Skip processing.'
2309:     call CNTERR( 'FATAL' )
2310:     NDTALY = 0
2311:     NETALY = 0
2312:     return
2313:   end if
2314: C
2315:   call KEEP( 'KDTAL', LKDTAL, NDTALY+1, 'I4', LAST, JDEBG )
2316:   call KEEP( 'KETAL', LKETAL, NETALY+1, 'I4', LAST, JDEBG )
2317:   call KEEP( 'JDTRG', LJDTRG, NREG*NDTALY, 'I4', LAST, JDEBG )
2318: C
2319:   if ( NPDET.gt.0 ) then
2320:     if ( JPTDT.ne.0 ) then
2321:       call KEPV( A(1), 'JPUSD', LJPUSD, NPDET, 'I4', LAST, 1,
2322: &   JDEBG )
2323:     else
2324:       call KEPV( A(1), 'JPUSD', LJPUSD, NPDET, 'I4', LAST, 0,
2325: &   JDEBG )
2326:       if ( JJPNT.ne.0 ) then
2327:         JPTDT = 1
2328:       end if
2329:     end if
2330:   end if
2331: C
2332: C ... work area
2333:   call KEEP( 'KIDT', LKIDT, NDTALY, 'I4', LAST, JDEBG )
2334: C
2335: C ... compose edited tally information ...
2336: C
2337:   if ( NETRV.gt.0 ) then
2338:     call KEEP( 'IETRV', LIETRV, 2*(NETRV+1), 'I4', LAST, JDEBG )
2339:   end if
2340: C

```

src/shared/talinp.f

```
2341:      call LRMAIN( 'IETAL', NLTEMP, 'I4D', LLAST )
2342:      call LKEEP( 'IETAL', LIETAL, NLTEMP, 'I4D', LAST, JDEBG )
2343:      call PACK00( H(LIETAL), 'E-TALLY', NLTEMP, IRET )
2344: C
2345: C -----
2346:      call TALIN2( CODE, IUGL, IUG2, NERR, A(KLPT), A(KLPT), NPS, ITAL,
2347: &      H(LIETAL), NLTEMP, NIETAL, A(LKETAL), A(LJDTRG), NREG,
2348: &      NDTALY, NETALY, NEDTMX, NDTWMX, NBINMX, MXRGBN, A(LKDTAL),
2349: &      A(LIDTAL), NIDTAL, NLDTAL, NLETAL, NETRV, METRV,
2350: &      A(LIETRV), A(LJPUSD),
2351: &      JNEUT, JPHOT, JTIME, JFISS, JEIGN, JRESP, JTLLT, JDEBG,
2352: &      A(LSWRK), A(LSWRK), A(LSWRK), NWORK,
2353: &      A(LKE2D), A(LKIDT) )
2354: C -----
2355: C
2356:      call PCTCLS( H(LIETAL), 'CLOSE E-TALLY', NPSIZE, NIETAL, IRET )
2357:      call LRSIZE( 'IETAL', LIETAL, NIETAL, 'I4D', LAST )
2358: C
2359:      if ( JDEBG(1).ne.0 ) then
2360:          call PCTDMP( IPR, H(LIETAL), H(LIETAL), H(LIETAL) )
2361:      end if
2362: C
2363: C
2364: C ... make event / d-tally table
2365: C
2366:      NTEVE = 12
2367:      call KEEP( 'JTEVE', LJTEVE, NTEVE, 'I4', LAST, JDEBG )
2368:      call KEEP( 'LTEVE', LLTEVE, NTEVE, 'I4', LAST, JDEBG )
2369:      call REMAIN( 'KTEVE', NLTEMP, 'I4D', LAST )
2370:      call KEEP( 'KTEVE', LKTEVE, NLTEMP, 'I4', LAST, JDEBG )
2371: C
2372: C -----
2373:      call TALIN3( A(LJTEVE), A(LLTEVE), NTEVE, A(LKTEVE), NLTEMP,
2374: &      NKTEVE, A(LIDTAL), NIDTAL, NDTALY, A(LSWRK), A(LSWRK),
2375: &      A(LSWRK), NWORK )
2376: C -----
2377: C
2378:      call RESIZE( 'KTEVE', LKTEVE, NKTEVE, 'I4D', LAST )
2379: C
2380:      return
2381: C
2382: C ===== error messages ===
2383: C
2384:      933 format(/' XXX(talinp) "',A,'" data has too many items or lacks',
2385: &      ' closing ").')
2386:      934 format(/' XXX(talinp) "',A,'" tally is not supported in GMV2.')
2387:      9901 format(/' XXX(talinp) Duplicate ',A,' input for ',I3,'-th tally.')
2388:      9001 continue
2389:          write(IMG,7040) VNAME(:NLEN)
2390:      7040 format(/1X,'XXX(talinp) Input error on data <',A,'>')
2391:      call PRSTOP( 1, 'Data input error in $TALLY block.' )
2392:      stop 888
2393: C
2394:      9002 continue
2395:          write(IMG,7060) VNAME(:NLEN), NA, NWORK
2396:      7060 format(/' XXX(talinp) Size of data in item <',A,'> exceeds that',
2397: &      ' of a working area for input:')
2398:      &/13X,'(num. of data ',I6,' > limit =',I6,')'
2399:      &/13X,'Number of data is invalid or you need larger working',
2400: &      ' memory size ( "NWORK" after $TALLY ).')
2401:      call PRSTOP( 1, 'Data size overflow in $TALLY block.' )
2402:      stop 888
2403: C
2404:      9003 continue
2405:          write(IMG,7080)

2406:      7080 format(/' XXX(talinp) Unexpected end of input data during',
2407: &      ' reading "$TALLY" block.')
2408:      call PRSTOP( 1, 'End of input data in "$TALLY" block.' )
2409:      stop 888
2410: C
2411: C
2412:      9004 write(IMG,7100)
2413:      7100 format(/' XXX(talinp) Unexpected end of data or data read error',
2414: &      ' during skipping unnecessary or invalid data ($TALLY)')
2415:      call PRSTOP( 1, 'ERROR DURING SKIPPING TALLY INPUT DATA.' )
2416:      stop 888
2417: C
2418:      end
```


src/shared/talsrf.f

```

1:      subroutine TALSRF( IUB1, JDEBG, NTSRF, IDSRF, ITSRF, MTSRF, IA,
2:      &                  LKTSRF,KTSRF, NLTEMP,LKTSFJ,NBODY, NSURF,
3:      &                  IBSDA, IPSDA, MXBDY, ISFWK )
4: C=====
5: C Purpose: compose tally-surface data from input data saved on I/O unit
6: C         IUB1
7: C=====
8:      integer JDEBG(*)
9:      integer IDSRF(NTSRF), ITSRF(NTSRF+1,2)
10:     integer KTSRF(2,NLTEMP)
11: C
12:     integer IBSDA(3,NBODY+1), IBSDA(3,NSURF+1)
13: C
14:     integer IA(*)
15: C
16:     include 'INC/_IOUNIT'
17: C
18: C ... working array
19: C
20:     integer ISFWK(MXBDY,2)
21: C
22:     character*4 BTYP
23: C
24: C-----
25: C
26:     call RWIND( IUB1 )
27: C
28:     MTSRF = 0
29:     ITSRF(1,1) = 1
30: C
31:     call LABEL( IPR, 'TALLY-SURFACE' )
32: C
33:     do 150 IS = 1, NTSRF
34: C
35:         read(IUB1) NB, IDSRF(IS)
36:         read(IUB1) (ISFWK(I,1),I=1,NB)
37:         read(IUB1) (ISFWK(I,2),I=1,NB)
38: C
39:         write(IPR,7000) IS, IDSRF(IS)
40:         if ( IDSRF(IS).lt.0 ) then
41:             write(IMG,*)
42:             &          'XXX(TALSRF) ID of TALLY-SURFACE is negative :',
43:             &          IDSRF(IS)
44:             call CNTERR( 'FATAL' )
45:             end if
46:         do 100 ISS = 1, IS - 1
47:             if ( IDSRF(ISS).eq.IDSRF(IS) ) then
48:                 write(IMG,*) 'XXX(TALSRF) Duplicate TALLY-SURFACE ID :',
49:                 &          IDSRF(IS)
50:                 call CNTERR( 'FATAL' )
51:             end if
52:         100 continue
53:         write(IPR,7040) (ISFWK(I,1),I=1,NB)
54:         write(IPR,7020)
55:         write(IPR,7040) (ISFWK(I,2),I=1,NB)
56: C
57: 7000 format(1x,' Tally-surface ',I3,' : ID = ',I7//1x,' Body: ')
58: 7020 format(1x,' Flag(1=actual tally-surface/0=for clipping): ')
59: 7040 format(5x,10i6)
60: C
61:         do 140 I = 1, NB
62:             IBD = ISFWK(I,1)
63:             KSIGN = SIGN(1,IBD)
64: C
65:             do 110 IB = 1, NBODY

```

```

66:             if ( ABS(ISFWK(I,1)).eq.IBSDA(2,IB) ) then
67:                 if ( I.eq.1 ) then
68:                     call TYPBOD( BTYP, '<', IBSDA(3,IB) )
69:                     if ( BTYP.eq.'ELT' ) then
70:                         write(IMG,*) 'XXX(TALSRF) surface ',IDSRF(IS),
71:                         &          ': Currently, the first body of tally-surface ',
72:                         &          '(actual tally surface) must not be "ELT".'
73:                     end if
74:                 end if
75:                 go to 120
76:             end if
77: 110 continue
78:             write(IMG,*) 'XXX(TALSRF) TALLY-SURFACE ', IDSRF(IS),
79:             &          ' includes non-existent body ', ABS(ISFWK(I,1)), '.'
80:             call CNTERR( 'FATAL' )
81: C
82:             IB = 1
83: C
84: 120 continue
85: C
86:             NNN = IBSDA(1,IB+1) - IBSDA(1,IB)
87:             M0 = MTSRF + 1
88:             MTSRF = MTSRF + NNN
89: C
90:             if ( MTSRF.gt.NLTEMP ) then
91:                 write(IMG,*) 'XXX(TALSRF) no-more memory to restore ',
92:                 &          ' TALLY-SURFACE data !!!'
93:                 call PRSTOP( 1, 'MEMORY OVER IN "$TALLY" BLOCK.' )
94:                 stop 999
95:             end if
96: C
97: C ... KTSRF(1:2,*) is similar to KZDA(1:2,*)
98: C
99:             do 130 M = 0, NNN - 1
100:                 KTSRF(1,M0+M) = KSIGN*IPSDA(1,IBSDA(1,IB)+M)
101:                 if ( NNN.gt.1.and.KSIGN.lt.0 ) then
102:                     KTSRF(2,M0+M) = 1
103:                 else
104:                     KTSRF(2,M0+M) = 0
105:                 end if
106: 130 continue
107: C
108:             if ( NNN.gt.1.and.KSIGN.lt.0 ) KTSRF(2,MTSRF) = 2
109: C
110: C ... currently (or forever?) the first body is actual
111: C tally surface and other bodies are for clipping.
112: C
113: C KTSRF(1:2,ITSRF(IS,1):KTSRF(IS,2)) is surface
114: C composition of actual tally-surface.
115: C
116: CCCCCCCCC if ( I.eq.1 ) ITSRF(IS,2) = MTSRF
117:             if ( I.eq.1 ) ITSRF(IS,2) = MTSRF*KSIGN
118: C
119: 140 continue
120:             ITSRF(IS+1,1) = MTSRF + 1
121: C
122: 150 continue
123: C
124: C ... adjust memory size for KTSRF
125: C
126:             call RESIZE( 'KTSRF', LKTSRF, 2*MTSRF, 'I4', LAST )
127: C
128: C
129: C ... read file again to store KTSFJ
130: C

```

src/shared/talsrf.f

```
131: C      call KEEP( 'KTSFJ', LKTSFJ, MTSRF, 'I4', LAST, JDEBG )
132: C
133: C      call RWIND( IUB1 )
134: C
135: C      do 160 IS = 1, NTSRF
136: C          M          = ITSRF(IS,1) - 1
137: C          MM         = ITSRF(IS+1,1) - ITSRF(IS,1)
138: C          read(IUB1) NB, IDUM
139: C          read(IUB1) (ISFWK(I,1),I=1,NB)
140: C          read(IUB1) (ISFWK(I,2),I=1,NB)
141: C 160 continue
142: C
143: C      return
144: C      end
```

src/shared/tblxy.f

```

1:      subroutine TBLXY( SRCSP, NSOUR, IERR, NERR, IWRK, SWRK, DWRK,
2:      &                NWORK )
3: C==<MVP/GMVP>=====
4: C purpose: input source sampling information of 'TABLEXY' type.
5: C called from: smpinf
6: C calls :      PUTV INTPLC CHREAD WALIAS R4READ PACKCS PACKND CKVAL4
7: C          CKODR4
8: C=====
9: C arguments (i=input, o=output, w=work)
10: C
11: C io srcsp : data packet container for source information
12: C i nsour : current source set #
13: C
14: C io nerr : total number of input data error
15: C
16: C w iwrk(nwork),swrk(nwork),dwork(*) :
17: C     working area for input & preprocessing (start on the
18: C     same address)
19: C w dwrk(nwork) : double word working area.
20: C-----
21:      integer SRCSP(*)
22: C
23: C ... iwrk & swrk has same starting address !!
24: C     (dwrk is not)
25: C
26:      integer IWRK(NWORK)
27:      real SWRK(NWORK)
28:      real*8 DWRK(NWORK)
29: C
30: C ==== local data ====
31: C
32:      character*256 CWRK
33:      character*16 INTSYM
34:      character*26 ALP
35: C
36:      include 'INC/_IOUNIT'
37:      data ALP /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
38: C
39: C-----
40: C execution starts after here -----
41: C-----
42: C
43: 7000 format(1X,'      >>  <',A,'>'/(15X,1P,10(E11.4)))
44: 7020 format(1X,'      >>  <',A,'>  ',A)
45: C
46: 7040 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',A,' <',A,'>'/)
47: 7060 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',A/(20X,A))
48: C
49: 7080 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ',
50: &          ' WORKING AREA FOR INPUT IS INSUFFICIENT.'/5X,
51: &          ' INCREASE WORKING AREA SIZE BY SPECIFYING OR MODIFYING',
52: &          ' "NWORK" AFTER "$SOURCE". (CURRENT NWORK =',I6,')'/)
53: 7100 format(/1X,'XXX <SAMPLING INFORMATION ERROR> ', 'NUNER OF DATA ',A,
54: &          ' IS NOT VALID'/)
55: C
56: C
57: C >>>> #TABLEXY      X( point ) PX(probability) INT(interpolation)
58: C          [INTY(...)] Y(points) PY(prob) ... ;
59: C
60: C
61:      INTX      = 0
62:      INTY      = 0
63:      NX        = 0
64:      NPX       = 0
65:      LX        = 0

```

```

66:      LPX      = 0
67: C
68:      IPDX     = -1
69: C
70:      IY       = 0
71:      KY       = 0
72:      KPY      = 0
73:      MODEY    = 0
74: C
75:      NWK      = NWORK
76:      LP       = 1
77: C
78:      LYI      = 0
79:      LNYI     = 0
80:      LPYI     = 0
81:      LPYPI    = 0
82:      LNPYI    = 0
83:      LINTYI   = 0
84: C
85:      LIPDYI   = 0
86: C
87: C
88: 100 call CHREAD( 'TABLEXY', CWRK, '(', NLEN, IT, JENDS )
89: C
90:      if ( JENDS.eq.0.and.IT.ne.1.and.NLEN.gt.0 ) then
91:          write(MSG,7040) 'Invalid data', CWRK(:NLEN)
92:          NERR = NERR + 1
93:          go to 100
94:      else if ( JENDS.ne.0 ) then
95:          write(MSG,7060) 'Unexpected end of data.'
96:          NERR = NERR + 1
97:          go to 110
98:      else if ( IT.eq.2 ) then
99:          go to 110
100:      end if
101: C
102: C
103: C ... X , PX & INTX ...
104: C
105: C
106:      if ( CWRK(:NLEN).eq.'X' .or. CWRK(:NLEN).eq.'PX'
107: & .or. CWRK(:NLEN).eq.'PDX' .or. CWRK(:NLEN).eq.'INTX' ) then
108: C
109:          if ( MODEY.gt.0 ) then
110:              write(MSG,7060) 'For "TABLEXY" sampling ',
111: & 'any specification for "X" must precede any "Y" specifications.'
112:              NERR = NERR + 1
113:          end if
114: C
115:      NA      = 0
116:      LX      = 0
117:      LPX     = 0
118: C
119:      if ( CWRK(:NLEN).eq.'X' ) then
120:          LX = LP
121:          call R4READ( 'X', SWRK(LP), NA, -NWK, JRET )
122:          NX = NA
123:      else if ( CWRK(:NLEN).eq.'PX' ) then
124:          LPX = LP
125:          call R4READ( 'PX', SWRK(LP), NA, -NWK, JRET )
126:          NPX = NA
127:          IPDX = 0
128:      else if ( CWRK(:NLEN).eq.'PDX' ) then
129:          LPX = LP
130:          call R4READ( 'PX', SWRK(LP), NA, -NWK, JRET )

```

src/shared/tblxy.f

```

131:      NPX      = NA
132:      IPDX     = 1
133:      else if ( CWRK(:NLEN).eq.'INTX' ) then
134:        call CHREAD( 'INT', INTSYM, ' ', NL, IT, IEND )
135:        if ( IEND.ne.0 ) then
136:          write(IMG,*) 'XXX No data for "INTX"'
137:          go to 160
138:        end if
139:        call INTPLC( INTSYM(:NL), INTX, IDIFF )
140:        if ( INTX.lt.0 ) then
141:          write(IMG,7040) 'Invalid interpolation', INTSYM(:NL)
142:          NERR      = NERR + 1
143:        end if
144:      end if
145:      NWK      = MAX(0,NWK-NA)
146:      LP       = LP + NA
147: C
148: C
149: C ... Y , PY & INTY ...
150: C
151: C
152:      else if ( CWRK(:NLEN).eq.'Y' .or. CWRK(:NLEN).eq.'PY'
153:      & .or. CWRK(:NLEN).eq.'PDY'.or. CWRK(:NLEN).eq.'INTY' ) then
154:        if ( MODEY.eq.0 ) then
155:          MODEY      = 1
156: C
157: C ... indices to y-data in working area ...
158: C
159:          LYI        = LP
160:          LNYI        = LYI + NPX
161:          LPYI        = LNYI + NPX
162:          LPYPI       = LPYI + NPX
163:          LNPYI       = LPYPI + NPX
164:          LNPYPI      = LNPYI + NPX
165:          LPAAYI      = LNPYPI + NPX
166:          LINTYI      = LPAAYI + NPX
167: C
168:          LIPDYI      = LINTYI + NPX
169: C
170:          LP          = LIPDYI + NPX
171: C
172: C ... lp0 is starting address of y-data ...
173: C
174:          LP0         = LP
175: C
176:          NWK         = MAX(0,NWORK-LP)
177:          if ( NWK.le.0 ) then
178:            write(IMG,7080) NWORK
179:            NERR      = NERR + 1
180:            return
181:          end if
182: C
183:          call PUTV( IWRK(LYI), NPX, 0 )
184:          call PUTV( IWRK(LNYI), NPX, 0 )
185:          call PUTV( IWRK(LPYI), NPX, 0 )
186:          call PUTV( IWRK(LNPYI), NPX, 0 )
187:          call PUTV( IWRK(LPYPI), NPX, 0 )
188:          call PUTV( IWRK(LNPYPI), NPX, 0 )
189:          call PUTV( IWRK(LPAAYI), NPX, 0 )
190:          call PUTV( IWRK(LINTYI), NPX, 0 )
191:          call PUTV( IWRK(LIPDYI), NPX, -1 )
192:
193:        end if
194: C
195:      NA          = 0

```

```

196: C
197:
198:
199:      IWRK(LYI+IY)      = LP
200:      call R4READ( 'Y', SWRK(LP), NA, -NWK, JRET )
201:      IWRK(LNYI+IY)      = NA
202:      KY                = 1
203:
204:
205:      else if ( CWRK(:NLEN).eq.'PY' ) then
206:
207:        IWRK(LPYI+IY)      = LP
208:        call R4READ( 'PY', SWRK(LP), NA, -NWK, JRET )
209:        IWRK(LNPYI+IY)      = NA
210:        if ( IWRK(LIPDYI+IY).eq.-1 ) then
211:          IWRK(LIPDYI+IY)      = 0
212:        else
213:          NERR      = NERR + 1
214:          write(IMG,*) ' '
215:          write(IMG,*) ' XXX *****'
216:          write(IMG,*) ' * PY or PDY is doubly defined **'
217:          write(IMG,*) ' ***** not allowed *****'
218:          write(IMG,*) ' '
219:        end if
220:        KPY          = 1
221:
222:      else if ( CWRK(:NLEN).eq.'PDY' ) then
223:
224:        IWRK(LPYI+IY)      = LP
225:        call R4READ( 'PY', SWRK(LP), NA, -NWK, JRET )
226:        IWRK(LNPYI+IY)      = NA
227:        if ( IWRK(LIPDYI+IY).eq.-1 ) then
228:          IWRK(LIPDYI+IY)      = 1
229:        else
230:          NERR      = NERR + 1
231:          write(IMG,*) ' '
232:          write(IMG,*) ' XXX *****'
233:          write(IMG,*) ' * PY or PDY is doubly defined **'
234:          write(IMG,*) ' ***** not allowed *****'
235:          write(IMG,*) ' '
236:        end if
237:        KPY          = 1
238:
239:      else if ( CWRK(:NLEN).eq.'INTY' ) then
240:
241:        call CHREAD( 'INT', INTSYM, ' ', NL, IT, IEND )
242:        if ( IEND.ne.0 ) then
243:          write(IMG,*) 'XXX No data for "INTY"'
244:          go to 160
245:        end if
246:        call INTPLC( INTSYM(:NL), INTY, IDIFF )
247:        if ( INTY.lt.0 ) then
248:          write(IMG,7040) 'Invalid interpolation', INTSYM(:NL)
249:          NERR      = NERR + 1
250:        end if
251:
252:      end if
253:      NWK      = MAX(0,NWK-NA)
254:      LP       = LP + NA
255: C
256:      if ( KY.eq.1.and.KPY.eq.1 ) then
257:        IWRK(LINTYI+IY) = INTY
258:        IY              = IY + 1
259:        KY              = 0
260:        KPY             = 0
261:      end if

```

src/shared/tblxy.f

```

261: C
262: C
263: C ... error : unrecognizable input ....
264: C
265: C
266: C     end if
267: C
268: C     go to 100
269: C
270: C
271: C ==== post processing of input data =====
272: C
273: C
274: C 110 continue
275: C
276: C     if ( LP.gt.NWORK ) then
277: C       write(IMG,7080) NWORK
278: C       NERR = NERR + 1
279: C       return
280: C     end if
281: C
282: C     LP1 = LP
283: C
284: C     if ( NX.eq.0 .or. NPX.eq.0 ) then
285: C       write(IMG,7060) 'No input for "X" or "PX"'
286: C       NERR = NERR + 1
287: C     else
288: C
289: C       write(IPR,7000) 'X', (SWRK(I),I=LX,LX+NX-1)
290: C       if ( IPDX.eq.0 ) then
291: C         write(IPR,7000) 'PX', (SWRK(I),I=LPX,LPX+NPX-1)
292: C       else if ( IPDX.eq.1 ) then
293: C         write(IPR,7000) 'PDX', (SWRK(I),I=LPX,LPX+NPX-1)
294: C       end if
295: C
296: C       .... check order of X ...
297: C
298: C       ICMOD = -1
299: C       call CKODR4( SWRK(LX), NX, ICMOD, ITEST )
300: C       if ( ITEST.eq.1 ) then
301: C         ICMOD = -2
302: C         call CKODR4( SWRK(LX), NX, ICMOD, ITEST )
303: C         if ( ITEST.eq.1 ) then
304: C           NERR = NERR + 1
305: C           write(IMG,*) ' '
306: C           write(IMG,*) ' XXX *****'
307: C           write(IMG,*) ' * X is out of order not allowed *'
308: C           write(IMG,*) ' *****'
309: C           write(IMG,*) ' '
310: C         end if
311: C       end if
312: C
313: C       .... check negative probability ...
314: C
315: C       call CKVAL4( 'PX', SWRK(LPX), NPX, ITEST, 0 )
316: C       if ( ITEST.eq.1 ) NERR = NERR + 1
317: C
318: C       if ( INTX.eq.0 ) then
319: C         if ( NPX.eq.NX ) then
320: C           INTSYM = 'DISCRETE'
321: C         else if ( NPX.eq.NX-1 ) then
322: C           INTSYM = 'STEP'
323: C         else
324: C           INTSYM = 'UNKNOWN'
325: C         end if

```

```

326: C       NL = ICLEN(INTSYM)
327: C       call INTPLC( INTSYM(:NL), INTX, IDIFF )
328: C     else if ( INTX.gt.0 ) then
329: C       INTSYM = ' '
330: C       call INTPLC( INTSYM, INTX, IDIFF )
331: C       NL = ICLEN(INTSYM)
332: C     end if
333: C
334: C     write(IPR,7020) 'INTERPOLATION', INTSYM(:NL)
335: C
336: C     if ( INTX.gt.0 ) then
337: C       if ( NX-NPX.ne.IDIFF ) then
338: C         write(IMG,7100) 'X' or "PX"
339: C         NERR = NERR + 1
340: C       else
341: C
342: C         ... probability data --> sampling form ...
343: C
344: C         ... sample intervals by PX(i)+PX(i+1) for linear interpolation
345: C
346: C         if ( INTSYM(:NL).eq.'LINEAR' .or. INTSYM.eq.'LOG-LINEAR'
347: C           &
348: C           ) then
349: C           if ( LP+NPX-1.gt.NWORK ) then
350: C             write(IMG,7080) NWORK
351: C             NERR = NERR + 1
352: C             return
353: C           end if
354: C           NPXP = NPX - 1
355: C           LPXP = LP
356: C           LP = LPXP + NPXP
357: C
358: C           if ( IPDX.eq.0 ) then
359: C             do 120 I = 0, NPXP - 1
360: C               SWRK(LPXP+I) = SWRK(LPX+I) + SWRK(LPX+I+1)
361: C             continue
362: C           else if ( IPDX.eq.1 ) then
363: C             do I = 0, NPXP - 1
364: C               if ( INTSYM(:NL).eq.'LINEAR' ) then
365: C                 DEL = SWRK(LX+I) - SWRK(LX+I+1)
366: C               else
367: C                 DEL = LOG( SWRK(LX+I) / SWRK(LX+I+1) )
368: C               end if
369: C               DEL = ABS( DEL )
370: C
371: C               SWRK(LPXP+I) = (SWRK(LPX+I)+SWRK(LPX+I+1))
372: C               * DEL
373: C             end do
374: C           end if
375: C
376: C           ... interval/point sampling probability is as given for
377: C           STEP & LOG-STEP interpolation
378: C
379: C           else
380: C             NPXP = NPX
381: C             LPXP = LPX
382: C
383: C             if ( IPDX.eq.1 .and.
384: C               &
385: C               INTSYM(:NL).ne.'DISCRETE' ) then
386: C               do I = 0, NPXP-1
387: C                 if ( INTSYM(:NL).eq.'STEP' ) then
388: C                   DEL = SWRK(LX+I) - SWRK(LX+I+1)
389: C                 else
390: C                   DEL = LOG( SWRK(LX+I) / SWRK(LX+I+1) )
391: C                 end if
392: C                 DEL = ABS( DEL )

```

src/shared/tblxy.f

```

391: C
392:         SWRK(LPXP+I) = SWRK(LPXP+I) * DEL
393:     end do
394: end if
395: end if
396:
397: if ( LP+2*NPXP.gt.NWORK ) then
398:     write(IMG,7080) NWORK
399:     NERR = NERR + 1
400:     return
401: else
402:     LPAA = LP + NPXP
403:     call WALIAS( SWRK(LPXP), SWRK(LP), SWRK(LPAA), NPXP )
404: C
405: C-----packing-----
406: call PACKCS( SRCSP, 'INTERPOLATION', INTSYM, IRET1 )
407: call PACKND( SRCSP, 'X-points', 'R4', SWRK(LX), NX,
408: &           IRET2 )
409: &           IRET3 = 0
410: if ( INTSYM(:NL).eq.'LINEAR'
411: &   .or. INTSYM(:NL).eq.'LOG-LINEAR' )
412: &   call PACKND( SRCSP, 'PROBABILITY as given', 'R4',
413: &               SWRK(LPX), NPX, IRET3 )
414:
415: call PACKND( SRCSP, 'PROBABILITY', 'R4', SWRK(LP),
416: &           NPXP, IRET4 )
417: call PACKND( SRCSP, 'ALIAS', 'I4', SWRK(LPAA), NPXP,
418: &           IRET5 )
419:
420:
421: if ( ABS(IRET1)+ABS(IRET2)+ABS(IRET3)+ABS(IRET4)
422: &   +ABS(IRET5).gt.0 ) NERR = NERR + 1
423: C-----
424: C
425:     end if
426: end if
427: end if
428: end if
429: C
430: C
431: C === post processing of y-data ===
432: C
433: C
434: C     LP = LP1
435: C
436: if ( IY.ne.NPXP ) then
437:     write(IMG,7060)
438: &     'For "TABLEXY" sampling, number of "Y" specification sets',
439: &     'does not match the number of "PX" data.'
440:     NERR = NERR + 1
441: end if
442: C
443: do 150 IY = 0, NPXP - 1
444: C
445:     INTY = IWRK(LINTYI+IY)
446:     IPDY = IWRK(LIPDYI+IY)
447: C
448:     NY = IWRK(LNYI+IY)
449:     NPY = IWRK(LNPYI+IY)
450: C
451:     LY = IWRK(LYI+IY)
452:     LPY = IWRK(LPYI+IY)
453: C
454: if ( NY.eq.0 .or. NPY.eq.0 ) then
455:     write(IMG,7060) 'No input for "Y" or "PY"'

```

```

456:         NERR = NERR + 1
457: else
458: C
459: write(IPR,7000) 'Y', (SWRK(I),I=LY,LY+NY-1)
460: if ( IPDY.eq.0 ) then
461:     write(IPR,7000) 'PY', (SWRK(I),I=LPY,LPY+NPY-1)
462: else if ( IPDY.eq.1 ) then
463:     write(IPR,7000) 'PDY', (SWRK(I),I=LPY,LPY+NPY-1)
464: end if
465: C
466: C     .... check order of Y ...
467: C
468: ICMOD = -1
469: call CKODR4( SWRK(LY), NY, ICMOD, ITEST )
470: if ( ITEST.eq.1 ) then
471:     ICMOD = -2
472:     call CKODR4( SWRK(LY), NY, ICMOD, ITEST )
473:     if ( ITEST.eq.1 ) then
474:         NERR = NERR + 1
475:         write(IMG,*) ' '
476:         write(IMG,*) ' XXX *****'
477:         write(IMG,*) ' * Y is out of order not allowed *'
478:         write(IMG,*) ' *****'
479:         write(IMG,*) ' '
480:     end if
481: end if
482: C     .... check negative probability ...
483: call CKVAL4( 'PY', SWRK(LPY), NPY, ITEST, 0 )
484: if ( ITEST.eq.1 ) NERR = NERR + 1
485: C
486: if ( INTY.eq.0 ) then
487:     if ( NPY.eq.NY ) then
488:         INTSYM = 'DISCRETE'
489:     else if ( NPY.eq.NY-1 ) then
490:         INTSYM = 'STEP'
491:     else
492:         INTSYM = 'UNKNOWN'
493:     end if
494:     NL = ICLEN(INTSYM)
495:     call INTPLC( INTSYM(:NL), INTY, IDIFF )
496: else if ( INTY.gt.0 ) then
497:     INTSYM = ' '
498:     call INTPLC( INTSYM, INTY, IDIFF )
499:     NL = ICLEN(INTSYM)
500: end if
501: IWRK(LINTYI+IY) = INTY
502: CM2015 NL = ICLEN(INTSYM)
503: C
504: write(IPR,7020) 'INTERPOLATION', INTSYM(:NL)
505: C
506: if ( INTY.gt.0 ) then
507:     if ( NY-NPY.ne.IDIFF ) then
508:         write(IMG,7100) '"Y" or "PY"'
509:         NERR = NERR + 1
510:     else
511: C
512: C     ... probability data --> sampling form ...
513: C
514: C     ... sample intervals by Py(i)+Py(i+1) for linear interpolation
515: C
516: if ( INTSYM(:NL).eq.'LINEAR'
517: &   .or. INTSYM(:NL).eq.'LOG-LINEAR' ) then
518: C
519: if ( LP+NPY-1.gt.NWORK ) then
520:     write(IMG,7080) NWORK

```

src/shared/tblxy.f

```

521:          NERR      = NERR + 1
522:          return
523:        end if
524:
525:        NPYP      = NPY - 1
526:        LPYP      = LP
527:        LP        = LPYP + NPYP
528:
529:        if ( IPDY.eq.0 ) then
530:          do 130 I = 0, NPYP - 1
531:            SWRK(LPYP+I) = SWRK(LPY+I) + SWRK(LPY+I+1)
532:          continue
533:        else if ( IPDY.eq.1 ) then
534:          do I = 0, NPYP - 1
535:            if ( INTSYM(:NL).eq.'LINEAR' ) then
536:              DEL = SWRK(LY+I) - SWRK(LY+I+1)
537:            else
538:              DEL = LOG( SWRK(LY+I) / SWRK(LY+I+1) )
539:            end if
540:            DEL = ABS( DEL )
541:          C
542:            SWRK(LPYP+I) = (SWRK(LPY+I)+SWRK(LPY+I+1))
543:            &              * DEL
544:          end do
545:        end if
546:
547:        ... interval/point sampling probability is as given for
548:        STEP & LOG-STEP interpolation
549:
550:        else
551:          NPYP      = NPY
552:          LPYP      = LPY
553:
554:          if ( IPDY.eq.1 .and.
555:            &      INTSYM(:NL).ne.'DISCRETE' ) then
556:            do I = 0, NPYP-1
557:              if ( INTSYM(:NL).eq.'STEP' ) then
558:                DEL = SWRK(LY+I) - SWRK(LY+I+1)
559:              else
560:                DEL = LOG( SWRK(LY+I) / SWRK(LY+I+1) )
561:              end if
562:              DEL = ABS( DEL )
563:
564:              SWRK(LPYP+I) = SWRK(LPYP+I) * DEL
565:            end do
566:          end if
567:        end if
568:
569:        IWRK(LPYPI+IY) = LPYP
570:        IWRK(LNPYPI+IY) = NPYP
571:
572:        if ( LP+2*NPYP.gt.NWORK ) then
573:          write(IMSG,7080) NWORK
574:          NERR      = NERR + 1
575:          return
576:        else
577:          LPAA      = LP
578:          LPWK      = LPAA + NPYP
579:
580:          call WALIAS( SWRK(LPYP), SWRK(LPWK), IWRK(LPAA),
581:            &          NPYP )
582:
583:          do 140 I = 0, NPYP - 1
584:            SWRK(LPYP+I) = SWRK(LPWK+I)
585:          continue

```

```

586:          IWRK(LPAAYI+IY) = LPAA
587:          LP      = LPWK
588:        end if
589:      end if
590:    end if
591:  end if
592:  IWRK(LYI+IY) = IWRK(LYI+IY) - LP0 + 1
593:  IWRK(LPYI+IY) = IWRK(LPYI+IY) - LP0 + 1
594:  IWRK(LPYPI+IY) = IWRK(LPYPI+IY) - LP0 + 1
595:  IWRK(LPAAYI+IY) = IWRK(LPAAYI+IY) - LP0 + 1
596: 150 continue
597: C
598:   NDY      = LP - LP0
599: C
600: C---<packing>-----
601: call PACKND( SRCSP, 'NUMBER OF Y', 'I4', IWRK(LNYI), NPX, IRET1 )
602: call PACKND( SRCSP, 'NUMBER OF PY', 'I4', IWRK(LNPYI), NPX, IRET2
603: &          )
604: call PACKND( SRCSP, 'NUMBER OF PYP', 'I4', IWRK(LNPYPI), NPX,
605: &          IRET3 )
606: call PACKND( SRCSP, 'INDEX TO YI', 'I4', IWRK(LYI), NPX, IRET4 )
607: call PACKND( SRCSP, 'INDEX TO PYI', 'I4', IWRK(LPYI), NPX, IRET5 )
608: call PACKND( SRCSP, 'INDEX TO PYPI', 'I4', IWRK(LPYPI), NPX, IRET6
609: &          )
610: call PACKND( SRCSP, 'INDEX TO ALIAS', 'I4', IWRK(LPAAYI), NPX,
611: &          IRET7 )
612: call PACKND( SRCSP, 'INTERPOLATION #', 'I4', IWRK(LINTYI), NPX,
613: &          IRET8 )
614:
615: call PACKND( SRCSP, 'Y SAMPLING', 'R4', SWRK(LP0), NDY, IRET9 )
616:
617: if ( ABS(IRET1)+ABS(IRET2)+ABS(IRET3)+ABS(IRET4)+ABS(IRET5)
618: &    +ABS(IRET6)+ABS(IRET7)+ABS(IRET8)+ABS(IRET9).gt.0 ) NERR = NERR
619: &    + 1
620: C-----
621: C
622:   return
623: C
624: C .... generic handler for end of data ...
625: C
626: 160 call PRSTOP( 1, 'UNEXPECTED END OF DATA' )
627: stop 888
628: end

```

src/shared/tlore.f

```

1:      subroutine TLOREG( JTLLT, NTREG, IPTRG, LPTRG, NLTEMP,
2:      &                  NLPTRG, ITRNM, LAST, IFILE, LINE, NCREG, NREG,
3:      &                  NINPZ, NSPACE, NSUZON, NZONE, NEST, ISPNM,
4:      &                  ISUSP, KMAT, KREG, KREGI, KCELI, KSUZN,
5:      &                  IRGSP, TNAMS, NNAMES, LIMIT )
6: C=====
7: C PURPOSE : CHECK RE-DEFINED REGION & MAKE REGION <-> TALLY-REGION
8: C          TABLE.
9: C CALLED IN : GEOMIN
10: C CALLS : CBSINC, CHREAD, LABEL, MEMERR, REGNAM, RGFIND, RIUNIT
11: C-----
12: C ARGUMENTS :
13: C
14: C === OUTPUT OR INPUT/OUTPUT VARIABLES ===
15: C
16: C LAST : STARTING POSITION OF AVAILABLE ARRAY-DATA STORAGE.
17: C
18: C IT MUST BE THE STARTING POSITION OF LPTRG AT THE BEGINNING
19: C OF THIS ROUTINE AND SET TO THE NEXT POSITION OF LPTRG.
20: C
21: C
22: C
23: C=====
24:      integer IPTRG(*), LPTRG(*), ITRNM(*)
25: C
26:      integer ISUSP(NSUZON, NSPACE), KMAT(NINPZ), KREG(NINPZ),
27:      &        KREGI(NINPZ), KCELI(NINPZ), KSUZN(NINPZ, 2), IRGSP(2, NREG)
28: C
29:      include 'INC/_LNAM'
30: C
31:      integer ISPNM(2, 0:NEST, NSPACE)
32:      integer NNAMES
33:      character*(LNAM) TNAMS(NNAMES)
34:      character*(*) LINE
35: C
36:      include 'INC/_IOUNIT'
37: C
38: C
39: C-----
40: C
41:      character*128 NAME
42: C
43: C=====
44: C
45: C
46:      NTREG = 0
47:      IPTRG(1) = 1
48:      NLPTRG = 0
49: C
50:      NERR = 0
51: C
52:      if ( NCREG.eq.0 ) go to 170
53: C
54: C
55: C *****
56: CC CALL LABEL( IOG, 'CHECK FOR TALLY-REGIONS' )
57: C *****
58: C
59: C .... FIND #REGION
60: C
61:      call RWIND( IFILE )
62: 100 read( IFILE, '(A)' ) LINE
63:      K = INDEX( LINE, '#REGION' )
64:      if ( K.eq.0 .or. ( K.gt.1.and.LINE(:K-1).ne.' ' ) ) go to 100
65: C

```

```

66: C
67:      call RIUNIT( IFILE )
68: C
69: C
70: C -----
71: C          IMODE = 1 / -1 : ADD / DEFINE
72: C -----
73: C
74:      IMODE = 1
75: C
76: 110 LINE = ' '
77:      call CHREAD( ' ', LINE, '( ', LTK, ITERM, IEND )
78: C
79: CCCC if ( IEND.eq.0 ) then
80:      if ( IEND.eq.0.and.LINE(:LTK).ne.'$END' ) then
81: C
82: C          .... MODE STRING ....
83: C
84:      if ( ITERM.eq.0 ) then
85:          if ( LINE(1:3).eq.'ADD' .or. LINE(1:3).eq.'DEF' ) then
86:              if ( LINE(1:3).eq.'DEF' ) then
87:                  IMODE = -1
88:              else
89:                  IMODE = 1
90:              end if
91:          else
92:              write( MSG, *) 'XXX(TLOREG) Invalid TALLY-REGION mode <',
93:              &              LINE(:LTK), '>'. "ADD" assumed temporary.'
94:              call CNTERR( 'FATAL' )
95:              LINE(1:3) = 'ADD'
96:              LTK = 3
97:              IMODE = 1
98:          end if
99:          go to 110
100:      end if
101: C
102: C .... TALLY REGION NAME .....
103: C
104:      NTREG = NTREG + 1
105:      NN = 0
106: C
107:      if ( LINE(1:1).ne.'@' ) then
108:          do 120 I = MIN(LTK, LNAM-1), 1, -1
109:              LINE(I+1:I+1) = LINE(I:I)
110:          continue
111:          LTK = MIN(LNAM, LTK+1)
112:          LINE(1:1) = '@'
113:      end if
114: C
115:      call CBSINC( TNAMS, NNAMES, LINE(:LTK), IPOS )
116: C
117:      if ( IPOS.eq.0 ) then
118:          write( MSG, *) 'XXX TALLY-REGION name <', LINE(:LTK),
119:          &              '> was not found in registered name list.'
120:          call PRSTOP( 1, 'INVALID TALLY REGION NAME.' )
121:          stop 666
122:      end if
123: C
124:      ITRNM(NTREG) = IMODE*IPOS
125: C
126:      ... GET REGION NAME COMPOSING TALLY-REGION ...
127: C
128: 130 call CHREAD( ' ', LINE, ' )', NLEN, ITERM, IEND )
129: C
130: C -----

```


src/shared/tlreg.f

```

131: C      .... REGION-NAME --> REGION #
132: C -----
133: C
134: C
135: C ----- LOOP 1150 -----
136: C
137: C      IBEGIN = 0
138: C
139: C 140 call RGFIND( IOG, LINE(1:NLEN), IBEGIN, JEND, IREG, IERR,
140: &      JTLLT, KMAT, KREG, KREGI, KCELI, NINPZ, ISPNM, ISUSP,
141: &      KSUZN, IRGSP, NEST, NSPACE, NSUZON, NZONE, NREG, TNAMS,
142: &      NNAMES )
143: C
144: C      if ( IREG.ne.0 ) then
145: C          do 150 I = 1, NN
146: C              if ( LPTRG(IPTRG(NTREG))+I-1).eq.IREG ) go to 160
147: C          continue
148: C
149: C
150: C      .... REGISTER REGION TO TALLY-REGION ...
151: C
152: C          LPTRG(IPTRG(NTREG)+NN) = IREG
153: C          NN = NN + 1
154: C
155: C          NLPTRG = NLPTRG + 1
156: C          if ( NLPTRG.gt.NLTEMP ) then
157: C              call GTLAST( LAST )
158: C              call MEMERR( 'TLOREG',
159: C                  &      'MEMORY OVER IN TALLY-REGION PROCESSING.',
160: C                  &      LSIZ(LAST+NLPTRG), LIMIT )
161: C              stop 999
162: C          end if
163: C
164: C 160 continue
165: C      end if
166: C
167: C      if ( IERR.ne.0 ) NERR = NERR + 1
168: C      if ( JEND.eq.0 ) go to 140
169: C
170: C -----:----- END OF LOOP 1150 -----
171: C
172: C
173: C      IPTRG(NTREG+1) = IPTRG(NTREG) + NN
174: C      if ( ITERM.eq.0 ) go to 130
175: C
176: C -----:----- END OF LOOP 1100 -----
177: C
178: C
179: C ----- TAIL OF LOOP 1000 -----
180: C
181: C      go to 110
182: C      end if
183: C
184: C -- END OF LOOP 1000 ( END OF FILE )
185: C
186: C -----
187: C SEARCH REGIONS WHOSE TALLIES ARE OUTPUT INDEPENDENTLY OF RE-DEFINED
188: C REGIONS.
189: C * NOT INCLUDED IN RE-DEFINED REGIONS OF 'DEFINE' MODE
190: C -----
191: C
192: C 170 NPT0 = NTREG
193: C
194: C      do 200 N = 1, NREG
195: C          do 190 I = 1, NPT0

```

```

196: C      if ( ITRNM(I).lt.0 ) then
197: C          do 180 K = IPTRG(I), IPTRG(I+1) - 1
198: C              if ( LPTRG(K).eq.N ) go to 200
199: C          continue
200: C      end if
201: C      continue
202: C
203: C      NLPTRG = NLPTRG + 1
204: C      if ( NLPTRG.gt.NLTEMP ) then
205: C          call GTLAST( LAST )
206: C          call MEMERR( 'TLOREG',
207: C              &      'MEMORY OVER IN TALLY-OUTPUT-REGION PROCESSING.',
208: C              &      LSIZ(LAST+NLPTRG), LIMIT )
209: C          stop 999
210: C      end if
211: C
212: C      NTREG = NTREG + 1
213: C      LPTRG(IPTRG(NTREG)) = N
214: C      IPTRG(NTREG+1) = IPTRG(NTREG) + 1
215: C
216: C      .... ITRNM : REGION # FOR NON-RE-DEFINED TALLY-OUTPUT-REGION
217: C
218: C      ITRNM(NTREG) = N
219: C      200 continue
220: C -----
221: C MAKE ITRNM(I) NEGATIVE FOR RE-DEFINED REGIONS.
222: C -----
223: C      do 210 I = 1, NCREG
224: C          ITRNM(I) = -ABS(ITRNM(I))
225: C      210 continue
226: C
227: C *****
228: C call LABEL( IOG, 'TALLY REGIONS' )
229: C *****
230: C
231: C      write(IOG,('( ' TOTAL NUMBER OF TALLY-REGION: ' ',I8/))' ) NTREG
232: C
233: C      write(IOG,('( ' ** NAMES OF TALLY-REGIONS ** ' ')/))' )
234: C
235: C      do 230 I = 1, NTREG
236: C          if ( ITRNM(I).lt.0 ) then
237: C              NAME = TNAMS(-ITRNM(I))
238: C              LNM = INDEX(NAME, ' ') - 1
239: C              if ( LNM.lt.0 ) LNM = LEN(TNAMS(1))
240: C
241: C              write(IOG,7000) I, NAME(:LNM)
242: C
243: C          do 220 K = IPTRG(I), IPTRG(I+1) - 1
244: C              call REGNAM( LPTRG(K), '>', NAME, LNM, JTLLT, KMAT, KREG,
245: C                  &      KREGI, KCELI, NINPZ, ISUSP, ISPNM, KSUZN, IRGSP,
246: C                  &      NZONE, NSPACE, NSUZON, NEST, NREG, TNAMS, NNAMES )
247: C              &
248: C
249: C              write(IOG,7020) LPTRG(K), NAME(:LNM)
250: C          continue
251: C      else
252: C          call REGNAM( ITRNM(I), '>', NAME, LNM, JTLLT, KMAT, KREG,
253: C              &      KREGI, KCELI, NINPZ, ISUSP, ISPNM, KSUZN, IRGSP,
254: C              &      NZONE, NSPACE, NSUZON, NEST, NREG, TNAMS, NNAMES )
255: C
256: C          write(IOG,7040) I, NAME(:LNM), ITRNM(I)
257: C      end if
258: C
259: C 7000 format(1X,I8,'. <',A,'>')
260: C 7020 format(1X,8X,' CONTAINS REGION (NO. ',I8,') <',A,'>')

```

src/shared/tlore.f

```
261: 7040    format(1X,I8,'.  <',A,'>    ( REGION # ',I8,') ')
262: 230 continue
263:    return
264:    end
```

SAFE

src/shared/tlstio.f

```
1:      subroutine OPEN_TLF(IDTASK, FNAME)
2: C=====
3: C purpose: Open separate files from each PE in a parallel
4: C          calculation to output time list data.
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C i IDTASK : MVP task ID (from 1 to NTASK) of current process.
8: C i TLFN : (base) file name of time list output.
9: C=====
10:      include 'INC/_IOUNIT'
11:      character*256 FNAME
12:
13: C ... local variables ...
14:      character*261 FILE
15:      character*4   ITOC
16:
17: C-----
18:
19: C ... determine file names for subtasks ...
20:
21:      do I = len(FNAME),1,-1
22:         IE = I
23:         if(FNAME(I:I).ne.' ') goto 99
24:      end do
25: 99 continue
26:      FILE = FNAME(1:IE)//' '//ITOC(IDTASK-1)
27:
28: C ... open file ...
29:
30:      IOS = 0
31:      open(unit=IOTL, file=FILE, form='unformatted', iostat=IOS)
32:      if(IOS.ne.0) then
33:         write(IMG,'(1x, 'XXX(OPEN_TLF) Task ',i5, '': failed to open'',
34:         & ' ' I/O unit ',i2, ' ' (code=' ',i7, ' '))')
35:         & IDTASK, IOTL, IOS
36:      end if
37:
38:      return
39:      end
40:
41:      subroutine CLOSE_TLF(IDTASK)
42: C=====
43: C purpose: Close eparate files from each PE in a parallel
44: C          calculation to output time list data.
45: C-----
46: C arguments (i=input,o=output,w=work)
47: C i IDTASK : MVP task ID (from 1 to NTASK) of current process.
48: C=====
49:      include 'INC/_IOUNIT'
50:      IOS = 0
51:      close(unit=IOTL, status='keep', iostat=IOS)
52:      if(IOS.ne.0) then
53:         write(IMG,'(1x, 'XXX(CLOSE_TLF) Task ',i5,
54:         & '': failed to open I/O unit ',i2, ' ' (code=' ',i7, ' '))')
55:         & IDTASK, IOTL, IOS
56:      end if
57:      return
58:      end
59:
60:      character*4 function ITOC(III)
61:      character*4 CCC
62:      write(CCC,'(i4.4)') III
63:      ITOC = CCC
64:      return
65:      end
```

src/shared/tregnm.f

```

1:      subroutine TREGNM( ITREG, DIRC, NAME, LNM, ITRNM, NTREG,
2:      & TNAMS, NNAMES, A, CHA )
3: C=====
4: C PURPOSE : TALLY REGION # <=> TALLY REGION NAME TRANSLATION.
5: C CALLED IN: TALLYO ETC.
6: C CALLS : REGNM0
7: C-----
8: C arguments ( i = input , o = output , c = constant , w = work )
9: C
10: C io ITREG : TALLY REGION # (GIVE OR GET)
11: C
12: C i DIRC : DIRECTION ('>'/'<' = # > NAME or # < NAME )
13: C
14: C io NAME : TALLY REGION NAME ( GET OR GIVE )
15: C o LNM : length of name ( returned when dirc = '>' )
16: C
17: C i ITRNM(NTREG) : TALLY-REGION NAME OR REGION NUMBER.
18: C FOR TALLY-REGION SPECIFIED IN '#REGION' SECTION IN INEUT:
19: C NAME (POSITION IN 'TNAMS' TABLE * (-1) )
20: C OTHER TALLY-REGION : REGION NUMBER.
21: C i NTREG : total number of tally-regions
22: C i TNAMS(NNAMES) : name table
23: C i NNAMES : number of entries in name table
24: C
25: C io A(*) : dynamic memory array (task shared)
26: C io CHA(*) : dynamic character memory array (task shared)
27: C-----
28: C WHEN ERROR OCCURED:
29: C
30: C # ==> name : name = ' ' & lnm = 0
31: C # <== name : itreg = 0
32: C
33: C=====
34: integer ITREG
35: integer ITRNM(NTREG)
36: character*1 DIRC
37: character*(*) NAME
38: C
39: include 'INC/_LNAME'
40: character*(LNAME) TNAMS(NNAMES)
41: C
42: real A(*)
43: character*4 CHA(*)
44: C
45: C ... local data ...
46: character*128 TMPNAM
47: C
48: if ( DIRC.eq.'>' ) then
49: if ( ITREG.le.0 .or. ITREG.gt.NTREG ) then
50: NAME = ' '
51: LNM = 0
52: return
53: end if
54: if ( ITRNM(ITREG).lt.0 ) then
55: NAME = TNAMS(ABS(ITRNM(ITREG)))
56: LNM = INDEX(NAME,' ') - 1
57: if ( LNM.lt.0 ) LNM = LEN(NAME)
58: else
59: call REGNM0( ITRNM(ITREG), '>', NAME, LNM, A, CHA )
60: end if
61: C
62: else
63: do 100 I = 1, NTREG
64: if ( ITRNM(I).lt.0 ) then
65: if ( NAME.eq.TNAMS(ABS(ITRNM(I))) ) then

```

```

66: ITREG = I
67: return
68: end if
69: else
70: TMPNAM = ' '
71: call REGNM0( ITRNM(I), '>', TMPNAM, LN, A, CHA )
72: if ( NAME.eq.TMPNAM(:LN) ) then
73: ITREG = I
74: return
75: end if
76: end if
77: 100 continue
78: ITREG = 0
79: end if
80: C
81: return
82: end

```

src/shared/tskctl.f

```

1:      subroutine TSKCTL( PROGRAM,      JEIGN, JWNND, JIMPT, JFISS,
2:      &                  JREPR,
3:      &                  NPART, NTHIST,NBATCH,NHIST, NHSUB, NBANK,
4:      &                  IMPMAX,NFBANK,NFBANK0,
5:      &                  NPART0,NHIST0,NHSUB0, NBANK0,NFBANK0 )
6: C=====
7: C purpose: check and reset (if necessary) history control & bank
8: C   parameters for multi tasking.
9: C   And spawn task control process (PVM) if necessary.
10: C
11: C   Input value of the following variable in main task
12: C
13: C   NPART, NHIST, NHSUB, NBANK, NFBANK
14: C
15: C   is saved on the following variables:
16: C
17: C   NPART0,NHIST0,NHSUB0,NBANK0,NFBANK0
18: C
19: C
20: C called in : INTRO2
21: C=====
22:      character*(*) PROGRAM
23: C/#IF INTEGER8
24: *      integer*8 NPART , NTHIST
25: *      integer*8 NPART0
26: *      integer*8 INT8
27: C/#ELSE
28:      integer      NPART , NTHIST
29:      integer      NPART0
30:      integer      INT8
31: C/#ENDIF
32:      external      INT8
33: C
34:      include '../shared/INC/_IOUNIT'
35: C/#IF PARA(MPI)
36: *      include 'mpif.h'
37: C/#ELSEIF PARA(PVM)
38:      include '../shared/INC/_PVM PARA'
39:      character*16 PMTASK
40: C/#ENDIF
41:      include '../shared/INC/_TASKDT'
42: C
43: C ... local variables ...
44: C
45:      real*8 RSP, SPSUM
46:      integer IWRK(16)
47: C/#IF INTEGER8
48: *      integer*8 NPPP, NPART2, NPART9, NTHIST9
49: C/#ELSE
50:      integer      NPPP, NPART2, NPART9, NTHIST9
51: C/#ENDIF
52: C
53: C=====
54: C
55: C ... negative NPART means calculation upto NTHIST+ABS(NPART) histories
56: C
57:      if ( NPART.lt.0 ) then
58:          NPPP = NPART
59:          NPART = ABS(NPART) + NTHIST
60:          write(IPR,7000) NPPP, NPART
61: 7000      format(/1X,'<<MESSAGE>> (TSKCTL) negative NPART value (',I10,
62:      &          '),'//8X,
63:      &          'which means calculation upto NTHIST+ABS(NPART) histories.',
64:      &          ' NPART was changed to ',I10,'.'/)
65:      call CNTERR( 'MESSAGE' )

```

```

66:      end if
67: C
68: C-----
69: C ... start task control process ("tasker") in 'NO-REPRODUCIBLE-RUN'
70: C   mode in distributed memory multitasking mode (currently on PVM)
71: C   Tasker is spawned when "SPAWN-TASK" option is active.
72: C-----
73: C
74:      if ( JREPR.eq.0.and.JEIGN.eq.0 ) then
75: C
76: C/#IF PARA( PVM )
77: C
78:      if ( JTSKR.eq.1 ) then
79:          if ( IDTASK.eq.1 ) then
80:              PMTASK = 'mtask.mvp'
81: C
82:              call PVMFMYTID( MYTID )
83:              if ( MYTID.lt.0 ) then
84:                  call PVMFPERROR( 'TSKCTL', MYTID )
85:                  write(IMG,*) 'XXX PVM error in TSKCTL routine.'
86:                  call CNTERR( 'FATAL' )
87:                  call PRSTOP( 0, 'PVM ERROR.' )
88:                  stop 777
89:              end if
90: C
91: C ... spawn task controller ( ictask: task ID of the process )
92: C
93:              call PVMFSPAWN( PMTASK, PVMDEFAULT, '*', 1, ICTASK, NUMT
94:      &
95: C
96:              if ( NUMT.lt.1 ) then
97:                  write(IMG,*)
98:      &                  'XXX CANNOT START TASK CONTROL PROCESS!! ',
99:      &                  PMTASK
100:                  call CNTERR( 'FATAL' )
101:                  do 100 I = 1, NTASK
102:                      call PVMFKILL( TASKID(I), INFO )
103: 100      continue
104:                  call PVMFEXIT( INFO )
105:                  call PRSTOP( 0, 'PVM ERROR.' )
106:                  stop 777
107:              end if
108:              write(IPR,*) ' A TASK CONTROL PROCESS STARTED : ', PMTASK
109: C
110: C
111: C ... send data to task control process :
112: C
113:              call PVMFINITSEND( PVMDEFAULT, IBUFID )
114: C
115:              call PVMFPACK( INTEGER4, LEN(PROGRAM), 1, 1, INFO )
116:              call PVMFPACK( STRING, PROGRAM, LEN(PROGRAM), 1, INFO )
117: C
118: C/#IF INTEGER8
119: *      call PVMFPACK( INTEGER8, NPART, 1, 1, INFO )
120: C/#ELSE
121:      call PVMFPACK( INTEGER4, NPART, 1, 1, INFO )
122: C/#ENDIF
123:      call PVMFPACK( INTEGER4, NHIST, 1, 1, INFO )
124: C/#IF INTEGER8
125: *      call PVMFPACK( INTEGER8, NTHIST, 1, 1, INFO )
126: C/#ELSE
127:      call PVMFPACK( INTEGER4, NTHIST, 1, 1, INFO )
128: C/#ENDIF
129:      call PVMFPACK( INTEGER4, NTASK, 1, 1, INFO )
130:      call PVMFPACK( INTEGER4, TASKID, NTASK, 1, INFO )

```

src/shared/tskctl.f

```

131: C
132:       call PVMFSEND( ICTASK, 1, INFO )
133: C
134: C ... receive verification message from controller
135: C
136:       call PVMFRECV( ICTASK, -1, INFO )
137: C
138:       call PVMFUNPACK( INTEGER4, IOK, 1, 1, INFO )
139: C
140:       if ( IOK.eq.0 ) then
141:         write(IMG,*)
142:         & 'XXX THE TASK CONTROL PROCESS REJECTED CONTROL JOB.'
143:         call CNTERR( 'FATAL' )
144:         do 110 I = 1, NTASK
145:           call PVMFKILL( TASKID(I), INFO )
146: 110 continue
147:         call PVMFEXIT( INFO )
148:         call PRSTOP( 0, 'ERROR IN TASK CONTROLLER PROCESS' )
149:         stop 777
150:       end if
151: C
152: C ... tell task ID of controller to other task ...
153: C
154:       call PVMFINITSEND( PVMDEFAULT, INFO )
155:       call PVMFPACK( INTEGER4, ICTASK, 1, 1, INFO )
156:       call PVMFMCAST( NTASK-1, TASKID(2), 3, INFO )
157: C
158: C ... task other than task #1 receive ictask.
159: C
160:       else
161:         call PVMFRECV( ITASK0, 3, INFO )
162:         call PVMFUNPACK( INTEGER4, ICTASK, 1, 1, INFO )
163:       end if
164: C
165:       end if
166: C
167: C/#ELSEIF PARA( CRAY* SX* )
168: C
169: C ... set size of "history heap" for shared memory multi tasking mode
170: C
171:       NHHEAP = NPART
172: C
173: C/#ELSEIF PARA
174: C
175: C
176: C
177: C/#ENDIF
178: C
179:       end if
180: C
181: C =====
182:       call HEADER( IPR,
183:         & 'HISTORY OR BATCH CONTROL PARAMETER CHECK FOR MULTI TASKING.'
184:         & )
185: C =====
186: C
187:       NPART0 = NPART
188:       NHIST0 = NHIST
189:       NHSUB0 = NHSUB
190: C
191:       if ( JEIGN.ne.0.and.NFBANK.eq.0 ) then
192:         NFBANK = NHIST
193:         write(IMG,*)
194:         & '!!!(TSKCTL) Fission bank length (NFBANK) is not specified.',
195:         & ' Set to ',NFBANK

```

```

196:       call CNTERR( 'WARNING' )
197:       end if
198: C
199:       NBANK0 = NBANK
200:       NFBANK0 = NFBANK
201: C
202: C
203:       write(IPR,'(1x,' Number of tasks : ',i6/)) NTASK
204: C
205: C
206: C ... histories are (initially) distributed uniformly on tasks ...
207: C
208: C
209:       if ( JRPCPU.eq.0 ) then
210: C
211: C ... for fixed source problem
212: C
213:       if ( JEIGN.eq.0 ) then
214:         NPART = ((NPART+NHIST-1)/NHIST)*NHIST
215: C
216:         NN = NPART/NHIST
217:         if ( MOD(NN,NTASK).ne.0 ) then
218:           NNN = NTASK - MOD(NN,NTASK) + NN
219:           NPART = INT8(NNN)*INT8(NHIST)
220: C
221:         write(IMG,*) '!!!(TSKCTL) Total number of batch (', NN,
222:         & ' ) is not a ', ' multiple of task number (',
223:         & NTASK, ').'
224:         write(IMG,*) ' It is modified to ', NNN
225:         call CNTERR( 'WARNING' )
226:       end if
227:       if ( NPART.ne.NPART0 ) then
228:         write(IMG,*)
229:         & '!!!(TSKCTL) Total history number (NPART) is modified ',
230:         & 'from input value (', NPART0, ' ) to ', NPART
231:         call CNTERR( 'WARNING' )
232:       end if
233: C
234:       NPART = NPART/NTASK
235: C
236: C ... for eigenvalue problem
237: C
238:       else
239:         NHIST = ((NHIST+NTASK-1)/NTASK)*NTASK
240:         if ( NHIST.ne.NHIST0 ) then
241:           write(IMG,*)
242:           & '!!!(TSKCTL) Histories per batch (NHIST=', NHIST0,
243:           & ' ) is not a ', ' multiple of task number (',
244:           & NTASK, ').'
245:           write(IMG,*) ' It is modified to ', NHIST
246:           call CNTERR( 'WARNING' )
247:           NBANK = NHIST*DBLE(NBANK)/DBLE(NHIST0)
248:           NFBANK = NHIST*DBLE(NFBANK)/DBLE(NHIST0)
249:         end if
250: C
251:         NPART = ((NPART+NHIST-1)/NHIST)*NHIST
252: C
253:         if ( NPART.ne.NPART0 ) then
254:           write(IMG,*)
255:           & '!!!(TSKCTL) Total history number (NPART) is modified ',
256:           & 'from input value (', NPART0, ' ) to ', NPART
257:           call CNTERR( 'WARNING' )
258:         end if
259:         nn = npart/nhist
260: C
261:         if( mod(nn,ntask).ne.0 ) then

```

src/shared/tskctl.f

```

261: C      write(ipr,*)
262: C      &      '!!! TOTAL NUMBER OF BATCH (',nn,') IS NOT A MULTIPLE',
263: C      &      ' OF TASK NUMBER (', ntask ,') .'
264: C      call cnterr('WARNING')
265: C      Ccccccc endif
266: C      NPART = NPART/NTASK
267: C      NTHIST = NTHIST/NTASK
268: C      NHIST = NHIST/NTASK
269: C      NBANK = NBANK/NTASK
270: C      NFBANK = NFBANK/NTASK
271: C
272: C      ... set actual size of fission bank array
273: C
274: C      C/IF PARA(CRAY* SX* VPP*)
275: C      NFBNK0 = NFBANK*NTASK
276: C      C/ELSEIF PARA
277: C      if ( IDTASK.eq.1 ) then
278: C      NFBNK0 = NFBANK*NTASK
279: C      else
280: C      NFBNK0 = NFBANK
281: C      end if
282: C      C/ENDIF
283: C      end if
284: C
285: C
286: C      ... calculate histories for each task from relative CPU power
287: C
288: C
289: C      else if ( JRPCPU.ne.0 ) then
290: C
291: C      C/IF PARA(PVM MPI)
292: C
293: C      if ( IDTASK.eq.1 ) then
294: C      SPSUM = 0.0
295: C      do 120 N = 1, NTASK
296: C      SPSUM = SPSUM + TSPEED(N)
297: C      120 continue
298: C
299: C      ... in eigenvalue problem, batch size is changed.
300: C
301: C      if ( JEIGN.ne.0 ) then
302: C      NHIST2 = 0
303: C      do 130 N = 1, NTASK
304: C      NHIST2 = NHIST2 + MAX(1,NINT(NHIST*TSPEED(N)/SPSUM))
305: C      130 continue
306: C
307: C      if ( NHIST.ne.NHIST2 ) then
308: C      write(IMG,*) '!!!(TSKCTL) Number of histories per
309: C      &      'batch is changed to ', NHIST2, ' from ', NHIST
310: C      &      ' to distribute histories proportionally to ',
311: C      &      'assumed CPU power of each task.'
312: C      call CNTERR( 'WARNING' )
313: C      end if
314: C
315: C      NPART2 = ((NPART+NHIST2-1)/NHIST2)*NHIST2
316: C      NBB = NPART2/NHIST2
317: C
318: C      if ( NPART2.ne.NPART0 ) then
319: C      write(IMG,*) '!!!(TSKCTL) TOTAL HISTORY NUMBER ',
320: C      &      '(NPART) IS MODIFIED ',
321: C      &      'FROM INPUT VALUE (', NPART0, ') TO ', NPART2
322: C      call CNTERR( 'WARNING' )
323: C      end if
324: C
325: C

```

```

326: C      ... in fixed source problem , total number of histories is changed.
327: C
328: C      else
329: C
330: C      NPART2 = 0
331: C      do 140 N = 1, NTASK
332: C      NNN = MAX(1,NINT(NPART*TSPEED(N)/SPSUM))
333: C      NNNN = ((NNN+NHIST-1)/NHIST)*NHIST
334: C      NPART2 = NPART2 + NNNN
335: C      140 continue
336: C
337: C      if ( NPART.ne.NPART2 ) then
338: C      write(IMG,*) '!!!(TSKCTL) Total number of histories',
339: C      &      'is changed to ', NPART2, ' from ', NPART,
340: C      &      ' to distribute histories proportionally to ',
341: C      &      'assumed CPU power of each task.'
342: C      call CNTERR( 'WARNING' )
343: C      end if
344: C      end if
345: C
346: C
347: C      .... send batch nhist etc. to subtasks.
348: C
349: C      NFBNK0 = 0
350: C      do 150 N = 2, NTASK
351: C      RSP = TSPEED(N) /SPSUM
352: C      if ( JEIGN.ne.0 ) then
353: C      NHIST9 = MAX(1,NINT(NHIST*RSP))
354: C      NPART9 = NBB*NHIST9
355: C      NTHIST9 = NBATCH*NHIST9
356: C      NBATCH9 = NBATCH
357: C      NBANK9 = MAX(1,INT(NBANK*RSP+1))
358: C      NFBANK9 = MAX(1,INT(NFBANK*RSP+1))
359: C      NFBNK0 = NFBNK0 + NFBANK9
360: C      else
361: C      NHIST9 = NHIST
362: C      NTHIST9 = NTHIST
363: C      NBATCH9 = NBATCH
364: C      NNN = MAX(1,NINT(NPART*RSP))
365: C      NPART9 = ((NNN+NHIST-1)/NHIST)*NHIST
366: C      NBANK9 = NBANK
367: C      NFBANK9 = 0
368: C      end if
369: C
370: C      IWRK(1) = NPART9
371: C      IWRK(2) = NTHIST9
372: C      IWRK(3) = NBATCH9
373: C      IWRK(4) = NHIST9
374: C      IWRK(5) = NBANK9
375: C      IWRK(6) = NFBANK9
376: C
377: C      call MVPCOM_SEND_I4( N, 1, IWRK, 6, INFO )
378: C
379: C      write(IPR,7020) N, NPART9, NHIST9, NBANK9, NFBANK9
380: C
381: C      7020 format(3X,'TASK ',I5,': NPART ',I10,' NHIST ',I8,
382: C      &      ' NBANK ',I10,' NFBANK ',I10)
383: C      150 continue
384: C
385: C
386: C      ... for main task ...
387: C
388: C      RSP = TSPEED(1) /SPSUM
389: C      if ( JEIGN.ne.0 ) then
390: C      NHIST = MAX(1,NINT(NHIST*RSP))

```

src/shared/tskctl.f

```

391:          NTHIST = NHIST*NBATCH
392:          NPART  = NBB*NHIST
393:          NBANK  = MAX(1,INT(NBANK*RSP+1))
394:          NFBANK = MAX(1,INT(NFBANK*RSP+1))
395: C
396: C      .... NFBANK0 of main task is summation of NFBANK over all tasks
397: C
398:          NFBANK0 = NFBANK0 + NFBANK
399:      else
400:          NNN      = MAX(1,NINT(NPART*RSP))
401:          NPART    = ((NNN+NHIST-1)/NHIST)*NHIST
402:          NFBANK   = 0
403:      end if
404:      write(IPR,7020) IDTASK, NPART, NHIST, NBANK, NFBANK
405:
406:      else
407:
408:          IT0      = 1
409:          call MVPCOM_RECV_14( IT0, 1, IWRK, 6, INFO )
410:
411:          NPART    = IWRK(1)
412:          NTHIST   = IWRK(2)
413:          NBATCH   = IWRK(3)
414:          NHIST    = IWRK(4)
415:          NBANK    = IWRK(5)
416:          NFBANK   = IWRK(6)
417:
418:          write(IPR,7020) IDTASK, NPART, NHIST, NBANK, NFBANK
419:          NFBANK0  = NFBANK
420:      end if
421: C/#ELSE
422:      write(IMG,*) 'XXX Irregal multitasking control mode.'
423:      write(IMG,*) ' History assignment according to relative CPU',
424:      &      ' power of each task is not supported in this mode.'
425:      call PRSTOP( 1, 'Illegal multitasking mode.' )
426:      stop 666
427: C/#ENDIF
428:      end if
429: C
430: C      ... adjust NHSUB if necessary.
431: C
432:      if ( NHSUB.gt.NHIST ) then
433:          write(IMG,*)
434:      &      '!!!(TSKCTL) Number of history per sub-batch (NHSUB=',NHSUB,
435:      &      ' ) is larger than than that of batch (NHIST=',NHIST,').'
436:          write(IMG,*) '      Set NHSUB to NHIST.'
437:          call CNTERR('WARNING')
438:          NHSUB = NHIST
439:      end if
440: C
441: C      ... adjust NBANK to honer ratio NBANK/NHSUB.
442: C      when both NBANK and NHSUB is already given in input.
443: C
444:      if ( NBANK0.gt.0.and.NHSUB0.gt.0 ) then
445:          NBB0 = NINT(MAX( 1.0D0, DBLE(NBANK0)/NHSUB0 )*NHSUB)
446:          if ( NBANK.gt.NBB0 ) then
447:              write(IMG,*)
448:      &      '!!!(TSKCTL) Current particle bank size (NBANK=',NBANK,
449:      &      ' ) is larger than that calculated by NHSUB and ',
450:      &      'ratio NBANK/NHSUB as input.'
451:              NBANK = NBB0
452:              write(IMG,*) '      NBANK is set to the value : ',NBANK
453:          end if
454:      end if
455: C
456: C
457: C      if( nbank.eq.0 ) then
458: C          nbank = nhist
459: C          if( jfiss.ne.0.or.jwvnd.ne.0.or.jimpt.ne.0) nbank = 2*nhist
460: C      endif
461: C
462: C      return
463: C      end

```


src/shared/tskmpi.f

```
1:      subroutine TSKMPI( NTASK0, TSKINF )
2: C=====
3: C purpose: Multitasking control in MPI system.
4: C
5: C      * initialize MPI
6: C      * Check parent <--> sub task communication
7: C
8: C called in: center
9: C-----
10: C <<Comments for MPI mode>>
11: C
12: C * Currently, task control parameter file tskinf is not effective.
13: C * Any load balancing is not considered in current MPI mode,
14: C * so this implimentation is more sustable to "real" parallel
15: C * machines rather than clusters which may be heterogeneous.
16: C
17: C * Using default MPI communicator MPI_COMM_WORLD.
18: C * Number of tasks (NTASK0) is set in this routine.
19: C-----
20: C/#IF PARA(MPI)
21: C
22: C      character*(*) TSKINF
23: C
24: C      include 'mpif.h'
25: C
26: C      include 'INC/_TASKDT'
27: C      include 'INC/_IOUNIT'
28: C
29: C ... I/O unit of task control file
30: C
31: C      parameter(ITF = 29)
32: C
33: C ... local data
34: C
35: C      character LINE*80
36: C      character CTEMP*20
37: C      character*64 EXFILE
38: C
39: C ... <note> ...
40: C The declaration and initialization of IERR, IRANK are not required
41: C for 32-bit compilation but the initialization is required for 64-bit
42: C compilation with intel compiler options -r8 and -i8 with
43: C OpenMPI 1.8.4.
44: C      integer IERR
45: C      data IERR/0/
46: C      integer IRANK
47: C      data IRANK/0/
48: C
49: C-----
50: C
51: C ... Start MPI ...
52: C
53: C      call MPI_INIT(IERR)
54: C
55: C      if ( IERR.ne.0 ) then
56: C          write(IMG,*) 'XXX Failed to initialize MPI ',
57: C          & ' (code returned=', IERR, ' )'
58: C          stop 888
59: C      end if
60: C
61: C ... MPI communicator is set to the default one ...
62: C
63: C      MVP_COMM = MPI_COMM_WORLD
64: C
65: C ... Number of Tasks ...
```

```
66: C
67: C      call MPI_COMM_SIZE(MVP_COMM, NTASK0, IERR)
68: C
69: C ... get task ID ( MPI task rank + 1 )
70: C
71: C      call MPI_COMM_RANK(MVP_COMM, IRANK, IERR)
72: C
73: C      IDTASK = IRANK + 1
74: C
75: C-----
76: C
77: C ... this is the parent task
78: C
79: C-----
80: C
81: C      if ( IDTASK.eq.1 ) then
82: C
83: C in future :
84: C ... Parent task inputs task control information file like PVM mode.
85: C
86: C Currently no tskinf file is effective
87: C
88: C      NHOST = 0
89: C      TWAIT = 30.0
90: C
91: C-----
92: C Task control file
93: C
94: C When '*' on the first column, the line is ignored as a comment line.
95: C
96: C      NTASK= number of tasks including the paranet task
97: C      EXFILE= name of excutable file spawned
98: C      HOST= host name[: speed factor]
99: C      Host names on wich subtasks spawned.
100: C      Users can input upto MAXHOST hosts and the hosts are used
101: C      cyclicaly to spawn tasks.
102: C
103: C      "Speed factor" is relative processing speed of the host,
104: C      however, currently this feature is not active.
105: C      ( Default = 1 )
106: C
107: C      TWAIT= maximum waiting time in message sending test.
108: C
109: C-----
110: C
111: C ... set relative speed specification mode, but actually all
112: C tasks are set having an identical speed.
113: C
114: C      JRPCPU = 1
115: C <<comment>> Y.Nagaya 2016/10/29
116: C JRPCPU is changed to 0 in mvp-2.0.38 or later because the relative
117: C speed mode is actually inactive.
118: C      JRPCPU = 0
119: C
120: C      do 190 N = 1, NTASK0
121: C          TSPEED(N) = 1.0
122: C 190 continue
123: C
124: C      end if
125: C
126: C
127: C      .... broadcast TSPEED & JRPCPU
128: C
129: C
130: C      if ( NTASK0.gt.1 ) then
```

src/shared/tskmpi.f

```
131:          IT0 = 1
132:          call MVPCOM_BCAST_R4(IT0, 123, TSPEED, NTASK0, IERR)
133:          call MVPCOM_BCAST_I4(IT0, 124, JRPCPU, 1, IERR)
134:          end if
135: C
136: C ... <note> ...
137: C STPRN2 tries to output from each subtask separately. But the scheme
138: C is not generally accepted.
139: C Work on : VPP5000, Altix, SX6
140: C Not work on : MPICH
141: C
142: C if(STFN(1:10).ne.'NOFILENAME') call STPRN2('ON', IDTASK, STFN)
143: C
144: C write(IPR, (('1')) )
145: C/#ENDIF
146: C return
147: C end
```

src/shared/tskpvm.f

```

1:      subroutine TSKPVM( NTASK0,TSKINF )
2: C=====
3: C purpose: Multitasking control in PVM system.
4: C
5: C      * Enroll in PVM
6: C      * Spawn subtasks
7: C      * Check parent <--> sub task communication
8: C
9: C      Task control parameters are written in file tskinf if any.
10: C
11: C called in: center
12: C=====
13: C/#IF PARA(PVM)
14: C
15:      character*(*) TSKINF
16: C
17:      include 'INC/ PVM PARA'
18:      include 'INC/ TASKDT'
19:      include 'INC/ _IOUNIT'
20: C
21: C      I/O unit of task control file
22: C
23:      parameter( ITF = 29 )
24: C
25: C      parameter ( maxhost = 32 )
26: C      character host(maxhost)*32
27: C      real      speed(maxhost)
28: C
29: C      .. local data
30: C
31: C
32:      character LINE*80
33:      character CTEMP*20
34: C
35:      character*64 EXFILE
36: C
37: C      if( ntask0 .le. 0 ) then
38: C          write(ipr,*) 'XXX For multitasking on PVM,',
39: C      &      ' you must specify number of task on command line ',
40: C      &      ' such as "ntask=...".'
41: C
42: C      call PVMFPERROR( 'PVM error at TSKPVM ', INFO )
43: C          stop
44: C      endif
45: C
46: C      call PVMFMYTID( IDT )
47: C
48: C
49: C      if ( IDT.lt.0 ) then
50: C          write(IMG,*) 'XXX Failed to enroll into PVM ',
51: C      &      ' (tid returned=', IDT, ' )'
52: C          stop 888
53: C      end if
54: C
55: C
56: C
57: C      ... parent task ID ( = PvmNoParent for parent task )
58: C
59: C      call PVMFPARENT( ITASK0 )
60: C
61: C-----
62: C
63: C      ... this is the parent task
64: C
65: C-----

```

```

66: C
67: C      if ( ITASK0.lt.0 ) then
68: C
69: C      ... Parent task inputs task control information file if any.
70: C
71: C      .... nhost > 0 if any host specification, else default host
72: C          allocation by PVM.
73: C
74: C
75: C      NHOST      = 0
76: C      TWAIT      = 30.0
77: C
78: C      EXFILE      = 'PVMexec'
79: C
80: C      LTS          = INDEX(TSKINF,' ') - 1
81: C      if ( LTS.lt.0 ) LTS = LEN(TSKINF)
82: C      write(IPR,*) 'tskinf=<', TSKINF(LTS), '>'
83: C      if ( TSKINF.ne.' ' ) then
84: C          open( ITF, file =TSKINF, form ='FORMATTED', status ='OLD',
85: C      &          iostat =IOS )
86: C          if ( IOS.ne.0 ) then
87: C              write(IMG,*)
88: C      &          'XXX FAILED TO OPEN task control information file <',
89: C      &          TSKINF, '> ERR CODE = ', IOS
90: C              call PVMFEXIT( INFO )
91: C              stop
92: C          end if
93: C
94: C-----
95: C      Task control file
96: C
97: C      When '*' on the first column, the line is ignored as a comment line.
98: C
99: C      NTASK= number of task including paranet task
100: C      EXFILE= name of excutable file spawned
101: C      HOST= host name[: speed factor]
102: C      Host names on wich subtasks spawned.
103: C      Users can input upto MAXHOST hosts and the hosts are used
104: C      cyclicaly to spawn tasks.
105: C
106: C      "Speed factor" is relative processing speed of the host,
107: C      however, currently this feature is not active.
108: C      ( Default = 1 )
109: C
110: C      TWAIT= maximum waiting time in message sending test.
111: C
112: C-----
113: C
114: C      write(IPR,'(/lx,' == TASK CONTROL FILE =='/)')
115: C
116: C
117: C      100      LINE      = ' '
118: C      read(ITF,'(a)',end =110) LINE
119: C
120: C      if ( LINE(1:1).eq.'*' ) go to 100
121: C
122: C      call CCOMP( LINE, LEN(LINE), LINE, IFL )
123: C
124: C      NL          = ICLEN(LINE)
125: C      write(IPR,'(lx,a)') LINE(:NL)
126: C
127: C      if ( 'NTASK='.eq.LINE(1:6) ) then
128: C          CTEMP      = LINE(INDEX(LINE(:NL),'')+1:NL)
129: C          read(CTEMP(:20),'(bn,i20)') NN
130: C

```

src/shared/tskpvm.f

```

131:         if ( NN.ne.NTASK0 ) then
132:             write(IPR,*) ' --> NTASK IS DIFFERENT FROM THAT ',
133: &             'GIVEN BEFORE. RESET THIS VALUE TO THAT OF ',
134: &             'CONTROL FILE.'
135:         end if
136:         NTASK0 = NN
137:     else if ( 'EXFILE='.eq.LINE(1:7) ) then
138:         EXFILE = LINE(8:NL)
139:     else if ( 'HOST='.eq.LINE(1:5) ) then
140:         NHOST = NHOST + 1
141:         if ( NHOST.gt.MAXHOST ) then
142:             write(IPR,*) ' --> too many hosts!! ignore this line.'
143:             NHOST = NHOST - 1
144:         else
145:             KK = INDEX(LINE(6:NL),':')
146:             if ( KK.eq.0 ) then
147:                 HOST(NHOST) = LINE(6:NL)
148:                 HSPEED(NHOST) = 1.0
149:             else
150:                 KK = 5 + KK
151:                 HOST(NHOST) = LINE(6:KK-1)
152:                 CTEMP = LINE(KK+1:NL)
153:                 read(CTEMP(:20),'(bn,f20.0)') HSPEED(NHOST)
154:             end if
155: C
156:         end if
157:     else if ( 'TWAIT='.eq.LINE(1:6) ) then
158:         CTEMP = LINE(INDEX(LINE(:NL),'=')+1:NL)
159:         read(CTEMP(:20),'(bn,f20.0)') TWAIT
160:     end if
161: C
162:     go to 100
163: C
164: 110     continue
165:     write(IPR,*) 'END OF CONTROL FILE'
166: C
167: C
168: C
169: C
170: C     ... try to add host dynamically ...
171: C     ( "PvmDupHost" when adding duplicatehost )
172: C
173: C
174:     do 120 I = 1, NHOST
175:         IL = ICLEN(HOST(I))
176:         call PVMFADDDHOST( HOST(I)(:IL), INFO )
177:         if ( INFO.lt.0.and.INFO.ne.PVMDUPHOST ) then
178:             write(IMG,*) '!!! Failed to add host <', HOST(I)
179: &             (:IL), '>'
180:             call PVMFPERROR( 'TSKPVM', INFO )
181: Ccccc     hspeed(i) = 0.0
182:         else if ( INFO.eq.PVMDUPHOST ) then
183:             write(IPR,*) 'Host <', HOST(I) (:IL),
184: &             '> was already added.'
185:         else if ( INFO.ge.0 ) then
186:             write(IPR,*) 'Host <', HOST(I) (:IL), '> added.'
187:         else
188:             write(IMG,*) '!!!Failed to add Host <',HOST(I)(:IL),
189: &             '> code =', INFO
190:             stop 888
191:         end if
192: 120     continue
193: C
194:     NHERR = 0
195:     do 130 NN = 1, NHOST

```

```

196:         IL = ICLEN(HOST(NN))
197:         call PVMFMSTAT( HOST(NN)(:IL), MSTAT )
198:         if ( MSTAT.eq.PVMOK ) then
199:             write(IPR,*) ' host <', HOST(NN) (:IL), '> is OK.'
200:         else if ( MSTAT.eq.PVMNOHOST ) then
201:             write(IMG,*) 'XXX Host <', HOST(NN) (:IL),
202: &             '> is not in virtual machine.'
203:             NHERR = NHERR + 1
204:         else if ( MSTAT.eq.PVMHOSTFAIL ) then
205:             write(IMG,*) 'XXX Host <', HOST(NN) (:IL),
206: &             '> is unreachable.'
207:             NHERR = NHERR + 1
208:         end if
209: 130     continue
210: C
211:     if ( NHERR.gt.0 ) then
212:         write(IMG,*) 'XXX host error ocured !!'
213:         stop 888
214:     end if
215: C
216:     NN = 0
217:     do 140 I = 1, NHOST
218:         if ( HSPEED(I).gt.0.0 ) then
219:             NN = NN + 1
220:             HOST(NN) = HOST(I)
221:             HSPEED(NN) = HSPEED(I)
222:         end if
223: 140     continue
224:     end if
225: C
226: C
227:     if ( NTASK0.le.0 ) then
228:         write(IMG,*) 'XXX For multitasking on PVM,',
229: &         ' you must specify number of task on command line ',
230: &         'or task control file such as "ntask=...".'
231:         stop
232:     end if
233: C
234: C
235: C     ... when "MYHOSTNAME" parameter (host0) is specified, insert or move
236: C     it to the top of host parameter lists.
237: C     else reset all elements of 'HSPEED' array to 1.0.
238: C
239:     JRPCPU = 0
240:     if ( NHOST.gt.0.and.HOST0.ne.' ' ) then
241:         do 160 N = 1, NHOST
242:             if ( HOST0.eq.HOST(N) ) then
243:                 JRPCPU = 1
244:                 if ( N.gt.1 ) then
245:                     HS = HSPEED(N)
246:                     do 150 K = N, 2, -1
247:                         HOST(K) = HOST(K-1)
248:                         HSPEED(K) = HSPEED(K-1)
249: 150                     continue
250:                     HOST(1) = HOST0
251:                     HSPEED(1) = HS
252:                     write(IPR,*) 'Host rotation is changed as;'
253:                     write(IPR,'(:5x,4(:' : '' : '' ,a))')
254: &                     (HOST(K),K=1,NHOST)
255:                 end if
256:                 go to 170
257:             end if
258: 160         continue
259:         JRPCPU = 0
260: 170     continue

```

```

261:      end if
262: C
263:      if ( NHOST.gt.0.and.JRPCPU.eq.0 ) then
264:          do 180 N = 1, NHOST
265:              HSPEED(N) = 1.0
266:          180      continue
267:      end if
268: C
269: C
270: C
271: C ... spawn task ...
272: C
273: C
274: C ... task 1 is the parent task ...
275: C
276:      IDTASK = 1
277:      TASKID(IDTASK) = IDT
278: C
279: C ... default relative processor speed ...
280: C
281:      do 190 N = 1, NTASK
282:          TSPEED(IDTASK) = 1.0
283:      190      continue
284: C
285: C
286:      NSERR = 0
287:      ILEX = INDEX(EXFILE,' ') - 1
288:      if ( ILEX.lt.0 ) ILEX = LEN(EXFILE)
289: C
290:      if ( NHOST.eq.0 ) then
291:
292:          if ( NTASK0.gt.1 ) then
293:              call PVMFSPAWN( EXFILE(:ILEX), PVMDEFAULT, '*', NTASK0-1,
294:                  &          TASKID(2), NUMT )
295:          else
296:              NUMT = 0
297:          end if
298: C
299:          write(IPR,*) NUMT, ' subtasks are spawned (task=',
300:              &          EXFILE(:ILEX), ') '
301: C
302:          if ( NTASK0-1-NUMT.gt.0 ) then
303:              write(IMG,'(1x,a)')
304:              &          'XXX Failed to spawn required number of tasks.'
305:              NSERR = NTASK0 - 1 - NUMT
306:              call CNTERR( 'FATAL' )
307:          end if
308: C
309:      else
310:          NN = 0
311:          write(IPR,*) NTASK0 - 1, ' subtasks are spawned on ', NHOST,
312:              &          ' hosts (task=', EXFILE(:ILEX), ') '
313: C
314:          if ( JRPCPU.ne.0 ) then
315:              TSPEED(1) = HSPEED(1)
316:              NN = 1
317:          end if
318: C
319:          if ( HOST0.ne.' ' ) then
320:              write(IPR,9000) 1, TASKID(1), HOST0, TSPEED(1)
321:          else
322:              write(IPR,9000) 1, TASKID(1), ' ', TSPEED(1)
323:          end if
324:          9000      format(1X, ' Task ',I5,' taskid: ',I10,' on host <',A,'> ',
325:              &          ' Assumed relative CPU power ',F10.5)

```

```

326: C      do 200 I = 2, NTASK0
327:         if ( NN.eq.NHOST ) NN = 0
328:         NN = NN + 1
329:         IL = ICLEN(HOST(NN))
330:
331: C      call PVMFSPAWN( EXFILE(:ILEX), PVMHOST, HOST(NN)(:IL), 1,
332: &          TASKID(I), NUMT )
333:
334: C      if ( JRPCPU.ne.0 ) TSPEED(I) = HSPEED(NN)
335:
336:      write(IPR,9000) I, TASKID(I), HOST(NN), TSPEED(I)
337:
338: C      if ( NUMT.eq.0 ) then
339:         write(IMG,'(1x,a,a,a)')
340:         & 'XXX Failed to spawn task on host <',
341:         & HOST(NN)(:IL), '>'
342:         NSERR = NSERR + 1
343:         call CNTERR( 'FATAL' )
344:         end if
345:
346: 200      continue
347: C
348:      end if
349: C
350: C
351: C ... When required number of tasks cannot be spawned,
352: C kill all tasks and stop execution.
353: C
354:      if ( NSERR.gt.0 ) then
355:         do 210 I = 2, NTASK0
356:            call PVMFKILL( TASKID(I), INFO )
357: 210      continue
358:         call PRSTOP( 0, 'PVM task initiation error.' )
359:         stop 888
360:      end if
361: C
362: C
363: C .... send family task # & ID s
364: C
365: C
366:      if ( NTASK0.gt.1 ) then
367:         call PVMFINITSEND( PVMDEFAULT, INFO )
368:         call PVMFPACK( INTEGER4, NTASK0, 1, 1, INFO )
369:         call PVMFPACK( INTEGER4, TASKID, NTASK0, 1, INFO )
370:         call PVMFPACK( INTEGER4, JRPCPU, 1, 1, INFO )
371:         call PVMFPACK( REAL4, TSPEED, NTASK0, 1, INFO )
372:         call PVMFMCAST( NTASK0-1, TASKID(2), 99, INFO )
373: C
374: C
375: C
376:         ITT = 0
377:         do 230 I = 2, NTASK0
378: C
379:            call TOKEI( T01, 0 )
380: 220      call PVMFNRCV( TASKID(I), 100, IBUFID )
381:            if ( IBUFID.eq.0 ) then
382:               call TOKEI( T02, 0 )
383:               if ( T02-T01.gt.TWAIT ) then
384:                  write(IMG,*) 'XXX Task ', I, ' (id=', TASKID(I),
385: &          ')', ' DOES NOT RESPOND MORE THAN ', TWAIT,
386: &          ' SECONDS.'
387:                  ITT = ITT + 1
388:                  call CNTERR( 'FATAL' )
389:                  go to 230
390:            else

```

src/shared/tskpvm.f

```

391:          go to 220
392:        end if
393:      end if
394: C
395:      call PVMFUNPACK( INTEGER4, ID, 1, 1, INFO )
396:      if ( TASKID(I).ne.ID ) then
397:        write(IMG,*) 'XXX INVALID TASK ID RETURNED FROM ', I,
398:      &      'TH TASK ( ID = ', TASKID(I), ' RETURNED ',
399:      &      ID, ' )'
400:      ITT = ITT + 1
401:      call CNTERR( 'FATAL' )
402:    else
403:      write(IMG,*) 'SUBTASK ', I, ' RESPONDED NORMALLY.'
404:    end if
405: C
406: 230    continue
407: C
408:      IOK = 1
409:      if ( ITT.gt.0 ) then
410:        write(IMG,*) (1x,a,a/5x,a)
411:      &      'XXX PVM network does not work properly.',
412:      &      ' You may need to do "reset" or "halt" on PVM',
413:      &      ' console manually.'
414:      IOK = 0
415:    end if
416: C
417: C ... send OK (or not OK) flags to sub tasks.
418: C
419:      call PVMFINITSEND( PVMDEFAULT, INFO )
420:      call PVMFPACK( INTEGER4, IOK, 1, 1, INFO )
421:      call PVMFMCST( NTASK0-1, TASKID(2), 199, INFO )
422: C
423:      if ( ITT.gt.0 ) then
424:        call PRSTOP( 1, 'Failed to Startup PVM Network.' )
425:        call PVMFEXIT( INFO )
426:        stop 888
427:      end if
428:    end if
429: C
430: C
431: C-----
432: C
433: C .... subtasks spawned from parent
434: C
435: C-----
436: C
437: C
438:    else
439:      call PVMFRECV( ITASK0, 99, IBUFID )
440:      call PVMFUNPACK( INTEGER4, NTASK0, 1, 1, INFO )
441:      call PVMFUNPACK( INTEGER4, TASKID, NTASK0, 1, INFO )
442:      call PVMFUNPACK( INTEGER4, JRPCPU, 1, 1, INFO )
443:      call PVMFUNPACK( REAL4, TSPEED, NTASK0, 1, INFO )
444:      do 240 IDTASK = 1, NTASK0
445:        if ( IDT.eq.TASKID(IDTASK) ) go to 250
446: 240    continue
447:      IDTASK = 1
448: C
449: C ... return taskid(idtask) as reception
450: C message to check PVM network is working well or not.
451: C
452: 250    call PVMFINITSEND( PVMDEFAULT, INFO )
453:      call PVMFPACK( INTEGER4, TASKID(IDTASK), 1, 1, INFO )
454:      call PVMFSEND( ITASK0, 100, INFO )
455: C

```

```

456: C ... receive OK (or not OK ) flags
457: C
458:      call PVMFRECV( ITASK0, 199, IBUFID )
459:      call PVMFUNPACK( INTEGER4, IOK, 1, 1, INFO )
460:      if ( IOK.eq.0 ) then
461:        write(IMG,*)
462:      &      'XXX RECEIVED "NOT OK" MESSAGE FROM PARENT TASK. STOP EXECUTION.'
463:      call PVMFEXIT( INFO )
464:      stop 888
465:    end if
466:  end if
467: C
468: C
469: C ... comment ...
470: C STPRN2 trys to output from each subtask separately. But the scheme
471: C is not generally accepted.
472: C Work on : VPP5000 with MPI
473: C Not work on : MPICH
474: C
475: C      call STPRN2('ON', IDTASK, STFN)
476: C
477:      write(IPR,('1'))
478: C/#ENDIF
479:      return
480:    end

```

src/shared/tsksm.f

```

1:      subroutine TSKSMI( NAME,  IA,    IB,    NE )
2:      character*(*) NAME
3:      integer IA(NE), IB(NE)
4:      real*8 DDUMA, DDUMB
5:      C/#IF INTEGER8
6:      *      integer*8 I8DUMA, I8DUMB
7:      C/#ELSE
8:      integer    I8DUMA, I8DUMB
9:      C/#ENDIF
10:      IMOD      = 1
11:      call TSKSM0( NAME, IMOD, NE, IA, IB, DUMA, DUMB, DDUMA, DDUMB,
12:      &          I8DUMA, I8DUMB )
13:      return
14:      end
15:
16:      subroutine TSKSMR( NAME,  A,    B,    NE )
17:      character*(*) NAME
18:      real A(NE), B(NE)
19:      real*8 DDUMA, DDUMB
20:      C/#IF INTEGER8
21:      *      integer*8 I8DUMA, I8DUMB
22:      C/#ELSE
23:      integer    I8DUMA, I8DUMB
24:      C/#ENDIF
25:      IMOD      = 2
26:      call TSKSM0( NAME, IMOD, NE, IDUMA, IDUMB, A, B, DDUMA, DDUMB,
27:      &          I8DUMA, I8DUMB )
28:      return
29:      end
30:
31:      subroutine TSKSMD( NAME,  DA,    DB,    NE )
32:      character*(*) NAME
33:      real*8 DA(NE), DB(NE)
34:      C/#IF INTEGER8
35:      *      integer*8 I8DUMA, I8DUMB
36:      C/#ELSE
37:      integer    I8DUMA, I8DUMB
38:      C/#ENDIF
39:      IMOD      = 3
40:      call TSKSM0( NAME, IMOD, NE, IDUMA, IDUMB, DUMA, DUMB, DA, DB,
41:      &          I8DUMA, I8DUMB )
42:      return
43:      end
44:
45:      subroutine TSKSMI8( NAME, I8A, I8B, NE )
46:      character*(*) NAME
47:      C/#IF INTEGER8
48:      *      integer*8 I8A(NE), I8B(NE)
49:      C/#ELSE
50:      *      integer    I8A(NE), I8B(NE)
51:      C/#ENDIF
52:      real*8 DDUMA, DDUMB
53:      IMOD      = 4
54:      call TSKSM0( NAME, IMOD, NE, IDUMA, IDUMB, RDUMA, RDUMB,
55:      &          DDUMA, DDUMB, I8A, I8B )
56:      return
57:      end
58:
59:      subroutine TSKSM0( NAME,  IMOD, NE,  IA,    IB,    A,    B,
60:      &          DA,    DB , I8A,    I8B )
61:      C=====
62:      C purpose: elementwise summation of two arrays on tasks.
63:      C          a(i) = a(i) + b(i)  (i = 1, ne)
64:      C
65:      C      tsksmi : integer array

```

```

66:      C      tsksmr : real array
67:      C      tsksmd : double precision real array
68:      C
69:      C      In PVM & VPP mode, summation are done on array ib,b & db, and
70:      C      arguments ia , ra & da have no meaning.
71:      C
72:      C=====
73:      character*(*) NAME
74:      integer IA(NE), IB(NE)
75:      real A(NE), B(NE)
76:      real*8 DA(NE), DB(NE)
77:      C/#IF INTEGER8
78:      *      integer*8 I8A(NE), I8B(NE)
79:      C/#ELSE
80:      integer    I8A(NE), I8B(NE)
81:      C/#ENDIF
82:      C
83:      include 'INC/_WORDL'
84:      include 'INC/_IUNIT'
85:      C
86:      C/#IF PARA(PVM MPI)
87:      C
88:      C/#IF PARA(PVM)
89:      include 'INC/_PVM PARA'
90:      CCCC/#ELSEIF PARA(PVM)
91:      C/#ELSEIF PARA(MPI)
92:      *      include 'mpif.h'
93:      C/#ENDIF
94:      include 'INC/_TASKDT'
95:      C
96:      parameter( MXBUF      = 2048 )
97:      integer IBUF(MXBUF)
98:      real RBUF(MXBUF)
99:      real*8 DBUF(1)
100:      C
101:      equivalence( IBUF(1), RBUF(1), DBUF(1) )
102:      C
103:      C-----
104:      C
105:      C/#ENDIF
106:
107:      CHECK %%%%%%%%%%%%%%%%%%%%%%%%%
108:      C      write(*,*) '%%% TSKSM task ', IDTASK, ' DATA <', NAME, '> mode ',
109:      C      &          IMOD, ' size ', NE
110:      C %%%%%%%%%%%%%%%%%%%%%%%%%
111:
112:      C/#IF PARA( CRAY* SX* )
113:      C
114:      C ..... Shared memory parallel
115:      C
116:      *      if( IMOD.eq.1 ) then
117:      *          call SUMONI( IA,IB,NE )
118:      *      else if( IMOD.eq.2 ) then
119:      *          call SUMONR( A,B,NE )
120:      *      else if( IMOD.eq.3 ) then
121:      *          call SUMOND( DA,DB,NE )
122:      *      else if( IMOD.eq.4 ) then
123:      *          call SUMONI8( I8A, I8B, NE )
124:      *      endif
125:      C
126:      C/#ELSEIF PARA( VPP )
127:      C
128:      C ..... VPP Fortran parallel
129:      C
130:      *      if( IMOD.eq.1 ) then

```

src/shared/tsksm.f

```

131: *      call PESUMI( IB,NE )
132: *      else if( IMOD.eq.2 ) then
133: *          call PESUM4( B,NE )
134: *      else if( IMOD.eq.3 ) then
135: *          call PESUM8( DB,NE )
136: *      else if( IMOD.eq.4 ) then
137: *          call PESUMI8( I8B, NE )
138: *      endif
139: C
140: C/#ELSEIF PARA( PVM MPI )
141: C
142: C ..... Distributed memory parallel
143: C
144: C      IT0 = 1
145: C      if ( IMOD.eq.1 ) then
146: C          call MVPCOM_SUM_I4( IT0, 234, IB, NE, IBUF, MXBUF, INFO )
147: C      else if ( IMOD.eq.2 ) then
148: C          call MVPCOM_SUM_R4( IT0, 234, B, NE, RBUF, MXBUF, INFO )
149: C      else if ( IMOD.eq.3 ) then
150: C          call MVPCOM_SUM_R8( IT0, 234, DB, NE, DBUF, MXBUF/MWORD, INFO )
151: C      else if ( IMOD.eq.4 ) then
152: C          call MVPCOM_SUM_I8( IT0, 234, I8B, NE, IBUF, MXBUF, INFO )
153: C      end if
154: C
155: C Summation process of this version is simple summation on parent task
156: C ( implimentaion of elaborate algorithm such as tree summation is
157: C for future modification )
158: C
159: C ... non parent task sends data ...
160: C
161: C      MS = MXBUF
162: C      if ( IMOD.eq.3 ) MS = MXBUF/MWORD
163: C
164: Cccc if ( ITASK0.ge.0 ) then
165: C      if ( IDTASK.gt.1 ) then
166: C
167: C          ... send number of data & type ...
168: C
169: C          IBUF(1) = NE
170: C          IBUF(2) = IMOD
171: C          IT0 = 1
172: C          call MVPCOM_SEND_I4( IT0, 234, IBUF, 2, INFO )
173: C
174: C          do 100 K = 1, NE, MS
175: C              K2 = MIN(NE,K+MS-1)
176: C              NN = K2 - K + 1
177: C              if ( IMOD.eq.1 ) then
178: C                  call MVPCOM_SEND_I4( IT0, K, IB(K), NN, INFO )
179: C              else if ( IMOD.eq.2 ) then
180: C                  call MVPCOM_SEND_R4( IT0, K, B(K), NN, INFO )
181: C              else if ( IMOD.eq.3 ) then
182: C                  call MVPCOM_SEND_R8( IT0, K, DB(K), NN, INFO )
183: C              end if
184: C
185: C 100 continue
186: C
187: C ... get reception signal ...
188: C
189: C      call MVPCOM_RECV_I4( IT0, 1999, IGET, 1, INFO )
190: C
191: C .... paranet task ...
192: C
193: C      else
194: C          do 150 N = 2, NTASK
195: C              IT0 = 1

```

```

196: C      call MVPCOM_RECV_I4( N, 234, IBUF, 2, INFO )
197: C      NNE = IBUF(1)
198: C      ITYP = IBUF(2)
199: C
200: C      if ( NNE.ne.NE ) then
201: C          write(IPR,*) 'XXX tsksm : <', NAME,
202: C              '> data number mismatch.'
203: C          &
204: C          write(IPR,*) ' parent : ', NE, ' task(', N, ') ', NNE
205: C          stop 666
206: C      end if
207: C
208: C      if ( ITYP.ne.IMOD ) then
209: C          write(IPR,*) 'XXX tsksm : <', NAME,
210: C              '> data type mismatch.'
211: C          &
212: C          write(IPR,*) ' parent : ', IMOD, ' task(', N, ') ',
213: C              ITYP
214: C          stop 666
215: C      end if
216: C
217: C      do 140 K = 1, NE, MS
218: C          K2 = MIN(NE,K+MS-1)
219: C          NN = K2 - K + 1
220: C
221: C          if ( IMOD.eq.1 ) then
222: C              call MVPCOM_RECV_I4( N, K, IBUF, NN, INFO )
223: C              do 110 I = 1, NN
224: C                  IB(K+I-1) = IB(K+I-1) + IBUF(I)
225: C              continue
226: C          else if ( IMOD.eq.2 ) then
227: C              call MVPCOM_RECV_R4( N, K, RBUF, NN, INFO )
228: C              do 120 I = 1, NN
229: C                  B(K+I-1) = B(K+I-1) + RBUF(I)
230: C              continue
231: C          else if ( IMOD.eq.3 ) then
232: C              call MVPCOM_RECV_R8( N, K, DBUF, NN, INFO )
233: C              do 130 I = 1, NN
234: C                  DB(K+I-1) = DB(K+I-1) + DBUF(I)
235: C              continue
236: C          end if
237: C
238: C 140 continue
239: C
240: C      ... send reception signal
241: C
242: C      IGET = 1
243: C      call MVPCOM_SEND_I4( N, 1999, IGET, 1, INFO )
244: C      continue
245: C      end if
246: C
247: C/#ELSE
248: C
249: C      return
250: C      end

```


src/shared/typbod.f

```
1:      subroutine TYPBOD( BTYPE, DIREC, IBTYP )
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C PURPOSE :   Body type string <=> body type number
7: C             ( DIREC  '>' : string -> NUMBER )
8: C             '<' : string <- NUMBER )
9: C=====
10:      character BTYPE*(*), DIREC*1
11: C
12:      parameter ( MB = 19 )
13:      character*4 BTP(MB)
14: C
15:      data BTP /
16:      & 'RPP ', 'BOX ', 'SPH ', 'RCC ', 'CYL ',
17:      & 'RCL ', 'IRC ', 'RHP ', 'HEX ', 'ELL ',
18:      & 'ARB ', 'WED ', 'RAW ', 'HAF ', 'GEL ',
19:      & 'ELT ', 'TEC ', 'GQS ', 'BBC ' /
20: C
21:      if ( DIREC.eq.'>' ) then
22:      do 100 I=1,MB
23:      if( BTYPE.eq.BTP(I) ) then
24:      IBTYP = I
25:      return
26:      end if
27: 100  continue
28:      IBTYP = 0
29:      else
30:      if( IBTYP.ge.1.and.IBTYP.le.MB ) then
31:      BTYPE = BTP(IBTYP)
32:      return
33:      end if
34:      BTYPE = ' '
35:      end if
36: C
37:      return
38:      end
```

src/shared/usrsrc.f

```

1:      subroutine usrsrc( type, ns, irand,
2:      &                  data, ndata, rwk, iwk,
3:      &                  x, y, z, a, b, c, e, t, w,
4:      &                  ierr )
5: c==<MVP/GMVP>=====
6: c purpose: generation of source particle from user supplied source.
7: c
8: c << this is a skelton of user supplied source routine >>
9: c
10: c Users can add sampling statements to this routine.
11: c Examples are shown in subroutine SYSSRC.
12: c
13: c-----
14: c
15: c << arguments >>
16: c
17: c (usage symbol: i=input, o=output, io=input & output, w=work )
18: c
19: c i type : user supplied source type (character string as 'USER1')
20: c i ns : number of source particles to be generated
21: c io irand : seed of random number (INTEGER)
22: c i data : input data to specify users source property (REAL*8)
23: c i ndata : number of effective data elements stored in 'data' array
24: c w rwk : working array of type REAL dimensioned as rwk(ns,4)
25: c w iwk : working array of type INTEGER dimensioned as iwk(ns,4)
26: c        Rwk and iwk occupy identical memory area, so users
27: c        should take care in using them.
28: c
29: c Users must give values for all or part of the following parameters.
30: c
31: c Beware that users cannot specify kind of generated particles as
32: c neutron, photon etc., and cannot give energy group #'s directly in
33: c multigroup calculation.
34: c
35: c o x,y,z : source position (unit: cm)
36: c o a,b,c : direction cosines (a**2 + b**2 + c**2 = 1.0)
37: c o e : energy (unit: eV)
38: c o t : time (unit: second)
39: c o w : particle weight (ordinary = 1.0 without biasing)
40: c
41: c Users must set and return an error code as follows:
42: c
43: c o ierr : .eq. 0 if sampling succeeded.
44: c        .ne. 0 if any sampling errors occurred.
45: c
46: c
47: c << Random number generation >>
48: c
49: c Use of 'RANU2' routine is recommended to keep consistency with
50: c other parts of the code.
51: c
52: c call ranu2( irand, r, n, icode )
53: c
54: c irand : seed of random number (renewed after each call)
55: c r : array of type REAL to save random number(s)
56: c n : number of random numbers generated
57: c icode : error code
58: c
59: c=====
60: c
61: c character*(*) type
62: c
63: c ... user supplied data as input ....
64: c
65: c real*8 data(ndata)

```

```

66: c
67: c ... parameters to be sampled ....
68: c
69: c real*8 x(ns), y(ns), z(ns), a(ns), b(ns), c(ns)
70: c real*8 e(ns), t(ns), w(ns)
71: c
72: c ... working area ( length 4*nr ) ...
73: c
74: c Rwk and iwk starts from the same address!!
75: c
76: c real rwk(ns,4)
77: c integer iwk(ns,4)
78: c##<2007/03/14:PN3:
79: c
80: c ... local variable ...
81: c
82: c real*8 xx,yy,zz,aa,bb,cc,ee,ww,nnnsrcc
83: c data iup,nnnsrcc / 0, 0.d0 /
84: c
85: c##>
86: c-----
87: c ierr = 0
88: c
89: c
90: c
91: c
92: c ==== description of users source ====
93: c
94: c
95: c
96: c
97: c * if( type.eq.'USER1' ) then
98: c * else if( type.eq.'USER2' ) then
99: c * else if( type.eq.'USER3' ) then
100: c * endif
101: c
102: c
103: c##<2007/03/14:PN3:
104: c
105: c .... using the photon source file from mcnpix
106: c
107: c if ( iup.eq.0 ) then
108: c open(79,file='f79.'//type(1:5),form='unformatted',status='old')
109: c iup = 79
110: c end if
111: c ii = 0
112: c
113: c 100 continue
114: c read(79,end=991) xx,yy,zz,aa,bb,cc,ee,ww,id
115: c if ( id.eq.1 ) then
116: c ii = ii + 1
117: c x(ii) = xx
118: c y(ii) = yy
119: c z(ii) = zz
120: c a(ii) = aa
121: c b(ii) = bb
122: c c(ii) = cc
123: c e(ii) = ee * 1.0d+6
124: c w(ii) = ww
125: c t(ii) = 0
126: c nnnsrcc = nnnsrcc + 1
127: c end if
128: c if ( ii.lt.ns ) go to 100
129: c return
130: c

```

src/shared/usrsrc.f

```
131: 991 write(6,901) nnnsr
132: 901 format(
133:  &' '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
134:  &' '!!! ... subroutine usrsr ... !!!'
135:  &' '!!! end of file is detected. !!!'
136:  &' '!!! number of photon source is lack. !!!'
137:  &' '!!! nnnsr =',lpe15.8,' ' !!!'
138:  &' '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
139: rewind 79
140: nnnsr = 0
141: go to 100
142: c##>
143: c
144: c
145: c
146: c === error message when no user supplied source. ===
147: c
148: c
149: c
150: c
151: c
152: 9999 write(6,*) '=== No user supplied source <',type,'> !'
153: ierr = 1
154: return
155: end
```

src/shared/value0.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine VALUE0( VNAME, ICALL, TYP,   X )
3: C
4: C   JAERI MONTE CARLO CODE UTILITY
5: C
6: C=====
7: C PURPOSE:  ASSIGN A VALUE TO VARIABLE X.
8: C           ( X may be single or double precision but declared as
9: C             real type in this sroutine )
10: C CALLED IN:  INTRO
11: C CALLS:  I4READ,R4READ,R8READ
12: C-----
13: C arguments (i=input, o=output, w=work)
14: C i VNAME : data name string
15: C o ICALL : set to one. Used to judge that this routine is called
16: C           at least once etc.
17: C i TYP  : data type string
18: C o X(*) : array data on which data are stored.
19: C-----
20:   character TYP*2, VNAME*(*)
21:   real X(*)
22:   include 'INC/_WORDL'
23: C
24:   ICALL   = 1
25: C
26:   NLV     = INDEX(VNAME,' ') - 1
27:   if ( NLV.lt.0 ) NLV = LEN(VNAME)
28: C
29:   if ( TYP.eq.'I4' ) call I4READ( VNAME(1:NLV), X, NA, -1, IERR )
30:   if ( TYP.eq.'I8' ) call I8READ( VNAME(1:NLV), X, NA, -1, IERR )
31:   if ( TYP.eq.'R4' ) call R4READ( VNAME(1:NLV), X, NA, -1, IERR )
32:   if ( TYP.eq.'R8' ) call R8READ( VNAME(1:NLV), X, NA, -1, IERR )
33: C
34:   return
35: end
```

src/shared/varreg.f

```

1:      subroutine VARREG( ISCOPE,VNAME, TYP,   X )
2:      C=<MVP/GMVP>=====
3:      C Purpose: registration of scalar variables to enable checking of
4:      C      a certain variable is already defined or not.
5:      C Called in: (anywhere)
6:      C=====
7:      C arguments (i=input, o=output, w=work )
8:      C i iscope : "Scope" is used to control the range of "visibility" of
9:      C      registered variable. ISCOPE=0 means global visibility all over
10:     C      the program. ISCOPE > 0 can be specified to control visibility
11:     C      of a variable effective in a specified subroutine only etc.
12:     C i vname : name of scalar variable registered.
13:     C i typ   : variable type ( I4/R4/R8 = integer/single real/double real)
14:     C i x     : value of variable. This argument is declared as "integer"
15:     C      array but the real data type on the calling side may be real or
16:     C      double. Only the starting address has meanings here.
17:     C      (ie. assuming argument passing convention of "by name".)
18:     C=====
19:     integer ISCOPE
20:     character*(*) VNAME
21:     character*(*) TYP
22:     integer X(*)
23: C
24:     include 'INC/_IUNIT'
25:     include 'INC/_WORDL'
26: C
27: C ... data storage of registered variables
28: C
29:     parameter( MXVARS   = 128 )
30:     parameter( LENVAR   = 8 )
31:     character*(LENVAR) VARNAM(MXVARS)
32:     character*4 VARTYP(MXVARS)
33:     integer VARVAL(2,MXVARS)
34:     integer IVSCOP(MXVARS)
35:     common /VARRG1/  VARNAM, VARTYP
36:     common /VARRG2/  VARVAL, IVSCOP, NVARS
37: C
38: C --- external filter functions to convert data type
39: C
40:     real CNVTOR
41:     real*8 CNVTOD
42: C
43: C-----
44:     LV   = MIN(LEN(VNAME),LENVAR)
45: C
46:     IV   = 0
47:     if ( NVARS.gt.0 ) then
48:       do 100 I = 1, NVARS
49:         if ( IVSCOP(I).eq.ISCOPE.and.VNAME(:LV).eq.VARNAM(I) ) then
50:           IV   = I
51:           if ( VARTYP(I).ne.TYP ) then
52:             write(IMG,*) 'XXX (internal variable registration)',
53:             &           ' Attempt to register same variable',
54:             &           ' with different data type.'
55:             write(IMG,*) '   VNAME <', VARNAM(I), '> Type <',
56:             &           VARTYP(I), ' --> <', TYP, '>'
57:             call PRSTOP( 1,
58:             &           'Error on internal variable registration' )
59:             stop 666
60:           end if
61:           go to 120
62:         end if
63:       100 continue
64:     end if
65: C

```

```

66: C ... newly registered variable ...
67: C
68:     if ( NVARS.ge.MXVARS ) then
69:       write(IMG,*) 'XXX (internal variable registration) Too many',
70:       &           ' attempt to register variable (limit= ', MXVARS,
71:       &           ' variables)'
72: C
73:       do 110 I = 1, NVARS
74:         if ( VARTYP(I).eq.'I4' ) then
75:           write(IMG,7000) VARNAM(I), VARTYP(I), IVSCOP(I),
76:           &           VARVAL(1,I)
77:         else if ( VARTYP(I).eq.'R4' ) then
78:           write(IMG,7020) VARNAM(I), VARTYP(I), IVSCOP(I),
79:           &           CNVTOR(VARVAL(1,I))
80:         else if ( VARTYP(I).eq.'R8' ) then
81:           write(IMG,7040) VARNAM(I), VARTYP(I), IVSCOP(I),
82:           &           CNVTOD(VARVAL(1,I))
83:         end if
84:       110 continue
85:       7000 format(3X,'NAME: <','A,> type <','A,> scope <','I3,>: ',I13)
86:       7020 format(3X,'NAME: <','A,> type <','A,> scope <','I3,>: ',E13.5)
87:       7040 format(3X,'NAME: <','A,> type <','A,> scope <','I3,>: ',E13.5)
88: C
89:       write(IMG,*)
90:       &           ' You may need modification for VARREG routine etc.'
91:       call PRSTOP( 1,
92:       &           'Too many attempt of internal variable registration' )
93:       stop 666
94:     end if
95: C
96:     NVARS   = NVARS + 1
97:     VARNAM(NVARS) = VNAME
98:     IVSCOP(NVARS) = ISCOPE
99:     VARTYP(NVARS) = TYP
100: C
101:     IV      = NVARS
102: C
103: 120 continue
104: C
105: C ... register ...
106: C
107:     if ( TYP.ne.'I4'.and.TYP.ne.'R4'.and.TYP.ne.'R8' ) then
108:       write(IMG,*) 'XXX (internal variable registration)',
109:       &           ' attempt to register variable with invalid data type <',
110:       &           TYP, '> (allowed typ I4/R4/R8)'
111:       write(IMG,7060) VARNAM(IV), IVSCOP(IV)
112: 7060 format(3X,'   Variable <','A,> Scope <','I3,>:')
113:       call PRSTOP( 1, 'Error on internal variable registration' )
114:       stop 666
115:     end if
116: C
117:     VARTYP(IV) = TYP
118: C
119:     if ( TYP.eq.'I4' ) then
120:       VARVAL(1,IV) = X(1)
121:     else if ( TYP.eq.'R4' ) then
122:       VARVAL(1,IV) = X(1)
123:     else if ( TYP.eq.'R8' ) then
124:       if ( MWORD.eq.2 ) then
125:         VARVAL(1,IV) = X(1)
126:         VARVAL(2,IV) = X(2)
127:       else
128:         VARVAL(1,IV) = X(1)
129:       end if
130:     end if

```

src/shared/varreg.f

```

131: C
132:   return
133:   end
134: C
135: C
136: C
137:   function IVARCK(ISCOPE,VNAME,TYP,X)
138: C=<MVP/GMVP>=====
139: C Purpose: check a scalar variable is registered with VARREG routine,
140: C   and return the registered value if any.
141: C
142: C Returns <scope> if specified variable is registered else returns -1.
143: C
144: C Called int (anywhere)
145: C history: Aug 1995 (M.Sasaki)
146: C=====
147: C arguments (i=input, o=output, w=work )
148: C i iscope : "Scope" is used to control the range of "visibility" of
149: C   registered variable. ISCOPE=0 means global visibility all over
150: C   the program.
151: C   iscope >= 0 :
152: C     Variable having a name "VNAME" is searched among registered
153: C     ones having scope value less than or equal to iscope.
154: C   iscope < 0 :
155: C     Variable having a name "VNAME" is searched among registered
156: C     ones having scope value equal to abs(iscope).
157: C i vname : name of scalar variable registered.
158: C i typ : variable type ( I4/R4/R8 = integer/single real/double real)
159: C o x : value of variable. This argument is declared as "integer"
160: C   array but the real data type on the calling side may be real or
161: C   double. Only the starting address has meanings here.
162: C   (ie. assuming argument passing convention of "by name".)
163: C=====
164:   integer ISCOPE
165:   character*(*) VNAME
166:   character*(*) TYP
167:   integer X(*)
168: C
169:   include 'INC/_IOUNIT'
170:   include 'INC/_WORDL'
171: C
172: C ... data storage of registered variables
173: C
174:   parameter( MXVARS = 128 )
175:   parameter( LENVAR = 8 )
176:   character*(LENVAR) VARNAM(MXVARS)
177:   character*4 VARTYP(MXVARS)
178:   integer VARVAL(2,MXVARS)
179:   integer IVSCOP(MXVARS)
180:   common /VARRG1/ VARNAM, VARTYP
181:   common /VARRG2/ VARVAL, IVSCOP, NVAR
182: C
183: C --- external filter functions to convert data type
184: C
185:   real CNVTOR
186:   real*8 CNVTOD
187: C
188: C-----
189:   LV = MIN(LEN(VNAME),LENVAR)
190: C
191:   IVARCK = -1
192: C
193:   IV = 0
194:   if ( NVAR.eq.0 ) then
195:     return

```

```

196:   else
197:     if ( ISCOPE.lt.0 ) then
198:       do 100 I = 1, NVAR
199:         if ( IVSCOP(I).eq.ABS(ISCOPE).and.VNAME(:LV).eq.VARNAM(I)
200:           & .and.TYP.eq.VARTYP(I) ) then
201:           IV = I
202:           go to 130
203:         end if
204:       100 continue
205:     else
206:       do 120 ISCP = ISCOPE, 0, -1
207:         do 110 I = 1, NVAR
208:           if ( IVSCOP(I).eq.ISCP.and.VNAME(:LV).eq.VARNAM(I)
209:             & .and.TYP.eq.VARTYP(I) ) then
210:             IV = I
211:             go to 130
212:           end if
213:         110 continue
214:       120 continue
215:     end if
216:     return
217:   end if
218: C
219: C ... registered variable ...
220: C
221: 130 continue
222:   IVARCK = 1
223: C
224: C
225:   VARTYP(IV) = TYP
226: C
227:   if ( TYP.eq.'I4' ) then
228:     X(1) = VARVAL(1,IV)
229:   else if ( TYP.eq.'R4' ) then
230:     X(1) = VARVAL(1,IV)
231:   else if ( TYP.eq.'R8' ) then
232:     if ( MWORD.eq.2 ) then
233:       X(1) = VARVAL(1,IV)
234:       X(2) = VARVAL(2,IV)
235:     else
236:       X(1) = VARVAL(1,IV)
237:     end if
238:   end if
239:   return
240: end

```

```
src/shared/vdentk.f
```

[illegible]

```
66: C  
67:     parameter( MSV = 32 )  
68:     integer LVSTK(MSV)  
69:     integer NVSTK  
70: C  
71: C      .... VECTOR VARIABLE & POINTER .....  
72: C  
73:     parameter( MVV = 32 )  
74:     character*16 VVAR(MVV)  
75:     integer LVVAR(MVV)  
76:     integer NVVAR  
77: C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  
78: C  
79: C  
80: C ===== LOCAL VARIABLES =====  
81: C  
82:     integer ISTK(MS)  
83:     integer LHSTK(MSV)  
84:     integer NARG(MS)  
85:     integer IVTYP(MSV)  
86:     character*64 FORMT  
87:     real*8 TVALUE  
88: C  
89: C      .... WORKING BUFFER  
90: C  
91: C      LIN:   input 'LINE' is compressed to this array.  
92: C      IPLIN: original position in 'LINE'  
93: C  
94:     parameter( MAXCOL = 256 )  
95:     character LIN*(MAXCOL)  
96:     integer IPLIN(MAXCOL)  
97: C  
98:     character OPERTR*9, NUM*11  
99: C  
100: C      .... FUNCTION SYMBOLS FUNC) & ARGUMRNT NUMBERS(NFARG) ....  
101: C          ( NFARG < 0 : at least -NFARG arguments required)  
102: C  
103:     parameter( NFUNC = 33 )  
104:     character*6 FUNC(NFUNC)  
105:     integer NFARG(NFUNC)  
106: C  
107: C =====  
108: C  
109: C  
110:     parameter( KFUNC = 0 )  
111:     parameter( KEND = 1 )  
112:     parameter( KLBRC = 2 )  
113:     parameter( KRBC = 3 )  
114:     parameter( KPLUS = 4 )  
115:     parameter( KMINUS = 5 )  
116:     parameter( KMULT = 6 )  
117:     parameter( KDIV = 7 )  
118:     parameter( KPOW = 8 )  
119:     parameter( KCOMMA = 9 )  
120:     parameter( KEQUAL = 10 )  
121:     parameter( KUNMNS = 11 )  
122:     parameter( NMOPR = KUNMNS )  
123: C  
124: C      .... TABLE OF CALCULATION RULES .....  
125: C  
126:     integer ICALC(0:NMOPR,0:NMOPR)  
127: C  
128:     include 'INC/_WORDL'  
129: C  
130:     parameter( IECHO = 66 )
```

src/shared/vdentk.f

```

131: C
132: C -----
133: C Data initialization
134: C -----
135: C
136: C
137: C .... OPERATOR, NUMBER ....
138: C ( POWER OPERATOR '**' WILL BE REPLACED BY '#' )
139: C
140: C DATA OPERTR / '( )+-*/#,,=' /, NUM /'0123456789.'/
141: C
142: C DATA (FUNC(I),I=1,NFUNC) /
143: C & 'EXP','LOG','INT','SIN','COS',
144: C & 'TAN','ASIN','ACOS','ATAN','SQRT',
145: C & 'ABS','MAX','MIN','MOD','LOG10',
146: C & 'SIND','COSD','TAND',
147: C & 'IF','AND','OR',
148: C & 'NOT','EQ','NE','GT','GE',
149: C & 'LT','LE','NINT','ATAN2',
150: C & 'SINH','COSH','TANH' /
151: C
152: C NFARG : function argument number required
153: C
154: C DATA (NFARG(I),I=1,NFUNC) /
155: C & 1, 1, 1, 1, 1,
156: C & 1, 1, 1, 1, 1,
157: C & 1, -2, -2, 2, 1,
158: C & 1, 1, 1, 1, 1,
159: C & 3, -2, -2, 2, -2,
160: C & 1, 2, 2, 2, -2, -2,
161: C & -2, -2, 1, 2,
162: C & 1, 1, 1 /
163: C
164: C .... STATE TABLE ....
165: C
166: C DATA ICALC /
167: C Z 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 0
168: C 1 1, 4, 1, 5, 1, 1, 1, 1, 1, 7, 1, 1
169: C 2 1, 5, 1, 3, 1, 1, 1, 1, 1, 6, 1, 1
170: C 3 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
171: C 4 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 5, 0
172: C 5 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 5, 0
173: C 6 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 5, 0
174: C 7 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 5, 0
175: C 8 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 5, 0
176: C 9 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
177: C A 1, 8, 1, 5, 1, 1, 1, 1, 1, 8, 1, 1
178: C B 1, 8, 1, 8, 8, 8, 8, 8, 8, 8, 0, 0
179: C X /
180: C
181: C
182: C F END ( ) + - * / ** , = -
183: C
184: C 0 F 2 2 1 2 2 2 2 2 2 2 1 0
185: C 1 END 1 4 1 5 1 1 1 1 1 7 1 1
186: C 2 ( 1 5 1 3 1 1 1 1 1 6 1 1
187: C 3 ) 0 0 0 0 0 0 0 0 0 0 0 0
188: C 4 + 1 2 1 2 2 2 2 1 1 1 2 5 0
189: C 5 - 1 2 1 2 2 2 2 1 1 1 2 5 0
190: C 6 * 1 2 1 2 2 2 2 2 2 1 2 5 0
191: C 7 / 1 2 1 2 2 2 2 2 2 1 2 5 0
192: C 8 ** 1 2 1 2 2 2 2 2 2 2 2 5 0
193: C 9 , 0 5 0 0 0 0 0 0 0 0 0 0
194: C 10 = 1 8 1 5 1 1 1 1 1 8 1 1
195: C 11 - 1 8 1 8 8 8 8 8 8 8 0 0

```

```

196: C
197: C
198: C
199: C ... operation mode flags ( 1/0 = ON/OFF )
200: C
201: C &&SILENT OFF
202: Cccc DATA JSILNT /0/
203: C &&REDEFINE ON
204: Cccc DATA JREDEF /1/
205: C &&ECHO ON
206: Cccc DATA JECHO , IECHK /0, 0/
207: C &&CHECK OFF
208: Cccc DATA JCHECK /0/
209: C
210: C
211: C
212: C ..... statement function to translate pointer to single precision
213: C array 'A' to that of double precision array 'D8'
214: C
215: C LDBLE(L) = L/MWORD + MWORD - 1
216: C
217: C===== VECTOR CALCULATION STARTS HERE =====
218: C
219: C ... initialize echoback message file ....
220: C
221: C IF( IECHK.EQ.0 .and. JECHO.eq.1 ) THEN
222: C IECHK = 1
223: C WRITE( IECHO, (('1',A/A)) )
224: C & ' === MONITOR OUTPUT OF <DENTAK> === '
225: C & ' (CALCULATOR UNIT / VERSION 3.1 JUN 1993 ) '
226: Cccc & ' (CALCULATOR UNIT / VERSION 3.0 FEB 1993 ) '
227: Cccc & ' (CALCULATOR UNIT / VERSION 2.0 AUG 1991 ) '
228: C ENDIF
229: C
230: C IST = 1
231: C IEND = LEN(LINE)
232: C IERR = 0
233: C
234: C ..... REMOVE BLANK & < > AND STORE RESULT IN 'LIN' .....
235: C
236: C NL = 0
237: C do 110 I = IST, IEND
238: C if ( INDEX('<>',LINE(I:I)).eq.0 ) then
239: C
240: C NL = NL + 1
241: C
242: C if ( NL.gt.LEN(LIN) ) then
243: C write(IPR,*) 'XXX(VECTOR-CALCULATOR) TOO LONG INPUT!!',
244: C & ' ( MAX=' , LEN(LIN), ' NON-BLANK CHARACTERS) XXX'
245: C IERR = 5
246: C do 100 JJ = 1, LEN(LINE), 72
247: C KK = MIN(JJ+71,LEN(LINE))
248: C write(IPR,('1x',' LINE:',A)) LINE(JJ:KK)
249: C 100 continue
250: C return
251: C end if
252: C
253: C LIN(NL:NL) = LINE(I:I)
254: C IPLIN(NL) = I
255: C end if
256: C 110 continue
257: C
258: C ... operation mode commands ....
259: C
260: C if ( LIN(1:2).eq.'&&' ) then

```


src/shared/vdentk.f

```

261:      if ( INDEX(LIN,'&&SILENT').eq.1 ) then
262:        K      = LEN('&&SILENT')
263:        if ( LIN(K+1:K+2).eq.'ON' ) JSILNT = 1
264:        if ( LIN(K+1:K+3).eq.'OFF' ) JSILNT = 0
265:      else if ( INDEX(LIN,'&&REDEFINE').eq.1 ) then
266:        K      = LEN('&&REDEFINE')
267:        if ( LIN(K+1:K+2).eq.'ON' ) JREDEF = 1
268:        if ( LIN(K+1:K+3).eq.'OFF' ) JREDEF = 0
269:      else if ( INDEX(LIN,'&&ECHO').eq.1 ) then
270:        K      = LEN('&&ECHO')
271:        if ( LIN(K+1:K+2).eq.'ON' ) JECHO = 1
272:        if ( LIN(K+1:K+3).eq.'OFF' ) JECHO = 0
273:      else
274:        write(IPR,*) 'XXX(CALCULATOR) invalid mode command :',
275:        &      LIN(:NL)
276:        IERR = 1
277:      end if
278:      return
279:    end if
280:  C
281:  .... vector data stack pointer ...
282:  C
283:  ISTN      = 0
284:  C
285:  .... function number stack pointer ...
286:  C
287:  ISTF      = 0
288:  C
289:  .... operator number stack pointer ...
290:  C
291:  (DUMMY OPERATOR '1' ( END ) is pushed )
292:  C
293:  ISTO      = 1
294:  ISTK(ISTO) = KEND
295:  C
296:  .... stack pointer of lefthand side variables ...
297:  C
298:  ISTLH     = 0
299:  C
300:  .... MEMORIZE MAXIMUM VALUE-STACK DEPTH & PARAMETER REFERENCE
301:  C
302:  ISTKM     = 0
303:  IPFL      = 0
304:  C
305:  .... number of substitution ...
306:  C
307:  ISUBN     = 0
308:  .... number of individual statements ...
309:  C
310:  ISTMT     = 1
311:  .... number of vector variable reference ...
312:  C
313:  IVECN     = 0
314:  C
315:  ... IS = current position in translation ...
316:  C
317:  IS        = 1
318:  C
319:  ... the following label is common returning point ...
320:  C
321:  1000 continue
322:  C
323:  if ( IS.gt.NL ) then
324:    K      = INDEX(OPERTR,LIN(NL:NL)) + 1
325:    if ( K.gt.1.and.K.ne.KRBRC ) then
326:      ... last character is operator and not ')' ...

```

```

326:  C
327:  if ( JSILNT.eq.0 ) write(IPR,*)
328:  &      'XXX(VECTOR-CALCULATOR) SYNTAX ERROR:'
329:  go to 9000
330:  end if
331:  K      = KEND
332:  go to 230
333:  end if
334:  C
335:  ..... FIND NEXT OPERATOR .....
336:  C
337:  IE : end position of operand in LIN
338:  IEO : end position of operator in LIN
339:  C
340:  K : operator number
341:  C
342:  parameter ( kfunc = 0 )
343:  parameter ( kend = 1 )
344:  parameter ( klbrc = 2 )
345:  parameter ( krbrc = 3 )
346:  parameter ( kplus = 4 )
347:  parameter ( kminus = 5 )
348:  parameter ( kmult = 6 )
349:  parameter ( kdiv = 7 )
350:  parameter ( kpow = 8 )
351:  parameter ( kcomma = 9 )
352:  parameter ( kequal = 10 )
353:  parameter ( kunmns = 11 )
354:  C
355:  0 : single/multi argument function
356:  C
357:  1 : END of input or stack empty
358:  2 : (
359:  3 : )
360:  4 : +
361:  5 : -
362:  6 : *
363:  7 : /
364:  8 : ** or # (power)
365:  9 : ,
366:  10 : =
367:  11 : - (unary minus)
368:  C
369:  JN      = INDEX(NUM,LIN(IS:IS))
370:  C
371:  do 130 I = IS, NL
372:    K      = INDEX(OPERTR,LIN(I:I)) + 1
373:    IEO     = I
374:  C
375:    if ( K.gt.1 ) then
376:  C
377:    .... +- in 3.0E-5 0.9D+12 etc. ....
378:  C
379:    if ( JN.gt.0.and.INDEX('+-',LIN(I:I)).ne.0.and.I.gt.IS
380:  &      .and.INDEX('ED',LIN(I-1:I-1)).ne.0 ) go to 130
381:  &      .and.(INDEX('ED',LIN(I-1:I-1)).ne.0
382:  &      .or.INDEX('ed',LIN(I-1:I-1)).ne.0 ) ) go to 130
383:  C
384:  .... '***' : power operator
385:  C
386:  if ( I.lt.NL.and.LIN(I:I+1).eq.'***' ) then
387:    K      = KPOW
388:    IEO     = IEO + 1
389:  end if
390:  go to 140

```

src/shared/vdentk.f

```

391:         end if
392: 130 continue
393: C
394: C .... end of input ....
395: C
396: C      K      = KEND
397: C
398: C
399: C
400: 140 IE      = I - 1
401: C
402: C
403: C ..... GET VALUES AND PUSH TO STACK .....
404: C
405: C
406: C      if ( INDEX(NUM,LIN(IS:IS)).ne.0 ) then
407: C
408: C          if ( K.eq.KEQUAL ) then
409: C              if ( JSILNT.eq.0 ) write(IPR,*)
410: C              & 'XXX(CALCULATOR) NUMERIC DATA IS ON THE',
411: C              & ' LEFTHAND-SIDE OF "=" !!! '
412: C              IERR = 1
413: C              go to 9000
414: C          end if
415: C
416: C          if ( K.eq.KLBRC ) then
417: C              if ( JSILNT.eq.0 ) write(IPR,*)
418: C              & 'XXX(CALCULATOR) NUMERIC DATA IS ON THE',
419: C              & ' LEFT OF "(" !!! '
420: C              IERR = 1
421: C              go to 9000
422: C          end if
423: C
424: C
425: C      ISTN      = ISTN + 1
426: C      ISTKM      = MAX(ISTKM,ISTN)
427: C
428: C ..... STACK OVERFLOW .....
429: C
430: C      if ( ISTN.gt.NVSTK ) go to 890
431: C
432: C ..... GET VALUE .....
433: C
434: C      IDGT      = IE - IS + 1
435: C      if ( IDGT.ge.10 ) then
436: C          write(FORMT,('(D'',I2,''.0)')) IDGT
437: C      else
438: C          write(FORMT,('(D'',I1,''.0)')) IDGT
439: C      end if
440: C
441: C      read(LIN(IS:IE),fmt =FORMT,iostat =IOSS) TVALUE
442: C      if ( IOSS.ne.0 ) then
443: C          IERR = 1
444: C          if ( JSILNT.eq.0 ) write(IPR,*)
445: C          & 'XXX(VECTOR-CALCULATOR) INVALID NUMERIC CONSTANT (',
446: C          & LIN(IS:IE), ') XXX '
447: C          go to 9000
448: C      end if
449: C
450: C ..... data type of pushed variable
451: C
452: C      1 : integer
453: C      0 : may be non-integer
454: C
455: C      This flag is to speedup power operation if exponent data is

```

```

456: C      an integer constant.
457: C
458: C      IVTYP(ISTN) = 0
459: C      if ( ABS(TVALUE).le.2**30.and.NINT(TVALUE).eq.TVALUE )
460: C          & IVTYP(ISTN) = 1
461: C      *VOCL LOOP,NOVREC
462: C      do 150 I = IELM1 - 1, IELM2 - 1
463: C          D8(LVSTK(ISTN)+I) = TVALUE
464: C      150 continue
465: C
466: C ..... FUNCTION OR NAMED-PARAMETER ??? .....
467: C
468: C
469: C      else if ( IE.ge.IS ) then
470: C
471: C          ..... FUNCTION .....
472: C
473: C          if ( IE.lt.NL.and.LIN(IE+1:IE+1).eq.'(' ) then
474: C
475: C              do 160 IFF = 1, NFUNC
476: C                  if ( LIN(IS:IE).eq.FUNC(IFF) ) go to 170
477: C              160 continue
478: C
479: C                  if ( JSILNT.eq.0 ) write(IPR,*)
480: C                  & 'XXX(CALCULATOR) UNSUPPORTED FUNCTION ', ' (',
481: C                  & LIN(IS:IE), ') XXX '
482: C                  IERR = 1
483: C                  go to 9000
484: C
485: C              ..... push function number as operand (obsolete)....
486: C
487: C              170 continue
488: C
489: C              .... reset function argument counter ...
490: C
491: C              ISTF = ISTF + 1
492: C              NARG(ISTF) = 1
493: C
494: C              ... set -<function number> as operator.
495: C
496: C              K = -IFF
497: C
498: C              ..... substitution .....
499: C
500: C
501: C      else if ( K.eq.KEQUAL ) then
502: C
503: C          ISUBN = ISUBN + 1
504: C
505: C          LP = IE - IS + 1
506: C
507: C          if ( LP.gt.LEN(VVAR(1)) ) then
508: C              if ( JSILNT.eq.0 ) write(IPR,*)
509: C              & 'XXX(VECTOR-CALCULATOR) TOO LONG', ' VECTOR VARIABLE',
510: C              & ' NAME <', LIN(IS:IE), '> (MAX=', LEN(VVAR(1)), ') '
511: C              IERR = 1
512: C              go to 9000
513: C          end if
514: C
515: C          do 180 IP = 1, NVVAR
516: C              if ( LIN(IS:IS+LP-1).eq.VVAR(IP) ) go to 190
517: C          180 continue
518: C
519: C          ..... undefined vector variable ....
520: C

```

src/shared/vdentk.f

```

521:         if ( JSILNT.eq.0 ) write(IPR,*) 'XXX(VECTOR-CALCULATOR) ',
522:         &         'UNDEFINED VECTOR VARIABLE!! <', LIN(IS:IS+LP-1), '> '
523:         IERR      = 1
524:         go to 9000
525: C
526: C         .... push variable # .....
527: C
528: 190         ISTLH   = ISTLH + 1
529:         if ( ISTLH.gt.MS ) go to 890
530:         LHSTK(ISTLH) = IP
531: C
532:         else
533: C
534: C         ..... VARIABLE REFERENCE .....
535: C
536:         JNONP      = 0
537: C
538:         do 200 IP = 1, NVVAR
539:         if ( LIN(IS:IE).eq.VVAR(IP) ) then
540:         go to 210
541:         end if
542: 200         continue
543: C
544: C         .... UNDEFINED VARIABLE ....
545: C
546:         if ( JSILNT.eq.0 ) write(IPR,*)
547:         &         'XXX(VECTOR-CALCULATOR) VARIABLE <', LIN(IS:IE),
548:         &         '> IS NOT DEFINED.'
549:         IERR      = 3
550:         go to 9000
551: C
552: C         .... PUSH VECTOR VARIABLE VALUE TO VECTOR STACK ....
553: C
554: 210         ISTN     = ISTN + 1
555:         ISTKM      = MAX(ISTKM,ISTN)
556: C
557:         if ( ISTN.gt.NVSTK ) go to 890
558: Ccccccc      STK(ISTN) = TVALUE
559:         IVTYP(ISTN) = 0
560: *VOCL LOOP,NOVREC
561:         do 220 I = IELM1 - 1, IELM2 - 1
562:         D8(LVSTK(ISTN)+I) = D8(LVVAR(IP)+I)
563: 220         continue
564:         end if
565: C
566: C
567: C         .... no operand before operator .....
568: C
569: C
570:         else if ( IE.lt.IS ) then
571: C
572: C         < unary operator ? >
573: C
574: C         ..... + OR - IMMEDIATELY AFTER OPERATER (,=
575: C
576:         if ( K.eq.KPLUS .or. K.eq.KMINUS ) then
577:         if ( IS.eq.1
578:         &         .or. (IS.gt.1.and.INDEX('(',',',LIN(IS-1:IS-1)).ne.0) )
579:         &         then
580: C
581: C         .... unary plus means doing nothing.
582: C
583:         if ( K.eq.KPLUS ) then
584:         IS      = IS + 1
585:         go to 1000

```

```

586:         else
587:         K      = KUNMNS
588:         end if
589:         else if ( LIN(IS-1:IS-1).ne.'') then
590:         if ( JSILNT.eq.0 ) write(IPR,*)
591:         &         'XXX(VECTOR-CALCULATOR) SYNTAX ERROR: ',
592:         &         ' INVALID USE OF "+" OR "-'.
593:         IERR      = 1
594:         go to 9000
595:         end if
596:         end if
597: C
598: C         ..... syntax error other than ), & END (
599: C
600:         if ( K.eq.KCOMMA
601:         &         .and.(IE.le.0.or.(IE.gt.0.and.LIN(IE:IE).ne.'')) ) then
602: C
603:         if ( JSILNT.eq.0 ) write(IPR,*)
604:         &         'XXX(VECTOR-CALCULATOR) SYNTAX ERROR: ',
605:         &         'INVALID USE OF COMMA.'
606: C
607:         IERR      = 1
608:         go to 9000
609:         end if
610:         end if
611: C
612: C
613: C         ..... CALCULATION .....
614: C
615: C
616: C         A / K
617: C         0 1 2 3 4 5 6 7 8 9 10 11
618: C
619: C         F END ( ) + - * / ** , = -
620: C
621: C 0 F 2 2 1 2 2 2 2 2 2 2 1 0
622: C 1 END 1 4 1 5 1 1 1 1 1 7 1 1
623: C 2 ( 1 5 1 3 1 1 1 1 1 6 1 1
624: C 3 ) 0 0 0 0 0 0 0 0 0 0 0 0
625: C 4 + 1 2 1 2 2 2 1 1 1 2 5 0
626: C 5 - 1 2 1 2 2 2 1 1 1 2 5 0
627: C 6 * 1 2 1 2 2 2 2 2 1 2 5 0
628: C 7 / 1 2 1 2 2 2 2 2 1 2 5 0
629: C 8 ** 1 2 1 2 2 2 2 2 2 2 5 0
630: C 9 , 0 5 0 0 0 0 0 0 0 0 0 0
631: C 10 = 1 8 1 5 1 1 1 1 1 8 1 1
632: C 11 - 1 8 1 8 8 8 8 8 8 8 0 0
633: C
634: C         A ... TOP OF OPERATION STACK <ISTK>
635: C         K ... NEW OPERATOR
636: C
637: C         < OPERATIONS >
638: C
639: C         0 : non-existent (A,K) pair
640: C         1 : PUSH K TO OPERATION STACK <ISTK>
641: C         2 : * OPERATION BY A.
642: C             * POP OPERATION STACK <ISTK> (discard)
643: C             * CONTINUE (A,K) OPERATION
644: C         3 : POP OPERATION STACK <ISTK>
645: C         4 : END OF CALCULATION. RETURN TOP OF VALUE STACK <STK>
646: C         5 : ERROR
647: C         6 : increment function-argument-counter
648: C         7 : discard K & top of <STK> return to input
649: C         8 : like 2. but operator is unary.
650: C

```

src/shared/vdentk.f

```

651: 230 KK      = K
652:      if ( KK.lt.0 ) K      = KFUNC
653:      KA      = ISTK(ISTO)
654:      if ( KA.lt.0 ) KA      = KFUNC
655: C
656: C
657: C
658:      go to(111,222,333,444,555,666,777,888) ICALC(K,KA)
659: C
660: C
661: C
662:      write(IPR,*) 'XXX(VECTOR-CALCULATOR) OPERATION ERROR',
663:      &      '(MAY BE A BUG) PLEASE CONTACT AUTHORS!!! '
664:      stop 666
665: C
666: C      .... PUSH .....
667: C
668: 111 ISTO      = ISTO + 1
669:      if ( ISTO.gt.MS ) go to 890
670:      ISTK(ISTO) = KK
671:      if ( K.eq.KFUNC ) then
672:          K      = KLBRC
673:          go to 230
674:      end if
675:      IS      = IEO + 1
676:      go to 1000
677: C
678: C      .... OPERATION BY AN OPERATOR AT THE TOP OF STACK ....
679: C
680: 222 IOP      = ISTK(ISTO)
681:      if ( IOP.lt.0 ) IOP = KFUNC
682: C
683: C      two operands
684: C
685:      if ( IOP.ne.KFUNC ) then
686:          if ( ISTN.le.1 ) then
687:              if ( JSILNT.eq.0 ) write(IPR,*)
688:              &      'XXX(VECTOR-CALCULATOR) SYNTAX ERORR'
689:              IERR      = 1
690:              go to 9000
691:          end if
692: C
693:          if ( IOP.eq.KPLUS ) then
694: *VOCL LOOP,NOVREC
695:              do 260 I = IELM1 - 1, IELM2 - 1
696:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I)
697:                  &      + D8(LVSTK(ISTN)+I)
698:              continue
699:          else if ( IOP.eq.KMINUS ) then
700: *VOCL LOOP,NOVREC
701:              do 270 I = IELM1 - 1, IELM2 - 1
702:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I)
703:                  &      - D8(LVSTK(ISTN)+I)
704:              continue
705:          else if ( IOP.eq.KMULT ) then
706: *VOCL LOOP,NOVREC
707:              do 280 I = IELM1 - 1, IELM2 - 1
708:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I)*
709:                  &      D8(LVSTK(ISTN)+I)
710:              continue
711:          else if ( IOP.eq.KDIV ) then
712: *VOCL LOOP,NOVREC
713:              do 290 I = IELM1 - 1, IELM2 - 1
714:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I) /
715:                  &      D8(LVSTK(ISTN)+I)

```

```

716: 290      continue
717:      else if ( IOP.eq.KPOW ) then
718:          IF( DA .LT.0.0 ) THEN
719: C              IF( DB.NE.0.0 .AND. ABS(INT(DB) - DB)/DB .LT.1.0D-5)
720: C              &      THEN
721: C                  DA = (1-2*MOD(INT(DB),2)) *( ABS(DA)**INT(DB))
722: C              ELSE IF( DB.EQ.0.0 ) THEN
723: C                  DA = 1.0
724: C              ELSE
725: C                  if(JSILNT.eq.0)
726: C                  &      WRITE(IPR,*)'XXX(CALCULATOR) UNPERMITTED POWER',
727: C                  &      ' FOR NEGATIVE NUMBER : (',DA,',')**(',DB,',') '
728: C                  IERR      = 2
729: C                  RETURN
730: C              ENDIF
731: C              ELSE
732: C                  IF( abs(DB).le.2**30.and.INT(DB).eq.DB ) THEN
733: C                      DA = DA**(INT(DB))
734: C                  ELSE
735: C                      DA      = DA**DB
736: C                  ENDIF
737: C              ENDIF
738: C
739:          if ( IVTYP(ISTN).eq.1 ) then
740: C              ... exponent is integer constant ...
741:              KE      = NINT(D8(LVSTK(ISTN)+IELM1-1))
742: *VOCL LOOP,NOVREC
743:              do 300 I = IELM1 - 1, IELM2 - 1
744:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I)**KE
745:              continue
746:          else
747: *VOCL LOOP,NOVREC
748:              do 310 I = IELM1 - 1, IELM2 - 1
749:                  D8(LVSTK(ISTN-1)+I) = D8(LVSTK(ISTN-1)+I)**
750:                  &      D8(LVSTK(ISTN)+I)
751:              continue
752:          end if
753: C          end if
754: C
755: C      .... POP NUMERICAL STACK ....
756: C
757: C
758:      ISTN      = ISTN - 1
759: CM2015 IVTYP(ISTN-1) = 0
760:      IVTYP(ISTN) = 0
761: C
762: C      .... POP OPERATOR STACK .....
763: C
764:      ISTO      = ISTO - 1
765: C
766: C      .... FUNCTION .....
767: C
768:      else
769:          if ( ISTF.le.0 ) go to 230
770:          KF      = -ISTK(ISTO)
771: C
772:          if ( NFARG(KF).gt.0 ) then
773: C
774:              if ( NARG(ISTF).ne.NFARG(KF) ) then
775:                  if ( JSILNT.eq.0 ) write(IPR,*)
776:                  &      'XXX(VECTOR-CALCULATOR) FUNCTION ', FUNC(KF),
777:                  &      ' REQUIRES ', NFARG(KF), ' ARGUMENTS BUT GIVEN ',
778:                  &      NARG(ISTF)
779:                  IERR      = 1
780:                  go to 9000

```

src/shared/vdentk.f

```

781:         end if
782:     else if ( NFARG(KF).lt.0 ) then
783:         if ( NARG(ISTF).lt.ABS(NFARG(KF)) ) then
784:             if ( JSILNT.eq.0 ) write(IPR,*)
785:             & 'XXX(VECTOR-CALCULATOR) FUNCTION ', FUNC(KF),
786:             & ' REQUIRES AT LEAST ', ABS(NFARG(KF)),
787:             & ' ARGUMENTS BUT GIVEN ONLY ', NARG(ISTF)
788:             IERR = 1
789:             go to 9000
790:         end if
791:     end if
792: C ..... exp .....
793:     if ( KF.eq.1 ) then
794: *VOCL LOOP,NOVREC
795:         do 320 I = IELM1 - 1, IELM2 - 1
796:             D8(LVSTK(ISTN)+I) = EXP(D8(LVSTK(ISTN)+I))
797:         320 continue
798: C ..... log .....
799:     else if ( KF.eq.2 ) then
800: Ccccccccccccc IF(STK(ISTN).LE.0.0) THEN
801: C         if(JSILNT.eq.0)
802: C & WRITE(IPR,*) 'XXX(CALCULATOR) NON-POSITIVE ARGUMENT',
803: C & ' FOR LOG ',STK(ISTN)
804: C         IERR = 2
805: C         GOTO 6666
806: Ccccccccccccc ENDIF
807: *VOCL LOOP,NOVREC
808:         do 330 I = IELM1 - 1, IELM2 - 1
809:             D8(LVSTK(ISTN)+I) = LOG(D8(LVSTK(ISTN)+I))
810:         330 continue
811: C ..... int .....
812:     else if ( KF.eq.3 ) then
813: *VOCL LOOP,NOVREC
814:         do 340 I = IELM1 - 1, IELM2 - 1
815:             D8(LVSTK(ISTN)+I) = INT(D8(LVSTK(ISTN)+I))
816:         340 continue
817: C ..... sin .....
818:     else if ( KF.eq.4 ) then
819: *VOCL LOOP,NOVREC
820:         do 350 I = IELM1 - 1, IELM2 - 1
821:             D8(LVSTK(ISTN)+I) = SIN(D8(LVSTK(ISTN)+I))
822:         350 continue
823: C ..... cos .....
824:     else if ( KF.eq.5 ) then
825: *VOCL LOOP,NOVREC
826:         do 360 I = IELM1 - 1, IELM2 - 1
827:             D8(LVSTK(ISTN)+I) = COS(D8(LVSTK(ISTN)+I))
828:         360 continue
829: C ..... tan .....
830:     else if ( KF.eq.6 ) then
831: *VOCL LOOP,NOVREC
832:         do 370 I = IELM1 - 1, IELM2 - 1
833:             D8(LVSTK(ISTN)+I) = TAN(D8(LVSTK(ISTN)+I))
834:         370 continue
835: C ..... asin/acos .....
836:     else if ( KF.eq.7 .or. KF.eq.8 ) then
837: C         IF(ABS(STK(ISTN)).GT.1.0) THEN
838: C             if(JSILNT.eq.0)
839: C & WRITE(IPR,*) 'XXX(CALCULATOR) XXX ARGUMENT INVALID',
840: C & ' IN ARCSIN/ARCCOS ',STK(ISTN)
841: C             IERR = 2
842: C             GOTO 6666
843: C         ENDIF
844:         if ( KF.eq.7 ) then
845: *VOCL LOOP,NOVREC

```

```

846:         do 380 I = IELM1 - 1, IELM2 - 1
847:             D8(LVSTK(ISTN)+I) = ASIN(D8(LVSTK(ISTN)+I))
848:         380 continue
849:     end if
850:     if ( KF.eq.8 ) then
851: *VOCL LOOP,NOVREC
852:         do 390 I = IELM1 - 1, IELM2 - 1
853:             D8(LVSTK(ISTN)+I) = ACOS(D8(LVSTK(ISTN)+I))
854:         390 continue
855:     end if
856: C ..... atan .....
857:     else if ( KF.eq.9 ) then
858: *VOCL LOOP,NOVREC
859:         do 400 I = IELM1 - 1, IELM2 - 1
860:             D8(LVSTK(ISTN)+I) = ATAN(D8(LVSTK(ISTN)+I))
861:         400 continue
862: C ..... sqrt .....
863:     else if ( KF.eq.10 ) then
864: C         IF(STK(ISTN).LT.0.0) THEN
865: C             if(JSILNT.eq.0)
866: C & WRITE(IPR,*) 'XXX(CALCULATOR)',
867: C & ' NEGATIVE ARGUMENT IN SQUAREROOT ',STK(ISTN)
868: C             IERR = 2
869: C             GOTO 6666
870: Ccccc ENDIF
871: *VOCL LOOP,NOVREC
872:         do 410 I = IELM1 - 1, IELM2 - 1
873:             D8(LVSTK(ISTN)+I) = SQRT(D8(LVSTK(ISTN)+I))
874:         410 continue
875: C ..... abs .....
876:     else if ( KF.eq.11 ) then
877: *VOCL LOOP,NOVREC
878:         do 420 I = IELM1 - 1, IELM2 - 1
879:             D8(LVSTK(ISTN)+I) = ABS(D8(LVSTK(ISTN)+I))
880:         420 continue
881: C ..... max .....
882:     else if ( KF.eq.12 ) then
883:         LN0 = LVSTK(ISTN-NARG(ISTF)+1)
884:         do 440 IN = ISTN - NARG(ISTF) + 2, ISTN
885: *VOCL LOOP,NOVREC
886:             do 430 I = IELM1 - 1, IELM2 - 1
887:                 D8(LN0+I) = MAX(D8(LN0+I),D8(LVSTK(IN)+I))
888:             430 continue
889:         440 continue
890: C ..... min .....
891:     else if ( KF.eq.13 ) then
892:         LN0 = LVSTK(ISTN-NARG(ISTF)+1)
893:         do 460 IN = ISTN - NARG(ISTF) + 2, ISTN
894: *VOCL LOOP,NOVREC
895:             do 450 I = IELM1 - 1, IELM2 - 1
896:                 D8(LN0+I) = MIN(D8(LN0+I),D8(LVSTK(IN)+I))
897:             450 continue
898:         460 continue
899: C ..... mod .....
900:     else if ( KF.eq.14 ) then
901:         LN0 = LVSTK(ISTN-1)
902: *VOCL LOOP,NOVREC
903:         do 470 I = IELM1 - 1, IELM2 - 1
904:             D8(LN0+I) = MOD(D8(LN0+I),D8(LVSTK(ISTN)+I))
905:         470 continue
906: C ..... log10 .....
907:     else if ( KF.eq.15 ) then
908: *VOCL LOOP,NOVREC
909:         do 480 I = IELM1 - 1, IELM2 - 1
910:             D8(LVSTK(ISTN)+I) = LOG10(D8(LVSTK(ISTN)+I))

```

src/shared/vdentk.f

```

911: 480      continue
912: C ..... sind .....
913:      else if ( KF.eq.16 ) then
914: *VOCL LOOP,NOVREC
915:      do 490 I = IELM1 - 1, IELM2 - 1
916:          D8(LVSTK(ISTN)+I) = SIN(DPAI*D8(LVSTK(ISTN)+I))
917: 490      continue
918: C ..... cosd .....
919:      else if ( KF.eq.17 ) then
920: *VOCL LOOP,NOVREC
921:      do 500 I = IELM1 - 1, IELM2 - 1
922:          D8(LVSTK(ISTN)+I) = COS(DPAI*D8(LVSTK(ISTN)+I))
923: 500      continue
924: C ..... tand .....
925:      else if ( KF.eq.18 ) then
926: *VOCL LOOP,NOVREC
927:      do 510 I = IELM1 - 1, IELM2 - 1
928:          D8(LVSTK(ISTN)+I) = TAN(DPAI*D8(LVSTK(ISTN)+I))
929: 510      continue
930: C .....
931: C .... functions for logical operation ...
932: C
933: C &      'IF'  ', 'AND'  ', 'OR'  ', '
934: C &      'NOT' ', 'EQ'  ', 'NE'  ', 'GT'  ', 'GE'  ', '
935: C &      'LT'  ', 'LE'  ', '
936: C ..... if .....
937:      else if ( KF.eq.19 ) then
938:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
939: *VOCL LOOP,NOVREC
940:      do 520 I = IELM1 - 1, IELM2 - 1
941:          if ( D8(LVSTK(ISTN-2)+I).ne.0.0D0 ) then
942:              D8(LN0+I) = D8(LVSTK(ISTN-1)+I)
943:          else
944:              D8(LN0+I) = D8(LVSTK(ISTN)+I)
945:          end if
946:      continue
947: C ..... and .....
948:      else if ( KF.eq.20 ) then
949:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
950: *VOCL LOOP,NOVREC
951:      do 530 I = IELM1 - 1, IELM2 - 1
952:          if ( D8(LN0+I).eq.0.0D0 ) then
953:              D8(LN0+I) = 0
954:          else
955:              D8(LN0+I) = 1
956:          end if
957: 530      continue
958:      do 550 N = ISTN - NARG(ISTF) + 2, ISTN
959: *VOCL LOOP,NOVREC
960:      do 540 I = IELM1 - 1, IELM2 - 1
961:          if ( D8(LVSTK(N)+I).eq.0.0D0 ) D8(LN0+I) = 0.0
962: 540      continue
963: 550      continue
964: C ..... or .....
965:      else if ( KF.eq.21 ) then
966:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
967: *VOCL LOOP,NOVREC
968:      do 560 I = IELM1 - 1, IELM2 - 1
969:          if ( D8(LN0+I).eq.0.0D0 ) then
970:              D8(LN0+I) = 0
971:          else
972:              D8(LN0+I) = 1
973:          end if
974: 560      continue
975:      do 580 N = ISTN - NARG(ISTF) + 2, ISTN

```

```

976: *VOCL LOOP,NOVREC
977:      do 570 I = IELM1 - 1, IELM2 - 1
978:          if ( D8(LVSTK(N)+I).ne.0.0D0 ) D8(LN0+I) = 1.0D0
979: 570      continue
980: 580      continue
981: C ..... not .....
982:      else if ( KF.eq.22 ) then
983:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
984: *VOCL LOOP,NOVREC
985:      do 590 I = IELM1 - 1, IELM2 - 1
986:          if ( D8(LN0+I).eq.0.0D0 ) then
987:              D8(LN0+I) = 1.0D0
988:          else
989:              D8(LN0+I) = 0
990:          end if
991: 590      continue
992: C ..... eq .....
993:      else if ( KF.eq.23 ) then
994:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
995: *VOCL LOOP,NOVREC
996:      do 600 I = IELM1 - 1, IELM2 - 1
997:          if ( D8(LVSTK(ISTN)+I).eq.D8(LN0+I) ) then
998:              D8(LN0+I) = 1.0D0
999:          else
1000:              D8(LN0+I) = 0
1001:          end if
1002: 600      continue
1003: C ..... ne .....
1004:      else if ( KF.eq.24 ) then
1005:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
1006: *VOCL LOOP,NOVREC
1007:      do 610 I = IELM1 - 1, IELM2 - 1
1008:          if ( D8(LVSTK(ISTN)+I).ne.D8(LN0+I) ) then
1009:              D8(LN0+I) = 1.0D0
1010:          else
1011:              D8(LN0+I) = 0
1012:          end if
1013: 610      continue
1014: C ..... gt .....
1015:      else if ( KF.eq.25 ) then
1016:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
1017:          LN1 = LVSTK(ISTN-NARG(ISTF)+2)
1018: *VOCL LOOP,NOVREC
1019:      do 620 I = IELM1 - 1, IELM2 - 1
1020:          if ( D8(LN0+I).le.D8(LN1+I) ) then
1021:              D8(LN0+I) = 0
1022:          else
1023:              D8(LN0+I) = 1
1024:          end if
1025: 620      continue
1026:      do 640 N = ISTN - NARG(ISTF) + 3, ISTN
1027: *VOCL LOOP,NOVREC
1028:      do 630 I = IELM1 - 1, IELM2 - 1
1029:          if ( D8(LVSTK(N-1)+I).le.D8(LVSTK(N)
1030:              & +I) ) D8(LN0+I) = 0.0D0
1031: 630      continue
1032: 640      continue
1033: C ..... ge .....
1034:      else if ( KF.eq.26 ) then
1035:          LN0 = LVSTK(ISTN-NARG(ISTF)+1)
1036:          LN1 = LVSTK(ISTN-NARG(ISTF)+2)
1037: *VOCL LOOP,NOVREC
1038:      do 650 I = IELM1 - 1, IELM2 - 1
1039:          if ( D8(LN0+I).lt.D8(LN1+I) ) then
1040:              D8(LN0+I) = 0

```

src/shared/vdentk.f

```

1041:          else
1042:            D8(LN0+I) = 1
1043:          end if
1044: 650      continue
1045:          do 670 N = ISTN - NARG(ISTF) + 3, ISTN
1046: *VOCL LOOP,NOVREC
1047:            do 660 I = IELM1 - 1, IELM2 - 1
1048:              if ( D8(LVSTK(N-1)+I).lt.D8(LVSTK(N)
1049:                &      +I) ) D8(LN0+I) = 0.0D0
1050:            660      continue
1051:            670      continue
1052: C ..... lt .....
1053:          else if ( KF.eq.27 ) then
1054:            LN0 = LVSTK(ISTN-NARG(ISTF)+1)
1055:            LN1 = LVSTK(ISTN-NARG(ISTF)+2)
1056: *VOCL LOOP,NOVREC
1057:            do 680 I = IELM1 - 1, IELM2 - 1
1058:              if ( D8(LN0+I).ge.D8(LN1+I) ) then
1059:                D8(LN0+I) = 0
1060:              else
1061:                D8(LN0+I) = 1
1062:              end if
1063:            680      continue
1064:            do 700 N = ISTN - NARG(ISTF) + 3, ISTN
1065: *VOCL LOOP,NOVREC
1066:              do 690 I = IELM1 - 1, IELM2 - 1
1067:                if ( D8(LVSTK(N-1)+I).ge.D8(LVSTK(N)
1068:                  &      +I) ) D8(LN0+I) = 0.0D0
1069:              690      continue
1070:              700      continue
1071: C ..... le .....
1072:          else if ( KF.eq.28 ) then
1073:            LN0 = LVSTK(ISTN-NARG(ISTF)+1)
1074:            LN1 = LVSTK(ISTN-NARG(ISTF)+2)
1075: *VOCL LOOP,NOVREC
1076:            do 710 I = IELM1 - 1, IELM2 - 1
1077:              if ( D8(LN0+I).gt.D8(LN1+I) ) then
1078:                D8(LN0+I) = 0
1079:              else
1080:                D8(LN0+I) = 1
1081:              end if
1082:            710      continue
1083:            do 730 N = ISTN - NARG(ISTF) + 3, ISTN
1084: *VOCL LOOP,NOVREC
1085:              do 720 I = IELM1 - 1, IELM2 - 1
1086:                if ( D8(LVSTK(N-1)+I).gt.D8(LVSTK(N)
1087:                  &      +I) ) D8(LN0+I) = 0.0D0
1088:              720      continue
1089:              730      continue
1090: C ..... nint .....
1091:          else if ( KF.eq.29 ) then
1092: *VOCL LOOP,NOVREC
1093:            do 740 I = IELM1 - 1, IELM2 - 1
1094:              D8(LVSTK(ISTN)+I) = NINT(D8(LVSTK(ISTN)+I))
1095:            740      continue
1096: C ..... atan2 .....
1097:          else if ( KF.eq.30 ) then
1098:            LN0 = LVSTK(ISTN-1)
1099: *VOCL LOOP,NOVREC
1100:            do 750 I = IELM1 - 1, IELM2 - 1
1101:              D8(LN0+I) = ATAN2(D8(LN0+I),D8(LVSTK(ISTN)+I))
1102:            750      continue
1103: C ..... sinh .....
1104:          else if ( KF.eq.31 ) then
1105: *VOCL LOOP,NOVREC

```

```

1106:          do 760 I = IELM1 - 1, IELM2 - 1
1107:            D8(LVSTK(ISTN)+I) = SINH(D8(LVSTK(ISTN)+I))
1108:          760      continue
1109: C ..... cosh .....
1110:          else if ( KF.eq.32 ) then
1111: *VOCL LOOP,NOVREC
1112:            do 770 I = IELM1 - 1, IELM2 - 1
1113:              D8(LVSTK(ISTN)+I) = COSH(D8(LVSTK(ISTN)+I))
1114:            770      continue
1115: C ..... tanh .....
1116:          else if ( KF.eq.33 ) then
1117: *VOCL LOOP,NOVREC
1118:            do 780 I = IELM1 - 1, IELM2 - 1
1119:              D8(LVSTK(ISTN)+I) = TANH(D8(LVSTK(ISTN)+I))
1120:            780      continue
1121:          end if
1122: C .....
1123: C .....
1124: 790      ISTN = ISTN - NARG(ISTF) + 1
1125:            IVTYP(ISTN) = 0
1126:            ISTF = ISTF - 1
1127: C .....
1128:            ISTO = ISTO - 1
1129:          end if
1130:          go to 230
1131: C .....
1132: C .....
1133: C ..... POP operator stack .....
1134: C .....
1135: 333 ISTO = ISTO - 1
1136:       IS = IEO + 1
1137:       go to 1000
1138: C .....
1139: C ..... TERMINATE CALCULATION .....
1140: C .....
1141: 444 go to 880
1142: C .....
1143: C ..... SYNTAX ERROR !! .....
1144: C .....
1145: 555 if ( JSILNT.eq.0 ) write(IPR,*) 'XXX(CALCULATOR) SYNTAX ERROR '
1146:       IERR = 1
1147:       go to 9000
1148: C .....
1149: C .....
1150: 666 if ( ISTO.gt.1.and.ISTK(ISTO-1).gt.0 ) then
1151:       if ( JSILNT.eq.0 ) write(IPR,*)
1152:       &      'XXX(CALCULATOR) SYNTAX ERROR: MISUSED COMMA!!'
1153:       IERR = 1
1154:       go to 9000
1155:     else
1156:       NARG(ISTF) = NARG(ISTF) + 1
1157:     end if
1158:     IS = IEO + 1
1159:     go to 1000
1160: C .....
1161: C .....
1162: 777 ISTN = ISTN - 1
1163:       IS = IEO + 1
1164: CM2015ISTMT = ISTMT + 1
1165:       go to 1000
1166: C .....
1167: C .....
1168: C ... unary operators ...
1169: C .....
1170: 888 if ( ISTK(ISTO).eq.KUNMNS ) then

```

```

      LLVV = LVAR(LHS1K(ISTLH))
      LOOP,NOVREC
      do 870 I = IELM1 - 1, IELM2 - 1
         D8(LLVV+I) = D8(LVSTK(ISTN)+I)
      continue
      ISTLH = ISTLH - 1
      ISTO = ISTO - 1
      go to 230
end if

.... END OF PROCESSING

DENTAK = STK(ISTN)
continue

IF( IECHO.eq.1.and. ISTKM.GT.1 .OR. IP
WRITE( IECHO,'('' '','A')' ) LINE(IS

```

```

      (LVSTK(ISTN)+I)
1
1
S
      STKM,GT.1 .OR. IPFL.GT.0 ) THEN
      ' ',A)' ) LINE(IST:IEND)
      '
      ==> ',stk(i),' ',REAL(stk(i))
      '
      FULL
      write(IPR,*)
      STACK IS FULL XXX'
      '
      N OF ERROR ....
      '
      when
      E)
      NE), 1, -1
      ne.' ' ) then
      '
      LINE: <' ',a,' '>'') LINE(:LK)

```

[illegible]


```
src/shared/vdentk.f
```

[illegible][illegible]

src/shared/walias.f

```
1:      subroutine WALIAS( F,      Q,      ALIAS, N1 )
2: C==<MVP/GMVP>=====
3: C  PURPOSE: PREPARATION FOR SAMPLING by WALKER'S ALIAS METHOD
4: C  CALLED IN: VARIOUS ROUTINES
5: C-----
6: C  arguments (i=input,o=output,w=work)
7: C
8: C  io F : Probability density.
9: C      Original values are changed to meaningless ones.
10: C  o Q : branch probability to alias
11: C  o ALIAS : aliases
12: C  i N1 : length of table
13: C=====
14:      real F(N1)
15:      real Q(N1)
16:      integer ALIAS(N1)
17: C
18:      S      = 0.0
19:      do 100 K = 1, N1
20:          S      = S + F(K)
21:      100 continue
22:      do 110 K = 1, N1
23:          F(K)    = F(K) /S
24:      110 continue
25:      FN1      = FLOAT(N1)
26:      do 160 K = 1, N1
27:          FMAX    = -0.5
28:          FMIN    = 2.0
29:          I      = 1
30:          J      = 1
31: C/#IF SYSTEM( SX* )
32: *      do 120 K1 = 1, N1
33: *          if ( F(K1).gt.FMAX ) then
34: *              I      = K1
35: *              FMAX    = F(K1)
36: *          end if
37: *          if ( F(K1).ge.0. ) FMIN = MIN(FMIN,F(K1))
38: *      120 continue
39: *      do 130 K1 = 1, N1
40: *      130 if ( FMIN.eq.F(K1) ) go to 140
41: *      140 J      = K1
42: C/#ELSE
43:      do 150 K1 = 1, N1
44:          if ( F(K1).gt.FMAX ) then
45:              I      = K1
46:              FMAX    = F(K1)
47:          end if
48:          if ( F(K1).ge.0. ) then
49:              if ( F(K1).lt.FMIN ) then
50:                  J      = K1
51:                  FMIN    = F(K1)
52:              end if
53:          end if
54:      150 continue
55: C/#ENDIF
56:      Q(J)      = 1.0 - FN1*F(J)
57:      F(J)      = -1.0
58:      F(I)      = F(I) - Q(J) /FN1
59:      ALIAS(J)   = I
60:      160 continue
61: C
62:      do 170 K = 1, N1
63:          if ( Q(K).lt.0.0 ) Q(K) = 0.0
64:      170 continue
65: C
66:      return
67:      end
```

src/shared/wdbl.f

```
1:      subroutine WDBL
2: C=====
3: C purpose: define constant MWORD which shows size of double precision
4: C      data (real*8) in word of the architecture.
5: C=====
6: Cccc  block data WDBL
7:      include 'INC/_WORDL'
8: C
9: C/IF WORD(64)
10: C ..... 64 BIT MACHINES (CRAY or other pure 64 bit architecture)
11: Cc      data MWORD /1/
12: C/ELSE
13: C ..... 32 BIT MACHINES (VP SX ETC)
14: Cc      data MWORD /2/
15: C/ENDIF
16: C
17:
18: C
19: C/IF WORD(64)
20: C ..... 64 BIT MACHINES (CRAY or other pure 64 bit architecture)
21: *      MWORD = 1
22: C/ELSE
23: C ..... 32 BIT MACHINES (VP SX ETC)
24:      MWORD = 2
25: C/ENDIF
26:      end
```

src/shared/wkaa2z.f

```

1:      subroutine WKAA2Z( PROG, A, H, LIMIT, LIMITL,
2:      &                  JLATT,JHLAT,JTLT, JDEBG,
3:      &                  IOVFL, LWORK2, MAXWK )
4: C=<MVP>=====
5: C PURPOSE : Calculate working-array-pointers and size of memory
6: C           for "underground" level modules such as 'ABS2ZN'.
7: C           Working arrays in this level should not overlap with those of
8: C           FLIONE,SEAONE,LATICE etc.
9: C
10: C CALLED IN : WRKARY
11: C CALLS : LKEEP
12: C-----
13: C arguments (i=input, o=output, w=work)
14: C i PROG : calling program name ('MVP'/'GMVP')
15: C i A(*) : dynamic memory array (task shared)
16: C i H(*) : dynamic memory array (task local)
17: C i LIMIT : size limit of dynamic memory array (task shared)
18: C i LIMITL : size limit of dynamic memory array (task local)
19: C i JLATT,JHLAT,JTLT, JDEBG...: option flags
20: C           (see (mvp/gmvp)/INC/_FLAGS)
21: C o IOVFL : returns non-zero if memory overflowed
22: C i LWORK2 : starting position array index
23: C o MAXWK : memory size necessary for working area
24: C-----
25:      real A(*)
26:      real H(*)
27: C
28: cccc integer ia(*)
29: c
30:      character*(*) PROG
31:      integer JDEBG(*)
32: c
33: CCCCC include 'INC/_ARRAY'
34:      include 'INC/_SIZES'
35:      include 'INC/_PTALY0'
36:      include 'INC/_STALY'
37:      include 'INC/_PTLSP'
38: C
39:      include 'INC/_WKAB2Z'
40: C
41:      include 'INC/_IOUNIT'
42: c
43: C-----
44: c
45:      write(IPR,7000) LWORK2
46: 7000 format(/1X,' == Working area for ABS2ZN module starts at ',
47:      &      I10,'th element',
48:      &      ' of task local data array. =='/)
49: C
50: C ..... WORKING ARRAY OVERFLOW COUNTER ....
51: C
52:      IOVFL = 0
53:      MAXWK = 0
54: C
55: C ... temporary bank ...
56: C
57:      call LSTLST( 'ABS2ZN', LWORK2, LASTP )
58: c
59:      call LKEEP( '/XXX', PAZ(1), NBNK2, 'R8', LASTP, JDEBG )
60:      call LKEEP( '/YYY', PAZ(2), NBNK2, 'R8', LASTP, JDEBG )
61:      call LKEEP( '/ZZZ', PAZ(3), NBNK2, 'R8', LASTP, JDEBG )
62:      call LKEEP( '/AAA', PAZ(4), NBNK2, 'R8', LASTP, JDEBG )
63:      call LKEEP( '/BBB', PAZ(5), NBNK2, 'R8', LASTP, JDEBG )
64:      call LKEEP( '/CCC', PAZ(6), NBNK2, 'R8', LASTP, JDEBG )
65:      call LKEEP( '/ZZZ', PAZ(7), NBNK2, 'I4', LASTP, JDEBG )

```

```

66:      if( JLATT.ne.0 ) then
67:          call LKEEP( '/LEVL', PAZ(8), NBNK2, 'I4', LASTP, JDEBG )
68:          call LKEEP( '/LZZ', PAZ(9), NBNK2*NEST, 'I4', LASTP, JDEBG )
69:          call LKEEP( '/LPOS', PAZ(10),NBNK2*NEST, 'I4', LASTP, JDEBG )
70:          if( JHLAT.ne.0 )
71:      &      call LKEEP( '/LCSR', PAZ(11),NBNK2*NEST, 'I4', LASTP, JDEBG )
72:      end if
73:      call LKEEP( '/IBREG', PAZ(12),NBNK2, 'I4', LASTP, JDEBG )
74:      if( JTLT.ne.0 ) then
75:          call LKEEP( '/IBSPC',PAZ(13),NBNK2*(NEST+1),'I4',LASTP,JDEBG )
76:      end if
77: C
78: C ... temporary particle stack ...
79: C
80:      call LKEEP( '/LSEND', PAZ(14),NBNK2, 'I4', LASTP, JDEBG )
81: C
82:      call LKEEP( '/LSSRC', PAZ(15),NBNK2, 'I4', LASTP, JDEBG )
83:      call LKEEP( '/IZNXT', PAZ(16),NBNK2, 'I4', LASTP, JDEBG )
84:      call LKEEP( '/NNXT', PAZ(17),NZONE+1, 'I4', LASTP, JDEBG )
85: C
86:      if( JLATT.ne.0 ) then
87:          call LKEEP( '/LSLAT', PAZ(18),NBNK2, 'I4', LASTP, JDEBG )
88:          call LKEEP( '/IZLAT', PAZ(19),NBNK2, 'I4', LASTP, JDEBG )
89:          call LKEEP( '/NXL', PAZ(20),NLBZ+1, 'I4', LASTP, JDEBG )
90:      end if
91: C
92: C ... working array passed from ABS2ZN to ACTA2Z ...
93:      call LKEEP( 'X', PAZ(21),NBNK2, 'R8', LASTP, JDEBG )
94:      call LKEEP( 'Y', PAZ(22),NBNK2, 'R8', LASTP, JDEBG )
95:      call LKEEP( 'Z', PAZ(23),NBNK2, 'R8', LASTP, JDEBG )
96:      call LKEEP( 'IFI', PAZ(24),NBNK2, 'I4', LASTP, JDEBG )
97:      call LKEEP( 'IFL', PAZ(25),NBNK2, 'I4', LASTP, JDEBG )
98:      call LKEEP( 'IWK1', PAZ(26),NBNK2, 'I4', LASTP, JDEBG )
99: C
100:      call LKEEP( 'AI', PAZL(1),NBNK2, 'R8', LASTP, JDEBG )
101:      call LKEEP( 'BI', PAZL(2),NBNK2, 'R8', LASTP, JDEBG )
102:      call LKEEP( 'CI', PAZL(3),NBNK2, 'R8', LASTP, JDEBG )
103:      call LKEEP( 'JKSF', PAZL(4),NZONE+1, 'I4', LASTP, JDEBG )
104:      call LKEEP( 'IPZONE', PAZL(5),NZONE+1, 'I4', LASTP, JDEBG )
105: C
106:      if ( LSIZL(LASTP).gt.LIMITL ) IOVFL = IOVFL + 1
107:      MAXWK = MAX(MAXWK,LASTP)
108:      write(IPR,7020) 'ABS2ZN', LASTP - LWORK2, LASTP
109: C
110: 7020 format(/1X,' == SUBROUTINE <',A6,'> NEEDS WORKING AREA OF ',I10,
111:      &      ' WORDS. ==== ( LAST POSITION ',I10,')')
112: C
113:      return
114: end

```

src/shared/wsizof.f

```
1:      function WSIZOF(TYPE)
2:      real*8 WSIZOF
3:      C
4:      C=====
5:      C PURPOSE : RETURN SIZE OF VARIABLE OF 'TYPE' (I4/R4/R8/CXXX)
6:      C           MEASURED BY 'WORD' OF CURRENT HARDWARE.
7:      C=====
8:      C
9:      character*(*) TYPE
10:     character*8 FORMT
11:     include 'INC/_WORDL'
12:     include 'INC/_IUNIT'
13:     C
14:     .... TYPE = 'R8' : DOUBLE PRECISION/REAL = 32/64 BIT MACHIENS
15:     C
16:     if ( TYPE.eq.'R8' ) then
17:     C       WSIZOF = 8/MWORD
18:       WSIZOF = MWORD
19:     C
20:     .... TYPE = 'CXX' : CHARACTER OF XX BYTES .....
21:     C
22:     else if ( TYPE(1:1).eq.'C' ) then
23:       write(FORMT,'(I',I1,',')') LEN(TYPE(2:))
24:       read(TYPE(2:LEN(TYPE)),fmt=FORMT) LC
25:       WSIZOF = DBLE(LC) /(8/MWORD)
26:     C
27:     .... TYPE = 'R4'/'I4' : REAL/INTEGER
28:     C
29:     else if ( TYPE.eq.'R4' .or. TYPE.eq.'I4' ) then
30:       WSIZOF = 1
31:     else
32:       write(IMG,*) 'XXX UNIDENTIFIABLE VARIABLE TYPE <' , TYPE,
33:       & '> PASSED TO "WSIZOF" FUNCTION. !! '
34: Ctemp %%%
35:     C       k = 1
36:     C       k = 1/(k-1)
37:     C       call PRSTOP( 1,
38:     & 'MAYBE BUGS IN PROGRAM. PLEASE CONTACT AUTHORS.' )
39:       stop 888
40:     end if
41:     return
42:     end
```

src/shared/zlbz.f

```
1:      subroutine ZLBZ( IOG,   KZMAT, NZONE, KZLBZ, NLBZ,  MLBZZ )
2: C=====
3: C  PURPOSE: TO PREPARE KZLBZ ARRAY (ZONE # FOR EACH LATTICE-BUFFER ZONE)
4: C           & MLBZZ ARRAY (LATTICE-BUFFER ZONE # FOR EACH ZONE)
5: C  CALLED IN: GEOMIN
6: C
7: C-----
8: C arguments (i=input, o=output, w=work)
9: C
10: C i IOG: message printout I/O unit
11: C i KZMAT(NZONE) : material ID's of zones
12: C i NZONE : number of zones
13: C i NLBZ : number of lattice-buffer-zones
14: C o KZLBZ(NLBZ) : Zone number of each lattice-buffer zone.
15: C o MLBZZ(NZONE) : Lattice-buffer zone #'s of each zone.
16: C=====
17:      integer KZMAT(NZONE), KZLBZ(NLBZ), MLBZZ(NZONE)
18: C
19:      include 'INC/_PMLATT'
20:      include 'INC/_SFLATT'
21: C-----
22: C
23:      NN      = 0
24:      do 100 K = 1, NZONE
25: CCCCCC if ( KZMAT(K).le.-1.and.KZMAT(K).ge.-999 ) then
26:      if ( ISLATT(KZMAT(K)).or.KZMAT(K).eq.-999 ) then
27:          NN      = NN + 1
28:          KZLBZ(NN) = K
29:          MLBZZ(K)  = NN
30:      end if
31: 100 continue
32: C
33:      call LABEL( IOG, 'ZONES TO JOINT SUBFRAME & CELL' )
34: C
35:      write(IOG,7000)
36: 7000 format(8X,'          ZONE #          MATERIAL      '/8X,
37: & '-----' )
38:      do 110 I=1,NN
39:          if ( ISLATT(KZMAT(KZLBZ(I))) ) then
40:              write(IOG,7040) I,KZLBZ(I),LATTNM(KZMAT(KZLBZ(I)))
41:          else
42:              write(IOG,7020) I,KZLBZ(I),KZMAT(KZLBZ(I))
43:          end if
44: 110 continue
45: 7020 format(8X,I4,3X,I6,6X,I8)
46: 7040 format(8X,I4,3X,I6,'      LAT ',I6)
47:      return
48:      end
```

src/shared/zonein.f

```

1:      subroutine ZONEIN( A,      LIMIT, CHA,      LIMITC,JDEBG, JLATT,
2:      &                  JSIMP, JTLLT, JSTGR, JFISX, SDA,      IPSDA, KZDA,
3:      &                  IBSDA, DBODI, IBODI, NBODI, KREG,      KMAT,
4:      &                  LKINPZ,LKZDA, LKZAA, LNKZAA,LKSFBDB,NBODY,
5:      &                  NINPZ, NZONE, NSURF, NSDA,      NZDA,      LKBZL,
6:      &                  LIBZL, NBZL,      LAST,      NWRK,      MAXSF, LINE,
7:      &                  NCELL, IDCEL, ICTYP, IPCEL, IPCLI, KREGN,
8:      &                  NLATT, IDLAT, LTYP,      NVLAT, NLTSPC,NSMSPC,IZNAM
9:      &                  )
10: C=<MVP/GMVP>=====
11: C  PURPOSE: INPUT ZONE DATA & PREPARE KZDA,KZAA,KMEMO ARRAYS.
12: C  CALLED IN: GEOMIN
13: C-----
14: C arguments (i=input, o=output, w=work)
15: C
16: C << only description of arguments output/defined under this routine >>
17: C
18: C io A(*) : dynamic memory array
19: C io CHA(*) : dynamic character memory
20: C o SDA(NSDA) : surface shape data
21: C o IPSDA(3,NSURF+1) : pointer to surface shape data SDA
22: C o KZDA(2,*) : zone surface composition
23: C o KMAT(2,NINPZ) : material ID for input-zones
24: C o LKINPZ,LKZDA, LKZAA, LNKZAA,LKSFBDB : array index pointers
25: C o NZONE : number of zones
26: C o NZDA : length of KZDA array (second dimension)
27: C o LKBZL, LIBZL: array index pointers
28: C o NBZL : Total number of body * sign input used to define zones
29: C o MAXSF: maximum number of necessary surface data to define a zone
30: C o IDCEL(NCELL) : lattice cell ID's
31: C o ICTYP(NCELL) : lattice cell type
32: C o IPCEL(NCELL+1) : zone pointers for each cell
33: C o IPCLI(NCELL+1) : input-zone pointers for each cell
34: C o KREGN(NINPZ) : region ID for input-zones
35: C o NLTSPC : Number of combinations of (frame-name, lattice)
36: C o NSMSPC : number of data given as "subframe" names of each cell
37: C o IZNAM : names of input-zones
38: C=====
39:      integer JDEBG(*)
40:      real*8 SDA(NSDA)
41:      integer IPSDA(3,NSURF+1)
42:      integer IBODI(NBODY+1)
43:      real*8 DBODI(NBODI)
44: C
45:      integer IBSDA(3,NBODY), KZDA(2,NWRK), KREG(NINPZ), KMAT(NINPZ,2),
46:      &      IDCEL(NCELL), ICTYP(NCELL), IPCEL(NCELL+1),
47:      &      IPCLI(NCELL+1), IDLAT(NLATT), LTYP(NLATT), NVLAT(4,NLAT)
48: C
49:      character*12 IZNAM(NINPZ)
50: C
51: CCC include 'INC/_ARRAY'
52:      real A(*)
53:      character*4 CHA(*)
54: CCC
55:      include 'INC/_IOUNIT'
56:      include 'INC/_LNAME'
57: C
58:      character*(LNAME) KREGN(NINPZ)
59: C
60: C ... local data
61: C
62:      integer IWK(10)
63:      character OUTLIN*130, LINE*72
64:      character HEAD1*72, HEAD2*72
65: C

```

```

66: C      .... FOR CELL ID & TYPE ...
67:      character VNAME*8, CTYPE*16
68: C
69:      character*32 EXREG
70: C
71: C      ... max buffer size of bodies per zone
72: C
73:      parameter( MXBPZ      = 1024 )
74:      integer KBBB(MXBPZ)
75: C
76: C      ... includes definition of ISLATT(m) and LATTNM(m)
77: C      and MLTOFF ....
78: C
79:      include 'INC/_PMLATT'
80:      include 'INC/_SFLATT'
81: C
82: C -----
83: C
84: C
85: C
86: C ..... MEANINGS OF IBSDA ARRAY ( SEE GEOMIN )
87: C
88: C*** IBSDA(1,I): POSITION OF HEAD OF I'TH BODY IN SDA ARRAY.
89: C IBSDA(1,I): surface # of the first surface of the body.
90: C IBSDA(2,I): BODY ID OF I'TH BODY.
91: C
92: C
93: C =====
94:      call HEADER( IOG, 'ZONE DATA' )
95: C =====
96: C
97: C
98: C HEAD1= ' NO. ZONE # ZONE NAME : REG. NAME : MAT. : @'
99:      IZNAM(1)      = ' ZONE NAME '
100:      KREGN(1)      = 'REG. NAME '
101:      HEAD1      = ' NO. ZONE # '//IZNAM(1) '//: '//KREGN(1) //'
102:      &      ': MAT. : @'
103:      LHEAD      = INDEX(HEAD1,'@') - 1
104:      HEAD2      = '-----'
105: C
106:      write(IOG,7000) HEAD1(:LHEAD), HEAD2(:LHEAD),
107:      &      ( ' ----',I=1,(LEN(OUTLIN)-LHEAD)/6)
108: C
109:      7000 format(/1X,A,' BODY # & SIGN '/1X,A,15A)
110: C
111:      NZONET      = NZONE
112:      INPZ      = 0
113:      NZONE      = 0
114:      NZDA      = 0
115:      INBODY      = 0
116:      ICZONE      = 0
117:      NCL      = 0
118:      MAXSF      = 0
119: C
120:      JOUT      = 0
121:      KJLAT      = 0
122:      NOR      = 0
123: C
124: C-----
125: C ..... CHANGE THE INPUT I/O UNIT IN FREE-FORM INPUT TO 'IUG1' ...
126: C-----
127:      call RIUNIT( IUG1 )
128: C
129:      call RWIND( IUG1 )
130: C

```

src/shared/zonein.f

```

131: C ..... I/O UNIT 'IUG2' IS USED TO SAVE SUBFRAME NAME SPECIFICATION
132: C      DATA FOR LATTICE REGION
133: C
134: C      NLTSPC : NUMBER OF SUBFRAME SPECIFICATION SET FOUND
135: C      NSMSPC : TOTAL NUMBER OF CELL-SUBFRAME SPECIFICATION
136: C
137: C      if ( JTLT.ne.0 ) then
138: C          call RWIND( IUG2 )
139: C          NLTSPC = 0
140: C          NSMSPC = 0
141: C      end if
142: C
143: C ..... save KZAA & KINPZ in working file IUB
144: C
145: C      call RWIND( IUB )
146: C
147: C      NBZL = 0
148: C      IBZ = 0
149: C
150: C -----
151: C ..... LZDAT: KZDA ARRAY POINTER (TEMPORARY) .....
152: C
153: C ..... READ-IN A LINE ( '*' ON 1ST COLUMN MEANS COMMENT LINE ) .....
154: C
155: C 100 call GETLIN( IUG1, IOG, LINE, NCHAR, IEND )
156: C
157: C      if ( IEND.ne.0 ) go to 260
158: C      if ( NCHAR.eq.0 ) go to 108
159: C      if ( LINE(1:4).eq.'$END' ) go to 260
160: C
161: C -----
162: C ..... SPECIAL LINE HEADED BY '#' (LATTICE GEOMETRY) .....
163: C -----
164: C
165: C      if ( LINE(1:1).eq.'#' ) then
166: C
167: C      CELL SPECIFICATION '#CELL ID(...) TYPE( ..... ) '
168: C
169: C -----
170: C Subframe specifications for TALLY-LATTICE option is active
171: C ("SUBFRAME" is called as "SPACE" in older version.)
172: C
173: C * Sandwiched by '#SUBFRAME' line & '#END SUBFRAME' line
174: C * Data must be described in column 6-72 or having '#' on the first
175: C column.
176: C * Contents assumed:
177: C * Subframe name (arbitrary number of strings up-to 12 characters)
178: C   NAMES( NAME1 NAME2 .... )
179: C * Subframe name #'s in NAMES(...) for each lattice-cell.
180: C   SPACE( ..... ) (NVLAT(1)*NVLAT(2)*NVLAT(3) DATA ASSUMED)
181: C
182: C   When NAMES(...) was not specified numbers in SPACE(...) are
183: C   taken as space names.
184: C -----
185: C
186: C      if ( NCHAR.ge.6.and.LINE(1:6).eq.'#SPACE' .or. NCHAR.ge.9
187: C      & .and.LINE(1:9).eq.'#SUBFRAME' ) then
188: C
189: C
190: C      JOUT = 1 : #SUBFRAME ... #END SUBFRAME specified
191: C      0 : #SUBFRAME ... #END SUBFRAME not specified
192: C      -1 : #SUBFRAME ... #END SUBFRAME specified but not valid
193: C
194: C      JOUT = 1
195: C

```

```

196: C      if ( ISLATT(KMAT(INPZ,1)) ) then
197: C          ILATT = LATTNM(KMAT(INPZ,1))
198: C
199: C      do 110 IL = 1, NLATT
200: C          if ( ILATT.eq.IDLAT(IL) ) go to 120
201: C          110 continue
202: C
203: C          write(IMG,'(1X,A,I6,A,A)') 'XXX LATTICE <', ILATT,
204: C          & '> does not ',
205: C          & 'exist. so your subframe data has no meaning!!'
206: C          call CNTERR( 'FATAL' )
207: C
208: C      ... assign lattice #1 to prevent immediate error stop ...
209: C
210: C          IL = 1
211: C          JOUT = -1
212: C      else
213: C          write(IMG,'(1X,A,A,I6,A)')
214: C          & ' !!! # SUBFRAME data is ignored because',
215: C          & ' input zone <', INPZ,
216: C          & '> is not a lattice FRAME!'
217: C          call CNTERR( 'WARNING' )
218: C          JOUT = -1
219: C          IL = 1
220: C      end if
221: C
222: C 120 continue
223: C
224: C      if ( JOUT.eq.1 ) then
225: C          NSMSPC = NSMSPC + NVLAT(1,IL)*NVLAT(2,IL)*NVLAT(3,IL)
226: C          NLTSPC = NLTSPC + 1
227: C
228: C      .... OUTPUT on unit IUG2: @SUBFRAME KREGN ( LATTICE # ) ...>
229: C
230: C          write(IUG2,'('@SUBFRAME ',A,'('',I6,'')')')
231: C          & KREGN(INPZ), ILATT
232: C
233: C 130 call GETLIN( IUG1, IOG, LINE, NCHAR, IEND )
234: C
235: C      if ( IEND.ne.0 ) then
236: C          write(IMG,'(1X,a,a)')
237: C          & 'XXX Unexpected end of input-data in ',
238: C          & '"#SUBFRAME" block!!!'
239: C          write(IMG,'(1X,a)')
240: C          & 'Please check existence of "#END SUBFRAME" line.'
241: C          call CNTERR( 'FATAL' )
242: C          call PRSTOP( 0, 'SUBFRAME INPUT ERROR.' )
243: C          stop 888
244: C      end if
245: C
246: C      if ( LINE(1:10).ne.'#END SPACE'
247: C      & .and.LINE(1:13).ne.'#END SUBFRAME' ) then
248: C          write(IUG2,'(A)') LINE(:NCHAR)
249: C          go to 130
250: C      else
251: C          write(IUG2,'(A)') '@END '
252: C      end if
253: C
254: C 140 call GETLIN( IUG1, IOG, LINE, NCHAR, IEND )
255: C
256: C      if ( IEND.ne.0 ) then
257: C          write(IMG,'(1X,a,a,1X,a)')
258: C          & 'XXX Unexpected end of input-data in ',
259: C          & '"#SUBFRAME (SPACE)" block!!!',
260: C          & 'Please check existence of "#END SUBFRAME" line.'

```


src/shared/zonein.f

```

261:      call CNTERR( 'FATAL' )
262:      call PRSTOP( 0, 'SUBFRAME INPUT ERROR.' )
263:      stop 888
264:      end if
265:
266:      if ( LINE(1:10).ne.'#END SPACE'
267:      &      .and.LINE(1:13).ne.'#END SUBFRAME' ) go to 140
268:      end if
269: C -----
270: C ..... START NEW CELL (LATTICE GEOMETRY) .....
271: C -----
272: C -----
273: C
274:      else if ( LINE(1:5).eq.'#CELL' .or. LINE(1:2).eq.'# '
275:      &      .or. INDEX('0123456789',LINE(2:2)).ne.0 ) then
276: C
277:      if ( INBODY.gt.0 ) then
278:      write(IOG,'(1X,A)') OUTLIN(:ICLEN2(OUTLIN))
279:      end if
280: C
281: C
282:      INBODY = 0
283:      NCL = NCL + 1
284:      ICZONE = 0
285: C -----
286: C ..... CELL SPECIFICATION ( #CELL ID(CELL-ID) TYPE( CELL-TYPE ) ) .....
287: C -----
288: C -----
289: C
290:      if ( NCHAR.gt.5.and.LINE(1:5).eq.'#CELL' ) then
291: C
292:      call VIFILE( 1, LINE(6:), 1, IERR )
293:      CTYPE = ' '
294:      LNN = 0
295:      IDCEL(NCL) = 0
296: C
297: 150      call FREADS( VNAME, NLEN, IEND )
298:      if ( IEND.eq.0 ) then
299:      if ( VNAME(:NLEN).eq.'ID' ) then
300:      call I4READ( ' ', IDCEL(NCL), NA, 1, IERR )
301:      go to 150
302:      else if ( VNAME(:NLEN).eq.'TYPE' ) then
303: 160      call CHREAD( ' ', CTYPE, ' ', LNN, ITERM, IEND )
304:      if ( IEND.ne.0 ) then
305:      write(IMG,'(1X,A)')
306:      &      'XXX No data for cell "TYPE".'
307:      call PRSTOP( 1, 'UNEXPECTED END OF DATA' )
308:      stop 888
309:      end if
310:      if ( ITERM.eq.0 ) go to 160
311:      end if
312:      end if
313: C
314:      if ( IDCEL(NCL).eq.0 ) then
315:      write(IMG,'(1X,A)')
316:      &      'XXX Cell ID (IDCEL(...)) is not specified.'
317:      call PRSTOP( 1, 'NO IDCEL() DATA FOR A CELL.' )
318:      stop 888
319:      end if
320:      if ( LNN.eq.0 ) then
321:      write(IMG,'(1X,A)') 'XXX Cell type is not specified.'
322:      call PRSTOP( 1, 'NO TYPE() DATA FOR A CELL.' )
323:      stop 888
324:      end if
325:      call CELTYP( CTYPE(1:LNN), '>', ICTYP(NCL) )

```

```

326:      if ( ICTYP(NCL).eq.0 ) then
327:      write(IMG,*) 'XXX Invalid cell type <', CTYPE(1:LNN),
328:      &      '>'
329:      call CNTERR( 'FATAL' )
330:      end if
331: C
332:      call VIPURG( 1 )
333: C
334: C .... OLD-FASHIONED CELL SPECIFICATION ( # CELL-ID CELL-TYPE )
335: C
336:      else
337: C
338:      write(IMG,'(1X,A/1X,A)')
339:      &      '!!! Your cell specification is old fashioned.',
340:      &      ' Please use "#CELL ID(...) TYPE(...)." '
341:      call CNTERR( 'WARNING' )
342: C
343:      LINE(1:1) = '('
344:      LINE(NCHAR:NCHAR) = ')'
345: C
346:      call VIFILE( 1, LINE(:NCHAR), 1, IERR )
347:      call I4READ( ' ', IWK, NA, 2, IERR )
348:      call VIPURG( 1 )
349: C
350:      IDCEL(NCL) = IWK(1)
351:      ICTYP(NCL) = IWK(2)
352:      end if
353: C
354:      IPCLI(NCL) = INPZ + 1
355:      IPCEL(NCL) = NZONE + 1
356: C
357:      write(IOG,'(/8X, '### CELL ',I5, ' '### TYPE ',I3/))'
358:      IDCEL(NCL), ICTYP(NCL)
359: C
360:      else if ( LINE(1:9).eq.'#END CELL' ) then
361:      go to 100
362:      else
363:      &      write(IMG,'(1X,A/2X,A,A)')
364:      &      'XXX Unrecognizable control line ', LINE
365:      call PRSTOP( 1, 'UNRECOGNIZABLE INPUT DATA.' )
366:      stop 888
367:      end if
368:      go to 100
369: C -----
370: C ..... START NEW INPUT-ZONE .....
371: C -----
372: C -----
373: C
374:      else if ( NCHAR.ge.5.and.LINE(1:5).ne.' ' ) then
375:      INPZ = INPZ + 1
376:      ICZONE = ICZONE + 1
377: C
378: C ..... #SUBFRAME MUST BE GIVEN AFTER LATTICE FRAME ZONES IF JTLLT.ne.0
379: C
380:      if ( INPZ.gt.1.and.JTLLT.ne.0 ) then
381:      MAT0 = KMAT(INPZ-1,1)
382:      if ( JOUT.eq.0.and.ISLATT(MAT0) ) then
383:      write(IMG,'(1X,A,A,A/4X,A)')
384:      &      'XXX A lattice-frame zone <',
385:      &      IZNAM(INPZ-1) (:ICLEN(IZNAM(INPZ-1))),
386:      &      '> must have a ',
387:      &      ' "#SUBFRAME ... #END SUBFRAME" block',
388:      &      ' in "FRAME-DEPENDENT-TALLY"/"TALLY-LATTICE" mode.'
389:      call CNTERR( 'FATAL' )
390:      end if

```

src/shared/zonein.f

```

391:         end if
392: C
393:         JOUT      = 0
394: C
395:         NOR       = 0
396: C -----
397: C ..... READ-IN  REGION AND MATERIAL # ( NAME:REG:MAT: ETC.) ...
398: C -----
399:         K         = INDEX(LINE(:NCHAR),':')
400: C
401:         if ( K.ne.0 ) then
402: C
403: C -----
404: C ..... INPUT -ZONE NAME
405: C -----
406: C
407:         IZNAM(INPZ) = ' '
408:         call CCOMP( IZNAM(INPZ), LEN(IZNAM(1)), LINE(:K-1), IFLG )
409: C
410: C -----
411: C .... 'REGION' NAME .....
412: C -----
413: C
414:         JJ        = INDEX(LINE(K+1:NCHAR),':')
415:         II         = JJ + K
416:         if ( II.gt.K+1 ) then
417:             call CCOMP( KREGN(INPZ), LEN(KREGN(INPZ)),
418: &                 LINE(K+1:II-1), IFLG )
419: C
420:             if ( IFLG.eq.1 ) then
421:                 write(IMG, '(//lx,a,a,i4,a//)')
422: &                 ' !!! Too long REGION name <', LINE(I+1:II-1),
423: &                 '>! Truncated to ', LEN(KREGN(INPZ)),
424: &                 ' characters!'
425:                 call CNTERR( 'WARNING' )
426:             end if
427:         else
428: C
429: C ... no ":" which separates region-name & mat # was found ...
430: C
431:             if ( JJ.lt.0 ) then
432:                 write(IMG, '(//lx,a//)')
433: &                 'XXX ":" is not found which separates region-name & mat #.'
434:                 call CNTERR( 'FATAL' )
435:                 II         = K + 1
436:             end if
437: C
438: C ... region-name is blank. ...
439: C
440:             KREGN(INPZ) = ' '
441:         end if
442: C
443: C ... check region name (added Jan 2000)
444: C
445:         KLL        = ICLEN2(KREGN(INPZ))
446:         EXREG       = '@!-+*?<>():;##$/&,"'''
447:         do 170 KII = 1, KLL
448:             KI       = INDEX('@-+*?<>():;##$/&', KREGN(INPZ)(KII:KII))
449:             if ( KI.ne.0 ) then
450:                 write(IMG,7020) INPZ, KREGN(INPZ) (:KLL),
451: &                 EXREG(KI:KI)
452:                 call CNTERR( 'FATAL' )
453:             end if
454:         continue
455: 7020 format(1x,'XXX Region name of ',I5,'th input zone <',A,

```

```

456: &         '< has an invalid character as region-name : <',A,
457: &         '>')
458: C
459: C -----
460: C .... MATERIAL ID # ....
461: C -----
462: C
463:         JJ        = INDEX(LINE(II+1:NCHAR),':')
464:         KK         = INDEX(LINE(II+1:NCHAR),':') + II
465: C
466:         KJLAT      = 0
467: C
468:         if ( JJ.gt.0 ) then
469:             III     = II + 1
470:             call CCOMP( LINE(III:KK-1), KK-III-1, LINE(III:KK-1), IFLG
471: &                 )
472: C
473: C .... "LAT=mat" or "L=mat" is lattice ID
474: C
475:             if ( LINE(III:III+3).eq.'LAT='
476: &                 .or. LINE(III:III+1).eq.'L=' ) then
477:                 KJLAT = 1
478:                 III   = III + INDEX(LINE(III:KK-1),'=')
479:             end if
480: C
481:             call READI4( LINE(:NCHAR), III, KK-1, KMAT(INPZ,1) )
482: C
483:             if ( KJLAT.eq.0.and.-1.ge.KMAT(INPZ,1)
484: &                 .and.KMAT(INPZ,1).ge.-998 ) then
485:                 KJLAT = 2
486:             end if
487: C
488:             ... convert lattice ID to an offsetted "material" ID
489: C
490:             if ( KJLAT.ne.0 ) then
491:                 KMAT(INPZ,1) = MLTOFF - ABS(KMAT(INPZ,1))
492:             end if
493: C
494:             KMAT(INPZ,2) = KMAT(INPZ,1)
495: C
496: C ... lattice frame has no "frame" name when JTLLT > 0 ...
497: C
498:             if ( JTLLT.ne.0.and.KREGN(INPZ).eq.' ' .and.KJLAT.ne.0 )
499: &                 then
500:                 write(IMG, '(//lx,a,a,i6,a//)')
501: &                 'XXX "REGION-NAME" as a frame name must be specified for',
502: &                 ' "frame" zone in TALLY-LATTICE mode. (lattice ID:',
503: &                 LATTNM(KMAT(INPZ,1)), ' )'
504:                 call CNTERR( 'FATAL' )
505:             end if
506: C
507: C ... check if the first input-zone of a cell has
508: C         KMAT(*,1) = -999.
509: C
510:             if ( NCL.gt.0 ) then
511:                 if ( ICZONE.eq.1.and. KMAT(INPZ,1).ne.-999 ) then
512:                     write(IMG, '(//lx,a,a,i6,a,i6,a//)')
513: &                     'XXX Material ID of the first input-zone in a CELL',
514: &                     ' must be -999. CELL:',IDCEL(NCL),' material: ',
515: &                     KMAT(INPZ,1)
516:                     call CNTERR( 'FATAL' )
517:                 else if ( ICZONE.gt.1.and. KMAT(INPZ,1).eq.-999 ) then
518:                     write(IMG, '(//lx,a,i6,a//)')
519: &                     'XXX Material ID of a non-first input-zone in CELL <',
520: &                     IDCEL(NCL),'> is -999.'

```

src/shared/zonein.f

```

521:         call CNTERR( 'FATAL' )
522:       end if
523:     end if
524: C
525: C       ... set KMAT(*,2) to zero (temporary) for STGM region.
526: C
527:       if ( JSTGR.ne.0 ) then
528:         ILATT = LATTNM(KMAT(INPZ,1))
529:         do 180 IL = 1, NLATT
530:           if ( ILATT.eq.IDLAT(IL) ) then
531:             if ( LTYP(IL).eq.10 ) KMAT(INPZ,2) = 0
532:           end if
533: 180      continue
534:       end if
535: C
536:       else
537:         write(IMG,'(1x,a,1x,a)')
538:         &       'XXX ":, which terminates material ID input,',
539:         &       ' is not found ',
540:         &       '(Temporary material # 0 is set)'
541:         call CNTERR( 'FATAL' )
542:       end if
543: C
544:       else
545:         write(IMG,'(1x,a,a)')
546:         &       'XXX Obsolete form in newer version:',
547:         &       ' You cannot input REGION & MATERIAL # separately'
548:         write(IMG,'(1x,a,a)') ' from zone specification. ',
549:         &       'Please input as <ZONE-NAME: REGION-NAME: MAT-ID :...>'
550:         call CNTERR( 'FATAL' )
551:         KK = INDEX(LINE(:NCHAR),':')
552:       end if
553: C
554: C-----
555: C ..... INPUT-ZONE # & ZDA POINTER OF ZONE (TEMPORARY) .....
556: C-----
557: C
558:       if ( NZONE.gt.0 ) then
559:         write(IUB) IBZ
560:         write(IUB) (KBBB(IB),IB=1,IBZ)
561:         NBZL = NBZL + IBZ
562:         IBZ = 0
563:       end if
564:       write(IUB) NZDA + 1, INPZ
565: C
566:       NZONE = NZONE + 1
567: C
568:       if ( INBODY.gt.0 ) write(IOG,'(1X,A)') OUTLIN(:ICLEN2(OUTLIN))
569: C
570:       INBODY = 0
571:       OUTLIN = ' '
572: C
573: C ... 12 CHARACTER REGION/ZONE NAME
574: C
575:       if ( ISLATT(KMAT(INPZ,1)) ) then
576:         write(OUTLIN(1:LHEAD),7060) INPZ, NZONE, IZNAM(INPZ),
577:         &       KREGN(INPZ), LATTNM(KMAT(INPZ,1))
578:       else
579:         write(OUTLIN(1:LHEAD),7040) INPZ, NZONE, IZNAM(INPZ),
580:         &       KREGN(INPZ), KMAT(INPZ,1)
581:       end if
582: 7040      format(I5,2X,I5,2X,A,' : ',A,' : ',I6,' :')
583: 7060      format(I5,2X,I5,2X,A,' : ',A,' : L',I5,' :')
584:       IE = KK
585: C-----

```

```

586: C CONTINUATION LINE ( COLUMNS 1 - 5 ARE BLANK )
587: C-----
588:       else if ( NCHAR.ge.5.and.LINE(1:5).eq.' ' ) then
589:         IE = 5
590:       end if
591: C
592: C =====
593: C Loop to input body #'s or "OR" operators
594: C =====
595: C
596: 190 IS = IE + 1
597: C
598: C-----
599: C ..... FIND NON BLANK CHARACTER ELEMENT ( 'OR' OR INTEGER ) .....
600: C-----
601: C
602:       call NOBLNK( LINE(:NCHAR), IS, IE, NCHAR )
603:       if ( IE.gt.NCHAR ) go to 100
604:      >NNL = IE - IS + 1
605: C-----
606: C ..... 'OR' WAS FOUND .....
607: C-----
608:       if ( LINE(IS:IE).eq.'OR' ) then
609: C
610:         if ( NOR.eq.0 ) then
611:           NOR = 1
612:           if ( ISLATT(KMAT(INPZ,1)) ) then
613:             write(IMG,'(1x,a,a)')
614:             &       'XXX An input-zone used for lattice frame',
615:             &       ' cannot be composed using "OR",'
616:             write(IMG,*) ' Use separate input-zones, or you may ',
617:             &       'have to define more lattices.'
618:             call CNTERR( 'FATAL' )
619:           end if
620:         else
621: C
622: C-----
623: C ..... INPUT-ZONE # & ZDA POINTER OF ZONE (TEMPORARY) .....
624: C-----
625: C
626:       if ( NZONE.gt.0 ) then
627:         write(IUB) IBZ
628:         write(IUB) (KBBB(IB),IB=1,IBZ)
629:         NBZL = NBZL + IBZ
630:         IBZ = 0
631:       end if
632:       write(IUB) NZDA + 1, INPZ
633: C
634:       NZONE = NZONE + 1
635:       if ( INBODY.gt.0 ) then
636:         write(IOG,'(1X,A)') OUTLIN(:ICLEN2(OUTLIN))
637:       end if
638: C
639:       INBODY = 0
640:       OUTLIN = ' '
641:       write(OUTLIN(1:LHEAD),7080) NZONE, IZNAM(INPZ), KREGN(INPZ),
642:       &       KMAT(INPZ,1)
643: 7080      format(7X,I5,2X,A,' : ',A,' : ',I6,' :')
644:       end if
645:       go to 190
646:     end if
647: C
648: C .... a body which is used only for "frame" lattice and
649: C not used for actual zone shape definition is
650: C govern as "FRAME=body" or "F=body"

```

src/shared/zonein.f

```

651: C
652:     KFRAME = 0
653:
654:     if ( NNL.ge.5
655:     & .and.LINE(IS:IS+5).eq.'FRAME=' .or. LINE(IS:IS+1).eq.'F=' )
656:     & then
657:
658:         if ( JFISX.eq.0 ) then
659:             write(IMG,'(1x,a,a)')
660:             & 'XXX(ZONEIN) "Frame-only" body input by "FRAME=body" ',
661:             & ' is available only under FREE-LATTICE-FRAME mode.'
662:             call CNTERR( 'FATAL' )
663:         end if
664:         if ( INBODY.gt.0 ) then
665:             write(IMG,'(1x,a,a)')
666:             & 'XXX(ZONEIN) "Frame-only" body input by ',
667:             & '"FRAME=body" must precede any body input of an input-zone.'
668:             call CNTERR( 'FATAL' )
669:         end if
670:         if ( KNLAT.eq.0 ) then
671:             write(IMG,'(1x,a,a)')
672:             & 'XXX(ZONEIN) "Frame-only" body input by ',
673:             & '"FRAME=body" is given for a non-lattice-frame zone.'
674:             call CNTERR( 'FATAL' )
675:         end if
676:
677:         KFRAME = 1
678:         IS = IS + INDEX(LINE(IS:IE),'=')
679:     end if
680: C
681: C .... BODY #'s may be <...> ...
682: C
683:     if ( LINE(IS:IS).eq.'<' ) then
684:         KKK = INDEX(LINE(IS:NCHAR),'>')
685:         if ( KKK.eq.0 ) then
686:             write(IMG,'(1x,a,a)') 'XXX No closing ">" for a "<...>" ',
687:             & 'expression in body specification for a zone.'
688:             write(IMG,'(1x,a,a)') ' LINE: ', LINE
689:             call CNTERR( 'FATAL' )
690:             call PRSTOP( 0, 'Invalid body# in zone data.' )
691:             stop 888
692:         end if
693:         if ( IS+KKK-1.gt.IE ) then
694:             write(IMG,'(1x,a,a)')
695:             & 'XXX invalid use of closing ">" for a "<...>" ',
696:             & 'expression in body specification for a zone.'
697:             write(IMG,'(1x,a,a,a)') ' DATA:<', LINE(IS:NCHAR), '>'
698:             call CNTERR( 'FATAL' )
699:             call PRSTOP( 0, 'Invalid body# in zone data.' )
700:             stop 888
701:         end if
702:         IE = IS + KKK - 1
703:     end if
704: C
705: C -----
706: C .... READ IN BODY # & SIGN -----
707: C -----
708: C
709:     call READI4( LINE(:NCHAR), IS, IE, IBODY )
710: C
711:     JSIG = SIGN(1,IBODY)
712:     IBODY = ABS(IBODY)
713:     INBODY = INBODY + 1
714: C 18/MAR/92 ZONE-NAME 5 -> 12 CHARACTERS ...
715:     LH = LHEAD + (INBODY-1)*6 + 1

```

```

716: C
717:     if ( KFRAME.eq.0 ) then
718:         write(OUTLIN(LH:LH+6),'(I6)') JSIG*IBODY
719:     else
720:         write(OUTLIN(LH:LH+6),'(''F'',I5)') IBODY
721:     end if
722: C
723:     if ( LH+6+6.gt.LEN(OUTLIN) ) then
724:         write(IMG,'(1x,A)') OUTLIN
725:         OUTLIN = ' '
726:         INBODY = 0
727:     end if
728:     NOR = 1
729: C
730:     ... check body type # of "BBC" body ...
731: C
732:     call TYPBOD( 'BBC', '>', KBBC )
733: C-----
734: C .... FIND SDA DATA OF THIS BODY -----
735: C-----
736:     do 200 IB = 1, NBODY
737:         if ( IBSDA(2,IB).eq.IBODY ) go to 210
738:     200 continue
739: C
740:     write(IMG,'(1X,A,I6,A)') 'XXX(ZONEIN) A body whose BODY ID is ',
741:     & IBODY, ' does not exist.'
742:     write(IMG,'(1X,A)') OUTLIN(:ICLEN2(OUTLIN))
743:     call CNTERR( 'FATAL' )
744: C
745: C ... assign body #1 to prevent immediate error stop ...
746: C
747:     IB = 1
748: C
749: 210 continue
750: C
751:     IBZ = IBZ + 1
752:     if ( IBZ.gt.MXBPZ ) then
753:         write(IMG,7100) MXBPZ
754: 7100 format(1x,'XXX(ZONEIN) Number of bodies for current zone ',
755:         & 'exceeds internal buffer size (MXBPZ='&I4,&')/5x,
756:         & 'If you wish to make this body as such, you must change',
757:         & ' buffer size MXBPZ in ZONEIN routine,/5x,
758:         & ' but it is probably a VERY BAD IDEA to use too many',
759:         & ' bodies to compose a zone'/5x,
760:         & ' because it will make the code run slowly!!/5x,
761:         & ' Splitting the zone definition or using lattice ',
762:         & ' geometry(if possible) is better solution.')
763:         call PRSTOP( 1, 'Internal body buffer overflow.' )
764:         stop 999
765:     end if
766: C
767:     KBBB(IBZ) = JSIG*IB
768: C-----
769: C .... BUILD-UP 'KZDA' ARRAY (POINTERS TO SDA ARRAY INCLUDED IN ALL
770: C ZONES)
771: C-----
772: C=====
773: C KZDA(2,IB) : 0 = SURFACE IS USED BY 'AND'.
774: C 1 = SURFACE IS ONE OF THOSE COMPOSING A BODY WHOSE SIGN
775: C IS 'MINUS' AND HAVING MORE THAN ONE SURFACES.
776: C 2 = SIMILAR TO KZDA(2,IB) = 1, BUT THE LAST SURFACE OF THE
777: C BODY.
778: C -1 = body having this surface is used "frame-only" body
779: C for current zone.
780: C=====

```

src/shared/zonein.f

```

781: 7120 format(1x,'XXX Memory insufficient for zone data processing',/1x,
782: & ' ( shortage by ',I5,' words )')
783: C
784: NBS = 0
785: KZDA2 = 0
786: NZDA0 = NZDA
787: C
788: C ... for non "BBC" bodies
789: C
790: if ( IBSDA(3,IB).ne.KBBC ) then
791: C
792: ISDA = IPSDA(1,IBSDA(1,IB))
793: IISDA = NSDA + 1
794: if ( IB.lt.NBODY ) IISDA = IPSDA(1,IBSDA(1,IB+1))
795: C
796: 220 NZDA = NZDA + 1
797: NBS = NBS + 1
798: C
799: if ( 2*NZDA.gt.NWRK ) then
800: write(1MSG,7120) 2*NZDA - NWRK
801: call PRSTOP( 1, 'MEMORY OVER.' )
802: stop 999
803: end if
804: C
805: if ( KZDA2.eq.0.and.NBS.gt.1.and.JSIG.eq.-1 ) then
806: KZDA2 = 1
807: KZDA(2,NZDA-1) = 1
808: end if
809: C
810: KZDA(1,NZDA) = ISDA*JSIG
811: KZDA(2,NZDA) = KZDA2
812: C-----
813: C ..... 10E4 DIGIT OF SDA(ISDA) IS THE NUMBER OF DATA NECESSARY
814: C TO DEFINE THE SURFACE.
815: C (MAXSF : MAXIMUM OF NUMBER OF NECESSARY SURFACE DATA)
816: C-----
817: M = NINT(SDA(ISDA)/10000.0D0)
818: MAXSF = MAX(M-1,MAXSF)
819: ISDA = ISDA + M
820: if ( ISDA.lt.IISDA ) go to 220
821: C
822: C ... "BBC" type body ...
823: C
824: else
825: do 240 J = IBODI(IB), IBODI(IB+1) - 1
826: IIB = NINT(DBODI(J))
827: KI = ABS(IIB)
828: do 230 K = IBSDA(1,KI), IBSDA(1,KI+1) - 1
829: NZDA = NZDA + 1
830: NBS = NBS + 1
831: if ( 2*NZDA.gt.NWRK ) then
832: write(1MSG,7120) 2*NZDA - NWRK
833: call PRSTOP( 1, 'MEMORY OVER.' )
834: stop 999
835: end if
836: if ( KZDA2.eq.0.and.NBS.gt.1.and.JSIG.eq.-1 ) then
837: KZDA2 = 1
838: KZDA(2,NZDA-1) = 1
839: end if
840: C
841: KZDA(1,NZDA) = IPSDA(1,K)*JSIG*SIGN(1,IIB)
842: KZDA(2,NZDA) = KZDA2
843: 230 continue
844: 240 continue
845: end if

```

```

846: C
847: if ( KZDA2.eq.1 ) KZDA(2,NZDA) = 2
848: C
849: C ... KZDA(2,*) for frame only body are forced to negative value
850: C
851: if ( KFRAME.ne.0 ) then
852: do 250 I = NZDA0 + 1, NZDA
853: KZDA(2,I) = -1
854: 250 continue
855: end if
856: C
857: go to 190
858: C
859: C=====
860: C END OF ZONE DATA
861: C=====
862: C
863: 260 continue
864: if ( NZONE.gt.0 ) then
865: write(IUB) IBZ
866: write(IUB) (KBBB(IB),IB=1,IBZ)
867: NBZL = NBZL + IBZ
868: end if
869: C
870: C-----
871: C Print or check for the last zone
872: C-----
873: C
874: if ( INBODY.gt.0 ) write(IOG,'(1X,A)') OUTLIN(:ICLEN2(OUTLIN))
875: C
876: C ..... #SUBFRAME MUST BE GIVEN AFTER LATTICE FRAME ZONES IF JTLLT.ne.0
877: C
878: if ( INPZ.gt.1.and.JTLLT.ne.0 ) then
879: MAT0 = KMAT(INPZ-1,1)
880: CCCC if ( JOUT.eq.0.and.(MAT0.le.-1.and.MAT0.ge.-998) ) then
881: if ( JOUT.eq.0.and.ISLATT(MAT0) ) then
882: write(1MSG,'(1x,a,a,a/4x,a)')
883: & 'XXX A lattice-frame zone <',
884: & IZNAM(INPZ-1) (:ICLEN(IZNAM(INPZ-1))),
885: & '> must have a',
886: & ' "#SUBFRAME ... #END SUBFRAME" block',
887: & 'in "FRAME-DEPENDENT-TALLY"/"TALLY-LATTICE" mode.'
888: call CNTERR( 'FATAL' )
889: end if
890: end if
891: C
892: write(IOG,7140) HEAD2(:LHEAD),
893: & (' ----',I=1,(LEN(OUTLIN)-LHEAD)/6)
894: 7140 format(1X,A,15(:A))
895: C
896: C-----
897: C ..... DETERMINE POINTERS & PRINT OUT NUMBERS .....
898: C-----
899: C
900: call RESIZE( 'KZDA', LKZDA, 2*NZDA, 'I4', LAST )
901: C
902: call KEEP( 'KSFBD', LKSFBD, NZDA, 'I4', LAST, JDEBG )
903: call KEEP( 'KINPZ', LKINPZ, NZONE, 'I4', LAST, JDEBG )
904: call KEEP( 'KZAA', LKZAA, NZONE+1, 'I4', LAST, JDEBG )
905: call KEEP( 'NKZAA', LNKZAA, NZONE, 'I4', LAST, JDEBG )
906: C
907: call KEEP( 'KBZL', LKBZL, NBZL, 'I4', LAST, JDEBG )
908: call KEEP( 'IBZL', LIBZL, NZONE+1, 'I4', LAST, JDEBG )
909: C
910: if ( LSIZ(LAST)-1.gt.LIMIT ) then

```

src/shared/zonein.f

```

911:      write(IMG,'(1x,a/1x,a,i12,a)')
912:      &      'XXX Memory insufficient for zone data input',
913:      &      ' ( shortage by ', LSIZ(LAST) - 1 - LIMIT, ' words )'
914:      call PRSTOP( 1, 'MEMORY OVER.' )
915:      stop 999
916:      end if
917: C
918: C ... restore KINPZ, KZAA from working file IUB etc. ...
919: C & construct ksfbnd(nzda) array which is body ID's
920: C corresponding to KZDA data.
921: C
922: C call KZRSTR( JDEBG, NZONE, KZDA, NZDA, NBDY, NSURF, A(LKINPZ),
923: C &      A(LKZAA), A(LNKZAA), IBSDA, IPSDA, A(LKSFBD), A(LKBZL),
924: C &      A(LIBZL), NBZL )
925: C
926: C write(IOG,7160) NINPZ, NZONE, 2*NZDA
927: 7160 format('//4X, '== SUMMARY OF ZONE-INPUTS ====='//4X,
928: C &      ' NUMBER OF INPUT-ZONES = ',I6/4X,
929: C &      ' NUMBER OF ZONES      = ',I6/4X,
930: C &      ' LENGTH OF KZDA ARRAY = ',I6/4X,
931: C &      '=====')
932: C
933: C-----
934: C ..... REMOVE 10000*N TYPE DATA (NUMBER OF DATA FOR A SURFACE)
935: C FROM SDA ARRAY .....
936: C-----
937: C
938: C ISDA = 1
939: C
940: C 270 continue
941: C if ( ISDA.le.NSDA ) then
942: C   NS = INT(SDA(ISDA)/10000.0)
943: C   SDA(ISDA) = SDA(ISDA) - NS*10000.0
944: C   ISDA = ISDA + NS
945: C   go to 270
946: C end if
947: C
948: C-----
949: C ..... CHECK WHETHER WHOLE GEOMETRY IS SIMPLE OR NOT .....
950: C-----
951: C
952: C JSIMP = 1
953: C do 280 I = 1, NZDA
954: C   if ( KZDA(2,I).ne.0 ) then
955: C     JSIMP = 0
956: C     go to 290
957: C   end if
958: C 280 continue
959: C
960: C 290 if ( JSIMP.eq.1 ) write(IOG,7180)
961: 7180 format('// <<< ALL ZONES ARE SIMPLE (SURFACES WITH .AND. COMBI',
962: C &      'NATIONS ONLY) >>> ')
963: C
964: C-----
965: C ..... SET THE LAST VALUE OF IPCEL .....
966: C-----
967: C
968: C if ( JLATT.ne.0 ) IPCEL(NCELL+1) = NZONE + 1
969: C call RIUNIT( IOIN )
970: C
971: C-----
972: C ..... DEBUG PRINT .....
973: C-----
974: C
975: * IF( JDEBG(1) .NE. 0 ) THEN
976: *      WRITE(IOG,7020) ' KZDA ARRAY ',(IA(I),I=LKZDA,LKINPZ-1)
977: *      WRITE(IOG,7020) ' KINPZ ARRAY',(IA(I),I=LKINPZ,LKZAA-1)
978: *      WRITE(IOG,7020) ' KZAA ARRAY ',(IA(I),I=LKZAA,LAST-1)
979: *7020 FORMAT(/1X,A12/(1X,10I8))
980: *      ENDIF
981: CC
982: return
983: end

```

src/shared/INC/_ARRAY

```

1: C*
2: C* .... STORAGE OF VARIABLE LENGTH DATA ....
3: C*
4: C* UPDATE:
5: C* 9 Sep 1994: 'iainf' array is put on /ARRAYZ/ and /ARRAY/ common
6: C* is restored old style to keep backward compatibility in use
7: C* on FACOM VP.
8: C* 5 Apr 1995: enable dynamic allocation for compilers not supporting
9: C* pointer operation. (not recommended)
10: C* 27 Jun 1997: assign IAINF(3), IAINFL(3) as max size of memory
11: C* requirement but unused now ....
12: C* 2 Nov 1997: separate character data area to CHA (task common) &
13: C* CHH ( task local: if necessary )
14: C*
15: C/#IF DYNAMIC
16:
17: C/# IF NOT.NOPOINTER
18: C*
19: C* .... DYNAMIC ALLOCATION ( CRAY / HP9000 / NEC-SX ETC. )
20: C* ( effective on systems supporting POINTER statement )
21: C*
22: C* ALLOCATE MEMORY SUCH AS; LPA = MALLOC( (8/MWORD)*LIMIT )
23: C*
24: C* LIMIT : available memory size in word
25: C* IAINF(1) : starting array index of available part of memory body
26: C* ( LSTART )
27: C* IAINF(2) : starting array index of next unused memory block
28: C* ( LAST )
29: C* IAINF(3) : maximum of required memory size in word.
30: C*
31: C* ( size of array iainf must be an odd number for non-dynamic mode
32: C* to make address of a(1) is on double word boundary. )
33: C*
34: C* a(*) : memory body ( pointed by LPA in dynamic mode )
35: C*
36: * common /ARRAY/ LIMIT, IDUM, LPA
37: * common /ARRAYZ/ IAINF(3)
38: * integer LIMIT,IDUM,IAINF
39: C/#IF PARA( SX* )
40: * local common /LARRAY/ LIMITL, ILDUM, LPH
41: * local common /LARRAYZ/ IAINFL(3)
42: * integer LIMITL,ILDUM,IAINFL
43: C/#ELSEIF PARA( CRAY* )
44: * task common /LARRAY/ LIMITL, ILDUM, LPH
45: * task common /LARRAYZ/ IAINFL(3)
46: * integer LIMITL,ILDUM,IAINFL
47: C/#ELSE
48: C*
49: C*.. for single tasking mode, array A & H are equivalent.
50: C*
51: * integer LIMITL,IAINFL(3)
52: * equivalence (LIMITL,LIMIT), (IAINF,IAINFL), (LPH,LPA)
53: C/#ENDIF
54: * real A
55: * pointer ( LPA , A(1) )
56: * real H
57: * pointer ( LPH , H(1) )
58: *
59: C/# ELSE
60: C
61: C ... dynamic allocation on compilers not supporting pointer operation
62: C
63: C iainf(1) is really starting array index of allocated memory.
64: C
65: C This mode is very dangerous on 64-bit addressing architectures.

```

```

66: C
67: common /ARRAY/ LIMIT, IDUM, A(1)
68: common /ARRAYZ/ IAINF(3)
69: integer LIMIT,IDUM,IAINF
70: C*
71: integer LIMITL,IAINFL(3)
72: real H(1)
73: EQUIVALENCE (LIMITL,LIMIT), (IAINF,IAINFL), (A(1),H(1))
74:
75: C/# ENDIF
76:
77: C/#ELSE
78: C**
79: C** .... NON DYNAMIC ALLOCATION ....
80: C**
81: * common /ARRAY/ LIMIT, IDUM, A(1)
82: * common /ARRAYZ/ iainf(3)
83: * integer LIMIT,IDUM,IAINF
84: * real A
85: C/#IF PARA( SX* )
86: * local common /LARRAY/ LIMITL, ILDUM, H(1)
87: * local common /LARRAYZ/ IAINFL(3)
88: * integer LIMITL,ILDUM,IAINFL
89: C/#ELSEIF PARA( CRAY* )
90: * task common /LARRAY/ LIMITL, ILDUM, H(1)
91: * task common /LARRAYZ/ IAINFL(3)
92: * integer LIMITL,ILDUM,IAINFL
93: C/#ELSE
94: * real H(1)
95: * integer LIMITL,IAINFL(3)
96: * equivalence (LIMITL,LIMIT), (IAINF,IAINFL),( A,H )
97: C/#ENDIF
98: *
99: C/#ENDIF
100: C
101: C ... character data area (task shared) ....
102: C
103: integer CHAUNT
104: parameter ( CHAUNT = 4 )
105: integer LIMITC,IDUMC
106: character*(CHAUNT) CHA
107: common /CARRAY/ CHA(1)
108: common /CARRAYZ/ LIMITC, IDUMC, IAINFC(3)
109: integer IAINFC
110: C
111: C ... character data area (task local) .... for future use
112: C
113: C integer LIMITC,IDUMC
114: C character*(CHAUNT) CHH
115: C common /CLARRAY/ LIMITLC, ILDUMC, CHH(1)
116: C common /CARRAYZ/ IAINFLC(3)
117: C integer IAINFLC
118: C

```

src/shared/INC/_COUNTS

```
1: C
2: C ... special event counters in randomwalk process
3: C
4:     integer  MTANG,MDEFR
5:     real*8    DESUM,DESUM2
6:     integer  JLOOP,NLOOP
7:     integer  IFNSSC
8: C/#IF PARA( SX* )
9: *      local common  /COUNTS/
10: C/#ELSEIF PARA( CRAY* )
11: *      task common  /COUNTS/
12: C/#ELSE
13: common  /COUNTS/
14: C/#ENDIF
15:     S  MTANG,MDEFR,
16:     S  DESUM,DESUM2,JLOOP,NLOOP,
17:     F  IFNSSC
```


src/shared/INC/_FRDATA

```
1: C
2: C  COMMON DATA FOR THE MVP/GMVP TYPE INPUT-DATA HANDLERS.
3: C
4: C      COMMON /VFREAD/  COMMON /VFILE/      : VIRTUAL FILE
5: C      COMMON /FRDATA/      : MISCELLANEOUS DATA
6: C
7: C      PARAMETER ( MAXCOL = 72 )
8: C
9: C      ..... COMMON AREA TO SHARE DATA WITH RELATED ROUTINES/FUNCTIONS
10: C
11: C      COMMON /FRDATA/ iuin,is,ie,nchar,lalpm,NAA
12: C      INTEGER      iuin,is,ie,nchar,lalpm,NAA
13: C
14: C      ..... character data .....
15: C
16: C      COMMON /FRDATC/ ALPNUM
17: C      CHARACTER*64 ALPNUM
18: C
19: C      ..... BUFFER AREA FOR VIRTUAL FILE ....
20: C
21: C      PARAMETER ( MXVLIN = 32 )
22: C      COMMON /VFREAD/ IOSAVE,ISSAVE,NVLINE,IVLINE
23: C      COMMON /VFILE / LNSAVE
24: C      CHARACTER*(MAXCOL) LNSAVE(0:MXVLIN)
25: C
26: C
```

src/shared/INC/_IDXOFF

```
1: C
2: C ... position offset values for direct tally data structure
3: C
4: C ... offset of direct-tally data in IDTAL(*)
5: C
6:     parameter( IDTOFF  = 12 )
7: C
8: C ... offset of tally dimension dependent data in IDTAL(*)
9: C
10:    parameter( IDMOFF   = 3 )
11: C
```

DATA

src/shared/INC/_IOUNIT

```
1: C*
2: C* I/O unit parameters commonly used in GMVP & MVP :
3: C*
4: C+MVP_DELETE
5: C* -----
6: C* (10 Jan 2001) Y.Nagaya
7: C* INP is moved to the IIOSSET routine for Fujitsu AP3000 system.
8: C* The aprun command on MPI/AP does not guarantee the standard
9: C* input/output in the outside between MPI_INIT and MPI_finalize.
10: C* -----
11: C* (January 2000)
12: C* Message I/O unit MSG is added.
13: C* It should be set identical to IPR in most case, but may be set to
14: C* different value for multi task calculation to see only error
15: C* messages and throw away other printout to IPR on /dev/null (in
16: C* UNIX and variants. "NULL" for MS-DOS family)
17: C* -----
18: C* (January 1999)
19: C* IOIN, IPR, IOG, IOW, IOT and IOF is declared as COMMON variables
20: C* and initial values are set in IIOSSET routine.
21: C* -----
22: C-MVP_DELETE
23: C*
24: C* INP : standard input (unit 5)
25: C* IOIN : data input unit
26: C* IPR : standard output (printout), but may be set to null device.
27: C* IOSTDO : for standard output of the process (should not be changed)
28: C*
29: C* MSG : "message" printout which includes error, warning messages
30: C* MSG will be identical to IPR in most single-task
31: C* calculation, but may differ in multi-task calculation.
32: C*
33: C* IOG : printout unit in geometry processing phase
34: C* IOW : printout unit in random walk phase
35: C* IOT : printout unit in tally data output phase (other than flux)
36: C* IOF : printout unit of calculated flux
37: C*
38: C* IUG1 : scratch unit for geometry processing (1) <text>
39: C* IUG2 : scratch unit for geometry processing (2) <text>
40: C* IUB : scratch unit for geometry processing etc. <binary>
41: C*
42: C* IROT : restart output (binary)20
43: C* IRIN : restart input (binary)10
44: C* IORS : output of calculation results (binary) 30
45: C* IOTL : output of time list data (binary) 31
46: C*
47: C* IOSI : unit of source input file
48: C* IOSO : unit of source output file
49: C*
50: C* IONULL : a constant used for I/O unit opened for null device
51: C*
52: C* -----
53: C* I/O units placed on COMMON area and preset in IIOSSET routine.
54: C* -----
55: C*
56: common /IOUIO/ INP, IOIN, IPR, IOSTDO, MSG, IOG, IOW, IOT, IOF
57: parameter ( IUG1=15, IUG2=75, IUB =16,
58: & IROT=20, IRIN=10, IORS=30, IOTL=31,
59: & IOSI= 8, IOSO= 9 )
60: parameter ( IONULL = 4 )
61: C
62: C+MVP_DELETE
63: CC/#IF PARA(VPP)
64: C COMMON /RWIOW/ IOW
65: C PARAMETER ( INP=5, IOIN=55, IPR=6 )
66: C PARAMETER ( IOG=6, IOT=6, IOF=6,
67: C & IUG1=15, IUG2=75, IUB = 16,
68: C & IROT=20, IRIN=10, IORS=30,
69: C & IOSI=8, IOSO=9 )
70: CC/#ELSEIF SYSTEM(PARAGON)
71: C PARAMETER ( INP=5, IOIN=55, IPR=7 )
72: C PARAMETER ( IOG=7, IOW=7, IOT=7, IOF=7,
73: C & IUG1=15, IUG2=75, IUB = 16,
74: C & IROT=20, IRIN=10, IORS=30,
75: C & IOSI=8, IOSO=9 )
76: CC/#ELSE
77: C PARAMETER ( INP=5, IOIN=55, IPR=6 )
78: C PARAMETER ( IOG=6, IOW=6, IOT=6, IOF=6,
79: C & IUG1=15, IUG2=75, IUB = 16,
80: C & IROT=20, IRIN=10, IORS=30,
81: C & IOSI=8, IOSO=9 )
82: CC/#ENDIF
83: C-MVP_DELETE
```

src/shared/INC/_LINEAR

```
1: C*
2: C* .... STATEMENT FUNCTION FOR LINEAR SAMPLING & OPTIONAL INT() ....
3: C*      ( JAN 1992   BY M.SASAKI )
4: C*
5: C*      LINEAR(N,R) : sample from 1 to N ( N>0 ) using
6: C*                    uniform random number 'R' ( 0 <= R <= 1)
7: C*      LINEA0(N,R) : sample from 0 to N-1 ( N>0 ) or 0 using
8: C*                    uniform random number 'R' ( 0 <= R <= 1)
9: C*      OPTINT(X,i) : optional integer function for machines
10: C*                   which make floating-point-roundoff not by
11: C*                   truncation but by adjustment to the nearest value.
12: C*
13: C/#IF ROUNDOFF(TRUNCATE)
14: C*-- FACOM VP or OTHER MAINFRAME MACHINES ---
15: *      INTEGER LINEAR,LINEA0,OPTINT
16: *      LINEAR(N,RANDOM) = N*RANDOM + 1
17: *      LINEA0(N,RANDOM) = N*RANDOM
18: *      OPTINT(X,I) = X + I
19: C/#ELSEIF ROUNDOFF(NEAREST)
20: *      INTEGER LINEAR,LINEA0,OPTINT
21: *      LINEAR(N,RANDOM) = MIN( N*RANDOM + 1, N )
22: *      LINEA0(N,RANDOM) = INT( N*RANDOM )
23: *      OPTINT(X,I) = INT(X) + I
24: C/#ELSE
25: *      INTEGER LINEAR,LINEA0,OPTINT
26: *      LINEAR(N,RANDOM) = MIN( N*RANDOM + 1, N )
27: *      LINEA0(N,RANDOM) = INT( N*RANDOM )
28: *      OPTINT(X,I) = INT(X) + I
29: C/# ENDIF
```

src/shared/INC/_LISTOFF

```
1: C/#IF SYSTEM(FACOM*)  
2: *      LIST OFF  
3: C/#ENDIF
```

SAFE

src/shared/INC/_LISTON

```
1: C/#IF SYSTEM(FACOM*)  
2: *      LIST ON  
3: C/#ENDIF
```

SAFE

src/shared/INC/_LNAM

```
1: C*
2: C*  LNAM   : ..... RESERVED CHARACTER "REGION" NAMES ...
3: C*  MAXLNM : ..... ALLOWED LENGTH OF "REGION" NAMES ...
4: CC/#IF WORD(64)
5: C*      PARAMETER ( LNAM = 16 , MAXLNM = 12  )
6: CC/#ELSE
7:      PARAMETER ( LNAM = 12 , MAXLNM = LNAM )
8: CC/#ENDIF
```

SAFE

src/shared/INC/_PGEOM

```

1: C*
2: C*.... GEOMETRY ARRAY POINTERS .....
3: C*
4:     COMMON /PGEOM/
5:     &          LSDA , LIPSDA, LKMAT , LKREG , LKZMAT, LKZREG,
6:     &          LKINPZ, LKZDA , LKZAA , LVOL , LPAPER ,
7:     &          LKSTREF, LKKREF,
8:     &          LKCELL, LIDCEL, LICTYP, LIPCEL, LCELSZ,
9:     &          LIDLAT, LLTYP , LNVLAT, LSZLAT, LIPLAT, LKLATT ,
10:    &          LKSLAT, LKZLBZ, LMLBZZ, LNKZAA, LKSFBZ,
11:    &          LDALT , LKDALI, LKHLAT, LKREGN, LKREGI, LKCELI,
12:    &          LIPCLI, LIBSDA, LDBODI, LIBODI, LGRBOD,
13:    &          LKBZL, LIBZL,
14:    &          LIZNAM, LPNND, LBNND,
15:    &          LPPPF , LKPPF
16: C*
17: C SDA(NSDA)      R8 Surface data.
18: C*** added 9 Oct 96 *** IPSDA(3,*) added May 1999
19: C IPSDA(3,NSURF+1) R8
20: C $ (1,i) : Pointer to SDA(*) for each surface.
21: C $ Surface 1 is from SDA(IPSDA(i)) to SDA(IPSDA(i+1)-1).
22: C $ (2,i) : Body ID of the surface if composing a body.
23: C $ (3,i) : Body sequence # of the surface if composing a body.
24: C KMAT(NINPZ) I4 Material # of each input-zone.
25: C KREG(NINPZ) I4
26: C $ Non-frame-dependent tally mode:
27: C $ Region # of each input-zone.
28: C $ Frame-dependent tally mode:
29: C $ Region # for zones not in subspace.
30: C $ If -1<= MATERIAL # < -999 : lattice #
31: C $ If in cell : negated sequential # of input zones
32: C in all sub-spaces.
33: C KREGN(NINPZ) CH12 Region name of each input-zone.
34: C KREGI(NINPZ) I4 Position of 'KREGN's in name table 'TNAMS'.
35: C*** MZMAT(*,2) is added Nov 1998 for STG cell treatment
36: C KZMAT(NZONE,2) I4 Material # of each zone.
37: C $ KZMAT(NZONE,1): Material ID (or sequential #) of
38: C negated lattice ID (lattice #).
39: C $ KZMAT(NZONE,2): Base material(matrix) material # for STG region,
40: C else zero.
41: C
42: C Some routines may declare KZMAT as KZMAT(NZONE) to use
43: C only KZMAT(*,1).
44: C
45: C KZREG(NZONE) I4 Region # of each zone.
46: C $Non-frame-dependent tally mode:
47: C $ Region # of each zone.
48: C $Frame-dependent tally mode:
49: C $ Region # for zones not in subspace.
50: C $ If -1<= MATERIAL # < -999 : lattice #
51: C $ If in cell : negated sequential # of input zones
52: C in all sub-spaces.
53: C KINPZ(NZONE) I4 Input zone # of each zone.
54: C KZDA(2,NZDA) I4 Zone shape data. Pointers to composing surface data
55: C 'SDA' and additional information. This array is pointed by 'KZAA'
56: C array.
57: C $ KZDA(1,I) : sign * <pointer to 'SDA' array>
58: C $ sign = 1 : inside of the surface
59: C $ sign = -1 : outside of the surface
60: C $ KZDA(2,I) : 0 = Surface is simply connected by 'AND' operator.
61: C $ 1 = Surface is one of those composing a body whose
62: C $ sign is 'minus' and having more than one surfaces.
63: C $ 2 = Similar to KZDA(2,I) = 1, but the last surface
64: C $ of the body.
65: C $ -1 = this surface is not used for zone shape definition

```

```

66: C $ (" -1" is introduced on Nov 1999).
67: C KSFBZ(NZDA) I4 Body ID of surfaces indicated by KZDA(1,*).
68: C KZAA(NZONE+1) I4 Pointer to 'KZDA' array. Zone K has$
69: C $ KZAA(K+1)-KZAA(K) surfaces.
70: C VOL(NINPZ) R4 Volumes of each input zone.
71: C PAPER(12,NPICT) R4 Data specifying drawing areas of each picture.$
72: C $ <JPIC>
73: C KSREF(NZDA) I4 Reflective surface #'s if non-zero <JREFL>
74: C If KSREF(i) is non-zero, KZDA(1,i) points a surface
75: C which compose reflective boundary (reflection surface) and
76: C KSREF(i) is reflection surface number.
77: C KKREF(2,NREFS) I4 Pointers to 'SDA' data of each reflection surfaces$
78: C $ & type of reflective material beyond them. <JREFL>
79: C*
80: C***** LATTICE GEOMETRY ***** < JLATT >
81: C*
82: C KCELL(NZONE) I4 I'th zone belongs to cell KCELL(i)'th cell. <JLATT>
83: C IDCEL(NCELL) I4 Cell ID numbers. <JLATT>
84: C ICTYP(NCELL) I4 Cell type of each cell. <JLATT>
85: C 1: Rectangular cell.
86: C 2: Hexagonal cell.
87: C 3: STG cell (sphere).
88: C 3: Free
89: C 3: Base material of STGM region (virtual cell)
90: C IPCEL(NCELL+1) I4 Zone pointers for each cell. <JLATT>
91: C IPCEL(N)'th zone to IPCEL(N+1)-1'th zone belong to N'th cell.
92: C CELSZ(6,NCELL) R8 Cell size (X,Y & Z). <JLATT>
93: C*
94: C IDLAT(NLATT) I4 ID numbers of lattice. <JLATT>
95: C LTYP (NLATT) I4 Type of lattice. <JLATT>
96: C 1: "BOX" or "RPP"-cell lattice (LTYP = 1)
97: C
98: C 2,3,4: "RHP" cell lattice ( LTYP = 2,3,4 )
99: C 2 = Lattice frame is "RHP" or "HEX"
100: C 3 = Lattice frame is cylinder.
101: C 4 = Lattice frame is "BOX" or "RPP"
102: C NVLAT(4,NLATT) I4 Division number of lattices. <JLATT>
103: C Lattice by "BOX"/"RPP" cell (LTYP = 1)
104: C NVLAT(1,N) : in "X" -direction of cell array
105: C NVLAT(2,N) : in "Y" -direction of cell array
106: C NVLAT(3,N) : in "Z" -direction of cell array
107: C Lattice by "RHP" cell ( LTYP = 2,3,4 )
108: C NVLAT(1,N) : in "I" -direction of cell array
109: C NVLAT(2,N) : in "J" -direction of cell array
110: C NVLAT(3,N) : in axis-direction of cell array
111: C*
112: C SZLAT(8,NLATT) R8 Lattice size.<JLATT>
113: C
114: C $ SZLAT * TYPE 1 * TYPE 2 * TYPE 3 * TYPE 4
115: C $ -----*-----*-----*-----*-----
116: C $ 1 * CELL-SIZE(X) * CELL PITCH
117: C $ 2 * (Y) * ANGLE OF CELL ARRAY
118: C $ 3 * (Z) * CELL HEIGHT
119: C $ -----*-----*-----*-----*-----
120: C $ 4 * ... * ORIGIN OF CELL ARRAY (X)
121: C $ 5 * ... * ORIGIN OF CELL ARRAY (Y)
122: C $ -----*-----*-----*-----*-----
123: C $ (LATTICE FRAME SIZE)
124: C $ -----*-----*-----*-----*-----
125: C $ 6 * ... * LATTICE- * CYLINDER * WIDTH (X)
126: C $ * * WIDTH * -RADIUS *
127: C $ 7 * ... * HEIGHT * HEIGHT * WIDTH (Y)
128: C $ 8 * ... * * * * WIDTH (Z)
129: C $ -----*-----*-----*-----*-----
130: C $

```


src/shared/INC/_PGEOM

```
131: C $ <For STG region (LTYP=10) >
132: C $
133: C $ SZLAT(1) PF : packing fraction.
134: C $ SZLAT(2) RSTG : spherical cell radius (cm) (optional)
135: C $ SZLAT(3) WNND1 : radius mesh width relative to 2*RSTG for NND-1
136: C $ (single value currently)
137: C $ SZLAT(4) WNND2 : radius mesh width relative to 2*RSTG for NND-2
138: C $ (single value currently)
139: C $ SZLAT(5) WNND3 : radius mesh width relative to 2*RSTG for NND-3
140: C $ (single value currently)
141: C*
142: C*
143: C IPLAT(NLATT) I4 Lattice data pointers. <JLATT>
144: C KLAT(IPLAT(NLATT+1)-1) I4 Cell #s in each lattice. <JLATT>
145: C KSLAT(IPLAT(NLATT+1)-1) I4 Cell directions in each lattice. <JLATT>
146: C KZLBZ(NLBZ) I4 Zone number of each lattice-buffer zone. <JLATT>
147: C MLBZZ(NZONE) I4 Lattice-buffer zone #'s of each zone. <JLATT>
148: C DALT(*) R8 Lattice geometry data. <JLATT>
149: C KDALT(NLBZ) I4 Pointers to lattice geometry data. <JLATT>
150: C KHLAT(IPLAT(NLATT+1)-1) I4 Flag indicating that a part or whole$
151: C $ of a cell in a hexagonal lattice is out of$
152: C $ lattice-frame or not.
153: C**** ADD 1988 NOV.
154: C NKZAA(NZONE) I4 Number of surfaces composing each zone.
155: C**** ADD 1992 APR.
156: C KCELI(NINPZ) I4 Belonging cell # of each input-zone.
157: C**** ADD 1992 APR.
158: C IPCLI(NCELL+1) I4 Input-zone pointers for each cell.
159: C IPCEL(N)'th to IPCEL(N+1)-1'th input-zones belong to N'th cell.
160: C**** ADD 1993 JUL.
161: C IBSDA(3,NBODY+1) I4 Surface data pointer & ID of each body.
162: C $ IBSDA(1,I) : Surface IBSDA(1,i) to IBSDA(1,i+1)-1 are composing
163: C $ i'th body.
164: C $ IBSDA(2,I) : ID of I'th body
165: C $ IBSDA(3,I) : type # of I'th body (body type is obtained by
166: C calling TYPBOD routine.
167: C**** added May 1999
168: C GRBOD(6,NBODY) R8 x,y,z coordinate range for each body.
169: C $ GRBOD(1,i) < x < GRBOD(2,i)
170: C $ GRBOD(3,i) < y < GRBOD(4,i)
171: C $ GRBOD(5,i) < z < GRBOD(6,i)
172: C*
173: C GRBOD(1,i) = +infinity and/or GRBOD(2,i) = -infinity
174: C when coordinate range of the body could not determined.
175: C*
176: C**** changed the meaning of IBSDA(1,*) -- 9 Oct 1996
177: C* IBSDA(1,I) : SDA(IBSDA(1,i)) to SDA(IBSDA(1,i+1)-1) are surface
178: C* data of I'th body.
179: C*
180: C**** added May 1999
181: C DBODI(*) R8 List of body geometry data as input. Data of I'th body
182: C is pointed by IBODI(I).
183: C IBODI(NBODY+1) I4 Index to body geometry data DBODI(*).
184: C Data of i'th body is from DBODI(IBODI(i)) to DBODI(IBODI(i+1)-1).
185: C
186: C IZNAM(NINPZ) C12 Name of input-zones.
187: C KBZL(NZBL) I4 list of body#*sign to define zones
188: C IBZL(NZONE+1) I4 pointers to KBZL(*) for each zone
189: C***
190: C*
191: C***** local data pointers in multi processing
192: C*
193: C COMMON /PGEOML/ LKMEMO, LMEMC, LMEMZ
194: C*
195: C*KMEMO(NMEMO,NZONE) I4 Memory of next-entering zones for each zone.
196: C KMEMO(NZONE,NMEMO) I4 Memory of next-entering zones for each zone.
197: C MEMC(NZONE,NMEMO) I4 Counter of entering events for zones$
198: C $ memorized in 'KMEMO' array.
199: C MEMZ(NZONE) I4 Used elements in 'KMEMO' array for each zone.
200:
201: C*
202: C***** DISTANCE LIMITS /PGEOM2/
203: C*
204: C COMMON /PGEOM2/ DEPS, DINF, GRANGE(6)
205: C REAL*8 DEPS,DINF, GRANGE
206: C*
207: C DEPS R8 Recognizable minimum distance. (default = 1.0d-05 cm)
208: C DINF R8 distance treated as infinity. (default = 1.0d+30 cm)
209: C GRANGE(6) R8 extent of geometry guessed from some type of bodies.
210: C $ GRANGE(1:2) : minimum and maximum of X coordinate
211: C $ GRANGE(3:4) : minimum and maximum of Y coordinate
212: C $ GRANGE(5:6) : minimum and maximum of Z coordinate
213: C*
214: C PNND(NPNND) R4 Nearest Neighbour Distribution for STG-region
215: C KNND(NPNND) I4 Aliases for alias sampling from
216: C Nearest Neighbour Distribution for STG-region
217: C*
218: C PPPF(NPPPF) R4 Partial Packing Fractions of each STG in STG-region
219: C KPPF(NPPPF) I4 Aliases for alias sampling from
220: C Partial Packing Fractions of each STG in STG-region
```

src/shared/INC/_PMLATT

```
1: C
2: C -----
3: C   Data declaration and parameter statement for
4: C   Statement functions ISLATT and LATNM to check lattice zone
5: C   from material # placed in _SFLATT
6: C
7: C -----
8: C
9: C   Data type declaration for ISLATT and LATNM and
10: C   parameter definition of MLTOFF is placed separatly in _PMLATT
11: C   to clear statement order standard of Fortran.
12: C
13: C -----
14: C
15: C   logical ISLATT
16: C   integer LATNM
17: C
18: C
19: C -----
20: C
21: C   parameter( MLTOFF = -10000 )
22: C
23: C -----
24: C
```

src/shared/INC/_PTALY0

```

1: C*
2: C*.... TALLY ARRAY POINTERS & PARAMETERS (FOR MVP & GMVP) .....
3: C*
4: C*
5: C* 2 Jul 1998: pointers to task local data are separated.
6: C* 23 Jun 1999: NCNTR(27)/WCNTR(27) is assigned for particle generation
7: C* cutoff.
8: C*
9: COMMON /PTALY0/ NEVENT,LTRVOL,LRVOL,
10: & LENGYB,LENGPB,LTIMEB,LRESP,
11: & NMXLG, NEITER, NMKREG, LMKREG,
12: & NNCST, LINCST, NNCST, LINCST,
13: & NTSRF,LIDSRF,MTSRF,LITSRF,LKTSRF,LKTSFJ,
14: & NANGLE,LANGLB,
15: c+beffl
16: & NEBEF,
17: c-beffl
18: c##<2007/03/14:PN3:
19: & NNCSTPN,LINCSTPN,NMTMPN,LMTMPN
20: c##>
21: C
22: common /TALY0D/ WFFACT
23: real*8 WFFACT
24: C*
25: C NEVENT I4 number of event counter entry (NCNTR & WCNTR)
26: C TRVOL(NREG) R4 Volumes of each tally-region.
27: C RVOL(NREG) R4 Volumes of each region.
28: C ENGYB(NGROUP+1 or NGROUP+2) R4 Energy group boundaries for neutron
29: C (and PHOTON for coupled problem).
30: C For NEUTRON & PHOTON problem, photon enrgy boundary data
31: C (ENGPB) is copied on ENGYB(NGP1+1:NGROUP+2).
32: C ENGPB(NGP2+1) R4 Energy boundaries (photon)
33: C*
34: C TIMEB(NTIME+1) R4 Time boundaries.
35: C*
36: C* ... added 7 Aug 1998
37: C*
38: C NMXLG I4 maximum generation lag for lag-i correlation calculation
39: C in "real variance" estimation in eigenvalue problems.
40: C NEITER I4 maximum iteration count in "real variance" estimation in
41: C eigenvalue problems.
42: C*
43: C*** Added 4 Oct 1999
44: C INCST(NUC,2) I4 List of nuclide # (NUCST(*,1)) or atom # (INCST(*,2))
45: C for which "SPREAD" mode of special tally is specified.
46: C This is used for ADAPTIVE-MICRO-CALCULATION mode, to
47: C calculate micro x-sec of these nuclides always after
48: C collision.
49: C
50: C NNCST I4 number of nuclides or atoms for which "SPREAD" mode of
51: C special tally is specified. (maximum of nuclide #'s or
52: C atom #'s )
53: C*** added 25 Jun 2000, INCSI and NNCST for next event estimator
54: C*
55: C* ... added Sep 1999
56: C*
57: C NMKREG I4 number of "marker" regions, tallies of particles passed
58: C the regions are taken separately by using the regions as
59: C tally bin.
60: C MKREG(NREG,2) I4
61: C MKREG(i,1): Marker region # for i'th region.
62: C Zero for non marker regions.
63: C MKREG(i,2): region # for i'th marker region.
64: C (NMKREG <= NREG, of course!!)
65: C

```

```

66: C* ... added 7 Feb 2000
67: C WFFACT R8 normalization factor of fission source weight calculated
68: C at the beginning of each batch (subroutine FISSET).
69: C
70: C* ... added Mar 2000
71: C NTSRF I4 number of surfaces for surface crossing tally
72: C ("tally surface")
73: C IDSRF(NTSRF) I4 surface ID list of tally surface.
74: C MTSRF I4 length of tally surface information array KTSRF(*)
75: C ITSRF(NTSRF+1,2) I4 pointer to KTSRF for each tally surface.
76: C See description of KTSRF.
77: C KTSRF(2,MTSRF) I4 tally surface composition data.
78: C this array stores pointers to surface data (SDA(*)) and
79: C other control information.
80: C
81: C Elements KTSRF(*,ITSRF(n,1):ITSRF(n+1,1)-1) are for surface n.
82: C KTSRF(*,ITSRF(n,1):ITSRF(n,2)|) is actual crossing surface
83: C and KTSRF(*,ITSRF(n,2)|+1:ITSRF(n+1,1)-1) compose psuedo-zone
84: C for clipping if |ITSRF(n,2)|<ITSRF(n+1,1)-1.
85: C Sign of ITSRF(n,2) shows that actual crossing surface is used
86: C as inside (+) or outside(-1) of a body.
87: C
88: C $ KTSRF(1,*) : pointer to surfaces
89: C $ KTSRF(2,*) : sign to surfaces
90: C $ KTSRF(1:2,*) works like KZDA(1:2,*) of zone definition.
91: C KTSFJ(MTSRF) I4 For surfaces composing tally-surface, surfaces
92: C having non-zero element are tallied-surfaces.
93: C Those having zero element are used only to "clip" actual
94: C tally-surfaces.
95: C Elements KTSFJ(ITSRF(n):ITSRF(n+1)-1) are for surface n.
96: C
97: C NANGLE I4 Number of angle tally bins.
98: C ANGLB(ANGLE+1) R8 Angle tally bin boundary (in cosine from angle 0
99: C to pai ).
100: c+beffl
101: C NEBEF I4 Number of event counter entries for beta-effective
102: C calculation.
103: c-beffl
104: c##<2007/03/14:PN3:
105: C
106: C INCSTPN(NUCPN,2) I4 list of photonuclear nuclide # for which
107: C "SPREAD" mode of special tally is specified, in
108: C INCSTPN(*,1).
109: C search index of (imt*100+nd) for microscopic response
110: C reactions, in INCSTPN(*,2).
111: C NNCSTPN I4 number of photonuclear nuclides for which "SPREAD" mode
112: C of special tally is specified. (maximum of nuclide #'s)
113: C
114: C MTMPN(NMTMPN) I4 list of reaction # using to microscopic photo-
115: C nuclear cross section.
116: C NMTMPN I4 total number of reaction # using to microscopic
117: C photonuclear.
118: c##>
119: C*.....
120: C*.....
121: C* pointers to task local data
122: C*.....
123: C*
124: COMMON /PTALYL/
125: & LWSUM , LWLEK , LNLOST ,
126: & LFLTR , LFLCL , LRETR , LRECL ,
127: & LSFLTR , LSFLCL , LSRETR , LSRECL ,
128: & LXSOC , LXSV , LXKEF ,
129: & LNCLSN , LNCLSN , LNLEAK , LNABSB , LWABSB , LNECUT , LWECUT ,
130: & LNTCUT , LWTCT , LNKILD , LWKILD , LNSURV , LWSURV , LNSPLT , LWSPLT ,

```

src/shared/INC/_PTALY0

```

131:      & LNCNTR, LWCNTR,
132:      & LXAVT, LAAVT, LEAVT,
133:      & LTFLH, LTFLHS,
134: c+beff1
135:      & LWCBEF, LXBEF,
136: c-beff1
137: c##<2007/03/14:PN3:
138:      & LNMPNWT, LWMPNWT
139: c##>
140: C*
141: C WSUM(1) R8 Sum of weights of particles (histories) run so far.
142: C WLEK(1) R8 Sum of weights of leaked particles. <JEIGN>
143: C NLOST(1) I4 Number of lost particles.
144: C***** ARRAYS *****
145: C RESP(NGROUP,NRESP) R4 Response function <JRESP>
146: C FLTR(NGROUP,NREG) R8 Flux by track length estimator (batch tally)
147: C SFLTR(NGROUP,NREG,2) R8 Flux by track length estimator
148: C      (,,1) : sum of batch tallys
149: C      (,,2) : square sum of batch tallys or variance
150: C FLCL(NGROUP,NREG) R8 Flux by collision estimator (batch tally)
151: C SFLCL(NGROUP,NREG,2) R8 Flux by collision estimator
152: C      (,,1) : sum of batch tallys
153: C      (,,2) : square sum of batch tallys or variance
154: C XSOC(NLENG) R8 Source neutron weight of each batch <JEIGN>
155: C XSXV(NLENG,3) R8 Variance of x,y,z of source neutron (batch) <JEIGN>
156: C XKEF(7,NLENG) R8 (MVP) Eigenvalue tallies of each batch.<JEIGN>
157: C
158: C      (1,) track length (production)
159: C      (2,) collision (production)
160: C      (3,) analog (production)
161: C      (4,) track length (absorption)
162: C      (5,) collision (absorption)
163: C      (6,) analog (absorption)
164: C      (7,) leakage
165: C
166: C XKEF(5,NLENG) R8 (GMVP) Eigenvalue tallies of each batch.<JEIGN>
167: C
168: C      1: production -track length estimator
169: C      2: production -collision estimator
170: C      3: absorption -track length estimator
171: C      4: absorption -collision estimator
172: C      5: leakage
173: C
174: C RETR(NREG,NRESP) R8 Reaction rate by track-length estimator
175: C      (batch tally) <JRESP>
176: C SRETR(NREG,NRESP,2) R8 Reaction rate by track-length estimator
177: C      <JRESP>
178: C      (,,1) sum of batch tallys
179: C      (,,2) square sum of batch tallys
180: C
181: C RECL(NREG,NRESP) R8 Reaction rate by collision estimator
182: C      (batch tally) <JRESP>
183: C SRECL(NREG,NRESP,2) R8 Reaction rate by collision estimator
184: C      <JRESP>
185: C      (,,1) sum of batch tallys
186: C      (,,2) square sum of batch tallys
187: C*
188: C NCLSN(NGROUP,NREG) R8 Number of collision <JMNTN>
189: C WCLSN(NGROUP,NREG) R8 Sum of collision weight <JMNTN>
190: C*NLEAK(NGROUP,NREG) R8 Number of leaked particle <JMNTN>
191: C*WLEAK(NGROUP,NREG) R8 Sum of leaked weight <JMNTN>
192: C*NABSB(NGROUP,NREG) R8 Number of absorbed particle <JMNTN>
193: C*WABSB(NGROUP,NREG) R8 Sum of absorbed weight <JMNTN>
194: C*NECUT(NREG) R8 Number of energy cut off <JMNTN>
195: C*WECUT(NREG) R8 Sum of energy cut off weight <JMNTN>

```

```

196: C*NTCUT(NGROUP,NREG) R8 Number of time cut off <JMNTN>
197: C*WTCUT(NGROUP,NREG) R8 Sum of time cut off weight <JMNTN>
198: C*NKILD(NGROUP,NREG) R8 Number of killed particle <JMNTN>
199: C*WKILD(NGROUP,NREG) R8 Sum of killed weight <JMNTN>
200: C*NSURV(NGROUP,NREG) R8 Number of survived particle <JMNTN>
201: C*WSURV(NGROUP,NREG) R8 Sum of survived weight <JMNTN>
202: C*NSPLT(NGROUP,NREG) R8 Number of splitted particle <JMNTN>
203: C*WSPLT(NGROUP,NREG) R8 Sum of splitted weight <JMNTN>
204: C*
205: c##<2007/03/14:PN3:PN4:
206: c##C NCNTR(NEVENT=30,2) R8 (MVP) Event counters. ( see WCNTR )
207: c##C WCNTR(NEVENT=30,2) R8 (MVP) Sums of weight for various events.
208: C NCNTR(NEVENT=34,2) R8 (MVP) Event counters. ( see WCNTR )
209: C WCNTR(NEVENT=34,2) R8 (MVP) Sums of weight for various events.
210: c##>
211: C
212: C ***** NEUTRON EVENTS ***** ( NCNTR(I,J) J=1 )
213: C
214: C      1: source 2: fission-s 3: (n,gamma) 4: collision
215: C      5: split (importance or weight window)
216: C      6: prevented splitting 7: leakage 8: energy cut
217: C      9: kill (importance or weight window)
218: C     10: survived (importance or weight window)
219: C     11: weight cut (killed) 12: weight cut (survived)
220: C     13: non-fission 14: production/batch (colisn)
221: C     15: production/batch (flight)
222: C     16: free-flight
223: C     17: boundary crossing
224: C     18: reflection
225: C     19: loss/batch (colisn)
226: C     20: loss/batch (flight)
227: C     21: production/batch (analog)
228: C     22: loss/batch (analog)
229: C     23: analog absorption
230: C     24: russian roulette killed at fisgen or phtgen
231: C     25: added (n,mn) neutrons
232: C     26: time cutoff
233: C     27: particle generation number cutoff
234: C     28: endless loop cutoff (photon is merged to this)
235: c##<2007/03/14:PN3:PN4:
236: c##C      29-30: for future use
237: C     29: generated by photonuclear reaction
238: C     30: delayed fission neutron
239: C     31: photonuclear_production/batch (collision)
240: C     32: photonuclear_production/batch (flight)
241: C     33: photonuclear_production/batch (analog)
242: C     34: for future use
243: c##>
244: C
245: C ***** PHOTON EVENTS ***** ( NCNTR(I,J) J=2 )
246: C
247: C      1: source 2: bremsstrahlung 3: (n,gamma) 4: collision
248: C      5: split (importance or weight window)
249: C      6: prevented splitting 7: leakage 8: energy cut
250: C      9: kill (importance or weight window)
251: C     10: survived (importance or weight window)
252: C     11: weight cut (killed) 12: weight cut (survived)
253: C     13: ----- 14: -----
254: C     15: -----
255: C     16: free-flight
256: C     17: boundary crossing
257: C     18: reflection
258: C     19: fluorescence photons
259: C     20: pair production
260: C     21: -----

```

src/shared/INC/_PTALY0

```

261: C      22: -----
262: C      23: analog absorption
263: C      24: russian roulette killed at TTER
264: C      25: (not used)
265: C      26: time cutoff
266: C      27: particle generation number cutoff
267: C      28: endless loop cutoff (not used. merged to neutron)
268: C##<2007/03/14:PN3:PN4:
269: C##C      29-30: for future use
270: C      29: generated by photonuclear reaction
271: C      30: delayed neutron from photo-fission
272: C      31: -----
273: C      32: -----
274: C      33: -----
275: C      34: for future use
276: C##>
277: C
278: C NCNTR(nevent=30) R8 (GMVP) Event counter (see 'WCNTR')
279: C WCNTR(nevent=30) R8 (GMVP) Sum of weight for various events.
280: C
281: C      1: source      2: fission-s      3: (gamma,n)      4: collision
282: C      5: split (importance or weight window)
283: C      6: prevented splitting      7: leakage      8: energy cut
284: C      9: kill (importance or weight window)
285: C     10: survived (importance or weight window)
286: C     11: Killed (weight cutoff)
287: C     12: Survived (weight cutoff)
288: C     13: fission/photon generation prevented (fixed source)
289: C     14: neutron production/batch (collision estimator)
290: C     15: neutron production/batch (track length estimator)
291: C     16: free-flight
292: C     17: boundary crossing
293: C     18: reflection
294: C     19: neutron loss/batch (collision estimator)
295: C     20: neutron loss/batch (track length estimator)
296: C     21-25 : (unused)
297: C     26: time cutoff
298: C     27: particle generation number cutoff
299: C     28: endless loop cutoff
300: C
301: C*
302: C XAVT(3)      R4 Average of source particle position (x,y,z).
303: C AAVT(3)      R4 Average of direction (a,b,c) of source particles.
304: C EAVT(1)      R4 Average of source particle energy.
305: C*
306: C TFLH(NPKIND) R8 weighted sum of flight-time in batch. <JTIME>
307: C TFLHS(NPKIND,2) R8 Averaged time of flight per history <JTIME>
308: C      (,,1) sum of batch tallies
309: C      (,,2) square sum of batch tallies
310: C c+beff1
311: C LWCBEF(NEBEF) R8 (MVP) Sum of weight for beta-effective calculation.
312: C      <JBEFF>
313: C XBEF(7,NLENG) R8 (MVP) Beta effective tallies of each batch.<JBEFF>
314: C c-beff1
315: C##<2007/03/14:PN3:
316: C
317: C NMPNWGT(MONPNW) R8 Number of monitored reactions      <JPHNU>
318: C WMPNWGT(MONPNW) R8 Sum of monitored reaction weight      <JPHNU>
319: C##>
320: C*=====
321: C*
322: C* ... POINT DETECTOR SPECIFIC DATA ...
323: C*      (moved from gmvp/INC/_PTALY, 18 Nov 1999)
324: C*
325: COMMON /PTDET/ NPDET, NPLEN, LXPDET,

```

```

326:      &      LIPDET,LIPDT2,LJSPDT
327: C*
328: C NPDET      I4 Number of point detector.
329: C NPLEN      I4 Number of XPDET data for each point detector.
330: C XPDET(NPLEN,NPDET) R4 Parameters for point detector.
331: C JSPDT(NPDET) I4 Flag to specify treatment of source particle
332: C      ( input data SPDET )
333: C      -1/0/1 = no contribution/isotropic/....
334: C*
335: C*...ZONE# & LATTICE GEOMETRY PARAMETERS AT DETECTOR POINTS
336: C      IPDET(NPDET,2) I4 Zone # or Lattice buffer zone #
337: C      lattice level number (LEVL)
338: C      IPDT2(NEST,3,NPDET) I4
339: C      (,1,) LZZ : Zone number to return
340: C      (,2,) LPOS: Position number in lattice
341: C      =-1 when STG region
342: C      (,3,) LCRS: Flag to particles whose flight-track may $
343: C      $cross lattice frame <JHLAT>
344: C**
345: C*=====

```

src/shared/INC/_PTLSP

```
1: C*
2: C* --- DATA FOR COMBINATION OF GEOMETRY & TALLY ----
3: C*
4:     COMMON /PTLSP/
5:     &   NNAMES, NLTSPC, NSMSPC,
6:     &   LTNAMS, LKLTSP, LILTSP, LKCLSP,
7:     &   LISPM, LISUSP, LKSPSU, LKTCSP, LIRGSP, LKSUZ,
8:     &   LIPTRG, LLPTRG, LITRNM, LKUNV, LKZUNV, NKTCSP
9: C*
10: C NNAMES I4 Number of "region","frame","subframe" or tally-region names.
11: C NLTSPC I4 Number of combinations of (frame-name, lattice)
12: C     $(frame-name) is a name given to a zone which is $
13: C     $used as the frame of a lattice. <JLTTL>
14: C NSMSPC I4 Total number of data given as "subframe" names of each cell$
15: C     $ in lattices (or universes)..
16: C TNAMS(NNAMES) CH12 List of names$
17: C     $ given as "region","frame","subframe" or tally-region names.
18: C     $( in an order judged by function 'ISTRM' )
19: C KLTSP(2,NLTSPC) I4 List of (frame: lattice) combinations.
20: C
21: C     (1, ) : Frame-name as position in 'TNAMS' list.
22: C     (2, ) : lattice(universe) ID.
23: C*
24: C ILTSP(NLTSPC+1) I4 Pointers to 'KLTSP' data for each $
25: C     $(frame-name: lattice) combination.
26: C KCLSP(ILTSP(NLTSPC+1)-1) I4 "Subframe" name data (position in 'TNAMS'
27: C     $ list) for (frame:lattice) combinations.
28: C ISPM(2,0:NEST,NSPACE) I4 Subspace name list.
29: C
30: C     (1,J,) : Frame name (position in 'TNAMS' list) for $
31: C             $J'th geometry nesting level.
32: C     (2,J,) : Subframe name("given" name) (position in 'TNAMS'
33: C             $ list) for J'th geometry nesting level.
34: C     (1,0,) : Geometry nesting levels for each subspace.
35: C*
36: C ISUSP(NSUZON,NSPACE) I4 Region #'s for each terminal-input-zone $
37: C     $(input-zones in cell and whose material # is non-negative.)
38: C     $ in each subspace.
39: C*
40: C KSPSU(NUNV,0:NSPACE) I4 Pointers to 'KTCSP' data (subspace #'s in $
41: C     $subspaces) for each lattice-frame-zone and subspaces (includes$
42: C     $ space 0).
43: C*
44: C KSUZ(NINPZ,2) I4
45: C     (1:NSUZON,1) : Input-zone # for terminal-input-zones in cell.
46: C     (1:NINPZ,2) : Terminal-input-zone # for each input-zone.
47: C*
48: C KTCSP(NKTCSP) I4 Subspace # list for each cell in lattice-frame-zone$
49: C     $ in subspaces. ( pointed by 'KSPSU' )
50: C NKTCSP I4 Number of data in 'KTCSP' array.
51: C*
52: C IIRGSP(2,NREG) I4
53: C     (1,I) : Subspace # of I'th region.
54: C     (2,I) : "region"-name number in 'TNAMS' table.
55: C*
56: C IPTRG(NTREG+1) I4 Pointer to region=tally-region table 'LPTRG'.
57: C*
58: C LPTRG(*) I4 Region=tally-region table.
59: C*
60: C ITRNM(NTREG) I4 Tally-region name or region number.
61: C     $ For tally-region specified '#TALLYREGION' section in input:
62: C     $ name (position in 'TNAMS' table * (-1) )
63: C     $ other tally-region : region number.
64: C*
65: C KUNV(NINPZ) I4 Lattice-frame-zone's sequential number if an
```

```
66: C           input-zone is a universe(lattice)-frame, else zero.
67: C KZUNV(NZONE) I4 Lattice-frame-zone sequential number if a zone
68: C           belongs to a universe(lattice)-frame, else zero.
69: C
```

src/shared/INC/_PVMPARA

```

1: C/#IF PARA(PVM)
2: C
3: C === This is a copy of the fortran include file of the PVM
4: C
5: C -----
6: C         PVM version 3.4:  Parallel Virtual Machine System
7: C         University of Tennessee, Knoxville TN.
8: C         Oak Ridge National Laboratory, Oak Ridge TN.
9: C         Emory University, Atlanta GA.
10: C    Authors:  J. J. Dongarra, G. E. Fagg, M. Fischer
11: C             G. A. Geist, J. A. Kohl, R. J. Manchek, P. Mucci,
12: C             P. M. Papadopoulos, S. L. Scott, and V. S. Sunderam
13: C             (C) 1997 All Rights Reserved
14: C
15: C         NOTICE
16: C
17: C    Permission to use, copy, modify, and distribute this software and
18: C    its documentation for any purpose and without fee is hereby granted
19: C    provided that the above copyright notice appear in all copies and
20: C    that both the copyright notice and this permission notice appear in
21: C    supporting documentation.
22: C
23: C    Neither the Institutions (Emory University, Oak Ridge National
24: C    Laboratory, and University of Tennessee) nor the Authors make any
25: C    representations about the suitability of this software for any
26: C    purpose.  This software is provided "as is" without express or
27: C    implied warranty.
28: C
29: C    PVM version 3 was funded in part by the U.S. Department of Energy,
30: C    the National Science Foundation and the State of Tennessee.
31: C -----
32: C
33: C -----
34: C    fpvm3.h
35: C
36: C    Definitions to be included with
37: C    User Fortran application
38: C -----
39: C
40: C    integer PVMTASKDEFAULT, PVMTASKHOST, PVMTASKARCH, PVMTASKDEBUG
41: C    integer PVMTASKTRACE, PVMMPFFRONT, PVMHOSTCOMPL, PVMNOSPARENT
42: C    integer PVMHOST, PVMARCH, PVMDEBUG, PVMTRACE
43: C    integer PVMDATADEFAULT, PVMDATARAW, PVMDATAINPLACE
44: C    integer PVMDATATRACE
45: C    integer PVMDEFAULT, PVMRAW, PVMINPLACE
46: C    integer PVMTASKEXIT, PVMHOSTDELETE, PVMHOSTADD, PVMROUTEADD
47: C    integer PVMROUTEDELETE, PVMNOTIFYCANCEL
48: C    integer PVMROUTE, PVMDEBUGMASK, PVMAUTOERR
49: C    integer PVMOUTPUTID, PVMOUTPUTCODE, PVMRESVTIDS
50: C    integer PVMTRACEID, PVMTRACECODE, PVMTRACEBUFFER
51: C    integer PVMTRACEOPTIONS, PVMFRAGSIZE, PVMSHOWTIDS, PVMNORESET
52: C    integer PVMTRACEFULL, PVMTRACETIME, PVMTRACECOUNT
53: C    integer PVMSOUTPUTID, PVMSOUTPUTCODE, PVMSTRACETID
54: C    integer PVMSTRACECODE, PVMSTRACEBUFFER, PVMSTRACEOPTIONS
55: C    integer PVMOUTPUTCTX, PVMTRACECTX, PVMOUTPUTCTCX, PVMSTRACECTX
56: C    integer PVMDONTROUTE, PVMALLOWDIRECT, PVMROUTEDIRECT
57: C    integer PVMPOLLYTYPE, PVMPOLLYTIME, PVMPOLLYCONSTANT, PVMPOLLYSLEEP
58: C    integer PVMBOXDEFAULT, PVMBOXPERSISTENT, PVMBOXMULTIINSTANCE
59: C    integer PVMBOXOVERWRITABLE, PVMBOXFIRSTAVAIL
60: C    integer PVMBOXREADANDDELETE, PVMBOXWAITFORINFO
61: C    string  STRING, BYTE1, INTEGER2, INTEGER4, INTEGER8
62: C    integer REAL4, COMPLEX8, REAL8, COMPLEX16
63: C
64: C    integer PvmOk, PvmSysErr, PvmBadParam, PvmMismatch
65: C    integer PvmNoData, PvmNoHost, PvmNoFile, PvmNoMem

```

```

66: C    integer PvmBadMsg, PvmNoBuf, PvmNoSuchBuf
67: C    integer PvmNullGroup, PvmDupGroup, PvmNoGroup
68: C    integer PvmNotInGroup, PvmNoInst, PvmHostFail, PvmNoParent
69: C    integer PvmNotImpl, PvmDSysErr, PvmBadVersion, PvmOutOfRes
70: C    integer PvmDupHost, PvmCantStart, PvmAlready, PvmNoTask
71: C    integer PvmNoEntry, PvmDupEntry, PvmOverflow, PvmDenied
72: C    integer PvmNotFound, PvmExists, PvmHostrNMstr, PvmParentNotSet
73: C    integer PvmIPLoopback
74: C
75: C -----
76: C    spawn 'flag' options
77: C -----
78: C    parameter( PVMTASKDEFAULT   = 0)
79: C    parameter( PVMTASKHOST     = 1)
80: C    parameter( PVMTASKARCH     = 2)
81: C    parameter( PVMTASKDEBUG    = 4)
82: C    parameter( PVMTASKTRACE    = 8)
83: C    parameter( PVMMPFFRONT     = 16)
84: C    parameter( PVMHOSTCOMPL    = 32)
85: C    parameter( PVMNOSPARENT    = 64)
86: C
87: C -----
88: C    old option names still supported
89: C -----
90: C    parameter( PVMHOST   = 1)
91: C    parameter( PVMARCH   = 2)
92: C    parameter( PVMDEBUG  = 4)
93: C    parameter( PVMTRACE  = 8)
94: C
95: C -----
96: C    buffer 'encoding' options
97: C -----
98: C    parameter( PVMDATADEFAULT = 0)
99: C    parameter( PVMDATARAW    = 1)
100: C    parameter( PVMDATAINPLACE = 2)
101: C    parameter( PVMDATATRACE  = 4)
102: C
103: C -----
104: C    old option names still supported
105: C -----
106: C    parameter( PVMDEFAULT = 0)
107: C    parameter( PVMRAW     = 1)
108: C    parameter( PVMINPLACE = 2)
109: C
110: C -----
111: C    notify 'about' options
112: C -----
113: C    parameter( PVMTASKEXIT   = 1)
114: C    parameter( PVMHOSTDELETE = 2)
115: C    parameter( PVMHOSTADD   = 3)
116: C    parameter( PVMROUTEADD  = 4)
117: C    parameter( PVMROUTEDELETE = 5)
118: C    parameter( PVMNOTIFYCANCEL = 256)
119: C
120: C -----
121: C    packing/unpacking 'what' options
122: C -----
123: C    parameter( STRING   = 0)
124: C    parameter( BYTE1    = 1)
125: C    parameter( INTEGER2 = 2)
126: C    parameter( INTEGER4 = 3)
127: C    parameter( REAL4    = 4)
128: C    parameter( COMPLEX8 = 5)
129: C    parameter( REAL8    = 6)
130: C    parameter( COMPLEX16 = 7)

```

src/shared/INC/_PVMPARA

```
131:     parameter( INTEGER8 = 8)
132:
133: c -----
134: c setopt/getopt options for 'what'
135: c -----
136:     parameter( PVMROUTE           = 1)
137:     parameter( PVMDEBUGMASK       = 2)
138:     parameter( PVMAUTOERR         = 3)
139:     parameter( PVMOUTPUTTID       = 4)
140:     parameter( PVMOUTPUTCODE      = 5)
141:     parameter( PVMTRACETID        = 6)
142:     parameter( PVMTRACECODE       = 7)
143:     parameter( PVMTRACEBUFFER     = 8)
144:     parameter( PVMTRACEOPTIONS    = 9)
145:     parameter( PVMFRAGSIZE        = 10)
146:     parameter( PVMRESVTIDS        = 11)
147:     parameter( PVMOUTPUTTID       = 12)
148:     parameter( PVMOUTPUTCODE      = 13)
149:     parameter( PVMSTRACETID       = 14)
150:     parameter( PVMSTRACECODE      = 15)
151:     parameter( PVMTRACEBUFFER     = 16)
152:     parameter( PVMTRACEOPTIONS    = 17)
153:     parameter( PVMSHOWTIDS        = 18)
154:     parameter( PVMPOLLTYPE        = 19)
155:     parameter( PVMPOLLTIME        = 20)
156:     parameter( PVMOUTPUTCTX       = 21)
157:     parameter( PVMTRACECTX        = 22)
158:     parameter( PVMOUTPUTCTX       = 23)
159:     parameter( PVMSTRACECTX       = 24)
160:     parameter( PVMNORESET         = 25)
161:
162: c -----
163: c tracing option values for setopt function
164: c -----
165:     parameter( PVMTRACEFULL       = 1)
166:     parameter( PVMTRACETIME       = 2)
167:     parameter( PVMTRACECOUNT     = 3)
168:
169: c -----
170: c poll type options for 'how' in setopt function
171: c -----
172:     parameter( PVMPOLLCONSTANT = 1)
173:     parameter( PVMPOLLSLEEP   = 2)
174:
175: c -----
176: c for message mailbox operations
177: c -----
178:     parameter( PVMBOXDEFAULT      = 0)
179:     parameter( PVMBOXPERSISTENT  = 1)
180:     parameter( PVMBOXMULTIINSTANCE = 2)
181:     parameter( PVMBOXOVERWRITABLE = 4)
182:     parameter( PVMBOXFIRSTAVAIL   = 8)
183:     parameter( PVMBOXREADANDDELETE = 16)
184:     parameter( PVMBOXWAITFORINFO  = 32)
185:
186: c -----
187: c routing options for 'how' in setopt function
188: c -----
189:     parameter( PVMONTROUTE      = 1)
190:     parameter( PVMALLOWDIRECT= 2)
191:     parameter( PVMROUTEDIRECT= 3)
192:
193: c -----
194: c error 'info' return values
195: c -----
196:     parameter( PvmOk           = 0)
197:     parameter( PvmBadParam     = -2)
198:     parameter( PvmMismatch     = -3)
199:     parameter( PvmOverflow     = -4)
200:     parameter( PvmNoData      = -5)
201:     parameter( PvmNoHost      = -6)
202:     parameter( PvmNoFile      = -7)
203:     parameter( PvmDenied      = -8)
204:     parameter( PvmNoMem       = -10)
205:     parameter( PvmBadMsg      = -12)
206:     parameter( PvmSysErr      = -14)
207:     parameter( PvmNoBuf       = -15)
208:     parameter( PvmNoSuchBuf   = -16)
209:     parameter( PvmNullGroup   = -17)
210:     parameter( PvmDupGroup    = -18)
211:     parameter( PvmNoGroup     = -19)
212:     parameter( PvmNotInGroup  = -20)
213:     parameter( PvmNoInst      = -21)
214:     parameter( PvmHostFail    = -22)
215:     parameter( PvmNoParent    = -23)
216:     parameter( PvmNotImpl     = -24)
217:     parameter( PvmDSysErr     = -25)
218:     parameter( PvmBadVersion  = -26)
219:     parameter( PvmOutOfRes    = -27)
220:     parameter( PvmDupHost     = -28)
221:     parameter( PvmCantStart   = -29)
222:     parameter( PvmAlready     = -30)
223:     parameter( PvmNoTask      = -31)
224:     parameter( PvmNotFound    = -32)
225:     parameter( PvmExists      = -33)
226:     parameter( PvmHostNMstr   = -34)
227:     parameter( PvmParentNotSet = -35)
228:     parameter( PvmIPLoopback  = -36)
229:
230: c -----
231: c these are going away in the next version.
232: c use the replacements
233: c -----
234:     parameter( PvmNoEntry      = -32)
235:     parameter( PvmDupEntry     = -33)
236:
237:
238: c Uncomment this include for use with the WIN32 WATCOM fortran compiler
239: c
240: c     include ' ../include/fpvm3_watcom.h'
241: c
242: C/#ENDIF
```


src/shared/INC/_PVRED

```
1: C*
2: C*.... Variance reduction array pointers
3: C*
4:      COMMON /PVRED/  LXIMP, LWKIL, LWSRV, LWTIME, LPSXYZ, LPSALP
5: C*
6: C XIMP(NGROUP,NREG)  R4  Importances of each region. <JIMPT>
7: C WKIL(NGROUP,NREG)  R4  Weight below which russian roulette occurs.
8: C WSRV(NGROUP,NREG)  R4  Weight given to survived particles in$
9: C                      $ russian roulette game.
10: C*
11: C WTIME(NTIME)      R4  importance factor in time
12: C PSXYZ(3)          R8  target point coordinates for PATH-STRETCHING
13: C PSALP(NREG)       R4  PATH-STRETCHING factors by region.
14: C*
15: C*
16:      real  WLLIM
17:      COMMON /PVREDF/ WLLIM
18: C*
19: C WLLIM R4  lower limit of particle weight for variance reduction
20: C            and secondary particle generation.
21: C*
```

src/shared/INC/_RAND

```
1: C
2: C ... seed & number of RNs in RANU2
3: C
4: C =<memo>= The seed value is saved as common variables
5: C           for a restart calculation. This treatment will be
6: C           modified in the future.
7: C
8: C           integer IRNSU, IRNSM, IRNSL
9: C           real*8  RNCTR
10: C
11: C/#IF PARA( SX* )
12: C * local common /CRANU2/
13: C/#ELSEIF PARA( CRAY* )
14: C * task common /CRANU2/
15: C/#ELSE
16: C * common /CRANU2/
17: C/#ENDIF
18: C & RNCTR, IRNSU, IRNSM, IRNSL
```

src/shared/INC/_SFLATT

```
1: C
2: C -----
3: C   Statement function to check lattice zone from material #
4: C   (KZMAT(1,*) --> KKZZMM )
5: C -----
6: C
7: C   Data type declaration for ISLATT and LATTNM and
8: C   parameter definition of MLTOFF is placed separatly in _PMLATT
9: C   to clear statement order standard of Fortran.
10: C
11: C -----
12: C
13: CC   logical ISLATT
14: CC   integer LATTNM
15: C
16: C=====
17: C   .... material in a zone indicates lattice farame zone ? ...
18: C
19: *CC   ISLATT(KKZZMM) = KKZZMM.le.-1.and.KKZZMM.ge.-998
20: C
21: C   .... lattice # from "material #" ...
22: C
23: *CC   LATTNM(KKZZMM) = -KKZZMM
24: C
25: C=====
26: C
27: *   parameter( MLTOFF = 0 )
28: *   parameter( MLTMIN = -998)
29: C
30: C   .... if MLTOFF = 0, needs lower boundary check
31: C
32: *   ISLATT(KKZZMM) = KKZZMM.lt.MLTOFF.and.KKZZMM.ge.MLTMIN
33: C
34: C   .... lattice # or ID from cell ...
35: C
36: *   LATTNM(KKZZMM) = MLTOFF-KKZZMM
37: C
38: C=====
39: C
40: CC   parameter( MLTOFF = -10000 )
41: C
42: C   ==== when "material #" of lattice frame zone is defined lower than
43: C   MLTOFF.
44: C
45: C   .... material in a zone indicates lattice farame zone ? ...
46: C
47: C   .... if MLTOFF < 0, no need for lower boundary check
48: C   ISLATT(KKZZMM) = KKZZMM.lt.MLTOFF
49: C
50: C   .... lattice # or ID from cell ...
51: C
52: C   LATTNM(KKZZMM) = MLTOFF-KKZZMM
53: C
54: C -----
55: C
```

src/shared/INC/_SIZES

```

1: C*
2: C*... SIZE & LIMIT PARAMETERS or counters
3: C* ( This common includes variables which do not indicate any size.
4: C* The common area name 'SIZES' is only by historical reason.)
5: C*
6: C* modified 7 June 1995:
7: C* modified 15 Nov 1995:
8: C* modified 4 Oct 1995: add NBPINT
9: C* modified 4 Aug 1997: add NRSINT
10: C* modified Dec 1998: add NPNND
11: C* modified 12 Jul 1999: add NBZL
12: C* modified 4 Feb 2000: add NHSUB
13: C* modified 13 Aug 2003: add NPPPF
14: C*
15: common /SIZES/ NPKIND,
16: & NGROUP, NZONE, NINPZ, NSURF,
17: & NSDA, NZDA, NMEMO, NMAT, NREG, NTIME,
18: & NSOUR, NRESP, NSTAL,
19: & NSKIP, NPICT, NREFS, NMEMS, NMEMOP,
20: & NLATT, NLEV, NCELL, NEST, NLBZ, NRSKIP, NLENG
21: & NGP1, NGP2, NSPACE, NUNV, NSUZON, NCREG, NTREG,
22: & NBODY, NBZL, NPNND, NPPPF,
23: & NRESP2, NSTAL2, NBPINT, NRSINT, NTMINT
24: C
25: C ... task shared variables to save input values for task local
26: C variables before/after multi tasking phase
27: C
28: common /SSIZES/
29: & NPART0, NTHIST0,
30: & NHIST0, NHSUB0, NBANK0, NBNK20, IMPMX0, IRAND0
31: C/#IF INTEGER8
32: * integer*8 NPART0, NTHIST0
33: C/#ELSE
34: integer NPART0, NTHIST0
35: C/#ENDIF
36: C
37: C
38: C ... task local scalar variables ...
39: C
40: C/#IF PARA(SX*)
41: * local common /LSIZES/
42: C/#ELSEIF PARA(CRAY)
43: * task common /LSIZES/
44: C/#ELSE
45: common /LSIZES/
46: C/#ENDIF
47: & NPART, NTHIST,
48: & NHIST, NHSUB, NBANK, NBNK2, IMPMAX,
49: & NFBANK, NFBNK0, NBATCH, IRAND
50: C/#IF INTEGER8
51: * integer*8 NPART, NTHIST
52: C/#ELSE
53: integer NPART, NTHIST
54: C/#ENDIF
55: C
56: common /IPARAM/ MXPGEN
57: common /RPARAM/ TCPU, ECUT, TCUT, SUPPLY, ELOOP
58: real TCPU, ECUT, TCUT, SUPPLY, ELOOP
59: C*
60: C NPKIND I4 Number of particle species in calculation.
61: C ( --> JNEUT + JPHOT )
62: C NPART I4/I8 Total number of particles (histories) in problem.
63: C ( includes previous histories of calculations in successive
64: C calculations by restart option. )
65: C negative input means running more '-NPART' histories

```

```

66: C in restart case. (in NO-RESTART case, abs(NPART) -> NPART)
67: C NHIST I4/I8 Number of particles (histories) run per batch.
68: C*** add 4 Feb 2000 ***
69: C NHSUB I4 Number of particles (histories) run per sub-batch.
70: C "Sub-batch" is a layer of history control of MVP/GMVP which
71: C compose a batch by processing NHSUB histories which is
72: C smaller than batch history size NHIST.
73: C*
74: C NBANK I4 particle bank length.
75: C*** add 14 Oct 1996 ***
76: C NBNK2 I4 working particle bank size use in "underground" level
77: C modules ( ABS2ZN routine etc.). Arrays of this level
78: C should not overlap with banks, working arrays etc. in
79: C routines directly called in ACTION routine.
80: C*
81: C NBPINT I4 interval to print intermediate monitoring results
82: C (K-eff, averaged source attributes etc.)
83: C If positive, monitor of every NBPINT'th batch is printed.
84: C If negative, print monitor batches including every
85: C abs(NBPINT)'th history. Default is 1.
86: C*
87: C NRSINT I4 Interval to output restart file.
88: C If positive, output after every NBPINT'th batch.
89: C If negative, output after batches including every
90: C abs(NBPINT)'th history.
91: C
92: C Output only after the last batch when zero.
93: C Non zero value is effective only in single task.
94: C Default is 0.
95: C*
96: C NTMINT I4 Interval to printout tally monitor.
97: C If positive, output tallies (K-eff or specified tally)
98: C after every NTMINT'th batch.
99: C Default is 10.
100: C*
101: C NGROUP I4
102: C For MVP,
103: C Sum of number of "energy group" for all particle species.
104: C This parameter is a
105: C number of energy bins for tallies or parameters
106: C which need an energy separation, and no relationship to
107: C the continuous cross section data itself.
108: C For GMVP,
109: C Total number of energy groups to be analyzed.
110: C It must be less than or equal to the number of energy group
111: C in cross section library.
112: C NZONE I4 Number of zones.
113: C NINPZ I4 Number of input zones.
114: C NBODY I4 Number of bodies.
115: C NBZL I4 Total number of body * sign input used to define zones
116: C NSURF I4 Total number of surfaces in geometry.
117: C NSDA I4 Length of surface data ('SDA' array)
118: C NZDA I4 Total number of (internal) zone-specification data
119: C NMEMO I4 Length of memorandum array (maximum number of memorized $
120: C $ next zones for each zone.)
121: C NMAT I4 Number of materials.
122: C NREG I4 Number of regions. "REGION"'s are defined as combination $
123: C $ of zones, and tallies for each batch are taken for "REGIONS"$
124: C $'s and variance-reduction parameters are defined also for $
125: C $"REGION"'s.
126: C Final output of tallies are done for "TALLY-REGION"'s$
127: C $ which can be combinations of 'REGION"'s.
128: C*
129: C NTIME I4 Number of time bins for tally <JTIME>
130: C***** NSECT I4 SECTOR NUMBER OF ZONE-DEPENDENT STACK

```

src/shared/INC/_SIZES

```
131: C NSOUR I4 Number of particle source sets. (not the number of source
132: C particles)
133: C NRESP I4 Number of response functions ( group-wise).
134: C NSTAL I4 Number of response functions ( continuous energy ).
135: C NRESP2 I4 Number of response functions (group-wise) used in STALY.
136: C NSTAL2 I4 Number of response functions (continuous energy ) used $
137: C $ in special tally.
138: C TCPU R4 Total cpu time allowed. (minuite)
139: C IRAND I4 Initial random number.
140: C NTHIST I4 Number of particles (histories) run so far in successive$
141: C $ runs. (counted by the code and stop calculation if
142: C NTHIST >= NPART).
143: C ECUT R4 Cutoff energy (eV) (lower energy boundary of calculation)
144: C TCUT R4 Cutoff time (sec)
145: C*SUPPLY R4 FACTOR OF SOURCE SUPPLY POINT ( 0.0 .GE. SUPPLY .LT.1.0 )
146: C*
147: C ELOOP R4 A threshold to terminate
148: C histories of "endless-loop" particles.
149: C
150: C $ SUM OF PROCESSED-PARTICLES IN DETECTION MODE
151: C $ When ----- > ELOOP
152: C $ SUM OF PROCESSED-PARTICLES IN THE BATCH
153: C NFBANK I4 Length of fission neutron bank in eigenvalue calculation.
154: C (actually, max size of stored data in fission bank)
155: C NFBANK0 I4 Actual length of fission neutron bank arrays.
156: C NSKIP I4 Number of batches skipped for tally $
157: C $in eigenvalue calculation.
158: C NBATCH I4 Number of batches run so far.
159: C NPICT I4 Number of pictures drawn <JPICT>
160: C NREFS I4 Number of reflection surfaces <JREFS>
161: C NMEMS I4 Maximum number of starting zones memorized for each$
162: C $ particle source set.
163: C NMEMOP I4 Optimize kmemo array while NBATCH .le. NMEMOP (default=2)
164: C*
165: C***** LATTICE RELATED SIZES < JLATT > *****
166: C*
167: C NLATT I4 Number of lattice.
168: C*NLLEV I4 Maximum geometry nesting level number.
169: C NCELL I4 Number of (lattice) cells.
170: C NEST I4 Maximum nest level in lattice geometry case.
171: C NLBZ I4 Number of lattice-buffer zones (in lattice cell and
172: C whose material is -999).
173: C
174: C NRSKIP I4 Number of random number skipped.
175: C < 0 : before each batch > 0 : before first batch only
176: C
177: C NLENG I4 Total number of batches in one or successive runs.
178: C (includes batches of the current run. To be set internally)
179: C*
180: C***** FOR PHOTON PROBLEMS ***
181: C*
182: C NGP1 I4 For MVP,
183: C Number of neutron energy groups (tally & source)
184: C For GMVP,
185: C Energy group number of primary particle to be analyzed.
186: C NGP2 I4 For MVP,
187: C Number of photon energy groups (tally & source)
188: C For GMVP,
189: C Energy group number of secondary particle to be analyzed.
190: C*
191: C***** FOR LATTICE OR FRAME DEPENDENT TALLY ****
192: C* (terminology "SUB-UNIVERSE" "SPACE" "SPACE-FAMILY" etc. are
193: C* replaced with "FRAME" "SUBFRAME" "SUBSPACE" etc in "JAERI data/code
194: C* 94-007" ( 7 Jun 1995 M.Sasaki )
195: C*
196: C NSPACE I4 Number of "subspace" defined to take tally separately for$
197: C $ zones in lattices or universes. <JLTTL>
198: C NUNV I4 Number of "frame"-input-zones which include geometric
199: C $ sub-structures such as lattice). <JLTTL>
200: C NSUZON I4 Number of "terminal"-zones which is defined as input-zones
201: C in lattice-cell and for which tallies can be taken (mat ID is
202: C non-negative) as regions in subspaces. <JLTTL>
203: C NCREG I4 Number of tally-regions specified in '#TALLYREGION'
204: C section of input data.
205: C NTREG I4 Total number of tally-regions.
206: C*** added Nov 1998
207: C NPNND I4 size of PNND array (nearest neighbour distribution for
208: C STG-region).
209: C*** added Aug 2003
210: C NPPPF I4 size of PPPF array (partial packing fractions in
211: C STG-region).
212: C*
213: C***** generation control for fixed source problem
214: C*
215: C*** added 23 June 1999 ***
216: C MXPGEN I4 particle generation cutoff used to prevent birth of
217: C too many children by fission and (n,gamma) reactions
218: C especially for fixed source problems with the RELATIVE-WEIGHT
219: C option.
220: C
221: C When MXPGEN > 0, birth of particles whose generation number
222: C from source particle is greater than MXPGEN is simply prevented.
223: C
224: C Ex. MXPGEN=2 allows birth of secondary neutron or photon,
225: C but doesn't allow tertiary particles.
226: C
```

src/shared/INC/_STALY

```

1: C*
2: C* ... data and array pointers related to "special" tally
3: C*
4:      common /STALLY/ NSTALY, NETALY, NDTALY, NIETAL, NIDTAL, NTEVE,
5:      &      LETALY, LIETAL, LDTALY, LIDTAL, LITBIN, LJTEVE, LKTEVE,
6:      &      LLTEVE, LJDTRG, LKDTAL, LKETAL,
7:      &      NLETAL, NLDTAL, NEDTMX, LRNM,  NBINMX,
8:      &      NETRV, METRV, LIETRV, NBETRV, LETRV,
9:      &      NDTWMX, MXRGN, LJFUSD,
10:     &      KPNPRD ! photo-nuc
11: C*
12: C NSTALY I4 Total number of "special" tallies. A "special" tally$
13: C $ is a tally specified in "$TALLY" section of input data$
14: C $ and taken separately from default tallies such as neutron$
15: C $ multiplication factor, region/group-wise flux etc. A special$
16: C $ tally can be time or angle dependent.
17: C $ A "special" tally is composed of one or more than one$
18: C $ "edited" tallies$
19: C $ which are combinations of$
20: C $ values tallied as the "special" tally in energy bins, time bins$
21: C $ etc. A "direct" tally$
22: C $ is a tally calculated on specified events in random walk process$
23: C $ and$
24: C $ "edited" tallies are calculated after each batch as combination$
25: C $ of "direct" tallies.
26: C
27: C NDTALY I4 Total number of "direct" tallies in all "edited" tallies.
28: C
29: C*
30: C DTALY(*) R8 Body of "direct" tallies. Positions and structures of each
31: C "direct" tally is stored in IDTAL array.
32: C NLDTAL I4 length of DTALY(*) array.
33: C
34: C IDTAL(NIDTAL) I4 Information of "direct" tallies;
35: C $ "Direct tallies" are tallies taken directly for random walk
36: C $ events, and "edited" tallies are composed from direct tallies
37: C $ and output as calculation results.
38: C $ Each direct tally has the following information;
39: C $ (array index is for each tally)
40: C $ IDTAL(1) : ID of "direct" tally (arbitrary)
41: C $ IDTAL(2) : garbage for debugging ( fixed to -999 )
42: C $ IDTAL(3) : starting address index of this direct tally
43: C $ in DTALY(*).
44: C $ IDTAL(4) : event # (see description of NTEVE)
45: C $ IDTAL(5) : particle type
46: C $ 1/2/3 = neutron/photon/...
47: C $ IDTAL(6) : number of dimensions of this direct tally.
48: C $ A direct tally is stored in memory as a matrix
49: C $ of IDTAL(6)-dimension.
50: C $ IDTAL(7) : weighting function specification.
51: C $ 0 = no weighting function is necessary.
52: C $ That means simple flux or current tally, and
53: C $ weights or track lengths multiplied by weight are
54: C $ tallied.
55: C $ 1000> N>1 : other special factor (not implemented now)
56: C $ -N : use N'th groupwise response function.
57: C $ 1001 : use microscopic cross section in SMIC (MVP)
58: C $ (calculate number density weighted rate)
59: C $ 1002 : use microscopic cross section in SMIC (MVP)
60: C $ ("spread" mode -> calculate per atom rate everywhere)
61: C $ 1003 : use microscopic photonuclear cross section in SMIC ! photo-nu
62: C $ (MVP) (calculate number density weighted rate) ! photo-nu
63: C $ 1004 : use microscopic photonuclear cross section in SMIC ! photo-nu
64: C $ (MVP) ("spread" mode -> calculate per atom rate ! photo-nu
65: C $ everywhere) ! photo-nu
66: C $ 2000 : (MVP) weighting function is from
67: C $ the special cross section data bank SGTAL(i,*).
68: C $ 3000 : use macroscopic cross section in SMAC (MVP/GMVP)
69: C $ 9001 : analog-event of scattering (MVP)
70: C $ 9002 : analog-event of capture (MVP)
71: C $
72: C $ IDTAL(8) : When IDTAL(7)=1001,1002 or 3000
73: C $
74: C $
75: C $ When SMIC is used as weighting factor.
76: C $ IDTAL(8) = 1024*m + n (n < 1024)
77: C $ Use SMIC(i,m,LMIC(n)) as factor ( m = nuc # )
78: C $
79: C $ When SMAC is used as weighting factor.
80: C $ IDTAL(8) = n
81: C $ Use SMAC(i,m,LMAC(n)) as factor ( m = material # )
82: C $
83: C $ When IDTAL(7) = 2000 , SGTAL(i,IDTAL(8))
84: C $
85: C $ IDTAL(9) : sequece # of this direct tally
86: C $ IDTAL(10) : custom tally # (non-zero means user customized
87: C $ tally treatment. Good boys/girls might not do
88: C $ such a dirty work ... but sometimes ...)
89: C $
90: C $ IDTAL(11) : detector # for point(next event) or surface tally
91: C $ When tally is surface tally, positive IDTAL(11) means
92: C $ tally for "particle current" and negative IDTAL(11)
93: C $ means "flux" tally.
94: C $
95: C $ IDTAL(12) to IDTAL(IDTOFF) : <unused>
96: C $
97: C $ The following data are prepared for each dimension of tally;
98: C $ ( "in" is offset from IDTAL(IDTOFF). zero for the first
99: C $ dimension). IDTOFF is a constant to specify length of
100: C $ "header" part of IDTAL for each tally
101: C $
102: C $ IDTAL(IDTOFF+in+1) : type of dimension:
103: C $ 0 : (unused)
104: C $ 1 : region
105: C $ 2 : energy
106: C $ 3 : time
107: C $ 4 : angle
108: C $ 5 : generation
109: C $ 6 : source set
110: C $ 7 : source-region
111: C $ 8 : marker-region
112: C $ 9 : spatial point
113: C $ 10 : produce-region ! photo-nuc
114: C $ 11 : produce-nuclide ! photo-nuc
115: C $ 12 : produce-reaction ! photo-nuc
116: C $ IDTAL(IDTOFF+in+2) : bin # of this dimension
117: C $ IDTAL(IDTOFF+in+3) : size of a table used for common bin # to
118: C $ tally bin # conversion.
119: C $ The "common bin #" is an integer particle attribute
120: C $ given independently of tallies or set for "basic" tallies
121: C $ such as energy group # in GMVP or region #.
122: C $ The time-dependent feature is possible only for
123: C $ special tallies, but a "common time bin set" is
124: C $ created by merging time bin sets of all direct-tallies,
125: C $ and "common bin #" is calculated independently of

```

src/shared/INC/_STALY

```

126: C $      direct tally events in time dependent problems.
127: C $
128: C $      The "energy group" in MVP has meanings only for
129: C $      tallies and direct tallies can have different energy
130: C $      divisions from the one for "basic" tallies, but such a
131: C $      possibility is reserved for future version and energy
132: C $      bin # of direct tallies in the current version is limited
133: C $      to combinations of energy bins of basic tallies.
134: C $
135: C $      Angle bins are probably highly dependent on each tally,
136: C $      so it is assumed that no "common angle bin #" is calculated.
137: C $      In such a case, this value is a negative number and its
138: C $      absolute value is pointer index to direct-tally
139: C $      dependent bin table (ITBIN(*)).
140: C $
141: C $      IDTAL(IDTOFF+in+4:IDTOFF+in+3+IDTAL(IDTOFF+in+3)) :
142: C $      Conversion table from common bin # to tally bin # of direct
143: C $      tally. ( IDTAL(IDTOFF+in+3) > 0 )
144: C $
145: C $      Ex. If there are 10 common energy bins and
146: C $      IDTAL(IDTOFF+in+4:IDTOFF+in+13) are given as;
147: C $      1 2 2 2 3 3 4 4 0 0
148: C $      common energy group 1 is bin # 1 of this d-tally,
149: C $      common energy groups 2, 3 and 4 are bin # 2,
150: C $      common energy groups 5 and 6 are bin # 3,
151: C $      common energy groups 7 and 8 are bin # 4,
152: C $      and common energy groups 9 and 10 are not tallied.
153: C $
154: C $
155: C $ ITBIN(*) I4 Tally bin information for direct tallies which does not
156: C $ use "common bin #" to tally bin # conversion table in IDTAL(*).
157: C $ This data may be used as real or double precision form using
158: C $ equivalent variable names RTBIN(*) <real> and DTBIN(*) <double>.
159: C $
160: C $ NIDTAL I4 Size of IDTAL array.
161: C $
162: C $ KDTAL(NDTALY+1) I4 starting position of each direct-tally in "DTALY"
163: C $ array. ( DTALY( KDTAL(i) : KDTAL(i+1)-1 ) is area for i'th tally).
164: C *
165: C * ... data for tally events
166: C *
167: C $ NTEVE I4 Number of Events to take "direct" tallies as "special" tally.
168: C $ ( "tally events" ) defined in TALINP
169: C $ Event #'s currently defined are as follows;
170: C $ 1 : Particle generation (source or gamma-generation or
171: C $      next event tally)
172: C $ 2 : Free flight
173: C $ 3 : Boundary/surface crossing
174: C $ 4 - 6 : (spare for future)
175: C $ 7 : leakage
176: C $ 8 : analog events of neutron for noise analysis
177: C $ 9 : analog events of photon for noise analysis
178: C $ 10 : neutron collision
179: C $ 11 : photon collision
180: C $ 12 : electron collision
181: C $
182: C $ The value of NTEVE is 12 in the current version.
183: C $
184: C $ JTEVE(NTEVE) I4 Number of "direct" tallies to be calculated on each
185: C $ tally event.
186: C $ KTEVE(*) I4 Pointers to IDTAL(*) array for each "direct" tally
187: C $ every tally event.
188: C $ This array is pointed by LTEVE(1:NTEVE) every tally event.
189: C $ LTEVE(NTEVE) I4 Pointers to KTEVE(*) array for each tally event.
190: C $ KTEVE(LTEVE(i):LTEVE(i)+JTEVE(i)-1) are pointers to IDTAL(*)

```

```

191: C      array for i'th tally event.
192: C $ JDTRG(NREG,NDTALY) I4 Flags to show that a "direct" tally
193: C $ should be taken in a region. ( for shortcut in tally procedure )
194: C $
195: C *****
196: C $
197: C $ NETALY I4 Total number of "edited" tallies in all "special" tallies.
198: C $ ETALY(*) R8 Body of "edited" tallies. Positions and structures of each
199: C $ "edited" tally is stored in IETAL data package.
200: C $ NLETAL I4 length of ETALY(*) array. Actual length is NLETAL*2.
201: C $ KETAL(NETALY+1) I4 starting position of each edited-tally in "ETALY"
202: C $ array. ( ETALY( KETAL(i) : KETAL(i+1)-1 ) is area for i'th tally).
203: C $ NEDTMX I4 maximum length of edited tally.
204: C $ RNM(NBINMX) C128 array to store character tally bin name (curently
205: C $      tally-region) for tally printout.
206: C $ NBINMX I4 maximum number of bins.
207: C $
208: C $ IETAL(NIETAL) I4 Information of "edited" tallies
209: C $ "Direct tallies" are tallies taken directly for random walk
210: C $ events, and "edited" tallies are compsed from direct tallies
211: C $ and output as calcualtion results.
212: C $ (in data-packet-container form handled with "PACKET" subrouinte
213: C $ group).
214: C $
215: C $ << Structure of IETAL(*) >>
216: C $
217: C $ +----- edited tally loop (NETALY)
218: C $ | label: TALLY.# ( '#' is edited tally # increasing from unity)
219: C $ | integer:
220: C $ | (1) tally ID
221: C $ | (2) particle type
222: C $ | (3) method to synthesize direct-tallies
223: C $ | 0 = no combination
224: C $ | 1 = weighted sum
225: C $ | 2 = multiplication / division
226: C $ | 3 = general expression
227: C $ | 4 = Feynman-Y values
228: C $ | (4) number of tally dimensions
229: C $ | (5) starting address of this tally in ETALY(*)
230: C $ | (6) event #
231: C $ | (7) weighting function
232: C $ | (8) nuclide/atom # & reaction #
233: C $ | (9) <unused>
234: C $ | (10) custom tally # (user customized treatment if non zero)
235: C $ | string: label string
236: C $ | +----- tally-dimension loop
237: C $ | | string: "dimension-label" ('ENERGY', 'TIME' etc.)
238: C $ | +-----
239: C $ | label: DTALLY.#
240: C $ | integer: (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[1],
241: C $ | (d-tally#, pointer-to-IDTAL,pointer-to-DTALY)[2],
242: C $ | ...
243: C $ | ( data to synthesize d-tallies : not available currently )
244: C $ |
245: C $ | label: DIMENSION.#
246: C $ | +----- tally-dimension loop
247: C $ | | integer: size of each dimension(number of bins) in ETALY
248: C $ | | REGION ---> negative, others ---> positive
249: C $ | | integer:
250: C $ | | .....E-tally-bin loop (in one packet)
251: C $ | | . table length,
252: C $ | | . D-tally -> E-tally translation table.
253: C $ | | . (list of d-tally bins composing this e-tally bin)
254: C $ | | .....
255: C $ | | integer:

```

src/shared/INC/_STALY

```
256: C $ | | .....E-tally-bin loop (in one packet)
257: C $ | | . table length,
258: C $ | | . original-tally bin -> E-tally bin translation table.
259: C $ | | . (list of original-tally-bins composing this e-tally bin)
260: C $ | | .....
261: C $ | | +----- E-tally-bin loop (when this dimension is REGION)
262: C $ | | string: "region name"
263: C $ | | +-----
264: C $ | +-----
265: C $ +-----
266: C*
267: C NIETAL I4 Size of IETAL array.
268: C*
269: C***** data for real variance estimation (added 30 Sep 1998) *****
270: C*
271: C NETRV I4 Number of special (edited) tallies whose "real variance"'s
272: C are estimated. Tallies of batches from batch NSKIP+1 are
273: C stored on array ETRV(*).
274: C METRV I4 Total length of special (edited) tallies whose
275: C "real variance"'s are estimated. (per batch length)
276: C IETRV(2,NETRV+1) I4 Pointers for special (edited) tallies whose
277: C "real variance"'s are estimated.
278: C*
279: C $ IETRV(1,*) : edited tally #.
280: C $ IETRV(2,*) : tally is stored on
281: C $ ETRV(IETRV(2,i):IETRV(2,i+1)-1,batch).
282: C*
283: C NBETRV I4 Number of batches for which special (edited) tallies
284: C whose "real variance"'s are estimated.
285: C This should be <tatal # of batches>-NSKIP, and
286: C increases for successive restart runs.
287: C ETRV(METRV,NBETRV+1) R8
288: C Special (edited) tallies whose "real variance"'s are estimated.
289: C $ METRV tally data are alloceted for each batch
290: C (from NSKIP+1'th batch to the last batch).
291: C $ Elements ETRV(*,NBETRV+1) are used to store estimated errors etc.
292: C*
293: C NDTWMX I4 maximum size of work area to process d-tally to e-tally
294: C except NEDTMX
295: C*
296: C MXRGBN I4 maximum number of region bin over special tallies.
297: C*
298: C JPUSD(NPDET) I4 flags to show point detector used.
299: C*
300: C KPNPRD I4 flag for particle production by PRODUCE-* dimensions. ! photo-nuc
301: C $ Flag #'s currently defined are as follows; ! photo-nuc
302: C $ 0 : no restriction ! photo-nuc
303: C $ 1 : restriction to photo-nuclear reaction [PNPRODUCE] ! photo-nuc
304: C*
```


src/shared/INC/_TASKDT

```

1: C*
2: C*
3: C*   TASK PARAMETERS EFFECTIVE IN MULTI TASKING
4: C*
5: C*
6: C*
7: C* ... Maximum allowable tasks ...
8: C*
9:       parameter ( MAXTSK = 512 )
10: C*
11:       common /TASKDT/ NHHEAP,
12:       & MNTASK, LITASK, ITASK0, LIRANT,
13:       & ILOCK(3), IBARR(2), IPWEV(MAXTSK),
14:       & JRPCPU, LIRSUT, LIRSMT, LIRSLT
15: C/IF INTEGER8
16: *     integer*8 NHHEAP
17: C/ELSE
18:     integer NHHEAP
19: C/ENDIF
20: C/IF PARA( CRAY )
21: *     PARAMETER ( MNTASK = 3 )
22: C/ELSE
23:     PARAMETER ( MNTASK = 2 )
24: C/ENDIF
25: C**** common /TASKDT/ ***
26: C NTASK I4  NUMBER OF TASKS IN MULTIPROCESSING MODE.
27: C ITASK(MNTASK,NTASK) I4  TASK IDENTIFICATION VALUE ETC.
28: C
29: C   For Cray Multitasking Library used as "task array"
30: C     MNTASK = 3
31: C     ITASK(1,n) : length of task array
32: C     ITASK(2,n) : task id given by library
33: C     ITASK(3,n) : user defined task information
34: C
35: C   For SX3 Multitasking Library used as:
36: C     ITASK(1,n) : task value given by library
37: C     ITASK(2,n) : user defined task information
38: C
39: C ITASK0 I4  Task identification value of a task that remains
40: C   active after 'ACTION' tasks are finished and performs tally post-
41: C   processing & output. (ie. Task ID of Parent task.)
42: C
43: C IRANT(NTASK) I4 Seed of random number given to each task.
44: C
45: C ILOCK(2) I4   "Lock" to control part of code that should be performed
46: C   exclusively in each task. (CRAY, SX3 )
47: C
48: C     ILOCK(1) : lock to take summation of tally of all task.
49: C     ILOCK(2) ; lock to print message in parallel processing.
50: C
51: C IBARR(2) I4 "Barrier" data to control barrier synchronization.
52: C
53: C     IBARR(1) : barrier to take summation of tally of all task.
54: C     IBARR(2) : unused
55: C
56: C IPWEV(NTASK) I4 "Event" data to control POST/EVENT synchronization.
57: C
58: C NHHEAP I4 number of histories remainnig in "history heap" that is
59: C   used yo control multitasking in 'NO-REPRODUCIBLE-RUN" on
60: C   shared memory machines.
61: C
62: C JRPCPU I4 If non zero, histories are assigned proportionally to
63: C   relative CPU speed of each parallel task.
64: C   (effective only in PVM case)
65: C

```

```

66: C IRSUT(NTASK) I4 Upper 21 bits of the seed of 63-bit random number
67: C   given to each task.
68: C IRSMT(NTASK) I4 Middle 21 bits of the seed of 63-bit random number
69: C   given to each task.
70: C IRLT(NTASK) I4 Lower 21 bits of the seed of 63-bit random number
71: C   given to each task.
72: C
73: C
74: C/IF PARA( SX* )
75: *     local common /LTSKDT/ IDTASK
76: C/ELSEIF PARA( CRAY )
77: *     task common /LTSKDT/ IDTASK
78: C/ELSE
79:     common /LTSKDT/ IDTASK
80: C/ENDIF
81: C
82: C*** COMMON /LTSKDT/
83: C
84: C IDTASK I4 Task identification number in multitasking mode.
85: C   (task local data for shared memory machines.)
86: C
87: C
88: C/IF PARA( PVM MPI )
89: C
90:     common /MVPCOMM/ TSPEED(MAXTSK)
91:     real TSPEED
92: C
93: C TSPEED(maxtsk) R4 Telative processor speed of each task
94: C
95: C*
96: C* ... PVM/MPI host parameters.
97: C*
98:     parameter ( MAXHOST = 32 )
99:     common /PVMHS1/ HOST0,HOST(MAXHOST)
100:    character*16 HOST, HOST0
101: C*
102: C HOST0 C16 Host name of the starting process of the multiprocessing
103: C   task group.
104: C   When this parameter is specified on command line as
105: C   "MYHOSTNAME=...", relative speed parameters in "HOST=...:xx"
106: C   lines of the task control information file becomes effective
107: C   and histories are assigned to each tasks proportional to
108: C   the speeds of running hosts. ( even history assignment
109: C   if not specified.)
110: C HOST(MAXHOST) C16 Hostname list given through the task control
111: C   information file.
112: C
113:     common /PVMHS2/ NHOST,HSPEED(MAXHOST)
114:     real HSPEED
115: C
116: C NHOST I4 Number of hosts available for multitasking by PVM/MPI.
117: C HSPEED(MAXHOST) R4 Relative processing speed of each host in
118: C   multitasking by PVM/MPI.
119: C
120: C/ENDIF
121: C
122: C
123: C/IF PARA( PVM )
124: C*
125: C* ... PVM task parameters.
126: C*
127:     common /PVMTID/ TASKID(MAXTSK) , ICTASK
128:     integer TASKID
129: C
130: C TASKID(NTASK) I4  Task ID of processes in multi tasking on PVM.

```

src/shared/INC/_TASKDT

```
131: C ICTASK I4 Task ID of "TASK CONTROLLER" process on PVM.
132: C This process is necessary in "NO-REPRODUCIBLE-RUN" mode.
133:
134: C
135: C
136: C/#ELSEIF PARA( MPI )
137: C
138: C ... declaration for MPI ... ( added June 1997 )
139: C
140: C*
141: C* ... MPI task parameters. ( mpif.h must be included already )
142: C*
143: CCCCC common /MPITID/ MVP_COMM, MPISTAT(MPI_STATUS_SIZE)
144: C* common /MPITID/ MVP_COMM
145: C* integer MVP_COMM
146: CCCC integer MPISTAT
147: C
148: C MVP_COMM I4 MPI's communicator handle.
149: C*** MPISTAT(MPI_STATUS_SIZE) I4 MPI's "status" array.
150: C
151:
152: C/#ENDIF
153: C
154: C
155: C parameter (MRBUF = 8192)
156: C
157: C*** added May 1997 **
158: C MRBUF C length of buffer array (integer,real,real*8,...) used to
159: C transport data by splitting large data into a reasonable size
160: C of packets in message passing etc.
161: C
162: C Should be used splitted record size in restart file input/output
163: C and local buffer array size.
164: C
165: C/#IF PARA(PVM MPI)
166: C common /SUBTSK/ STFNN, TLFNN
167: C character*256 STFNN, TLFNN
168: C/#ENDIF
169: C
170: C STFNN : C : 21 Jan 2001, Y.Nagaya
171: C STFNN is a file name to which printing data is output from
172: C each sub-task. This data read from the command line of the
173: C standard input like 'subtaskout=XXXXX'.
174: C TLFNN : C : 06 Oct 2015, Y.Nagaya
175: C TLFNN is a file name to which tim-list data is output from
176: C each sub-task. This data read from the command line of the
177: C standard input like 'timelistout=XXXXX'.
```

src/shared/INC/_TIMEDT

```
1: c*
2: c*
3: c*  Data for elapsed time & CPU time
4: c*
5: c*
6: c*  position where paramters are defined
7: c*    (elapsed time is defined by calling 'TOKEI')
8: c*    (CPU time is defined by calling 'CPUTM')
9: c*
10: c*  T00    : beginning time of this code (center.f)
11: c*  T11    : finishing time of this code (center.f)
12: c*  TE0    : before random walk start (mvp/action.f)
13: c*  TE1    : after end of random walk (mvp/action.f)
14: c*  TE2    : after summation of task data (mvp/mtsums.f)
15: c*
16: c*-----
17:      real    T00,T11,TE0,TE1,TE2
18:      common /ETDATA/ T00,T11,TE0,TE1,TE2
```

src/shared/INC/_VPPARA

```
1: C
2: C ==== VPP Fortran parallel mode timing data etc. =====
3: C
4: C <elapsed times>
5: C
6: C      tmtotl   : total
7: C      tmintr   : "INTRO" phase
8: C      tmpara   : parallel region
9: C
10: C      tminpp(mpe) : restart input
11: C      tmac(mpe)  : "ACTION" routine
12: C      tmsum(mpe) : tally summation
13: C      tmoutp(mpe) : file output (restart, source)
14: C
15: C      tmprint    : gather printout of "ACTION"
16: C
17: C      tmcdnz     : "CADENZ" phase
18: C
19: C ivpppr : printout I/O units in ACTION phase.
20: C
21: C vppprt : printout file name base
22: C
23: C
24: C      parameter( MPE = 4 )
25: C      parameter( MPE = 8 )
26: C      parameter( MPE = 16 )
27: C
28: C      common /VPPTM/ TMTOTL, TMINTR, TMPARA,
29: C      &              TMINPP(MPE), TMACT(MPE), TMSUM(MPE),
30: C      &              TMOUTP(MPE), TMPRINT,
31: C      &              TMCDNZ, IVPPPR(MPE)
32: C
33: C      real*8 TMTOTL, TMINTR, TMPARA,
34: C      &      TMINPP, TMACT, TMSUM, TMOUTP, TMPRINT, TMCDNZ
35: C
36: C      common /VPPP2/ VPPPRT
37: C
38: C      character*128 VPPPRT
39: C
```

src/shared/INC/_WKAB2Z

```
1: C*
2: C*.... POINTERS TO WORKING ARRAYS : ACTA2Z
3: C*
4:      COMMON /WKA2Z/  PAZ(30),PAZL(20)
5:      INTEGER      PAZ,PAZL
6: C*
7: C PAZ(30) I4  Pointers to working arrays for 'ACTA2Z' routine.
8: C PAZL(20) I4  Pointers to working arrays for 'LATDWN' routine
9: C              called from 'ACTA2Z'.
```

SAFE

src/shared/INC/_WORDL

```
1: C*
2: C*--- WORD LENGTH PARAMETER ( MWORD = 2 / 1 ... VP,SX/CRAY )
3: C*
4: C* (WORD LENGTH OF 8 BYTE REAL OR 'CHARACTER*8' VARIABLES)
5: C* SEE BLOCKDATA WDBL
6: C*
7: COMMON /WORDL/ MWORD
8: C*
9: C MWORD I4 Word length for 8-byte floating point number on users$
10: C $ machine.
11: C*
12: C =2 : 32 bit machines.
13: C =1 : 64 bit machines such as CRAY.
14: C*
```

src/cgview/aaalloc.f

```
1:      subroutine AAALOC
2:      c
3:      C/#IF .NOT.DYNAMIC
4:      c
5:      C=<MVP>=< for CGVIEW >=====
6:      C Purpose: memory size determination in non-dynamic (static) memory
7:      C      allocation mode.
8:      C
9:      C      In environment such as FACOM-M series or VP series,
10:     C      users can set real memory size as follows;
11:     C      * copy this routine.
12:     C      * modify size parameters (MAX & LMAX)
13:     C      * compile & link before execution.
14:     C
15:     C called in : MAIN
16:     C=====
17:     c
18:     c === size of shared memory (word) ==
19:     c
20:     c      parameter (MAXMEM = 200000000)
21:     c
22:     c === size of local memory (word) for multi tasking ==
23:     c
24:     C/#IF PARA
25:     *      parameter (LMAX = 200 0000)
26:     C/#ENDIF
27:
28:     common /ARRAY/ LIMIT,ILDUM,MAXMEM
29:     common /ARRAYZ/ IAINFL(3)
30:     c
31:     c ... for single tasking mode, array A & H are equivalent.
32:     c
33:     C/#IF .NOT.PARA.OR.PARA(VPP)
34:     REAL H(1)
35:     integer IAINFL(3)
36:     equivalence (LIMITL,LIMIT), (IAINF,IAINFL), (H,A)
37:     C/#ENDIF
38:     c
39:     c ... for multi tasking mode ....
40:     c
41:     C/#IF PARA( SX* )
42:     *      local common /LARRAY/ LIMITL, ILDUM, H(LMAX)
43:     *      local common /LARRAYZ/ IAINFL(3)
44:     C/#ELSEIF PARA( CRAY )
45:     *      task common /LARRAY/ LIMITL, ILDUM, H(LMAX)
46:     *      task common /LARRAYZ/ IAINFL(3)
47:     C/#ENDIF
48:     c
49:     c -----
50:     c
51:     LIMIT = MAXMEM
52:
53:     C/# IF PARA( CRAY SX* )
54:     *      LIMITL = LMAX
55:     C/# ENDIF
56:
57:     C/#ENDIF
58:     return
59:     end
```

src/cgview/acaloc.f

```
1:      subroutine ACALOC
2:      C
3:      C=<MVP>=====
4:      C Purpose: memory size setting for character memory
5:      C called in : MAIN
6:      C-----
7:      C
8:      C Estimation of variable sized character area requirement for MVP.
9:      C (November 1997)
10:     C
11:     C IZNAM(NINPZ)*12 : input zone name
12:     C NUCID(NUC)*16   : nuclide ID
13:     C MATP(NUC)*136   : neutron library comment
14:     C MATP(NUC)*136   : photon library comment
15:     C KREGS(NREG)*(LNAM) : list of region/frame names
16:     C TNAME(*)*(LNAM)   : list of region/frame names
17:     C
18:     C LNAM=12 or 16 depending on system.
19:     C
20:     C << required size (bytes) >>
21:     C
22:     C NINPZ*12 + NUC*( 16 + 136 + 136) + NREG*LNAM + <number of names>*LNAM
23:     C
24:     C For a large problem (LNAM=16);
25:     C
26:     C   #of input zone : 2000
27:     C   #of nuclide   : 300
28:     C   #of region    : 2000
29:     C   #of names     : 2000
30:     C
31:     C size is 174400 bytes = 43600 * 4 bytes
32:     C
33:     C So 60000*4 bytes may be sufficient for most problem, I hope.
34:     C
35:     C=====
36:     C
37:     C === size of character memory ( 4 bytes as a unit) ==
38:     C
39:     C   parameter( LCMAX   = 600000 )
40:     C
41:     C ... character data area (task shared) ....
42:     C
43:     C   integer LIMITC, IDUMC
44:     C   common /CARRAY/  CHA(LCMAX)
45:     C   common /CARRAYZ/ LIMITC, IDUMC, IAINFC(3)
46:     C   character*4 CHA
47:     C   integer IAINFC
48:     C
49:     C ... character data area (task local) .... for future use
50:     C
51:     C   integer LIMITC, IDUMC
52:     C   character*4 CHH
53:     C   common /CLARRAY/ CHH(1)
54:     C   common /CLARRAYZ/ LIMITLC, ILDUMC, IAINFLC(3)
55:     C
56:     C
57:     C   LIMITC = LCMAX
58:     C
59:     C   IAINFC(1) = 1
60:     C   IAINFC(2) = IAINFC(1)
61:     C   IAINFC(3) = 0
62:     C
63:     C   return
64:     C   end
```


src/cgview/action.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine ACTION(TITLE,A, IA,  H, IH, CHA)
3: C=<CGVIEW>=====
4: C  PURPOSE:  CONTROL OF MONTE CARLO RUN
5: C  CALLED IN: CENTER
6: C  CALLS:  SOURCE,SELECT,TALSUM,FLIGHT,SEARCH,LATICE
7: C
8: C=====
9: C/#IF MS_VISUAL
10: C
11: C ... This interface block is for CUTIL & MS-Visual tools. ...
12: C
13: *   interface
14: *     subroutine PIXSIZE( DXPPIX, DYPPIX )
15: *       real*8 DXPPIX, DYPPIX
16: *CDEC$      ATTRIBUTES C :: PIXSIZE
17: *CDEC$      ATTRIBUTES REFERENCE :: DXPPIX, DYPPIX
18: *     end subroutine
19: *   end interface
20: C/#ENDIF
21: C
22: C ... A & IA must have the same address
23:   real A(*)
24:   integer IA(*)
25: C
26: C ... H & IH must have the same address
27:   real H(*)
28:   integer IH(*)
29: C
30:   character*4 CHA(*)
31: C
32:   include 'INC/_KPIDS'
33:   include 'INC/_NGPS'
34: C
35:   include 'INC/_FLAGS'
36:   include '../shared/INC/_SIZES'
37:   include 'INC/_XBANK'
38:   include 'INC/_STACK'
39:   include 'INC/_XWORK'
40: C
41:   include '../shared/INC/_PGEOM'
42: C
43:   include '../shared/INC/_PTALY0'
44:   include '../shared/INC/_PTLSP'
45:   include 'INC/_WKSOU'
46:   include 'INC/_WKFL1'
47:   include 'INC/_WKSEL'
48:   include 'INC/_WKLAT'
49:   include 'INC/_WKGRA'
50:   include '../shared/INC/_IOUNIT'
51: C
52: C ... for CGVIEW
53: C
54:   include 'INC/_MVIEW'
55:   include 'INC/_MVGEO'
56: C
57:   character TITLE(2)*72
58:   integer JSTOP
59: C ... for "pixel" size
60:   real*8 DXPPIX, DYPPIX
61:   real*8 DYY0
62: C
63: C-----
64: C
65:   NZ1 = NZONE + 1

```

```

66: C
67: C .... GRAPHICS SUBROUTINE
68: C
69:   call GRAINT(A(LPAPER))
70: C
71: C ... get pixel size
72: C
73:   call PIXSIZE( DXPPIX, DYPPIX )
74:   write(IPR,'(1x,'Pixelsize in cm on display : ','2ell.4)')
75:   & DXPPIX,DYPPIX
76: C
77: C ... negative DYCG value means 1/(scan-line distance by pixel).
78: C
79:   DXCG = DYCG
80:   if( DYCG.lt.0.0 ) then
81:     DYY0 = abs(DYCG)
82:     DXCG = DXPPIX / DYY0
83:     DYCG = DYPPIX / DYY0
84:     write(IPR,'(1x,A,E12.5,'','',E12.5,A/))'
85:     & 'Line spacing was given by pixel. ',
86:     & DXCG, DYCG,' cm. x & y.'
87:   end if
88: C
89: C write(IPR,*) ' *** X INITIALIZATION FINISHED ***'
90: C
91:   ILOOP = 1
92:   JSTOP = 0
93: C
94: C .... START DRAWING. GENERATE PARTICLES IN BANK .....
95: C   NTGEN: TOTAL NUMBER OF GENERATED PARTICLES IN THIS RUN
96: C   NGENE: GENERATED PARTICLES IN A BATCH
97: C   NTHIST: TOTAL NUMBER OF GENERATED PARTICLES IN SUCCESSIVE RUNS
98: C           (EFFECTIVE IN RESTRT RUN)
99: C
100: Cccc CALL HEADER(IOW,' MONTE CARLO RUN ')
101: Cccc CALL HEADER(IOW,'CGVIEW RUN')
102: C write(IOW,7000) (TITLE(i):(iclen2(title(i))),i=1,2)
103: C7000 format(/1x,' PROBLEM TITLE: ',A/18x,A)
104: C
105: C
106: C
107: 4444 continue
108: C
109:   NTHIST = 0
110:   NTGEN = NTHIST
111:   NGENE = 0
112:   NGENE0 = 0
113:   NGENE2 = 0
114:   NDEAD = NBANK
115:   NPART = 0
116:   JREVR = 0
117: C
118:   if ( JGFLOOD.eq.0 ) then
119:     if ( ILOOP.eq.1 ) then
120:       7080 format(1x,
121:       & '>> draw vertical scan lines in +x/-x')
122:       if( JSCANX.gt.0 ) then
123:         write(ipr,7080)
124:         NPART = int(XMAX/DXCG) + 1
125:       else
126:         goto 3000
127:       end if
128:     end if
129: C
130:   if ( ILOOP.eq.2 ) then

```

src/cgview/action.f

```

131: 7090      format(1x,
132:      &      '>> draw horizontal scan lines in +y/-y')
133:      if( JSCANY.gt.0 ) then
134:        write(IPR,7090)
135:        NPART = INT(YMAX/DYCG) + 1
136:      else
137:        goto 3000
138:      end if
139:    end if
140: C
141: C      .... particle flooding mode
142: C
143: C      else
144: C        NPART = NGFLOOD
145: C      end if
146: C
147: C      CALL CPUTM(T0)
148: C      WRITE(IOW,'(1H0,' ' START ==== CPU ' ',E11.4,' ' (SEC) ==== ' ') TO
149: C
150: C
151: C *****
152: C ***** ONE-ZONE FLIGHT & SEARCH *****
153: C *****
154: C
155: C .... SELECT NEXT ACTION (STACK LENGTH CHECK ETC.) .....
156: C
157: 2222 continue
158:   call SELONE( MACT , MZONE, NZONE, NBANK, NHIST, NGEN, NPART,
159:   &           NGENE,   H(LNFFL), H(LNNXT), H(LNBREF),NREPS,JREEL,
160:   &           IDEFER, NCOLS, NDEAD,
161:   &           JLATT , H(LNXLT), NLBZ, JSTOP)
162: C
163: C   NN = NN + 1
164: C   JDEBG = 1
165: C   CALL CHECK(' == AFTER SELONE')
166: C   CALL DUMPST(A, ' == AFTER SELONE == ')
167: C   CALL DUMPBK( 1 ,NBANK,' == AFTER SELONE == ')
168: C   CHECK%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
169: C   WRITE(6,*) ' %%% MACT =' ,MACT,' MZONE ' ,MZONE,
170: C   &           ' NGENE,NDEAD ' ,NGENE,NDEAD
171: C   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172: C
173: C .....
174: C   MACT = 0 : GENERATE NEXT BATCH OF PARTICLES.
175: C   = 1 : FREE FLIGHT
176: C   = 2 : NEXT ZONE SEARCH
177: C   = 3 : COLLISION
178: C   = 5 : REFLECTION
179: C   = 6 : LATTICE-SEARCH
180: C   = 99 : END OF RUN
181: C   MZONE : SELECTED ZONE NUMBER (IF ZONEWISE LOGIC)
182: C .....
183: C
184:   if( NTHIST.eq.0 ) then
185:     call LOSTCH(NGENE2,JCGVDB,MACT,H(LLSDED),H(LXPLT),H(LYPLT),
186:   *       H(LBXPLT),H(LBYPLT),
187:   *       H(LGDIST),H(LAAAG),H(LBBBG),H(LCCCG),
188:   *       H(LCOLOR),H(LBCOLOR),
189:   *       STYLE,ILOOP,XMAX,YMAX)
190:   end if
191: C
192:   if( MACT.eq.0 ) then
193: C
194: C ..... TAKE SUMMATION OF TALLY ARRAYS EXEPT FOR THE FIRST BATCH
195: C

```

```

196: C
197: C ..... CHECK CPU TIME
198: C   CALL CPUTM(T1)
199: C   WRITE(IOW,7120) T1
200: 7120   format(/1X,' --- CPU ',E11.4,' (SEC) ----')
201: C
202:   if( NTHIST.gt.0 ) then
203:     call LOSTCH(NGENE2,JCGVDB,MACT,A(LLSDED),H(LXPLT),H(LYPLT),
204:   *       H(LBXPLT),H(LBYPLT),
205:   *       H(LGDIST),H(LAAAG),H(LBBBG),H(LCCCG),
206:   *       H(LCOLOR),H(LBCOLOR),
207:   *       STYLE,ILOOP,XMAX,YMAX)
208:   end if
209: C
210: C ..... GENERATE PARTICLES .....
211: C
212:   call SOURCE( IOW, IRAND,JREVR5,A(LPAPER), GRANGE,
213: 1 H(LXXX), H(LYYY), H(LZZZ), H(LAAA), H(LBBB), H(LCCC),
214: 2 H(LIZZ), NBANK,NZONE, NEST,
215: * H(LLEVL), H(LLZZ), H(LLPOS), H(LLCRS), H(LKLSF) ,
216: * H(LIBREG), A(LIBSPC), H(LIZSS),
217: & H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST), H(LKGCNT),
218: B H(LDBNK),KDBNK,MDBNK, H(LIBNK), KIBNK,MIBNK,
219: * NMEMS,
220: 6 H(LLSFFL), H(LNFFL), H(LIZFFL), H(LLSDED), NDEAD,
221: A A(LKZMAT), A(LKINPZ), CHA(LIZNAM), A(LKZREG), CHA(LKREGN),
222: B NINPZ,NMAT,NREG,A(LSDA),A(LKZDA),A(LKZAA),NSDA,
223: C NZDA,A(LKCELL),A(LIPCEL),NLBZ,A(LMLBZZ),DINF,DEPS,A(LKSREF),
224: C H(LIGMCK), H(LIPGMCK),H(LXGMCK),
225: S H(LLSLAT), H(LIZLAT), H(LNXLT) ,
226: H H(LXPLT), H(LYPLT) ,A(LLSCAN), ILOOP ,
227: * NGENE, NGENE0,
228: & NGEN, NPART,
229: * H(P1(1) ), H(P1(2) ), H(P1(3) ), H(P1(4) ),H(P1(5) ),
230: * H(P1(6) ), H(P1(7) ), H(P1(8) ), H(P1(9) ),H(P1(10)) )
231: C
232:   NTHIST = NTHIST + NGENE
233:   NGENE2 = NGENE
234: C
235:   call GRAPHS(A,1,NGENE,NGENE,'SOURCE',0,
236: &   H(LLSFFL), H(LIZFFL), H(LNFFL),
237: &   H(LXPLT),H(LYPLT),
238: &   H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST), H(LKGCNT),
239: &   A(LLSCAN),
240: &   H(LIZZ),H(LIBREG),A(LKMAT),A(LKREG),A(LKZMAT),A(LIDLAT),
241: &   A(LLTYP), A(LKZREG),A(LKINPZ),
242: &   A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
243: &   H(LLPOS),H(LLEVL),
244: &   H(LIBPWK),H(LLPOS0),H(LLEVL0),
245: &   H(P3(1) ),H(P3(2) ),H(P3(3) ),H(P3(4) ),H(P3(5) ),
246: &   ILOOP ,H(LCOLOR),H(LBCOLOR),H(LBXPLT),H(LBYPLT),
247: &   H(P3(6) ),H(P3(7) ),H(P3(8) ),H(P3(9) ),H(P3(10)),H(P3(11)))
248: C
249: C .... MOVE PARTICLES TO NEXT COLLISION OR CROSSING POINT .....
250: C
251:   else if( MACT.eq.1 ) then
252:
253:     call FLIONE( IOW,IRAND,
254: N       NBANK, NZONE, NSDA, NZDA, NCELL,
255: N NDEAD, NREG,           DINF, DEPS, NLATT,
256: N NUNV, NSPACE,NKTCSP, NEST, NLBZ, MSGNST,
257: B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
258: B H(LIZZ ),           H(LLEVL ), H(LLZZ ), H(LLPOS ),
259: B H(LLCRS ), H(LKLSF ), H(LIBSPC),
260: B H(LDBNK),KDBNK,MDBNK, H(LIBNK), KIBNK,MIBNK,

```

src/cgview/action.f

```

261:      S H(LLSFFL), H(LNFFL ), H(LIZFFL),
262:      S H(LLSSRC), H(LNNXT ), H(LIZNXT), H(LLSDED),
263:      G A(LSDA),A(LKZMAT),A(LKZREG),A(LKSPSU),A(LKTCSP),
264:      G A(LKZDA),A(LKZAA),
265:      G A(LDALT ), A(LKDALT), A(LNVLAT), A(LSZLAT), A(LIPLAT),A(LIPCEL),
266:      G A(LKSLAT), A(LLTYP ), A(LMLBZZ), A(LKZLBZ),A(LKCELL), A(LICTYP),
267:      G A(LKLATT), A(LCELSZ), A(LPNND),A(LKNND),NPNND,
268:      G A(LPPPF), A(LKPPF), NPPPF,
269:      G H(LXPLT ), H(LYPLT ), H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST),
270:      G A(LPAPER), ILOOP ,
271:      W H(P0(1) ), H(P0(2) ), H(P0(3) ), H(P0(4) ), H(P0(5) ), H(P0(6) ),
272:      W H(P0(7) ), H(P0(8) ), H(P0(9) ), H(P0(10)), H(P0(11)), H(P0(12)),
273:      W H(P0(13)), H(P0(14)), H(P0(15)), H(P0(16)), H(P0(17)), H(P0(18)),
274:      W H(P0(19)), H(P0(20)), H(P0(21)), H(P0(22)), H(P0(23)), H(P0(24)),
275:      W H(P0(25)), H(P0(26)), H(P0(27)), H(P0(28)), H(P0(29)), H(P0(30)),
276:      W H(P0(31)), H(P0(32)), H(P0(33)), H(P0(34)), H(P0(35)), H(P0(36)),
277:      & MZONE, LEVEL, MCELL, IMCELL, JCGVDB )
278: C
279: C
280: C .... FIND NEXT ZONE TO ENTER .....
281: C
282:      else if(MACT.eq.2) then
283:
284:          N1 = IH(LNFFL + NZONE)
285: C
286:          call SEAONE( IOW, JREVR, JGMCK,
287:      N H(LIGMCK), H(LIPGMCK), H(LXGMCK), NIPGMCK,MIPGMCK, DOVERLAP,
288:      N MPLIMIT,
289:      N NBANK,NINPZ,NZONE,NSDA,NZDA,IDEFER, NDEAD,
290:      N NREFS, NREG, NLBZ, NEST, NSUZON, NSPACE, NSURF, DEPS, MSGLST,
291:      N H(LNDEFSRC),
292:      B H(LXXX ), H(LYYY ), H(LZZZ ), H(LAAA ), H(LBBB ), H(LCCC ),
293:      B H(LIZZ ), H(LLEVL ), H(LZZZ ),
294:      B H(LLPOS ), H(LLCRS ), H(LKLSF ), H(LIBREG), H(LIBSPC),
295:      B H(LIZSS), H(LIBNK), KIBNK, MIBNK,
296:      S H(LLSFFL), H(LNFFL ), H(LIZFFL), H(LLSSRC), H(LNNXT ), H(LIZNXT),
297:      S H(LLSDED), H(LLSREF), H(LISREF), H(LIZREF), H(LNGBREF), A(LKSREF),
298:      S A(LLSLAT), A(LIZLAT), A(LNXLT ),
299:      G A(LSDA),A(LIPSDA),A(LIBSDA),CHA(LKREGN),A(LKINPZ),CHA(LIZNAM),
300:      G A(LKZMAT),A(LKZREG),A(LKZDA),A(LKZAA),
301:      G A(LKCELL), A(LIPCEL), A(LMLBZZ), A(LISUSP), H(LLPOS0),
302:      W H(P2(1) ), H(P2(2) ), H(P2(3) ), H(P2(4) ), H(P2(5) ), H(P2(6) ),
303:      W H(P2(7) ), H(P2(8) ), H(P2(9) ), H(P2(10)), H(P2(11)), H(P2(12)),
304:      P LEVEL, MCELL, IMCELL,
305:      P MZONE, NMEMO,A(LKMEMO2),A(LMEMC2),A(LMEMZ2))
306: C
307:      N11= IH(LNFFL + NZONE)
308: C
309: C ... "reflected" particles may be added to flight stack ...
310: C
311:      N2 = IH(LNFFL + NZONE) + IH(LNBREF)
312: C
313:      if( (N2-N1).gt.0 ) then
314: C
315:          call GRAPHS(A,N1+1,N2,NGENE,'SEAONE1',MZONE,
316:      & H(LLSFFL), H(LIZFFL), H(LNFFL),
317:      & H(LXPLT),H(LYPLT),
318:      & H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST), H(LKGCNT),
319:      & A(LLSCAN),
320:      & H(LIZZ),H(LIBREG),A(LKMAT),A(LKREG),A(LKZMAT),A(LIDLAT),
321:      & A(LLTYP),A(LKZREG),A(LKINPZ),
322:      & A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
323:      & H(LLPOS),H(LLEVL),
324:      & H(LIBPWK),H(LLPOS0),H(LLEVL0),
325:      & H(P3(1) ),H(P3(2) ),H(P3(3) ),H(P3(4) ),H(P3(5) ),

```

```

326:      & ILOOP, H(LCOLOR),H(LBCOLOR),H(LBXPLT),H(LBYPLT),
327:      & H(P3(6) ),H(P3(7) ),H(P3(8) ),H(P3(9) ),H(P3(10)),
328:      & H(P3(11)))
329: C
330:      end if
331: C
332: C ... draw pathes in reflector
333: C elements after LSFLL(NFFL(NZONE+1)+1) includes
334: C particles in reflector (dirty trick in SEAONE...)
335: C
336:      if((N2-N11).gt.0 .and. STYLE.eq.0) then
337: C
338: C ..... cut flight distance on drawing range boundary
339: C
340:      call FLYEND(N11+1,N2,H(LXPLT),H(LYPLT),
341:      & H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST),
342:      & A(LLSFFL),XMAX,YMAX,
343:      & ILOOP)
344: C
345:      call GRAPHS(A,N11+1,N2,NGENE,'SEAONE2',MZONE,
346:      & H(LLSFFL), H(LIZFFL), H(LNFFL),
347:      & H(LXPLT),H(LYPLT),
348:      & H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST), H(LKGCNT),
349:      & A(LLSCAN),
350:      & H(LIZZ),H(LIBREG),A(LKMAT),A(LKREG),A(LKZMAT),A(LIDLAT),
351:      & A(LLTYP),A(LKZREG),A(LKINPZ),
352:      & A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
353:      & H(LLPOS),H(LLEVL),
354:      & H(LIBPWK),H(LLPOS0),H(LLEVL0),
355:      & H(P3(1) ),H(P3(2) ),H(P3(3) ),H(P3(4) ),H(P3(5) ),
356:      & ILOOP, H(LCOLOR),H(LBCOLOR),H(LBXPLT),H(LBYPLT),
357:      & H(P3(6) ),H(P3(7) ),H(P3(8) ),H(P3(9) ),H(P3(10)),
358:      & H(P3(11)))
359: C
360:      end if
361: C
362: C .... LATTICE SEARCH .....
363: C
364:      else if( MACT.eq.6 ) then
365: C
366: C CALL LATTICE(IOW, JDEBG, JVMNT,JTLT,JHLAT, NLOST,
367: C N NBANK,NZONE,NLATT,NCELL,NLBZ,NEST,NSPACE,NUNV, DEPS,
368: C B A(LXXX ), A(LYYY ), A(LZZZ ), A(LAAA ), A(LBBB ), A(LCCC ),
369: C B A(LIZZ ), A(LLEVL ), A(LZZZ ), A(LLPOS ), A(LLSDED), NDEAD,
370: C B A(LLCRS ), A(LIBREG), A(LIBSPC),
371: C S A(LLSSRC), A(LIZNXT), A(LNNXT ), A(LLSLAT), A(LIZLAT), A(LNXLT ),
372: C G A(LKZMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
373: C G A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
374: C G A(LLTYP), A(LICTYP), A(LKDALT), A(LDALT ),
375: C G A(LKZREG), A(LKSPSU), A(LKTCSP), NKTCSP, A(LKHLAT),
376: C W A(P5( 1)), A(P5( 2)), A(P5( 3)), A(P5( 4)), A(P5( 5)), A(P5( 6)),
377: C W A(P5( 7)), A(P5( 8)), A(P5( 9)), A(P5(10)) )
378: C
379: C
380:      N1 = IH(LNFFL + NZONE)
381: C
382: C call LATTICE( JDEBG, JVMNT,JTLT,JHLAT,JFISX,H(LNLOST),
383: C N NBANK,NZONE,NREG,NLATT,NCELL,NLBZ,NEST,NSUZON,NSPACE,NUNV,
384: C N DINF,DEPS,IRAND,NMKREG,
385: C B H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),H(LCCC),H(LIZZ),
386: C B H(LLEVL),H(LZZZ),H(LLPOS),H(LLSDED),NDEAD,H(LLCRS),
387: C B H(LKLSF), H(LIBREG),H(LIBSPC),
388: C B H(LDBNK), KDBNK,MDBNK,H(LIBNK), KIBNK, MIBNK,
389: C S H(LLSSRC), H(LIZNXT), H(LNNXT ), H(LLSLAT), H(LIZLAT), H(LNXLT ),
390: C S H(LLSFFL), H(LNFFL), H(LIZFFL),

```

src/cgview/action.f

```

391:      G A(LSDA), A(LKZDA), A(LKZAA),
392:      G A(LKZMAT), A(LKCELL), A(LKZLBZ), A(LMLBZZ), A(LCELSZ), A(LSZLAT),
393:      G A(LIDLAT), A(LIPLAT), A(LKLATT), A(LKSLAT), A(LNVLAT), A(LIPCEL),
394:      G A(LLTYP), A(LICTYP), A(LKDALT), A(LDALT), A(LKZREG),
395:      G A(LISUSP), A(LKSPSU), A(LKTCSP), NKTCSP, A(LKHLAT), A(LMKREG),
396:      G A(LPPPF), A(LKPPF), NPPPF,
397:      W H(P5( 1)), H(P5( 2)), H(P5( 3)), H(P5( 4)), H(P5( 5)), H(P5( 6)),
398:      W H(P5( 7)), H(P5( 8)), H(P5( 9)), H(P5(10)),
399:      W H(P5(11)), H(P5(12)), H(P5(13)), H(P5(14)), H(P5(15)), H(P5(16))
400: C
401:      N2= IH(LNFFL + NZONE)
402:      if( (N2-N1).gt.0) then
403: C
404:          call GRAPHS(A,N1+1,N2,NGENE,'LATICE',0,
405:      &      H(LLSFFL), H(LIZFFL), H(LNFFL),
406:      &      H(LXPLT), H(LYPLT),
407:      &      H(LAAAG), H(LBBBG), H(LCCCG), H(LGDIST), H(LKGENT),
408:      &      A(LLSCAN),
409:      &      H(LIZZ), H(LIBREG), A(LKMAT), A(LKREG), A(LKZMAT), A(LIDLAT),
410:      &      A(LLTYP), A(LKZREG), A(LKINPZ),
411:      &      A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
412:      &      H(LLPOS), H(LLEVL),
413:      &      H(LIBPWK), H(LLPOS0), H(LLEVL0),
414:      &      H(P3( 1)), H(P3( 2)), H(P3( 3)), H(P3( 4)), H(P3( 5)),
415:      &      ILOOP, H(LCOLOR), H(LBCOLOR), H(LBXPLT), H(LBYPLT),
416:      &      H(P3( 6)), H(P3( 7)), H(P3( 8)), H(P3( 9)), H(P3(10)),
417:      &      H(P3(11)))
418: C
419:      end if
420: C
421: C
422: C
423:      else if(MACT.eq.99) then
424: C
425:          if(NGENE2.gt. 0) then
426:              call LOSTCH(NGENE2,JCGVDB,MACT,H(LLSDDED),H(LXPLT),H(LYPLT),
427:      *      H(LBXPLT),H(LBYPLT),
428:      *      H(LGDIST),H(LAAAG),H(LBBBG),H(LCCCG),
429:      *      H(LCOLOR),H(LBCOLOR),
430:      *      STYLE,ILOOP,XMAX,YMAX)
431:          end if
432:          goto 3000
433:      end if
434: C
435: C
436:      go to 2222
437: C
438: C ... proceed to next stage or quit
439: C
440: 3000 continue
441:      if( JGFLOOD.eq.0 ) then
442:          if( ILOOP.gt.1 ) then
443:              call GRAPHD(A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
444:      &      A(LIDLAT) )
445:          goto 9999
446:      else
447:          ILOOP = ILOOP + 1
448:          MACT = 0
449:          JSTOP = 0
450:          goto 4444
451:      end if
452:      else
453:          call GRAPHD(A(LZONCLR), A(LIZNCLR), A(LREGCLR), A(LMATCLR),
454:      &      A(LIDLAT) )
455:          goto 9999

```

```

456:      end if
457: C
458: C ..... END OF RANDOM WALK .....
459: C
460: 9999 continue
461: C
462: C9999 CALL CPUTM(T1)
463: C      WRITE(IOW,7100) NTGEN,T1-T0
464: C7100 FORMAT(1H0,'      === ',I8,' PARTICLES.   CPU FOR RUN =' ,F9.3,
465: C
466:      write(IOW,7100) NTGEN
467: 7100 format(1X,'>> Scan lines drawn : ',i5)
468: C
469: C ... sumary of zone overlap check ....
470: C
471:      if ( JGMCK.ne.0 ) then
472:          call PRGMCK( IOW, H(LIGMCK), H(LIPGMCK), H(LXGMCK), NIPGMCK,
473:      &      MIPGMCK,
474:      &      CHA(LIZNAM), A(LKINPZ), NZONE, NINPZ )
475:      end if
476: C
477: C ... deferred search warning ...
478: C
479:      call CKDEFSRC( IOW, H(LNDEFSRC), NINPZ, NMEMO, CHA(LIZNAM) )
480: C
481:      return
482:      end

```

src/cgview/astoeb.c

```
1: /*****
2:  *
3:  *  astoeb.c : Transform ASCII character to EBCDIC character
4:  *
5:  *****/
6:
7: astoeb( asc, ebc )
8:     int *ebc;
9:     char *asc;
10: {
11:     static int ebcdic[128] = { 0x00, 0x01, 0x02, 0x03, 0x37, 0x2d, 0x2e, 0x2f
12:                                0x16, 0x05, 0x09, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f
13:                                0x10, 0x11, 0x12, 0xb0, 0x3c, 0x3d, 0x32, 0x26
14:                                0x18, 0x19, 0x37, 0x27, 0x1c, 0x1d, 0x1e, 0x1f
15:                                0x40, 0x5a, 0x7f, 0x7b, 0xe0, 0x6c, 0x50, 0x7d
16:                                0x4d, 0x5d, 0x5c, 0x4e, 0x6b, 0x6d, 0x4b, 0x61
17:                                0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7
18:                                0xf8, 0xf9, 0x7a, 0x5e, 0x4c, 0x7e, 0x6e, 0x6f
19:                                0x7c, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7
20:                                0xc8, 0xc9, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6
21:                                0xd7, 0xd8, 0xd9, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6
22:                                0xe7, 0xe8, 0xe9, 0x4a, 0x5b, 0x5a, 0x5f, 0x61
23:                                0x79, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87
24:                                0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96
25:                                0x97, 0x98, 0x99, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6
26:                                0xa7, 0xa8, 0xa9, 0xc0, 0xf, 0xd0, 0xa1, 0x07
27: };
28:     *ebc = ebcdic[*asc];
29: }
```

src/cgview/calplot.c

```
1: #ifdef PIFFLIB
2: /*****
3:  *
4:  *   calplot.c : This parts is Calcomp Interface Routine
5:  *
6:  * *****/
7:
8: #include <X11/Xlib.h>
9: #include <X11/Xutil.h>
10: #include <stdio.h>
11: #include "piflib.h"
12:
13: #ifdef CALL_C_CRAY
14: #include <fortran.h>
15: #endif
16:
17: /*****
18:  *   get output device # and PS/EPSoutput file name
19:  * *****/
20:
21: /** static int   output_device = 0 ; **/
22: static char ps_file[256] ;
23:
24: #ifdef _NEED_UBAR
25: plotdevice_( idevice, psfile, lpsf )
26: #elif _UPPERCASE
27: PLOTDEVICE( idevice, psfile, lpsf )
28: #else
29: plotdevice( idevice, psfile, lpsf )
30: #endif
31: {
32:     int *idevice ;
33:     #ifdef CALL_C_CRAY
34:     _fcd psfile ;
35:     #else
36:     char *psfile ;
37:     #endif
38:     int *lpsf ;
39:     {
40:         output_device = *idevice ;
41:         ps_file[0] = '\0' ;
42:     }
43:     #ifdef CALL_C_CRAY
44:     if( *lpsf > 0 ) strncpy( ps_file, _fcdtocr(psfile), *lpsf ) ;
45:     #else
46:     if( *lpsf > 0 ) strncpy( ps_file, psfile, *lpsf ) ;
47:     #endif
48: }
49:
50: #ifdef _NEED_UBAR
51: winname_( name, length )
52: #elif _UPPERCASE
53: WINNAME( name, length )
54: #else
55: winname( name, length )
56: #endif
57: {
58:     char *name ;
59:     int *length ;
60:     {
61:         if ( output_device == 1 ){
62:             xwinname( name, *length ) ;
63:         }
64:     }
65: }
66:
67: #ifdef _UPPERCASE
68: PLOTS( dummy, jxps )
69: #else
70: plots( dummy, jxps )
71: #endif
72: {
73:     char dummy[] ;
74:     int *jxps ;
75:
76:     if( !output_device ) {
77:         START:
78:         printf( "**** Plot Interface Library Version 1.30 ***\n" );
79:         printf( "   Input to output Device      \n" );
80:         printf( "       1 : to X-Window                      \n" );
81:         printf( "       2 : to PostScript File               \n" );
82:         printf( "       3 : to EPS File for TeX              \n" );
83:         printf( "   Select [ 1 - 3 ] :   " );
84:
85:         i = 0 ;
86:         while((c = getchar()) != '\n') {
87:             item[i] = c ;
88:             i++ ;
89:             if( i > 5 )
90:                 break ;
91:         }
92:
93:         switch( item[0] ){
94:             case '1':
95:                 output_device = 1 ;
96:                 break ;
97:             case '2':
98:                 output_device = 2 ;
99:                 break ;
100:             case '3':
101:                 output_device = 3 ;
102:                 break ;
103:             default:
104:                 item[0] = '\0' ;
105:                 goto START ;
106:         }
107:     }
108:
109:     /**
110:     jxps == 1 for non-postscript output
111:     jxps == 2 for Postscript or EPS
112:     ***/
113:
114:     switch( output_device ) {
115:         case 1:
116:             xplots() ;
117:             *jxps=1 ;
118:             break ;
119:         case 2:
120:             psplots(ps_file) ;
121:             *jxps=2 ;
122:             break ;
123:         case 3:
124:             f_value = 1.0 ;
125:             EPS_page = 1 ;
126:             texplots(ps_file) ;
127:             *jxps=2 ;
128:             break ;
129:     }
130: }
```

src/cgview/calplot.c

```
131: }
132:
133: #ifdef _NEED_UBAR
134: getwinid( windowid )
135: #elif _UPPERCASE
136: GETWINID( windowid )
137: #else
138: getwinid( windowid )
139: #endif
140: int *windowid;
141: {
142:     switch( output_device ){
143:     case 1:
144:         xwinid( windowid );
145:         break;
146:     case 2:
147:         *windowid=-999 ;
148:         break;
149:     case 3:
150:         *windowid=-999 ;
151:         break;
152:     }
153: }
154:
155: #ifdef _NEED_UBAR
156: plot_( x_point1, y_point1, iflag )
157: #elif _UPPERCASE
158: PLOT( x_point1, y_point1, iflag )
159: #else
160: plot( x_point1, y_point1, iflag )
161: #endif
162: float *x_point1, *y_point1;
163: int *iflag;
164: {
165:     switch( output_device ){
166:     case 1:
167:         xplot( *x_point1, *y_point1, *iflag );
168:         break;
169:     case 2:
170:         psplot( *x_point1, *y_point1, *iflag );
171:         break;
172:     case 3:
173:         texplot( *x_point1, *y_point1, *iflag );
174:         break;
175:     }
176: }
177:
178: #ifdef _NEED_UBAR
179: dot_( x1, y1 )
180: #elif _UPPERCASE
181: DOT( x1, y1 )
182: #else
183: dot( x1, y1 )
184: #endif
185: float *x1, *y1;
186: {
187:     switch( output_device ){
188:     case 1:
189:         xdot( x1, y1 );
190:         break;
191:     case 2:
192:         psdot( x1, y1 );
193:         break;
194:     case 3:
195:         texdot( x1, y1 );
```

```
196:         break;
197:     }
198: }
199:
200: #ifdef _NEED_UBAR
201: setrgb_( red, green, blue )
202: #elif _UPPERCASE
203: SETRGB( red, green, blue )
204: #else
205: setrgb( red, green, blue )
206: #endif
207: unsigned int *red, *green, *blue;
208: {
209:     switch( output_device ){
210:     case 1:
211:         xsetrgb( red, green, blue );
212:         break;
213:     case 2:
214:         pssetrgb( red, green, blue );
215:         break;
216:     case 3:
217:         pssetrgb( red, green, blue );
218:         break;
219:     }
220: }
221:
222: #ifdef _NEED_UBAR
223: setrgbnum_( color_number, red, green, blue )
224: #elif _UPPERCASE
225: SETRGBNUM( color_number, red, green, blue )
226: #else
227: setrgbnum( color_number, red, green, blue )
228: #endif
229: unsigned int * color_number ;
230: unsigned int *red, *green, *blue;
231: {
232:     switch( output_device ){
233:     case 1:
234:         xsetrgbnum( color_number, red, green, blue );
235:         break;
236:     case 2:
237:         case 3:
238:             pssetrgbnum( color_number, red, green, blue );
239:             break;
240:     }
241: }
242:
243: #ifdef _NEED_UBAR
244: setclrs_( color_number )
245: #elif _UPPERCASE
246: SETCLRS( color_number )
247: #else
248: setclrs( color_number )
249: #endif
250: unsigned int * color_number ;
251: {
252:     switch( output_device ){
253:     case 1:
254:         xsetclrs( color_number );
255:         break;
256:     case 2:
257:     case 3:
258:         pssetclrs( color_number );
259:         break;
260:     }
```

src/cgview/calplot.c

```
261: }
262:
263:
264: #ifdef _NEED_UBAR
265: setclr_( color_number )
266: #elif _UPPERCASE
267: SETCLR( color_number )
268: #else
269: setclr( color_number )
270: #endif
271: int *color_number;
272: {
273:     long color_numberl = *color_number ;
274:     switch( output_device ){
275:     case 1:
276:         xsetclr( &color_numberl );
277:         break;
278:     case 2:
279:         pssetclr( &color_numberl );
280:         break;
281:     case 3:
282:         pssetclr( &color_numberl );
283:         break;
284:     }
285: }
286:
287: #ifdef _NEED_UBAR
288: setltp_( kind_line )
289: #elif _UPPERCASE
290: SETLTP( kind_line )
291: #else
292: setltp( kind_line )
293: #endif
294: int *kind_line;
295: {
296:     switch( output_device ){
297:     case 1:
298:         xsetltp( kind_line );
299:         break;
300:     case 2:
301:         pssetltp( kind_line );
302:         break;
303:     case 3:
304:         pssetltp( kind_line );
305:         break;
306:     }
307: }
308:
309: #ifdef _NEED_UBAR
310: setlwd_( wd_line )
311: #elif _UPPERCASE
312: SETLWD( wd_line )
313: #else
314: setlwd( wd_line )
315: #endif
316: int *wd_line;
317: {
318:     switch( output_device ){
319:     case 1:
320:         xsetlwd( wd_line );
321:         break;
322:     case 2:
323:         pssetlwd( wd_line );
324:         break;
325:     case 3:
326:         pssetlwd( wd_line );
327:         break;
328:     }
329: }
330:
331: #ifdef _NEED_UBAR
332: cpfont_( font_name )
333: #elif _UPPERCASE
334: CFFONT( font_name )
335: #else
336: cpfont( font_name )
337: #endif
338: char *font_name;
339: {
340:     switch( output_device ){
341:     case 1:
342:         xcpfont( font_name );
343:         break;
344:     case 2:
345:         pscpfont( font_name );
346:         break;
347:     case 3:
348:         pscpfont( font_name );
349:         break;
350:     }
351: }
352:
353: #ifdef _NEED_UBAR
354: factor_( factor_value )
355: #elif _UPPERCASE
356: FACTOR( factor_value )
357: #else
358: factor( factor_value )
359: #endif
360: float *factor_value;
361: {
362:     switch( output_device ){
363:     case 1:
364:         xfactor( factor_value );
365:         break;
366:     case 2:
367:         psfactor( factor_value );
368:         break;
369:     case 3:
370:         psfactor( factor_value );
371:         break;
372:     }
373: }
374:
375: #ifdef _NEED_UBAR
376: newpen_( ipen )
377: #elif _UPPERCASE
378: NEWPEN( ipen )
379: #else
380: newpen( ipen )
381: #endif
382: int *ipen;
383: {
384:     long ipenl = *ipen;
385:     switch( output_device ){
386:     case 1:
387:         xnewpen( &ipenl );
388:         break;
389:     case 2:
390:         psnewpen( &ipenl );
```


src/cgview/calplot.c

```
391:         break;
392:     case 3:
393:         psnewpen( &ipenl );
394:         break;
395:     }
396: }
397:
398: /**/ Added Sep 1999 by M.Sasaki ***/
399: /**/ iflag=1/2 = CapRound/CapButt **/
400:
401: #ifdef _NEED_UBAR
402: capline( iflag );
403: #elif _UPPERCASE
404: CAPLINE( iflag );
405: #else
406: capline( iflag );
407: #endif
408: int *iflag;
409: {
410:     switch( output_device ){
411:     case 1:
412:         xcapline( *iflag );
413:         break;
414:     case 2:
415:     case 3:
416:         pscapline( *iflag );
417:         break;
418:     }
419: }
420:
421: #ifdef _NEED_UBAR
422: getcapline_( iflag );
423: #elif _UPPERCASE
424: GETCAPLINE( iflag );
425: #else
426: getcapline( iflag );
427: #endif
428: int *iflag;
429: {
430:     switch( output_device ){
431:     case 1:
432:         xgetcapline( iflag );
433:         break;
434:     case 2:
435:     case 3:
436:         psgetcapline( iflag );
437:         break;
438:     }
439: }
440:
441: #ifdef _NEED_UBAR
442: symfnt_( x_str, y_str, height, string, angle, no_of_letters )
443: #elif _UPPERCASE
444: SYMFNT( x_str, y_str, height, string, angle, no_of_letters )
445: #else
446: symfnt( x_str, y_str, height, string, angle, no_of_letters )
447: #endif
448:     float *x_str, *y_str;
449:     float *height, *angle;
450:     int *no_of_letters;
451:     char *string;
452: {
453:     switch( output_device ){
454:     case 1:
455:         xsymfnt( x_str, y_str, height, string, angle, no_of_letters );
```

```
456:         break;
457:     case 2:
458:         pssymfnt( x_str, y_str, height, string, angle, no_of_letters );
459:         break;
460:     case 3:
461:         texsymfnt( x_str, y_str, height, string, angle, no_of_letters );
462:         break;
463:     }
464: }
465:
466: #ifdef _NEED_UBAR
467: symbol_(x, y, hh, ibcda, angle, nchar )
468: #elif _UPPERCASE
469: SYMBOL(x, y, hh, ibcda, angle, nchar )
470: #else
471: symbol(x, y, hh, ibcda, angle, nchar )
472: #endif
473:     float *x, *y, *hh, *angle;
474:     int *nchar;
475:     int *ibcda;
476: {
477:     switch( output_device ){
478:     case 1:
479:         xsymbol(x, y, hh, ibcda, angle, nchar );
480:         break;
481:     case 2:
482:         pssymbol(x, y, hh, ibcda, angle, nchar );
483:         break;
484:     case 3:
485:         texsymbol(x, y, hh, ibcda, angle, nchar );
486:         break;
487:     }
488: }
489:
490: #ifdef _NEED_UBAR
491: where_(x, y, fact)
492: #elif _UPPERCASE
493: WHERE(x, y, fact)
494: #else
495: where(x, y, fact)
496: #endif
497:     float *x, *y, *fact;
498: {
499:     xwhere(x,y,fact);
500: }
501:
502: #ifdef _NEED_UBAR
503: number_(x, y, hh, flt, angle, n)
504: #elif _UPPERCASE
505: NUMBER(x, y, hh, flt, angle, n)
506: #else
507: number(x, y, hh, flt, angle, n)
508: #endif
509:     float *x, *y, *hh, *flt, *angle;
510:     int *n;
511: {
512:
513: #define MAXS 80
514:
515:     unsigned int nchar;
516:     char numb[MAXS];
517:     char i;
518:
519:     if ( *n > 9 ) {
520:         *n = 9;
```

src/cgview/calplot.c

```
521:     }
522:
523:     sprintf( numb, "%.9f", *flt );
524:
525:     for ( i = 0 ; i < MAXS ; i++ ) {
526:         if ( numb[i] == '.' ) {
527:             nchar = i;
528:             break;
529:         }
530:     }
531:
532:     if ( *n == 0 ) {
533:         nchar++;
534:         numb[nchar] = '\\0';
535:     } else if ( *n < 0 ) {
536:         numb[nchar] = '\\0';
537:     } else {
538:         nchar++;
539:         nchar += *n;
540:         numb[nchar] = '\\0';
541:     }
542:
543:     switch( output_device ){
544:     case 1:
545:         xsymbol(x, y, hh, numb, angle, &nchar );
546:         break;
547:     case 2:
548:         pssymbol(x, y, hh, numb, angle, &nchar );
549:         break;
550:     case 3:
551:         texsymbol(x, y, hh, numb, angle, &nchar );
552:         break;
553:     }
554: }
555:
556: #ifdef _NEED_UBAR
557: plotlines_( nml, xpoints, ypoints )
558: #elif _UPPERCASE
559: PLOT_LINES( nml, xpoints, ypoints )
560: #else
561: plotlines( nml, xpoints, ypoints )
562: #endif
563: {
564:     int *nml;
565:     float *xpoints, *ypoints;
566:     {
567:         switch( output_device ){
568:         case 1:
569:             xplotlines( *nml, xpoints, ypoints );
570:             break;
571:         case 2:
572:             psplotlines( *nml, xpoints, ypoints );
573:             break;
574:         case 3:
575:             texplotlines( *nml, xpoints, ypoints );
576:             break;
577:         }
578:     }
579: #endif /** end of ifdef PIFFLIB **/
```

src/cgview/canvs2.f

```

1:      subroutine CANVS2( ZONCLR, IZNCLR, REGCLR, MATCLR, IDLAT )
2: C=====
3: C   Draw legend for colors
4: C=====
5: C/IF MS_VISUAL
6: C
7: C ... This interface block is for CUTIL & MS-Visual tools. ...
8: C
9: C   interface
10: C      subroutine PLOT( RRR1, RRR2, III1 )
11: C         integer      III1
12: C         real          RRR1, RRR2
13: C         *CDEC$        ATTRIBUTES C :: PLOT
14: C         *CDEC$        ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
15: C      end subroutine
16: C      subroutine SETLWD( III2 )
17: C         integer      III2
18: C         *CDEC$        ATTRIBUTES C :: SETLWD
19: C         *CDEC$        ATTRIBUTES REFERENCE :: III2
20: C      end subroutine
21: C   end interface
22: C/ENDIF
23: C
24: C      common /XYP/      ICBK, BACK
25: C      integer BACK(200), ICBK
26: C
27: C      include 'INC/_FLAGS'
28: C      include '../shared/INC/_SIZES'
29: C      include '../shared/INC/_PGEOM'
30: C      include 'INC/_MVIEW'
31: C
32: C      integer ZONCLR(NZONE), IZNCLR(NINPZ), REGCLR(NREG),
33: C      &      MATCLR(0:MXMAT)
34: C
35: C      integer IDLAT(NLATT)
36: C
37: C ... local data
38: C
39: C      integer IU(200)
40: C      character*72 C
41: C      real X0, Y0
42: C
43: C ..... statement functions
44: C
45: C      include '../shared/INC/_PMLATT'
46: C      include '../shared/INC/_SFLATT'
47: C
48: C ... statement function to refer CLDAT
49: C
50: C      KCLDAT(I) = CLDAT(MIN(I, MXCLDAT))
51: C      KMATCLR(I) = MATCLR(MIN(I, MXMAT))
52: C
53: C-----
54: C
55: C      if ( STYLE.ne.0 ) then
56: C         call FRAME( 0.0, 0.0, XFRM, YFRM )
57: C         return
58: C      end if
59: C
60: C      if ( ICBK.gt.200 ) ICBK = 200
61: C
62: Cccc call SETCLR( 0 )
63: C      call NEWP( 0 )
64: C      call SETLWD( 1 )
65: C      call FRAME( 0.0, 0.0, XFRM, YFRM )

```

```

66: C
67: C ..... drawing position ....
68: C
69: Cccc X0      = UFRM + 3.*CDT
70: Cccc X0      = UFRM + (WFRAME-UFRM)/2.0+0.5*CDT
71: C      X0      = UFRM + (WFRAME-UFRM) /2.0 + 0.3*CDT
72: C      XT      = X0 + 1.2*CDT
73: Cccc D00      = CDT*0.8
74: C      D00      = WFRAME/16*0.9
75: C      DY0      = D00/3.0
76: Cccc Y0      = UFRM + CDT + 5./2.*DY0
77: C      Y0      = UFRM + (WFRAME-UFRM) /2.0 - 0.3*D00
78: C
79: C
80: C      do 160 I = 0, 16
81: C
82: C         ... draw color # (2 digit)
83: C
84: C         call NEWP( 0 )
85: C         write(C(1:2), '(i2)') I
86: C         SSYM = 0.16*CDT
87: C         call JSYMBOL( X0, Y0-0.3*DY0, SSYM, C, 0.0, 2 )
88: C
89: C         X1      = X0 + 0.35*CDT
90: C
91: C
92: C         call NEWP( I )
93: C         if ( JXPS.eq.1 ) then
94: C            call SETLWD( 3 )
95: C         else
96: C            call SETLWD( 5 )
97: C         end if
98: C         call PLOT( X1, Y0, 3 )
99: C         call PLOT( X0+CDT, Y0, 2 )
100: C         if ( JXPS.eq.1 ) then
101: C            call SETLWD( 1 )
102: C         else
103: C            call SETLWD( 1 )
104: C         end if
105: Cccc call SETCLR( 0 )
106: C      call NEWP( 0 )
107: C
108: C      N = 0
109: C      do 100 J = 1, ICBK
110: C
111: C1999.12.20 if ( BACK(J).gt.0 ) then
112: C      if ( BACK(J).gt.0 .or. SPTYP.eq.2.and.BACK(J).eq.0 ) then
113: C         if ( SPTYP.eq.-1
114: C            &      .and.ZONCLR(ABS(BACK(J))).eq.I .or. SPTYP.eq.0
115: C            &      .and.IZNCLR(ABS(BACK(J))).eq.I .or. SPTYP.eq.1
116: C            &      .and.REGCLR(ABS(BACK(J))).eq.I .or. SPTYP.eq.2
117: C            &      .and.KMATCLR(ABS(BACK(J))).eq.I ) then
118: C            N = N + 1
119: C            IU(N) = BACK(J)
120: C         end if
121: C
122: C      == lattice region in material drawing mode
123: C
124: CCCCCCCCC else if( SPTYP.eq.2.and.BACK(J).gt.-999 ) then
125: C      else if ( SPTYP.eq.2.and.ISLATT(BACK(J)) ) then
126: C         LID = LATNM(BACK(J))
127: CCCCCCCCC if ( CLDAT(ABS(BACK(J))).eq.I ) then
128: C         if ( KCLDAT(LID).eq.I ) then
129: C            N = N + 1
130: C            IU(N) = BACK(J)

```

src/cgview/canvs2.f

```

131:         end if
132: C
133: C      === invalid region or material.
134: C
135:         else if ( I.eq.0 ) then
136:
137: CCCCCC      if ( SPTYP.eq.1.and.BACK(J).eq.-999 ) then
138: C      if ( SPTYP.eq.1.and.BACK(J).eq.-9999 ) then
139: C          N      = N + 1
140: C          IU(N)   = BACK(J)
141: C      end if
142:
143:         if ( SPTYP.eq.2 ) then
144: CCCCCC      if ( BACK(J).eq.-999 )
145: C      if ( BACK(J).eq.-9999 .or. (BACK(J).ne.-1000
146: C          .and.BACK(J).ne.-2000.and.BACK(J).ne.-3000
147: C          .and.BACK(J).ne.-4000.and.BACK(J).ne.-999
148: C          .and. .not.ISLATT(BACK(J)) ) ) then
149: C          N      = N + 1
150: C          IU(N)   = BACK(J)
151: C
152: C      if ( JCGVDB(1).ne.0 ) then
153: C          write(*,*) '%%% CANVS2 invalid material? N ' N,
154: C          ' J ',J,' material ',BACK(J)
155: C      end if
156: C
157:         end if
158:     end if
159:
160:     else if ( I.eq.15 ) then
161: C      if ( SPTYP.eq.2.and.BACK(J).eq.-1000 .or. SPTYP.eq.1
162: C          .and.BACK(J).eq.-1 ) then
163: C          N      = N + 1
164: C          IU(N)   = BACK(J)
165: C      end if
166:     else if ( I.eq.16 ) then
167: C      if ( (SPTYP.eq.2.and.(BACK(J).eq.-2000
168: C          .or.BACK(J).eq.-3000.or.BACK(J).eq.-4000
169: C          .or.BACK(J).eq.-999))
170: C          .or. (SPTYP.eq.1.and.BACK(J).eq.-1000) ) then
171: C          N      = N + 1
172: C          IU(N)   = BACK(J)
173: C      end if
174:     end if
175:     100 continue
176: C
177: CCCCCC K      = 6
178: C      K      = 5
179: C      NN      = 0
180: C
181: C      SSYM    = 0.16*CDT
182: C
183: C      if ( I.eq.0.and.N.gt.0 ) then
184: C      Ccccc  call SETCLR( 2 )
185: C      call NEWP( 2 )
186: C      C      = 'INVALID REGION OR MATERIAL'
187: C      call JSYMBOL( XT, Y0-(NN+0.3)*DY0, SSYM, C, 0.0, ICLEN2(C) )
188: C      NN      = 1
189: C      end if
190: C
191: C
192: C      do 140 J = 1, N, K
193: C      if ( NN.gt.2 ) go to 150
194: C      KK      = MIN(J+K-1,N)
195: C      NNN     = KK - J + 1

```

```

196:         if ( I.lt.15.and.I.ne.0 ) then
197:         if ( SPTYP.eq.2 ) then
198:             do 130 M = J, KK
199:                 L1      = (M-J)*5 + 1
200:                 if ( ISLATT(IU(M)) ) then
201:                     write(C(L1:L1+4),'(i5)') IDLAT(LATTNM(IU(M)))
202:                     do 110 L = 4, 0, -1
203:                         if ( C(L1+L:L1+L).eq.' ' ) then
204:                             C(L1+L:L1+L) = '#'
205:                             go to 120
206:                         end if
207:                     110 continue
208:                     120 continue
209:                 else
210:                     write(C(L1:L1+4),'(i5)') IU(M)
211:                 end if
212:             130 continue
213:         else
214:             write(C,'(10i5)') (IU(M),M=J,KK)
215: C      ... fixed Jan 2000 (passing too long character length!!)
216: C      CCCCCCCCCC NNN      = NNN*5
217: C      end if
218: C      Check %%%%
219: C      write(*,*) 'canvs2 C <','> NNN ',NNN
220: C      %%%%%%%%%
221: C
222: C      call JSYMBOL( XT, Y0-(NN+0.3)*DY0, SSYM, C, 0.0, NNN*5 )
223: C
224: C      CCCCCC write(C,'(10i4)') (IU(M),M=J,KK)
225: C      CCCCCC call JSYMBOL( XT, Y0-(NN+0.3)*DY0, SSYM, C, 0.0, NNN*4 )
226: C      else
227: C      if ( SPTYP.ne.1 ) then
228: C          write(C,'(10i5)') (IU(M),M=J,KK)
229: C          NNN      = NNN*5
230: C      else
231: C          if ( I.eq.15 ) then
232: C              C      = 'LATTICE ZONE'
233: C              NNN      = 12
234: C          else if ( I.eq.16 ) then
235: C              C      = 'BOUNDARY ZONE'
236: C              NNN      = 13
237: C          end if
238: C      end if
239: C      Check %%%%
240: C      write(*,*) ' C <','> NNN ',NNN
241: C      %%%%%%%%%
242: C      call JSYMBOL( XT, Y0-(NN+0.3)*DY0, SSYM, C, 0.0, NNN )
243: C      end if
244: C      NN      = NN + 1
245: C      140 continue
246: C      150 continue
247: C
248: C      Y0      = Y0 - D00
249: C
250: C      160 continue
251: C      return
252: C      CC/#ENDIF
253: C      end

```

src/cgview/center.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine CENTER( JXPS0, MLIMIT, CODE )
3: C=====
4: C  CGVIEW
5: C=====
6: C  PURPOSE:   CENTRAL CONTROL ROUTINE
7: C  CALLED IN: MAIN
8: C  CALLS   DATE,TIME,INTRO,ACTION
9: C=====
10: C/#IF MS_VISUAL
11: C
12: C ... This interface block is for CUTIL & MS-Visual tools. ...
13: C
14: C   interface
15: C     subroutine WINNAME( VSTR, LLL )
16: C       character*64 VSTR
17: C       integer      LLL
18: C     *CDEC$ ATTRIBUTES C :: WINNAME
19: C     *CDEC$ ATTRIBUTES REFERENCE :: VSTR, LLL
20: C   end subroutine
21: C     subroutine PLOTS( DUMMY, JXPS0 )
22: C       real      DUMMY
23: C       integer   JXPS0
24: C     *CDEC$ ATTRIBUTES C :: PLOTS
25: C     *CDEC$ ATTRIBUTES REFERENCE :: DUMMY, JXPS0
26: C   end subroutine
27: C   end interface
28: C/#ENDIF
29: C
30:   character*(*) CODE
31: C
32:   INCLUDE '../shared/INC/_ARRAY'
33:   INCLUDE '../shared/INC/_PGEOM'
34:   character*72 TITLE(2), XDATE*12
35:   character*8 TIME8
36:   INCLUDE 'INC/_FLAGS'
37:   INCLUDE '../shared/INC/_IOUNIT'
38: C
39:   INCLUDE 'INC/_MVIEW'
40: C
41: C ... local variables
42: C
43:   character*64 VSTR
44:   character*32 DTSTR
45: C
46: C ... data statements should be placed after declarations ...
47:   INCLUDE 'INC/_PROGV'
48: C
49: C-----
50: C
51: C
52: C ..... PRINT DATE, TIME & PROGRAM DESCRIPTION .....
53: C
54: C   CALL HIZUKE(XDATE)
55: C   CALL JIKAN(TIME8)
56: C   WRITE(IPR,7000) XDATE,TIME8,(PROGV(I),I=1,40)
57: C
58: C7000 format(//////40X,' DATE   ',A/40X,' TIME   ',A//35X,'*',6
59: C   &      ('*****'),'*****'/35X,'*',64X,'**/
60: C   &      (35X,'* ',A60,' **'))
61: C   WRITE(IPR,7010)
62: C7020 format(35X,'!',64X,'!'/35X,'+',6('-----'),'-----+')
63: C
64: C ..... PREPARATION for MONTE CARLO RUN
65: C

```

```

66:   call INTRO( TITLE, MLIMIT, CODE )
67: C
68:   close( IUG1 )
69:   close( IUG2 )
70:   close( IUB )
71: C
72: C ..... DRAWING PICTURES
73: C
74:   call GETVER( VSTR, DTSTR )
75:   LLL = ICLEN2(VSTR)
76:   call WINNAME( VSTR(:LLL), LLL )
77: C
78:   call PLOTS( DUMMY, JXPS0 )
79: C
80: C/#IF PIFFLIB
81:   JXPS = JXPS0
82: C/#ENDIF
83: C
84: CCCC IF(JPICT.NE.0) CALL PABLO
85:   call HEADER( IPR, 'CGVIEW RUN' )
86: C
87:   100 continue
88:   call PABLO1( A, H, CHA, LIMIT, LIMITL, LIMITC, CODE, ICHK1,
89:   &      A(LPAPER), TITLE, A(LZONCLR), A(LIZNCLR), A(LREGCLR),
90:   &      A(LMATCLR), H(LIGMCK),H(LXGMCK), H(LNDEFSRC) )
91: C
92: C ..... MONTE CARLO RUN
93: C
94:   if ( ICHK1.eq.0 ) then
95:     call ACTION( TITLE, A, A, H, H, CHA )
96: C
97: C ..... DISPOSAL OF ACCUMULATED DATA ETC.
98: C
99:   go to 100
100:  else
101:    call GRAPHE
102:  end if
103: C
104:  return
105: end

```

src/cgview/cgparam.f

```

1: C -----
2: C
3: C -----
4: C .... echo current drawing parameters
5: C -----
6: C
7:       subroutine ECHOPARAM( IPR,      PAPER, XMAX, YMAX, JSCANX,JSCANY,
8:       &                     SPTYP,   STYLE, STYLE2,LEVEL, INTREFRESH,
9:       &                     MCELL,   IMCELL,JZREVERSE,  GRANGE,JGFLOOD )
10: C
11:       real*8 PAPER(10), XMAX, YMAX
12:       real*8 GRANGE(6)
13:       integer JSCANX, JSCANY, SPTYP, STYLE, STYLE2, LEVEL, INTREFRESH
14: C
15:       write(IPR,'(/lx,' <<< Current drawing parameters >>>'')')
16:
17:       write(IPR,7000) (PAPER(I),I=1,9)
18: 7000 format(1X,' CORNER : ',3E13.5/1X,' X-AXIS : ',3E13.5/1X,
19: &          ' Y-AXIS : ',3E13.5)
20:
21:       write(IPR,7020) XMAX, YMAX
22: 7020 format(1X,' X-AXIS LENGTH : ',E12.5,' Y-AXIS LENGTH : ',E12.5)
23:       write(IPR,'(lx,' LINES PER CM : ',E12.5)' PAPER(10)
24:       write(IPR,7040) SPTYP, MCELL, IMCELL
25: 7040 format(1X,' WHAT TO DRAW (SPTYP) : ',I5,' CELL TO DRAW : ',
26: &          I5,' (zone#',I5,'')')
27:       write(IPR,7060) JSCANX, JSCANY
28: 7060 format(1X,' SCAN MODE-X (SCAN(1)) : ',I5,
29: &          ' SCAN MODE-Y (SCAN(2)) : ',I5)
30:       write(IPR,7080) STYLE, STYLE2
31: 7080 format(1X,' DRAWING STYLE(1) : ',I5,
32: &          ' DRAWING STYLE(2) : ',I5)
33:       write(IPR,'(lx,' DISPLAY REFRESH INTERVAL : ',I5)' INTREFRESH
34:       write(IPR,7100) LEVEL, JZREVERSE, JGFLOOD
35: 7100 format(1X,' NEST LEVEL TO DRAW : ',I5,
36: &          ' ZONE SEARCH ORDER : ',I5/1X,
37: &          ' PARTILCE FLOOD MODE : ',I5)
38:       write(IPR,7120) (GRANGE(I),I=1,6)
39: 7120 format(1X,' COORDINAME RANGE : '/5X,1P,E12.5,'< X <',E12.5/5X,
40: &          E12.5,'< Y <',E12.5/5X,E12.5,'< Z <',E12.5)
41:       return
42:       end
43: C
44: C -----
45: C .... save drawing parameters
46: C
47: C saved parameter set are marked with IDSAVE
48: C -----
49: C
50:       subroutine SAVEPARAM( IPR,      IDSAVE,TITLGR,PAPER, XMAX, YMAX,
51:       &                     JSCANX, JSCANY,SPTYP, STYLE, STYLE2,LEVEL,
52:       &                     INTREFRESH, OFRAME,OVERSION,  OLOSTSYM,
53:       &                     MCELL,   IMCELL,JZREVERSE,  GRANGE,JGFLOOD )
54: C
55: C ... arguments
56:       character*(*) TITLGR
57:       real*8 PAPER(10), XMAX, YMAX
58:       real*8 GRANGE(6)
59:       integer JSCANX, JSCANY, SPTYP, STYLE, STYLE2, LEVEL, INTREFRESH
60:       integer OFRAME, OVERSION, OLOSTSYM
61: C
62: C --- saved
63: C
64:       parameter( MSET = 128 )
65:       common /SCENES0/ NSET, ISET, IDSETS(MSET)

```

```

66:       common /SCENES1/ GPAPER, GXMAX, GYMAX, GGRANGE, GJSCANX,
67:       &             GJSCANY, GSPTYP, GSTYLE, GSTYLE2, GLEVEL,
68:       &             GINTREFRESH, GOFRAME, GOVERSION,
69:       &             GOLOSTSYM, GMCELL, GIMCELL, GJZREVERSE,
70:       &             GJGFLOOD
71:       common /SCENES2/ GTITLGR
72:
73:       real*8 GPAPER(10,MSET), GXMAX(MSET), GYMAX(MSET)
74:       real*8 GGRANGE(6,MSET)
75:       integer GJSCANX(MSET), GJSCANY(MSET), GSPTYP(MSET), GSTYLE(MSET),
76:       &         GSTYLE2(MSET), GLEVEL(MSET), GINTREFRESH(MSET),
77:       &         GOFRAME(MSET), GOVERSION(MSET), GOLOSTSYM(MSET),
78:       &         GMCELL(MSET), GIMCELL(MSET), GJZREVERSE(MSET),
79:       &         GJGFLOOD(MSET)
80:       character*72 GTITLGR(MSET)
81: C
82: C -----
83: C
84:       if ( IDSAVE.eq.0 ) then
85:         NSET = 0
86:         do 100 I = 1, MSET
87:           IDSETS(I) = -1
88:         100 continue
89:       end if
90: C
91:       if ( NSET.ge.MSET ) then
92:         write(IPR,*) '!!! save space is full.',
93:         &          ' you can save no more scenes'
94:         return
95:       end if
96: C
97: C ... if IDSAVE is negative, assign an ID automatically.
98: C
99:       if ( IDSAVE.lt.0 ) then
100:         IDSAVE = NSET + 1
101:         continue
102:       110 do 120 I = 1, NSET
103:         if ( IDSETS(I).eq.IDSAVE ) then
104:           IDSAVE = IDSAVE + 1
105:           go to 110
106:         end if
107:       120 continue
108:       end if
109: C
110:       write(IPR,*) ' Save parameter set with ID <', IDSAVE, '>'
111: C
112:       if ( IDSAVE.gt.0 ) then
113:         do 130 II = 1, NSET
114:           if ( IDSAVE.eq.IDSETS(II) ) go to 140
115:         130 continue
116:         NSET = NSET + 1
117:         II = NSET
118:       140 continue
119:       else if ( NSET.eq.0 ) then
120:         NSET = 1
121:         II = NSET
122:       else
123:         write(IPR,*) '!!! you can''t change scene ID 0.'
124:         return
125:       end if
126: C
127:       ISET = II
128: C
129:       IDSETS(II) = IDSAVE
130:       do 150 I = 1, 10

```

src/cgview/cgparam.f

```

131:      GPAPER(I,II)   = PAPER(I)
132: 150 continue
133:      GXMAX(II)      = XMAX
134:      GYMAX(II)      = YMAX
135:      GJSCANX(II)    = JSCANX
136:      GJSCANY(II)    = JSCANY
137:      GSPTYP(II)     = SPTYP
138:      GSTYLE(II)      = STYLE
139:      GSTYLE2(II)     = STYLE2
140:      GLEVEL(II)      = LEVEL
141:      GINTREFRESH(II) = INTREFRESH
142:      GOFRAME(II)     = OFRAME
143:      GOVERSION(II)   = OVERSION
144:      GOLOSTSYM(II)   = OLOSTSYM
145:      GMCELL(II)      = MCELL
146:      GIMCELL(II)     = IMCELL
147:      GTITLGR(II)     = TITLGR
148:      GJZREVERSE(II)  = JZREVERSE
149:      GJGFLOOD(II)    = JGFLOOD
150:      do 160 I = 1, 6
151:          GGRANGE(I,II) = GRANGE(I)
152: 160 continue
153: C
154:      return
155:      end
156: C
157: C-----
158: C .... restore drawing parameters
159: C
160: C      mode = 'RESTORE' : restore a set whose ID is IMODE
161: C      mode = 'BACK'   : restore (ISET - IMODE)'th set
162: C      mode = 'FORWARD' : restore (ISET + IMODE)'th set
163: C
164: C      idrestore : ID of restored parameter set.
165: C-----
166: C
167:      subroutine RESTOREPARAM( IPR,  MODE,  IMODE, IDRESTORE,
168:      &                        TITLGR,  PAPER, XMAX,  YMAX,  JSCANX,
169:      &                        JSCANY,  SPTYP, STYLE, STYLE2, LEVEL,
170:      &                        INTREFRESH, OFRAME, OVERSION, GOLOSTSYM,
171:      &                        MCELL,   IMCELL, JZREVERSE, GRANGE,
172:      &                        JGFLOOD )
173: C
174: C ... arguments
175:      character*(*) MODE
176:      character*(*) TITLGR
177:      real*8 PAPER(10), XMAX, YMAX
178:      real*8 GRANGE(6)
179:      integer JSCANX, JSCANY, SPTYP, STYLE, STYLE2, LEVEL, INTREFRESH
180:      integer OFRAME, OVERSION, OLOSTSYM
181: C
182: C --- saved
183: C
184:      parameter( MSET = 128 )
185:      common /SCENES0/ NSET, ISET, IDSETS(MSET)
186:      common /SCENES1/ GPAPER, GXMAX, GYMAX, GGRANGE, GJSCANX,
187:      &      GJSCANY, GSPTYP, GSTYLE, GSTYLE2, GLEVEL,
188:      &      GINTREFRESH, GOFRAME, GOVERSION, GOLOSTSYM,
189:      &      GMCELL, GIMCELL, GJZREVERSE,
190:      &      GJGFLOOD
191:      common /SCENES2/ GTITLGR
192:
193:      real*8 GPAPER(10,MSET), GXMAX(MSET), GYMAX(MSET)
194:      real*8 GGRANGE(6,MSET)
195:      integer GJSCANX(MSET), GJSCANY(MSET), GSPTYP(MSET), GSTYLE(MSET),

```

```

196:      &      GSTYLE2(MSET), GLEVEL(MSET), GINTREFRESH(MSET),
197:      &      GOFRAME(MSET), GOVERSION(MSET), GOLOSTSYM(MSET),
198:      &      GMCELL(MSET), GIMCELL(MSET), GJZREVERSE(MSET),
199:      &      GJGFLOOD(MSET)
200:      character*72 GTITLGR(MSET)
201: C
202: C -----
203: C
204:      II      = 0
205: C
206:      if ( MODE.eq.'GET' ) then
207:          do 100 II = 1, NSET
208:              if ( IMODE.eq.IDSETS(II) ) go to 110
209: 100          continue
210:              write(IPR,*) '!!! parameter set with ID <', IMODE, '>',
211:      &                  ' is not stored.'
212:              return
213: 110          ISET      = II
214:              else if ( MODE.eq.'FORWARD' ) then
215:                  ISET      = MIN(NSET,ISET+IMODE)
216:              else if ( MODE.eq.'BACK' ) then
217:                  ISET      = MAX(1,ISET-IMODE)
218:              end if
219: C
220:      IDRESTORE      = IDSETS(ISET)
221: C
222:      write(IPR,*) ' Restore parameter set with ID <', IDRESTORE, '>'
223: C
224:      II      = ISET
225:      do 120 I = 1, 10
226:          PAPER(I)      = GPAPER(I,II)
227: 120          continue
228: C
229:      XMAX      = GXMAX(II)
230:      YMAX      = GYMAX(II)
231:      JSCANX    = GJSCANX(II)
232:      JSCANY    = GJSCANY(II)
233:      SPTYP     = GSPTYP(II)
234:      STYLE     = GSTYLE(II)
235:      STYLE2    = GSTYLE2(II)
236:      LEVEL     = GLEVEL(II)
237:      INTREFRESH = GINTREFRESH(II)
238:      OFRAME    = GOFRAME(II)
239:      OVERSION  = GOVERSION(II)
240:      OLOSTSYM  = GOLOSTSYM(II)
241:      MCELL     = GMCELL(II)
242:      IMCELL    = GIMCELL(II)
243:      TITLGR    = GTITLGR(II)
244:      JZREVERSE = GJZREVERSE(II)
245:      JGFLOOD   = GJGFLOOD(II)
246:      do 130 I = 1, 6
247:          GRANGE(I) = GGRANGE(I,II)
248: 130          continue
249: C
250:      return
251:      end

```

src/cgview/cgvinit.f

```
1:      subroutine CGVINIT
2: C=====
3: C CGVIEW default parameter initialization
4: C Called in MAIN
5: C
6: C=====
7:      include 'INC/_MVIEW'
8: C
9:      do I = 1, MPRNCOM
10:         PRNCOM(I) = ' '
11:      end do
12: C
13:      PRNNAM = 'cgvimg'
14: C
15:      IDENT = 0
16:      MGGLST = 20
17: C
18:      return
19: end
```


src/cgview/chkenv.f

```
1:      subroutine CHKENV( PROG, MLIMIT )
2:
3:      C/#!/IF .NOT.NOGETENV
4:
5:      C=<CGVIEW>=====
6:      C Purpose: Check environment variables
7:      C=====
8:      C argument :
9:      C
10:     C i prog : code name ( "CGVIEW" )
11:     C
12:     C=====
13:     C/#!/IF MS_VISUAL
14:     C
15:     C ... This interface block is for CUTIL & MS-Visual tools. ...
16:     C
17:     * interface
18:     *      subroutine FUNBUF()
19:     *CDEC$ ATTRIBUTES C :: FUNBUF
20:     *      end subroutine
21:     *      end interface
22:     C/#!/ENDIF
23:     C
24:     include '../shared/INC/_IOUNIT'
25:     C
26:     include 'INC/_MVIEW'
27:     C
28:     C ... external data ...
29:     C
30:     character*(*) PROG
31:     C
32:     C ... local data ...
33:     C
34:     character*64 VAR
35:     character*128 STRING
36:     character*20 NUMS
37:     C
38:     C-----
39:     C
40:     C ==== check CGV_MEM : --> MLIMIT
41:     C
42:     C CGVIEW memory size in word if dynamic allocation is possible
43:     C
44:     STRING = ' '
45:
46:     call ENVGET( 'CGV_MEM', STRING )
47:
48:     if( STRING.ne.' ' ) then
49:       write(IPR,*) '==(CHKENV) CGV_MEM <',
50:     & STRING(:ICLEN2(STRING)), '>'
51:       NUMS = STRING(:ICLEN2(STRING))
52:       ML0 = 0
53:       read(NUMS, '(BN,I20)') ML0
54:       if ( ML0.gt.0 ) then
55:         MLIMIT = ML0
56:         write(IPR,*) '== Memory size is set to : ', MLIMIT
57:       end if
58:     end if
59:     C-----
60:     C
61:     C ==== check MVPUNBUF : set standard output unbufferd
62:     C ==== check MVP_UNBUF : set standard output unbufferd
63:     C ("line buffered" saying exactly )
64:     C Other than null or blank should be defined for the variable
65:     C to activate it.
66:     C
67:     STRING = ' '
68:
69:     call ENVGET( 'MVPUNBUF', STRING )
70:     if( STRING.eq.' ' ) then
71:       call ENVGET( 'MVP_UNBUF', STRING )
72:     end if
73:     if( STRING.ne.' ' ) then
74:       call FUNBUF()
75:       write(IPR,*) '==(CHKENV) Standard output is line buffered.'
76:     end if
77:     C-----
78:     C
79:     C ==== check CGV_PRINT? : image printout command templates
80:     C
81:     STRING = ' '
82:     call ENVGET( 'CGV_PRINT1', STRING )
83:     if( STRING.ne.' ' ) then
84:       PRNCOM(1) = STRING
85:     end if
86:     STRING = ' '
87:     call ENVGET( 'CGV_PRINT2', STRING )
88:     if( STRING.ne.' ' ) then
89:       PRNCOM(2) = STRING
90:     end if
91:     STRING = ' '
92:     call ENVGET( 'CGV_PRINT3', STRING )
93:     if( STRING.ne.' ' ) then
94:       PRNCOM(3) = STRING
95:     end if
96:     STRING = ' '
97:     call ENVGET( 'CGV_PRINT4', STRING )
98:     if( STRING.ne.' ' ) then
99:       PRNCOM(4) = STRING
100:    end if
101:    C
102:    C ==== check CGV_PRBASE : image printout name-base to replace
103:    C %f keyword in print command template.
104:    C
105:    STRING = ' '
106:    call ENVGET( 'CGV_PRBASE', STRING )
107:    if( STRING.ne.' ' ) then
108:      PRNNAM = STRING
109:    end if
110:    C/#!/ENDIF
111:    C
112:    return
113:  end
```

src/cgview/ckdefsrc.f

```
1:      subroutine CKDEFSRC( IOW,   NDEFSRC,      NINPZ, NMEMO, IZNAM )
2: C=====
3: C purpose: check couurence of defered search and output message
4: C=====
5:      integer NDEFSRC(NINPZ)
6:      character*12 IZNAM(NINPZ)
7: C
8:      parameter( MZLINE   = 5 )
9:      integer ITEMP(MZLINE)
10: C
11:      KK      = 0
12:      do I = 1, NINPZ
13:          KK      = KK + NDEFSRC(I)
14:      end do
15:      if ( KK.eq.0 ) return
16: C
17:      write(IOW,7000) KK, NMEMO
18: 7000 format(1X,'>> Ineffetive "deferred" zone search required',I6,
19: &          ' times.'/1X,'>>> You''d better increasing NMEMO (=',I4,
20: &          ' or modify definition of the following'
21: &          /1X,'>>> input-zones (ignore this message '
22: &          'if it contains reflector virtual material).')
23: C
24:      II      = 0
25:      do I = 1, NINPZ
26:          if ( NDEFSRC(I).ne.0 ) then
27:              II      = II + 1
28:              ITEMP(II) = I
29:          end if
30:          if ( II.eq.MZLINE .or. (I.eq.NINPZ.and.II.gt.0) ) then
31:              write(IOW,7020)
32:              &          (ITEMP(K), IZNAM(ITEMP(K))(:ICLEN2(IZNAM(ITEMP(K))))),
33:              &          K=1, II)
34:          end if
35:      end do
36: 7020 format(1X,'>>> input-zone ',5(I5,'<',A,'>:'))
37:      return
38:      end
```

src/cgview/cnum.f

```
1:      function CNUM(AA)
2:      character*4 AA, CNUM
3:      CNUM    = AA
4:      return
5:      end
```

SAE

src/cgview/dmblk.f

```
1:      subroutine DMBLCK( VNAME )
2: C=====
3: C  PURPOSE: skip a block ( dummy read )
4: C  CALLED IN: INTRO
5: C  CALLS:
6: C=====
7:      character*(*) VNAME
8:      character*72 LINE
9:      INCLUDE '../shared/INC/_IUNIT'
10: c
11:      I1 = 5
12: c
13: 100 continue
14: c
15:      read(IQIN,'(a)',end=900) LINE
16: c
17:      if ( LINE(1:5).ne.'$END ' ) then
18:          if ( LINE(1:1).eq.'%' ) call PARA( LINE, IPR )
19:          go to 100
20:      end if
21: c
22:      return
23: c
24: 900 write(IPR,*) 'XXX End of file during skipping block <',vname,>'
25:      stop 888
26: c
27:      end
```

src/cgview/dmsour.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine DMSOUR
3: C=====
4: C  PURPOSE: SKIP SOURCE-DATA BLOCK ( DUMMY READ )
5: C  CALLED IN: INTRO
6: C  CALLS:
7: C=====
8: C    CHARACTER*80 TITLE
9: C    character*72 TITLE
10: C    real DUMMY(50)
11: C    integer IDUM(50)
12: C    equivalence(DUMMY,IDUM)
13: C    INCLUDE '../shared/INC/_IOUNIT'
14: C
15: C    !1      = 5
16: C
17: C    100 continue
18: C
19: C    read(IOIN,7000) TITLE
20: C    7000 format(A)
21: C
22: C    if ( TITLE(1:5).ne.'$END ' ) then
23: C
24: C        if ( TITLE(1:1).eq.'%' ) call PARA( TITLE, IPR )
25: C
26: C        go to 100
27: C
28: C    end if
29: C
30: C    return
31: C    end
```

src/cgview/dmxsec.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine DMXSEC( VNAME )
3:
4: C=====
5: C  PURPOSE: INPUT CROSS-SECTION RELATED DATA ( DUMMY READ )
6: C  CALLED IN: INTRO
7: C  CALLS:
8: C=====
9:
10:   character*(*) VNAME
11:
12: C   CHARACTER*80 TITLE
13:
14:   character*72 TITLE
15:   real DUMMY(50)
16:   integer IDUM(50)
17:   equivalence(DUMMY,IDUM)
18:
19:   INCLUDE '../shared/INC/_IOUNIT'
20: C
21:   I1      = 5
22:   TITLE = ' '
23: C
24:   if ( VNAME(1:6).eq.'$CROSS' ) then
25:     MVP    = 1
26:   else
27:     read(IOIN,1020) TITLE
28: 1020 format(A72)
29:     MVP    = 0
30:     if ( TITLE(1:72).eq.' ' ) MVP = 1
31:     if ( TITLE(1:1).eq.'&' ) MVP = 1
32:     if ( TITLE(1:1).eq.'*' ) MVP = 1
33:     if ( TITLE(1:1).eq.'%' ) MVP = 1
34:   end if
35: C
36:   if ( MVP.eq.0 ) then
37: C
38:     write(IPR,*) ' == SKIP Cross section block as GMVP type.'
39: C
40: C---- GMVP
41: C
42: C..... READ DATA X-1 .....
43: C
44:     read(IOIN,*) NGP, NDS, NGG, NDSG, INGP, ITBL, ISGG, NMED,
45:   &     NELEM, NMIX, NCOEF, NSCT, ISTAT, IDDX
46:
47: C           5  10  15   20   25   30   35   40   45   50   55
48: C
49: C..... READ DATA X-2 .....
50: C
51:     read(IOIN,*) IRDSG, ISTR, IFMU, IMOM, IPRIN, IPUN, IDTF,
52:   &     IXTAPE, JXTAPE
53:
54:     if ( IDDX.ne.0 ) then
55:       read(IOIN,*) (DUMMY(I),I=1,NSCT+1)
56:     end if
57:
58:     NEC      = 0
59:
60:     if ( IXTAPE.gt.0 ) then
61:       NEC      = NELEM*NCOEF
62: C
63: C..... READ DATA X-3 .....
64: C
65:     read(IOIN,*) (IDUM(I),I=1,NEC)

```

```

66: C
67:   end if
68: C
69:   if ( IXTAPE.ge.0 ) then
70:
71:     do 100 I = 1, NEC
72:       if ( IDUM(I).lt.0 ) read(IOIN,*) MAT, ST, SP, SC, SF, SNF
73: 100   continue
74:
75:     do 110 I = 1, NMIX
76:       read(IOIN,*) MIX, NEL, RHO
77: 110   continue
78:
79:   end if
80: C
81:   else
82: C---- MVP
83: CM 5000 IF(TITLE(1:4).EQ.'END ') GOTO 5100
84:
85: 120   continue
86: C
87:     write(IPR,*)
88:   &     ' == SKIP Cross section block as MVP or new GMVP type.'
89:     write(IPR,*) '<',TITLE(1:72),>'
90: C
91:     if ( TITLE(1:1).eq.'%' ) then
92:       call PARA( TITLE, IPR )
93:     end if
94:
95:     if ( TITLE(1:4).eq.'END ' ) go to 130
96:     if ( TITLE(1:5).eq.'$END ' ) go to 130
97:     read(IOIN,1020,end=900) TITLE
98:     go to 120
99:   end if
100: C
101: 130   continue
102: C
103:     return
104: C
105: 900 write(6,*) 'XXX unexpected end of input data in skipping',
106:   & ' cross section block.'
107:   stop 888
108:   end

```

src/cgview/dumpbk.f

```

1:      subroutine DUMPBK( N1,N2,IPRT,CHAR,H,IH)
2: C=====
3: C PURPOSE : PRINTOUT OF PARTICLE BANK. (FOR DEBUG ONLY)
4: C      (BANK # N1 TO N2 )
5: C      WHEN N1 = 0 : ALL SURVIVING PARTICLE.
6: C      N1 = -1 : ALL PARTICLES IN BANK.
7: C CALLED IN : VARIOUS ROUTINES
8: C=====
9: CCC  INCLUDE '../shared/INC/_ARRAY'
10:     character*(*) CHAR
11:     real H(*)
12:     integer IH(*)
13: C
14:     INCLUDE 'INC/_XBANK'
15:     INCLUDE 'INC/_FLAGS'
16:     INCLUDE '../shared/INC/_SIZES'
17:     INCLUDE '../shared/INC/_PGFOM'
18:     INCLUDE 'INC/_MVIEW'
19:     INCLUDE 'INC/_STACK'
20: C
21: C ..... COMMENT .....
22:     if ( CHAR.ne.' ' ) write(IPRT,'(/1X,A)') CHAR
23: C
24: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP,
25: C
26:     call DUMPXX( N1,N2,IPRT, H(LXXX),H(LYYY),H(LZZZ),H(LAAA),H(LBBB),
27:     & H(LCCC), H(LIZZ) )
28: C
29: C === LATTICE GEOMETRY ===
30: C ..... COORDINATES, DIRECTION, WEIGHT, ZONE#, GROUP, LEVEL, POS, CRS
31: C
32:     if ( JLATT.ne.0 ) then
33:         ML = 25
34:         do 110 K = 1, NEST
35:             write(IPRT,7000) K
36:             do 100 L = N1, N2, ML
37:                 L2 = MIN(L+ML-1,N2)
38:                 K1 = NBANK*(K-1) + L - 1
39:                 K2 = NBANK*(K-1) + L2 - 1
40:                 write(IPRT,7020) ' --- ', (I,I=L,L2)
41:                 write(IPRT,7020) 'LEVL ', (IH(LLEVL+I),I=K1,K2)
42:                 write(IPRT,7020) 'LZZ ', (IH(LLZZ+I),I=K1,K2)
43:                 write(IPRT,7020) 'LPOS ', (IH(LLPOS+I),I=K1,K2)
44:                 write(IPRT,7020) 'LCRS ', (IH(LLCRS+I),I=K1,K2)
45:             100 continue
46:         110 continue
47:     end if
48: C
49: 7000 format(/1X,'      LEVEL ',I2,'          ',10X:)
50: 7020 format(1X,A5,25I5)
51: C
52:     return
53:     end
54: C
55: C ===== AUXILIARY SUBROUTINE =====
56: C
57:     subroutine DUMPXX( N1, N2, IPRT, XXX,   YYY,   ZZZ,   AAA,   BBB,
58:     & CCC,   IZZ )
59:     real*8 XXX(*), YYY(*), ZZZ(*), AAA(*), BBB(*), CCC(*)
60:     integer IZZ(*)
61:     write(IPRT,7000)
62:     & (I,XXX(I),YYY(I),ZZZ(I),AAA(I),BBB(I),CCC(I),IZZ(I),I=N1,
63:     & N2)
64: 7000 format(/1X,'      # ',XXX      ',YYY      ',ZZZ      ',
65:     & 'AAA      ',BBB      ',CCC      ',IZZ '//

```

```

66:     & (1X,I5,1X,1P,6E11.4,I5))
67:     return
68:     end

```

src/cgview/dumpst.f

```
1:      subroutine DUMPST( CHAR, IPRT, H,IH )
2: C=====
3: C  PURPOSE : PRINTOUT OF STACK DATA. (FOR DEBUG ONLY)
4: C  CALLED IN : VARIOUS ROUTINES
5: C=====
6:      real H(*)
7:      integer IH(*)
8: C
9:      character*(*) CHAR
10:     INCLUDE 'INC/_FLAGS'
11:     INCLUDE 'INC/_STACK'
12:     INCLUDE '../shared/INC/_SIZES'
13:     INCLUDE 'INC/_XBANK'
14:     INCLUDE '../shared/INC/_PGEOM'
15:     INCLUDE 'INC/_MVIEW'
16: C
17: C ..... COMMENT .....
18:     write(IPRT,'(/1X,A)') CHAR
19: C
20: C ..... FLIGHT STACK ....
21: C
22:     ML      = 20
23:     write(IPRT,7000) 'NFFL ', (IH(LNFFL+I),I=0,NZONE)
24:     MM      = IH(LNFFL+NZONE)
25:     do 100 K1 = 1, MM, ML
26:         K2      = MIN(K1+ML-1,MM) - 1
27:         write(IPRT,7020) 'LSFFL', (IH(LLSFFL+I),I=K1-1,K2)
28:         write(IPRT,7020) 'IZFFL', (IH(LIZFFL+I),I=K1-1,K2)
29:         write(IPRT,7020) 'IZZ ', (IH(LIZZ+IH(LLSFFL+I)-1),I=K1-1,K2)
30:     100 continue
31: C
32: C ..... SEARCH STACK ....
33: C
34:     write(IPRT,7000) 'NNXT ', (IH(LNNXT+I),I=0,NZONE)
35:     MM      = IH(LNNXT+NZONE)
36:     do 110 K1 = 1, MM, ML
37:         K2      = MIN(K1+ML-1,MM) - 1
38:         write(IPRT,7020) 'LSSRC', (IH(LLSSRC+I),I=K1-1,K2)
39:         write(IPRT,7020) 'IZNXT', (IH(LIZNXT+I),I=K1-1,K2)
40:         write(IPRT,7040) 'IZZ ', (IH(LIZZ+IH(LLSSRC+I)-1),I=K1-1,K2)
41:     110 continue
42: C
43: C ..... REFLECTION STACK
44: C
45:     if ( JREFL.ne.0 ) then
46:         write(IPRT,7000) 'NBREF', (IH(LNBREF+I),I=0,NREFS)
47:         MM      = IH(LNBREF+NREFS)
48:         do 120 K1 = 1, MM, ML
49:             K2      = MIN(K1+ML-1,MM) - 1
50:             write(IPRT,7020) 'LSREF', (IH(LLSREF+I),I=K1-1,K2)
51:             write(IPRT,7020) 'IZREF', (IH(LIZREF+I),I=K1-1,K2)
52:             write(IPRT,7040) 'ISREF', (IH(LISREF+I),I=K1-1,K2)
53:         120 continue
54:     end if
55: C
56: C ..... LATTICE SEARCH STACK .....
57: C
58:     if ( JLATT.ne.0 ) then
59:         write(IPRT,7000) 'NXLT ', (IH(LNXLT+I),I=0,NLBZ)
60:         MM      = IH(LNXLT+NLBZ)
61:         do 130 K1 = 1, MM, ML
62:             K2      = MIN(K1+ML-1,MM) - 1
63:             write(IPRT,7020) 'LSLAT', (IH(LLSLAT+I),I=K1-1,K2)
64:             write(IPRT,7040) 'IZLAT', (IH(LIZLAT+I),I=K1-1,K2)
65:         130 continue
66:     end if
67: C
68: C
69: 7000 format(1X,'* ',A6,': ',20I5/(1X,9X,20I5))
70: 7020 format(1X,A5,20I5)
71: 7040 format(1X,A5,20I5/)
72:     return
73:     end
```


src/cgview/flagin.f

```
1:      subroutine FLAGIN
2: C=====
3: C=====
4:      INCLUDE 'INC/_FLAGS'
5: C
6:      include 'INC/_KPIDS'
7: C
8:      do 110 I=1,MKPAR
9:          JKPAR(I) = 0
10: 110 continue
11:      JKPAR(KPNEUT) = 0
12:      JNEUT      = 1
13:      JPHOT      = 0
14:      JGAMM      = 0
15:
16:      JBRLT      = 1
17:      JIMPT      = 0
18:      JWWND      = 0
19:      JPSTR      = 0
20:      JWTIM      = 0
21:
22:      JALLZ      = 0
23:      JONEZ      = 1
24:      JLATT      = 0
25:      JHLAT      = 0
26:      JTLLT      = 0
27:      JFISX      = 0
28:      JSTGR      = 0
29:      JVMNT      = 0
30:      JMCHK      = 0
31:      do 100 I = 1, MAXJDB
32:          JDEBG(I)      = 0
33: 100 continue
34: C
35: C
36:      return
37:      end
```

src/cgview/flichk.f

```
1:      subroutine FLICHK( XPLT,  YPLT, GDIST,
2:      &                  XMAX,  YMAX,  NZONE, ILOOP, N1,
3:      &                  LSFFL, NFFL,  IZFFL, NDEAD )
4: C=====
5: C  check termination of drawing scanlines
6: C=====
7:      real*8 XPLT(*), YPLT(*), XMAX, YMAX
8:      real*8 GDIST(*)
9:      integer LSFFL(*), NFFL(*), IZFFL(*)
10: C
11:      N2      = NFFL(NZONE+1)
12:      ISAFE   = N1 - 1
13:
14:      do 100 J = N1, N2
15:
16:          IZ   = IZFFL(J)
17:      CS      if ( (ILOOP.eq.1.and.YPLT(LSFFL(J)).ge.YMAX)
18:      CS  &    .or. (ILOOP.gt.1.and.XPLT(LSFFL(J)).ge.XMAX) ) then
19:
20:          if ( GDIST(LSFFL(J)).le.0.0d0) then
21:              NFFL(IZ) = NFFL(IZ) - 1
22:          else
23:              ISAFE   = ISAFE + 1
24:              LSFFL(ISAFE) = LSFFL(J)
25:              IZFFL(ISAFE) = IZFFL(J)
26:          end if
27:      100 continue
28:      IDEAD   = N2 - ISAFE
29:      NFFL(NZONE+1) = ISAFE
30:      NDEAD   = NDEAD + IDEAD
31:      return
32:      end
```

src/cgview/flione.f

```

1:      subroutine FLIONE(IOW, IRAND,
2:      N      NBANK, NZONE, NSDA, NZDA, NCELL,
3:      N      NDEAD, NREG,          DINF,DEPS, NLATT,
4:      N      NUNV, NSPACE,NKTCSP, NEST, NLBZ, MSGSLST,
5:      B      XXX ,YYY ,ZZZ ,AAA ,BBB ,CCC ,
6:      B      IZZ ,          LEVL ,LZZ ,LPOS ,
7:      B      LCRS ,KLSF ,IBSPC,
8:      B      DBNK,KDBNK,MDBNK, IBNK, KIBNK,MIBNK,
9:      S      LSFFL,NFFL ,IZFFL ,
10:     S      LSSRC ,NNXT ,IZNXT ,LSDED ,
11:     G      SDA ,KZMAT ,KZREG ,KSPSU,KTCSP,KZDA ,KZAA ,
12:     G      DALT ,KDALT ,NVLAT ,SZLAT ,IBPAT ,IPCEL,
13:     G      KSLAT ,LTYP ,MLBZZ ,KZLBZ,KCELL ,ICTYP ,KLATT ,CELSZ ,
14:     G      PNND ,KNND ,NPNND ,PPPF ,KPPF ,NPPPF ,
15:     T      XPLT ,YPLT ,
16:     T      AAAG ,BBBG ,CCCG ,GDIST,
17:     T      PAPER ,ILOPF ,
18:     W      X ,Y ,Z ,AI ,BI ,CI ,
19:     W      W ,DI ,IGI ,IBP ,IFC ,R ,
20:     W      DLOC1 ,DLOC2 ,IKL ,IKL2 ,
21:     W      XUP ,YUP ,ZUP ,AUP ,BUP ,CUP,TI ,ISTG ,ILUP,ILST,
22:     W      XYZ1 ,ABC1 ,XYZ2 ,ABC2 ,DDI ,IBP0 ,IBP1 ,IBP2 ,KPLT ,JKSF,
23:     &      MZONE ,LEVEL ,MCELL ,IMCELL ,JCGVDB)
24: C
25: C====< CGVIEW >=====
26: C PURPOSE:   FREE-FLIGHT OR ONE-ZONE LOGIC
27: C CALLED IN: ACTION
28: C CALLS:     SPEAR1,LFRAME,RANU2
29: C=====
30: C
31: C   === INTER-STACK DATA FLOW ===
32: C
33: C   FREE-FLIGHT STACK  -----+----->  NEXT-ZONE-SEARCH STACK
34: C   (LSFFL,IZFFL,NFFL)      !      (LSSRC,IZNXT,NNXT)
35: C                           +----->
36: C                           !
37: C   (LOST)  +----->  DEAD PARTICLE STACK
38: C   CCCC      (LSDED,NDEAD)
39: C
40: C   === BANK DATA TO BE UPDATED ===
41: C
42: C   (XXX,YYY,ZZZ) : POSITION
43: C   (AAA,BBB,CCC) : DIRECTION      (IF HEXAGONAL-LATTICE EXISTS)
44: C   IZZ           :
45: C   LEVL          : LATTICE LEVEL (IF HEXAGONAL-LATTICE EXISTS)
46: C
47: C   === BANK DATA ADDED NEWLY ===
48: C
49: C   (NOTHING)
50: C
51: C   implicit real*8(D)
52: C   include 'INC/_FLAGS'
53: C
54: C   .... PARTICLE BANK .....
55: C
56: C   real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
57: C   & CCC(NBANK)
58: C   integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
59: C   & LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK)
60: C   integer IBSPC(NBANK,0:NEST)
61: C
62: C ... optional bank parameters
63: C
64: C   real*8 DBNK(NBANK,*)
65: C   integer KDBNK(0:MDBNK)

```

```

66:     integer IBNK(NBANK,*)
67:     integer KIBNK(0:MIBNK)
68: C
69: C   .... STACKS .....
70: C
71: C   integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSSRC(NBANK),
72: C   & NNXT(NZONE+1), IZNXT(NBANK), LSDED(NBANK)
73: C
74: C   .... GEOMETRY .....
75: C
76: C   real*8 SDA(NSDA), DALT(1), SZLAT(8,NLATT), CELSZ(6,1)
77: C   integer KZMAT(NZONE,2), KZREG(NZONE), KZDA(2,NZDA), KZAA(NZONE+1),
78: C   & KDALT(NLBZ), MLBZZ(NLBZ), LTYP(NLATT), IPLAT(1), KSLAT(1),
79: C   & KCELL(NZONE), ICTYP(NCELL), KLATT(1), NVLAT(4,1)
80: C   integer IPCEL(NCELL)
81: C   integer KSPSU(NUNV,0:NSPACE), KTCSP(NKTCSP)
82: C
83: C   ... Nearest Neighbour Distribution for STGM region.
84: C
85: C   real PNND(NPNND)
86: C   integer KNND(NPNND)
87: C
88: C   ... Partial Packing Fraction for STGM-region
89: C
90: C   real PPPF(NPPPF)
91: C   integer KPPF(NPPPF)
92: C
93: C   .... WORK AREA (LENGTH = NBANK) .....
94: C
95: C   real*8 X(NBANK), Y(NBANK), Z(NBANK), AI(NBANK), BI(NBANK),
96: C   & CI(NBANK), DI(NBANK), W(NBANK), DLOC1(NBANK),
97: C   & DLOC2(NBANK), XUP(NBANK), YUP(NBANK), ZUP(NBANK),
98: C   & AUP(NBANK), BUP(NBANK), CUP(NBANK)
99: C   real R(3*NBANK)
100: C
101: C   integer IGI(NBANK), IBP(NBANK), IFC(NBANK)
102: C   integer IKL(NBANK), IKL2(NBANK)
103: C
104: C   integer ISTG(NBANK)
105: C   integer ILST(NBANK)
106: C   integer ILUP(NBANK)
107: C   integer IBP0(NBANK), IBP1(NBANK), IBP2(NBANK)
108: C   integer KPLT(NLBZ+1)
109: C   integer JKSF(NLBZ)
110: C   real*8 XYZ1(NBANK,3), ABC1(NBANK,3)
111: C   real*8 XYZ2(NBANK,3), ABC2(NBANK,3)
112: C   real*8 DDI(NBANK)
113: C   real*8 TI(NBANK)
114: C
115: C   real*8 XPLT(NBANK), YPLT(NBANK)
116: C   real*8 AAAG(*),BBBG(*),CCCG(*),GDIST(*)
117: C
118: C   real*8 PAPER(*)
119: C
120: C   integer JCGVDB(*)
121: C
122: C   ... local data .....
123: C
124: C   integer IPZZ(2)
125: C   logical JSPECIAL
126: C
127: C   include '../shared/INC/_PMLATT'
128: C   include '../shared/INC/_SFLATT'
129: C
130: C-----

```

src/cgview/flione.f

```

131: C
132: C      LEVELA = ABS(LEVEL)
133: C
134: C
135: C      When LEVEL() < NEST, procedures in this routine is performed twice
136: C      for particles whose level.le.abs(LEVEL) (KLOOP=0), and others
137: C      for second time (KLOOP=1).
138: C
139: C      KLOOP = 0
140: C
141: C 100 continue
142: C
143: C .... GATHER VECTORS .....
144: C
145: C      NFF = 0
146: C      JK = 1
147: C
148: C      if ( ISLATT.ne.0.and.LEVELA.lt.NEST
149: C      & .and. ISLATT(KZMAT(MZONE,1)) ) then
150: C
151: C          if ( KLOOP.eq.0 ) then
152: C              do 110 I = 1, NFFL(NZONE+1)
153: C                  if ( IZFFL(I).eq.MZONE.and.LEVL(LSFFL(I)).lt.LEVELA )
154: C                      &
155: C                          then
156: C                              NFF = NFF + 1
157: C                              IBP(NFF) = LSFFL(I)
158: C                          end if
159: C                  continue
160: C              else
161: C                  do 120 I = 1, NFFL(NZONE+1)
162: C                      if ( IZFFL(I).eq.MZONE.and.LEVL(LSFFL(I)).ge.LEVELA )
163: C                          &
164: C                              then
165: C                                  NFF = NFF + 1
166: C                                  IBP(NFF) = LSFFL(I)
167: C                              end if
168: C                      continue
169: C                  end if
170: C              else
171: C                  JK = 0
172: C                  do 130 I = 1, NFFL(NZONE+1)
173: C                      if ( IZFFL(I).eq.MZONE ) then
174: C                          NFF = NFF + 1
175: C                          IBP(NFF) = LSFFL(I)
176: C                      end if
177: C                  continue
178: C              end if
179: C          do 140 I = 1, NFF
180: C              X(I) = XXX(IBP(I))
181: C              Y(I) = YYY(IBP(I))
182: C              Z(I) = ZZZ(IBP(I))
183: C              AI(I) = AAA(IBP(I))
184: C              BI(I) = BBB(IBP(I))
185: C              CI(I) = CCC(IBP(I))
186: C              IKL(I) = KLSF(IBP(I))
187: C              IKL2(I) = 0
188: C
189: C              DI(I) = DINF
190: C              ILST(I) = 0
191: C 140 continue
192: C -----
193: C .... Material #
194: C -----
195: C

```

```

196: C      MAT = KZMAT(MZONE,1)
197: C
198: C ... this is a base material (matrix) region of STG region.
199: C
200: C      JJJNND = 0
201: C      if ( KLOOP.eq.0.and.MAT.lt.0 ) then
202: C          if ( ISLATT(MAT).and.LTYP(LATTNM(MAT)).eq.10 ) then
203: C              MAT = KZMAT(MZONE,2)
204: C              JJJNND = 1
205: C          else
206: C              write(IOW,*)'XXX(FLIONE) Program error ? Invalid zone :',
207: C              & ' MZONE=',mzone,' KZMAT(MZONE,*) ',
208: C              & kzmat(MZONE,1),kzmat(MZONE,2),' LTYP ',LTYP(ABS(MAT))
209: C              stop 666
210: C          end if
211: C      end if
212: C
213: C      JSPECIAL = MAT.eq. - 1000 .or. MAT.eq. - 2000 .or. MAT.eq.
214: C      & - 3000 .or. MAT.eq. - 4000 .or. MAT.eq.-999
215: C
216: C .... CALCULATE DISTANCES TO ZONE BOUNDARY .....
217: C
218: C      if ( JFISX.eq.0 .or. JJJNND.eq.0 ) then
219: C
220: C          --- in CGVIEW , spearl is called with suraface(body) detection
221: C          mode, and TI(*) is used for working array here,
222: C
223: C          JSS = 1
224: C          call SPEAR1( MZONE, KZDA, KZAA, SDA, NFF, NBANK, JSS,
225: C          & X, Y, Z, AI, BI, CI, DI,
226: C          & DLOC1, DLOC2, IKL, IKL2, TI,
227: C          & DINF, DEPS )
228: C
229: C          ... "lost" particle index to "1"
230: C
231: C *VOCL LOOP,NOVREC
232: C          do 150 I = 1, NFF
233: C              if ( DI(I).eq.DINF ) then
234: C                  ILST(I) = 1
235: C              end if
236: C          150 continue
237: C          end if
238: C
239: C ----- HEXAGONAL LATTICE -----
240: C ... CALCULATE DISTANCE TO LATTICE FRAME & COORDINATES IN UPPER LEVEL .
241: C      IFC(I) = N/0 = CROSS THE OUTER FRAME (N=ZONE# IN UPPER LEVEL)/ NO
242: C      VALUE OF DI(I) MAY BE CHANGED.
243: C      ( IZS=1, IZE=1, LLS AS IFC, DSDA0 TO DSDA5 AS XUP TO CUP )
244: C
245: C      CCCC IF(JHLAT.NE.0.AND.ICTYP(KCELL(MZONE)).GE.2) THEN
246: C          IHF = 0
247: C          if ( KLOOP.eq.0.and.JFISX.eq.0.and.JHLAT.ne.0 ) then
248: C              if ( KCELL(MZONE).gt.0.and.ICTYP(KCELL(MZONE)).ge.2 ) then
249: C                  IHF = 1
250: C                  IPZZ(1) = 1
251: C                  IPZZ(2) = NFF + 1
252: C                  call LFRAME( IOW, JDEBG, NBANK, NEST, NLATT, NLBZ, NZONE,
253: C                  & DINF, IBP, IPZZ, 1, 1, DI, IFC,
254: C                  & XUP, YUP, ZUP, AUP, BUP, CUP,
255: C                  & LEVL, LZZ, LPOS, LCRS,
256: C                  & KZMAT, KDALT, DALT, NVLAT, SZLAT, CELSZ,
257: C                  & IPLAT, KLATT, KSLAT, LTYP, MLBZZ,
258: C                  & X, Y, Z, AI, BI, CI, R )
259: C          *VOCL LOOP,NOVREC
260: C          do 160 I = 1, NFF

```

src/cgview/flione.f

```

261: C
262: C      .... this case moves only one upper lattice level
263: C
264: C      ILUP(I) = LEVL(IBP(I)) - 1
265: C
266: C      ... lost in frame distance calculation
267: C
268: C      if ( DI(I).eq.DINF ) then
269: C          ILST(I) = ILST(I) + 2
270: C      end if
271: 160      continue
272: C      end if
273: C
274: C      ... distance to frames until current lattice level is
275: C      calculated if flight path count is zero.
276: C
277: C      else if ( KLOOP.eq.0.and.JFISX.ne.0
278: C      & .and.(KCELL(MZONE).gt.0.or.JJJNND.gt.0) ) then
279: C
280: C      << CGVIEW specific >> particle may fly in "-999" material
281: C
282: C      else if ( KLOOP.eq.0.and.JFISX.ne.0
283: C      & .and.
284: C      & ((MAT.ne.-999.and.
285: C      & KCELL(MZONE).gt.0.and.KCELL(MZONE).ne.KCELL(IMCELL))
286: C      & .or.JJJNND.gt.0) ) then
287: C          IHF = 1
288: C          KK = 0
289: C      *VOCL LOOP,NOVREC
290: C          do 170 I = 1, NFF
291: C              IFC(I) = 0
292: C              if ( IBNK(IBP(I),KIBNK(5)).eq.0 ) then
293: C                  KK = KK + 1
294: C              end if
295: C          170      continue
296: C              if ( KK.gt.0 ) then
297: C                  call DFRAME( IOW, MZONE, NFF, JDEBG, NBANK, NEST, NLATT,
298: C                  & NLBZ, NZONE, DINF, DEPS, X, Y, Z, AI, BI, CI, IBP,
299: C                  & DI, IFC, XUP, YUP, ZUP, AUP, BUP, CUP, ILUP, ILST,
300: C                  & LEVL, LZZ, LPOS, LCRS, DBNK, KDBNK, MDBNK, IBNK,
301: C                  & KIBNK, MIBNK, KZMAT, KZDA, KZAA, SDA, KCELL, KDALI,
302: C                  & DALI, NVLAT, SZLAT, CELSZ, IPLAT, KLAT, KSLAT,
303: C                  & LTP, MLBZZ, KZLBZ, KPLT, JKSP, IBP0, IBP1, IBP2,
304: C                  & XYZ1(1,1), XYZ1(1,2), XYZ1(1,3), ABC1(1,1),
305: C                  & ABC1(1,2), ABC1(1,3), XYZ2(1,1), XYZ2(1,2),
306: C                  & XYZ2(1,3), ABC2(1,1), ABC2(1,2), ABC2(1,3), DBI,
307: C                  & IKL, IKL2, DLOC1, DLOC2 )
308: C              end if
309: C
310: C      ... get frame distance and upper level postion etc. from
311: C      banked data
312: C
313: C      if ( NFF-KK.gt.0 ) then
314: C      *VOCL LOOP,NOVREC
315: C          do 180 I = 1, NFF
316: C              KL = IBNK(IBP(I),KIBNK(6)+LEVL(IBP(I))-1)
317: C              KI = KDBNK(4) + KL - 1
318: C              DDDD = DBNK(IBP(I),KI)
319: C              if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.DDDD.lt.DI(I) ) then
320: C                  IFC(I) = LZZ(IBP(I),KL)
321: C                  ILUP(I) = KL - 1
322: C                  K6 = KDBNK(6) + 6*(KL-1)
323: C                  AUP(I) = DBNK(IBP(I),K6+3)
324: C                  BUP(I) = DBNK(IBP(I),K6+4)
325: C                  CUP(I) = DBNK(IBP(I),K6+5)

```

```

326: C                  XUP(I) = DBNK(IBP(I),K6) - DDDD*AUP(I)
327: C                  YUP(I) = DBNK(IBP(I),K6+1) - DDDD*BUP(I)
328: C                  ZUP(I) = DBNK(IBP(I),K6+2) - DDDD*CUP(I)
329: C                  DI(I) = DDDD
330: C              end if
331: 180      continue
332: C          end if
333: C      end if
334: C
335: C      .... CHECK LOST PARTICLE .....
336: C
337: C          ISAFE = NFF
338: C          II = 0
339: C          do 190 I = 1, ISAFE
340: C              IGI(I) = 1
341: C              if ( ILST(I).ne.0.and..not.JSPECIAL ) then
342: C                  II = II + 1
343: C                  IGI(I) = -1
344: C              end if
345: C          190      continue
346: C
347: C      .... COUNT THE NUMBER OF FREE FLIGHT ....
348: C
349: C          if ( II.gt.0 ) then
350: C              write(IOW,*) ' (FLIONE) ', II, ' PARTICLES ARE LOST !! '
351: C              NDEAD = NDEAD + II
352: C
353: C      .... DELETE LOST PARTICLES .....
354: C          ISAFE = 0
355: C      *VOCL LOOP,NOVREC
356: C          do 200 I = 1, NFF
357: C              if ( IGI(I).gt.0 ) then
358: C                  ISAFE = ISAFE + 1
359: C                  IBP(ISAFE) = IBP(I)
360: C                  DI(ISAFE) = DI(I)
361: C                  IKL2(ISAFE) = IKL2(I)
362: C                  if ( IHF.ne.0 ) then
363: C                      ILUP(ISAFE) = ILUP(I)
364: C                      IFC(ISAFE) = IFC(I)
365: C                      XUP(ISAFE) = XUP(I)
366: C                      YUP(ISAFE) = YUP(I)
367: C                      ZUP(ISAFE) = ZUP(I)
368: C                      AUP(ISAFE) = AUP(I)
369: C                      BUP(ISAFE) = BUP(I)
370: C                      CUP(ISAFE) = CUP(I)
371: C                  end if
372: C              else
373: C          ....LOST
374: C              LSDDED(IBP(I)) = 2
375: C          end if
376: C          200      continue
377: C
378: C      .... GATHER UNLOST PARTICLES AGAIN .....
379: C          do 210 I = 1, ISAFE
380: C              X(I) = XXX(IBP(I))
381: C              Y(I) = YYY(IBP(I))
382: C              Z(I) = ZZZ(IBP(I))
383: C              AI(I) = AAA(IBP(I))
384: C              BI(I) = BBB(IBP(I))
385: C              CI(I) = CCC(IBP(I))
386: C          210      continue
387: C          end if
388: C
389: C-----
390: C      .... sampling of STG sphere position from

```

src/cgview/flione.f

```

391: C          Nearest Neighbour Distribution (NND)
392: C-----
393: C
394:       if ( JJJNND.ne.0 ) then
395:         MLT = LATTNM(KZMAT(MZONE,1))
396:         NN = ISAFE
397: C
398: C         ... ISTG (cell buffer zone of STG cell region) is returned
399: C         ... IKL (cell position in STG lattice) is returned
400: C         if sampled distance to STG is smaller than DI(I)
401: C
402: C         XUP,YUP,ZUP here is not a coordinate in "upper level" in this
403: C         case.
404: C
405: C         call SMPSTG( IOW, MLT, X, Y, Z, AI, BI, CI, DI, ISTG, IBP,
406: C         & NN, PNND, KNND, NNNND, IPLAT, KLATT, KSLAT, SZLAT,
407: C         & NIATT, CELSZ, IBCEL, PPPF, KPPF, NPPPF,
408: C         & IRAND, LEVL, DBNK, KDBNK, MDBNK, IBNK,
409: C         & KIBNK, MIBNK, NBANK, XUP, YUP, ZUP, IKL, IBP2, TI, R )
410: C
411: C         ... override frame distance flag
412: C
413:       do 220 I = 1, ISAFE
414:         if ( ISTG(I).ne.0 ) then
415:           IFC(I) = ISTG(I)
416:           AUP(I) = AI(I)
417:           BUP(I) = BI(I)
418:           CUP(I) = CI(I)
419:           ILUP(I) = LEVL(IBP(I))
420:         end if
421:       220 continue
422:     end if
423: C
424: C=====
425: C .... CALCULATE DISTANCE TO COLLISION POINT & COMPARE WITH DI(I) .....
426: C=====
427: C
428: C
429: C .... count up path flight counter here (for collided particle
430: C it is cleared to zero afterwards)
431: C
432:       KI5 = KIBNK(5)
433:       KI7 = KIBNK(7)
434:       if ( KI5.ne.0 ) then
435:         do 230 I = 1, ISAFE
436:           if ( IBNK(IBP(I),KI5).lt.0 ) then
437:             IBNK(IBP(I),KI5) = 1
438:           else
439:             IBNK(IBP(I),KI5) = IBNK(IBP(I),KI5) + 1
440:           end if
441:         230 continue
442:       end if
443: C
444:       INZ1 = NNXT(NZONE+1)
445: C
446: C ----- Free lattice frame or hexagonal lattice option -----
447: C
448:       if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
449: C
450:         INXT = ISAFE
451:         if ( IHF.eq.0 ) then
452:           do 240 I = 1, INXT
453:             LSSRC(INZ1+I) = IBP(I)
454:             IZNXT(INZ1+I) = MZONE
455:           240 continue

```

```

456:         NNXT(MZONE) = NNXT(MZONE) + INXT
457:       else
458:         IHHHH = 0
459:         do 250 I = 1, INXT
460:           LSSRC(INZ1+I) = IBP(I)
461:           if ( IFC(I).eq.0 ) then
462:             IZNXT(INZ1+I) = MZONE
463:           else
464:             IHHHH = IHHHH + 1
465:             IZNXT(INZ1+I) = IFC(I)
466:             IZZ(IBP(I)) = IFC(I)
467:             LEVL(IBP(I)) = ILUP(I)
468:             NNXT(IFC(I)) = NNXT(IFC(I)) + 1
469:             X(I) = XUP(I)
470:             Y(I) = YUP(I)
471:             Z(I) = ZUP(I)
472:             AI(I) = AUP(I)
473:             BI(I) = BUP(I)
474:             CI(I) = CUP(I)
475: C         ... NOT ON-SURFACE
476:           IKL2(I) = 0
477:         end if
478:       250 continue
479:       NNXT(MZONE) = NNXT(MZONE) + INXT - IHHHH
480:     end if
481: C
482:     NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
483: C
484: C     ... "cell position" LPOS is changed here for STG cell
485: C
486:     if ( IHF.ne.0.and.IHHHH.gt.0 ) then
487:       if ( JJJNND.ne.0 ) then
488:         *VOCL LOOP,NOVREC
489:         do 260 I = 1, ISAFE
490:           if ( IFC(I).ne.0.and.ISTG(I).ne.0 ) then
491:             LPOS(IBP(I),LEVL(IBP(I))) = IKL(I)
492:           end if
493:         260 continue
494:       end if
495:       if ( JTLT.ne.0.and.JJJNND.ne.0 ) then
496:         *VOCL LOOP,NOVREC
497:         do 270 I = 1, ISAFE
498:           if ( IFC(I).ne.0.and.ISTG(I).ne.0 ) then
499:             IBSPC(IBP(I),LEVL(IBP(I))) =
500:             & KTCSP(
501:             & KSPSU(KZREG(MZONE)),
502:             & IBSPC(IBP(I),LEVL(IBP(I))-1))+IKL(I))
503:           end if
504:         270 continue
505:       end if
506:     end if
507: C
508: C ---- NON-HEXAGONAL LATTICE ----
509: C
510:       else
511:         INXT = ISAFE
512:         do 280 I = 1, INXT
513:           LSSRC(NNXT(NZONE+1)+I) = IBP(I)
514:         280 continue
515: C
516:         do 290 I = 1, INXT
517:           IZNXT(NNXT(NZONE+1)+I) = MZONE
518:         290 continue
519:         NNXT(NZONE+1) = NNXT(NZONE+1) + INXT
520:         NNXT(MZONE) = NNXT(MZONE) + INXT

```

src/cgview/flione.f

```

521:      end if
522: C
523: C .... MOVE PARTICLES TO NEXT POINTS AND PUT VALUES IN BANK.....
524: C
525: C << CGVIEW >>
526: C
527: C Flight pathes in reflector material or "-999" material are
528: C given infinite flight distance but not tretaed as "lost"
529: C
530: *VOCL LOOP,NOVREC
531:   do 300 I = 1, ISAFE
532:     KLSF(IBP(I)) = IKL2(I)
533:     DI2 = DI(I)
534:     if ( DI(I).eq.DINF )then
535:       DI2 = GDIST(IBP(I))+100.0D0
536:     end if
537:     XXX(IBP(I)) = X(I) + AI(I)*DI2
538:     YYY(IBP(I)) = Y(I) + BI(I)*DI2
539:     ZZZ(IBP(I)) = Z(I) + CI(I)*DI2
540:     XPLT(IBP(I)) = XPLT(IBP(I)) + AAAG(IBP(I))*DI2
541:     YPLT(IBP(I)) = YPLT(IBP(I)) + BBBG(IBP(I))*DI2
542:     GDIST(IBP(I)) = GDIST(IBP(I)) - DI2
543:   300 continue
544: C
545: C ... reduce distance to frame in lattice levels
546: C
547:   if ( JFISX.ne.0 ) then
548:     MXLV = 0
549:     KK = 0
550:   *VOCL LOOP,NOVREC
551:     do 310 I = 1, ISAFE
552:       if ( IBNK(IBP(I),KIBNK(5)).ne.0 ) then
553:         MXLV = MAX(MXLV,LEVL(IBP(I)))
554:         KK = KK + 1
555:       end if
556:     310 continue
557:     if ( MXLV.gt.0.and.KK.gt.0 ) then
558:       KD4 = KDBNK(4)
559:       do 330 LV = 1, MXLV
560:   *VOCL LOOP,NOVREC
561:         do 320 I = 1, ISAFE
562:           if ( IBNK(IBP(I),KIBNK(5)).ne.0.and.LEVL(IBP(I)).ge.LV
563:             &
564:               ) then
565:             DBNK(IBP(I),KD4+LV-1) = DBNK(IBP(I),KD4+LV-1)
566:             &
567:               - DI(I)
568:           end if
569:         320 continue
570:       330 continue
571:     end if
572:   end if
573:   if ( IHF.ne.0 ) then
574:   *VOCL LOOP,NOVREC
575:     do 340 I = 1, ISAFE
576:       AAA(IBP(I)) = AI(I)
577:       BBB(IBP(I)) = BI(I)
578:       CCC(IBP(I)) = CI(I)
579:     340 continue
580:   end if
581: C
582: C .... DELETE PARTICLES FROM FLIGHT STACK .....
583: C
584: C 350 continue
585: C
586: C ==== KLOOP=1 means treatment of lattice frame zone as an ordinary
587: C zone when LEVEL < NEST.

```

```

586: C
587:   if ( KLOOP.eq.0.and.NFF.lt.NFFL(MZONE) ) then
588:     KLOOP = 1
589:     go to 100
590:   end if
591: C
592: C
593: C
594:   II = 0
595:   *VOCL LOOP,NOVREC
596:     do 360 N = 1, NFFL(NZONE+1)
597:       if ( IZFFL(N).ne.MZONE ) then
598:         II = II + 1
599:         LSFFL(II) = LSFFL(N)
600:         IZFFL(II) = IZFFL(N)
601:       end if
602:     360 continue
603:     NFFL(NZONE+1) = NFFL(NZONE+1) - NFFL(MZONE)
604:     NFFL(MZONE) = 0
605: C
606:   return
607: end

```

src/cgview/flyend.f

```
1:      subroutine FLYEND( N11, N2, XPLT, YPLT, AAAG, BBBG, CCCG, GDIST,
2:      &                    LSFFL, XMAX, YMAX,
3:      &                    ILOOP )
4: C=====
5: C  give maximum flight distance for particles whose pathes are
6: C  terminated after drawing.
7: C=====
8:      real*8 XPLT(*), YPLT(*), XMAX, YMAX
9:      real*8 AAAG(*),BBBG(*),CCCG(*),GDIST(*)
10:     integer LSFFL(*)
11: C
12:     do 100 I = N11, N2
13:       J      = LSFFL(I)
14:       XPLT(J) = XPLT(J) + GDIST(J)*AAAG(J)
15:       YPLT(J) = YPLT(J) + GDIST(J)*BBBG(J)
16:       CCCG(J) = ZPLT(J) + GDIST(J)*CCCG(J)
17:       GDIST(J) = 0.0d0
18:
19:       CS      if ( ILOOP.eq.2 ) then
20:       CS        XPLT(J) = XMAX
21:       CS      else
22:       CS        YPLT(J) = YMAX
23:       CS      end if
24:     100 continue
25:     return
26:     end
```


src/cgview/fxcheck.f

```

1:      subroutine FXCHECK( IPR,   IBBBB, X,      Y,      Z,      IBSDA,
2:      &                   NBODY, IPSDA, NSURF, SDA,    NSDA )
3: C=====
4: C  <NVP/GMVP>
5: C PURPOSE: show surface F(X,Y,Z) value for a body.
6: C CALLED IN: anywhere
7: C CALLS:    NONE
8: C UPDATE :
9: C -----
10: C arguments  ( i=input o=output w= work )
11: C
12: C i  IPR      : message output I/O unit
13: C i  IBBBB    : target body ID
14: C i  X        : x-coordinate
15: C i  Y        : y-coordinate
16: C i  Z        : z-coordinate
17: C i  IBSDA(3,NBODY) : body information
18: C i  IPSDA(3,NSURF+1) : body information
19: C i  SDA(NSDA) : surface data
20: C -----
21:      implicit real*8(D)
22: C
23:      real*8 X, Y, Z
24: C
25:      integer IBSDA(3,NBODY)
26:      integer IPSDA(3,NSURF+1)
27:      real*8 SDA(NSDA)
28: C
29: C ..... WORKING AREA .....
30: C
31:      real*8 FXYZ
32:      character*3 TYPE
33:      character*80 LIN
34: C
35: C -----
36: C
37:      KB      = 0
38:      do KB = 1, NBODY
39:      if ( IBSDA(2,KB).eq.IBBBB ) then
40:      go to 100
41:      end if
42:      end do
43: C
44:      if ( IBBBB.ne.0 ) then
45:      write(IPR,*) 'xxx body ', IBBBB, ' does not exist.'
46:      end if
47: C
48:      write(IPR,7000)
49: 7000 format(1X,'>>> Possible body IDs')
50:      NB      = 1
51:      LIN      = ' '
52:      do I = 1, NBODY
53:      call TYPBOD( TYPE, '<', IBSDA(3,I) )
54:      write(LIN(NB+11),'(I6,'('',A,'') ''') IBSDA(2,I),
55:      &      TYPE(1:3)
56:      NB      = NB + 12
57:      if ( NB+11.gt.LEN(LIN) ) then
58:      write(IPR,7020) LIN(:ICLEN2(LIN))
59: 7020 format(1X,'>> ',A)
60:      NB      = 1
61:      LIN      = ' '
62:      end if
63:      end do
64:      if ( NB.gt.1 ) write(IPR,7020) LIN(:ICLEN2(LIN))
65:      return

```

```

66: C
67: 100 continue
68: C
69: C
70: 7040 format(1X,'>> Surface #',I4,' Type <',A,'>')
71: 7060 format(1X,'>> Distance = ',E12.5,' FX = ',E12.5)
72: 7080 format(1X,'>> Distance = ',E12.5,' FX1,FX2 = ',2E12.5)
73: C
74:      do 110 ISP = IBSDA(1,KB), IBSDA(1,KB+1) - 1
75: C
76:      LS      = IPSDA(1,ISP)
77:      KSF      = NINT(SDA(LS))
78: C
79:      LS      = LS + 1
80: C
81: C << Surface type 1 >>
82: C ..... SLAB .....
83: C
84: C -----
85: C      Space between two parallel planes;
86: C
87: C      A*X + B*X + C*Y - P1 = 0
88: C      A*X + B*X + C*Y - P2 = 0
89: C
90: C      A**2 + B**2 + C**2 = 1
91: C
92: C      SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
93: C      SDA(LS+4) = P1, SDA(LS+5) = P2
94: C -----
95: C
96:      if ( KSF.eq.1 ) then
97:      *VOCL LOOP,NOVREC
98:      D1      = SDA(LS)*X + SDA(LS+1)*Y + SDA(LS+2)*Z
99:      D3      = (D1-SDA(LS+3))
100:      D4      = (D1-SDA(LS+4))
101: C
102: C      ... D1 is distance to the nearest point on the surface.
103: C
104:      DFX1    = D3
105:      DFX2    = D4
106:      DIST    = MIN(ABS(D3),ABS(D4))
107:      write(IPR,7040) ISP, 'Slab'
108:      write(IPR,7080) DIST, DFX1, DFX2
109: C
110: C << Surface type 2 >>
111: C ..... SPHERE .....
112: C -----
113: C      (X-A)**2 + (Y-B)**2 + (Z-C)**2 - R**2 < 0
114: C
115: C      SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
116: C      SDA(LS+3) = R**2
117: C -----
118:      else if ( KSF.eq.2 ) then
119:      D3      = 0.5D0/(SQRT(SDA(LS+3))+1.0D-30)
120:      D1      = (SDA(LS)-X)**2 + (SDA(LS+1)-Y)**2 + (SDA(LS+2)
121:      &      -Z)**2 - SDA(LS+3)
122: C
123: C      ... D1 is approximately distance to the surface when the point is
124: C      near the surface.
125: C      D1 = |P**2-R**2|/(2R) = |P+R| |P-R|/(2R) => |P-R|
126: C
127:      DFX    = D1
128:      DIST    = ABS(D1)*D3
129:      write(IPR,7040) ISP, 'Sphere'
130:      write(IPR,7060) DIST, DFX

```

src/cgview/fxcheck.f

```

131: C
132: C << Surface type 3 >>
133: C ..... CYLINDER .....
134: C -----
135: C (X-A)**2 + (Y-B)**2 + (Z-C)**2
136: C -( (X-A)*K + (Y-B)*L + (Z-C)*M )**2 - R**2 < 0
137: C
138: C Where K**2 + L**2 + M**2 = 1
139: C
140: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
141: C SDA(LS+3) = K, SDA(LS+4) = L, SDA(LS+5) = M
142: C SDA(LS+6) = R**2
143: C -----
144: C else if ( KSF.eq.3 ) then
145: C   D3 = 0.5D0/(SQRT(SDA(LS+6))+1.0D-30)
146: C   DX = X - SDA(LS)
147: C   DY = Y - SDA(LS+1)
148: C   DZ = Z - SDA(LS+2)
149: C
150: C   DFX = DX**2 + DY**2 + DZ**2 - SDA(LS+6)
151: C   & - ( SDA(LS+3)*DX+SDA(LS+4)*DY+SDA(LS+5)*DZ )**2
152: C
153: C ... D1 is approximately distance to the surface when the point is
154: C near the surface.
155: C   D1 = |P**2-R**2|/(2R) = |P+R| |P-R|/(2R) = |P-R|
156: C
157: C   DIST = ABS(DFX)*D3
158: C   write(IPR,7040) ISP, 'Cylinder'
159: C   write(IPR,7060) DIST, DFX
160: C
161: C << Surface type 4 >>
162: C ..... GENERAL QUADRATIC SURFACE
163: C -----
164: C A*X**2 + B*Y**2 + C*Z**2 +
165: C D*X*Y + E*Y*Z + F*Z*X +
166: C G*X + H*Y + I*Z + J < 0
167: C
168: C or
169: C
170: C   +-      +-      +-
171: C   ( X Y Z 1 ) | A  D  F  G | | X |
172: C               | 0  B  E  H | | Y | < 0
173: C               | 0  C  I  J | | Z |
174: C               +-      +-      +-
175: C
176: C   SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
177: C   SDA(LS+3) = D, SDA(LS+4) = E, SDA(LS+5) = F
178: C   SDA(LS+6) = G, SDA(LS+7) = H, SDA(LS+8) = I
179: C   SDA(LS+9) = J
180: C
181: C << Distance to the surface >>
182: C
183: C When current point {p} is near the surface F{x} = 0, the surface is
184: C approximately a plane like;
185: C
186: C ([grad F{x}]{x=p},{X}) + F{p} = 0
187: C
188: C where {X} is displacement vector from {p}.
189: C
190: C So the distance is approximately;
191: C
192: C   D = |F{p} / grad F{p}|
193: C
194: C -----
195: C else if ( KSF.eq.4 ) then

```

```

196:
197: C   D1 = X*(X*SDA(LS)+Y*SDA(LS+3)+Z*SDA(LS+5)+SDA(LS+6))
198: C   & + Y*(Y*SDA(LS+1)+Z*SDA(LS+4)+SDA(LS+7))
199: C   & + Z*(Z*SDA(LS+2)+SDA(LS+8)) + SDA(LS+9)
200: C   DGFY = 2*X*SDA(LS) + Y*SDA(LS+3) + Z*SDA(LS+5) +
201: C   & SDA(LS+6)
202: C   DGFY = 2*Y*SDA(LS+1) + Z*SDA(LS+4) + X*SDA(LS+3)
203: C   & + SDA(LS+7)
204: C   DGFZ = 2*Z*SDA(LS+2) + X*SDA(LS+5) + Y*SDA(LS+4)
205: C   & + SDA(LS+8)
206: C
207: C   ... |grad F{p}|
208: C   DGF = SQRT(DGFY*DGFY+DGFZ*DGFZ) + 1.0D-30
209: C   DFX = D1
210: C   DIST = ABS(D1) / DGF
211: C   write(IPR,7040) ISP, 'Quadratic'
212: C   write(IPR,7060) DIST, DFX
213: C
214: C << Surface type 5 >>
215: C ..... PLANE (HALF SPACE)
216: C -----
217: C A*X + B*Y + C*Z - D < 0
218: C
219: C A**2 + B**2 + C**2 = 1
220: C
221: C SDA(LS) = A, SDA(LS+1) = B, SDA(LS+2) = C
222: C SDA(LS+3) = D
223: C -----
224: C else if ( KSF.eq.5 ) then
225: C   D1 = (X*SDA(LS)+Y*SDA(LS+1)+Z*SDA(LS+2)-SDA(LS+3))
226: C   DFX = D1
227: C   DIST = ABS(D1)
228: C   write(IPR,7040) ISP, 'Half space'
229: C   write(IPR,7060) DIST, DFX
230: C
231: C << Surface type 6 >>
232: C ..... TORUS .....
233: C -----
234: C
235: C   2      2
236: C   B T + (R-A) + C T(R-A) - D = 0
237: C
238: C Where
239: C
240: C T = NX*(X-VX) + NY*(Y-VY)+NZ*(Z-VZ)
241: C (VX,VY,VZ) : Center of the torus
242: C (NX,NY,NZ) : Rotation axis vector of the torus (unit vector)
243: C
244: C R : Distance from rotation axis
245: C   2      2      2      2      2
246: C   R = (X-VX) + (Y-VY) + (Z-VZ) - T
247: C
248: C A : Rotation radius
249: C
250: C SDA(LS) = VX, SDA(LS+1) = VY, SDA(LS+2) = VZ
251: C SDA(LS+3) = NX, SDA(LS+4) = NY, SDA(LS+5) = NZ
252: C SDA(LS+6) = A
253: C SDA(LS+7) = B
254: C SDA(LS+8) = C
255: C SDA(LS+9) = D
256: C
257: C << Distance to the surface >>
258: C
259: C When current point {p} is near the surface of the torus;
260: C gradient of F{x} in (T,R,theta) coordinates is approximated by

```

src/cgview/fxcheck.f

```
261: C
262: C   grad F{T,R,theta} = ( dF/dT, dF/dR, 0 )
263: C   So the norm of grad F is approximately;
264: C
265: C   |grad F| => sqrt[(2B T+C*(R-A))**2 + (2*(R-A)+C*T)**2]
266: C
267: C-----
268: C
269: C       else if ( KSF.eq.6 ) then
270: C           DX      = X - SDA(LS)
271: C           DY      = Y - SDA(LS+1)
272: C           DZ      = Z - SDA(LS+2)
273: C           DXX     = SDA(LS+3)*DX + SDA(LS+4)*DY + SDA(LS+5)*DZ
274: C           DX2     = DXX*DXX
275: C           DR      = SQRT(DX**2+DY**2+DZ**2-DX2) - SDA(LS+6)
276: C           D1      = SDA(LS+7)*DX2+DR*(DR+DXX*SDA(LS+8))-SDA(LS+9)
277: C           ... |grad F{p}|
278: C           DGF     = SQRT((2.0D0*SDA(LS+7)*DXX+SDA(LS+8)*DR)**2
279: C   &              +(2.0D0*DR+SDA(LS+8)*DXX)**2) + 1.0D-30
280: C
281: C           DFX     = D1
282: C           DIST    = ABS(D1) /DGF
283: C           write(IPR,7040) ISP, 'Torus'
284: C           write(IPR,7060) DIST, DFX
285: C
286: C       else
287: C           write(IPR,*) '!!(FXCHECK) INVALID SURFACE TYPE ', KSF
288: C           write(IPR,*) ' (called from ', SUBNM, ' )'
289: C       end if
290: C
291: C   110 continue
292: C       return
293: C   end
```

src/cgview/gpbody.f

```
1:      subroutine GPBODY( IPR,   IBBBB, MXYZ,
2:      &                   PAPER, XMAX, YMAX, GRANGE, DINF,
3:      &                   IBSDA, GRBOD, NBODY )
4: C=====
5: C Purpose: automatic setting of geometry display parameter PAPER for
6: C           a body.
7: C Called in: PABLO1
8: C -----
9: C i IBBBB : body ID ( > input-zone , < zone )
10: C i MXYZ : slice plane ( 1/2/3/-1/-2/-3 = X-Y/X-Z/Y-Z/Y-X/Z-X/Z-Y )
11: C o PAPER : CGVIEW drawing area specification
12: C o XMAX, YMAX: X-Y size of drawing area
13: C o GRANGE(6) : X,Y,Z coordinate range
14: C -----
15: C
16:      include 'INC/_FLAGS'
17: C
18:      real*8 PAPER(10)
19:      real*8 XMAX, YMAX
20:      real*8 GRANGE(6)
21: C
22:      real*8 DINF
23:      real*8 GRBOD(6,NBODY)
24:      integer IBSDA(3,NBODY)
25: C
26: C -----
27: C
28:      KB      = 0
29:      if ( IBBBB.gt.0 ) then
30:          do KB=1,NBODY
31:              if ( IBSDA(2,KB).eq.IBBBB ) then
32:                  goto 100
33:              end if
34:          end do
35:          write(IPR,*) 'xxx Body ', KB, ' does not exist'
36:          return
37:      else
38:          return
39:      end if
40: 100 continue
41: C
42:      GRANGE(1)  = GRBOD(1,KB)
43:      GRANGE(2)  = GRBOD(2,KB)
44:      GRANGE(3)  = GRBOD(3,KB)
45:      GRANGE(4)  = GRBOD(4,KB)
46:      GRANGE(5)  = GRBOD(5,KB)
47:      GRANGE(6)  = GRBOD(6,KB)
48: C
49:      write(IPR,*) ' === Geometry range of specified body < ',
50:      &            ' IZZZZ, '>'
51:      write(IPR,7000) GRANGE
52: 7000 format(2X,E15.7,' < X < ',E15.7/2X,E15.7,' < Y < ',E15.7/2X,E15.7,
53:      &            ' < Z < ',E15.7)
54: C
55: C ... make PAPER(1:9) and XMAX, YMAX
56: C
57:      call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
58: C
59:      return
60:      end
```

src/cgview/gpcell.f

```

1:      subroutine GPCELL( IPR,MCELL,IMCELL,MXYZ,
2:      &                  PAPER, XMAX,  YMAX, GRANGE,  DINF,
3:      &                  IBSDA, GRBOD, NBODY, IDCEL, IPCEL, ICTYP,
4:      &                  NCELL, KZAA,  NZONE, KZDA,  KSFBD, NZDA )
5: C=====
6: C Purpose: automatic setting of geometry display parameter PAPER for
7: C         cell MCELL
8: C         (or global geometry when MCELL = 0,
9: C         or cell ID inquiry when MCELL < 0 )
10: C Called in: PABLO1
11: C-----
12: C io MCELL : cell ID
13: C o IMCELL : first zone # of cell MCELL (material must be -999)
14: C i MXYZ : slice plane ( 1/2/3/-1/-2/-3 = X-Y/X-Z/Y-Z/Y-X/Z-X/Z-Y )
15: C o PAPER : CGVIEW drawing area specification
16: C o XMAX, YMAX: X-Y size of drawing area
17: C o GRANGE(6) : X,Y,Z coordinate range
18: C-----
19: C
20:      include 'INC/_FLAGS'
21: C
22:      real*8 PAPER(10)
23:      real*8 XMAX, YMAX
24:      real*8 GRANGE(6)
25: C
26:      real*8 DINF
27:      real*8 GRBOD(6,NBODY)
28:      integer IBSDA(3,NBODY)
29:      integer IDCEL(NCELL), IPCEL(NCELL+1), ICTYP(NCELL)
30:      integer KZAA(NZONE+1)
31:      integer KZDA(2,NZDA), KSFBD(NZDA)
32: C
33: C ... local variable
34: C
35:      character*8 TCELL
36:      integer IBM(6)
37: C-----
38: C-----
39: C
40:      if ( MCELL.lt.0 ) then
41:          call DPYCEL( IPR,  NCELL, IDCEL, ICTYP )
42:          return
43:      end if
44: C
45: C
46:      if ( MCELL.gt.0 ) then
47:          IMCELL = 0
48:          do 100 I = 1, NCELL
49:              if ( IDCEL(I).eq.MCELL ) then
50:                  IM = I
51:                  go to 110
52:              end if
53:          100 continue
54: C
55:          write(IPR,*) 'XXX Cell with ID <', MCELL, '> is not defined.'
56:          call DPYCEL( IPR,  NCELL, IDCEL, ICTYP )
57:          return
58: C
59:          110 continue
60:          IMCELL = IPCEL(IM)
61:          if ( IMCELL.gt.0 ) then
62:              write(IPR,*) ' == Display cell <', MCELL, '> (zone # ',
63:      &                  IMCELL, ' )'
64:          end if
65: C
66:      else
67:          IMCELL = 0
68:      end if
69: C
70: C .... range of zone # .....
71: C
72:      120 continue
73: C
74:      if ( IMCELL.le.0 ) then
75:          MZ1 = 1
76:          MZ2 = NZONE
77:          if ( JLATT.ne.0 ) MZ2 = IPCEL(1) - 1
78:      else
79:          MZ1 = IPCEL(IM)
80:          MZ2 = IPCEL(IM+1) - 1
81:      end if
82: C
83:      GRANGE(1) = DINF
84:      GRANGE(2) = -DINF
85:      GRANGE(3) = DINF
86:      GRANGE(4) = -DINF
87:      GRANGE(5) = DINF
88:      GRANGE(6) = -DINF
89: C
90:      IIS0 = 0
91:      do 160 IZ = MZ1, MZ2
92:          CCCCC write(ipr,*) " IZ= ",IZ
93:          do 150 IS = KZAA(IZ), KZAA(IZ+1) - 1
94:              C
95:              ... body ID
96:              C
97:              IIS = ABS(KSFBD(IS))
98:              if ( IIS.eq.IIS0 ) go to 150
99:              IIS0 = IIS
100:          C
101:          ... body #
102:          C
103:          do 130 IB = 1, NBODY
104:              if ( IIS.eq.IBSDA(2,IB) ) go to 140
105:          130 continue
106:          140 continue
107:          CCCCC write(ipr,*) " IB= ",IB," GRBOD ",(GRBOD(i,ib),i=1,6)
108:          C
109:          GRANGE(1) = MIN(GRANGE(1),GRBOD(1,IB))
110:          GRANGE(2) = MAX(GRANGE(2),GRBOD(2,IB))
111:          GRANGE(3) = MIN(GRANGE(3),GRBOD(3,IB))
112:          GRANGE(4) = MAX(GRANGE(4),GRBOD(4,IB))
113:          GRANGE(5) = MIN(GRANGE(5),GRBOD(5,IB))
114:          GRANGE(6) = MAX(GRANGE(6),GRBOD(6,IB))
115:          C
116:          if( GRBOD(1,IB) .lt. GRANGE(1) ) then
117:              GRANGE(1) = GRBOD(1,IB)
118:          CCCC ibm(1) = ib
119:          end if
120:          if( GRBOD(2,IB) .gt. GRANGE(2) ) then
121:              GRANGE(2) = GRBOD(2,IB)
122:          CCCC ibm(2) = ib
123:          end if
124:          if( GRBOD(3,IB) .lt. GRANGE(3) ) then
125:              GRANGE(3) = GRBOD(3,IB)
126:          CCCC ibm(3) = ib
127:          end if
128:          if( GRBOD(4,IB) .gt. GRANGE(4) ) then
129:              GRANGE(4) = GRBOD(4,IB)
130:          CCCC ibm(4) = ib

```

src/cgview/gpcell.f

```
131:         end if
132:         if( GRBOD(5,IB) .lt. GRANGE(5) ) then
133:             GRANGE(5) = GRBOD(5,IB)
134: CCCC         ibm(5) = ib
135:         end if
136:         if( GRBOD(6,IB) .gt. GRANGE(6) ) then
137:             GRANGE(6) = GRBOD(6,IB)
138: CCCC         ibm(6) = ib
139:         end if
140: 150         continue
141: 160         continue
142: C
143:     if ( MCELL.gt.0 ) then
144:         write(IPR,*) ' === Geometry range of specified cell <', MCELL,
145:         & ' >'
146:     else
147:         write(IPR,*) ' === Geometry range for global geometry!'
148:     end if
149:     write(IPR,7000) GRANGE
150: 7000 format(2X,E15.7,' < X < ',E15.7/2X,E15.7,' < Y < ',E15.7/2X,E15.7,
151: & ' < Z < ',E15.7)
152: check %%%%%%%%%
153: C     write(ipr,*) '%%% ibm->ID : ',(IRSDA(2,IBM(I)),I=1,6)
154: C %%%%%%%%%
155: C
156:     call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
157: C
158:     return
159: end
160: C
161: C=====
162: C
163:     subroutine DPYCEL( IPR, NCELL, IDCEL, ICTYP )
164: C
165: C     ... local routine to show cell ID and type
166: C
167:     integer IDCEL(NCELL)
168:     integer ICTYP(NCELL)
169: C
170: C     ... local data
171: C
172:     parameter( NLL = 5 )
173:     character*5 TYPE(NLL)
174:     integer IC(NLL)
175: C
176: C-----
177: C
178:     write(IPR,*) ' == Defined cells(type) are ...'
179:     do N = 1, NCELL, NLL
180:         N2 = MIN(NCELL,N+NLL-1)
181:         do J = N, N2
182:             K = J - N + 1
183:             IC(K) = IDCEL(J)
184:             call CELTYP( TYPE(K), '<', ICTYP(J) )
185:         end do
186:         write(IPR,7000) (IC(K),TYPE(K),K=1,N2-N+1)
187: 7000 format(3X,5(I6,'('',A,')':))
188:     end do
189:     return
190: end
```

src/cgview/gpzone.f

```

1:      subroutine GPZONE( IPR,  IZZZZ, MXYZ, MCELL, IMCELL,
2:      &                  PAPER, XMAX, YMAX, GRANGE, DINF,
3:      &                  IBSDA, GRBOD, NBODY, NINPZ,
4:      &                  KINPZ,KZAA, NZONE, KZDA, KSFB, NZDA,
5:      &                  KCELL, IPCEL, IDCEL, NCELL )
6: C=====
7: C Purpose: automatic setting of geometry display parameter PAPER for
8: C         input-zone/zone.
9: C Called in: PABLO1
10: C-----
11: C i IZZZZ : zone ID ( > input-zone, < zone )
12: C i MXYZ : slice plane ( 1/2/3/-1/-2/-3 = X-Y/X-Z/Y-Z/Y-X/Z-X/Z-Y )
13: C o PAPER : CGVIEW drawing area specification
14: C o XMAX, YMAX: X-Y size of drawing area
15: C o GRANGE(6) : X,Y,Z coordinate range
16: C-----
17: C
18:      include 'INC/_FLAGS'
19: C
20:      real*8 PAPER(10)
21:      real*8 XMAX, YMAX
22:      real*8 GRANGE(6)
23: C
24:      real*8 DINF
25:      real*8 GRBOD(6,NBODY)
26:      integer IBSDA(3,NBODY)
27:      integer KINPZ(NZONE)
28:      integer KZAA(NZONE+1)
29:      integer KZDA(2,NZDA), KSFB(NZDA)
30: C
31:      integer KCELL(NZONE), IPCEL(NCELL+1), IDCEL(NCELL)
32: C
33:      character*8 TCELL
34: C
35: C-----
36: C
37:      KZ      = IZZZZ
38:      if ( IZZZZ.gt.0 ) then
39:        if ( IZZZZ.gt.NINPZ ) then
40:          write(IPR,*) 'xxx input zone ', IZZZZ,
41:          &          ' does not exist (ninpz=', NINPZ, ')'
42:          return
43:        end if
44:      else if ( IZZZZ.lt.0 ) then
45:        KZ      = -IZZZZ
46:        if ( KZ.gt.NZONE ) then
47:          write(IPR,*) 'xxx zone ', KZ, ' does not exist (nzone=
48:          &          NZONE, ')'
49:          return
50:        end if
51:      else
52:        return
53:      end if
54: C
55: C .... range of zone # .....
56: C
57:      if ( IZZZZ.lt.0 ) then
58:        MZ1      = -IZZZZ
59:        MZ2      = MZ1
60:      else
61:        MZ1      = 0
62:        do IZ=1,NZONE
63:          if ( KINPZ(IZ).eq.IZZZZ ) then
64:            if ( MZ1.eq.0 ) MZ1 = IZ
65:            MZ2      = IZ

```

```

66:          end if
67:        end do
68:      end if
69: C
70: C .... cell .....
71: C
72:      if ( JLATT.ne.0 ) then
73:        MCELL = KCELL(MZ1)
74:        if ( MCELL.gt.0 ) then
75:          IMCELL = IPCEL(MCELL)
76:          MCELL = IDCEL(MCELL)
77:        else
78:          IMCELL = 0
79:        end if
80:        write(IPR,*) ' === Zone is in cell <', MCELL, '>'
81:      end if
82: C
83:      GRANGE(1) = DINF
84:      GRANGE(2) = -DINF
85:      GRANGE(3) = DINF
86:      GRANGE(4) = -DINF
87:      GRANGE(5) = DINF
88:      GRANGE(6) = -DINF
89: C
90:      IIS0      = 0
91:      do 160 IZ = MZ1, MZ2
92:        CCCCC write(ipr,*) " IZ= ",IZ
93:        do 150 IS = KZAA(IZ), KZAA(IZ+1) - 1
94:          C
95:          ... body ID
96:          C
97:          IIS      = ABS(KSFB(IS))
98:          if ( IIS.eq.IIS0 ) go to 150
99:          IIS0      = IIS
100: C
101: C ... body #
102: C
103:        do 130 IB = 1, NBODY
104:          if ( IIS.eq.IBSDA(2,IB) ) go to 140
105:        130 continue
106:        140 continue
107:        CCCCC write(ipr,*) " IB= ",IB," GRBOD ",(GRBOD(i,ib),i=1,6)
108: C
109:        GRANGE(1) = MIN(GRANGE(1),GRBOD(1,IB))
110:        GRANGE(2) = MAX(GRANGE(2),GRBOD(2,IB))
111:        GRANGE(3) = MIN(GRANGE(3),GRBOD(3,IB))
112:        GRANGE(4) = MAX(GRANGE(4),GRBOD(4,IB))
113:        GRANGE(5) = MIN(GRANGE(5),GRBOD(5,IB))
114:        GRANGE(6) = MAX(GRANGE(6),GRBOD(6,IB))
115:      150 continue
116:    160 continue
117: C
118:      if ( IZZZZ.gt.0 ) then
119:        write(IPR,*) ' === Geometry range of specified input-zone <',
120:        &          IZZZZ,>'
121:      else
122:        write(IPR,*) ' === Geometry range of specified zone <',
123:        &          IZZZZ,>'
124:      end if
125:      write(IPR,7000) GRANGE
126:      7000 format(2X,E15.7,' < X < ',E15.7/2X,E15.7,' < Y < ',E15.7/2X,E15.7,
127:      &          ' < Z < ',E15.7)
128: C
129: C ... make PAPER(1:9) and XMAX, YMAX
130: C

```

src/cgview/gpzone.f

```
131:      call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
132: C
133:      return
134:      end
```

SAE

src/cgview/gr2paper.f

```

1:      subroutine GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
2: C=====
3: C purpose: create viewplane parameters (PAPER, XMAX/YMAX) from
4: C      geometry range data (GRANGE) and plane orientation (MXYZ)
5: C=====
6:      real*8 GRANGE(6), DINF
7:      real*8 PAPER(9), XMAX, YMAX
8: C
9: C-----
10: C
11: C      ... X-Y plane
12: C      if ( MXYZ.eq.1 ) then
13: C          IX = 1
14: C          IY = 2
15: C          IZ = 3
16: C          KX = 0
17: C          KY = 2
18: C          KZ = 4
19: C          PAPER(4) = 1.0D0
20: C          PAPER(5) = 0.0D0
21: C          PAPER(6) = 0.0D0
22: C          PAPER(7) = 0.0D0
23: C          PAPER(8) = 1.0D0
24: C          PAPER(9) = 0.0D0
25: C      ... Y-X plane
26: C      else if ( MXYZ.eq.-1 ) then
27: C          IX = 2
28: C          IY = 1
29: C          IZ = 3
30: C          KX = 2
31: C          KY = 0
32: C          KZ = 4
33: C          PAPER(4) = 0.0D0
34: C          PAPER(5) = 1.0D0
35: C          PAPER(6) = 0.0D0
36: C          PAPER(7) = 1.0D0
37: C          PAPER(8) = 0.0D0
38: C          PAPER(9) = 0.0D0
39: C      ... X-Z plane
40: C      else if ( MXYZ.eq.2 ) then
41: C          IX = 1
42: C          IY = 3
43: C          IZ = 2
44: C          KX = 0
45: C          KY = 4
46: C          KZ = 2
47: C          PAPER(4) = 1.0D0
48: C          PAPER(5) = 0.0D0
49: C          PAPER(6) = 0.0D0
50: C          PAPER(7) = 0.0D0
51: C          PAPER(8) = 0.0D0
52: C          PAPER(9) = 1.0D0
53: C      ... Z-X plane
54: C      else if ( MXYZ.eq.-2 ) then
55: C          IX = 3
56: C          IY = 1
57: C          IZ = 2
58: C          KX = 4
59: C          KY = 0
60: C          KZ = 2
61: C          PAPER(4) = 0.0D0
62: C          PAPER(5) = 0.0D0
63: C          PAPER(6) = 1.0D0
64: C          PAPER(7) = 1.0D0
65: C          PAPER(8) = 0.0D0

```

```

66:      PAPER(9) = 0.0D0
67: C      ... Y-Z plane
68: C      else if ( MXYZ.eq.3 ) then
69: C          IX = 2
70: C          IY = 3
71: C          IZ = 1
72: C          KX = 2
73: C          KY = 4
74: C          KZ = 0
75: C          PAPER(4) = 0.0D0
76: C          PAPER(5) = 1.0D0
77: C          PAPER(6) = 0.0D0
78: C          PAPER(7) = 0.0D0
79: C          PAPER(8) = 0.0D0
80: C          PAPER(9) = 1.0D0
81: C      ... Z-Y plane
82: C      else if ( MXYZ.eq.-3 ) then
83: C          IX = 3
84: C          IY = 2
85: C          IZ = 1
86: C          KX = 4
87: C          KY = 2
88: C          KZ = 0
89: C          PAPER(4) = 0.0D0
90: C          PAPER(5) = 0.0D0
91: C          PAPER(6) = 1.0D0
92: C          PAPER(7) = 0.0D0
93: C          PAPER(8) = 1.0D0
94: C          PAPER(9) = 0.0D0
95: C      end if
96: CCCCC PAPER(10) = -1.0D0
97: C
98: C      XMARGIN = 0.05
99: C
100: C      if( abs( GRANGE(KX+1) ).ne.DINF.and.abs( GRANGE(KX+2) ).ne.DINF ) then
101: C          XMAX = ABS( GRANGE(KX+2)-GRANGE(KX+1) )*(1.0D0+2*XMARGIN)
102: C      else
103: C          XMAX = 0.0
104: C      end if
105: C
106: C      if( abs( GRANGE(KY+1) ).ne.DINF.and.abs( GRANGE(KY+2) ).ne.DINF ) then
107: C          YMAX = ABS( GRANGE(KY+2)-GRANGE(KY+1) )*(1.0D0+2*XMARGIN)
108: C      else
109: C          YMAX = 0.0
110: C      end if
111: C
112: C      if ( XMAX.eq.0.0.and.YMAX.eq.0.0 ) then
113: C          XMAX = 10.0
114: C          YMAX = 10.0
115: C      else
116: C          XMAX = MAX(XMAX, YMAX)
117: C          YMAX = YMAX
118: C      end if
119: C
120: C      if( abs( GRANGE(KX+1) ).ne.DINF.and.abs( GRANGE(KX+2) ).ne.DINF ) then
121: C          PAPER(IX) = ( GRANGE(KX+1)+GRANGE(KX+2) )*(0.5 - 0.55*XMAX/
122: C      &      (1.0D0+2*XMARGIN)
123: C      else
124: C          if ( GRANGE(KX+1).ne.DINF ) then
125: C              PAPER(IX) = GRANGE(KX+1) - 0.05*XMAX
126: C          else if ( GRANGE(KX+2).ne.-DINF ) then
127: C              PAPER(IX) = GRANGE(KX+2) - 1.0*XMAX
128: C          else
129: C              PAPER(IX) = -0.5*XMAX
130: C          end if

```

src/cgview/gr2paper.f

```
131:      end if
132:
133:      if( abs(GRANGE(KY+1)).ne.DINF.and.abs(GRANGE(KY+2)).ne.DINF ) then
134:         PAPER(IY) = (GRANGE(KY+1)+GRANGE(KY+2))*0.5 - 0.55*YMAX/
135: &         (1.0D0+2*XMERGIN)
136:      else
137:         if ( GRANGE(KY+1).ne.DINF ) then
138:            PAPER(IY) = GRANGE(KY+1) - 0.05*YMAX
139:         else if ( GRANGE(KY+2).ne.-DINF ) then
140:            PAPER(IY) = GRANGE(KY+2) - 1.0*YMAX
141:         else
142:            PAPER(IY) = -0.5*YMAX
143:         end if
144:      end if
145:
146:      if ( abs(GRANGE(KZ+1)).ne.DINF.and.abs(GRANGE(KZ+2)).ne.DINF ) then
147:         PAPER(IZ) = (GRANGE(KZ+1)+GRANGE(KZ+2))*0.5
148:      else if ( GRANGE(KZ+1).ne.DINF ) then
149:         PAPER(IZ) = GRANGE(KZ+1) + XMAX*0.5
150:      else if ( GRANGE(KZ+2).ne.-DINF ) then
151:         PAPER(IZ) = GRANGE(KZ+2) - XMAX*0.5
152:      else
153:         PAPER(IZ) = 0.0D0
154:      end if
155: C
156:      return
157:      end
```


src/cgview/graint.f

```

131: C-----
132: C
133:       if ( OVERSION.ne.0 ) then
134:         call GETVER( VERSION, RELDAT )
135:         LLV      = ICLEN2(VERSION)
136:         call JSYMBOL( 5.0, 1.0, 2.0, VERSION, 0.0, LLV )
137:       end if
138: C
139: C
140: C-----
141: C      .... drawing size factor and setting of plotting origin
142: C-----
143: C
144:       XFRM      = XMAX
145:       YFRM      = YMAX
146: C
147:       UFRM      = MAX(XFRM,YFRM)
148: C
149: C      ... when "LOST" symbol description is necessary
150: C
151:       if ( OLOSTSYM.ne.0 ) then
152:         CDT      = UFRM/10.0
153:         WFRAME   = UFRM + 4.0*CDT
154:         FACT     = XLENG/WFRAME
155:         X00      = 2.*CDT*FACT
156:         Y00      = X00
157:       else
158:         CDT      = UFRM/10
159: CCCCC  XMGN = 1.5*CDT
160:         XMGN     = 1.2*CDT
161:         WFRAME   = UFRM + 2*XMGN
162:         FACT     = XLENG/WFRAME
163:         X00      = XMGN*FACT
164:         Y00      = X00
165:       end if
166: C      .... origin is moved ....
167: C
168:       call PLOT( X00, Y00, -3 )
169:       call FACTOR( FACT )
170: C
171: C      ... drawing region frame
172: C
173: C      call FRAME( 0.0, 0.0, XFRM, YFRM )
174: C
175: C-----
176: C      ... drawing title
177: C-----
178: C
179:       X(1)      = XMIN
180:       X(2)      = XMAX/2.0
181:       X(3)      = XMAX
182:       Y(1)      = YMIN
183:       Y(2)      = YMAX/2.0
184:       Y(3)      = YMAX
185:       do 100 I = 72, 1, -1
186:         if ( TITLGR(I:1).ne.' ' ) go to 110
187:       100 continue
188:       I         = 1
189:       110 continue
190: C
191:       XLEFT     = (UFRM-I*0.17*CDT) /2.0
192:       if ( OLOSTSYM.ne.0 ) then
193:         SSYM     = 0.17*CDT
194:         YLEFT    = YFRM + 1.8*CDT
195:       else

```

```

196:         SSYM     = 0.18*CDT
197:         YLEFT    = YFRM + 0.8*CDT
198: CCCCC  YLEFT    = YFRM + 1.0*CDT
199:       end if
200: C
201:       call JSYMBOL( XLEFT, YLEFT, SSYM, TITLGR, 0.0, I )
202: C
203:       call SCALE( X, XFRM, 3, 1 )
204:       call SCALE( Y, YFRM, 3, 1 )
205: C
206: C-----
207: C      PLOT AXIS X
208: C-----
209: C
210:       call AXSPRM( XFRM/10.0, 0.14*CDT, 0.13*CDT, 0.2*CDT, 0.28*CDT,
211:         &          0.825*CDT, 0.1*CDT )
212:       JNDIR = 0
213:       call AXISX( JNDIR,0.0,0.0,' ',-1,XFRM,0.0,X(4)/2.0, X(5)/2.0, 1 )
214: C
215: C-----
216: C      PLOT AXIS Y
217: C-----
218: C
219:       call AXSPRM( YFRM/10.0, 0.14*CDT, 0.13*CDT, 0.2*CDT, 0.28*CDT,
220:         &          0.825*CDT, 0.1*CDT )
221:       if ( JHYAXIS .ne.0 ) then
222:         JNDIR = 1
223:         call AXISX(JNDIR, 0.0,0.0,' ',1,YFRM,90.0,Y(4)/2.0,Y(5)/2.0,1)
224:       else
225:         JNDIR = 0
226:         call AXISX(JNDIR, 0.0,0.0,' ',1,YFRM,90.0,Y(4)/2.0,Y(5)/2.0,1)
227:       end if
228:       X(4)    = 0.0
229:       Y(4)    = 0.0
230:       X(5)    = 0.0
231:       Y(5)    = 0.0
232: C
233: C-----
234: C      SET LINE'S MARK SIZE
235: C-----
236: C
237:       call LINMRK( 0.2*CDT*0.75 )
238: C
239: C-----
240: C      Lost point symbol
241: C-----
242: C
243:       if ( OLOSTSYM.ne.0 ) then
244: C
245:         call JSYMBOL( 0.0*CDT, YFRM+1.4*CDT, 0.17*CDT,
246:           &          'LOST FLIGHT X : ', 0.0, 17 )
247:         call JSYMBOL( 17*0.17*CDT, YFRM+1.5*CDT, 0.17*CDT, SY(1), 0.0,
248:           &          SYNO )
249: C *****
250:         call JSYMBOL( 23.0*0.17*CDT, YFRM+1.4*CDT, 0.17*CDT,
251:           &          'LOST FLIGHT Y : ', 0.0, 17 )
252:         call JSYMBOL( 23.0*0.17*CDT+17*0.17*CDT, YFRM+1.5*CDT, 0.17*
253:           &          CDT, SY(2), 0.0, SYNO )
254: C *****
255:         call JSYMBOL( 0.0*CDT, YFRM+1.15*CDT, 0.17*CDT,
256:           &          'LOST SOURCE X : ', 0.0, 17 )
257:         call JSYMBOL( 17*0.17*CDT, YFRM+1.25*CDT, 0.17*CDT, SY(3), 0.0,
258:           &          SYNO )
259: C *****
260:         call JSYMBOL( 23.0*0.17*CDT, YFRM+1.15*CDT, 0.17*CDT,

```

src/cgview/graint.f

```

261:      &      'LOST SOURCE Y : ', 0.0, 17 )
262:      call JSYMBOL( 23.0*0.17*CDT+17*0.17*CDT, YFRM+1.25*CDT, 0.17*
263:      &      CDT, SY(4), 0.0, SYNO )
264: C *****
265:      call JSYMBOL( 0.0*CDT, YFRM+0.9*CDT, 0.17*CDT,
266:      &      'LOST SEARCH X : ', 0.0, 17 )
267:      call JSYMBOL( 17*0.17*CDT, YFRM+1.0*CDT, 0.17*CDT, SY(5), 0.0,
268:      &      SYNO )
269: C *****
270:      call JSYMBOL( 23.0*0.17*CDT, YFRM+0.9*CDT, 0.17*CDT,
271:      &      'LOST SEARCH Y : ', 0.0, 17 )
272:      call JSYMBOL( 23.0*0.17*CDT+17*0.17*CDT, YFRM+1.0*CDT, 0.17*
273:      &      CDT, SY(6), 0.0, SYNO )
274:      end if
275: C
276: C-----
277: C lattice level, cell, and what is drawn
278: C-----
279: C
280:      YNAM = ' '
281:      XNAM = ' '
282:      XNAM(1:7) = 'LEVEL: '
283:      WORK = ' '
284:      write(WORK,'(I2)') LEVEL
285:      call CCOMP(WORK, LEN(WORK),WORK, III)
286:      XNAM(8:) = WORK
287:
288:      WIDE = 1.0
289:
290:      if ( OLOSTSYM.ne.0 ) then
291:          SSYM = 0.17*CDT
292:          call JSYMBOL( 46.0*SSYM, YFRM+1.15*CDT, SSYM, YNAM, 0.0, 11 )
293:          YNAM = ' '
294:      else
295:          YNAM = XNAM(:ICLEN2(XNAM))
296:          CCC SSYM = 0.15*CDT
297:          CCC call JSYMBOL( 26.0*SSYM, YFRM+0.9*SSYM, SSYM, XNAM, 0.0, 11 )
298:      end if
299:
300:      if ( OLOSTSYM.eq.0.and.IMCELL.gt.0 ) then
301:          XNAM(1:6) = 'CELL: '
302:          WORK = ' '
303:          write(WORK,'(I5)') MCELL
304:          call CCOMP(WORK, LEN(WORK),WORK, III)
305:          XNAM(7:) = WORK
306:          ILY = ICLEN2(YNAM)
307:          YNAM(ILY+4:) = XNAM(:ICLEN2(XNAM))
308:      end if
309:
310:      XNAM = ' '
311:      XNAM(1:6) = 'WHAT: '
312:      if ( SPTYP.eq.-1 ) then
313:          XNAM(7:) = 'zone'
314:      else if ( SPTYP.eq.0 ) then
315:          XNAM(7:) = 'input zone'
316:      else if ( SPTYP.eq.1 ) then
317:          XNAM(7:) = 'region'
318:      else if ( SPTYP.eq.2 ) then
319:          XNAM(7:) = 'material'
320:      end if
321:
322:      WIDE = 1.0
323:
324:      ILY = ICLEN2(YNAM)
325:      if ( OLOSTSYM.ne.0 ) then

```

```

326:          call JSYMBOL( 46.0*SSYM, YFRM+0.9*CDT, SSYM, XNAM, 0.0, 13 )
327:      else
328:          ILY = iclen2(ynam)
329:          YNAM(ILY+4:) = xnam(:iclen2(xnam))
330:          ILY1 = iclen2(ynam)
331:          call JSYMBOL( XFRM-ILY1*SSYM, YFRM+0.9*SSYM, SSYM, YNAM, 0.0,
332:          &      ICLEN2(YNAM) )
333:      end if
334: C
335: C-----
336: C ccoordinate of corners
337: C-----
338: C
339:      XNAM = ' '
340:      YNAM = ' '
341: C
342: C ... lower left corner
343: C
344:      XNAM(1:1) = '('
345:      write(WORK,'(F7.2)') PAPER(1)
346:      XNAM(2:8) = WORK(1:7)
347:      XNAM(9:9) = ','
348:      write(WORK,'(F7.2)') PAPER(2)
349:      XNAM(10:16) = WORK(1:7)
350:      XNAM(17:17) = ','
351:      write(WORK,'(F7.2)') PAPER(3)
352:      XNAM(18:24) = WORK(1:7)
353:      XNAM(25:25) = ')'
354:      call JSYMBOL( -1.0*CDT, -0.7*CDT, 0.14*CDT, XNAM, 0.0, 25 )
355: C
356: C ... lower right corner
357: C
358:      XNAM(1:1) = '('
359:      write(WORK,'(F7.2)') PAPER(11)
360:      XNAM(2:8) = WORK(1:7)
361:      XNAM(9:9) = ','
362:      write(WORK,'(F7.2)') PAPER(12)
363:      XNAM(10:16) = WORK(1:7)
364:      XNAM(17:17) = ','
365:      write(WORK,'(F7.2)') PAPER(13)
366:      XNAM(18:24) = WORK(1:7)
367:      XNAM(25:25) = ')'
368:      call JSYMBOL( XFRM+(1.2-0.14*25.0)*CDT, -0.7*CDT, 0.14*CDT, XNAM,
369:      &      0.0, 25 )
370: C
371: C ... upper left corner
372: C
373:      YNAM(1:1) = '('
374:      write(WORK,'(F7.2)') PAPER(14)
375:      XNAM(2:8) = WORK(1:7)
376:      XNAM(9:9) = ','
377:      write(WORK,'(F7.2)') PAPER(15)
378:      XNAM(10:16) = WORK(1:7)
379:      XNAM(17:17) = ','
380:      write(WORK,'(F7.2)') PAPER(16)
381:      XNAM(18:24) = WORK(1:7)
382:      XNAM(25:25) = ')'
383:      if ( JHYAXIS.eq.0 ) then
384:          call JSYMBOL( -1.0*CDT, YFRM+0.55*CDT, 0.14*CDT,XNAM,0.0,25 )
385:      else
386:          call JSYMBOL( -1.0*CDT, YFRM+0.15*CDT, 0.14*CDT,XNAM,0.0,25 )
387:      end if
388: C
389:      if ( JXPS.eq.1 .or. STYLE.ne.0 ) then
390:          call SETLWD( 1 )

```

src/cgview/graint.f

```
391:      else
392:        call SETLWD( 2 )
393:      end if
394: C
395: C      .... refresh screen
396: C
397: C      call PLOT( 0.0, 0.0, 777 )
398: C
399:      return
400:      end
```

SAFE

src/cgview/graphd.f

```
1:      subroutine GRAPHD ( ZONCLR, IZNCLR, REGCLR, MATCLR, IDLAT )
2: C=====
3: C  FINISH PLOT AND TERMINATE DEVICE
4: C=====
5: C/#IF MS_VISUAL
6: C
7: C ... This interface block is for CUTIL & MS-Visual tools. ...
8: C
9: *      interface
10: *      subroutine PLOT( RRR1, RRR2, III1 )
11: *          integer    III1
12: *          real        RRR1, RRR2
13: *CDEC$      ATTRIBUTES C :: PLOT
14: *CDEC$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
15: *      end subroutine
16: *      subroutine CAPLINE( III2 )
17: *          integer    III2
18: *CDEC$      ATTRIBUTES C :: CAPLINE
19: *CDEC$      ATTRIBUTES REFERENCE :: III2
20: *      end subroutine
21: *      subroutine FACTOR( RRR3 )
22: *          real        RRR3
23: *CDEC$      ATTRIBUTES C :: FACTOR
24: *CDEC$      ATTRIBUTES REFERENCE :: RRR3
25: *      end subroutine
26: *      end interface
27: C/#ENDIF
28: C
29:      include '../shared/INC/_SIZES'
30:      include 'INC/_MVIEW'
31:      integer ZONCLR(NZONE), IZNCLR(NINPZ), REGCLR(NREG), MATCLR(0:MXMAT)
32:      integer IDLAT(NLATT)
33: C
34: C-----
35: C
36:      call CAPLINE(1)
37:
38: C      call CANVS2( ZONCLR, IZNCLR, REGCLR, MATCLR, IDLAT, NLATT )
39:      call CANVS2( ZONCLR, IZNCLR, REGCLR, MATCLR, IDLAT )
40:
41: CC/#IF PIFFLIB
42:      write(6,*) '>> One figure has been drawn'
43:      call PLOT( 0.0, 0.0, 777 )
44: C      call plot(0.0,0.0,666)
45:      call FACTOR( 1.0 )
46: CC/#ENDIF
47:      return
48:      end
```

src/cgview/graphe.f

```
1:      subroutine GRAPHE
2: C
3: C/#IF MS_VISUAL
4: C
5: C ... This interface block is for CUTIL & MS-Visual tools. ...
6: C
7: *      interface
8: *          subroutine PLOT( RRR1, RRR2, III1 )
9: *              integer      III1
10: *              real         RRR1, RRR2
11: *CDECS$      ATTRIBUTES C :: PLOT
12: *CDECS$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
13: *          end subroutine
14: *      end interface
15: C/#ENDIF
16: C
17:      call PLOT( 0.0, 0.0, 999 )
18:      return
19: end
```


src/cgview/graphs.f

```
1:      subroutine GRAPHS( IA,      N1,      N2,      N3, WHERE, MZONE,
2:      &      LSFFL, IZFFL, NFFL,
3:      &      XPLT, YPLT,
4:      &      AAAG, BBBG, CCCG, GDIST, KGCNT, LSCAN, IZZ,
5:      &      IBREG, KMAT, KREG, KZMAT, IDLAT, LTYP,
6:      &      KZREG, KINPZ, ZONCLR, IZNCLR, REGCLR, MATCLR,
7:      &      LPOS, LEVL, IBPWK, LPOS0, LEVL0, X01, Y01,
8:      &      IZZX, IZ0, IZ01, ILOOP, COLOR, BCOLOR,
9:      &      BXPLT, BYPLT, CLR, X02, Y02, BCLR, XC01,
10:     &      YC01 )
11: C
12: C=====
13: C Purpose: draw scan pathes.
14: C
15: C Comment for arguments and others:
16: C
17: C (It is not a full description of arguments and writer of this comment
18: C (M.Sasaki) might not be perfectly understanding what an anonymous
19: C programmer of original code was going to do ...)
20: C
21: C *arguments:
22: C
23: C N1, N2: particles pointed by index LSFFL(I), I = N1 to N2, are drawn
24: C whose flight pathes.
25: C
26: C Flight particle stack LSFFL(*), IZFFL(*) and NFFL(1:NZONE+1) is
27: C used to specify particles, but contents of it may not be the
28: C actual "in-flight" particle (for example, particle entered
29: C "reflector" or "outer void" zone.)
30: C
31: C N3 : in "boundary drawing" mode (STYLE=1), LSFFL(I), I =1 to N3
32: C is checked.
33: C
34: C *Bugs (not all of them):
35: C
36: C *Boundary drawing mode by STYLE=1 has not been working correctly
37: C after modification for version3.0 by M.Sasaki, and he has currently
38: C no power or will to correct it (sorry :-)).
39: C He hopes using STYLE=0 and STYLE2=1 be a remedy for this bug.
40: C
41: C=====
42: C/#IF MS_VISUAL
43: C
44: C ... This interface block is for CUTIL & MS-Visual tools. ...
45: C
46: *      interface
47: *      subroutine PLOT( RRR1, RRR2, III1 )
48: *          integer      III1
49: *          real          RRR1, RRR2
50: *CDEC$      ATTRIBUTES C :: PLOT
51: *CDEC$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
52: *      end subroutine
53: *      subroutine CAPLINE( III2 )
54: *          integer      III2
55: *CDEC$      ATTRIBUTES C :: CAPLINE
56: *CDEC$      ATTRIBUTES REFERENCE :: III2
57: *      end subroutine
58: *      subroutine GETCAPLINE( III3 )
59: *          integer      III3
60: *CDEC$      ATTRIBUTES C :: GETCAPLINE
61: *CDEC$      ATTRIBUTES REFERENCE :: III3
62: *      end subroutine
63: *      end interface
64: C/#ENDIF
65: C
```

```
66:     common /DATA/      X,      Y
67:     common /XYP/      ICBK,      BACK
68: C
69:     dimension X(5), Y(5)
70:     integer BACK(200), ICBK
71: C
72:     INCLUDE '../shared/INC/_IUNIT'
73:     INCLUDE '../shared/INC/_SIZES'
74:     INCLUDE 'INC/_FLAGS'
75:     INCLUDE 'INC/_STACK'
76:     INCLUDE 'INC/_XBANK'
77:     INCLUDE '../shared/INC/_PGEOM'
78:     INCLUDE 'INC/_MVIEW'
79: C
80:     integer IA(*)
81:     integer N1, N2, N3
82: C
83: C
84:     character*(*) WHERE
85: C
86:     integer KMAT(NINPZ), KREG(NINPZ), KZMAT(NZONE,2), KZREG(NZONE),
87:     &      KINPZ(NZONE), IDLAT(NLATT), LTYP(NLATT)
88: C
89:     integer LSFFL(*)
90:     integer IZFFL(*)
91:     integer NFFL(NZONE+1)
92: C
93: ... particle attributes (bank)
94: C
95:     integer LPOS(NBANK,NEST), LEVL(NBANK), IBREG(NBANK)
96:     integer IZZ(NBANK)
97: C
98: ... zone, region and materla color #'s (CGVIEW specific)
99: C
100:     integer ZONCLR(NZONE), IZNCLR(NINPZ), REGCLR(NREG),
101:     &      MATCLR(0:MXMAT)
102: C
103: ... other particle attributes specific to CGVIEW
104: C
105:     real*8 XPLT(NBANK), YPLT(NBANK), BXPLT(NBANK), BYPLT(NBANK)
106:     real*8 AAAG(NBANK), BBBG(NBANK), CCCG(NBANK), GDIST(NBANK)
107:     integer KGCNT(NBANK)
108:     integer LSCAN(NBANK)
109:     integer COLOR(NBANK), BCOLOR(NBANK), CLR(NBANK), BCLR(NBANK)
110:     integer IBPWK(NBANK)
111:     integer LPOS0(NBANK), LEVL0(NBANK)
112: C
113: ... working variable ...
114: C
115:     integer IZZX(NBANK), IZ0(NBANK), IZ01(NBANK)
116:     real X01(NBANK), Y01(NBANK), X02(NBANK), Y02(NBANK)
117:     real XC01(NBANK), YC01(NBANK)
118: C
119: ... local variable
120: C
121:     integer ICUTI, IIX, IIY
122:     real X001(2), Y001(2)
123:     real*8 DLY
124: C
125:     include '../shared/INC/_PMLATT'
126: C
127:     data ICALL /0/
128: C
129:     include '../shared/INC/_SFLATT'
130: C
```

src/cgview/graphs.f

```

131: C ... statement function to refer CLDAT
132: C
133:       KCLDAT(I) = CLDAT(MIN(I,MXCLDAT))
134:       KMATCLR(I) = MATCLR(MIN(I,MXMAT))
135: C
136: C
137: C -----
138: C
139:       if ( JCGVDB(1).ge.2 ) then
140:         write(*,*) '=== pause GRAPHS <','WHERE','> MZONE ',MZONE,
141:         & ' N1 N2 N3 ',n1,n2,n3,' STYLE ',STYLE, STYLE2
142:         call getcapline( iff )
143:         write(*,*) ' --- capline mode ',iff
144:         read(*,'(i1)') KKKKK
145:       end if
146: C=====
147: C Initial setting for newly generated particles
148: C=====
149: C
150:       if ( where.eq.'SOURCE' ) then
151: C
152:         do 120 I = N1, N2
153: C
154:           if ( IZZ(I).eq.0 ) go to 120
155: C-----
156: C zone is drawn
157: C-----
158:           if ( SPTYP.eq.-1 ) then
159:             IBPWK(I) = IZZ(I)
160:             COLOR(I) = ZONCLR(IZZ(I))
161: C-----
162: C input-zone is drawn
163: C-----
164:           else if ( SPTYP.eq.0 ) then
165:             IBPWK(I) = KINPZ(IZZ(I))
166:             COLOR(I) = IZNCLR(KINPZ(IZZ(I)))
167: C-----
168: C region is drawn
169: C-----
170:           else if ( SPTYP.eq.1 ) then
171:             if ( JTLLT.eq.0 ) then
172:               IBPWK(I) = KZREG(IZZ(I))
173:             else
174:               IBPWK(I) = IBREG(I)
175:             end if
176:
177:             if ( IKIND.eq.2.and.KZMAT(IZZ(I),1).lt.0 ) then
178:               COLOR(I) = 12
179:             else
180:               COLOR(I) = REGCLR(IBPWK(I))
181:             end if
182:
183: C
184: C .... zones not defined as any region ...
185: C
186:           if ( IBPWK(I).eq.0 ) then
187:             if ( KZMAT(IZZ(I),1).le.-999.and.
188:             & .not.ISLATT(KZMAT(IZZ(I),1)) ) then
189:               COLOR(I) = 16
190:               IBPWK(I) = -1000
191: CCCCCCCCCC
192:             else if ( KZMAT(IZZ(I),1).lt.0 ) then
193:               COLOR(I) = 15
194:               IBPWK(I) = -1
195:             else

```

```

196:       COLOR(I) = 0
197: CCCCCCCCCCCC
198:       IBPWK(I) = -999
199:       IBPWK(I) = -9999
200: C
201:       if ( JCGVDB(1).ne.0 ) then
202:         write(IPR,*) '%%% invalid region I ', I,
203:         & ' IZZ ', IZZ(I), ' KZMAT ',
204:         & KZMAT(IZZ(I),1), ' KZREG ',
205:         & KZREG(IZZ(I))
206:       end if
207:     end if
208:   end if
209:
210: C-----
211: C material # is drawn
212: C-----
213:       else if ( SPTYP.eq.2 ) then
214:         IBPWK(I) = KZMAT(IZZ(I),1)
215:         if ( IBPWK(I).ge.0 ) then
216:           COLOR(I) = KMATCLR(KZMAT(IZZ(I),1))
217:         else if ( IBPWK(I).eq.-1000 ) then
218:           COLOR(I) = 15
219:         else if ( IBPWK(I).eq.-2000 ) then
220:           COLOR(I) = 16
221:         else if ( IBPWK(I).eq.-3000 ) then
222:           COLOR(I) = 16
223:         else if ( IBPWK(I).eq.-4000 ) then
224:           COLOR(I) = 16
225: CCCCCC
226:         else if ( IBPWK(I).gt.-999 ) then
227:         else if ( ISLATT(IBPWK(I)) ) then
228: C
229: C ... STGM region ...
230:
231:         if ( LTYP(ABS(IBPWK(I))).eq.10 .and.
232:         & ((ABS(LEVEL).ge.NEST.and.LEVL(I).le.ABS(LEVEL)).or.
233:         & (ABS(LEVEL).lt.NEST.and.LEVL(I).lt.ABS(LEVEL)))
234:         & ) then
235:           IBPWK(I) = KZMAT(IZZ(I),2)
236:           COLOR(I) = KMATCLR(IBPWK(I))
237:         else
238:           IBPWK(I) = -IDLAT(ABS(IBPWK(I)))
239:           COLOR(I) = CLDAT(ABS(IBPWK(I)))
240: CCCCCC
241:           IBPWK(I) = -IDLAT(LATTNM(IBPWK(I)))
242:           IBPWK(I) = KZMAT(IZZ(I),1)
243:           COLOR(I) = KCLDAT(LATTNM(IBPWK(I)))
244:
245:           if ( JCGVDB(1).ne.0 ) then
246:             write(IPR,*) '%%% lattice material I ', I,
247:             & IBPWK(I),' ZONE ', IZZ(I),' Where ',where
248:           end if
249:         end if
250:       else if ( IMCELL.eq.IZZ(I).and.IBPWK(I).eq.-999 ) then
251:         COLOR(I) = 16
252:       else
253: C
254:       if ( JCGVDB(1).ne.0 ) then
255:         write(IPR,*) '%%% invalid material I ', I,
256:         & ' IZZ ', IZZ(I), ' KZMAT ',
257:         & KZMAT(IZZ(I),1), ' KZREG ', KZREG(IZZ(I))
258:       end if
259: C
260:       COLOR(I) = 0

```

src/cgview/graphs.f

```

261:         end if
262: C
263: C
264:         end if
265: C
266:         BXPLT(I) = XPLT(I)
267:         BYPLT(I) = YPLT(I)
268:         BCOLOR(I) = COLOR(I)
269: C
270:         if ( JLATT.ne.0 ) then
271:             LPOS0(I) = -1
272:             LEVL0(I) = 0
273:         end if
274: C
275:         RFLG = 0
276:         if ( ICBAK.le.199 ) then
277:             KM = KZMAT(IZZ(I),1)
278:             if ( LEVEL.eq.0 .or. KM.ge.0 .or.
279: & (KM.le.-999.and..not.ISLATT(KM)) ) then
280:                 do 100 ICUTI = 1, ICBAK
281:                     if ( BACK(ICUTI).eq.IBPWK(1) ) go to 110
282: 100             continue
283:                 ICBAK = ICBAK + 1
284:                 BACK(ICBAK) = IBPWK(I)
285:             if ( JCGVDB(1).ne.0 ) then
286:                 write(*,*) '%%% new color(S)(BACK) ',BACK(ICBAK),
287: & ' I ',I,
288: & ' IZZ ',IZZ(I),' KZMAT ',KZMAT(IZZ(I),1)
289:             end if
290:         end if
291:
292: 110         continue
293:         end if
294:         end if
295: C
296: 120         continue
297: C
298: C
299: C ... use cap butt for paint mode
300: C
301:         if ( STYLE.eq.0.and.STYLE2.eq.0 ) then
302:             call CAPLINE( 2 )
303: C
304: C ... use cap round
305: C
306:         else
307:             call CAPLINE( 1 )
308:         end if
309: C
310:         go to 230
311:     end if
312: C
313: C
314: C=====
315: C Draw lines when particles are moved to flight stack.
316: C=====
317: C
318: C
319: C ... default line cap style is Round ....
320: C ... use CapButt mode to avoid jaggy boundaries in paint mode
321: C
322:         if ( STYLE.eq.0.and.STYLE2.eq.0 ) then
323:             call CAPLINE( 2 )
324:         else
325:             call CAPLINE( 1 )

```

```

326:         end if
327: C
328:         if ( STYLE.eq.0 ) then
329:             NTP = N2 - N1 + 1
330:         else
331:             NTP = N2 - N1
332:         end if
333: C
334:         if ( NTP.gt.0 ) then
335: C
336:             IPRECLR = -99999
337: C
338:             do 150 J = N1, N2
339:                 I = LSFFL(J)
340:                 ISCAN = LSCAN(I)
341: C
342: C ... input-zone ...
343: C
344:                 if ( SPTYP.eq.-1 ) then
345:                     IZZX(I) = IZZ(I)
346:                     COLOR(I) = ZONCLR(IZZX(I))
347: C
348: C ... input-zone ...
349: C
350:                 else if ( SPTYP.eq.0 ) then
351:                     IZZX(I) = KINPZ(IZZ(I))
352:                     COLOR(I) = IZNCLR(IZZX(I))
353: C
354: C ... region ...
355: C
356:                 else if ( SPTYP.eq.1 ) then
357: C
358:                     if ( JTLT.eq.0 ) then
359:                         IZZX(I) = KZREG(IZZ(I))
360:                     else
361:                         IZZX(I) = IBREG(I)
362:                     end if
363: C
364:                     if ( IKIND.eq.2.and.KZMAT(IZZ(I),1).lt.0 ) then
365:                         COLOR(I) = 12
366:                     else
367:                         COLOR(I) = REGCLR(IZZX(I))
368:                     end if
369: C
370:                     if ( IZZX(I).eq.0 ) then
371: CCCCCCCCC
372: CCCCCCCCC
373:                         if ( KZMAT(IZZ(I),1).le.-999 ) then
374:                             if ( KZMAT(IZZ(I),1).le.-1000 ) then
375:                                 &
376:                                     .not.ISLATT(KZMAT(IZZ(I),1)) ) then
377:                                     COLOR(I) = 16
378:                                     IZZX(I) = -1000
379:                                 else if ( IMCELL.eq.IZZ(I) ) then
380:                                     COLOR(I) = 16
381:                                     IZZX(I) = -999
382:                                 else if ( KZMAT(IZZ(I),1).lt.0 ) then
383:                                     else if ( ISLATT(KZMAT(IZZ(I),1)) ) then
384:                                         COLOR(I) = 15
385:                                         IZZX(I) = -1
386:                                     else
387:                                         COLOR(I) = 0
388:                                         IZZX(I) = -999
389:                                         IZZX(I) = -9999
390:                                     if ( JCGVDB(1).ne.0 ) then
391:                                         write(IPR,*) '%%%', where, ' invalid region I ',

```

src/cgview/graphs.f

```

391:      &                I, ' IZZ ', IZZ(I), ' KZMAT ',
392:      &                KZMAT(IZZ(I),1), ' KZREG ',
393:      &                KZREG(IZZ(I))
394:      write(IPR,*) ' IBPWK ', IBPWK(I), ' IZZX ',
395:      &                IZZX(I)
396:      end if
397: C
398:      end if
399:      end if
400: C
401: C ... material ...
402: C
403:      else if ( SPTYP.eq.2 ) then
404:      IZZX(I) = KZMAT(IZZ(I),1)
405:      if ( IZZX(I).ge.0 ) then
406:      COLOR(I) = KMATCLR(KZMAT(IZZ(I),1))
407:      else if ( IZZX(I).eq.-1000 ) then
408:      COLOR(I) = 15
409:      else if ( IZZX(I).eq.-2000 ) then
410:      COLOR(I) = 16
411:      else if ( IZZX(I).eq.-3000 ) then
412:      COLOR(I) = 16
413:      else if ( IZZX(I).eq.-4000 ) then
414:      COLOR(I) = 16
415:      else if ( IMCELL.eq.IZZ(I) ) then
416:      COLOR(I) = 16
417:
418:      CCCCCC      else if ( IZZX(I).gt.-999 ) then
419:      else if ( ISLATT(IZZX(I)) ) then
420:
421:      ILID = LATTNM(IZZX(I))
422:      if ( LTPY(ILID).eq.10
423:      &
424:      &                .and.
425:      &                ((ABS(LEVEL).ge.NEST.and.LEVL(I).le.ABS(LEVEL)).or.
426:      &                (ABS(LEVEL).lt.NEST.and.LEVL(I).lt.ABS(LEVEL)))
427:      &                ) then
428:
429:      IZZX(I) = KZMAT(IZZ(I),2)
430:      COLOR(I) = KMATCLR(ILID)
431:      COLOR(I) = KMATCLR(IZZX(I))
432:      C1999.12.20 CCCCC      check %%%%%%%%%%%
433:      c      write(*,*) '%%(stg ) level ',levl(i),
434:      c      &                ' izz ',izz(i)
435:      c %%%%%%%%%%%
436:      CCCCCCCCCCCCCC      else
437:      IZZX(I) = -IDLAT(ILID)
438:      COLOR(I) = KCLDAT(ILID)
439:
440:      if ( JCGVDB(1).ne.0 ) then
441:      write(IPR,*) '%% lattice material I ', I,
442:      &                IZZX(I), ' ZONE ', IZZ(I), ' Where ',where
443:      end if
444:      end if
445:      else
446:      COLOR(I) = 0
447:      IZZX(I) = -9999
448: C
449:      if ( JCGVDB(1).ne.0 ) then
450:      write(IPR,*) '%%', where, ' invalid material I ',
451:      &                I, ' IZZ ', IZZ(I), ' KZMAT ',
452:      &                KZMAT(IZZ(I),1), ' KZREG ', KZREG(IZZ(I))
453:      write(IPR,*) ' IBPWK ', IBPWK(I), ' IZZX ', IZZX(I)
454:      end if
455: C

```

```

456:      end if
457:      end if
458: C
459: C .... skip drawing when particle is not moved from previous call
460: C to this routine.
461: C
462:      JDRAW = 1
463:      if ( XPLT(I).eq.BXPLT(I).and.YPLT(I).eq.BYPLT(I) ) then
464:      JDRAW = 0
465:      end if
466: C
467:      KFLG = 0
468: C
469: CCCCC      if ( ICBK.le.199 ) then
470:      if ( JDRAW.ne.0.and.ICBAK.le.199 ) then
471:      do 130 ICUTI = 1, ICBK
472:      if ( BACK(ICUTI).eq.IBPWK(I) ) go to 140
473:      130      continue
474:      if ( GDIST(I).gt.0 ) then
475:      ICBK = ICBK + 1
476:      BACK(ICBK) = IBPWK(I)
477:
478:      if ( JCGVDB(1).ne.0 ) then
479:      write(*,*) '%% new color (BACK) ',BACK(ICBK),
480:      &                ' WHERE <',WHERE,'> I ',I,
481:      &                ' IBPWK ',IBPWK(I), ' IZZX ',IZZX(I),
482:      &                ' IZZ ',IZZ(I), ' KZMAT ',KZMAT(IZZ(I),1)
483:      end if
484:
485:      end if
486:      140      continue
487:      end if
488: C
489: C=====
490: C painting mode (STYLE = 0)
491: C When STYLE2 = 1, only put dots on boundary.
492: C=====
493: C
494:      if ( STYLE.eq.0 ) then
495:
496:      if ( GDIST(I).lt.0.0d0 ) then
497:      COLOR(I) = BCOLOR(I)
498:      XPLT(I) = XPLT(I) + GDIST(I)*AAAG(I)
499:      YPLT(I) = YPLT(I) + GDIST(I)*BBBG(I)
500:      end if
501: C
502:      X001(1) = BXPLT(I)
503:      X001(2) = XPLT(I)
504: C
505:      Y001(1) = BYPLT(I)
506:      Y001(2) = YPLT(I)
507:
508:      if ( STYLE2.eq.0 ) then
509: C
510:      if ( IZZX(I).ne.IBPWK(I) .or. KGCNT(I).le.1
511:      &                .or. GDIST(I).le.0.0D0 ) then
512: C
513:      if ( JDRAW.ne.0 ) then
514:      IPEN = BCOLOR(I)
515:      if ( IPRECLR.ne.IPEN ) then
516:      call NEWP( IPEN )
517:      IPRECLR = IPEN
518:      end if
519:
520:      call LINE( X001, Y001, 2, 1, 0, 0, X(4), Y(4),

```

src/cgview/graphs.f

```

521:      &                X(5), Y(5) )
522:      &                end if
523:
524:      BXPLT(I)    = XPLT(I)
525:      BYPLT(I)    = YPLT(I)
526:      BCOLOR(I)   = COLOR(I)
527:      end if
528: C
529: C      ... boundary drawing mode by STYLE2 = 1 ...
530: C
531: C      else
532: C          IIDT = 0
533: C          IPEN = 0
534: C
535: C      ... when LEVEL < 0 (draw subframe boundary) ...
536: C
537: C      if ( LEVEL.lt.0 .and.
538: C          & (LEVL0(I).ne.LEVL(I) .or.
539: C          & (LEVL(I).gt.0.and.LEVL0(I).eq.LEVL(I)
540: C          & .and.LPOS0(I).ne.LPOS(I,LEVL(I)))
541: C          & ) ) then
542: C          IIDT = 2
543: C          IPEN = 2
544: C      else if ( IZZX(I).ne.IBPWK(I) .or. KGCNT(I).le.1
545: C      else if ( IZZX(I).ne.IBPWK(I)
546: C          & .or. GDIST(I).le.0.0d0 ) then
547: C          IIDT = 1
548: C      end if
549:
550: C      if ( IIDT.ne.0 ) then
551: C          if ( JDRAW.ne.0 ) then
552: C              if ( IPRECLR.ne.IPEN ) then
553: C                  call NEWP( IPEN )
554: C                  IPRECLR = IPEN
555: C              end if
556:
557: Ctemporary %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
558: C          if ( where.eq.'SEAONE1' ) then
559: C              call NEWP( 2 )
560: C          else if ( where.eq.'SEAONE2' ) then
561: C              call NEWP( 0 )
562: C          else if ( where.eq.'LATICE' ) then
563: C              call NEWP( 3 )
564: C          end if
565: C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
566:
567: C          X001(1) = X001(2)
568: C          Y001(1) = Y001(2)
569: C          call LINE( X001, Y001, 2, 1, 0, 0, X(4), Y(4),
570: C          &                X(5), Y(5) )
571: C          &
572: C          end if
573:
574: C          BXPLT(I)    = XPLT(I)
575: C          BYPLT(I)    = YPLT(I)
576: C          BCOLOR(I)   = COLOR(I)
577:
578: C      end if
579: C
580: C          IBPWK(I)    = IZZX(I)
581: C          KGCNT(I)    = KGCNT(I) + 1
582: C
583: C      end if
584: C      150      continue
585: C

```

```

586: C      ... refresh screen ... but calling too many times makes drawing
587: C      on display slower !!!
588: C
589: C          ICALL    = ICALL + 1
590: C          if ( ICALL.gt.INTREFRESH ) then
591: C              ICALL    = 0
592: C              call PLOT( 0.0, 0.0, 777 )
593: C          end if
594: C      end if
595: C
596: C=====
597: C      Boundary curve drawing mode (STYLE=1)
598: C=====
599: C
600: C      if ( STYLE.eq.1.and.N2.gt.N1 ) then
601: C
602: C      .... IZ01(i) = -9009 : no zone change for this path ..
603: C
604: C          do 160 I = 1, N3
605: C              IZ01(I) = -9009
606: C          continue
607: C
608: C      ... check zone change ...
609: C
610: C          do 170 I = N1, N2
611: C              II    = LSFFL(I)
612: C              if ( IBPWK(II).ne.IZZX(II)
613: C                  & .or. (LEVEL.lt.0.and.LPOS0(II).ne.LPOS(II,LEVL(II)))
614: C                  & .or. (LEVEL.lt.0.and.LEVL0(II).ne.LEVL(II)) ) then
615: C                  IZ01(II)    = IZZX(II)
616: C              end if
617: C          170      continue
618: C
619: C      ... ICUT is number of node points in a boundary curve if drawn.
620: C
621: C          ICUT    = 0
622: C
623: C          do 210 J = 1, N3
624: C
625: C              if ( IZ01(J).eq.-9009 ) go to 200
626: C
627: C              if ( JLATT.eq.0 ) then
628: C
629: C      ... non-lattice geometry : only check zone change
630: C
631: C              if ( GDIST(J).gt.0.0d0 ) then
632: C                  ICUT    = ICUT + 1
633: C                  X01(ICUT)    = XPLT(J)
634: C                  Y01(ICUT)    = YPLT(J)
635: C              else
636: C                  go to 200
637: C              end if
638: C
639: C      ... lattice geometry : check nest level ...
640: C
641: C      else
642: C          if ( GDIST(I).gt.0.0d0
643: C              & .and.(LEVL(J).le.ABS(LEVEL).or.LEVL0(J).le.ABS(LEVEL))
644: C              & ) then
645: C              ICUT    = ICUT + 1
646: C              X01(ICUT)    = XPLT(J)
647: C              Y01(ICUT)    = YPLT(J)
648: C          else
649: C              go to 200
650: C

```

src/cgview/graphs.f

```

651:         end if
652:     end if
653: C
654: C ... check connection with other parallel scan lines
655: C
656:         do 180 JJ = J + 1, N3
657:
658:             if ( IZ01(JJ).eq.-9009 ) go to 190
659:
660:             if ( JLATT.eq.0 ) then
661:
662:                 if ( IZ01(J).ne.IZ01(JJ) ) go to 190
663:
664:                 if ( GDIST(JJ).gt.0.0d0 ) then
665: C
666: C ... assuming scaline direction is in just X or Y direction
667: C
668:                     if ( ILOOP.eq.1 ) then
669:                         DLY = ABS(YPLT(JJ)-YPLT(JJ-1))
670:                     else
671:                         DLY = ABS(XPLT(JJ)-XPLT(JJ-1))
672:                     end if
673:                     if ( DLY.lt.DYCG*1.5 ) then
674:                         ICUT = ICUT + 1
675:                         X01(ICUT) = XPLT(JJ)
676:                         Y01(ICUT) = YPLT(JJ)
677:                         IZ01(JJ) = -9009
678:                     else
679:                         go to 190
680:                     end if
681:                 else
682:                     go to 190
683:                 end if
684:             else
685:                 if ( IZ01(J).eq.IZ01(JJ)
686: & .and.LPOS(J,LEVL(J)).eq.LPOS(JJ,LEVL(JJ)) ) then
687:                 if ( LEVL(JJ).le.ABS(LEVEL)
688: & .or. LEVL0(JJ).le.ABS(LEVEL) ) then
689:                     if ( GDIST(JJ).gt.0.0d0 ) then
690:                         if ( ILOOP.eq.1 ) then
691:                             DLY = ABS(YPLT(JJ)-YPLT(JJ-1))
692:                         else
693:                             DLY = ABS(XPLT(JJ)-XPLT(JJ-1))
694:                         end if
695:                         if ( DLY.lt.DYCG*1.5 ) then
696:                             ICUT = ICUT + 1
697:                             X01(ICUT) = XPLT(JJ)
698:                             Y01(ICUT) = YPLT(JJ)
699:                             IZ01(JJ) = -9009
700:                         else
701:                             go to 190
702:                         end if
703:                     else
704:                         IZ01(JJ) = -9009
705:                         go to 190
706:                     end if
707:                 else
708:                     IZ01(JJ) = -9009
709:                     go to 190
710:                 end if
711:             else
712:                 IZ01(JJ) = -9009
713:                 go to 190
714:             end if
715:         end if
716:     180 continue
717: C
718: C PLOT CURVE
719: C
720:     190 continue
721: C
722:         call LINE( X01, Y01, ICUT, 1, 0, 0, X(4), Y(4), X(5), Y(5) )
723: C
724:     200 continue
725:         ICUT = 0
726:     210 continue
727:         ICALL = ICALL + 1
728:         if ( ICALL.gt.INTREFRESH ) then
729:             ICALL = 0
730:             call PLOT( 0.0, 0.0, 777 )
731:         end if
732:     end if
733: C
734:     do 220 I = N1, N2
735:         II = LSFFL(I)
736:         IBPWK(II) = IZZX(II)
737:         if ( JLATT.ne.0 ) then
738:             LEVL0(II) = LEVL(II)
739:             if ( LEVL(II).gt.0 ) then
740:                 LPOS0(II) = LPOS(II,LEVL(II))
741:             else
742:                 LPOS0(II) = -1
743:             end if
744:         end if
745:     220 continue
746: C
747:         call FLCHK( XPLT, YPLT, GDIST, XMAX, YMAX, NZONE, ILOOP, N1,
748: & LSFFL, NFFL, IZFFL, NDEAD )
749: C
750: C
751:     230 continue
752:         return
753:     end

```

src/cgview/gtvval.f

```

1:      subroutine GTVVAL( VNAME, VALUE, IERR )
2: C=<GMVP>=====
3: C
4: C  PURPOSE: RETURN THE VALUE OF AN INTERNAL VARIABLE NAMED VNAME:
5: C  CALLED IN:  DENTAK, PARA & OTHERS
6: C  ARGUMENTS:
7: C      VNAME : VARIABLE NAME (NOT VARIABLE ITSELF !)
8: C      VALUE : VALUE (EXPANDED TO DOUBLE PRECISION IF NECESSARY)
9: C      IERR  : ERROR CODE
10: C          0 : SUCCESSFUL
11: C          1 : NON-EXISTENT VARIABLE NAME OR VALUE FETCHING
12: C          NOT ALLOWED.
13: C
14: C          2 : PROBABLY VALUE UNDEFINED YET
15: C
16: C  CAUTION: THIS ROUTINE IS CALLED FROM UTILITY ROUTINE DENTAK & PARAM
17: C          BUT CODE-DEPENDENT (MVP & GMVP USE DIFFRENT SOURCE FOR
18: C          THIS ROUTINE.
19: C
20: C  HISTORY:  PROGRAMMED BY M.SASAKI   16 MAR 1992
21: C          MODIFIED FOR GMVP       7 FEB 1993
22: C
23: C=====
24:      character*(*) VNAME
25:      real*8 VALUE
26:      integer IERR
27: C
28:      INCLUDE '../shared/INC/_SIZES'
29:      INCLUDE '../shared/INC/_PGEOM'
30:      INCLUDE 'INC/_MVIEW'
31: C
32:      real*8 V0
33: C
34: C      .... VJ : A JUNK VALUE TO CONFIRM VALUE ASSIGNMENT ....
35: C
36:      real*8 VJ
37:      parameter( VJ = -9.9876543D37 )
38: C
39:      IERR = 0
40:      V0 = VJ
41: C
42: C  AVAILABLE VRIABLES :   ( 16 MAR 1992 )
43: C
44: C
45: C---- COMMON /SIZES/ ----
46: C
47:      if ( VNAME(1:2).ge.'NA'.and.VNAME(1:2).le.'NF' ) then
48:          if ( VNAME.eq.'NBATCH' ) V0 = NBATCH
49:          if ( VNAME.eq.'NCELL' ) V0 = NCELL
50:          if ( VNAME.eq.'NEST' ) V0 = NEST
51:          if ( VNAME.eq.'NFBANK' ) V0 = NFBANK
52:      else if ( VNAME(1:2).ge.'NG'.and.VNAME(1:2).le.'NI' ) then
53:          if ( VNAME.eq.'NGP1' ) V0 = NGP1
54:          if ( VNAME.eq.'NGP2' ) V0 = NGP2
55:          if ( VNAME.eq.'NGROUP' ) V0 = NGROUP
56:          if ( VNAME.eq.'NHIST' ) V0 = NHIST
57:          if ( VNAME.eq.'NHSUB' ) V0 = NHSUB
58:          if ( VNAME.eq.'NINPZ' ) V0 = NINPZ
59:      else if ( VNAME(1:2).ge.'NJ'.and.VNAME(1:2).le.'NN' ) then
60:          if ( VNAME.eq.'NMAT' ) V0 = NMAT
61:          if ( VNAME.eq.'NMEMS' ) V0 = NMEMS
62:          if ( VNAME.eq.'NLATT' ) V0 = NLATT
63:          if ( VNAME.eq.'NLLEV' ) V0 = NLLEV
64:          if ( VNAME.eq.'NLBZ' ) V0 = NLBZ
65:      else if ( VNAME(1:2).ge.'NO'.and.VNAME(1:2).le.'NZ' ) then

```

```

66:          if ( VNAME.eq.'NPART' ) V0 = NPART
67:          if ( VNAME.eq.'NZONE' ) V0 = NZONE
68:          if ( VNAME.eq.'NREG' ) V0 = NREG
69:          if ( VNAME.eq.'NSOUR' ) V0 = NSOUR
70:          if ( VNAME.eq.'NRESP' ) V0 = NRESP
71:          if ( VNAME.eq.'NTHIST' ) V0 = NTHIST
72:          if ( VNAME.eq.'NTREG' ) V0 = NTREG
73:          if ( VNAME.eq.'NPICT' ) V0 = NPICT
74: CCCCC      IF( VNAME.EQ. 'NPKIND' ) V0 = NPKIND
75:      else
76:          if ( VNAME.eq.'TCPU' ) V0 = TCPU
77: CCCCC      IF( VNAME.EQ. 'ECUT' ) V0 = BANKP
78:      end if
79:      if ( V0.ne.VJ ) go to 100
80: C
81: C      .... COMMON /PGEOM/ .....
82: C
83:          if ( VNAME.eq.'DEPS' ) V0 = DEPS
84:          if ( VNAME.eq.'DINF' ) V0 = DINF
85:          if ( V0.ne.VJ ) go to 100
86: C
87: C      .... COMMON /PXSEC/ .....
88: C
89:          if ( VNAME.eq.'NGPX' ) V0 = NGPX
90:          if ( VNAME.eq.'NGGX' ) V0 = NGGX
91:          if ( VNAME.eq.'NTGX' ) V0 = NTGX
92:          if ( V0.ne.VJ ) go to 100
93: C
94: C      .... VERIFICATION ...
95: C
96:      100 continue
97:          if ( V0.ne.VJ ) then
98:              VALUE = V0
99:              IERR = 0
100:          else
101:              IERR = 1
102:          end if
103:          return
104:      end

```

src/cgview/helppg.f

```
1:      subroutine HELPPG( IPR, VNM )
2: C=====
3: C Purpose: help message of CGVIEW
4: C=====
5:      character*(*) VNM
6: C=====
7:      7040 format(1X,'=',A)
8:      7060 format(1X,'=',A,A)
9: C
10:     if ( VNM.eq.'HELP' .or. VNM.eq.'H'
11: &       .or. VNM.eq.'?' .or. VNM.eq.'HELP1'
12: &       .or. VNM.eq.'H1' ) then
13:       write(IPR,7060)
14: &===== CGVIEW COMMANDS (page 1/5) =====
15: & '=====
16: & write(IPR,7040)
17: & ' "/" : draw, REDRAW|RD : redraw current window'
18: & write(IPR,7060) ' HELP | H : echo this help message. ',
19: & ' HELPh | Hn: more help (n>1)'
20: & write(IPR,7040) ' HELPW | HW : help for window manipulation.'
21: & write(IPR,7040) ' QUIT : quit program.'
22: & write(IPR,7040) ' TITLE(''title string'')'
23: & write(IPR,7060) ' WHAT | SPTYP(what): what to draw',
24: & ' (-1/0/1/2=zone/i-zone/region/material )'
25: & write(IPR,7040) ' PAPER( cx cy cz dx dy az bx by bz p )'
26: & write(IPR,7060) ' c{x,y,z} : lower left corner, ',
27: & ' {a,b}{x,y,z} : X,Y axes vector'
28: & write(IPR,7040)
29: & ' p : { >0 lines per cm | <0 lines on every |p| pixels}'
30: & write(IPR,7040) ' LINES( p ) : same as p in PAPER.'
31: & write(IPR,7040) ' SIZE | XMAX( xl yl ) : X/Y-Axis length(cm)'
32: & write(IPR,7040) ' XY|YX|XZ|ZX|YZ|ZY : plane orientation'
33: & write(IPR,7040)
34: & ' XAXIS|XA(vx vy vz) YAXIS|YA(vx vy vz) : X/Y-Axis vector'
35: & write(IPR,7040)
36: & ' SCAN( sx sy ) : X|Y-SCAN (0/1/2/3 = none/xy/-yx/+-yx/)'
37: & write(IPR,7040)
38: & ' STYLE | ST( s1 [s2] ) s1 : 0=raster scan/1=boundary curve'
39: & write(IPR,7040)
40: & ' s2 : 0=color paint/1=mark boundary'
41: & write(IPR,7040)
42: & ' PAINT | PNT : paint drawing style (=STYLE(0 0))'
43: & write(IPR,7040)
44: & ' BOUNDARY | BDY : boundary drawing style (=STYLE(0 1))'
45: & write(IPR,7060)
46: &=====
47: & '=====
48: C
49: C
50: C
51: C
52: C
53:     else if ( VNM.eq.'HELP2' .or. VNM.eq.'H2' ) then
54:       write(IPR,7060)
55: &===== CGVIEW COMMANDS (page 2/5) =====
56: & '=====
57: & write(IPR,7040) ' LEVEL( max-geometry-nesting-level-to-draw )'
58: & write(IPR,7040)
59: & ' LEVEL < 0 : draw subframe boundary'
60: & write(IPR,7040)
61: & ' SUBF | NOSUBF : toggle subframe boundary drawing'
62: & write(IPR,7060) ' FLOOD | FL(flag [num]) : ',
63: & ' particle flood check (flag>0) with num particles.'
64: & write(IPR,7040) ' flag=1 - flat, =2 - cosine'
65: & write(IPR,7040)
66: & ' OVERLAP | OVL(flag) : 0/1=on/off overlap check'
```

```
66:     write(IPR,7060) ' DOVERLAP | DOVL(d) : ',
67: & ' torelance for overlap check. (d>0: d*DEPS, d<0:|d|)'
68:     write(IPR,7060) ' ZSEARCH | ZS(flag) : ',
69: & ' Zone search order (0/1=forward/reverse,2=both)'
70:     write(IPR,7060) ' MPLIM | MPL(limit) : ',
71: & ' Overlap message count limit (0/-n=show/unlimited)'
72:     write(IPR,7040) ' CENTER | C(x y z) : centering to (x,y,z)'
73:     write(IPR,7060) ' BFX( body x y z ) : ',
74: & ' for a body, check in/out etc. of point (x,y,z).'
75:     write(IPR,7060) ' CELL( cell-ID [plane] ) : ',
76: & ' cell display (ID=0: non-cell mode, ID<0: inquiry)'
77:     write(IPR,7040)
78: & ' plane=1/-1/2/-2/3/-3=xy(default)/yx/xz/zx/yz/zy'
79:     write(IPR,7060)
80: & ' ZONE | Z( iz ... ) : information of input-zone(s).',
81: & ' (zone |iz| if iz < 0)'
82:     write(IPR,7040)
83: & ' BODY | BD( body-ID ... ) : information of bodies.'
84:     write(IPR,7060) ' MAT ( material-ID ... ) : ',
85: & ' show material composition (MVP only)'
86:     write(IPR,7060)
87: &=====
88: & '=====
89: C
90:     else if ( VNM.eq.'HELP3' .or. VNM.eq.'H3' ) then
91:       write(IPR,7060)
92: &===== CGVIEW COMMANDS (page 3/5) =====
93: & '=====
94: & write(IPR,7060) ' ZOOMZONE|ZZ( i-zone-ID [plane] ) : ',
95: & ' Zoom-in to an input-zone (ID<0: zone-ID)'
96: & write(IPR,7060) ' ZOOMBODY|ZB( body-ID [plane] ) : ',
97: & ' Zoom-in to a body.'
98: & write(IPR,7040) ' F[ORWARD]|B[ACK] : Scene(page) forward|back'
99: & write(IPR,7040)
100: & ' SAVE|S[( id )] : Save current scene parameter with scene ID.'
101: & write(IPR,7040)
102: & ' RESTORE|R[( id )] : Restore saved scene parameter (with ID).'
103: & write(IPR,7060)
104: & ' ZCOLOR | ZC( zone# color# ) : ',
105: & ' Change zone color index.'
106: & write(IPR,7060) ' IZCOLOR | IZC( i-zone# color# ) : ',
107: & ' Change input-zone color index.'
108: & write(IPR,7060) ' MCOLOR | MC( material# color# ) : ',
109: & ' Change material color index.'
110: & write(IPR,7060) ' RCOLOR | RC( region# color# ) : ',
111: & ' Change region color index.'
112: & write(IPR,7060)
113: & ' ZCR|IZCR|MCR|RCR : ',
114: & ' Reset zone|i-zone|material|region color index to default.'
115: & write(IPR,7060)
116: &=====
117: & '=====
118: C
119: C
120:     else if ( VNM.eq.'HELP4' .or. VNM.eq.'H4' ) then
121:       write(IPR,7060)
122: &===== CGVIEW COMMANDS (page 4/5) =====
123: & '=====
124: & write(IPR,7040)
125: & ' WAITOFF | WOFF : do not wait key event on window (PIFF)'
126: & write(IPR,7040)
127: & ' WAITON | WON : wait key event on window (PIFF default)'
128: & write(IPR,7040)
129: & ' PRINT | PR([[command#] [sequence#]]) : output image.'
130: & write(IPR,7040)
```


src/cgview/helppg.f

```
131:      &      ' PRNAME( name ) : base file name of output image (%f).'  
132:      write(IPR,7040) ' CHECK      : echo current scene parameters.'  
133:      write(IPR,7060) ' KMEMO( zone# [search-dir] ) :',  
134:      &      ' show contents of current next-zone memory.'  
135:      write(IPR,7060) ' EVAL | E ( <expression> ) :',  
136:      &      ' print value of numeric expression.'  
137:      write(IPR,7060) ' VERSION | NOVERSION :',  
138:      &      ' display version string or not.'  
139:      write(IPR,7060) ' LOSTSYM | NOLOSTSYM :',  
140:      &      ' display lost path symbol description or not.'  
141:      write(IPR,7060) ' FRAME | NOFRAME :',  
142:      &      ' draw image region frame or not.'  
143:      write(IPR,7060) ' OLDFACE | NEWFACE :',  
144:      &      ' display layout to old style or new style'  
145:      write(IPR,7040) ' REFRESH|RF(int) : display refresh interval'  
146:  
147:      write(IPR,7060) ' MCNP      : generate input for the famous code',  
148:      &      ' ;-) (highly experimental)'  
149:  
150:      write(IPR,7040) ' DEBUG | DB(...) : debug CGVIEW itself'  
151:  
152:      write(IPR,7060)  
153:      &'=====',  
154:      &      '=====',  
155: C  
156: C  
157: C -----  
158: C ... Help mode for Window size(position handling ...  
159: C  
160:      else if ( VNM.eq.'HELPW' .or. VNM.eq.'HW'  
161:      &      .or. VNM.eq.'HELP5' .or. VNM.eq.'H5' ) then  
162:      write(IPR,7060)  
163:      &'==== CGVIEW WINDOW OPERATION COMMANDS (page 5/5) =====',  
164:      &      '=====',  
165:      write(IPR,7040) ' ZOOM | ZM( f [ cx [cy]] )',  
166:      &      ' f : {>1 = zoom in | <1 or <0 = zoom out}',  
167:      &      ' cx,cy : zooming center (default 0.5,0.5)',  
168:      write(IPR,7040) ' RESIZE | RS( dx [ dy ] )',  
169:      &      ' resize X-Y size of window',  
170:      &      ' dx,dy : resize factor ( 0 is no change )'  
171:      write(IPR,7040)  
172:      &      ' MOVE|MOVEA( x y z ) : relative|absolute movement'  
173:      write(IPR,7040) ' ( or MX|MXA(x) MY|MYA(y) MZ|MZA(z) )'  
174:      write(IPR,7040)  
175:      &      ' x,y,z is defined on window(paper) coord. even for MOVEA.'  
176:      &      ' z is relative to X-AXIS length for MOVE.'  
177:      write(IPR,7040) ' ROT( d [x [y]] ) : rotate window',  
178:      &      ' d : anti-clockwise rotation angle(degree)'  
179:      write(IPR,7060)  
180:      &      ' x,y : rotation axis xy position relative to window',  
181:      &      ' (default 0.5,0.5)'  
182:      write(IPR,7040)  
183:      &      ' ROT{X|Y}( d [pos] ) : rotate window round X|Y direction',  
184:      &      ' d : anti-clockwise rotation angle(degree)'  
185:      write(IPR,7060)  
186:      &      ' pos : rotation axis Y|X position relative to window',  
187:      &      ' (default 0.5)'  
188:      write(IPR,7060)  
189:      &'=====',  
190:      &      '=====',  
191:      end if  
192: C  
193:      return  
194:      end
```

src/cgview/igtflg.f

```
1:      function IGTFLG(NAME)
2: C==<MVP>=====
3: C  PURPOSE: GET VALUE OF OPTION PARAMETERS(FLAG) BY NAME
4: C  CALLED IN: ANYWHERE
5: C  CALLS      : (NONE)
6: C=====
7:      include 'INC/_FLAGS'
8:      include '../shared/INC/_IUNIT'
9: C
10:     character*(*) NAME
11: C
12:     if ( NAME.eq.'JDEBG' ) then
13:         IGTFLG = JDEBG(1)
14:     else if ( NAME.eq.'JMCHK' ) then
15:         IGTFLG = JMCHK
16:     else if ( NAME.eq.'JTLLT' ) then
17:         IGTFLG = JTLLT
18:     else
19:         write(IPR,*) 'XXX (IGTFLG) CANNOT GET FLAG VALUE !! (' , NAME,
20:         & ' )'
21:         stop 666
22:     end if
23:     return
24: end
```

src/cgview/inpt1.f

```

1:      subroutine INPTL( IPR,   INP,   IOIN )
2: C=<MVP/GMVP UTILITY>=====
3: C PURPOSE: INPUT FILE LISTING
4: C CALLED IN: main
5: C=====
6: C
7: C   ... linpl : length of line input buffer
8: C   ... leff  : length of effective parts in input lines
9: C
10:      parameter( LINPL   = 80, LEFF  = 72 )
11: c
12:      character LINE*(LINPL)
13:      character DTYP*16
14:      logical OPD
15: c
16:      character*16 CDUMMY
17: C
18: C-----
19: C
20:      NL      = 0
21:      call HEADER( IPR, 'INPUT DATA LISTING' )
22: C
23: C
24:      write(IPR,7000) (I,I=1,LEN(LINE)/10)
25: 7000 format(/1X,5X,' 0',8('...+',I1),4X,'DATA TYPE'/1X,'LINE#')
26: C
27: C
28: C-----
29: C   STATUS   :   ISTAT
30: C   ISTAT = 1 : TITLE
31: C           = 2 : OPTIONS
32: C           = 3 : OTHERS
33: C-----
34: C
35: C
36:      ISTAT   = 1
37:      NEXTRA   = 0
38: C
39: 100 read(INP,'(A)',end =120) LINE
40: c
41: c
42: c   ... line block @PREINP --- @END [PREINP] is lines passed to
43: c   'PREINP' routine which translate extra options.
44: c
45:      if ( ISTAT.eq.1.and.LINE(1:7).eq.'@PREINP' ) then
46:      write(IPR,
47:      &      '/1x,' '!!! @PREINP block is placed before input.' ' ' )
48: c
49:      IPREIN   = 1
50: c
51: 110 read(INP,'(A)',end =120) LINE
52:      if ( LINE(1:4).eq.'@END' ) then
53:      write(IPR,'(/1x,' ' @PREINP Block ended.' ' ' )
54:      go to 100
55:      else
56: c
57: c   ... call preinp routine.
58: c
59: c   Attention: MVP and GMVP both have their own 'PREINP' routine.
60: c
61: c      call preinp(iprein, line, ntask0, cdummy, mlimit )
62: c      go to 110
63: c   end if
64: c   end if
65: c

```

```

66: c
67:      NL      = NL + 1
68: c
69:      if ( NL.eq.1.and.INP.ne.IOIN ) then
70:      inquire(IOIN,opened =OPD)
71:      if ( .not.OPD ) open( IOIN, status = 'UNKNOWN', form =
72:      &      'FORMATTED' )
73:      end if
74: c
75:      IEXTRA   = 0
76: C
77:      if ( NL.le.2 ) then
78:      DTYP      = 'TITLE'
79:      else
80:      if ( LINE(1:1).eq.'*' ) then
81:      DTYP      = 'COMMENT'
82:      else
83:      if ( LINE(LEFF+1:).ne.' ' ) then
84:      IEXTRA   = 1
85:      NEXTRA   = NEXTRA + 1
86:      end if
87:      if ( LINE(1:1).eq.'%' ) then
88:      DTYP      = 'PARAMETER LINE'
89:      else
90:      if ( ISTAT.eq.1 ) ISTAT = 2
91:      if ( ISTAT.eq.2 ) then
92:      if ( LINE(1:LEFF).ne.' ' ) then
93:      DTYP      = 'OPTION'
94:      else
95:      DTYP      = ' '
96:      ISTAT     = 3
97:      end if
98:      else if ( ISTAT.eq.3 ) then
99:      DTYP      = 'DATA'
100:      end if
101:      end if
102:      end if
103:      end if
104: C
105:      if ( IEXTRA.eq.0 ) then
106:      write(IPR,'(1X,I5,' ' : '' ,A,' ' : '' ,A)') NL, LINE, DTYP
107:      else
108:      write(IPR,'(1X,I5,' ' :>' ,A,' ' : '' ,A)') NL, LINE, DTYP
109:      end if
110: C
111:      if ( INP.ne.IOIN ) then
112: cccc WRITE(IOIN,'(A72,I8)') LINE(1:72),NL
113:      write(IOIN,'(A)') LINE(1:LEFF)
114:      end if
115: C
116:      go to 100
117: C
118: C
119: 120 write(IPR,7020) (I,I=1,LEN(LINE)/10), NL
120: 7020 format(/1X,'      ' 0',8('....+',I1))// ' TOTAL LINES : ',I5//
121: C
122:      if ( NEXTRA.gt.0 ) then
123:      write(IPR,'(1x,a,i3,a/1x,a)')
124:      &      '      !!! Some input lines have extra characters after ',
125:      &      LEFF, ' ' 'th column.',
126:      &      '      Such lines are marked as "line#:>".'
127:      call CNTERR( 'WARNING' )
128:      end if
129: C
130:      rewind IOIN

```

src/cgview/inpt1.f

```
131: C
132:   return
133: end
```

SAFE

src/cgview/intro.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine INTRO( TITLE, MLIMIT, CODE )
3: C=====
4: C PURPOSE:  DATA INPUT & PREPARATION FOR RANDOM WALK.
5: C CALLED IN: CENTER
6: C
7: C=====
8: C/#IF MS_VISUAL
9: C
10: C ... This interface block is for CUTIL & MS-Visual tools. ...
11: C
12: C   interface
13: C     subroutine FLUSHSTD()
14: C#DEC$   ATTRIBUTES C :: FLUSHSTD
15: C     end subroutine
16: C   end interface
17: C/#ENDIF
18: C
19: C   character TITLE(2)*72
20: C   character*(*) CODE
21: C
22: C   include 'INC/_KPIDS'
23: C   include 'INC/_KPSYMS'
24: C   include 'INC/_NGPS'
25: C
26: C   include '../shared/INC/_ARRAY'
27: C   include 'INC/_FLAGS'
28: C   include '../shared/INC/_SIZES'
29: C   include 'INC/_XBANK'
30: C   include 'INC/_STACK'
31: C   include 'INC/_XWORK'
32: C   include '../shared/INC/_PGEOM'
33: C   include '../shared/INC/_PVRED'
34: C   include '../shared/INC/_PTALY0'
35: C   include '../shared/INC/_PTLSP'
36: C   include '../shared/INC/_WORDL'
37: C   include '../shared/INC/_IOUNIT'
38: C
39: C   include '../mvp/INC/_PXSEC'
40: C
41: C   ... CGVIEW specific global data ...
42: C
43: C   include 'INC/_MVIEW'
44: C   include 'INC/_MVGEO'
45: C
46: C ... for MVP type $CROSS SECTION block
47: C
48: C   include '../mvp/INC/_CXSEC'
49: C
50: C   character VNM*6, VNAME*72, VSUBF*1
51: C
52: C   ... dummy value for cutoff time
53: C
54: C   parameter( TCUTDM = -1.0E30 )
55: C
56: C-----
57: C
58: C ... check unopened scratch file
59: C
60: C   call OPENSC( 'MVP' )
61: C
62: C ... initialize random number generator
63: C
64: C   call RNINIT( 1 )
65: C
66: C   ... DATA INPUT UNIT SETTING ....
67: C
68: C   call RIUNIT( IOIN )
69: C
70: C   .... PASS VERSION DESCRIPTION TO HEADER ROUTINE...
71: C
72: Ccccc CALL HVERSN( 'SLICE V2.0' )
73: Ccccc CALL HVERSN( 'SLICE V2.2' )
74: Ccccc CALL HVERSN( 'SLICE V2.4' )
75: Ccccc CALL HVERSN( 'SLICE V2.5beta' )
76: Ccccc call HVERSN( 'SLICE V3.0pre1' )
77: C   call VERSTR
78: C
79: C..... ECHOBACK PRINTING OF INPUT DATA .....
80: C
81: C#sasa CALL INPLST( IPR,IOIN,IOIN )
82: C
83: C ..... PAGE HEADER ....
84: C
85: C   call HEADER( IPR, 'PROBLEM TITLE & OPTIONS' )
86: C
87: C ..... READ TITLE
88: C
89: C   read(IOIN,'(A72)') (TITLE(I),I=1,2)
90: C   write(IPR,7000) (TITLE(I),I=1,2)
91: C   7000 format('1',130('='))/' ',A72/' ',A72/' ',130('='))
92: C
93: C ..... READ OPTION PARAMETERS
94: C
95: C   call OPTION( MLIMIT )
96: C
97: C   .... MEMORY ALLOCATION CHECK ....
98: C   LSTT : STARTING POSITION INDEX IN DYNAMIC MEMORY ARRAY A
99: C   IN COMMON /ARRAY/ .
100: C
101: C   call MEMALC( IPR, LSTT, LSTTL, IERR )
102: C
103: C   write(IPR,'(1H1)')
104: C
105: C ..... DEFAULT VALUES (TRUE DEFAULT VALUES OF DEPS & DINF WILL
106: C   BE GIVEN IN 'GEOMIN'.)
107: C   DEPS = 0.0
108: C   DINF = 0.0
109: C   if ( JTIME.ne.0 ) then
110: C     TCUT = TCUTDM
111: C   else
112: C     TCUT = 0.0
113: C   end if
114: C
115: C   NGROUP = 0
116: C   do 100 I = 1, KPLIM
117: C     NGP(I) = 0
118: C     KNGP(I) = 0
119: C     KENGP(I) = 0
120: C   100 continue
121: C   KNGP(KPLIM+1) = 0
122: C   KENGP(KPLIM+1) = 0
123: C
124: C   NZONE = 0
125: C   NINPZ = 0
126: C   NSURF = 0
127: C   NSDA = 0
128: C   NZDA = 0
129: C   NMEMO = 0
130: C   NMAT = 0
```

src/cgview/intro.f

```

131:      NREG      = 0
132:      NTIME     = 0
133:      NSOUR     = 0
134:      NMEMS     = 0
135:      NRESP     = 0
136:      NRESP2    = 0
137:      NRSKIP    = 0
138:      NSKIP     = 0
139:      NGP1      = 0
140:      NGP2      = 0
141:      NSPACE    = 0
142:      NBODY     = 0
143:      NTREG     = 0
144:      NSTALY    = 0
145:      NETALY    = 0
146:      NDTALY    = 0
147:      NLDTAL    = 0
148:      NLDIAL    = 0
149:      NBINMX    = 0
150:      IRAND     = 0
151: C
152:      NMKREG     = 0
153: C
154:      WLLIM      = 1.0E-10
155: C
156: C ..... CLEAR SOME POINTERS .....
157: C
158:      NHIST      = 0
159:      NBANK      = 0
160: C
161:      LXIMP      = 0
162:      LWKIL      = 0
163:      LWSRV      = 0
164:      LWTIME     = 0
165:      LPSXYZ     = 0
166:      LPSALP     = 0
167:      LVOL       = 0
168:      LRVOL      = 0
169:      LTRVOL     = 0
170: C
171:      LWGTF      = 0
172:      LWGTP      = 0
173: C
174: C ... initialize extra bank parameter index
175:   do 110 K = 0, MDBNK
176:     KDBNK(K) = 0
177: 110 continue
178:   do 120 K = 0, MIBNK
179:     KIBNK(K) = 0
180: 120 continue
181: C
182: C ... local flags ...
183:      ICHK       = 0
184:      ICK        = 0
185:      LKR        = 0
186: C
187: C ..... JNP = 1 : MEANS NEUTRON - PHOTON JOINT PROBLEM
188: C
189:      JNP        = JNEUT*JPHOT
190: C
191: C ..... READ OTHER DATA .....
192: C
193: C .... TO SET NULL ADDRESS ASSIGNMENT CHCKER ....
194: C
195: C .... INITIAL VALUE OF 'LAST' IS SET IN THIS ROUTINE ...

```

```

196: C
197:      call CKNUL0( A, LAST, LSTT )
198: C
199: C
200: 130 call FREADS( VNAME, NLEN, IEND )
201: C
202:      if ( IEND.ne.0 ) go to 140
203:      ICALL      = 0
204: C
205: C ..... $CROSS SECTION or $XSEC block .....
206: C
207:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$XSEC'
208: & .or. VNAME(1:6).eq.'$CROSS' ) then
209: C        CALL CPUTM(TT0)
210: C
211: C      ... XSEC DUMMY READ ...
212:      if ( CODE.ne.'MVP' ) then
213:        call DMXSEC( VNAME(:NLEN) )
214:      else
215: C
216: C      ... Try input as MVP's $CROSS SECTION block ...
217: C      Program may abort when GMVP input is passed to this
218: C      procedure ...
219: C
220:      JXSKIP     = 1
221:      call CTXSIN( A, A, CHA, LIMIT, LIMITC, JXSKIP )
222:      end if
223: C
224:      call RESET
225: C
226:      go to 130
227:      end if
228: C
229: C ..... GEOMETRY .....
230: C
231:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$GEOM' ) then
232: C        CALL CPUTM(TT0)
233: C        CALL GEOMIN(ICALL, LAST, MAXSF,
234: C & JDEBG, JHLAT, JLATT, JREFL, JSIMP, JTLLT )
235: C
236:      call GEOMIN( ICALL, A, LIMIT, CHA, LIMITC, LAST, MAXSF, JDEBG,
237: C & JHLAT, JLATT, JREFL, JSIMP, JTLLT, JFISX, JSTGR )
238: C
239: C        CALL CPUTM(TT1)
240: C        WRITE(IPR,*) ' ==== CPU FOR GEOMETRY PROCESSING ', TT1-TT0, ' SEC'
241: C        call FLUSHSTD()
242: C        go to 130
243: C      end if
244: C
245: C      .... "REGION"-DEPENDENT DATA
246: C
247: C      !REGION-NAME( VNAME(DATA) .... )
248: C OR VNAME( !REGION-NAME( DATA ) .... )
249: C OR VNAME( DATA .... )
250: C
251:      IPPP       = INDEX(VNAME(:NLEN),'.') - 1
252:      if ( IPPP.lt.0 ) IPPP = NLEN
253:      if ( VNAME(1:1).eq.'!' .or. VNAME(1:IPPP).eq.'XIMP'
254: & .or. VNAME(1:IPPP).eq.'WKIL' .or. VNAME(1:IPPP).eq.'WSRV'
255: & .or. VNAME(1:IPPP).eq.'WGTF' .or. VNAME(1:IPPP).eq.'WGTP'
256: & .or. VNAME(1:IPPP).eq.'RVOL' .or. VNAME(1:IPPP).eq.'TRVOL'
257: & .or. VNAME(1:IPPP).eq.'PSALP' ) then
258: C
259: C      .... TEMPORARY WORKING ARRAY ....
260: C

```

src/cgview/intro.f

```

261:      NRRRO  = MAX(NREG,NTREG)
262:      if ( LKR.eq.0 ) then
263:        call KEEP( 'KR', LKR, NRRRO, 'I4', LAST, JDEBG )
264:      end if
265: C
266: C
267: c##<2007/03/14:PN3: added by Y.Nagaya 2008/10/30
268:      NUCPN_DMY = 0
269: c##>
270:      call REGDAT( MSG, VNAME(:NLEN), VNM, VSUBF, A, A, LIMIT, LAST,
271: &      JNEUT, JPHOT, JDEBG, NREG, NGP(1), JKPAR(1), KPSYM,
272: &      KPLIM, KNGP(1), NGROUP, NGP1, NGP2, LXIMP, LWKIL,
273: &      LWSRV, LWGTF, LWGTP, LRVOL, LTRVOL, LPSALP, JTLT,
274: &      A(LKMAT), A(LKREG), A(LKREGI), A(LKCELI), NINPZ,
275: &      A(LISPNM), A(LISUSE), A(LKSUZN), A(LIRGSP), NEST,
276: &      NSPACE, NSUZON, NZONE, CHA(LTNAMS), NNAMES, A(LITRNM),
277: &      NTRREG, A(LIPTRG), A(LLPTRG), A(LKR), NRRRO,
278: c##<2007/03/14:PN3: added by Y.Nagaya 2008/10/30
279: &      NUCPN_DMY, NUCPN_DMY, NUCPN_DMY )
280: c##>
281: C
282: C
283: C
284:      ICALL  = 1
285:      go to 130
286: end if
287: C
288: C ..... SOURCE .....
289: C
290:      if ( NLEN.ge.5.and.VNAME(1:5).eq.'$SOUR' ) then
291:        call DMSOUR
292:        call RESET
293:        go to 130
294:      end if
295: C
296: C ..... Other blocks unnecessary(?) to CGVIEW .....
297: C
298:      if ( VNAME(1:1).eq.'$' ) then
299:        write(IPR,*) ' !!! SKIP BLOCK <', VNAME(:NLEN), '>'
300:        call DMBLCK( VNAME(:NLEN) )
301:        call RESET
302:        go to 130
303:      end if
304: C
305: C ..... OTHERS (SIZE PARAMETER ETC.) .....
306: C
307: C .....
308: C ..... EXTRACT 'VARIABLE NAME' FROM VNAME .....
309: C .....
310: C
311:      VNM      = ' '
312:      VSUBF     = ' '
313:      IPER      = INDEX(VNAME,'.')
314:      IPR1      = IPER + 1
315: C
316: C ..... VNAME WITH SUBFIELD .....
317: C
318:      if ( IPER.ne.0 ) then
319:        VNM(1:IPER-1) = VNAME(1:IPER-1)
320:        VSUBF = VNAME(IPER+1:IPER+1)
321:        IPR2 = INDEX('NP',VSUBF(1:1))
322:        if ( IPR2.eq.0 ) then
323:          write(IPR,'(1X,A,A,A,A,A,A)') ' XXX SUFFIX( ' , VSUBF,
324: &          ' ) OF ' , VNAME, ' IS, ' INCORRECT XXX '
325:          call CNTERR( 'FATAL' )

```

```

326:      call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
327:      go to 130
328: end if
329: C
330: C ..... VNAME WITHOUT SUBFIELD .....
331: C
332: else
333:      NLLL = MIN(NLEN,LEN(VNM))
334:      VNM(1:NLLL) = VNAME(1:NLLL)
335: end if
336: C
337:      if ( VNM.eq.'DEPS ' ) call VALUE0( VNM, ICALL, 'R8', DEPS )
338:      if ( VNM.eq.'DINF ' ) call VALUE0( VNM, ICALL, 'R8', DINF )
339:      if ( VNM.eq.'NPART ' ) call VALUE0( VNM, ICALL, 'I4', NPART )
340: C      IF(VNM.EQ. 'NHIST ' ) CALL VALUE0(VNM,ICALL, 'I4', NHIST )
341:      if ( VNM.eq.'NBANK ' ) call VALUE0( VNM, ICALL, 'I4', NBANK )
342:      if ( VNM.eq.'NMEMO ' ) call VALUE0( VNM, ICALL, 'I4', NMEMO )
343:      if ( VNM.eq.'NPIC' ) call VALUE0( VNM, ICALL, 'I4', NPIC )
344:      if ( VNM.eq.'NMEMS ' ) call VALUE0( VNM, ICALL, 'I4', NMEMS )
345: C
346:      if ( VNM.eq.'NGROUP' ) then
347:        if ( VSUBF.eq.' ' .or. JNP.eq.0 ) then
348:          call VALUE0( VNAME, ICALL, 'I4', NGROUP )
349:          if ( JNEUT.ne.0 ) NGP1 = NGROUP
350:          if ( JPHOT.ne.0 ) NGP2 = NGROUP
351:        else if ( VSUBF.eq.'N' ) then
352:          call VALUE0( VNAME, ICALL, 'I4', NGP1 )
353:          if ( JPHOT.eq.0 ) NGROUP = NGP1
354:          if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
355:        else if ( VSUBF.eq.'P' ) then
356:          call VALUE0( VNAME, ICALL, 'I4', NGP2 )
357:          if ( JNEUT.eq.0 ) NGROUP = NGP2
358:          if ( JNP.ne.0 ) NGROUP = NGP1 + NGP2
359:        end if
360:      end if
361:      if ( VNM.eq.'NGP1' ) call VALUE0( VNAME, ICALL, 'I4', NGP1 )
362:      if ( VNM.eq.'NGP2' ) call VALUE0( VNAME, ICALL, 'I4', NGP2 )
363: C
364: C ..... OTHERS (ARRAYS)
365: C
366: C ...../PGEOM/.....
367: C
368:      if ( VNM.eq.'KMAT' .or. VNM.eq.'KREG' ) then
369:        write(IPR,7020)
370:        7020 format(/1X,' XXX OLD-FASHIONED INPUT FOR MAT # & REGION #.'/
371: &        ' SPECIFYING "KMAT" & "KREG" SEPARATELY FROM ZONE DATA '
372: &        ',
373: &        'PORTION OF GEOMETRY INPUT IS NOT ALLOWED IN NEW VERSION.')
374:        call CNTERR( 'FATAL' )
375:        call DMREAD( VNM, IDUMMY, IDUMMY, IRET )
376:      end if
377: C
378: C .....
379: C .... target position of path stretching .....
380: C .....
381: C
382:      if ( VNM.eq.'PSXYZ' ) then
383: C
384:        call ARAYIN( VNAME, A, ICALL, 'R8', LPSXYZ, ICK, JDEBG, LAST,
385: &        3, ' ' )
386: C
387:      if ( MOD(NFINP(0),3).ne.0 ) then
388:        write(IPR,7040) NFINP(0)
389:        7040 format(/1X,'XXX Number of input data PSXYZ() is 'I4,
390: &        ' while a multiple of 3 is required.')

```

src/cgview/intro.f

```

391:      call CNTERR( 'FATAL' )
392:      else if ( NFINP(0).eq.0 ) then
393:        call PUTVD( A(LPSXYZ), 3, 0.0D0 )
394:      end if
395:    end if
396:  C
397:  C
398:  C ..... TRIED TO INPUT INVALID ARRAY OR VARIABLES !!!! .....
399:  C
400:      if ( ICALL.eq.0 ) then
401:        WRITE(IPR,7100) VNM
402: C7100      FORMAT(' !!! ARRAY OR VARIABLE ',A6,'> DOES NOT EXIST ',
403: C          'OR INHIBITTED DIRECT INPUT !!! ')
404:        call DMREAD( VNAME, IDUMMY, IDUMMY, IRET )
405:      end if
406:      go to 130
407: C=====
408:  C
409:  C      INPUT END .....
410:  C
411:      140 if ( ICHK.ne.0 ) then
412:        write(IPR,7060)
413: C7060      format('1'///' ',10('XXXXXXXXXX')/' ',10('XXXXXXXXXX')/' '
414: C          & ' YOUR INPUT INCLUDES DATA OF INVALID SIGN !! '
415: C          & ' ' ' PLEASE CHECK INPUT DATA OR ERROR-MESSAGES !!! '
416: C          & '/// ',10('XXXXXXXXXX')/' ',10('XXXXXXXXXX'))
417:        call CNTERR( 'FATAL' )
418:      else if ( ICK.ne.0 ) then
419:        write(IPR,7080)
420: C7080      format('1'///1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX')/1X,
421: C          & ' YOUR INPUT INCLUDES DATA OF INVALID LENGTH !! '
422: C          & '1X,' PLEASE CHECK INPUT DATA OR ERROR-MESSAGES !!! '
423: C          & '//1X,10('XXXXXXXXXX')/1X,10('XXXXXXXXXX'))
424:        call CNTERR( 'FATAL' )
425:      end if
426:      call FLUSHSTD()
427:  C ... KEEP MEMORY AREAS FOR NON-INPUT PARAMETERS & GIVE VALUES TO THEM
428:  C
429:  C      .... /SIZES/ .....
430:  C
431:      if ( NBANK.le.0.and.NHIST.gt.0 ) NBANK = NHIST
432:  C
433:  C      if ( NBANK.gt.1000 .or. NBANK.lt.10 ) NBANK = 1000
434:  C      if ( NBANK.gt.500 .or. NBANK.lt.10 ) NBANK = 500
435:  C
436:      NHIST = NBANK
437:  C
438:  C      .... /PGEOM/ .....
439:  C
440: C-----
441:  C ..... TRANSLATE MAT ID # TO MAT # R(FOR MVP: USE IN FUTURE FOR GMVP
442: C-----
443:  C
444:  C      CALL MID2NM( A(LKZMAT),A(LKMAT),A(LKINPZ),
445: C          @      NINPZ,NZONE,JIMAG, A(LIDMAT), NMAT )
446:  C
447: C-----
448:  C
449:      if ( NMEMO.le.0 ) then
450:        NMEMO = 5
451:        write(IPR,7100) NMEMO
452:        call CNTERR( 'WARNING' )
453:  C
454:  C7100      format(/1X,'!!! DATA "NMEMO" WAS SET TO DEFAULT VALUE (',I4,
455: C          & ' ). ' /

```

```

456:      & ' "NMEMO" IS THE NUMBER OF MEMORIES OF NEXT-ENTERING-ZONES' /
457:      & 'FOR EACH ZONE, AND IT IS DESIRABLE THAT "NMEMO" BE THE' /
458:      & 'MAXIMUM NUMBER OF POSSIBLE NEXT-ZONES.' /
459:      & ' THIS PARAMETER MIGHT BE IMPORTANT TO REDUCE TIME CONSUMED',
460:      & 'FOR PARTICLE TRACKING IN SOME CASES. ' / )
461:  C
462:  C      end if
463:  C
464:  C      call KEPV( 'KMEMO ', LKMEMO, NMEMO*NZONE, 'I4', LAST, 0, JDEBG )
465:  C      call KEPV( 'MEMC ', LMEMC, NMEMO*NZONE, 'I4', LAST, 0, JDEBG )
466:  C      call KEPV( 'MEMZ ', LMEMZ, NZONE, 'I4', LAST, 0, JDEBG )
467:  C
468:  C      ... "KMEMO" arrays for CGVIEW has extra components for
469:  C          reversed zone search mode.
470:  C
471:  C      call KEPV( A(1), 'KMEMO2', LKMEMO2, NMEMO*NZONE*2, 'I4', LAST, 0,
472: C          & JDEBG )
473:  C      call KEPV( A(1), 'MEMC2 ', LMEMC2, NMEMO*NZONE*2, 'I4', LAST, 0,
474: C          & JDEBG )
475:  C      call KEPV( A(1), 'MEMZ2 ', LMEMZ2, NZONE*2, 'I4', LAST, 0, JDEBG )
476:  C
477:  C          (ZONCLR, REGCLR, MATCLR )
478:  C
479:  C      call KEPV( A(1), 'PAPER ', LPAPER, 30, 'R8', LAST, 0.0D0, JDEBG )
480:  C      call KEEP( 'ZONCLR', LZONCLR, NZONE, 'I4', LAST, JDEBG )
481:  C      call KEEP( 'IZNCLR', LIZNCLR, NINPZ, 'I4', LAST, JDEBG )
482:  C      call KEEP( 'REGCLR', LREGCLR, NREG, 'I4', LAST, JDEBG )
483:  C      call KEEP( 'MATCLR', LMATCLR, 1+MXMAT, 'I4', LAST, JDEBG )
484:  C
485: C-----
486:  C ..... VARIANCE REDUCTION PARAMETER .....
487: C-----
488:  C
489: C=====
490:  C      call HEADER( IPR, 'VARIANCE REDUCTION PARAMETERS' )
491:  C      =====
492:  C
493:      if ( JIMPT.ne.0.and.LXIMP.eq.0 ) then
494:        call KEPV( A(1), 'XIMP', LXIMP, NGROUP*NREG, 'R4', LAST, 1.0,
495: C          & JDEBG )
496:      end if
497:      if ( JRRLT.ne.0 .or. JWWND.ne.0 ) then
498:        if ( LWKIL.eq.0 ) then
499:          call KEPV( A(1), 'WKIL', LWKIL, NGROUP*NREG, 'R4', LAST,
500: C          & 1.E-5, JDEBG )
501:        end if
502:        if ( LWSRV.eq.0 ) then
503:          call KEPV( A(1), 'WSRV', LWSRV, NGROUP*NREG, 'R4', LAST,
504: C          & 1.E-4, JDEBG )
505:        end if
506:      end if
507:  C
508:  C
509:      if ( LSTCK(0).lt.0 ) then
510:        write(IPR,7180) 'variance reduction parameter printout.'
511:        call PRSTOP( 1, 'MEMORY OVER.' )
512:      end if
513:  C
514:  C      if ( JIMPT.ne.0 ) then
515:  C
516: C=====
517: C          call LABEL( IPR, 'IMPORTANCE (GROUP & REGION-WISE)' )
518: C          =====
519:  C
520:  C      call R4PRNT( 'XIMP', A(LXIMP), NGROUP, NREG, 'GROUP', 'REGION',

```


src/cgview/intro.f

```

521:      &          IPR )
522:      end if
523: C
524:      if ( JRRLT.ne.0 .or. JWWND.ne.0 ) then
525: C *****
526:      call LABEL( IPR, 'WEIGHT THRESHOULD FOR RUSSIAN-ROULETTE' )
527: C *****
528: C
529:      call R4PRNT( 'WKIL', A(LWKIL), NGROUP, NREG, 'GROUP', 'REGION',
530: &          IPR )
531: C
532: C *****
533:      call LABEL( IPR, 'SURVIVAL WEIGHT IN RUSSIAN-ROULETTE' )
534: C *****
535: C
536:      call R4PRNT( 'WSRV', A(LWSRV), NGROUP, NREG, 'GROUP', 'REGION',
537: &          IPR )
538: C
539: C ... check consistency of WSRV and WKIL
540: C
541:      call CKWGT( A(LWSRV), A(LWKIL), NGROUP, NREG )
542: C
543:      end if
544: C
545:      if ( JTIME.ne.0.and.LWTIME.ne.0 ) then
546:          JWTIM = 1
547: C *****
548:      call LABEL( IPR, 'TIME BIN WEIGHTING FACTOR' )
549: C *****
550: C
551:      call R4PRNT( 'WTIME', A(LWTIME), NTIME, 1, 'TIME', ' ', IPR )
552: C
553: C ... WTIME must be non positive ...
554:      call CKVAL4( 'WTIME', A(LWTIME), NTIME, IFL, 1 )
555:      if ( IFL.ne.0 ) then
556:          write(IPR,*) 'XXX Time bin weight factor WTIME includes ',
557: &          IFL, ' non positive value.'
558:          call CNTERR( 'FATAL' )
559:      end if
560:      end if
561: C
562:      if ( JPSTR.ne.0 ) then
563: C *****
564:      call LABEL( IPR, 'PATH STRETCHING PARAMETERS' )
565: C *****
566:      if ( JLATT.ne.0 ) then
567:          write(IPR,*)
568: &          '!!! Both PATH-STRETCHING option and LATTICE option ',
569: &          ' are selected.',
570: &          ' Path stretching in lattice region may not be effective',
571: &          ' in current version.'
572:          call CNTERR( 'WARNING' )
573:      end if
574:      if ( LPSXYZ.eq.0 ) then
575:          write(IPR,*) 'XXX PATH-STRETCHING option is selected but',
576: &          ' no target points PSXYZ(x y z) are given.'
577:          call CNTERR( 'FATAL' )
578:      else
579:          write(IPR,7120)
580: 7120      format(/1X,3X,'=== target point coordinates ===')
581:          call PRTXYZ( IPR, 'PSXYZ', A(LPSXYZ) )
582:      end if
583:      if ( LPSALP.ne.0 ) then
584:          call R4PRNT( 'PSALP', A(LPSALP), NREG, 1, 'REGION', ' ', IPR
585: &          )

```

```

586:          call CKALP( A(LPSALP), NREG )
587:      end if
588:      end if
589: C
590: C ... CGVIEW specific data
591: C
592:      call KEPV( A(1), 'XGMCK', LXGMCK, 4*MIPGMCK, 'R8', LAST, 0.0D0,
593: &          JBEDG )
594:      call KEPV( A(1), 'IGMCK', LIGMCK, NZONE, 'I4', LAST, 0, JBEDG )
595:      call KEPV( A(1), 'IPGMCK', LIPGMCK, 3*MIPGMCK, 'I4', LAST, 0,
596: &          JBEDG )
597:      call KEPV( A(1), 'NDEFSRC', LNDEFSRC, NINPZ, 'I4', LAST, 0, JBEDG
598: &          )
599: C
600: C ===== particle stack =====
601: C
602:      call LKEPV( H(1), 'LSFFL', LLSFFL, NBANK, 'I4', LAST, 0, JBEDG )
603:      call LKEPV( H(1), 'LNFFL', LNFFL, NZONE+1, 'I4', LAST, 0, JBEDG )
604:      call LKEPV( H(1), 'LIZFFL', LIZFFL, NBANK, 'I4', LAST, 0, JBEDG )
605:      NCOLS = 0
606:      call LKEPV( H(1), 'LSSRC', LLSSRC, NBANK, 'I4', LAST, 0, JBEDG )
607:      call LKEPV( H(1), 'LNXT', LNNXT, NZONE+1, 'I4', LAST, 0, JBEDG )
608:      call LKEPV( H(1), 'LIZNXT', LIZNXT, NBANK, 'I4', LAST, 0, JBEDG )
609:      IDEFER = 0
610:      NDEAD = 0
611: C
612: C ..... IN THE CASE OF REFLECTIVE BOUNDARY .....
613: C
614: C ... in CGVIEW, keep memory for KSREF even if no-reflective surface
615: C exists, because distance calculation to the nearest
616: C surface may be necessary in JUDGE routine by JRR=.true. .
617: C
618:      call KEPV( A(1), 'KSREF', LKSREF, NZDA, 'I4', LAST, 0, JBEDG )
619:      call LKEPV( H(1), 'LSREF', LLSREF, NBANK, 'I4', LAST, 0, JBEDG )
620: C
621:      if ( JREFL.ne.0 ) then
622: C
623: C
624:      call LKEPV( H(1), 'IZREF', LIZREF, NBANK, 'I4', LAST, 0, JBEDG
625: &          )
626:      call LKEPV( H(1), 'ISREF', LISREF, NBANK, 'I4', LAST, 0, JBEDG
627: &          )
628: C ..... SEARCH REFLECTIVE BOUNDARIES .....
629: C      LKKREF = LAST
630:      call LRMAIN( 'KKREF', NLTEMP, 'I4', LAST )
631:      call LKEEP( 'KKREF', LKKREF, NLTEMP, 'I4', LAST, JDEBG )
632: C
633:      call REFZON( A(LSDA), A(LKZDA), A(LKZAA), A(LKZMAT), A(LKSREF),
634: &          NZDA, NZONE, A(LKKREF), NLTEMP, NREFS, JDEBG )
635: C
636:      call LRSIZE( 'KKREF', LKKREF, 2*NREFS, 'I4', LAST )
637:      call LKEPV( H(1), 'NBREF', LNBREF, NREFS+1, 'I4', LAST, 0,
638: &          JDEBG )
639:      else
640:          call LKEPV( H(1), 'NBREF', LNBREF, 2, 'I4', LAST, 0, JBEDG )
641:      end if
642: C
643: C ..... IN THE CASE OF LATTICE GEOMETRY .....
644: C
645:      if ( JLATT.ne.0 ) then
646:          call LKEPV( H(1), 'LSLAT', LLSLAT, NBANK, 'I4', LAST, 0, JBEDG
647: &          )
648:          call LKEPV( H(1), 'LIZLAT', LIZLAT, NBANK, 'I4', LAST, 0, JBEDG
649: &          )
650:          call LKEPV( H(1), 'NLXT', LNLXT, NLBZ+1, 'I4', LAST, 0, JBEDG

```

src/cgview/intro.f

```

651:      &
652:      end if
653: C
654:      write(IPR,*) ' '
655:      write(IPR,*) ' MEMORY FOR STACK DATA: FROM ', LLSFFL, ' TO ', LAST
656:      &
657:      write(IPR,*) ' '
658: C
659: C ===== partilce bank =====
660: C
661: C ..... /XBANK/ .....
662: C
663:      call LKEPV( H(1), 'XXX ', LXXX, NBANK, 'R8', LAST, 0D0, JBEDG )
664:      call LKEPV( H(1), 'YYY ', LYYY, NBANK, 'R8', LAST, 0D0, JBEDG )
665:      call LKEPV( H(1), 'ZZZ ', LZZZ, NBANK, 'R8', LAST, 0D0, JBEDG )
666:      call LKEPV( H(1), 'AAA ', LAAA, NBANK, 'R8', LAST, 0D0, JBEDG )
667:      call LKEPV( H(1), 'BBB ', LBBB, NBANK, 'R8', LAST, 0D0, JBEDG )
668:      call LKEPV( H(1), 'CCC ', LCCC, NBANK, 'R8', LAST, 0D0, JBEDG )
669:      call LKEPV( H(1), 'IZZ ', LIZZ, NBANK, 'I4', LAST, 0, JBEDG )
670: C ... 3/6/1991 ...
671:      call LKEPV( H(1), 'KLSF ', LKLSF, NBANK, 'I4', LAST, 0, JBEDG )
672: C
673: C LXPLT   R8 GRAPHICS WORK
674: C LYPLT   R8 GRAPHICS WORK
675: C LIBPWK  I4 ZONE      WORK
676: C LLSDED  I4 ZONE      WORK
677: C
678:      call LKEPV( H(1), 'XPLT ', LXPLT, NBANK, 'R8', LAST, 0D0, JBEDG )
679:      call LKEPV( H(1), 'YPLT ', LYPLT, NBANK, 'R8', LAST, 0D0, JBEDG )
680:      call LKEPV( H(1), 'LSCAN ', LLSCAN, NBANK, 'I4', LAST, 0, JBEDG )
681: C
682: C ... X,Y,POINT BACKUP BANK
683: C
684:      call LKEPV( H(1), 'BXPLT ', LBXPLT, NBANK, 'R8', LAST, 0D0, JBEDG )
685:      &
686:      call LKEPV( H(1), 'BYPLT ', LBYPLT, NBANK, 'R8', LAST, 0D0, JBEDG )
687:      &
688: C
689: C ... COLOR BANK
690: C
691:      call LKEPV( H(1), 'COLOR ', LCOLOR, NBANK, 'I4', LAST, 0, JBEDG )
692: C
693: C ... COLOR BACKUP BANK
694: C
695:      call LKEPV( H(1), 'BCOLOR', LBCOLOR, NBANK, 'I4', LAST, 0, JBEDG )
696:      call LKEPV( H(1), 'IBPWK ', LIBPWK, NBANK, 'I4', LAST, 0, JBEDG )
697: C
698: C ... zone search direction flag
699: C
700:      call LKEPV( H(1), 'IZSS ', LIZSS, NBANK, 'I4', LAST, 0, JBEDG )
701: C
702: C ... scanline direction
703: C
704:      call LKEPV( H(1), 'AAAG ', LAAAG, NBANK, 'R8', LAST, 0D0, JBEDG )
705:      call LKEPV( H(1), 'BBBG ', LBBBG, NBANK, 'R8', LAST, 0D0, JBEDG )
706:      call LKEPV( H(1), 'CCCG ', LCCCG, NBANK, 'R8', LAST, 0D0, JBEDG )
707:      call LKEPV( H(1), 'GDIST', LGDIST, NBANK, 'R8', LAST, 0D0, JBEDG )
708:      call LKEPV( H(1), 'KGCNT', LKGCNT, NBANK, 'I4', LAST, 0, JBEDG )
709: C
710: C ... "dead particle stack" LSDED for MVP/GMVP is particle status
711: C      bank in CGVIEW
712: C      ( a monologue of a hacker of CGVIEW ...
713: C      why did not use different name ??? boooooo... !!!)
714: C
715:      call LKEPV( H(1), 'LSDED ', LLSDED, NBANK, 'I4', LAST, 0, JBEDG )

```

```

716: C
717: C.....  BANK DATA FOR LATTICE GEOMETRY .....
718: C
719: C      IF(JLATT.NE.0) THEN
720: C
721:      call LKEPV( H(1), 'LEVL', LLEVL, NBANK, 'I4', LAST, 0, JBEDG )
722:      call LKEPV( H(1), 'LZZ ', LLZZ, NBANK*NEST, 'I4', LAST, 0, JBEDG )
723:      call LKEPV( H(1), 'LPOS', LLPOS, NBANK*NEST, 'I4', LAST, 0, JBEDG )
724:      &
725:      call LKEPV( H(1), 'LPOS0 ', LLPOS0, NBANK, 'I4', LAST, 0, JBEDG )
726:      call LKEPV( H(1), 'LEVLO ', LLEVLO, NBANK, 'I4', LAST, 0, JBEDG )
727:      if ( JHLAT.ne.0 )
728:      & call LKEPV( H(1), 'LCRS', LLCRS, NBANK*NEST, 'I4', LAST, 0,
729:      & JBEDG )
730: C      ENDIF
731: C
732:      if ( JTLT.ne.0 ) then
733:          N1 = NEST + 1
734:          call LKEPV( H(1), 'IBSPC', LIBSPC, NBANK*N1, 'I4', LAST, 0,
735:      & JDEBG )
736:      end if
737: C
738: C ... always keep IBREG (from Jan 2000)
739:      call LKEPV( H(1), 'IBREG', LIBREG, NBANK, 'I4', LAST, 0, JDEBG )
740: C
741: C
742:      call LKEPV( H(1), 'NLOST', LNLOST, 1, 'I4', LAST, 0, JDEBG )
743: C
744: C-----
745: C.....  optional bank data .....
746: C-----
747: C
748: C ... save particle weight on birth
749: C      if ( JRWVR.ne.0 ) then
750: C          KDBNK(1) = 1
751: C      end if
752: C
753: C ... save distance to lattice frames (NEST+1 banks)
754: C      and position on frame(to be reached) and direction
755: C      if ( JFISX.ne.0 ) then
756: C          KDBNK(4) = 1
757: C          KDBNK(6) = 1
758: C      end if
759: C
760: C ... save position of STG cell (3*NEST banks)
761: C      if ( JSTGR.ne.0 ) then
762: C          KDBNK(5) = 1
763: C      end if
764: C
765: C      NNND = 0
766: C      do 150 K = 1, MDBNK
767: C          if ( KDBNK(K).ne.0 ) then
768: C              KDBNK(K) = NNND + 1
769: C              if ( K.eq.4 ) then
770: C                  NNND = NNND + NEST + 1
771: C              else if ( K.eq.5 ) then
772: C                  NNND = NNND + 3*NEST
773: C              else if ( K.eq.6 ) then
774: C                  NNND = NNND + 6*NEST
775: C              else
776: C                  NNND = NNND + 1
777: C              end if
778: C          end if
779: C      150 continue
780: C      KDBNK(0) = NNND

```

src/cgview/intro.f

```

781:      call LKEPV( H(1), 'DBNK', LDBNK, NBANK*NNND, 'R8', LAST, OD0,
782:      &          JDEBG )
783: C
784: C ... save flight count in a path by default in CGVIEW
785: C
786:      KIBNK(5)    = 1
787: C
788:      if ( JFISX.ne.0 ) then
789: C ... save flight count in a path
790: CCCC      KIBNK(5)    = 1
791: C ... lattice level giving the smallest distance until the level
792:      KIBNK(6)    = 1
793: C ... required NND type
794:      KIBNK(7)    = 1
795:      end if
796: C
797:      if ( JDEBG(6).gt.0 ) then
798:      KIBNK(8)    = 1
799:      end if
800: C
801: C ... marker region passing flag (currently not used in CGVIEW)
802: C
803:      if ( KIBNK(9).gt.0 ) then
804:      KIBNK(9)    = NMKREG
805:      end if
806: C
807:      NNNI      = 0
808:      do 160 K = 1, MIBNK
809:      if ( KIBNK(K).ne.0 ) then
810:      KIBNK(K)    = NNNI + 1
811:      if ( K.eq.6 ) then
812:      NNNI      = NNNI + NEST
813:      else if ( K.eq.8 ) then
814:      NNNI      = NNNI + JDEBG(6)
815:      else
816:      NNNI      = NNNI + 1
817:      end if
818:      end if
819: 160 continue
820:      KIBNK(0)    = NNNI
821:      call LKEPV( H(1), 'IBNK', LIBNK, NBANK*NNNI, 'I4', LAST, 0, JDEBG
822:      &          )
823: C
824:      write(IPR,*) ' MEMORY FOR PARTICLE BANK: FROM ', LXXX, ' TO ',
825:      &          LAST - 1
826:      write(IPR,*) ' '
827: C
828: C .....
829:      write(IPR,*) ' TOTAL OF MEMORY EXCEPT WORKING AREA = ', LAST - 1,
830:      &          ' (WORD) '
831: C
832: C ..... /XWORK/ .....
833: C ..... KEEP WORK AREA OF LENGTH NZONE+1 OR NREFS+1
834: C
835:      call WRKARY( LWORK, LAST, LIMIT, MAXSF )
836: C
837: C
838:      write(IPR,*) ' TOTAL OF MEMORY = ', LSIZ(LAST) - 1, ' (WORD) '
839:      write(IPR,*) ' '
840:      if ( LSIZ(LAST-1).gt.LIMIT ) then
841:      write(IPR,*) ' XXX MEMORY OVER !! LIMIT = ', LIMIT, ' (WORD) '
842:      call CNTERR( 'FATAL ' )
843:      stop 999
844:      end if
845: C .....

```

```

846: C ..... INPUT OF RESTART FILE & PREPARATION FOR TALLY
847: C
848: C      CALL CPUTM(T1)
849: C      WRITE(IPR,
850: C      @          '(1H0,' CPU FOR DATA PREPARATION = ',F9.3,' (SEC)')')
851: C      @          T1-T0
852: C
853: C ..... PRINT CONTENTS OF COMMON /SIZES/
854: C
855:      call PRSIZE
856: C
857: C-----
858: C ..... CHECK INPUT OR PROCESSING ERROR .....
859: C-----
860: C
861:      call PRNERR( IPR, 'AFTER DATA INPUT' )
862:      call CHKERR( 'WARNING ', KK )
863:      if ( KK.gt.0 ) then
864: 7140      format(' *** SOME WARNING MESSAGES HAVE BEEN ISSUED. ****')
865:      &          ' I RECOMMEND YOU CHECKING YOUR INPUT DATA',
866:      &          ' ONCE MORE. '//' WARNING MESSAGE IS PRINTED ',
867:      &          'WITH '!!!' SYMBOL ON THE HEAD OF PRINTED LINE.')
868:      end if
869: C
870:      JSTOP      = 0
871:      call CHKERR( 'FATAL ', KK )
872: C
873:      if ( KK.gt.0 ) then
874:      JSTOP      = 1
875:      write(IPR,7160)
876: 7160      format('1','*** FATAL ERRORS HAVE BEEN DETECTED',
877:      &          ' IN INPUT-PHASE. STOP EXECUTION. ****')
878:      &          ' MESSAGE FOR FATAL ERROR IS PRINTED ',
879:      &          'WITH 'XXX' SYMBOL ON THE HEAD OF PRINTED LINE.'//
880:      &          ' YOU MUST CHECK YOUR INPUT DATA',' CAREFULLY !! ')
881:      stop 888
882:      end if
883: C
884: 7180 format(/1X,'XXX(INTRO2) Cannot proceed input processing any more',
885:      &          ' because of memory shortage.'/1X,' * Stop before ',A)
886: C
887:      return
888:      end

```

src/cgview/inum.f

```
1: c
2: c-----
3: c
4:     function INUM(AA,BB)
5:     integer AA, BB
6:     INUM      = AA + BB
7:     return
8:     end
```



src/cgview/jnumber.f

```
1: CC/#IF .NOT.FSYMBOL
2: *      subroutine JNUMBER( X, Y, HEIGHT,FNUM, THETA, NDIGIT )
3: *      call NUMBER( X, Y, HEIGHT,FNUM, THETA, NDIGIT )
4: *      return
5: CCC      end
6: CC/#ELSE
7: C*****
8: C      NUMBER DRAWING
9: C*****
10: C
11: C ===== A SUBROUTINE EQUIVALENT TO 'NUMBER' OF THE CALCOMP
12: C      PLOTTER LIBRARY ( 1990 FEB. 15   MADE BY M.SASAKI )
13: C
14: C      subroutine JNUMBER( X, Y, HEIGHT,FNUM,THETA,NDIGIT,JXPOS,JYPOS )
15: C-----
16: C      X, Y : string position
17: C      JXPOS = 0,1,2 : left/center/right
18: C      JYPOS = 0,1,2 : lower/center/upper
19: C-----
20: C      character*9 NUM, NUM2, FOMT*8
21: C
22: C      NUM = ' '
23: C      if ( NDIGIT.gt.0 ) then
24: C        write(FOMT,'(F9.0,I1,'}') NDIGIT
25: C        write(NUM,fmt =FOMT) FNUM
26: C      else
27: C        write(NUM,fmt ='(I9)' ) INT(SIGN(1.0,FNUM)*INT(ABS(FNUM)))
28: C      end if
29: C
30: C      K      = 0
31: C      do 100 I = 1, len(NUM)
32: C        if ( NUM(I:I).ne.' ' ) then
33: C          K      = K + 1
34: C          NUM2(K:K) = NUM(I:I)
35: C        end if
36: C      100 continue
37: C
38: C      ... get string length in XL0
39: C
40: C      XL0 = 0.0
41: C      YL0 = 0.0
42: C      TH0 = -999.0
43: C      call JSYMBOL( XL0, YL0, HEIGHT, NUM2(:K), TH0 , K )
44: C
45: C      if ( JXPOS.eq.1 ) then
46: C        XX = X - XL0*0.5
47: C      else if ( JXPOS.eq.2 ) then
48: C        XX = X - XL0
49: C      else
50: C        XX = X
51: C      end if
52: C
53: C      if ( JYPOS.eq.1 ) then
54: C        YY = Y-HEIGHT*0.5
55: C      else if ( JYPOS.eq.2 ) then
56: C        YY = Y-HEIGHT
57: C      else
58: C        YY = Y
59: C      end if
60: C
61: C      call JSYMBOL( XX, YY, HEIGHT, NUM2(:K), THETA, K )
62: C
63: C      if ( NODISP.eq.0 ) call XXFLUSH( )
64: C
65: C      return
66: CC/#ENDIF
67:      end
```

src/cgview/jsymbol.f

```

1: CC/IF .NOT.FSYMBOL
2: *      subroutine JSYMBOL( XI,      YI,      HIGHT, NBCD0, THETA, NC )
3: *      character NBCD0(4)*1
4: *      call symbol(XI,      YI,      HIGHT, NBCD0, THETA, NC )
5: *      return
6: *      end
7: CC/ELSE
8:      subroutine JSYMBOL( XI,      YI,      HIGHT, NBCD0, THETA, NC )
9: C=====
10: C
11: C  Calc comp type character plotting routine having some extensions.
12: C
13: C  Using only calcomp type PLOT routine as actual drawing.
14: C
15: C  Programmed by M.Sasaki ( unknown dates in 1980's ... )
16: C
17: C=====
18: C  Extensions from original calcomp routine:
19: C
20: C  * possible to draw lowercase characters, greek characters
21: C  backspace and upper/lower subscripts using "shift" characters.
22: C
23: C  * when THETA = -999.0 and YI=0.0, actual character length on
24: C  display is returned on XI.
25: C
26: C  * when HIGHT < 0, parameters such as character aspect ratio,
27: C  spacing are modified.
28: C
29: C=====
30: C
31: C/IF MS_VISUAL
32: C
33: C ... This interface block is for CUTIL & MS-Visual tools. ...
34: C
35: *      interface
36: *      subroutine PLOT( RRR1, RRR2, III1 )
37: *      integer      III1
38: *      real          RRR1, RRR2
39: *CDEC$      ATTRIBUTES C :: PLOT
40: *CDEC$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
41: *      end subroutine
42: *      end interface
43: C/ENDIF
44: C
45:      character NBCD0(4)*1
46: C
47: C===== GLOBAL DATA SECTION =====
48: C
49:      common /SASAKI/ XB,      YB,      HTC,      THC,      ASPECT, TALIC,
50:      &      SPACE,      ISTYLE, NLET,      NADDR(300),      LSHIFT(72),
51:      &      JT(3,3000)
52:      integer JT
53:      common /SASQA/ CH(8),      GREEK, SFTKEY, SFTCH,      SMALL, BACKSP,
54:      &      SUB,      SUPER
55:      character*20 CH
56:      character CHAR*160, GREEK*48
57:      equivalence( CH(1), CHAR )
58:      character SFTCH*72, SFTKEY*1
59:      character*1 SMALL, BACKSP, SUB, SUPER
60: C
61: C ... local data ...
62: C
63:      dimension NCODE(30)
64:      character AA*4, NBCD(400)*1
65:      character*1 SCODE, ECODE
66:      character      CCDD*6
67: C
68:      parameter ( LCWRK = 200 )
69:      character*1 CWRK(LCWRK)
70:      dimension XXXXXX(1)
71:      equivalence(XB,XXXXXX(1))
72: C
73: C ... shift code to embed CALCOMP character code ...
74: C      [code#] --> plotted as CALCOMP character having the number
75: C
76:      data SCODE /['/', ECODE /['/']]/
77: C
78: C===== OPERATION SECTION =====
79: C
80: C ---- SET MODE -----
81: C
82: C ----- THETA = -999.0 : Calculate actual length of character string
83: C
84:      if ( THETA.eq.-999.0.and.YI.ne.0.0 ) go to 220
85: C
86: C ----- HIGHT < 0 : RESET SIZE PARAMETERS
87: C
88:      if ( HIGHT.lt.0.0 ) then
89:          if ( XI.ne.999.0 ) ASPECT = XI
90:          if ( YI.ne.999.0 ) TALIC = YI
91:          if ( THETA.ne.999.0 ) SPACE = THETA
92:          if ( NC.ne.999 ) ISTYLE = NC
93:          return
94:      end if
95: C
96: C ---- WRITE MODE -----
97: C
98:      if ( HIGHT.eq.999.0 ) HIGHT = HTC
99:      if ( THETA.eq.999.0 ) THETA = THC
100:      WIDTH = HIGHT*ASPECT
101:      SP = WIDTH*SPACE
102:      C = COS(THETA*(3.14159265/180.))
103:      S = SIN(THETA*(3.14159265/180.))
104:      CCX = C*WIDTH
105:      SSX = S*WIDTH
106:      CCY = C*HIGHT
107:      SSY = S*HIGHT
108:      CCSP = C*SP
109:      SSSP = S*SP
110:      if ( XI.ne.999.0 ) XB = XI
111:      if ( XI.ne.999.0 ) YB = YI
112:      ISUB = 0
113: C
114: C ***** ONE CHARACTER *****
115: C
116:      if ( NC.le.0 ) then
117:          AA = NBCD0(1) //NBCD0(2) //NBCD0(3) //NBCD0(4)
118:          read(AA,'(A4)') IBCD
119: C
120:          if ( NC.eq.0.and.IBCD.gt.127 ) then
121:              L = INDEX(CHAR,NBCD0(4))
122:              if ( NBCD0(4).eq.' ' ) L = 65
123:          else
124:              L = IBCD + 1
125:          end if
126: C
127:      do 100 I = NADDR(L), NADDR(L+1) - 1
128:          J1 = JT(1,I)
129:          J2 = JT(2,I)
130:          XYJ1 = JT(1,I) / (2.0**14)

```

src/cgview/jsymbol.f

```

131:      XYJ2      = JT(2,I) /(2.0**14)
132:      XT        = XYJ1 + TALIC*XYJ2
133:      XX        = XB + CCX*XT - SSY*XYJ2
134:      YY        = YB + SSX*XT + CCY*XYJ2
135:
136:      if ( I.eq.NADDR(L).and.NC.le.-2 ) then
137:        call PLOT( XX, YY, 2 )
138:      else
139:        IP        = JT(3,I)
140:        call PLOT( XX, YY, IP )
141:      end if
142:
143: 100 continue
144:
145:      XB        = XB + CCSP
146:      YB        = YB + SSSP
147:      return
148:    end if
149:  C
150:  C ***** CHARACTER OF ONE LINE *****
151:  C
152:  C
153: 110 ISHIFT = 0
154:
155:      do I = 1, NC
156:        NBKD(I) = NBKD0(I)
157:      end do
158:  C
159:  CC =====
160:  C
161:      MC        = 0
162:      KSF        = 0
163:      NCD        = 0
164:      KCD        = 0
165:
166:      do 150 I = 1, NC
167:
168:  CCCC ----- KCD = 1: CHARACTER CODE NUMBER -----
169:
170:        if ( KCD.eq.1 ) then
171:          if ( NBKD(I).eq.ECODE ) KCD = 0
172:          go to 150
173:        end if
174:  CCCC
175:        MC      = MC + 1
176:        if ( MC.gt.LCWRK ) then
177:          write(6,*) ' XXX TOO LONG CHARACTER STRING',
178:            & ' FOR TRANSFORMATION OF EMBEDDED CHARACTER CODES',
179:            & ' XXX STOP'
180:          stop 666
181:        end if
182:        CWRK(MC) = NBKD(I)
183:
184:  C ----- CHARACTER CODE FOUND -----
185:
186:      check %%%
187:  C      write(*,*) ' I ',I,' NBKD <','NBKD(I),'>'
188:  C %%%%%%%%%
189:      if ( NBKD(I).eq.SCODE ) then
190:        KCD = 1
191:        NCD = NCD + 1
192:
193:  C ----- GET CHARACTER CODE # -----
194:
195:      CCDD      = ' '
196:  C
197:  C ... fixed ( 27 Jan 2000 )
198:  C
199:  CCCCC      do 130 KK = I + 1, LCWRK
200:  C
201:        do 130 KK = I + 1, NC
202:          if ( NBKD(KK).eq.ECODE ) go to 140
203:          CCDD(KK-I:KK-I) = NBKD(KK)
204: 130 continue
205:
206: 140      KKK      = KK - I - 1
207:      check %%%
208:  C      write(*,*) ' I ',I,' <','ccdd,'> NCD ',NCD, ' MC ',MC
209:  C %%%%%%%%%
210:          read(CCDD,'(BN,I6)') NCODE(NCD)
211:        end if
212: 150 continue
213:
214:      do I = 1, MC
215:        NBKD(I) = CWRK(I)
216:      end do
217:  C
218:  C == TO GET CHARACTER LENGTH =====
219:  C
220:      if ( THETA.eq.-999.0 ) go to 200
221:  C
222:  C == PLOTTING =====
223:  C
224:      KCD      = 0
225:
226:      do 190 I = 1, MC
227:
228:  C ----- CHARACTER CODE # -----
229:
230:        if ( NBKD(I).eq.SCODE ) then
231:          KCD = KCD + 1
232:          L = NCODE(KCD) + 1
233:          go to 170
234:        end if
235:
236:  C -----
237:
238:        if ( NBKD(I).eq.' ' ) then
239:          L = 65
240:          go to 170
241:        end if
242:
243:        if ( ISHIFT.eq.0 ) then
244:
245:          if ( NBKD(I).eq.SFTKEY ) then
246:            ISHIFT = 1
247:            go to 190
248:          end if
249:
250:          L = INDEX(CHAR,NBKD(I))
251:          go to 170
252:        else
253:
254:          if ( NBKD(I).eq.SFTKEY ) then
255:            ISHIFT = 0
256:            go to 190
257:          end if
258:  C
259:          J      = INDEX(GREEK,NBKD(I))
260:

```

src/cgview/jsymbol.f

```

261:         if ( J.gt.0 ) then
262:             L = 154 + J
263:             go to 170
264:         end if
265:
266:         K = INDEX(SFTCH,NBCD(I))
267:         if ( K.gt.0 ) L = LSHIFT(K) + 1
268:         if ( K.eq.0 ) L = INDEX(CHAR,NBCD(I))
269:     end if
270:
271: C << CODE = 25 : NO OPERATION >>
272:
273: 170 if ( L.eq.26 ) go to 190
274:
275: C << CODE = 46,47 : SUB- OR SUPER- SCRIPT >>
276:
277: if ( NBCD(I).eq.SUB .or. NBCD(I).eq.SUPER ) then
278:     if ( NBCD(I).eq.SUB ) L = 48
279:     if ( NBCD(I).eq.SUPER ) L = 47
280:     if ( ISUB.eq.0 ) then
281:         ISUB = 1
282:         XB = XB - SSY*(REAL(47-L)+0.8333)
283:         YB = YB + CCY*(0.87*REAL(47-L)+0.7000)
284:         CCX = 0.6*CCX
285:         CCY = 0.6*CCY
286:         SSX = 0.6*SSX
287:         SSY = 0.6*SSY
288:         CCSP = 0.6*CCSP
289:         SSSP = 0.6*SSSP
290:     else
291:         ISUB = 0
292:         CCX = CCX/0.6
293:         CCY = CCY/0.6
294:         SSX = SSX/0.6
295:         SSY = SSY/0.6
296:         CCSP = CCSP/0.6
297:         SSSP = SSSP/0.6
298:         XB = XB + SSY*(REAL(47-L)+0.8333)
299:         YB = YB - CCY*(0.87*REAL(47-L)+0.7000)
300:     end if
301:
302:     go to 190
303: end if
304:
305: CC << CODE = 17 : BACK SPACE >>
306:
307: if ( NBCD(I).eq.BACKSP ) then
308:     XB = XB - CCSP
309:     YB = YB - SSSP
310:     go to 190
311: end if
312:
313: C <<<<< DRAWING LETTER >>>>>
314:
315: do 180 J = NADDR(L), NADDR(L+1) - 1
316:     XYJ1 = real(JT(1,J)) /(2.0**14)
317:     XYJ2 = real(JT(2,J)) /(2.0**14)
318:     XT = XYJ1 + TALIC*XYJ2
319:     XX = XB + CCX*XT - SSY*XYJ2
320:     YY = YB + SSX*XT + CCY*XYJ2
321:     IP = JT(3,J)
322:     call PLOT( XX, YY, IP )
323: 180 continue
324:
325:     XB = XB + CCSP

```

```

326:         YB = YB + SSSP
327: 190 continue
328:
329:         HTC = HIGHT
330:         THC = THETA
331:         return
332: C
333: C *** CALCULATE THE LENGTH OF CHARACTER STRING ***
334: C
335: 200 XD = 0.0
336:     SP = HIGHT*ASPECT*SPACE
337:
338:     do 210 I = 1, MC
339:         if ( NBCD(I).eq.SFTKEY ) go to 210
340: C
341:         L = INDEX(CHAR,NBCD(I))
342: C
343: C << CODE = 25 : NO OPERATION >>
344:
345:         if ( L.eq.26 ) go to 210
346:
347: C << CODE = 46,47 : SUB- OR SUPER- SCRIPT >>
348:
349:         if ( NBCD(I).eq.SUB .or. NBCD(I).eq.SUPER ) then
350:
351:             if ( ISUB.eq.0 ) then
352:                 ISUB = 1
353:                 SP = 0.6*SP
354:             else
355:                 ISUB = 0
356:                 SP = SP/0.6
357:             end if
358:
359:             go to 210
360:         end if
361:
362: C << CODE = 17 : BACK SPACE >>
363:
364:         if ( NBCD(I).eq.BACKSP ) then
365:             XD = XD - SP
366:             go to 210
367:         end if
368:
369:         XD = XD + SP
370: 210 continue
371:
372:         XI = XD
373:         return
374: C
375: C *** DUMP CONTENTS OF /SASAKI/ FROM N1 TO N2 ***
376: C
377: 220 write(AA,'(A4)') XI
378:     read(AA,'(A4)') N1
379:     write(AA,'(A4)') YI
380:     read(AA,'(A4)') N2
381:
382:     do 240 I = N1, N2
383:         K = (I-N1)*4
384:         write(AA,'(A4)') XXXXXX(I)
385:
386:     do 230 J = 1, 4
387:         NBCD(K+J) = AA(J:J)
388: 230 continue
389: 240 continue
390: C

```


src/cgview/jsymbol.f

```
391:      return
392: C
393:      end
394: C
395: C-----
396: C
397: C Block data is moved to the file including SYMBOL, because
398: C GNU-Fortran 0.5.18 does not initialize common data in block data
399: C sub program when placed separately.
400: C
401: C-----
402: C
403: C ----- CHARACTER DATA (VERSION 2 , 5/29/1987 )
404: C BLOCK DATA JIBIKI
405: C
406: C ===== MAY 1989 == JAIS =====
407: C COMMON /SASAKI/ XB,YB,HTC,THC,ASPECT,TALIC,
408: C * SPACE,IDUMMY, NLET,NADDR(300), LSHIFT(72)
409: C * JT(3,3000)
410: C CHARACTER*1 SFTKEY,SFTCH*72,SMALL,BACKSP,SUB,SUPER
411: C COMMON /SASAQ /
412: C * CH(8),GREEK,SFTKEY,SFTCH,SMALL,BACKSP,SUB,SUPER
413: C =====
414: C
415: C CHARACTER CH*20,GREEK*48
416: C INTEGER JT
417: C
418: C DATA XB,YB,HTC,THC,ASPECT,TALIC,SPACE,IDUMMY /
419: C * 0.0,0.0,2.80000E+00,0.0,1.00000E+00,0.0,9.50000E-01,0.0 /
420: C
421: C DATA NLET / 213 /
422: C
423: C DATA (NADDR(I),I= 1, 100)/
424: C * 1, 8, 26, 31, 35, 39, 45, 50, 54, 60
425: C * , 65, 77, 85, 91, 93, 102, 104, 106, 107, 110
426: C * , 116, 121, 122, 128, 134, 136, 137, 139, 157, 168
427: C * , 171, 184, 210, 225, 240, 254, 264, 283, 302, 315
428: C * , 327, 354, 366, 384, 411, 434, 450, 451, 452, 459
429: C * , 471, 476, 481, 485, 489, 493, 495, 508, 512, 518
430: C * , 526, 531, 535, 540, 545, 546, 568, 583, 598, 611
431: C * , 626, 635, 656, 668, 682, 702, 711, 714, 721, 725
432: C * , 727, 751, 772, 778, 782, 805, 817, 836, 851, 868
433: C * , 878, 887, 896, 902, 909, 928, 931, 933, 935, 963
434: C * /
435: C DATA (NADDR(I),I= 101, 200)/
436: C * 975, 988, 991, 996, 1000, 1005, 1009, 1042, 1052, 1088
437: C * , 1090, 1093, 1112, 1133, 1138, 1151, 1170, 1175, 1192, 1215
438: C * , 1218, 1251, 1274, 1292, 1300, 1335, 1339, 1343, 1351, 1356
439: C * , 1378, 1396, 1409, 1416, 1421, 1441, 1447, 1453, 1465, 1471
440: C * , 1475, 1480, 1484, 1503, 1517, 1538, 1552, 1580, 1584, 1595
441: C * , 1598, 1603, 1607, 1612, 1616, 1634, 1659, 1670, 1697, 1720
442: C * , 1744, 1760, 1779, 1789, 1799, 1811, 1825, 1834, 1862, 1883
443: C * , 1893, 1911, 1930, 1943, 1958, 1977, 1989, 2002, 2029, 2034
444: C * , 2056, 2060, 2064, 2071, 2075, 2081, 2104, 2110, 2116, 2121
445: C * , 2126, 2130, 2144, 2165, 2175, 2187, 2194, 2200, 2209, 2236
446: C * /
447: C DATA (NADDR(I),I= 201, 300)/
448: C * 2240, 2257, 2276, 2293, 2313, 2334, 2360, 2364, 2373, 2377
449: C * , 2381, 2386, 2394, 2458, 0, 0, 0, 0, 0, 0
450: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
451: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
452: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
453: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
454: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
455: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
456: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
457: C * , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
458: C * /
459: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 1, 40)/
460: C * 0, 0, 3, 0, 7372, 2, -7372, 7372, 2, -7372, -7372, 2
461: C * , 7372, -7372, 2, 7372, 7372, 2, 0, 7372, 2, 0, 0, 3
462: C * , 0, 8192, 2, -3134, 7568, 2, -5792, 5792, 2, -7568, 3134, 2
463: C * , -8192, 0, 2, -7568, -3134, 2, -5792, -5792, 2, -3134, -7568, 2
464: C * , 0, -8192, 2, 3134, -7568, 2, 5792, -5792, 2, 7568, -3134, 2
465: C * , 8192, 0, 2, 7568, 3134, 2, 5792, 5792, 2, 3134, 7568, 2
466: C * , 0, 8192, 2, 0, 0, 3, 0, 8192, 2, -8192, -4096, 2
467: C * , 8192, -4096, 2, 0, 8192, 2, 0, 8192, 3, 0, -8192, 2
468: C * , -8192, 0, 3, 8192, 0, 2, 6963, 6963, 3, -6963, -6963, 2
469: C * , -6963, 6963, 3, 6963, -6963, 2, 0, 0, 3, 0, 8192, 2
470: C * /
471: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 41, 80)/
472: C * -8192, 0, 2, 0, -8192, 2, 8192, 0, 2, 0, 8192, 2
473: C * , 0, -8192, 3, 0, 8192, 2, -8192, 0, 2, 8192, 0, 2
474: C * , 0, 8192, 2, -6963, -6963, 3, 6963, 6963, 2, -6963, 6963, 2
475: C * , 6963, -6963, 2, -6963, 6963, 3, 6963, 6963, 2, -6963, -6963, 2
476: C * , 6963, -6963, 2, -2785, 0, 3, 2785, 0, 2, 0, 0, 3
477: C * , -8192, 8192, 2, 0, -8192, 3, 0, 0, 2, 8192, 8192, 2
478: C * , 0, 0, 3, 8192, 8192, 2, 4096, -4096, 3, 4096, 4096, 2
479: C * , -4096, 4096, 2, -8192, 8192, 2, -4096, 4096, 3, -4096, -4096, 2
480: C * , -8192, -8192, 2, -4096, -4096, 3, 4096, -4096, 2, 8192, -8192, 2
481: C * , 0, -6963, 3, 0, 6963, 2, -6963, -6963, 3, 6963, 6963, 2
482: C * /
483: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 81, 120)/
484: C * -6963, 0, 3, 6963, 0, 2, -6963, 6963, 3, 6963, -6963, 2
485: C * , 0, 0, 3, -6963, 6963, 2, 6963, 6963, 2, -6963, -6963, 2
486: C * , 6963, -6963, 2, 0, 0, 2, 0, -8192, 3, 0, 8192, 2
487: C * , 0, 0, 3, 0, 8192, 2, -7372, -4096, 2, 7372, -4096, 2
488: C * , 0, 8192, 2, 7372, 4096, 3, -7372, 4096, 2, 0, -8192, 2
489: C * , 7372, 4096, 2, -8192, 0, 3, 8192, 0, 2, 5455, 0, 3
490: C * , 5455, 16384, 2, -16384, 0, 3, 0, 0, 3, 4505, 10923, 2
491: C * , 9011, 0, 2, 0, 8192, 3, 10923, 8192, 2, 0, 4915, 3
492: C * , 10923, 4915, 2, 0, 1638, 3, 10923, 1638, 2, 0, 5455, 3
493: C * , 10923, 5455, 2, 7372, 8732, 2, 10923, 5455, 3, 7372, 2179, 2
494: C * /
495: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 121, 160)/
496: C * 0, 0, 3, 1638, 6553, 3, 10928, 6553, 2, 1638, 3276, 3
497: C * , 10923, 3276, 2, 4915, 0, 3, 7646, 9830, 2, 819, 1638, 3
498: C * , 10977, 1638, 2, 5898, 1638, 3, 5898, 11796, 2, 819, 6717, 3
499: C * , 10977, 6717, 2, 0, -4096, 3, 14745, -4096, 2, 0, 0, 3
500: C * , 0, 18022, 3, 14745, 18022, 2, 4915, -901, 3, 5347, -1418, 2
501: C * , 5918, -1732, 2, 6553, -1802, 2, 7194, -1605, 2, 7723, -1162, 2
502: C * , 8065, -535, 2, 8172, 186, 2, 8028, 901, 2, 5079, 13516, 2
503: C * , 4935, 14230, 2, 5041, 14953, 2, 5384, 15580, 2, 5912, 16023, 2
504: C * , 6553, 16220, 2, 7188, 16150, 2, 7759, 15836, 2, 8192, 15319, 2
505: C * , 0, 9830, 3, 7372, 9830, 2, 8940, 9518, 2, 10269, 8630, 2
506: C * /
507: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 161, 200)/
508: C * 11157, 7301, 2, 11468, 5734, 2, 11157, 4166, 2, 10269, 2838, 2
509: C * , 8940, 1950, 2, 7372, 1638, 2, 0, 1638, 2, 0, 10928, 3
510: C * , 4505, 0, 2, 9011, 10928, 2, 0, 5455, 3, 615, 6458, 2
511: C * , 1578, 7134, 2, 2731, 7372, 2, 3881, 7135, 2, 4844, 6461, 2
512: C * , 5460, 5460, 2, 6078, 4461, 2, 7041, 3787, 2, 8192, 3550, 2
513: C * , 9342, 3787, 2, 10305, 4461, 2, 10923, 5460, 2, 0, 7094, 3
514: C * , 615, 8096, 2, 1578, 8772, 2, 2731, 9011, 2, 3881, 8773, 2
515: C * , 4844, 8099, 2, 5460, 7099, 2, 6078, 6099, 2, 7041, 5426, 2
516: C * , 8192, 5188, 2, 9342, 5426, 2, 10305, 6099, 2, 10923, 7099, 2
517: C * , 0, 3817, 3, 615, 4820, 2, 1578, 5496, 2, 2731, 5734, 2
518: C * /
519: C DATA (JT(1,I),JT(2,I),JT(3,I),I= 201, 240)/
520: C * 3881, 5496, 2, 4844, 4822, 2, 5460, 3822, 2, 6078, 2822, 2
```

src/cgview/jsymbol.f

```
521: *, 7041, 2149,2, 8192, 1912,2, 9342, 2149,2, 10305, 2822,2
522: *, 10923, 3822,2, 3276, 16384,3, 4530, 16134,2, 5593, 15423,2
523: *, 6302, 14363,2, 6553, 13107,2, 6553, 9011,2, 7372, 8192,2
524: *, 8192, 8192,2, 7372, 8192,3, 6553, 7372,2, 6553, 3276,2
525: *, 6304, 2023,2, 5593, 960,2, 4530, 249,2, 3276, 0,2
526: *, 7634, 0,3, 6381, 249,2, 5318, 960,2, 4608, 2020,2
527: *, 4358, 3276,2, 4358, 7372,2, 3538, 8191,2, 2719, 8191,2
528: *, 3538, 8191,3, 4358, 9011,2, 4358, 13107,2, 4607, 14360,2
529: *, 5318, 15423,2, 6381, 16134,2, 7634, 16383,2, -737, -4096,3
530: * /
531: DATA (JT(1,I),JT(2,I),JT(3,I),I= 241, 280)/
532: *, 1330, 0,2, 4603, 10877,2, 2293, 3276,3, 2925, 1996,2
533: *, 4148, 1242,2, 5750, 1146,2, 7459, 1729,2, 8967, 2889,2
534: *, 10010, 4423,2, 11976, 10977,2, 9666, 3276,3, 8929, 819,2
535: *, 10321, 0,2, 491, 9830,3, 3915, 10321,2, 6897, 9338,2
536: *, 10321, 9830,2, 4617, 9928,3, 0, 0,2, 7372, 9830,3
537: *, 5898, 4915,2, 6324, 1966,2, 9011, 0,2, 3276, 3276,3
538: *, 8192, 13107,2, 5734, 0,3, 3851, 374,2, 2256, 1441,2
539: *, 1191, 3038,2, 819, 4915,2, 1195, 6800,2, 2261, 8393,2
540: *, 3854, 9456,2, 5734, 9830,2, 7615, 9456,2, 9209, 8390,2
541: *, 10275, 6796,2, 10649, 4915,2, 10275, 3034,2, 9209, 1439,2
542: * /
543: DATA (JT(1,I),JT(2,I),JT(3,I),I= 281, 320)/
544: *, 7615, 374,2, 5734, 0,2, 2457, 8191,3, 9011, 8191,2
545: *, 9702, 11326,2, 9789, 13984,2, 9258, 15760,2, 8191, 16383,2
546: *, 6752, 15761,2, 5158, 13989,2, 3651, 11335,2, 2457, 8191,2
547: *, 2457, 8191,3, 1767, 5063,2, 1679, 2402,2, 2209, 624,2
548: *, 3276, 0,2, 4717, 623,2, 6313, 2399,2, 7821, 5057,2
549: *, 9011, 8191,2, 491, 9830,3, -491, 6553,2, -265, 5012,2
550: *, 780, 3850,2, 2457, 3276,2, 4400, 3382,2, 6319, 4123,2
551: *, 7886, 5372,2, 8833, 6917,2, 8999, 8492,2, 8355, 9830,2
552: *, 5898, 14745,3, 1474, 0,2, 0, 0,3, 9994, 11468,2
553: *, 3440, 11468,3, 4641, 11293,2, 5555, 10796,2, 6042, 10954,2
554: * /
555: DATA (JT(1,I),JT(2,I),JT(3,I),I= 321, 360)/
556: *, 6029, 9175,2, 3964, 2293,2, 3951, 1416,2, 4438, 672,2
557: *, 5352, 174,2, 6553, 0,2, 4153, 10977,3, 2842, 9258,2
558: *, 1778, 7106,2, 1104, 4810,2, 912, 2679,2, 1227, 1000,2
559: *, 2007, 0,2, 3227, -170,2, 4668, 602,2, 6110, 2201,2
560: *, 7335, 4383,2, 8156, 6817,2, 8448, 9132,2, 8157, 10977,2
561: *, 6779, 9132,2, 5682, 6817,2, 5043, 4383,2, 4958, 2201,2
562: *, 5441, 602,2, 6418, -170,2, 7741, 0,2, 5121, 1000,2
563: *, 10443, 2679,2, 11529, 4810,2, 12234, 7106,2, 12461, 9258,2
564: *, 12181, 10977,2, 983, 0,3, 7110, 10977,2, 4259, 16384,3
565: *, 5718, 16138,2, 6854, 15428,2, 7493, 14363,2, 7536, 13107,2
566: * /
567: DATA (JT(1,I),JT(2,I),JT(3,I),I= 361, 400)/
568: *, 5570, 3276,2, 5613, 2020,2, 6252, 955,2, 7388, 245,2
569: *, 8847, 0,2, 12124, 13107,3, 9666, 4915,2, 8741, 3050,2
570: *, 7220, 1463,2, 5331, 392,2, 3355, -5,2, 1586, 330,2
571: *, 289, 1350,2, -341, 2900,2, -212, 4751,2, 684, 6668,2
572: *, 2217, 8310,2, 4146, 9421,2, 6168, 9826,2, 7969, 9463,2
573: *, 9266, 8388,2, 9830, 0,2, 10928, 0,2, 5816, 9830,3
574: *, 7114, 9456,2, 7991, 8390,2, 8311, 6796,2, 8028, 4915,2
575: *, 7183, 3034,2, 5905, 1439,2, 4390, 374,2, 2867, 0,2
576: *, 1568, 374,2, 692, 1439,2, 371, 3034,2, 655, 4915,2
577: *, 1500, 6796,2, 2777, 8390,2, 4293, 9456,2, 5816, 9830,2
578: * /
579: DATA (JT(1,I),JT(2,I),JT(3,I),I= 401, 440)/
580: *, 4907, 10092,2, 4293, 10838,2, 4069, 11954,2, 4268, 13271,2
581: *, 4859, 14587,2, 5753, 15703,2, 6814, 16449,2, 7880, 16711,2
582: *, 10092, 16711,2, 12009, 15728,2, 6219, 768,3, 4879, 199,2
583: *, 3440, 0,2, 2138, 193,2, 1144, 744,2, 605, 1573,2
584: *, 600, 2557,2, 1130, 3550,2, 2117, 4407,2, 3415, 5000,2
585: *, 4833, 5242,2, 6635, 5242,2, 4833, 5242,3, 3730, 5474,2
```

```
586: *, 2905, 5995,2, 2468, 6737,2, 2477, 7602,2, 2950, 8500,2
587: *, 3830, 9281,2, 4991, 9834,2, 6270, 10081,2, 7488, 9988,2
588: *, 8473, 9568,2, 491, 1638,3, 3194, 10649,2, 2473, 10977,2
589: *, 2295, 7651,3, 2911, 8920,2, 3942, 10000,2, 5228, 10723,2
590: * /
591: DATA (JT(1,I),JT(2,I),JT(3,I),I= 441, 480)/
592: *, 6569, 10977,2, 7759, 10723,2, 8611, 10000,2, 8994, 8920,2
593: *, 8848, 7651,2, 7536, 3276,2, 5734, 0,2, 5242, -1638,2
594: *, 5324, -4096,2, 0, 0,3, 0, 0,3, 9830, 13107,3
595: *, 9830, 16384,2, 0, 16384,2, 4915, 8192,2, 0, 0,2
596: *, 9830, 0,2, 9830, 3276,2, 0, 6553,3, 11468, 6553,2
597: *, 6144, 9011,3, 5324, 9011,2, 5324, 10649,2, 6144, 10649,2
598: *, 6144, 9011,2, 6144, 2457,3, 5324, 2457,2, 5324, 4096,2
599: *, 6144, 4096,2, 6144, 2457,2, 10928, 12020,3, 0, 8196,2
600: *, 10928, 3278,2, 0, 1092,3, 10928, 1092,2, 0, 12015,3
601: *, 10923, 8192,2, 0, 3276,2, 10923, 1092,3, 0, 1092,2
602: * /
603: DATA (JT(1,I),JT(2,I),JT(3,I),I= 481, 520)/
604: *, 0, 0,3, 4915, 16384,2, 9830, 0,2, 0, 0,2
605: *, 8192, 16384,3, 3276, 16384,2, 3276, 0,2, 8192, 0,2
606: *, 1638, 16384,3, 6553, 16384,2, 6553, 0,2, 1638, 0,2
607: *, 0, 16384,3, 10923, 0,2, 10048, 10377,3, 7549, 9961,2
608: *, 5911, 9961,2, 4350, 10739,2, 2712, 10739,2, 1038, 10381,2
609: *, 4350, 10739,3, 3510, 2334,2, 3529, 1460,2, 3920, 700,2
610: *, 4719, 138,2, 5734, 0,2, 9011, 0,2, 4915, 3276,3
611: *, 8192, 0,2, 11468, 18022,2, 16384, 18022,2, 4915, 16384,3
612: *, 4915, 0,2, 3276, 12288,3, 6553, 12288,2, 3276, 4096,3
613: *, 6553, 4096,2, 4915, 16384,3, 4915, 0,2, 3276, 13380,3
614: * /
615: DATA (JT(1,I),JT(2,I),JT(3,I),I= 521, 560)/
616: *, 6553, 13380,2, 3276, 8192,3, 6553, 8192,2, 3276, 2731,3
617: *, 6553, 2731,2, 10921, 5465,3, -1, 5465,2, 1091, 3827,2
618: *, -1, 5465,3, 1091, 7104,2, 1359, 2457,3, 9551, 10649,2
619: *, 1359, 10649,3, 9551, 2457,2, 4915, 0,3, 4915, 16384,2
620: *, 3276, 14745,2, 4915, 16384,3, 6553, 14745,2, 4915, 16384,3
621: *, 4915, 0,2, 3276, 1638,2, 4915, 0,3, 6553, 1638,2
622: *, 0, 0,3, 9338, 1474,3, 7004, 589,2, 3891, 0,2
623: *, 2457, 60,2, 1184, 689,2, 310, 1768,2, 0, 3096,2
624: *, 310, 4424,2, 1184, 5503,2, 2457, 6132,2, 3891, 6193,2
625: *, 7004, 5603,2, 9338, 5013,2, 1556, 9584,3, 2959, 10613,2
626: * /
627: DATA (JT(1,I),JT(2,I),JT(3,I),I= 561, 600)/
628: *, 4669, 11059,2, 6645, 10781,2, 8308, 9731,2, 9338, 8110,2
629: *, 9338, 737,2, 10381, 0,2, 11206, 0,2, 0, 0,3
630: *, 0, 16384,2, 0, 2129,3, 1638, 819,2, 4369, 0,2
631: *, 6459, 415,2, 8231, 1599,2, 9414, 3371,2, 9830, 5460,2
632: *, 9414, 7551,2, 8231, 9323,2, 6459, 10507,2, 4369, 10923,2
633: *, 1638, 10104,2, 0, 8793,2, 9830, 1638,3, 8192, 491,2
634: *, 5460, 0,2, 3371, 415,2, 1600, 1598,2, 416, 3369,2
635: *, 0, 5460,2, 414, 7549,2, 1598, 9322,2, 3370, 10506,2
636: *, 5460, 10923,2, 8192, 10431,2, 9830, 9284,2, 9830, 1638,3
637: *, 8192, 491,2, 5460, 0,2, 3371, 415,2, 1600, 1598,2
638: * /
639: DATA (JT(1,I),JT(2,I),JT(3,I),I= 601, 640)/
640: *, 416, 3369,2, 0, 5460,2, 414, 7549,2, 1598, 9322,2
641: *, 3370, 10506,2, 5460, 10923,2, 8192, 10431,2, 9830, 9284,2
642: *, 9830, 16384,3, 9830, 0,2, 9830, 1638,3, 8192, 491,2
643: *, 5460, 0,2, 3371, 415,2, 1600, 1598,2, 416, 3369,2
644: *, 0, 5460,2, 414, 7549,2, 1598, 9322,2, 3370, 10506,2
645: *, 5460, 10923,2, 8192, 10267,2, 9830, 8192,2, 9830, 6553,2
646: *, 163, 6553,2, 3276, 0,3, 3276, 13107,2, 4096, 15155,2
647: *, 5734, 16384,2, 7372, 16384,2, 9011, 15155,2, 9830, 14745,2
648: *, 0, 9830,3, 8192, 9830,2, 1638, -3276,3, 3108, -4432,2
649: *, 4915, -4915,2, 6840, -4619,2, 8476, -3561,2, 9534, -1925,2
650: * /
```

src/cgview/jsymbol.f

```
651: DATA (JT(1,I),JT(2,I),JT(3,I),I= 641, 680)/
652: * 9830, 0,2, 9830, 10923,2, 9830, 2129,3, 8192, 983,2
653: *, 5460, 491,2, 3446, 794,2, 1701, 1845,2, 491, 3483,2
654: *, 0, 5460,2, 345, 7590,2, 1522, 9399,2, 3330, 10576,2
655: *, 5460, 10923,2, 8192, 10431,2, 9830, 9284,2, 0, 0,3
656: *, 0, 16384,2, 0, 6008,3, 512, 7734,2, 1608, 9162,2
657: *, 3144, 10103,2, 4915, 10431,2, 6685, 10103,2, 8221, 9162,2
658: *, 9318, 7734,2, 9830, 6008,2, 9830, 0,2, 5734, 0,3
659: *, 5734, 10813,2, 3276, 10813,2, 6144, 13926,3, 5324, 13926,2
660: *, 5324, 14745,2, 6144, 14745,2, 6144, 13926,2, 6144, 14745,3
661: *, 5324, 13926,2, 5324, 14745,3, 6144, 13926,2, 2785, 0,3
662: * /
663: DATA (JT(1,I),JT(2,I),JT(3,I),I= 681, 720)/
664: * 8683, 0,2, 5455, -2184,3, 5455, 13107,2, 10923, 8738,3
665: *, 9970, 9798,2, 8525, 10571,2, 6767, 10962,2, 4915, 10923,2
666: *, 3294, 10496,2, 1372, 9741,2, 1092, 8738,2, 0, 6553,2
667: *, 0, 4369,2, 1092, 2184,2, 1972, 1182,2, 3294, 426,2
668: *, 4915, 0,2, 6767, -39,2, 8525, 351,2, 9870, 1124,2
669: *, 10923, 2184,2, 4915, 0,3, 3276, 0,2, 3276, 1638,2
670: *, 4915, 1638,2, 4915, 0,2, 4915, 1638,3, 3276, 0,2
671: *, 3276, 1638,3, 4915, 0,2, 10923, 10923,3, 0, 5461,2
672: *, 10923, 0,2, 8847, 16384,3, 6821, 14768,2, 5417, 11846,2
673: *, 4915, 8192,2, 5415, 4544,2, 6819, 1617,2, 8847, 0,2
674: * /
675: DATA (JT(1,I),JT(2,I),JT(3,I),I= 721, 760)/
676: * 5455, 11468,3, 5455, 3276,2, 1359, 7372,3, 9551, 7372,2
677: *, 0, 16384,3, 0, 0,2, 10026, -159,3, 1568, 10576,2
678: *, 1142, 11503,2, 1148, 12504,2, 1586, 13426,2, 2388, 14128,2
679: *, 3410, 14498,2, 4513, 14499,2, 5535, 14132,2, 6327, 13450,2
680: *, 6772, 12554,2, 6804, 11574,2, 6420, 10655,2, 5676, 9933,2
681: *, 2052, 7677,2, 858, 6569,2, 233, 5139,2, 270, 3605,2
682: *, 966, 2201,2, 2212, 1140,2, 3821, 584,2, 5547, 617,2
683: *, 7127, 1235,2, 11958, 4242,2, 0, 0,3, 0, -1228,2
684: *, 219, -2328,2, 841, -3257,2, 1770, -3878,2, 2867, -4096,2
685: *, 3964, -3877,2, 4894, -3256,2, 5516, -2326,2, 5734, -1228,2
686: * /
687: DATA (JT(1,I),JT(2,I),JT(3,I),I= 761, 800)/
688: * 5734, 10813,2, 3276, 10813,2, 6144, 13926,3, 5324, 13926,2
689: *, 5324, 14745,2, 6144, 14745,2, 6144, 13926,2, 6144, 14745,3
690: *, 5324, 13926,2, 5324, 14745,3, 6144, 13926,2, 0, 0,3
691: *, 0, 16384,2, 0, 5734,3, 8192, 10649,2, 2457, 6963,3
692: *, 9830, 0,2, 4096, 16384,3, 4096, 819,2, 4915, 0,2
693: *, 8192, 0,2, 1228, 0,3, 1228, 9650,2, 204, 10381,2
694: *, 1228, 8124,3, 1302, 8961,2, 1828, 9699,2, 2710, 10202,2
695: *, 3788, 10381,2, 4866, 10202,2, 5749, 9699,2, 6274, 8961,2
696: *, 6348, 8124,2, 6348, 0,2, 6348, 8124,3, 6422, 8961,2
697: *, 6948, 9699,2, 7830, 10202,2, 8908, 10381,2, 9986, 10202,2
698: * /
699: DATA (JT(1,I),JT(2,I),JT(3,I),I= 801, 840)/
700: * 10869, 9699,2, 11394, 8961,2, 11468, 8124,2, 11468, 0,2
701: *, 819, 0,3, 819, 10158,2, 0, 10928,2, 819, 6832,3
702: *, 1508, 8778,2, 2968, 10239,2, 4915, 10928,2, 6734, 10784,2
703: *, 8360, 9955,2, 9545, 8567,2, 10108, 6832,2, 10108, 0,2
704: *, 9830, 4915,3, 9456, 3034,2, 8390, 1439,2, 6796, 374,2
705: *, 4915, 0,2, 3034, 374,2, 1439, 1439,2, 374, 3034,2
706: *, 0, 4915,2, 0, 6008,2, 375, 7891,2, 1440, 9484,2
707: *, 3035, 10549,2, 4915, 10923,2, 6795, 10549,2, 8389, 9484,2
708: *, 9455, 7891,2, 9830, 6008,2, 9830, 4915,2, 0, -4587,3
709: *, 0, 10923,2, 0, 2293,3, 1638, 1310,2, 4369, 655,2
710: * /
711: DATA (JT(1,I),JT(2,I),JT(3,I),I= 841, 880)/
712: * 6109, 1145,2, 7573, 2205,2, 8581, 3705,2, 9011, 5460,2
713: *, 8759, 7413,2, 7794, 9129,2, 6256, 10359,2, 4369, 10923,2
714: *, 1638, 10267,2, 0, 9284,2, 9011, 2293,3, 7372, 1474,2
715: *, 5460, 655,2, 3471, 920,2, 1735, 1927,2, 516, 3521,2
```

```
716: *, 0, 5460,2, 320, 7604,2, 1495, 9426,2, 3316, 10602,2
717: *, 5460, 10923,2, 7372, 10104,2, 9011, 9284,2, 9011, 10923,3
718: *, 9011, -3276,2, 10321, -4587,2, 10923, -4587,2, 1966, 0,3
719: *, 1966, 9830,2, 327, 10928,2, 1966, 5734,3, 3276, 8519,2
720: *, 3993, 9941,2, 5242, 10928,2, 6653, 11290,2, 8093, 11076,2
721: *, 9338, 10321,2, 4014, 16384,3, 4096, 3276,2, 4177, 16384,2
722: * /
723: DATA (JT(1,I),JT(2,I),JT(3,I),I= 881, 920)/
724: * 4014, 16384,2, 4505, 0,3, 3686, 0,2, 3686, 819,2
725: *, 4505, 819,2, 4505, 0,2, 0, 16384,3, 4915, 9830,2
726: *, 9830, 16384,2, 819, 9830,3, 9011, 9830,2, 819, 6553,3
727: *, 9011, 6553,2, 4915, 9830,3, 4915, 0,2, 5464, 12566,3
728: *, 5464, 1638,2, 0, 4370,3, 10928, 9834,2, 0, 9834,3
729: *, 10928, 4370,2, 1966, 16384,3, 3993, 14767,2, 5397, 11843,2
730: *, 5898, 8192,2, 5397, 4540,2, 3993, 1616,2, 1966, 0,2
731: *, 6553, 8192,3, 4915, 8192,2, 4915, 9830,2, 6553, 9830,2
732: *, 6553, 8192,2, 6553, 9830,3, 4915, 8192,2, 4915, 9830,3
733: *, 6553, 8192,2, 6553, 0,3, 4915, 0,2, 4915, 1638,2
734: * /
735: DATA (JT(1,I),JT(2,I),JT(3,I),I= 921, 960)/
736: * 6553, 1638,2, 6553, 0,2, 6553, 1638,3, 4915, 0,2
737: *, 4915, 1638,3, 6553, 0,2, 4915, -2457,2, 0, 7372,3
738: *, 10928, 7372,2, 10928, 3276,2, 1638, 7372,3, 9284, 7372,2
739: *, 0, 0,3, 10928, 16384,2, 0, 2894,3, 257, 1783,2
740: *, 1009, 841,2, 2139, 214,2, 3473, 0,2, 5269, 0,2
741: *, 6598, 220,2, 7725, 847,2, 8478, 1786,2, 8742, 2894,2
742: *, 8478, 4002,2, 7725, 4941,2, 6598, 5568,2, 5269, 5789,2
743: *, 3473, 5789,2, 2294, 5984,2, 1295, 6541,2, 627, 7373,2
744: *, 393, 8356,2, 627, 9338,2, 1295, 10171,2, 2294, 10727,2
745: *, 3473, 10923,2, 5269, 10923,2, 6448, 10720,2, 7446, 10158,2
746: * /
747: DATA (JT(1,I),JT(2,I),JT(3,I),I= 961,1000)/
748: * 8107, 9330,2, 8345, 8356,2, 0, 10928,3, 8192, 10928,2
749: *, 3276, 14745,3, 3276, 3276,2, 3527, 2020,2, 4237, 958,2
750: *, 5300, 249,2, 6553, 0,2, 7807, 249,2, 8870, 959,2
751: *, 9580, 2022,2, 9830, 3276,2, 9011, 10921,3, 9011, 763,2
752: *, 10813, -6,2, 9011, 4089,3, 8667, 2520,2, 7690, 1192,2
753: *, 6229, 304,2, 4505, -6,2, 2781, 305,2, 1319, 1193,2
754: *, 342, 2521,2, 0, 4089,2, 0, 10921,2, 0, 10923,3
755: *, 4914, 0,2, 9830, 10923,2, 0, 10923,3, 2730, 0,2
756: *, 5461, 9830,2, 8192, 0,2, 10922, 10923,2, 0, 0,3
757: *, 9830, 10923,2, 0, 10923,3, 9830, 0,2, 819, -4096,3
758: * /
759: DATA (JT(1,I),JT(2,I),JT(3,I),I=1001,1040)/
760: * 2457, -4096,2, 9830, 10923,2, 0, 10928,3, 5406, 1966,2
761: *, 491, 10923,3, 9338, 10923,2, 0, 0,2, 9830, 0,2
762: *, 6225, 4423,3, 5988, 3357,2, 5314, 2454,2, 4304, 1850,2
763: *, 3112, 1638,2, 1921, 1850,2, 911, 2454,2, 236, 3357,2
764: *, 0, 4423,2, 236, 5489,2, 911, 6393,2, 1921, 6996,2
765: *, 3112, 7208,2, 4304, 6996,2, 5314, 6393,2, 5988, 5489,2
766: *, 6225, 4423,2, 6462, 3357,2, 7137, 2454,2, 8147, 1850,2
767: *, 9338, 1638,2, 10530, 1850,2, 11540, 2454,2, 12214, 3357,2
768: *, 12451, 4423,2, 12214, 5489,2, 11540, 6393,2, 10530, 6996,2
769: *, 9338, 7208,2, 8147, 6996,2, 7137, 6393,2, 6462, 5489,2
770: * /
771: DATA (JT(1,I),JT(2,I),JT(3,I),I=1041,1080)/
772: * 6225, 4423,2, 3276, 0,3, 1638, 0,2, 1638, 1638,2
773: *, 3276, 1638,2, 3276, 0,2, 3276, 1638,3, 1638, 0,2
774: *, 1638, 1638,3, 3276, 0,2, 1638, -2457,2, 819, 0,3
775: *, 12287, 14745,2, 819, 11468,3, 1007, 12412,2, 1540, 13208,2
776: *, 2337, 13739,2, 3276, 13926,2, 4216, 13739,2, 5013, 13207,2
777: *, 5546, 12411,2, 5734, 11468,2, 5547, 10528,2, 5014, 9731,2
778: *, 4217, 9198,2, 3276, 9011,2, 2336, 9198,2, 1539, 9731,2
779: *, 1006, 10528,2, 819, 11468,2, 7372, 3276,3, 7561, 4220,2
780: *, 8094, 5016,2, 8891, 5547,2, 9830, 5734,2, 10770, 5547,2
```

src/cgview/jsymbol.f

```
781:      *, 11568, 5014,2, 12100, 4217,2, 12287, 3276,2, 12100, 2336,2
782:      * /
783:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1081,1120)/
784:      * 11568, 1539,2, 10770, 1006,2, 9830, 819,2, 8889, 1006,2
785:      *, 8091, 1540,2, 7558, 2339,2, 7372, 3276,2, 0, 5455,3
786:      *, 16384, 5455,2, 0, 10923,3, 10923, 5461,2, 0, 0,2
787:      *, 409, 11468,3, 824, 13350,2, 1941, 14921,2, 3583, 15931,2
788:      *, 5488, 16220,2, 7282, 15776,2, 8778, 14692,2, 9759, 13126,2
789:      *, 10081, 11307,2, 9697, 9500,2, 8664, 7968,2, 7132, 6936,2
790:      *, 5324, 6553,2, 5324, 3276,2, 5734, 0,3, 4915, 0,2
791:      *, 4915, 819,2, 5734, 819,2, 5734, 0,2, 6796, 623,3
792:      *, 4915, 0,2, 3034, 623,2, 1439, 2399,2, 374, 5057,2
793:      *, 0, 8191,2, 374, 11326,2, 1439, 13984,2, 3034, 15760,2
794:      * /
795:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1121,1160)/
796:      * 4915, 16384,2, 6796, 15760,2, 8390, 13984,2, 9456, 11326,2
797:      *, 9830, 8191,2, 9456, 5057,2, 8390, 2399,2, 6796, 623,2
798:      *, 4915, 0,2, 3034, 623,2, 1439, 2399,2, 374, 5057,2
799:      *, 1638, 14745,3, 4096, 16384,2, 4096, 0,2, 1638, 0,3
800:      *, 6553, 0,2, 0, 11714,3, 405, 13435,2, 1502, 14880,2
801:      *, 3121, 15827,2, 5013, 16130,2, 6837, 15761,2, 8378, 14793,2
802:      *, 9411, 13366,2, 9785, 11689,2, 9446, 10005,2, 8444, 8560,2
803:      *, 0, 0,2, 10321, 0,2, 1033, 13862,3, 2923, 15627,2
804:      *, 4310, 16131,2, 6200, 16383,2, 8216, 15753,2, 9477, 14368,2
805:      *, 9603, 12602,2, 9351, 10712,2, 7964, 8821,2, 4688, 8821,2
806:      * /
807:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1161,1200)/
808:      * 7964, 8821,3, 9477, 7939,2, 10737, 6049,2, 10737, 3654,2
809:      *, 9477, 1260,2, 7586, 252,2, 5696, 0,2, 3175, 252,2
810:      *, 655, 1008,2, 6717, 16384,3, 0, 4915,2, 9830, 4915,2
811:      *, 7372, 9011,3, 7372, 0,2, 0, 2981,3, 819, 2097,2
812:      *, 2141, 713,2, 3840, -54,2, 5681, -102,2, 7412, 577,2
813:      *, 8794, 1890,2, 9639, 3657,2, 9830, 5636,2, 9262, 718,2
814:      *, 7977, 9388,2, 6185, 10371,2, 4179, 10507,2, 2287, 5773,2
815:      *, 819, 8290,2, 819, 16252,2, 9011, 16252,2, 0, 4915,3
816:      *, 374, 3034,2, 1439, 1439,2, 3034, 374,2, 4915, 0,2
817:      *, 6796, 374,2, 8390, 1439,2, 9456, 3034,2, 9830, 4915,2
818:      * /
819:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1201,1240)/
820:      * 9456, 6796,2, 8390, 8390,2, 6796, 9456,2, 4915, 9830,2
821:      *, 3034, 9456,2, 1439, 8390,2, 374, 6796,2, 0, 4915,2
822:      *, 0, 7045,2, 327, 10321,2, 1966, 13271,2, 1259, 15073,2
823:      *, 6553, 16056,2, 8683, 16384,2, 819, 16384,3, 9830, 16384,2
824:      *, 4423, 0,2, 4915, 0,3, 3034, 346,2, 1439, 1334,2
825:      *, 374, 2811,2, 0, 4554,2, 374, 6297,2, 1439, 7775,2
826:      *, 3034, 8762,2, 4915, 9109,2, 3409, 9387,2, 2133, 10177,2
827:      *, 1281, 11361,2, 983, 12753,2, 1283, 14149,2, 2135, 15330,2
828:      *, 3411, 16120,2, 4915, 16397,2, 6419, 16119,2, 7695, 15329,2
829:      *, 8548, 14147,2, 8847, 12753,2, 8548, 11358,2, 7695, 10176,2
830:      * /
831:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1241,1280)/
832:      * 6419, 9386,2, 4915, 9109,2, 6796, 8762,2, 8390, 7775,2
833:      *, 9456, 6297,2, 9830, 4554,2, 9456, 2811,2, 8390, 1334,2
834:      *, 6796, 346,2, 4915, 0,2, 9830, 11468,3, 9456, 13349,2
835:      *, 8390, 14944,2, 6796, 16009,2, 4915, 16383,2, 3034, 16009,2
836:      *, 1439, 14944,2, 374, 13349,2, 0, 11468,2, 374, 9587,2
837:      *, 1439, 7993,2, 3034, 6927,2, 4915, 6553,2, 6796, 6927,2
838:      *, 8390, 7993,2, 9456, 9587,2, 9830, 11468,2, 9830, 9338,2
839:      *, 9502, 6062,2, 7864, 3112,2, 5570, 1310,2, 3276, 327,2
840:      *, 1146, 0,2, 6553, 8192,3, 4915, 8192,2, 4915, 9830,2
841:      *, 6553, 9830,2, 6553, 8192,2, 6553, 9830,3, 4915, 8192,2
842:      * /
843:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1281,1320)/
844:      * 4915, 9830,3, 6553, 8192,2, 6553, 0,3, 4915, 0,2
845:      *, 4915, 1638,2, 6553, 1638,2, 6553, 0,2, 6553, 1638,3
```

```
846:      *, 4915, 0,2, 4915, 1638,3, 6553, 0,2, 2732, 0,3
847:      *, 2732, 11468,2, 8196, 11468,3, 8196, 0,2, 0, 8028,3
848:      *, 10928, 9175,2, 0, 2293,3, 10928, 3440,2, 0, 8028,3
849:      *, 1064, 10406,2, 3019, 12048,2, 5464, 12615,2, 7545, 12162,2
850:      *, 9310, 10917,2, 10496, 9065,2, 10928, 6881,2, 10928, 2293,2
851:      *, 10989, 1636,2, 10820, 999,2, 10442, 470,2, 9907, 121,2
852:      *, 9288, 0,2, 8669, 121,2, 8135, 470,2, 7757, 999,2
853:      *, 7587, 1636,2, 7649, 2293,2, 7649, 6881,2, 7649, 5734,3
854:      * /
855:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1321,1360)/
856:      * 6757, 7056,2, 5453, 7925,2, 3934, 8211,2, 2424, 7872,2
857:      *, 1149, 6958,2, 298, 5606,2, 0, 4014,2, 297, 2425,2
858:      *, 1146, 1071,2, 2421, 156,2, 3932, -183,2, 5452, 102,2
859:      *, 6756, 971,2, 7649, 2293,2, 5865, 16384,3, 5046, 16384,2
860:      *, 5455, 12288,2, 5865, 16384,2, 1638, 8192,3, 10928, 8192,2
861:      *, 1638, 4096,3, 10928, 4096,2, 5865, 16384,3, 5046, 16384,2
862:      *, 5455, 12288,2, 5865, 16384,2, 7913, 16384,3, 7094, 16384,2
863:      *, 7503, 12288,2, 7913, 16384,2, 0, 0,3, 5460, 16384,2
864:      *, 10923, 0,2, 1912, 5734,3, 9011, 5734,2, 0, 0,3
865:      *, 0, 16384,2, 6225, 16384,2, 7699, 16090,2, 8948, 15256,2
866:      * /
867:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1361,1400)/
868:      * 9783, 14007,2, 10076, 12533,2, 9783, 11060,2, 8948, 9811,2
869:      *, 7699, 8976,2, 6225, 8683,2, 0, 8683,2, 6225, 8683,3
870:      *, 7887, 8353,2, 9296, 7411,2, 10237, 6003,2, 10567, 4341,2
871:      *, 10237, 2680,2, 9296, 1271,2, 7887, 330,2, 6225, 0,2
872:      *, 0, 0,2, 10923, 13107,3, 9970, 14696,2, 8525, 15856,2
873:      *, 6767, 16443,2, 4915, 16384,2, 3294, 15744,2, 1972, 14610,2
874:      *, 1092, 13107,2, 0, 9830,2, 0, 6553,2, 1092, 3276,2
875:      *, 1972, 1773,2, 3294, 639,2, 4915, 0,2, 6767, -59,2
876:      *, 8525, 527,2, 9970, 1687,2, 10923, 3276,2, 0, 0,3
877:      *, 0, 16384,2, 4915, 16384,2, 7214, 15926,2, 9163, 14624,2
878:      * /
879:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1401,1440)/
880:      * 10465, 12675,2, 10923, 10376,2, 10923, 5996,2, 10463, 3703,2
881:      *, 9163, 1759,2, 7214, 457,2, 4915, 0,2, 0, 0,2
882:      *, 491, 0,3, 491, 16384,2, 10923, 16384,2, 491, 8683,3
883:      *, 9830, 8683,2, 491, 0,3, 10923, 0,2, 0, 0,3
884:      *, 0, 16384,2, 10923, 16384,2, 0, 8683,3, 9830, 8683,2
885:      *, 10923, 13107,3, 9970, 14696,2, 8525, 15856,2, 6767, 16443,2
886:      *, 4915, 16384,2, 3294, 15744,2, 1972, 14610,2, 1092, 13107,2
887:      *, 0, 9830,2, 0, 6553,2, 1092, 3276,2, 1972, 1773,2
888:      *, 3294, 639,2, 4915, 0,2, 6767, -59,2, 8525, 527,2
889:      *, 9970, 1687,2, 10923, 3276,2, 10923, 7372,2, 6553, 7372,2
890:      * /
891:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1441,1480)/
892:      * 0, 0,3, 0, 16384,2, 0, 8683,3, 10928, 8683,2
893:      *, 10928, 16384,3, 10928, 0,2, 4096, 0,3, 7372, 0,2
894:      *, 5734, 0,3, 5734, 16384,2, 4096, 16384,3, 7372, 16384,2
895:      *, 0, 4096,3, 311, 2528,2, 1199, 1199,2, 2528, 311,2
896:      *, 4096, 0,2, 5663, 311,2, 6992, 1199,2, 7880, 2528,2
897:      *, 8192, 4096,2, 8192, 16384,2, 6553, 16384,3, 9830, 16384,2
898:      *, 0, 0,3, 0, 16384,2, 10923, 16384,3, 0, 7372,2
899:      *, 2785, 9175,3, 10923, 0,2, 0, 16384,3, 0, 0,2
900:      *, 10928, 0,2, 10928, 2457,2, 0, 0,3, 0, 16384,2
901:      *, 5460, 7372,2, 10923, 16384,2, 10923, 0,2, 327, 0,3
902:      * /
903:      DATA (JT(1,I),JT(2,I),JT(3,I),I=1481,1520)/
904:      * 327, 16384,2, 10923, 0,2, 10923, 16384,2, 0, 5460,3
905:      *, 0, 10923,2, 417, 13017,2, 1601, 14786,2, 3372, 15968,2
906:      *, 5460, 16384,2, 7551, 15968,2, 9323, 14784,2, 10507, 13012,2
907:      *, 10923, 10923,2, 10923, 5460,2, 10507, 3371,2, 9323, 1599,2
908:      *, 7551, 415,2, 5460, 0,2, 3371, 415,2, 1600, 1598,2
909:      *, 416, 3369,2, 0, 5460,2, 819, 0,3, 819, 16384,2
910:      *, 7045, 16384,2, 8518, 16090,2, 9767, 15256,2, 10602, 14007,2
```


src/cgview/jsymbol.f

```
1041: *, 4096, 0,2, 7372, 0,2, 3622, 10977,3, 3604, 8192,2
1042: *, 2793, 5734,2, 1081, 3276,2, 941, 2025,2, 1371, 961,2
1043: *, 2306, 249,2, 3604, 0,2, 5066, 249,2, 6469, 959,2
1044: *, 7602, 2022,2, 8290, 3276,2, 9371, 6553,2, 9551, 8192,2
1045: *, 9914, 10928,2, 3276, -3276,3, 8192, 13107,2, 5734, 0,3
1046: * /
1047: DATA (JT(1,I),JT(2,I),JT(3,I),I=1961,2000)/
1048: *, 3851, 374,2, 2256, 1441,2, 1191, 3038,2, 819, 4915,2
1049: *, 1195, 6800,2, 2261, 8393,2, 3854, 9456,2, 5734, 9830,2
1050: *, 7615, 9456,2, 9209, 8390,2, 10275, 6796,2, 10649, 4915,2
1051: *, 10275, 3034,2, 9209, 1439,2, 7615, 374,2, 5734, 0,2
1052: *, 0, 0,3, 9994, 11468,2, 3440, 11468,3, 4641, 11293,2
1053: *, 5545, 10796,2, 6042, 10054,2, 6029, 9175,2, 3964, 2293,2
1054: *, 3991, 1416,2, 4438, 672,2, 5352, 174,2, 6553, 0,2
1055: *, 491, 9830,3, -491, 6553,2, -265, 5012,2, 780, 3850,2
1056: *, 2457, 3276,2, 4400, 3392,2, 6319, 4123,2, 7886, 5372,2
1057: *, 8833, 6917,2, 8999, 8492,2, 8355, 9830,2, 5898, 14745,3
1058: * /
1059: DATA (JT(1,I),JT(2,I),JT(3,I),I=2001,2040)/
1060: *, 1474, 0,2, 4153, 10977,3, 2842, 9258,2, 1778, 7106,2
1061: *, 1104, 4810,2, 912, 2679,2, 1227, 1000,2, 2007, 0,2
1062: *, 3227, -170,2, 4668, 602,2, 6110, 2201,2, 7335, 4383,2
1063: *, 8156, 6817,2, 8448, 9132,2, 8167, 10977,2, 6779, 9132,2
1064: *, 5682, 6817,2, 5043, 4383,2, 4958, 2201,2, 5441, 602,2
1065: *, 6418, -170,2, 7741, 0,2, 9121, 1000,2, 10443, 2679,2
1066: *, 11529, 4810,2, 12234, 7106,2, 12461, 9258,2, 12181, 10977,2
1067: *, 0, 0,3, 8737, 16384,2, 10923, 0,2, 3932, 7372,3
1068: *, 9940, 7372,2, 0, 0,3, 3276, 16384,2, 9502, 16384,2
1069: *, 10917, 16090,2, 11999, 15256,2, 12584, 14007,2, 12582, 12533,2
1070: * /
1071: DATA (JT(1,I),JT(2,I),JT(3,I),I=2041,2080)/
1072: *, 11995, 11060,2, 10910, 9811,2, 9494, 8976,2, 7962, 8683,2
1073: *, 1736, 8683,2, 7962, 8683,3, 9558, 8353,2, 10748, 7411,2
1074: *, 11437, 6003,2, 11436, 4341,2, 10773, 2680,2, 9650, 1271,2
1075: *, 7953, 330,2, 6225, 0,2, 0, 0,2, 0, 0,3
1076: *, 3276, 16384,2, 13107, 16384,2, 12779, 14745,2, 0, 0,3
1077: *, 6553, 16384,2, 9830, 0,2, 0, 0,2, 0, 0,3
1078: *, 3276, 16384,2, 13107, 16384,2, 1736, 8683,3, 9928, 8683,2
1079: *, 0, 0,3, 9830, 0,2, 3822, 16384,2, 13107, 16384,2
1080: *, 0, 0,2, 9830, 0,2, 0, 0,3, 3276, 16384,2
1081: *, 1736, 8683,3, 11567, 8683,2, 13107, 16384,3, 9830, 0,2
1082: * /
1083: DATA (JT(1,I),JT(2,I),JT(3,I),I=2081,2120)/
1084: *, 4915, 0,3, 3158, 623,2, 1919, 2399,2, 1385, 5057,2
1085: *, 1638, 8192,2, 2639, 11326,2, 4236, 13984,2, 6186, 15760,2
1086: *, 8192, 16384,2, 9948, 15760,2, 11187, 13984,2, 11721, 11326,2
1087: *, 11468, 8192,2, 10467, 5057,2, 8870, 2399,2, 6920, 623,2
1088: *, 4915, 0,2, 3276, 6553,3, 3932, 9830,2, 3604, 8192,2
1089: *, 9502, 8192,2, 9830, 9830,3, 9175, 6553,2, 4096, 0,3
1090: *, 7372, 0,2, 5734, 0,3, 9011, 16384,2, 7372, 16384,3
1091: *, 10649, 16384,2, 0, 0,3, 3276, 16384,2, 13107, 16384,3
1092: *, 1474, 7372,2, 4685, 9502,3, 9830, 0,2, 0, 0,3
1093: *, 8192, 16384,2, 9011, 0,2, 7372, 0,3, 10649, 0,2
1094: * /
1095: DATA (JT(1,I),JT(2,I),JT(3,I),I=2121,2160)/
1096: *, 0, 0,3, 3276, 16384,2, 6553, 8192,2, 13107, 16384,2
1097: *, 9830, 0,2, 0, 0,3, 3276, 16384,2, 9830, 0,2
1098: *, 13107, 16384,2, 2621, 13107,3, 3276, 16384,2, 14254, 16384,2
1099: *, 13598, 13107,2, 3506, 6553,3, 4161, 9830,2, 3833, 8192,3
1100: *, 10420, 8192,2, 10747, 9830,3, 10092, 6553,2, 655, 3276,3
1101: *, 0, 0,2, 10977, 0,2, 11632, 3276,2, 4915, 0,3
1102: *, 3158, 623,2, 1919, 2399,2, 1385, 5057,2, 1638, 8192,2
1103: *, 2639, 11326,2, 4236, 13984,2, 6186, 15760,2, 8192, 16384,2
1104: *, 9948, 15760,2, 11187, 13984,2, 11721, 11326,2, 11468, 8192,2
1105: *, 10467, 5057,2, 8870, 2399,2, 6920, 623,2, 4915, 0,2
```

```
1106: * /
1107: DATA (JT(1,I),JT(2,I),JT(3,I),I=2161,2200)/
1108: *, 3158, 623,2, 1919, 2399,2, 1385, 5057,2, 1638, 8192,2
1109: *, 3276, 16384,3, 14581, 16384,2, 6103, 16384,3, 2826, 0,2
1110: *, 11755, 16384,3, 8478, 0,2, 1884, 0,3, 3768, 0,2
1111: *, 7536, 0,3, 9420, 0,2, 0, 0,3, 3276, 16384,2
1112: *, 8880, 16384,2, 10147, 16090,2, 11104, 15256,2, 11606, 14007,2
1113: *, 11575, 12533,2, 11016, 11060,2, 10015, 9811,2, 8724, 8976,2
1114: *, 7340, 8683,2, 1736, 8683,2, 12451, 13107,3, 13107, 16384,2
1115: *, 3276, 16384,2, 6553, 8192,2, 0, 0,2, 9830, 0,2
1116: *, 10485, 3276,2, 2949, 14745,3, 3276, 16384,2, 13107, 16384,2
1117: *, 12779, 14745,2, 8192, 16384,3, 4915, 0,2, 3276, 16384,3
1118: * /
1119: DATA (JT(1,I),JT(2,I),JT(3,I),I=2201,2240)/
1120: *, 4339, 15903,2, 4587, 14745,2, 1638, 0,2, 0, 0,3
1121: *, 3276, 0,2, 3276, 8192,3, 6553, 12288,2, 13107, 16384,2
1122: *, 5849, 2339,3, 3610, 2784,2, 1713, 4052,2, 445, 5950,2
1123: *, 0, 8188,2, 445, 10427,2, 1713, 12324,2, 3610, 13592,2
1124: *, 5849, 14037,2, 8087, 13592,2, 9985, 12324,2, 11252, 10427,2
1125: *, 11698, 8188,2, 11252, 5950,2, 9985, 4052,2, 8087, 2784,2
1126: *, 5849, 2339,2, 3610, 2784,2, 1713, 4052,2, 445, 5950,2
1127: *, 0, 8188,2, 5849, 0,3, 5849, 16377,2, 7018, 16377,3
1128: *, 4679, 16377,2, 4679, 0,3, 7018, 0,2, 0, 0,3
1129: *, 13107, 16384,2, 3276, 16384,3, 9830, 0,2, 2457, 12288,3
1130: * /
1131: DATA (JT(1,I),JT(2,I),JT(3,I),I=2241,2280)/
1132: *, 1966, 9830,2, 1962, 7641,2, 2832, 5778,2, 4444, 4533,2
1133: *, 6553, 4096,2, 8835, 4532,2, 10944, 5775,2, 12559, 7635,2
1134: *, 13434, 9830,2, 13926, 12288,2, 5734, 0,3, 9011, 16384,2
1135: *, 10649, 16384,3, 7372, 16384,2, 4096, 0,3, 7372, 0,2
1136: *, 0, 0,3, 2457, 0,2, 3440, 4915,2, 1819, 6253,2
1137: *, 968, 8208,2, 1015, 10487,2, 1953, 12751,2, 3642, 14662,2
1138: *, 5830, 15937,2, 8192, 16384,2, 10374, 15937,2, 12052, 14662,2
1139: *, 12977, 12751,2, 13009, 10487,2, 12145, 8208,2, 10512, 6253,2
1140: *, 8355, 4915,2, 7372, 0,2, 9830, 0,2, 0, 14745,3
1141: *, 125, 15375,2, 481, 15905,2, 1012, 16259,2, 1638, 16384,2
1142: * /
1143: DATA (JT(1,I),JT(2,I),JT(3,I),I=2281,2320)/
1144: *, 2264, 16259,2, 2795, 15905,2, 3151, 15374,2, 3276, 14745,2
1145: *, 3152, 14118,2, 2796, 13587,2, 2265, 13231,2, 1638, 13107,2
1146: *, 1011, 13231,2, 479, 13587,2, 124, 14118,2, 0, 14745,2
1147: *, 4096, 8192,3, 2528, 8503,2, 1199, 9391,2, 311, 10720,2
1148: *, 0, 12288,2, 311, 13855,2, 1199, 15184,2, 2528, 16072,2
1149: *, 4096, 16384,2, 5663, 16072,2, 6992, 15184,2, 7880, 13855,2
1150: *, 8192, 12288,2, 7880, 10720,2, 6992, 9391,2, 5663, 8503,2
1151: *, 4096, 8192,2, 4096, 0,2, 0, 4915,3, 8192, 4915,2
1152: *, 4096, 8192,3, 5663, 7880,2, 6992, 6992,2, 7880, 5663,2
1153: *, 8192, 4096,2, 7880, 2528,2, 6992, 1199,2, 5663, 311,2
1154: * /
1155: DATA (JT(1,I),JT(2,I),JT(3,I),I=2321,2360)/
1156: *, 4096, 0,2, 2528, 311,2, 1199, 1199,2, 311, 2528,2
1157: *, 0, 4096,2, 311, 5663,2, 1199, 6992,2, 2528, 7880,2
1158: *, 4096, 8192,2, 4096, 16384,2, 0, 13107,2, 4096, 16384,3
1159: *, 8192, 13107,2, 8192, 4915,3, 7892, 3034,2, 7040, 1439,2
1160: *, 5764, 374,2, 4259, 0,2, 2755, 374,2, 1479, 1439,2
1161: *, 627, 3034,2, 327, 4915,2, 627, 6796,2, 1479, 8390,2
1162: *, 2755, 9456,2, 4259, 9830,2, 5764, 9456,2, 7040, 8390,2
1163: *, 7892, 6796,2, 8192, 4915,2, 8192, 11468,2, 7990, 13418,2
1164: *, 7146, 15077,2, 5819, 16132,2, 4259, 16384,2, 3052, 15975,2
1165: *, 2028, 15077,2, 1309, 13797,2, 983, 12288,2, 9830, 10977,3
1166: * /
1167: DATA (JT(1,I),JT(2,I),JT(3,I),I=2361,2400)/
1168: *, 1638, 10977,2, 5734, 0,2, 9830, 10977,2, 6553, 6553,3
1169: *, 4915, 6553,2, 4915, 8192,2, 6553, 8192,2, 6553, 6553,2
1170: *, 6553, 8192,3, 4915, 6553,2, 4915, 8192,3, 6553, 6553,2
```

src/cgview/jsymbol.f

```
1171:      *,      0,      0,3,  5406, 16384,2, 10928, 16384,3,  5406,      0,2
1172:      *,      0,      0,3, 10813,      0,2,  5406, 10813,3,  5406,      0,2
1173:      *,      0, 17203,3, 10923, 17203,2,  9830, 15564,2, 10923, 17203,3
1174:      *,  9830, 18841,2,      0, 14745,3, 13107, 14745,2,  819,  9830,3
1175:      *, 12288,  9830,2,  2457, 14745,3,  1638,      0,2, 10649, 14745,3
1176:      *, 11468,      0,2,  1097,  4915,3,  494,  4397,2,  124,  3845,2
1177:      *,      0,  3276,2,  623,  2022,2,  2399,  959,2,  5057,  249,2
1178:      *
1179:      DATA (JT(1,I),JT(2,I),JT(3,I),I=2401,2440)/
1180:      *  8192,      0,2,  9473,      40,2, 10723,      160,2, 11911,      357,2
1181:      * 13007,      625,2, 13984,      959,2, 14819,      1350,2, 15491,      1789,2
1182:      * 15983,      2264,2, 16283,      2764,2, 16384,      3276,2, 15286,      4915,2
1183:      *  4915,      3604,3,  4246,      4087,2,      3723,      4781,2,      3391,      5627,2
1184:      *  3276,      6553,2,      3478,      7774,2,      4053,      8809,2,      4915,      9502,2
1185:      *  5583,      9985,2,      6106,      10679,2,      6439,      11526,2,      6553,      12451,2
1186:      *  6351,      13672,2,      5776,      14707,2,      4915,      15400,2,      8192,      3604,3
1187:      *  7522,      4087,2,      6999,      4783,2,      6666,      5631,2,      6553,      6553,2
1188:      *  6756,      7777,2,      7331,      8810,2,      8192,      9502,2,      8809,      9985,2
1189:      *  9383,      10679,2,      9716,      11526,2,      9830,      12451,2,      9628,      13672,2
1190:      *
1191:      DATA (JT(1,I),JT(2,I),JT(3,I),I=2441,2480)/
1192:      *  9053,      14707,2,      8192,      15400,2,      11468,      3604,3,      10801,      4086,2
1193:      *  10278,      4779,2,      9945,      5624,2,      9830,      6553,2,      10031,      7771,2
1194:      *  10606,      8808,2,      11468,      9502,2,      12137,      9985,2,      12660,      10679,2
1195:      *  12992,      11526,2,      13107,      12451,2,      12905,      13672,2,      12330,      14707,2
1196:      *  11468,      15400,2,      0,      0,0,      0,      0,0,      0,      0,0
1197:      *  0,      0,0,      0,      0,0,      0,      0,0,      0,      0,0
1198:      *  0,      0,0,      0,      0,0,      0,      0,0,      0,      0,0
1199:      *  0,      0,0,      0,      0,0,      0,      0,0,      0,      0,0
1200:      *  0,      0,0,      0,      0,0,      0,      0,0,      0,      0,0
1201:      *  0,      0,0,      0,      0,0,      0,      0,0,      0,      0,0
1202:      *
1203:      DATA (LSHIFT(I),I=1,72) /
1204:      * 113, 114, 115, 116, 117, 118, 119, 120, 121, 112
1205:      *  106,  31,  22,  27,  49,  48,  53,  54,  19,  26
1206:      *  30, 206, 205,  52, 202,  33,  57,  32,  51,  50
1207:      *  25,  28,  61,  23,  24, 207, 209, 208,  20, 109
1208:      *  203, 204, 211, 212,  25,  25,  25,  25,  25,  25
1209:      *  25,  25,  25,  25,  25,  25,  25,  25,  25,  25
1210:      *  25,  25,  25,  25,  25,  25,  25,  25,  25,  25
1211:      *  25,  25
1212:      *
1213: CCCCC DATA SFTKEY /'[' /
1214: CCCCC === JULY 1989      JAIS =====
1215: CCCCC DATA SFTKEY /'@' /
1216: CCCCC DATA SMALL /'@' /
1217: CCCCC DATA BACKSP /';' /
1218: CCCCC DATA SUB /'_' /
1219: CCCCC DATA SUPER /'"' /
1220: CCCCC =====
1221: CCCCC DATA SFTCH(1:36),SFTCH(37:72) /
1222: CCCCC *'1234567890!#$%&()'=-^_{ } :<>+_.',
1223: CCCCC *',/]?
1224: CCCCC *
1225: C
1226: CCCCC DATA GREEK /'abgdezhciklmnoprstufyvwABGDEZHCIKLMNOPRSTUFYVW' /
1227: C
1228: CCCCC DATA CH /
1229: CCCCC *      ~_      { ' , '      }      ' ,
1230: CCCCC *      ' & jklmnopqr!$*);^~/st' , 'uvwxyz , % >?01234567',
1231: CCCCC *      '89:#@' '=' "ABCDEFGHIJKL', 'MNOPQRSTUVWXYZ' /
1232: C
1233: CC/#ENDIF
1234: C
1235: END
```

src/cgview/kpinit.f

```
1:      subroutine KPINIT
2: C=====
3: C purpose: initialize particle symbol strings for MVP
4: C called in: MAIN
5: C calls : none
6: C-----
7: C initialization in this routine must synchronize with contents of
8: C included files _KPIDS and _KPSYMS
9: C=====
10:      include '../shared/INC/_IOUNIT'
11:      include 'INC/_FLAGS'
12: C
13:      include 'INC/_KPIDS'
14:      include 'INC/_KPSYMS'
15: C
16: C
17: C ... number of particles possible to treat in this code
18: C (this should be maximum of possible KP* )
19: C
20: C parameter ( KPLIM = ? )
21: C
22: C
23: C ... particle symbol string and their alias (or short form)
24: C
25: C character*16 KPSYM
26: C common /CKPSYM/ KPSYM(2,KPLIM)
27: C
28: C
29: C-----
30: C
31:      if ( MKPAR.lt.KPLIM ) then
32:          write(IMG,*) 'XXX(KPINIT) Array size of JKPAR(*) must be ,
33:          & 'greater than KPLIM. Program error. Check your source!!!'
34:          stop 666
35:      end if
36: C
37: C-----
38: C
39: C == possible initialization
40: C
41: C ... neutron and photon
42: C KPSYM(1,KPNEUT) = 'NEUTRON'
43: C KPSYM(2,KPNEUT) = 'N'
44: C KPSYM(1,KPPHOT) = 'PHOTON'
45: C KPSYM(2,KPPHOT) = 'P'
46: C ... electron and proton
47: C KPSYM(1,KPELEC) = 'ELECTRON'
48: C KPSYM(2,KPELEC) = 'EL'
49: C KPSYM(1,KPPROT) = 'PROTON'
50: C KPSYM(2,KPPROT) = 'PR'
51: C ... pi mesons (pi-0 pi+ pi- )
52: C KPSYM(1,KPPIO) = 'PIO'
53: C KPSYM(2,KPPIO) = 'PI0'
54: C KPSYM(1,KPPIP) = 'PI-PLUS'
55: C KPSYM(2,KPPIP) = 'PIP'
56: C KPSYM(1,KPPIM) = 'PI-MINUS'
57: C KPSYM(2,KPPIM) = 'PIM'
58: C ... mu mesons (mu+ mu- )
59: C KPSYM(1,KPMUP) = 'MU-PLUS'
60: C KPSYM(2,KPMUP) = 'MUP'
61: C KPSYM(1,KPMUM) = 'MU-MINUS'
62: C KPSYM(2,KPMUM) = 'MUM'
63: C
64:      return
65:      end
```


src/cgview/kpsymb.f

```
1:      subroutine KPSYMB( PSYM, DIR,  PID,  MODE )
2: C=====
3: C purpose: conversion to/from particle symbol from/to particle ID
4: C         for CGVIEW code
5: C-----
6: C arguments (i=input,o=output,w=work)
7: C
8: C i/o PSYM : particle symbol string
9: C i DIR : a character to specify direction of conversion
10: C      '>' : from symbol to ID
11: C          returns PID=0 if no matching symbol is found.
12: C      '<' : from ID to symbol
13: C          returns PSYM=' ' if no matching ID is found.
14: C o/i PID : particle ID number
15: C i MODE : returned symbol string when DIR='<';
16: C         MODE=0 : full symbol name
17: C         MODE=1(<>0) : short symbol name
18: C=====
19:      include '../shared/INC/_IOUNIT'
20: C
21:      include 'INC/_KPLIDS'
22:      include 'INC/_KPSYMS'
23: C
24:      character*(*) PSYM, DIR
25:      integer PID, MODE
26: C
27: C-----
28: C
29:      if ( DIR.eq.'>' ) then
30:          PID = 0
31: C
32: C      .... check full name first ....
33: C
34:          do 100 I = 1, KPLIM
35:              if ( KPSYM(1,I).eq.PSYM ) then
36:                  PID = I
37:                  go to 110
38:              end if
39:          100 continue
40:          110 continue
41:          if ( PID.gt.0 ) return
42: C
43: C      .... check short name (alias) ....
44: C
45:          do 120 I = 1, KPLIM
46:              if ( KPSYM(2,I).eq.PSYM ) then
47:                  PID = I
48:                  go to 130
49:              end if
50:          120 continue
51:          130 continue
52:          return
53: C
54:      else if ( DIR.eq.'<' ) then
55: C
56:          PSYM = ' '
57:          if ( PID.lt.1 .or. PID.gt.KPLIM ) return
58: C
59:          if ( MODE.eq.0 ) then
60:              PSYM = KPSYM(1,PID)
61:          else
62:              PSYM = KPSYM(2,PID)
63:          end if
64: C
65:      else
```

```
66:          write(IMGMSG,*) 'XXX(KPSYMB) Program error:',
67:      &          ' invalid conversion direction key ',
68:      &          '<', DIR,'> is given (must be ">" or "<").'
69:          stop 666
70:      end if
71: C
72:      return
73:      end
```

src/cgview/lostch.f

```

1:      subroutine LOSTCH( NGENE2, JCGVDB, MACT,
2:      &                  LSDED, XPLT, YPLT, BXPLT, BYPLT,
3:      &                  GDIST, AAAG, BBBG, CCCG,
4:      &                  COLOR, BCOLOR, STYLE, ILOOP, XMAX, YMAX )
5: C=====
6: C This routine draws "lost particle" marks.
7: C LSDED(I) has non-zero value when i'th particle is lost.
8: C=====
9: C/#IF MS_VISUAL
10: C
11: C ... This interface block is for CUTIL & MS-Visual tools. ...
12: C
13: *      interface
14: *      subroutine SETLWD( IIII )
15: *      integer      IIII
16: *CDEC$      ATTRIBUTES C : SETLWD
17: *CDEC$      ATTRIBUTES REFERENCE :: IIII
18: *      end subroutine
19: *      end interface
20: C/#ENDIF
21: C
22:      include 'INC/_FLAGS'
23: C
24:      integer NGENE2, LSDED(NGENE2), ILOOP, COLOR(*), BCOLOR(*), STYLE
25:      real*8 XPLT(NGENE2), YPLT(NGENE2), BXPLT(*), BYPLT(*)
26:      real*8 GDIST(*), AAAG(*), BBBG(*), CCCG(*)
27:      real*8 XMAX, YMAX
28:      integer JCGVDB(*)
29: C
30:      integer IFLG
31: C
32: C      common /DATA/      X,      Y
33:      real X(5), Y(5)
34: C
35: C -----
36: C
37: C      if ( JCGVDB(1).ne.0 ) then
38: C          write(*,*) '=== LOSTCH : NGENE2 ', NGENE2, ' MACT ', MACT
39: C      end if
40: C
41: C      CC/#IF PIFFLIB
42: C          call SETLWD( 1 )
43: C      CC/#ENDIF
44: C
45:      do 100 I = 1, NGENE2
46: C
47:          if ( LSDED(I).ne.0 ) then
48: C
49:              if ( XPLT(I).gt.XMAX ) then
50: C                  X(2) = XMAX
51: C                  IFLG1 = 0
52: C              else
53: C                  X(2) = XPLT(I)
54: C                  IFLG1 = 1
55: C              end if
56: C
57:              if ( YPLT(I).gt.YMAX ) then
58: C                  Y(2) = YMAX
59: C                  IFLG2 = 0
60: C              else
61: C                  Y(2) = YPLT(I)
62: C                  IFLG2 = 1
63: C              end if
64: C
65: C          .... IFLG = 0 : lost position is out of bound of drawing range.

```

```

66: C
67: C          IFLG = IFLG1*IFLG2
68: C
69: C          if ( GDIST(I).ge.0 ) then
70: C              IFLG = 1
71: C              X(2) = XPLT(I)
72: C              Y(2) = YPLT(I)
73: C          else
74: C              IFLG = 0
75: C              X(2) = XPLT(I) + GDIST(I)*AAAG(I)
76: C              Y(2) = YPLT(I) + GDIST(I)*BBBG(I)
77: C          end if
78: C
79: C -----
80: C .. lost particles in particle flight .....
81: C -----
82: C
83: C          if ( LSDED(I).eq.2 ) then
84: C
85: C              if ( STYLE.eq.0 ) then
86: C                  call NEWP( 0 )
87: C              end if
88: C
89: C              MARK = 1
90: C              if ( ILOOP.eq.2 ) then
91: C                  MARK = 2
92: C              end if
93: C
94: C          ... two lines "points" are on the same position , so
95: C          "LINE" routine is used only to draw a mark ...
96: C
97: C              X(1) = X(2)
98: C              Y(1) = Y(2)
99: C              call LINE( X, Y, 2, 1, -1*IFLG, MARK*IFLG, X(4), Y(4),
100: C                  &
101: C                      X(5), Y(5) )
102: C -----
103: C .. lost particles in source .....
104: C -----
105: C
106: C          else if ( LSDED(I).eq.1 ) then
107: C
108: C              if ( STYLE.eq.0 ) then
109: C                  call NEWP( 0 )
110: C              end if
111: C
112: C              MARK = 3
113: C              if ( ILOOP.eq.2 ) then
114: C                  MARK = 4
115: C              end if
116: C
117: C              X(1) = X(2)
118: C              Y(1) = Y(2)
119: C              call LINE( X, Y, 2, 1, -1*IFLG, MARK*IFLG, X(4), Y(4),
120: C                  &
121: C                      X(5), Y(5) )
122: C -----
123: C .. lost particles in next zone search .....
124: C -----
125: C
126: C          CCCCCC else if ( LSDED(I).eq.3.and.ILOOP.eq.1 ) then
127: C              else if ( LSDED(I).eq.3 ) then
128: C                  if ( STYLE.eq.0 ) then
129: C
130: C          ... draw line to lost point ...

```

src/cgview/lostch.f

```
131: C
132:         X(1)    = BXPLT(I)
133:         Y(1)    = BYPLT(I)
134:         call NEWP( BCOLOR(I) )
135:         call LINE( X, Y, 2, 1, 0, 0, X(4), Y(4), X(5), Y(5) )
136:         call NEWP( 0 )
137:     end if
138: C
139:         MARK     = 5
140:         if ( ILOOP.eq.2 ) then
141:             MARK  = 6
142:         end if
143: C
144: C     ... draw mark ...
145: C
146:         X(1)     = X(2)
147:         Y(1)     = Y(2)
148:         call LINE( X, Y, 2, 1, -1*IFLG, MARK*IFLG, X(4), Y(4)
149: &               X(5), Y(5) )
150:     end if
151: C
152:         LSDED(I) = 0
153:     end if
154: 100 continue
155: C
156: C
157:     call NEWP( 0 )
158:     if ( JXPS.eq.1 .or. STYLE.ne.0 ) then
159:         call SETLWD( 1 )
160:     else
161:         call SETLWD( 2 )
162:     end if
163: C
164:     return
165: end
```

src/cgview/main.f

```
1: C
2: C=====
3: C   JAEA CGVIEW-SLICE VERSION 3
4: C=====
5: C
6: C   CGVIEW_SLICE is a tool to draw 2D slice images of MVP/GMVP's
7: C   3D geometry.
8: C
9: C   * Version 1 developed on the ancient OpenWindows of Sun Microsystems,
10: C   with a splended Graphical User Interface which needs a paper manual
11: C   in your hand every time you use it ;- ) and many orphan
12: C   processes were remained after it crashed !!!
13: C   (and often it crashed...)
14: C
15: C   * In version 2, graphics was changed to pure X-Window for portability,
16: C   and enabled output in PostScript form.
17: C   The G.U.I. is cleverly removed.
18: C
19: C   * In version 3, (Jan 1998)
20: C
21: C   Line tracking core routines are synchronized with newest
22: C   ones in MVP/GMVP source, and new features such as
23: C   STGM region and free lattice frame and many commands convenient for
24: C   interactive (but non-GUI!!) use are supported.
25: C
26: C   =====
27: C   Apology for version 3
28: C   =====
29: C
30: C   Some comments are added to obscure parts in the code by M.Sasaki,
31: C   a hacker of CGVIEW V3, but they may not match what anonymous
32: C   programmers of version 1 and 2 had been going to do, because
33: C   no documents or valuable comments are given to the hacker ....
34: C
35: C   Anyway, this code includes tons of misleading and buggy codings ;- ).
36: C
37: C=====
38: C
39: C/ #IF CMAIN
40: C   subroutine FTMAIN
41: C/ #ELSE
42: C   program CGVIEW
43: C/ #ENDIF
44: C
45: C/ #IF MS_VISUAL
46: C
47: C   ... enable to use iargc, getarg for Visual Fortran.
48: C
49: C   use dflib
50: C   use dfport
51: C
52: C   ... This interface block is for CUTIL & MS-Visual tools. ...
53: C
54: C   interface
55: C   subroutine PLOTDEVICE( IDEVICE, PSFILE, LPSF )
56: C   integer IDEVICE, LPSF
57: C   character PSFILE*256
58: C* CDEC$ ATTRIBUTES C :: PLOTDEVICE
59: C* CDEC$ ATTRIBUTES REFERENCE :: IDEVICE, PSFILE, LPSF
60: C   end subroutine
61: C   end interface
62: C/ #ENDIF
63: C
64: C   common /PROID/ INFILE
65: C   INCLUDE '../shared/INC/_IOUNIT'
```

```
66: C   character INFILE*72, OFL*15
67: C   character PSFILE*256
68: C   character*256 CWRK
69: C   character*256 CSTRNG
70: C
71: C   character*8 CODE
72: C
73: C-----
74: C
75: C   call VERSTR
76: C
77: C   ... initialize I/O units
78: C
79: C   call IIOSSET
80: C
81: C   IOIN   = 5
82: C
83: C   MLIMIT = 200000000
84: C
85: C   ... common area data initialization (do not use BLOCK DATA)
86: C
87: C   call WDBL
88: C   call FLAGIN
89: C   call ERINIT
90: C
91: C   call CGVINIT
92: C   call KPINIT
93: C
94: C   ... path large value for MXCERR in CGVIEW to prevent sudden stop
95: C   in interactive mode
96: C   call FRINIT( INP, IPR, 1000 )
97: C   call INDROT
98: C
99: C   IDEVICE = 1
100: C   PSFILE = ' '
101: C
102: C   IPREIN = 0
103: C
104: C   ... check environment variables if possible ...
105: C
106: C/ #IF .NOT.NOGETENV
107: C   call CHKENV( 'CGVIEW', MLIMIT )
108: C/ #ENDIF
109: C
110: C-----
111: C   Program control information from command line arguments.
112: C-----
113: C
114: C   ... ARGNONE : FAT flag indicating no means to get command argument.
115: C
116: C/ #IF ARGV
117: C
118: C/ #   IF ARGV( GETARG2 )
119: C*     KJ = IARGC() - 1
120: C/ #   ELSEIF ARGV( MSF )
121: C*     KJ = NARGS() - 1
122: C/ #   ELSE
123: C*     KJ = IARGC()
124: C/ #   ENDIF
125: C
126: C   do 100 I = 1, KJ
127: C
128: C       IPREIN = IPREIN + 1
129: C/ #   IF SYSTEM( CRAY )
130: C*       call PXFGETARG(I, CSTRNG, LLLLL, ierr )
```

src/cgview/main.f

```

131: C/# ELSEIF ARGV( IGETARG )
132: *      LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
133: C/# ELSE
134: C/# IF ARGV( GETARG2 )
135: *      call GETARG(I+1, CSTRNG )
136: C/# ELSEIF ARGV( MSF )
137: C ... LL2 should be integer*2
138: *      call GETARG(I, CSTRNG, LL2 )
139: C/# ELSE
140: call GETARG( I, CSTRNG )
141: C/# ENDIF
142: LLLLL = MIN(INDEX(CSTRNG, ' ') - 1, LEN(CSTRNG))
143: if ( LLLLL.eq.0 ) LLLLL = LEN(CSTRNG)
144: C/# ENDIF
145:
146: C
147: C ... quit command line check if '@@' is encountered ...
148: C
149: C      if ( CSTRNG(1:LLLLL).eq.'@@' ) go to 110
150:
151: C
152: C ... "-v" "--version" "--version" output version and quit ...
153: C
154: C      if ( CSTRNG(1:LLLLL).eq.'-v' .or. CSTRNG(1:LLLLL).eq.'--version'
155: & .or. CSTRNG(1:LLLLL).eq.'--version' ) then
156: call GETVER( CSTRNG(1:128), CSTRNG(129:256) )
157: write(IPR,*) 'Version: ', CSTRNG(1:ICLEN2(CSTRNG(1:128)))
158: write(IPR,*) 'Release date: ',
159: & CSTRNG(129:128+ICLEN2(CSTRNG(129:256)))
160: stop
161: end if
162:
163: call PREINP( IPREIN, CSTRNG(1:LLLLL), IDEVICE, PSFILE, MLIMIT,
164: & CODE )
165:
166: 100 continue
167: C
168: 110 continue
169: C
170: if ( KJ.gt.0 ) write(6,(''1''))
171:
172: C ... END OF C/#IF ARGV ...
173: C/#ENDIF
174: C
175: C
176: C ..... input data from stdin
177: C
178: C      INFILE = ' '
179: C
180: C      write(6,*) ' Enter input file name for CGVIEW-SLICE'
181: C      read(5,('A72')) INFILE
182: C      do 100 I = 1, 72
183: C          IS = I
184: C          if ( INFILE(I:I).ne.' ' ) go to 110
185: C 100 continue
186: C 110 continue
187: C      do 120 I = 72, 1, -1
188: C          IE = I
189: C          if ( INFILE(I:I).ne.' ' ) go to 130
190: C 120 continue
191: C 130 continue
192: C
193: C      write(6,*) ' Input file: <', INFILE(IS:IE), '>'
194: C
195: C      write(6,*) ' Input file name : ', INFILE(IS:IE)

```

```

196: c      open( IOIN, file =INFILE(IS:IE), form ='FORMATTED', status ='OLD',
197: c & iostat =IOS )
198: c      if ( IOS.ne.0 ) then
199: c          write(6,*) 'XXX Failed to open input file : code ', IOS
200: c          stop 888
201: c      end if
202: c
203: c 140 continue
204: c      write(6,*) '*** Plot Interface Library Version 1.30 ***'
205: c      write(6,*) '*** Modified for CGVIEW by M.Sasaki ***'
206: c      write(6,*) ' Output Device'
207: c      write(6,*) '      1 : X-Window'
208: c      write(6,*) '      2 : PostScript File'
209: c      write(6,*) '      3 : EPS File for TeX'
210: c      write(6,*) ' Select [ 1 - 3 ] : '
211: c      read(*,*) IDEVICE
212: c      if ( IDEVICE.ne.1.and.IDEVICE.ne.2.and.IDEVICE.ne.3 ) go to 140
213: c
214: c      PSFILE = ' '
215: c      LPSF = 0
216: c      if ( IDEVICE.eq.2 .or. IDEVICE.eq.3 ) then
217: c 150 write(6,*) 'Input PS or EPS file name. '
218: c      write(6,*) '(default PS is piflib.PS)'
219: c      write(6,*) '(default EPS is piflib[page].EPS)'
220: c      CWRK = ' '
221: c      read(*,('a')) CWRK
222: c      PSFILE = ' '
223: c      LPSF = 0
224: c
225: c      do I = 1, LEN(CWRK)
226: c          if ( CWRK(I:I).ne.' ' ) then
227: c              LPSF = LPSF + 1
228: c              PSFILE(LPSF:LPSF) = CWRK(I:I)
229: c          end if
230: c      end do
231: c      end if
232: c
233: c ==== Set output device =====
234: c
235: c      LPSF = ICLEN2(PSFILE)
236: c      call PLOTDEVICE( IDEVICE, PSFILE(:LPSF), LPSF )
237: c
238: c      call AAALOC
239: c      call ACALOC
240: c
241: c ==== Tracking control =====
242: c
243: c      call CENTER( JXPS, MLIMIT, CODE )
244: c
245: c      stop
246: c      end

```

src/cgview/mcncso.f

```

1:      subroutine MCNCISO( IMC,   IPR,   IUG1,  IUB,   IUG2,  KS,    MS,
2:      &                   TITLE, CODE,  KBSM,  IBSM,  KSMCN, SMCN,  TMCN,
3:      &                   KINPZ, KZDA,  KZAA,  SDA,   KZMAT, KMAT,
4:      &                   KSFBD, IZNAM, IBSDA, KREG,  KREGN, KREGI,
5:      &                   TNAMS, IDMAT, MNUC, INUCT, DENST, LPDEN,
6:      &                   NCIDI, NUCID, TEMPN )
7: C=====
8: C purpose: output MCNP input data on I/O unit IMC
9: C=====
10: C
11:      implicit real*8(D)
12: C
13:      character*72 TITLE(2)
14:      character*(*) CODE
15: C
16:      integer KBSM(NBODY+1), IBSM(KS)
17:      integer KSMCN(2,KS+1)
18:      real*8 SMCN(MS)
19: C
20:      character*4 TMCN(KS)
21: C
22:      include 'INC/_KPIDS'
23: C
24:      include '../shared/INC/_SIZES'
25:      include '../mvp/INC/_CXSEC'
26:      include 'INC/_FLAGS'
27:      include '../shared/INC/_LNAM'
28: C
29:      integer KINPZ(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
30:      real*8 SDA(NSDA)
31:      integer KZMAT(NZONE,2), KMAT(NINPZ)
32:      integer KSFBD(NZDA)
33:      integer IBSDA(3,NBODY)
34:      integer KREG(NINPZ)
35:      character*12 KREGN(NINPZ)
36:      integer KREGI(NINPZ)
37:      character*12 IZNAM(*)
38:      character*(LNAM) TNAMS(*)
39: C
40:      integer IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1)
41:      integer INUCT(*)
42:      real DENST(*)
43:      character*16 NCIDI(NUC), NUCID(NUC)
44:      real*8 TEMPN(NUC)
45: C
46: C ..... local data
47: C
48:      character*12 DT
49:      character*8 TM
50:      character*72 LINE
51:      character*200 BUF
52:      character*200 BUF2
53:      character*6 CWK
54: C
55: C-----
56: C
57:      call RWIND( IUG1 )
58:      call RWIND( IUG2 )
59:      call RWIND( IMC )
60: C
61:      call HIZUKE( DT )
62:      call JIKAN( TM )
63:      LINE = '=== Date : '//DT//' Time '//TM
64: C
65:      write(IMC,'(1x,A)') TITLE(1) (:ICLEN2(TITLE(1)))

```

```

66:      write(IMC,'(''C      '' ,A)') TITLE(2) (:ICLEN2(TITLE(2)))
67:      write(IMC,7000)
68:      &      '=== geometry data generated from MVP/GMVP input by CGVIEW',
69:      &      LINE(:ICLEN2(LINE)), ' '
70:      7000 format('C      ',A)
71: C
72: C
73: C === cell cards =====
74: C
75:      ICELL = 0
76:      do IZ = 1, NZONE
77: C
78: C      ... cell ID is MVP/GMVP zone # times 10
79: C
80:          IDC = 10*IZ
81: C
82:          write(IMC,7020) IZNAM(KINPZ(IZ)) (:ICLEN2(IZNAM(KINPZ(IZ)))),
83:          &          KREGN(KINPZ(IZ))
84:          7020 format('C      === ',A,' reg=',A)
85: C
86: C      ... cell : cell-ID mat# [factor] ...
87: C
88:          LINE = ' '
89:          M = KZMAT(IZ,1)
90:          XXIMP = 1.0
91: C
92:          if ( M.eq.0 .or. M.eq.-1000 .or. M.eq.-2000 .or. M.eq.-3000 )
93:          &          then
94:              if ( M.ne.0 ) XXIMP = 0
95:              write(LINE,7040) IDC
96:              7040 format(I5,' 0')
97:          else
98:              do IM = 1, NMAT
99:                  if ( IDMAT(IM).eq.KZMAT(IZ,1) ) go to 100
100:              end do
101:              DENNUM = 0.0
102:              do N = LPDEN(IM), LPDEN(IM+1) - 1
103:                  DENNUM = DENNUM + DENST(N)
104:              end do
105:              write(LINE,7060) IDC, M, DENNUM
106:              7060 format(I5,I5,1X,1P,E11.4)
107:          end if
108: C
109:          LN = ICLEN2(LINE) + 1
110: C
111: C      ... compose body --> surface combination
112: C
113:          IBID = 0
114:          do J = KZAA(IZ), KZAA(IZ+1) - 1
115:              if ( KSFBD(J).ne.IBID ) then
116:                  IBID = KSFBD(J)
117: C
118:                  do IB = 1, NBODY
119:                      if ( IBSDA(2,IB).eq.IBID ) go to 110
120:                  end do
121:                  110 continue
122: C
123:          JS = SIGN(1,KZDA(1,J))
124:          BUF = ' '
125:          if ( JS.lt.0 ) then
126:              BUF = '#('
127:          end if
128:          CHECK %%%
129:          c      write(*,*) '%%% ',ib,' kbsm ',KBSM(IB), KBSM(IB+1) - 1
130:          c %%%

```

src/cgview/mcncso.f

```

131:      do K = KBSM(IB), KBSM(IB+1) - 1
132:      ISS      = IBSM(K)
133:      ISSA     = ABS(ISS)
134: C
135: C      ... this surface is identical to other one.
136: C      if ( ABS(KSMCN(2,ISSA)).ne.ISSA ) then
137: C      ISS      = SIGN(1,ISS)*KSMCN(2,ISSA)
138: C      end if
139: C
140: C      CWK      = ' '
141: C
142: C      ... surface ID is 10 * surface #
143: C
144: C      IDS      = 10*ISS
145: C
146: C      write(CWK,'(i6)') IDS
147: C      call CCOMP( CWK, LEN(CWK), CWK, IIE )
148: C
149: C      BUF2     = BUF(:ICLEN2(BUF)) //' '//CWK
150: C      BUF      = BUF2
151: C      end do
152: C      if ( JS.lt.0 ) then
153: C      BUF2     = BUF(:ICLEN2(BUF)) //' '
154: C      BUF      = BUF2
155: C      end if
156: C
157: C      ....
158: C
159: C      NSPACE = 2
160: C      call ADDLIN( IMC, LINE, LN, ' ', BUF )
161: C      end if
162: C      end do
163: C
164: C      ... append cell parameters ...
165: C
166: C      if ( XXIMP.eq.0 ) then
167: C      BUF      = 'IMP:N=0'
168: C      else
169: C      BUF      = 'IMP:N=1'
170: C      end if
171: C
172: C      NSPACE = 2
173: C      call ADDLIN( IMC, LINE, LN, ' ', BUF )
174: C
175: C      .....
176: C      if ( ICLEN2(LINE).gt.0 ) write(IMC,'(a)') LINE(:ICLEN2(LINE))
177: C
178: C      end do
179: C      write(IMC,7080)
180: C      7080 format('C      == End of cell definition ====')
181: C
182: C      === surface =====
183: C
184: C      write(IMC,'()')
185: C
186: C      do IB = 1, NBODY
187: C      check %%%%%%%%%
188: C      write(*,*) '%% -- IBSDA(3,IB) ',IBSDA(3,IB)
189: C      %%%%%%%%%
190: C      call TYPBOD( CWK, '<', IBSDA(3,IB) )
191: C      write(IMC,7100) CWK, IBSDA(2,IB)
192: C      7100 format('C      == ',A,I6)
193: C
194: C      do K = KBSM(IB), KBSM(IB+1) - 1
195: C      ISS      = ABS(IBSM(K))

```

```

196:      IDS      = 10*ISS
197: C      if ( KSMCN(2,ISS).eq.ISS ) then
198: C      write(IMC,7120) IDS, TMCN(ISS),
199: C      &      (SMCN(I),I=KSMCN(1,ISS),KSMCN(1,ISS+1)-1)
200: C      7120 format('I5,I4,1X,1P,4E15.7/11X,4E15.7:/11X,4E15.7:/11X,
201: C      &      4E15.7:/11X,4E15.7)
202: C      else
203: C      write(IMC,7140) IDS, 10*KSMCN(2,ISS)
204: C      7140 format('C      ** surface ',I5,' is same as surface ',I6)
205: C      end if
206: C      end do
207: C      end do
208: C      write(IMC,7160)
209: C      7160 format('C      == End of surface definition ====')
210: C      write(IMC,'()')
211: C
212: C      .... material composition ....
213: C
214: C      if ( CODE.eq.'MVP' ) then
215: C      NERR      = 0
216: C      do IM = 1, NMAT
217: C      write(IMC,7180) IDMAT(IM)
218: C      7180 format('C      == Material ',I6)
219: C      LINE      = ' '
220: C      write(LINE,'(''M'',i8)') IDMAT(IM)
221: C      call CCOMP( LINE, LEN(LINE), LINE, IIE )
222: C      write(IPR,7000)
223: C      do N = LPDEN(IM), LPDEN(IM+1) - 1
224: C      K          = INUCT(N)
225: C      call MCNNID( NCIDI(K), MMMM, IE )
226: C      if ( IE.ne.0 ) then
227: C      NERR      = NERR + 1
228: C      write(IMC,7200) 'C', NCIDI(K)
229: C      write(IPR,7200) ' ', NCIDI(K)
230: C      7200 format(A,'      *** no matching MCNP ID for ',A)
231: C      end if
232: C      write(IMC,7220) LINE(1:9), MMMM, DENST(N), NCIDI(K)
233: C      LINE      = ' '
234: C      7220 format(A,I8,2X,E15.7,' $ ',A)
235: C      end do
236: C      end do
237: C      end if
238: C
239: C      write(IPR,*) '=== Conversion to MCNP geometry is completed.'
240: C
241: C      return
242: C      end
243: C
244: C
245: C=====
246: C      output MCNP input card by appending a string
247: C
248: C      * up to 72 characters on a line.
249: C      * place 6 blanks on the head of new line.
250: C
251: C      subroutine ADDLIN( IMC, LINE, LN, SPACE, BUF )
252: C
253: C      character*(*) LINE, BUF, SPACE
254: C
255: C      ... local
256: C
257: C      character*256 CWK
258: C
259: C-----
260: C

```

src/cgview/mcncso.f

```
261:      LB      = ICLEN2(BUF)
262:      JAPP     = 1
263: C
264:      if ( LN+LEN(SPACE)+LB.gt.72 ) then
265:          write(IMC,'(a)') LINE(:ICLEN2(LINE))
266:          LINE   = ' '
267:          LN     = 6
268:          JAPP   = 0
269:      end if
270: C
271: C ... << caution : assuming length of buf is less than 72-6 !!!>>
272: C
273:      CWK      = LINE
274:      if ( JAPP.eq.0 ) then
275:          LINE   = CWK(:LN) //BUF(:LB)
276:      else
277:          LINE   = CWK(:LN) //SPACE//BUF(:LB)
278:      end if
279:      LN       = ICLEN2(LINE)
280: C
281:      return
282: end
```


src/cgview/mcnnid.f

```

1:      subroutine MCNNID( MVPID, MCNID, IERR )
2: C=====
3: C purpose: convert MVP nuclide ID to MCNP nuclide ID
4: C
5: C Assuming MVPID is in standard form as follows:
6: C
7: C   ABNWxxxx :
8: C   AB: element name(B may be '0' for single character atom)
9: C   N : last digit of mass number (ouch! why not full number...)
10: C      or 'N' for natural composition
11: C   W : '0' or special symbol for data including thermal scattering
12: C      data etc.
13: C
14: C=====
15:      character*(*) MVPID
16:      integer MCNID
17: C
18: C ---- CHEMICAL SYMBOLS IN ORDER OF ATOMIC NUMBER ----
19: C
20:      parameter( MATOM = 103 )
21:      character*2 CSYMBL(MATOM)
22: C
23: C ... first 3 character of MVPID and mass number
24: C   ( taken from JENDL3.2 ARLIB of Nov 1998 )
25: C
26:      parameter( MCV= 300 )
27:      character*3 CMV(MCV)
28:      integer MASS(MCV)
29: C
30:      character*2 AN
31: C
32:      data (CSYMBL(I),I=1,MATOM) /
33: 1 'H ',
34: 2 'LI','BE',
35: 3 'NA','MG',
36: 4 'K ', 'CA','SC','TI','V ', 'CR','MN','FE','CO','NI','CU','ZN',
37: & 'GA','GE','AS','SE','BR','ER',
38: 5 'RB','SR','Y ', 'ZR','NB','MO','TC','RU','RH','PD','AG','CD',
39: & 'IN','SN','SB','TE','I ', 'XE',
40: 6 'CS','BA',
41: L 'LA','CE','PR','ND','PM','SM','EU','GD','TB','DY','HO',
42: L 'ER','TM','YB','LU',
43: & 'HF','TA','W ', 'RE','OS','IR','PT','AU','HG',
44: & 'TL','PB','BI','PO','AT','RN',
45: 7 'FR','RA',
46: A 'AC','TH','PA','U ', 'NP','PU','AM','CM','BK','CF','ES',
47: A 'FM','MD','NO','LR' /
48: C
49: C
50:      data (CMV(I),MASS(I),I=1,50) /
51: &'H01',001, 'D02',002, 'HE3',003, 'HE4',004, 'LI6',006,
52: &'LI7',007, 'BE9',009, 'B00',010, 'B01',011, 'C02',012,
53: &'N04',014, 'N05',015, 'O06',016, 'F09',019, 'NA3',023,
54: &'AL7',027, 'P01',031, 'V01',051, 'MN5',055, 'CO9',059,
55: &'Y09',089, 'ZR0',090, 'ZR1',091, 'ZR2',092, 'ZR3',093,
56: &'ZR4',094, 'ZR6',096, 'NB3',093, 'MO5',095, 'TC9',099,
57: &'RU1',101, 'RH3',103, 'PD5',105, 'AG7',107, 'AG9',109,
58: &'CD3',113, 'IN3',113, 'IN5',115, 'I07',127, 'I09',129,
59: &'XE1',131, 'XE5',135, 'CS3',133, 'CS4',134, 'CS7',137,
60: &'CE0',140, 'CE2',142, 'PR3',143, 'ND3',143, 'ND5',145/
61:
62:      data (CMV(I),MASS(I),I=51,100) /
63: &'ND7',147, 'ND8',148, 'PM7',147, 'PMG',148, 'PM9',149,
64: &'SM7',147, 'SM8',148, 'SM9',149, 'SM0',150, 'SM1',151,
65: &'SM2',152, 'EU3',153, 'EU4',154, 'EU5',155, 'EU6',156,

```

```

66: &'GD2',152, 'GD4',154, 'GD5',155, 'GD6',156, 'GD7',157,
67: &'GD8',158, 'GD0',160, 'HF4',174, 'HF6',176, 'HF7',177,
68: &'HF8',178, 'HF9',179, 'HF0',180, 'TA1',181, 'BI9',209,
69: &'TH0',230, 'TH2',232, 'PA1',231, 'PA3',233, 'U02',232,
70: &'U03',233, 'U04',234, 'U05',235, 'U06',236, 'U07',237,
71: &'U08',238, 'NP6',236, 'NP7',237, 'NP8',238, 'NP9',239,
72: &'PU6',236, 'PU8',238, 'PU9',239, 'PU0',240, 'PU1',241/
73:
74:      data (CMV(I),MASS(I),I=101,150) /
75: &'PU2',242, 'AM1',241, 'AMG',242, 'AM3',243, 'AM4',244,
76: &'CM1',241, 'CM2',242, 'CM3',243, 'CM4',244, 'CM5',245,
77: &'CM6',246, 'CM7',247, 'CM8',248, 'CM9',249, 'CM0',250,
78: &'BK9',249, 'BK0',250, 'CF9',249, 'CF0',250, 'CF1',251,
79: &'CF2',252, 'CF4',254, 'ES4',254, 'ES5',255, 'FM5',255,
80: &'ND7',147, 'ND8',148, 'CS4',134, 'CS7',137, 'H01',001,
81: &'BE9',009, 'B00',010, 'B01',011, 'N04',014, 'O06',016,
82: &'F09',019, 'NA3',023, 'AL7',027, 'CR0',050, 'CR2',052,
83: &'CR3',053, 'CR4',054, 'MN5',055, 'FE4',054, 'FE6',056,
84: &'FE7',057, 'FE8',058, 'CO9',059, 'NI8',058, 'NI0',060/
85:
86:      data (CMV(I),MASS(I),I=151,200) /
87: &'NI1',061, 'NI2',062, 'NI4',064, 'CU3',063, 'CU5',065,
88: &'ZR0',090, 'ZR1',091, 'ZR2',092, 'ZR3',093, 'ZR4',094,
89: &'ZR5',095, 'ZR6',096, 'NB3',093, 'I05',135, 'ER6',166,
90: &'ER7',167, 'TA1',181, 'AU7',197, 'PB6',206, 'PB7',207,
91: &'PB8',208, 'TH0',230, 'TH2',232, 'PA1',231, 'PA3',233,
92: &'U02',232, 'U03',233, 'U04',234, 'U05',235, 'U06',236,
93: &'U07',237, 'U08',238, 'NP7',237, 'NP8',238, 'NP9',239,
94: &'PU6',236, 'PU7',237, 'PU8',238, 'PU9',239, 'PU0',240,
95: &'PU1',241, 'PU2',242, 'PU3',243, 'AM1',241, 'AMG',242,
96: &'AM3',243, 'CM1',241, 'CM2',242, 'CM3',243, 'CM4',244/
97:
98:      data (CMV(I),MASS(I),I=201,250) /
99: &'CM5',245, 'CM6',246, 'CM7',247, 'CM8',248, 'BK9',249,
100: &'CF9',249, 'CF0',250, 'CF1',251, 'CF2',252, 'CF3',253,
101: &'ES3',253, 'AL7',027, 'U05',235, 'PU1',241, 'AM1',241,
102: &'ER2',162, 'ER4',164, 'ER6',166, 'ER7',167, 'ER8',168,
103: &'ER0',170, 'H01',001, 'B00',010, 'B01',011, 'N04',014,
104: &'N05',015, 'O06',016, 'F09',019, 'AL7',027, 'CR0',050,
105: &'CR2',052, 'CR3',053, 'CR4',054, 'MN5',055, 'FE4',054,
106: &'FE6',056, 'FE7',057, 'FE8',058, 'NI8',058, 'NI0',060,
107: &'NI1',061, 'NI2',062, 'NI4',064, 'ER6',166, 'ER7',167,
108: &'TH2',232, 'U03',233, 'U04',234, 'U05',235, 'U06',236/
109:
110:      data (CMV(I),MASS(I),I=251,300) /
111: &'U08',238, 'NP7',237, 'PU8',238, 'PU9',239, 'PU0',240,
112: &'PU1',241, 'PU2',242, 'AM1',241, 'AM3',243, 'CM4',244,
113: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
114: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
115: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
116: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
117: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
118: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
119: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0,
120: &' ',0, ' ',0, ' ',0, ' ',0, ' ',0, ' ',0/
121: C
122: C
123: C-----
124: C
125:      IERR = 0
126:      if ( index('0123456789',MVPID(2:2)).ne.0 ) then
127:          AN = MVPID(1:1)
128:      else
129:          AN = MVPID(1:2)
130:      end if

```

src/cgview/mcnnid.f

```
131:
132: C   .. deuterion
133:       if ( AN.eq.'D0' ) then
134:           KA = 1
135:           goto 100
136:       end if
137: C
138:       do KA=1,MATOM
139:           if(CSYMBL(KA).eq.AN) goto 100
140:       end do
141: C
142:       IERR = 1
143:       MCNID = 0
144:       return
145: C
146: 100 MCNID = KA*1000
147: C
148: C   ... natural composition
149: C
150:       if(MVPID(3:3).eq.'N' ) then
151:           return
152:       end if
153: C
154: C   ... mass number determination is ... highly ad hoc ....
155: C
156:       do k=1,MCV
157:           if(MVPID(1:3).eq.CMV(K)) then
158:               MCNID = MCNID + MASS(K)
159:               return
160:           end if
161:       end do
162: C
163:       IERR = 2
164: C
165:       return
166: end
```

src/cgview/mcnout.f

```

1:      subroutine MCNOUT( IMC, IPR, IUG1, IUG2, IUB, IERR,
2:      &                  LIMIT, A, DA8, IA, LIMITC,CHA, TITLE,
3:      &                  CODE,
4:      &                  DBODI, IBODI, IPSDA, KINPZ, KZDA, KZAA, SDA,
5:      &                  KZMAT, KMAT, KSFBD, IZNAM, IBSDA, KREG,
6:      &                  KREGN, KREGI, TNAMS,
7:      &                  IDMAT, MNUC, INUCT, DENST, LPDEN,
8:      &                  NCIDI, NUCID, TEMPN
9:      &                  )
10: C=====
11: C Purpose: output geometry data of MCNP (experimental!!!!)
12: C
13: C <<attention>> array KMAT and KZMAT contain material ID as input
14: C when called in CGVIEW code which does not convert
15: C material IDs to material numbers.
16: C
17: C-----
18: C RESTRICTIONS: (Jun 1999)
19: C * cannot convert lattice geometry.
20: C * cannot treat cones(leaf selection is not supported) and tori.
21: C * cannot create TR card.
22: C * cannot create cell parameters other than IMP:N=1 (or 0)
23: C (MVP/GMVP importance is energy dependent but MCNPs is not.)
24: C * do not check identity of general quadratic surfaces (GQ)
25: C * reflective material zone is converted into outer void !!!
26: C-----
27: C i IMC : I/O unit on which MCNP input data is output
28: C i IPR : I/O unit of message output
29: C i IUG1 : scratch I/O unit (text)
30: C i IUG2 : scratch I/O unit (text)
31: C i IUB : scratch I/O unit (binary)
32: C o IERR : 0 when conversion is successful else non-zero.
33: C=====
34:      implicit real*8(D,H)
35: C
36:      include 'INC/_KPIDS'
37: C
38:      include '../shared/INC/_SIZES'
39:      include '../mvp/INC/_CXSEC'
40:      include 'INC/_FLAGS'
41:      include '../shared/INC/_LNAME'
42: C
43:      real A(*)
44:      real*8 DA8(*)
45:      integer IA(*)
46:      character*4 CHA(*)
47: C
48:      character*72 TITLE(2)
49:      character*(*) CODE
50: C
51:      real*8 DBODI(*)
52:      integer IBODI(NBODY+1)
53:      integer IPSDA(3,NSURF+1)
54: C
55:      integer KINPZ(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
56:      real*8 SDA(NSDA)
57:      integer KZMAT(NZONE,2), KMAT(NINPZ)
58:      integer KSFBD(NZDA)
59:      integer IBSDA(3,NBODY)
60:      integer KREG(NINPZ)
61:      character*12 KREGN(NINPZ)
62:      integer KREGI(NINPZ)
63:      character*12 IZNAM(*)
64:      character*(LNAME) TNAMS(*)
65: C

```

```

66:      integer IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1)
67:      integer INUCT(*)
68:      real DENST(*)
69:      character*16 NCIDI(NUC), NUCID(NUC)
70:      real*8 TEMPN(NUC)
71: C
72: C ... local data
73: C
74:      character*6 TYPE
75:      integer IMTMP(300)
76: C
77: C-----
78: C
79:      call RWIND( IMC )
80:      call RWIND( IUG1 )
81:      call RWIND( IUG2 )
82:      call RWIND( IUB )
83: C
84:      IERR = 0
85: C
86: C==== Check bodies =====
87: C
88:      KS = 0
89:      MS = 0
90:      do IB = 1, NBODY
91:         J2 = NSURF
92:         if ( IB.lt.NBODY ) J2 = IBSDA(1,IB+1) - 1
93: C ... body ID ...
94:         IBID = IBSDA(2,IB)
95: C ... number of MCNP surfaces to compose this body
96:         NMCS = 0
97:         KS0 = KS
98: C
99:         do J = IBSDA(1,IB), J2
100:            K = IPSDA(1,J)
101:            IST = NINT(SDA(K))
102: C
103: C=== slab ==
104: C
105:            if ( IST.eq.1 ) then
106: C
107: C ... PX s
108: C
109:            if ( ABS(SDA(K+1)).eq.1.0D0 ) then
110:               D1 = SDA(K+4) /SDA(K+1)
111:               D2 = SDA(K+5) /SDA(K+1)
112: C
113:               NMCS = NMCS + 1
114:               IMTMP(NMCS) = KS + NMCS
115:               MS = MS + 1
116:               write(IUB) 'PX ', IBID, MS
117:               write(IUB) MIN(D1,D2)
118: C
119:               NMCS = NMCS + 1
120:               IMTMP(NMCS) = -(KS+NMCS)
121:               MS = MS + 1
122:               write(IUB) 'PX ', IBID, MS
123:               write(IUB) MAX(D1,D2)
124: C
125: C ... PY s
126: C
127:            else if ( ABS(SDA(K+2)).eq.1.0D0 ) then
128:               D1 = SDA(K+4) /SDA(K+2)
129:               D2 = SDA(K+5) /SDA(K+2)
130: C

```

src/cgview/mcnout.f

```

131:      NMCS      = NMCS + 1
132:      IMTMP(NMCS) = KS + NMCS
133:      MS        = MS + 1
134:      write(IUB) 'PY ', IBID, MS
135:      write(IUB) MIN(D1,D2)
136:
137:      NMCS      = NMCS + 1
138:      IMTMP(NMCS) = -(KS+NMCS)
139:      MS        = MS + 1
140:      write(IUB) 'PY ', IBID, MS
141:      write(IUB) MAX(D1,D2)
142: C
143: C      .. PZ s
144: C
145:      else if ( ABS(SDA(K+3)).eq.1.0D0 ) then
146:      D1      = SDA(K+4) / SDA(K+3)
147:      D2      = SDA(K+5) / SDA(K+3)
148:
149:      NMCS      = NMCS + 1
150:      IMTMP(NMCS) = KS + NMCS
151:      MS        = MS + 1
152:      write(IUB) 'PZ ', IBID, MS
153:      write(IUB) MIN(D1,D2)
154:
155:      NMCS      = NMCS + 1
156:      IMTMP(NMCS) = -(KS+NMCS)
157:      MS        = MS + 1
158:      write(IUB) 'PZ ', IBID, MS
159:      write(IUB) MAX(D1,D2)
160: C
161: C      ... P s
162: C
163:      else
164:      NMCS      = NMCS + 1
165:      IMTMP(NMCS) = KS + NMCS
166:      MS        = MS + 4
167:      write(IUB) 'P ', IBID, MS
168:      write(IUB) SDA(K+1), SDA(K+2), SDA(K+3),
169:      &      MIN(SDA(K+4),SDA(K+5))
170:      NMCS      = NMCS + 1
171:      IMTMP(NMCS) = -(KS+NMCS)
172:      MS        = MS + 4
173:      write(IUB) 'P ', IBID, MS
174:      write(IUB) SDA(K+1), SDA(K+2), SDA(K+3),
175:      &      MAX(SDA(K+4),SDA(K+5))
176:      end if
177: C
178: C      === sphere == ( SO,SX,SY & SZ are not used for simplicity :- )
179: C
180:      else if ( IST.eq.2 ) then
181:      NMCS      = NMCS + 1
182:      IMTMP(NMCS) = -(KS+NMCS)
183:      MS        = MS + 4
184:      write(IUB) 'S ', IBID, MS
185:      write(IUB) SDA(K+1), SDA(K+2), SDA(K+3), SQRT(SDA(K+4))
186: C
187: C      === cylinder ==
188: C
189:      else if ( IST.eq.3 ) then
190: C      ... a point on axis
191:      DX      = SDA(K+1)
192:      DY      = SDA(K+2)
193:      DZ      = SDA(K+3)
194: C      ... axis vector
195:      DA      = SDA(K+4)

```

```

196:      DB      = SDA(K+5)
197:      DC      = SDA(K+6)
198: C
199:      DR      = SQRT(SDA(K+7))
200: C
201:      NMCS      = NMCS + 1
202:      IMTMP(NMCS) = -(KS+NMCS)
203: C
204: C      ... C/X or CX
205: C
206:      if ( ABS(DA).eq.1.0D0 ) then
207:      if ( DY.eq.0.0D0.and.DZ.eq.0.0D0 ) then
208:      MS      = MS + 1
209:      write(IUB) 'CX ', IBID, MS
210:      write(IUB) DR
211:      else
212:      MS      = MS + 3
213:      write(IUB) 'C/X ', IBID, MS
214:      write(IUB) DY, DZ, DR
215:      end if
216: C
217: C      ... C/Y or CY
218: C
219:      else if ( ABS(DB).eq.1.0D0 ) then
220:      if ( DZ.eq.0.0D0.and.DX.eq.0.0D0 ) then
221:      MS      = MS + 1
222:      write(IUB) 'CY ', IBID, MS
223:      write(IUB) DR
224:      else
225:      MS      = MS + 3
226:      write(IUB) 'C/Y ', IBID, MS
227:      write(IUB) DX, DZ, DR
228:      end if
229: C
230: C      ... C/Z or CZ
231: C
232:      else if ( ABS(DC).eq.1.0D0 ) then
233:      if ( DX.eq.0.0D0.and.DY.eq.0.0D0 ) then
234:      MS      = MS + 1
235:      write(IUB) 'CZ ', IBID, MS
236:      write(IUB) DR
237:      else
238:      MS      = MS + 3
239:      write(IUB) 'C/Z ', IBID, MS
240:      write(IUB) DX, DY, DR
241:      end if
242:      else
243: C
244: C      ... general cylinder --> GQ
245: C
246: C      (X-A)**2 + (Y-B)**2 + (Z-C)**2
247: C      - ( (X-A)*K + (Y-B)*L + (Z-C)*M )**2 - R**2 < 0
248: C
249:      HX2      = 1.0D0 - DA**2
250:      HY2      = 1.0D0 - DB**2
251:      HZ2      = 1.0D0 - DC**2
252:      HXY      = -2.0D0*DA*DB
253:      HYZ      = -2.0D0*DB*DC
254:      HZX      = -2.0D0*DC*DA
255:      HX        = 2.0D0*(-DX*HX2+DA*(DY*DB+DZ*DC))
256:      HY        = 2.0D0*(-DY*HY2+DB*(DZ*DC+DX*DA))
257:      HZ        = 2.0D0*(-DZ*HZ2+DC*(DX*DA+DY*DB))
258:      HC        = HX2*DX**2 + HY2*DY**2 + HZ2*DZ**2 - 2.0D0*
259:      &      (DX*DY*DA*DB+DY*DZ*DB*DC+DZ*DX*DC*DA)
260:      &      - SDA(K+7)

```

src/cgview/mcnout.f

```

261:          MS      = MS + 10
262:          write(IUB) 'GQ ', IBID, MS
263:          write(IUB) HX2, HY2, HZ2, HXY, HYZ, HZX, HX, HY, HZ,
264:          &          HC
265:          end if
266: C
267: C === general quadratic ==
268: C
269:          else if ( IST.eq.4 ) then
270:              NMCS      = NMCS + 1
271:              IMTMP(NMCS) = -(KS+NMCS)
272:              MS      = MS + 10
273:              write(IUB) 'GQ ', IBID, MS
274:              write(IUB) (SDA(K+1),I=1,10)
275: C
276: C === half space ==
277: C
278:          else if ( IST.eq.5 ) then
279:              NMCS      = NMCS + 1
280:              IMTMP(NMCS) = -(KS+NMCS)
281:              MS      = MS + 4
282:              write(IUB) 'P ', IBID, MS
283:              write(IUB) SDA(K+1), SDA(K+2), SDA(K+3), SDA(K+4)
284: C
285: C === torus ==
286: C
287:          else if ( IST.eq.6 ) then
288:              write(IPR,*) 'xxx(MCNOUT) torus is not supported now.'
289:              IERR      = 1
290:              go to 100
291:          end if
292:      end do
293: C
294:          KS      = KS + NMCS
295: C
296:          write(IPR,*) '+ body ', IBID, ' has ', NMCS, ' MCNP surfaces'
297: C
298:          write(IUG1,'(2i7)') IBID, NMCS
299:          write(IUG1,'(10i7)') (IMTMP(I),I=1,NMCS)
300:      end do
301: C
302:          write(IPR,*) '+ number of MCNP surfaces created temporary ',KS
303: C
304: C == temporary arrays
305: C
306:          call GTLAST( LAST0 )
307:          call KEEP( 'kbsm', LKBSM, NBODY+1, 'I4', LAST, 0 )
308:          call KEEP( 'ibsm', LIBSM, KS, 'I4', LAST, 0 )
309:          call KEEP( 'ksmcn', LKSMCN, 2*(KS+1), 'I4', LAST, 0 )
310:          call KEEP( 'smcn', LSMCN, MS, 'R8', LAST, 0 )
311: C
312:          if ( LSIZE(LAST).gt.LIMIT ) then
313:              write(IPR,*) 'xxx(MCNOUT) insufficient memory (limit=', LIMIT,
314:              &          ', required=', LAST, ' )'
315:              call STLAST( ' ', LAST0, LAST )
316:              IERR      = 1
317:              return
318:          end if
319: C
320:          call GTLASTC( LASTC0 )
321:          call KEEP( 'tmcn', LTMCN, KS, 'C4', LASTC, 0 )
322:          if ( LASTC.gt.LIMITC ) then
323:              write(IPR,*) 'xxx(MCNOUT) insufficient character memory ',
324:              &          ' (limitc=', LIMITC, ' , required=', LASTC, ' )'
325:              call STLASTC( ' ', LASTC0, LAST )

```

```

326:          IERR      = 1
327:          return
328:      end if
329: C
330: C ===== restore body/surface data from scratch I/O units
331: C          and check MCNP surfaces not to define identical surfaces.
332: C
333:          call MCNSCK( IPR, IUG1, IUB, NBODY, KS, MS, A(LKBSM), A(LIBSM),
334:          &          A(LKSMCN), A(LSMCN), CHA(LTMCN) )
335: C
336: C ===== output cell & surface data
337: C
338:          call MCNCSO( IMC, IPR, IUG1, IUB, IUG2, KS, MS, TITLE, CODE,
339:          &          A(LKBSM),
340:          &          A(LIBSM), A(LKSMCN), A(LSMCN), CHA(LTMCN), KINPZ, KZDA,
341:          &          KZAA, SDA, KZMAT, KMAT, KSFBD, IZNAM, IBSDA, KREG, KREGN,
342:          &          KREGI, TNAMS,
343:          &          IDMAT, MNUC, INUCT, DENST, LPDEN,
344:          &          NCIDI, NUCID, TEMPN
345:          & )
346: C
347: C
348: C ... free memory
349: C
350: 100 continue
351:          call STLAST( ' ', LAST0, LAST )
352:          call STLASTC( ' ', LASTC0, LAST )
353: C
354:          return
355:      end

```

src/cgview/mcnsck.f

```

1: C
2: C=====
3: C
4:       subroutine MCNSCK( IPR,   IUG1,  IUB,   NBODY, KS,    MS,    KBSM,
5:       &                  IBSM,   KSMCN, SMCN,  TMCN )
6: C
7:       implicit real*8(D)
8: C
9:       integer KBSM(NBODY+1), IBSM(KS)
10:      integer KSMCN(2,KS+1)
11:      real*8 SMCN(MS)
12: C
13:      character*4 TMCN(KS)
14: C
15: C ... external function
16: C
17:      logical LSAME
18:      external LSAME
19: C
20: C-----
21: C
22: C ---- load body shape by MCNP surface ----
23: C
24:       call RWIND( IUG1 )
25: C
26:       KBSM(1) = 1
27:       do IB = 1, NBODY
28:         read(IUG1,'(2i7)') IBB,NMCS
29:       check %%%
30:       write(*,*) '%%%' ,IBB, NMCS, KBSM(IB), NBODY
31:       c %%%%%%%%%
32:       KBSM(IB+1) = KBSM(IB) + NMCS
33:       read(IUG1,'(10i7)') (IBSM(I),I=KBSM(IB),KBSM(IB+1)-1)
34:       check %%%
35:       c write(*,*) '%%%' ibsm ', (IBSM(I),I=KBSM(IB),KBSM(IB+1)-1)
36:       c %%%%%%%%%
37:       end do
38: C
39: C ---- load surface type string and surface data
40: C
41:       call RWIND( IUB )
42: C
43:       KSMCN(1,1) = 1
44:       do IS = 1, KS
45:         KK      = KSMCN(1,IS)
46:         read(IUB) TMCN(IS), IBID, IM
47:       check %%%
48:       c write(*,*) '%%%' , IS, TMCN(IS), IBID, IM, KK
49:       c %%%%%%%%%
50:       read(IUB) (SMCN(I),I=KK,IM)
51:       KSMCN(1,IS+1) = IM + 1
52:       end do
53: C
54: C ===== MCNP surface check not to use identical surface more than once
55: C       and output MCNP surface card on I/O unit IUG2
56: C
57: C ksmcn(2,*) : surface # of identical one (or the surface itself)
58: C       Has opposite sign with the identical surface
59: C       if current surface is complement of the surface.
60: C
61:       KSMCN(2,1) = 1
62:       do IS = 2, KS
63: C
64:         KSMCN(2,IS) = IS
65:         do JS = 1, IS-1

```

```

66:         if ( TMCN(JS).eq.TMCN(IS) ) then
67:           KKI      = KSMCN(1,IS)
68:           KKJ      = KSMCN(1,JS)
69: C
70: C ... two planes ...
71: C
72:           if ( TMCN(IS).eq.'PX' .or. TMCN(IS).eq.'PY'
73:             & .or. TMCN(IS).eq.'PZ' ) then
74:             if ( LSAME(SMCN(KKI),SMCN(KKJ)) ) KSMCN(2,IS) = JS
75:           else if ( TMCN(IS).eq.'P' ) then
76:             DD      = SMCN(KKI)*SMCN(KKJ) + SMCN(KKI+1)*
77:             &       SMCN(KKJ+1) + SMCN(KKI+2)*SMCN(KKJ+2)
78:             if ( LSAME(DD,1.0D0)
79:               & .and.LSAME(SMCN(KKI+3),SMCN(KKJ+3)) ) then
80:               KSMCN(2,IS) = JS
81:             else if ( LSAME(DD,-1.0D0)
82:               & .and.LSAME(SMCN(KKI+3),-SMCN(KKJ+3)) ) then
83:               KSMCN(2,IS) = -JS
84:             end if
85: C
86: C ... two cylinders ...
87: C
88:           else if ( TMCN(IS).eq.'CX' .or. TMCN(IS).eq.'CY'
89:             & .or. TMCN(IS).eq.'CZ' ) then
90:             if ( LSAME(SMCN(KKI),SMCN(KKJ)) ) KSMCN(2,IS) = JS
91: C
92:           else if ( TMCN(IS)(1:2).eq.'C/' ) then
93:             if ( LSAME(SMCN(KKI),SMCN(KKJ))
94:               & .and.LSAME(SMCN(KKI+1),SMCN(KKJ+1))
95:               & .and.LSAME(SMCN(KKI+2),SMCN(KKJ+2)) ) KSMCN(2,IS) =
96:             & JS
97:           else
98:             end if
99: C
100: C == other planes ...
101: C
102:           end if
103:           if ( KSMCN(2,IS).ne. IS ) goto 100
104:         end do
105:       100 continue
106:       end do
107: C
108:       return
109:       end
110: C
111: C=====
112: C .... two floating value is identical or not ???
113: C
114:       function LSAME(X,Y)
115: C
116:       logical LSAME
117:       real*8 X, Y
118: C
119:       real*8 D
120: C
121:       LSAME = .true.
122:       if ( ABS(X).gt.1.0D-5 .or. ABS(Y).gt.1.0D-5 ) then
123:         D = ABS(X) + ABS(Y)
124:         if ( ABS(X-Y)/D.gt.1.0D-8 ) LSAME = .false.
125:       end if
126:       return
127:       end

```

src/cgview/newp.f

```

1:      subroutine NEWP( IPEN )
2:      C
3:      C -----
4:      C change line color (foreground color)
5:      C
6:      C -----
7:      C/#IF MS_VISUAL
8:      C
9:      C ... This interface block is for CUTIL & MS-Visual tools. ...
10:     C
11:     * interface
12:     *      subroutine SETLWD( III1 )
13:     *          integer      III1
14:     *CDEC$      ATTRIBUTES C :: SETLWD
15:     *CDEC$      ATTRIBUTES REFERENCE :: III1
16:     *      end subroutine
17:     *      subroutine SETRGBNUM( III2, III3, III4, III5 )
18:     *          integer      III2, III3, III4, III5
19:     *CDEC$      ATTRIBUTES C :: SETRGBNUM
20:     *CDEC$      ATTRIBUTES REFERENCE :: III2, III3, III4, III5
21:     *      end subroutine
22:     *      subroutine SETCLRS( IP )
23:     *          integer      IP
24:     *CDEC$      ATTRIBUTES C :: SETCLRS
25:     *CDEC$      ATTRIBUTES REFERENCE :: IP
26:     *      end subroutine
27:     *      end interface
28:     C/#ENDIF
29:     C
30:     CC/#IF PIFFLIB
31:     integer R(0:16), G(0:16), B(0:16)
32:     C
33:     C      data R /0, 200, 255, 0, 0, 255, 255, 0, 127, 127, 0, 0, 127, 127,
34:     C      &      0, 190, 127/
35:     C      data G /0, 200, 0, 255, 0, 255, 0, 255, 127, 0, 127, 0, 127, 0,
36:     C      &      127, 255, 127/
37:     C      data B /0, 200, 0, 0, 255, 0, 255, 255, 127, 0, 0, 127, 0, 127,
38:     C      &      127, 190, 255/
39:     C
40:     C      data r/ 0,255,255, 0, 0,255,255, 0,127,127,
41:     C      @      0, 0,127,127, 0,190,127/
42:     C      data g/ 0,255, 0,255, 0,255, 0,255,127, 0,
43:     C      @      127, 64,127, 0,127,255,127/
44:     C      data b/ 0,255, 0, 0,255, 0,255,255,127, 0,
45:     C      @      0,127, 0,127,127,190,255/
46:     C
47:     C
48:     C ... color map (has been used until Apr 1999)
49:     C
50:     C      data ( R(I),G(I),B(I),I=0,16 ) /
51:     C      0  0,      0,      0,
52:     C      1  200,     200,     200,
53:     C      2  255,     0,      0,
54:     C      3  0,      255,     0,
55:     C      4  0,      0,      255,
56:     C      5  255,     255,     0,
57:     C      6  255,     0,      255,
58:     C      7  0,      255,     255,
59:     C      8  127,     127,     127,
60:     C      9  127,     0,      0,
61:     C      A  0,      127,     0,
62:     C      B  0,      0,      127,
63:     C      C  127,     127,     0,
64:     C      D  127,     0,      127,
65:     C      E  0,      127,     127,
66:     C      F  190,     255,     190,
67:     C      G  127,     127,     255
68:     C      & /
69:     C
70:     C      data ( R(I),G(I),B(I),I=0,16 ) /
71:     C      0  0,      0,      0,
72:     C      1  200,     200,     200,
73:     C      2  255,     0,      0,
74:     C      3  0,      255,     0,
75:     C      4  0,      0,      255,
76:     C      5  255,     255,     0,
77:     C      6  255,     0,      255,
78:     C      7  0,      255,     255,
79:     C      8  127,     127,     127,
80:     C      9  200,     127,     0,
81:     C      A  0,      200,     127,
82:     C      B  127,     0,      200,
83:     C      C  127,     200,     64,
84:     C      D  0,      127,     200,
85:     C      E  200,     0,      127,
86:     C      F  190,     255,     190,
87:     C      G  127,     127,     255
88:     C      & /
89:     C
90:     C      data init /0/
91:     C
92:     C .... register all colors ....
93:     C
94:     C      if( init.eq.0 ) then
95:     C          do i=0,16
96:     C              call SETRGBNUM( i, R(i),G(i),B(i))
97:     C              call SETLWD( 1 )
98:     C          end do
99:     C          init = 1
100:    C      end if
101:    C
102:    C      if ( IPEN.eq.0 .or. IPEN.eq.15 .or. IPEN.eq.16 ) then
103:    C          IP      = IPEN
104:    C      else
105:    C          IP      = MOD(IPEN-1,14) + 1
106:    C      end if
107:    C      Ccccc call SETRGB( R(IP), G(IP), B(IP) )
108:    C      call SETCLRS( IP )
109:    C      CC/#ENDIF
110:    C      return
111:    C      end

```

src/cgview/nucfis.f

```
1:      subroutine NUCFIS( E,      NS,      INUC,  IRAND, IERR,  RWK )
2: C=<MVP>=====
3: C PURPOSE: DUMMY SUBROUTINE IN SLICE
4: C CALLED IN : SYSSRC
5: C-----
6: C
7: C   ARGUMENTS ( I= INPUT, O = OUTPUT, W = WORK )
8: C
9: C O   E(NS) : ENERGY SAMPLED
10: C I   NS : NUMBER OF PARTICLES WHOSE ENERGY E IS SAMPLED
11: C I   INUC : NUCLIDE #
12: C O   IERR : ERROR CODE #
13: C W   RWK(NS,5) : WORKING ARRAY
14: C=====
15: C
16:      return
17:      end
```


src/cgview/option.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine OPTION( MLIMIT )
3: C=====
4: C PURPOSE: (CGVIEW) READ OPTION PARAMETERS
5: C CALLED IN:  INTRO
6: C=====
7:   include '../shared/INC/_ARRAY'
8:   include '../shared/INC/_IUNIT'
9: CCCC include '../shared/INC/_TASKDT'
10:  include 'INC/_KPIDS'
11:  include 'INC/_FLAGS'
12: C
13: C
14: C --- PRINT OUT OF COMMON /FLAGS/ IS SUPPRESSED. -----
15: C
16:   character LINE*72, CH*1, PFLAG*72
17:   parameter( MAXPRT = 20 )
18:   integer IT(MAXPRT)
19: C
20: CCC include 'INC/_REACC'
21: C
22: C/#IF DYNAMIC
23: C
24: C ... reset limit size of array A in common /ARRAY/ here,
25: C   when dynamic allocation of memory is available.
26: C
27:   LIMIT = MLIMIT
28: C/#ENDIF
29: C
30: C-----
31: C ..... READ NEW RECORD .....
32: C-----
33: C
34: 100 call FREADB( LINE, NLEN, IEND )
35: Cccc IF(IEND.EQ.1) GOTO 2000
36:   if ( IEND.ne.0 ) go to 150
37:   JJ = 1
38: C-----
39: C ..... A 'NO-' ON HEAD MEANS NOT TO USE THIS OPTION .....
40: C-----
41:   IS = 1
42:   IE = NLEN
43:   if ( NLEN.gt.3.and.LINE(IS:IS+2).eq.'NO-' ) then
44:     JJ = 0
45:     IS = IS + 3
46:   end if
47: C   IEI = IE + 1
48: C
49:   NHYP = 0
50:   do 110 I = IS, IE
51:     if ( LINE(I:I).eq.'-' ) NHYP = NHYP + 1
52: 110 continue
53: C
54: C-----
55: C ..... CHECK OPTIONS WITH NUMERICAL DATA ... XXXX-YYY(N)
56: C-----
57: C*****IBR = INDEX(LINE(IS:IE),'(')
58: C ... GET NEXT NON-BLANK CHARACTER IN THE INPUT FILE ....
59: C   .... ( DETECT CHARACTER OTHER THAN ' ' )
60: C
61:   IBR = 0
62:   call FPROBE( IBR, ' ', CH )
63:   if ( IBR.ne.0.and.CH.ne.'(' ) IBR = 0
64: C
65: C   .... IBRU IS SET TO 0 IF NUMERICAL DATA ARE USED ....

```

```

66: C
67:   IBRU = 1
68: C
69: C**** IF(IBR.NE.0) IEI = IBR
70: C
71: C-----
72: C   SET FLAGS
73: C-----
74: C
75: C ===== RESTART
76: C
77: *   if ( NHYP.eq.0.and.IMATCH('REST*',LINE(IS:IE)).eq.1 ) then
78: *     JREST = JJ
79: C
80: C ===== NEUTRON
81: C
82:   if ( NHYP.eq.0.and.IMATCH('NEUT*',LINE(IS:IE)).eq.1 ) then
83:     JNEUT = JJ
84: C
85: C ===== PHOTON
86: C
87:   else if ( NHYP.eq.0.and.IMATCH('PHOT*',LINE(IS:IE)).eq.1 ) then
88:     JPHOT = JJ
89: C
90:   if ( IBR.ne.0 ) then
91:     PFLAG = ' '
92: 120   call CHREAD( ' ', PFLAG, ' ', IJ, ITERM, IEND )
93:   if ( IEND.ne.0 ) then
94:     write(IPR,*) 'XXX UNEXPECTED END OF DATA IN OPTION BLOCK'
95:     call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
96:     stop 888
97:   end if
98: C
99:   if ( IJ.ne.0 ) then
100:     IBRU = 0
101: C
102: C   ..... "GAMMA RAY" MODEL FOR PHOTON .....
103: C
104:   if ( PFLAG(1:1).eq.'G' ) then
105:     JGAMM = 1
106:   else if ( PFLAG(1:1).eq.'P' ) then
107:     JGAMM = 0
108:   end if
109:   end if
110:   if ( ITERM.eq.0 ) go to 120
111: else
112:   JGAMM = 0
113: end if
114: end if
115: C
116: C ===== PARTICLE
117: C
118:   if ( NHYP.eq.0.and.IMATCH('PART*',LINE(IS:IE)).eq.1 ) then
119: C
120:   if ( IBR.ne.0 ) then
121:     PFLAG = ' '
122: 130   call CHREAD( ' ', PFLAG, ' ', IJ, ITERM, IEND )
123:   if ( IEND.ne.0 ) then
124:     write(IMG,*)
125:     & 'XXX Unexpected end of data in option block'
126:     call PRSTOP( 1, 'UNEXPECTED END OF DATA.' )
127:     stop 888
128:   end if
129: C
130:   if ( IJ.ne.0 ) then

```

src/cgview/option.f

```

131:      IBRU      = 0
132: C
133: C      ..... check supported particle or not .....
134: C
135:      call KPSYMB( PFLAG(:IJ), '>', IKPID, 0 )
136: C
137:      if ( IKPID.eq.0 ) then
138:      write(IMG,*)
139: &      'XXX unsupported particle in PARTICLE()',
140: &      ' <', PFLAG(:IJ), '>'
141:      call CNTERR( 'FATAL' )
142:      else
143:      JKPAB(IKPID) = JJ
144:      if ( IKPID.eq.KPNEUT ) then
145:      JNEUT = JJ
146:      end if
147:      if ( IKPID.eq.KPPHOT ) then
148:      JPHOT = JJ
149:      end if
150:      end if
151:      end if
152:      if ( ITERM.eq.0 ) go to 130
153:      end if
154:      end if
155: C
156: C ===== BREMSSTRAHLUNG
157: C
158: *      else if ( NHYP.eq.0.and.IMATCH('BREM*',LINE(IS:IE)).eq.1 ) then
159: *      JBREM = JJ
160: C
161: C ===== IMAGINARY-PARTICLE
162: C
163: *      else if ( NHYP.eq.1.and.IMATCH('IMAG*-PART*',LINE(IS:IE)).eq.1 )
164: *      &      then
165: *      JIMAG = JJ
166: C
167: C ===== TIME-DEPENDENT
168: C
169:      if ( NHYP.eq.1.and.IMATCH('TIME*-DEPE*',LINE(IS:IE)).eq.1 ) then
170:      JTIME = JJ
171: C
172: C ===== DELTA-TRACKING
173: C
174:      else if ( NHYP.eq.1.and.IMATCH('DELT*-TRAC*',LINE(IS:IE)).eq.1 )
175:      &      then
176:      JDELT = JJ
177: C
178: C ===== RELATIVE-WEIGHT
179: C
180: *      else if ( NHYP.eq.1.and.IMATCH('RELA*-WEIG*',LINE(IS:IE)).eq.1 )
181: *      &      then
182: *      JRWVR = JJ
183: C
184: C ===== FISSION
185: C
186: *      else if ( NHYP.eq.0.and.IMATCH('FISS*',LINE(IS:IE)).eq.1 ) then
187: *      JFISS = JJ
188: C
189: C ===== EIGEN-VALUE
190: C
191: *      else if ( NHYP.eq.1.and.IMATCH('EIGE*-VALU*',LINE(IS:IE)).eq.1 )
192: *      &      then
193: *      JEIGN = JJ
194: *      JFIXD = 1 - JJ
195: C

```

```

196: C ===== FIXED-SOURCE
197: C
198: *      else if ( NHYP.eq.1.and.IMATCH('FIXE*-SOUR*',LINE(IS:IE)).eq.1 )
199: *      &      then
200: *      JFIXD = JJ
201: *      JEIGN = 1 - JJ
202: C
203: C ===== RUSSIAN-ROULETTE
204: C
205: *      else if ( NHYP.eq.1.and.IMATCH('RUSS*-ROUL*',LINE(IS:IE)).eq.1 )
206: *      &      then
207: *      JRRLT = JJ
208: *      if ( JRRLT.eq.1 ) JWWND = 0
209: C
210: C ===== IMPORTANCE
211: C
212: *      else if ( NHYP.eq.0.and.IMATCH('IMPO*',LINE(IS:IE)).eq.1 ) then
213: *      JIMPT = JJ
214: *      if ( JIMPT.eq.1 ) JWWND = 0
215: C
216: C ===== WEIGHT-WINDOW
217: C
218: *      else if ( NHYP.eq.1.and.IMATCH('WEIG*-WIND*',LINE(IS:IE)).eq.1 )
219: *      &      then
220: *      JWWND = JJ
221: *      if ( JWWND.eq.1 ) JIMPT = 0
222: *      if ( JWWND.eq.1 ) JRRLT = 0
223: C
224: C ===== PATH-STRETCHING
225: C
226: *      else if ( NHYP.eq.1.and.IMATCH('PATH*-STRE*',LINE(IS:IE)).eq.1 )
227: *      &      then
228: *      JPSTR = JJ
229: C
230: C ===== FORCED-COLLISION
231: C
232: *      else if ( NHYP.eq.1.and.IMATCH('FORC*-COLL*',LINE(IS:IE)).eq.1 )
233: *      &      then
234: *      JFCOL = JJ
235: C
236: C ===== COLLISION-LESS
237: C
238: *      else if ( NHYP.eq.1.and.IMATCH('COLL*-LESS',LINE(IS:IE)).eq.1 )
239: *      &      then
240: *      JNCOL = JJ
241: C
242: C ===== SOURCE-BIASING
243: C
244: *      else if ( NHYP.eq.1.and.IMATCH('SOUR*-BIAS*',LINE(IS:IE)).eq.1 )
245: *      &      then
246: *      JBIAS = JJ
247: C
248: C ===== RESPONSE
249: C
250: *      else if ( NHYP.eq.0.and.IMATCH('RESP*',LINE(IS:IE)).eq.1 ) then
251: *      JRESP = JJ
252: C
253: C ===== MONITOR
254: C
255: *      else if ( NHYP.eq.0.and.IMATCH('MONI*',LINE(IS:IE)).eq.1 ) then
256: *      JMNTR = JJ
257: C
258: C ===== VP-MONITOR
259: C
260: *      else if ( NHYP.eq.1.and.IMATCH('VP-MONI*',LINE(IS:IE)).eq.1 ) then

```

src/cgview/option.f

```

261: *      JVMNT      = JJ
262: C
263: C ===== DEBUG-PRINT
264: C
265: C      else if ( NHYP.eq.1.and.IMATCH('DEBU*-PRIN*',LINE(IS:IE)).eq.1 )
266: C      &      then
267: C      if ( JJ.eq.0 ) then
268: C      do 140 I = 1, MAXJDB
269: C      JDEBG(I) = 0
270: 140 continue
271: C      else if ( IBR.ne.0 ) then
272: C      call I4READ( 'DEBUG-PRINT', JDEBG, NA, -MAXJDB, IERR )
273: C      if ( IERR.ne.0 ) then
274: C      write(IPR,*)
275: C      &      'XXX INVALID PARAMETER FOR OPTION "DEBUG-PRINT"'
276: C      call CNTERR( 'FATAL' )
277: C      end if
278: C      IBRU = 0
279: C      else
280: C      JDEBG(1) = JJ
281: C      end if
282: C
283: C ===== VPP-SPECIFIC
284: C
285: C      else if ( NHYP.eq.1.and.IMATCH('VPP-SPEC*',LINE(IS:IE)).eq.1 )
286: C      &      then
287: C      if ( JJ.eq.0 ) then
288: C      do 132 I = 1, MXJVP
289: C      JVPPS(I) = 0
290: 132 continue
291: C      else if ( IBR.ne.0 ) then
292: C      call I4READ( 'VPP-SPECIFIC', JVPPS, NA, -MXJVP, IERR )
293: C      if ( IERR.ne.0 ) then
294: C      write(IPR,*)
295: C      &      'XXX INVALID PARAMETER FOR OPTION "VPP-SPECIFIC".'
296: C      call CNTERR( 'FATAL' )
297: C      end if
298: C      IBRU = 0
299: C      else
300: C      JVPPS(1) = JJ
301: C      end if
302: C
303: C ===== PICTURE
304: C
305: C      else if ( NHYP.eq.0.and.IMATCH('PICT*',LINE(IS:IE)).eq.1 ) then
306: C      JPICT = JJ
307: C
308: C ===== ALL-ZONE (1988/2/25)
309: C OR EVENT-SELECTION (1989/2/14)
310: C
311: C      else if ( NHYP.eq.1.and.IMATCH('ALL-ZONE',LINE(IS:IE)).eq.1
312: C      &      .or. IMATCH('EVEN*-SELE*',LINE(IS:IE)).eq.1 ) then
313: C      JALLZ = JJ
314: C      JONEZ = 1 - JJ
315: C
316: C ===== ONE-ZONE (1988/2/25)
317: C OR ZONE-SELECTION (1989/2/14)
318: C
319: C      else if ( NHYP.eq.1.and.IMATCH('ONE-ZONE',LINE(IS:IE)).eq.1
320: C      &      .or. IMATCH('ZONE-SELE*',LINE(IS:IE)).eq.1 ) then
321: C      JONEZ = JJ
322: C      JALLZ = 1 - JJ
323: C
324: C ===== LATTICE (1988/10/19)
325: C

```

```

326: C      else if ( NHYP.eq.0.and.IMATCH('LATT*',LINE(IS:IE)).eq.1 ) then
327: C      JLATT = JJ
328: C
329: C ===== REPEATED-GEOMETRY
330: C ( ALIAS OF LATTICE )
331: C      else if ( NHYP.eq.1.and.IMATCH('REPE*-GEOM*',LINE(IS:IE)).eq.1 )
332: C      &      then
333: C      JLATT = JJ
334: C
335: C ===== FREE-LATTICE-FRAME (1988/12/28)
336: C
337: C      else if ( NHYP.eq.2
338: C      &      .and.IMATCH('FREE*-LATT*-FRAM*',LINE(IS:IE)).eq.1 ) then
339: C      JFISX = JJ
340: C
341: C ===== TALLY-LATTICE (1992/3/18)
342: C
343: C      else if ( NHYP.eq.1.and.IMATCH('TALL*-LATT*',LINE(IS:IE)).eq.1 )
344: C      &      then
345: C      JTLLT = JJ
346: C
347: C ===== UNIVERSE-DEPENDENT-TALLY
348: C FRAME-DEPENDENT-TALLY
349: C ( ALIAS OF TALLY-LATTICE )
350: C
351: C      else if ( NHYP.eq.2.and.IMATCH('UNIV*-DEPE*-TALL*',LINE(IS:IE))
352: C      &      .eq.1 .or. IMATCH('FRAM*-DEPE*-TALL*',LINE(IS:IE)).eq.1 )
353: C      &      then
354: C      JTLLT = JJ
355: C
356: C ===== FLUX-PRINT (1989/08/22)
357: C
358: C      else if ( NHYP.eq.1.and.IMATCH('FLUX-PRIN*',LINE(IS:IE)).eq.1 )
359: C      &      then
360: C      JFPRT = JJ
361: C
362: C ===== EDIT-BY-TRACK-LENGTH
363: C
364: C      else if ( NHYP.eq.3
365: C      &      .and.IMATCH('EDIT-BY-TRAC*-LENG*',LINE(IS:IE)).eq.1 ) then
366: C      JRTTR = JJ
367: C      JRTCL = 1 - JJ
368: C
369: C ===== EDIT-BY-COLLISION
370: C
371: C      else if ( NHYP.eq.2.and.IMATCH('EDIT-BY-COLL*',LINE(IS:IE)).eq.1 )
372: C      &      then
373: C      JRTCL = JJ
374: C      JRTTR = 1 - JJ
375: C
376: C ===== EDIT-MICROSCOPIC-DATA
377: C
378: C      else if ( NHYP.eq.2.and.IMATCH('EDIT-MICR*-DATA',LINE(IS:IE)).eq.1
379: C      &      ) then
380: C
381: C      if ( IBR.ne.0 ) then
382: C
383: C      call I4READ( 'EDIT-MICROSCOPIC-DATA', IT, NA, -1, IERR )
384: C      if ( IERR.ne.0 ) then
385: C      write(IPR,*)
386: C      &      'XXX INVALID PARAMETER FOR OPTION "EDIT-MICRO-DATA".'
387: C      call CNTERR( 'FATAL' )
388: C      end if
389: C      IBRU = 0
390: C

```

src/cgview/option.f

```

391: *      if ( NA.gt.0 ) then
392: *          IJJ = IT(1)
393: *          do 140 I = 1, 8
394: *              JMICE(I) = IJJ/(10**(8-I))
395: *              IJJ = IJJ - 10**(8-I)*JMICE(I)
396: *              JMICE(I) = JMICE(I)*JJ
397: *              if ( JMICE(I).gt.4 ) JMICE(I) = 0
398: * 140      continue
399: *          end if
400: *      end if
401: C
402: C ===== EDIT-MACROSCOPIC-DATA
403: C
404: *      else if ( NHYP.eq.2.and.IMATCH('EDIT-MACR*-DATA',LINE(IS:IE)).eq.1
405: *      &      ) then
406: *
407: *          if ( IBR.ne.0 ) then
408: *
409: *              call I4READ( 'EDIT-MACROSCOPIC-DATA', IT, NA, -1, IERR )
410: *
411: *              if ( IERR.ne.0 ) then
412: *                  write(IPR,*)
413: *                  &      'XXX INVALID PARAMETER FOR OPTION "EDIT-MACROSCOPIC-DATA"'
414: *                  call CNTERR( 'FATAL' )
415: *              end if
416: *              IBRU = 0
417: C
418: *              if ( NA.gt.0 ) then
419: *                  IJJ = IT(1)
420: *                  do 150 I = 1, 8
421: *                      JMACE(I) = IJJ/(10**(8-I))
422: *                      IJJ = IJJ - 10**(8-I)*JMACE(I)
423: *                      JMACE(I) = JMACE(I)*JJ
424: *                      if ( JMACE(I).gt.4 ) JMACE(I) = 0
425: * 150      continue
426: *                  end if
427: *              end if
428: C
429: C ===== PRINT-SUPPRESS
430: C
431: *      else if ( NHYP.eq.1.and.JJ.eq.1
432: *      &      .and.IMATCH('PRIN*-SUPP*',LINE(IS:IE)).eq.1 ) then
433: *          if ( IBR.ne.0 ) then
434: C
435: *              call I4READ( 'PRINT-SUPPRESS', IT, NA, -MAXPRT, IERR )
436: *              if ( IERR.ne.0 ) then
437: *                  write(IPR, '/(lx,a/)' )
438: *                  &      'XXX INVALID PARAMETER FOR OPTION "PRINT-SUPPRESS"'
439: *                  call CNTERR( 'FATAL' )
440: *              end if
441: *              IBRU = 0
442: C
443: *              if ( NA.gt.0 ) then
444: *                  do 160 I = 1, NA
445: *                      if ( IT(I).ge.1.and.IT(I).le.MAXPRT ) JPRTS(IT(I)) = 1
446: * 160      continue
447: *                  end if
448: *              end if
449: C
450: C ===== RUN-MODE[ ( mode ) ]
451: C
452: *      else if ( NHYP.eq.1.and.JJ.eq.1.and.'RUN-MODE*'.eq.LINE(IS:IE) )
453: *      &      then
454: *          if ( IBR.ne.0 ) then
455: C

```

```

456: *      call I4READ( 'RUN-MODE', IT, NA, -1, IERR )
457: *      if ( IERR.ne.0 ) then
458: *          write(IPR,*)
459: *          &      'XXX INVALID PARAMETER FOR OPTION "RUN-MODE"'
460: *          call CNTERR( 'FATAL' )
461: *      end if
462: *      IBRU = 0
463: *      if ( NA.gt.0 ) JRUNM = IT(1)
464: *      else
465: *          JRUNM = 0
466: *      end if
467: C
468: C << OPTION FOR DEBUGGING >> CHECKK FOR /ARRAY/
469: C
470: C ===== MEMORY-CHECK
471: C
472: *      else if ( NHYP.eq.1.and.JJ.eq.1
473: *      &      .and.IMATCH('MEMO*-CHEC*',LINE(IS:IE)).eq.1 ) then
474: *          JMCHK = JJ
475: C
476: C .... SIZE OF DYNAMICALLY ALLOCATED MEMORY ...
477: C
478: C ===== DYNAMIC-MEMORY
479: C
480: *      else if ( NHYP.eq.1.and.JJ.eq.1
481: *      &      .and.IMATCH('DYNA*-MEMO*',LINE(IS:IE)).eq.1 ) then
482: *          if ( IBR.ne.0 ) then
483: *              IT(1) = 0
484: *              IT(2) = 0
485: *              call I4READ( 'DYNAMIC-MEMORY', IT, NA, -MAXPRT, IERR )
486: *              if ( IERR.ne.0 ) then
487: *                  write(IPR,*)
488: *                  &      'XXX INVALID PARAMETER FOR OPTION "DYNAMIC-MEMORY"'
489: *                  call CNTERR( 'FATAL' )
490: *              end if
491: *              IBRU = 0
492: C/#IF DYNAMIC
493: C/# IF PARA( CRAY SX* )
494: C      if ( NA.gt.0 ) LIMIT = IT(1)
495: C      if ( NA.gt.1 ) LIMITL = IT(2)
496: C/# ELSE
497: C      if ( NA.gt.0 ) LIMIT = IT(1) + IT(2)
498: C/# ENDIF
499: C/#ELSE
500: C      write(IPR,7000) LIMIT
501: C      format(/lx,'<<MESSAGE>> DYNAMIC-MEMORY option is not ',
502: C      &      'effective for this installation of the code.'/
503: C      &      ' When you want to change memory size, please',
504: C      &      ' change "MAXMEM" parameter in the AALOC routine and',
505: C      &      ' remake load-module.'/lx,' (current limit = ',I10,
506: C      &      ' words)')
507: C
508: *      CALL CNTERR('MESSAGE')
509: C
510: C/# IF .NOT.PARA
511: C      if ( NA.gt.0.and.IT(1)+IT(2).gt.LIMIT ) then
512: C          write(IPR,*) '!!! YOUR DEMAND FOR DYNAMIC MEMORY',
513: C          &      ' EXCEEDS FIXED LIMIT OF ', LIMIT, ' WORDS OF ',
514: C          &      ' CURRENT LOAD-MODULE.'
515: C      end if
516: C/# ELSE
517: C      if ( NA.eq.1 ) then
518: C          write(IPR,*) ' IT MAY CAUSE ERROR STOP BEFORE ',
519: C          &      ' STARTING HISTORIES.'
520: C          call CNTERR( 'WARNING' )

```

src/cgview/option.f

```

521: c          end if
522: C/# ENDIF
523: C/#ENDIF
524: *          else
525: *            write(IPR,*)
526: *            &      'XXX NO SIZE-PARAMETER GIVEN FOR DYNAMIC-MEMORY OPTION.'
527: *            call CNTERR( 'FATAL' )
528: C
529: *          end if
530: C
531: C      .... NUMBER OF TASKS FOR MULTI-TASKING MODE ...
532: C
533: C      ===== MULTI-TASK
534: C
535: *      else if ( NHYP.eq.1.and.JJ.eq.1
536: *      &      .and.IMATCH('MULT*-TASK*',LINE(IS:IE)).eq.1 ) then
537: *      if ( IBR.ne.0 ) then
538: *      IT(1) = 0
539: *      call I4READ( 'MULTI-TASK', IT, NA, -MAXPRT, IERR )
540: *      if ( IERR.ne.0 ) then
541: *      write(IPR,*)
542: *      &      'XXX INVALID PARAMETER FOR OPTION "MULTI-TASKING"'
543: *      call CNTERR( 'FATAL' )
544: *      end if
545: *      if ( NA.gt.0 ) NTASK = IT(1)
546: *      IBRU = 0
547: *      else
548: *      write(IPR,*)
549: *      &      'XXX NO TASK-NUMBER GIVEN FOR MULTI-TASK OPTION.'
550: *      call CNTERR( 'FATAL' )
551: *      end if
552: C
553: C      .... spawn "tasker" in multitask mode if necessary (only in PVM)
554: C
555: C      ===== SPAWN-TASKER
556: C
557: *      else if ( NHYP.eq.1.and.IMATCH('SPAW*-TASK*',LINE(IS:IE)).eq.1 )
558: *      &      then
559: *      JTSKR = JJ
560: C
561: C      ... run multitasking mode in a way to obtain reproducible result.
562: C
563: C      ===== REPRODUCIBLE-RUN
564: C
565: *      else if ( NHYP.eq.1.and.IMATCH('REPR*-RUN',LINE(IS:IE)).eq.1 )
566: *      &      then
567: *      JREPR = JJ
568: C
569: C      ... save source data on file
570: C      (currently data of last batch on eigenvalue problem)
571: C
572: C      ===== SOURCE-OUTPUT
573: C
574: *      else if ( NHYP.eq.1.and.IMATCH('SOUR*-OUT*',LINE(IS:IE)).eq.1 )
575: *      &      then
576: *      JOSRC = JJ
577: C
578: C      ... Open scratch I/O units if they are not opened ...
579: C
580: C      ===== OPEN-SCRATCH
581: C
582: *      else if ( NHYP.eq.1.and.IMATCH('OPEN-SCRA*',LINE(IS:IE)).eq.1 )
583: *      &      then
584: *      JSCRT = JJ
585: C

```

```

586: C      ... output restart file. Restart file is output by default,
587: C      so this option is used to suppress restart file output.
588: C
589: C      ===== RESTART-FILE
590: C
591: *      else if ( NHYP.eq.1.and.IMATCH('REST*-FILE',LINE(IS:IE)).eq.1 )
592: *      &      then
593: *      JRSTF = JJ
594: *      else
595: C
596: C      ..... ERROR .....
597: C
598: C      write(IPR,'(/1X,'XXX INVALID OPTION: ',A)') LINE(IS:IE)
599: C      call CNTERR( 'FATAL' )
600: C
601: *      else
602: *      write(IPR,'(/' This option is ignored in CGVIEW : ',A)')
603: *      &      LINE(IS:IE)
604: *      end if
605: C
606: C      ..... SKIP UNUSED NUMERIC PARAMETERS ....
607: C
608: *      if ( IBR.ne.0.and.IBRU.ne.0 ) then
609: *      call DMREAD( ' ', NA, NB, IRET )
610: *      write(IPR,7000) LINE(IS:IE)
611: *      7000 format(/1X,'!!! NUMERIC PARAMETERS ARE GIVEN BUT NOT',
612: *      &      ' EFFECTIVE FOR OPTION <',A,'>')
613: *      call CNTERR( 'WARNING' )
614: *      end if
615: C
616: C      go to 100
617: C
618: C      150 continue
619: C
620: C      ..... ERROR OR WARNING .....
621: C      AND ADJUSTMENT OF FLAGS
622: C
623: C      ..... PRINT OUT OPTION PARAMETERS
624: C
625: C      write(IPR,7020)
626: C      7020 format(/1X,'      OPTION PARAMETERS      (1/0 = ON/OFF) '///
627: C      &      FLAG      KEY-WORD      ON/OFF      INCOMPATIBLE OPTIONS '//
628: C      &      '-----'
629: C      &      '-----')
630: C      7040 format(1X,2X,A5,3X,A20,2X,I2,5X,5(:('(',A5,') '))
631: C      7060 format(1X,2X,A5,3X,A20,2X,I2,5X,5(:('(',A5,') '))
632: C
633: C      write(IPR,7060) 'JREST', 'RESTART', ' ', JREST
634: C      write(IPR,7060) 'JRSTF', 'RESTART-FILE', ' ', JRSTF
635: C      write(IPR,7060) 'JRUNM', 'RUN_MODE', ' ', JRUNM
636: C      write(IPR,7060) 'JNEUT', 'NEUTRON', ' ', JNEUT, 'JIMAG'
637: C      write(IPR,7060) 'JPHOT', 'PHOTON', ' ', JPHOT, 'JIMAG'
638: C      if ( JPHOT.ne.0 ) write(IPR,7060) 'JGAMM', ' GAMMA-RAY MODEL', ' ',
639: C      &      JGAMM
640: C      write(IPR,7060) 'JBREM', 'BREMSSTRAHLUNG', ' ', JBREM
641: C      write(IPR,7060) 'JIMAG', 'IMAGINARY-PARTICLE', ' ', JIMAG, 'JNEUT',
642: C      &      'JPHOT'
643: C      write(IPR,7060) 'JNCOL', 'COLLISION-LESS', ' ', JNCOL
644: C      write(IPR,7060) 'JTIME', 'TIME-DEPENDENT', ' ', JTIME
645: C      WRITE(IPR,7200) 'JDELT', 'DELTA-TRACKING', ' ', JDELT
646: C      write(IPR,7060) 'JEIGN', 'EIGEN-VALUE', ' ', JEIGN, 'JFIXD'
647: C      write(IPR,7060) 'JFIXD', 'FIXED-SOURCE', ' ', JFIXD, 'JEIGN'
648: C      write(IPR,7060) 'JFISS', 'FISSION PROBLEM', ' ', JFISS
649: C      write(IPR,7060) 'JOSRC', 'SOURCE-OUTPUT', ' ', JOSRC
650: C      write(IPR,7040) 'JRRLT', 'RUSSIAN-ROULETTE', ' ', JRRLT, 'JWWND'

```

src/cgview/option.f

```

651:      write(IPR,7040) 'JIMPT', 'IMPORTANCE', 'JIMPT', 'JWWND'
652:      write(IPR,7040) 'JWWND', 'WEIGHT-WINDOW', 'JWWND', 'JIMPT',
653:      & 'JRRLT'
654:      write(IPR,7040) 'JPSTR', 'PATH-STRETCHING', 'JPSTR'
655: c      WRITE(IPR,7060) 'JRWVR', 'RELATIVE-WEIGHT', 'JRWVR'
656: CCC      WRITE(IPR,7200) 'JFCOL', 'FORCED-COLLISION', 'JFCOL'
657: CCC      WRITE(IPR,7200) 'JBIAS', 'SOURCE-BIAS', 'JBIAS'
658: c      write(IPR,7060) 'JRESP', 'RESPONSE', 'JRESP'
659: c      write(IPR,7060) 'JMNTR', 'MONITOR', 'JMNTR'
660: c      write(IPR,7060) 'JVMNT', 'VP-MONITOR', 'JVMNT'
661:      do 160 I = 1, MAXJDB
662:      if ( I.eq.1 .or. JDEBG(I).ne.0 ) then
663:      LINE = ' '
664:      write(LINE,(' 'DEB-PRINT(' ',i2,' '))) I
665:      call CCOMP( LINE, LEN(LINE), LINE, IERR )
666:      write(IPR,7040) 'JDEBG', LINE(:20), JDEBG(I)
667:      end if
668:      160 continue
669: c      write(IPR,7060) 'JSCRT', 'OPEN-SCRATCH', 'JSCRT'
670: c      write(IPR,7060) 'JALLZ', 'EVENT-SELECTION', 'JALLZ', 'JONEZ'
671: c      write(IPR,7060) 'JONEZ', 'ZONE-SELECTION', 'JONEZ', 'JALLZ'
672: c      write(IPR,7060) 'JREPR', 'REPRODUCIBLE-RUN', 'JREPR'
673: c      write(IPR,7060) 'JTSKR', 'SPAWN-TASKR', 'JTSKR'
674: c      write(IPR,7040) 'JLATT', 'LATTICE', 'JLATT'
675: c      write(IPR,7060) 'JFPRT', 'FLUX-PRINT', 'JFPRT'
676: c      write(IPR,7040) 'JTLLT', 'TALLY-LATTICE', 'JTLLT'
677: c      write(IPR,7060) 'JRTTR', 'EDIT-BY-TRACK-LENGTH', 'JRTTR', 'JRTCL'
678: c      write(IPR,7060) 'JRTCL', 'EDIT-BY-COLLISION', 'JRTCL', 'JRTTR'
679: c      write(IPR,7100) 'JMICE', 'EDIT-MICROSCOPIC-DATA'
680: c      write(IPR,7120) (REAC(L),JMICE(L),L=1,8)
681: c      write(IPR,7100) 'JMACE', 'EDIT-MACROSCOPIC-DATA'
682: c      write(IPR,7120) (REAC(L),JMACE(L),L=1,8)
683:      7080 format(/1X,2X,A5,3X,A32)
684:      7100 format(1X,'<',A12,'>',I2)
685: c      write(IPR,*) ' '
686: c      if ( JPRTS(1).eq.1 ) write(IPR,*)
687: c      & ' * PRINT OF GROUP-WISE FLUX IS SUPPRESSED.'
688: c      if ( JPRTS(2).eq.1 ) write(IPR,*)
689: c      & ' * PRINT OF ANGULAR FLUX IS SUPPRESSED.'
690: c      if ( JPRTS(3).eq.1 ) write(IPR,*)
691: c      & ' * PRINT OF TIME-DEPENDENT FLUX IS SUPPRESSED.'
692: c      if ( JPRTS(4).eq.1 ) write(IPR,*)
693: c      & ' * PRINT OF RESPONSE IS SUPPRESSED.'
694: c      if ( JPRTS(5).eq.1 ) write(IPR,*)
695: c      & ' * PRINT OF TIME-DEPENDENT RESPONSE ', ' IS SUPPRESSED.'
696: c      if ( JPRTS(6).eq.1 ) write(IPR,*)
697: c      & ' * PRINT OF SOURCE INFORMATION #1 ', ' IS SUPPRESSED.'
698: c      if ( JPRTS(7).eq.1 ) write(IPR,*)
699: c      & ' * PRINT OF SOURCE INFORMATION #2 ', ' IS SUPPRESSED.'
700: c      if ( JPRTS(17).eq.1 ) write(IPR,*)
701: c      & ' * PRINT OF MICROSCOPIC REACTION RATE', ' IS SUPPRESSED.'
702: c      if ( JPRTS(18).eq.1 ) write(IPR,*)
703: c      & ' * PRINT OF MICROSCOPIC CROSS SECTION', ' IS SUPPRESSED.'
704: c      if ( JPRTS(19).eq.1 ) write(IPR,*)
705: c      & ' * PRINT OF MACROSCOPIC REACTION RATE', ' IS SUPPRESSED.'
706: c      if ( JPRTS(20).eq.1 ) write(IPR,*)
707: c      & ' * PRINT OF MACROSCOPIC CROSS SECTION', ' IS SUPPRESSED.'
708: c
709:      return
710:      end

```

src/cgview/ovlpmsg.f

```

1:      subroutine OVLPMSG( IOW, JTLLT, IMCELL, MZONE, KZ,
2:      &                    DDD2, MPLIMIT,
3:      &                    NINPZ, NZONE, NSURF, NSUZON, NEST, NSPACE,
4:      &                    IPSDA, IBSDA, IZNAM, KREGN, KZMAT, KZREG, KINPZ,
5:      &                    ISUSP, INX,
6:      &                    IBP, IFI, DIST, IGMCK, IPGMCK, XGMCK,
7:      &                    NIPGMCK, MIPGMCK,
8:      &                    XXX, YYY, ZZZ, AAA, BBB, CCC, KLSF,
9:      &                    LEVL, IBSPC,
10:     &                    IBNK, KIBNK, MIBNK, NBANK )
11: C=====
12: C Purpose: Output overlapping region/material message
13: C
14: C <<how to check zone overlap>>
15: C
16: C DIST(*) is the distance to boundary of a zone into which a particle
17: C just entered and which has different material or region# (zone KZ
18: C in a picture below), and if DIST(*) has a value larger than a small
19: C distance (given as DDD2), the newly entered point is not on boundary
20: C of new zone KZ and the zone MZONE and zone KZ are overlapping.
21: C
22: C
23: C
24: C
25: C
26: C
27: C
28: C
29: C
30: C
31: C
32: C
33: C
34: C <<limitation>>
35: C
36: C If shape of overlapping two zones are identical or one of them
37: C is covered wholly by the other one, the method described above fails
38: C to detect overlapping zones.
39: C
40: C=====
41:      real*8 DDD2
42:      integer KZMAT(NZONE), KZREG(NZONE), KINPZ(NZONE)
43:      integer IPSDA(3,NSURF+1)
44:      integer IBSDA(3,*)
45:      integer ISUSP(NSUZON,NSPACE)
46:      character*12 IZNAM(NINPZ)
47:      character*12 KREGN(NINPZ)
48: C
49:      integer IGMCK(NZONE), IPGMCK(3,MIPGMCK)
50:      real*8 XGMCK(4,MIPGMCK)
51: C
52:      real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
53:      &      CCC(NBANK)
54:      integer KLSF(NBANK)
55:      integer LEVL(NBANK), IBSPC(NBANK,0:NEST)
56:      integer IBNK(NBANK,*), KIBNK(0:MIBNK)
57: C
58:      integer IBP(INX)
59:      logical IFI(INX)
60:      real DIST(INX)
61: C
62: C .... local data
63: C
64:      character*3 BTYP
65:      character*16 CRGM, CRG

```

```

66:
67: C
68: C-----
69: C
70: Check %%%
71: c      write(*,*) '%%OVLP MZONE ',mzone,' KZ ',KZ
72: C
73: C      if ( KZMAT(KZ).ne.KZMAT(MZONE) .or. KZREG(KZ).ne.KZREG(MZONE) )
74: C &      then
75: C
76:      KZI = KINPZ(KZ)
77:      KZM = KINPZ(MZONE)
78:      if ( KZI.ne.KZM ) then
79: C
80:      KK = 0
81: C
82:      IZ1 = MIN(KZ,MZONE)
83:      IZ2 = MAX(KZ,MZONE)
84:      MM = 0
85: C
86: C      ... CRGM/CRG : region # or region name (KREGN)
87: C
88: C      For cell display mode (IMCELL>0) on TALLY-LATTICE
89: C      option, KZREG and ISUSP may not give collect
90: C      region #, so regin name string KREGN is used for
91: C      region overlap check.
92: C
93:      CRGM = ' '
94:      CRG = ' '
95:      if ( IMCELL.eq.0 ) then
96:      if ( JTLLT.eq.0 ) then
97:      write(CRGM,'(I8)') KZREG(MZONE)
98:      write(CRG,'(I8)') KZREG(KZ)
99:      else
100: C      write(CRGM,'(I8)')
101: C &      ISUSP(KZREG(MZONE),IBSPC(IB,LEVL(IB)))
102: C      write(CRG,'(I8)')
103: C &      ISUSP(KZREG(KZ),IBSPC(IB,LEVL(IB)))
104: C      CRGM = '<'//KREGN(KZM)//>'
105: C      CRG = '<'//KREGN(KZI)//>'
106:      end if
107:      else
108:      CRGM = '<'//KREGN(KZM)//>'
109:      CRG = '<'//KREGN(KZI)//>'
110:      end if
111:      ILM = max(1,ICLEN2(CRGM))
112:      ILI = max(1,ICLEN2(CRG))
113: C
114:      do 140 I = 1, INX
115: C
116:      IB = IBP(I)
117: C
118:      if ( IMCELL.eq.0 .and. JTLLT.ne.0 ) then
119: C      write(CRGM,'(I8)')
120: C &      ISUSP(KZREG(MZONE),IBSPC(IB,LEVL(IB)))
121: C      write(CRG,'(I8)')
122: C &      ISUSP(KZREG(KZ),IBSPC(IB,LEVL(IB)))
123: C      end if
124: C
125: C      ... KOV oberlapping flag : 0 = input-zone only
126: C      1 = region only
127: C      2 = material only
128: C      3 = material and region
129: C
130: C      KOV = 0

```

src/cgview/ovlpmsg.f

```

131:         if ( IFI(I).and.IBNK(IB,KIBNK(5)).gt.0.and.DIST(I).gt.DDD2 )
132:         &         then
133: C
134:             IGMCK(MZONE) = IGMCK(MZONE) + 1
135:             IGMCK(KZ) = IGMCK(KZ) + 1
136:             KK = KK + 1
137: C
138:             if ( IGMCK(MZONE).lt.MPLIMIT.or.MPLIMIT.lt.0 ) then
139: C
140:                 if ( KZMAT(KZ).ne.KZMAT(MZONE)
141: &                 .and.CRG.ne.CRGM ) then
142:                     write(IOW,7040) MZONE, KZM,
143: &                     IZNAM(KZM) (:ICLEN2(IZNAM(KZM))),
144: &                     KZMAT(MZONE), CRGM(:ILM), KZ, KZI,
145: &                     IZNAM(KZI) (:ICLEN2(IZNAM(KZI))),
146: &                     KZMAT(KZ), CRG(:ILI)
147: &                     KOV = 3
148:                 else if ( KZMAT(KZ).ne.KZMAT(MZONE) ) then
149:                     write(IOW,7000) MZONE, KZM,
150: &                     IZNAM(KZM) (:ICLEN2(IZNAM(KZM))),
151: &                     KZMAT(MZONE), KZ, KZI,
152: &                     IZNAM(KZI) (:ICLEN2(IZNAM(KZI))), KZMAT(KZ)
153: &                     KOV = 2
154:                 else if ( CRG.ne.CRGM ) then
155:                     write(IOW,7020) MZONE, KZM,
156: &                     IZNAM(KZM) (:ICLEN2(IZNAM(KZM))),
157: &                     CRGM(:ILM), KZ, KZI,
158: &                     IZNAM(KZI) (:ICLEN2(IZNAM(KZI))), CRG(:ILI)
159: &                     KOV = 1
160:                 else
161:                     write(IOW,7060) MZONE, KZM,
162: &                     IZNAM(KZM) (:ICLEN2(IZNAM(KZM))), KZ, KZI,
163: &                     IZNAM(KZI) (:ICLEN2(IZNAM(KZI))),
164: &                     KZMAT(MZONE), CRGM(:ILM)
165: &                     KOV = 0
166:                 end if
167: C
168: C
169: C
170:             ... print body
171:             if ( KLSF(IBP(I)).ne.0 ) then
172:                 do IS = 1, NSURF
173:                     if ( IPSDA(1,IS).eq.KLSF(IBP(I)) ) then
174:                         go to 100
175:                     end if
176:                 end do
177:                 go to 110
178:                 continue
179:                 BTYP = ' '
180:                 call TYPBOD( BTYP, '<', IBSDA(3,IPSDA(3,IS)) )
181:                 write(IOW,*) ' On the surface of body ',
182: &                 IPSDA(2,IS), '(', BTYP, ')'
183:                 continue
184:             end if
185: C
186:             write(IOW,7080) MZONE, KZ, XXX(IBP(I)), YYY(IBP(I)),
187: &             ZZZ(IBP(I)), AAA(IBP(I)), BBB(IBP(I)),
188: &             CCC(IBP(I)), DIST(I)
189:         else
190:             if ( IGMCK(MZONE).eq.MPLIMIT ) then
191:                 write(IOW,*) 'XXX more than ', MPLIMIT,
192: &                 ' point of overlapping material and/or region',
193: &                 ' for zone(inp-zone) ', MZONE, '(', KZM, ')'
194:             end if
195:             if ( KZMAT(KZ).ne.KZMAT(MZONE)

```

```

196:         &             .and.CRG.ne.CRGM ) then
197:             KOV = 3
198:         else if ( KZMAT(KZ).ne.KZMAT(MZONE) ) then
199:             KOV = 2
200:         else if ( CRG.ne.CRGM ) then
201:             KOV = 1
202:         else
203:             KOV = 0
204:         end if
205:         end if
206: C
207: C
208: C
209:         if ( MM.eq.0 ) then
210:             do 120 M = 1, MIN(NIPGMCK,MIPGMCK)
211:                 if ( IPGMCK(1,M).eq.IZ1.and.IPGMCK(2,M).eq.IZ2 )
212:                     &                     then
213:                         go to 130
214:                     end if
215:                 continue
216:                 NIPGMCK = NIPGMCK + 1
217:                 if ( NIPGMCK.le.MIPGMCK ) then
218:                     M = NIPGMCK
219:                     IPGMCK(1,NIPGMCK) = IZ1
220:                     IPGMCK(2,NIPGMCK) = IZ2
221:                 else
222:                     M = 0
223:                 end if
224:                 continue
225:                 MM = M
226:             end if
227: C
228:         if ( MM.ne.0 ) then
229:             IPGMCK(3,MM) = KOV
230:             if ( ABS(DIST(I)).gt.XGMCK(4,MM) ) then
231:                 XGMCK(4,MM) = DIST(I)
232:                 XGMCK(1,MM) = XXX(IBP(I))
233:                 XGMCK(2,MM) = YYY(IBP(I))
234:                 XGMCK(3,MM) = ZZZ(IBP(I))
235:             end if
236:             end if
237: C
238:             end if
239:             continue
240:             end if
241: C
242:             7000 format(1X,'xx Overlapping material?'/3X,'from zone(izone)',I5,
243: &             '(',I5,')<',A,'> mat',I6/3X,'to zone(izone)',I5,('(',I5,
244: &             ')<',A,'> mat',I6)
245:             7020 format(1X,'xx Overlapping region?'/3X,'from zone(izone)',I5,('(',
246: &             I5,')<',A,'> region',a/3X,'to zone(izone)',I5,('(',I5,
247: &             ')<',A,'> region',a)
248:             7040 format(1X,'xx Overlapping material&region?'/3X,
249: &             'from zone(izone)',I5,('(',I5,')<',A,'> mat',I6,' region',
250: &             a/3X,'to zone(izone)',I5,('(',I5,')<',A,'> mat',I6,
251: &             ' region',a)
252:             7060 format(1X,'!! Overlapping input-zone?'/3X,'from zone(izone)',I5,
253: &             '(',I5,')<',A,'>', ' to zone(izone)',I5,('(',I5,')<',A,'>',
254: &             3X,'mat',I6,' region',a)
255:             7080 format(3X,' ZONE',I5,' XYZ=',I6,3E13.5/3X,' DIR=',3E12.4,
256: &             ' DIST=',E11.4)
257:             return
258:         end

```


src/cgview/pablo1.f

```

1: *VOCL TOTAL,SCALAR
2:   subroutine PABLO1( A,      H,      CHA,  LIMIT, LIMITL,LIMITC,CODE,
3:     &                ICHK1, PAPER, TITLE, ZONCLR,IZNCLR,REGCLR,
4:     &                MATCLR,IGMCK, XGMCK, NDEFSRC )
5: C=====
6: C PURPOSE: Get commands from viewer controller.
7: C CALLED IN: CENTER
8: C CALLS:  NONE
9: C=====
10: C
11:   implicit real*8(D)
12: C
13: C/#IF MS_VISUAL
14: C
15: C ... This interface block is for CUTIL & MS-Visual tools. ...
16: C
17: *   interface
18: *     subroutine GETWINID( IWID )
19: *       integer    IWID
20: *CDEC$  ATTRIBUTES C :: GETWINID
21: *CDEC$  ATTRIBUTES REFERENCE :: IWID
22: *     end subroutine
23: *   subroutine FTSYSTEM( ICODE, CWORK, LLL )
24: *     integer ICODE, LLL
25: *     character*256 CWORK
26: *CDEC$  ATTRIBUTES C :: FTSYSTEM
27: *CDEC$  ATTRIBUTES REFERENCE :: ICODE, CWORK, LLL
28: *     end subroutine
29: *   subroutine PLOT( RRR1, RRR2, III1 )
30: *     integer    III1
31: *     real       RRR1, RRR2
32: *CDEC$  ATTRIBUTES C :: PLOT
33: *CDEC$  ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
34: *     end subroutine
35: *   end interface
36: C/#ENDIF
37: C
38: C   COMMON /ARRAY/ LIMIT,IDUM,A(1)
39: C
40:   real A(*)
41:   real H(*)
42:   character*4 CHA(*)
43:
44:   character*72 TITLE(2)
45:
46:   character*(*) CODE
47:
48:   include 'INC/_MVIEW'
49:   include 'INC/_MVGEO'
50: C
51:   include 'INC/_KPIDS'
52:
53: CCC  INCLUDE '../shared/INC/_ARRAY'
54:   include '../shared/INC/_SIZES'
55:   include '../shared/INC/_PGEOM'
56:   include '../shared/INC/_PTLSP'
57:   include 'INC/_FLAGS'
58:   include '../shared/INC/_IOUNIT'
59: C
60:   include '../mvp/INC/_CXSEC'
61: C
62:   real*8 PAPER(30)
63:   integer ZONCLR(NZONE)
64:   integer IZNCLR(NINPZ)
65:   integer REGCLR(NREG)

```

```

66:   integer MATCLR(0:MXMAT)
67: C
68:   integer IGMCK(NZONE)
69:   real*8 XGMCK(4,MIPGMCK)
70:   integer NDEFSRC(NINPZ)
71: C
72: C ..... local data .....
73: C
74:   character LINE1*72
75:   character VNM*16
76:   character*1 CH
77:   real*8 XI(6)
78: C
79:   character*256 CWORK
80:   character*256 CWORK2
81: C
82:   parameter( MXITMP = 1000 )
83:   integer ITEMP(MXITMP)
84: C
85:   data IDENT /0/
86:   data IPNUM /1/
87:   data IWON  /1/
88: C
89: C-----
90: C
91:   call RIUNIT( IOIN )
92: C
93:   ICALL = 0
94:   IEND = 0
95:   IRR = 0
96: C
97: C ... number of colors used for zone/region/material
98: C
99:   NCC = 14
100: C
101: C .... on the first call to this routine
102: C
103:   if ( IDENT.eq.0 ) then
104: C
105:     IDENT = 1
106: C
107: C ... WAITON / WAITOFF status
108:   IWON = 1
109: C
110:   call LABEL( IPR, 'CGVIEW specific input starts here!!' )
111: C
112: C ... set prompt string and echoback flag
113: C
114:   call GTLSET( 'CGVIEW>', 1, 2 )
115: C
116:   do 100 I = 1, NCGVDB
117:     JCGVDB(I) = 0
118:   100 continue
119: C
120:   TITLGR = TITLE(1)
121: C
122:   LEVEL = NEST
123:   STYLE = 0
124:   STYLE2 = 0
125:   SPTYP = 2
126:   IKIND = 1
127: C
128:   MSGSLST = 20
129: C
130: C ...set default colormap for zone, region and material

```

src/cgview/pablo1.f

```

131: C
132:      call ZCLR00( ZONCLR, NZONE, NCC )
133:      call IZCLR00( IZNCLR, NINPZ, NCC )
134:      call RCLR00( REGCLR, NREG, NCC )
135:      call MCLR00( MATCLR, MXMAT, NCC )
136: C
137: C      ... CLDAT is color map for lattice region etc.
138: C
139:      do 110 I = 0, MXCLDAT
140: CCCCCCCC CLDAT(I) = MOD(I,NCC) + 1
141:      CLDAT(I) = NCC-MOD(I,NCC)
142: 110 continue
143: C
144: C      ... default scanning mode is +X/-X alternative only
145: C
146:      JSCANX = 0
147:      JSCANY = 3
148:      JSCANX0 = JSCANX
149:      JSCANY0 = JSCANY
150: C
151:      JGFLOOD = 0
152:      NGFLOOD = 3000
153:      MFLOOD = 0
154: C
155:      INTREFRESH = 200
156: C
157:      JZREVERSE = 0
158:      JZREVERSE = 2
159: C
160:      JGMCK = 1
161:      DOVERLAP = 2.0D0
162: C
163:      MPLIMIT = 10
164: C
165:      OFRAME = 1
166:      OFRAME = 0
167:      OVERSION = 1
168:      OLOSTSYM = 0
169:      JHYAXIS = 1
170: C
171:      MCELL = 0
172:      IMCELL = 0
173: C
174: C      .... initial values for zoom, move commands
175: C
176:      ZMSAVE = 2.0D0
177:      XMVSAVE = 0.5D0
178:      YMVSAVE = 0.5D0
179:      ZMVSARE = 0.1D0
180: C
181:      ROTSAVE = 45.0D0
182:      XROTSARE = 45.0D0
183:      YROTSARE = 45.0D0
184: C
185:      IPNUM = 1
186: C
187:      do 10 I = 1, MPRNCOM
188:          if (PRNCOM(I).ne.' ' ) goto 20
189: 10 continue
190:      PRNCOM(1) = 'xwd -id %w > %f_%p.xwd'
191: 20 continue
192: C
193: C      .... set default value of PAPER & XMAX ...
194: C
195:      PAPER(10) = -1.0D0

```

```

196:      MXYZ = 1
197: C
198:      call GPCELL( IPR, MCELL, IMCELL, MXYZ, PAPER, XMAX, YMAX,
199:      & GRANGE, DINF, A(LIBSDA), A(LGRBOD), NBODY, A(LIDCEL),
200:      & A(LIPCEL), A(LICTYP), NCELL, A(LKZAA), NZONE, A(LKZDA),
201:      & A(LKSFBD), NZDA )
202: C
203:      call SAVEPARAM( IPR, 0, TITLGR, PAPER, XMAX, YMAX, JSCANX,
204:      & JSCANY, SPTYP, STYLE, STYLE2, LEVEL, INTREFRESH,
205:      & OFRAME, OVERSION, OLOSTSYM, MCELL, IMCELL, JZREVERSE,
206:      & GRANGE, JGFLOOD )
207: C
208:      end if
209: C
210: C      .... scanning mode may be changed to different one from input
211: C      when STYLE = 0.
212: C      Recover them here.
213: C
214:      JSCANX = JSCANX0
215:      JSCANY = JSCANY0
216: C
217:      NERR = 0
218:      IDSAVE = -1
219:      JFWBK = 0
220: C
221: C-----
222: C
223: C      ... FREADL is special routine for CGVIEW which allows lowercase
224: C      alphabets in VNM ...
225: C
226: C
227: 120 call FREADL( VNM, NLEN, IEND )
228: C
229: 110 call FREADS( VNM, NLEN, IEND )
230: C
231: C      ... physical end of file ...
232: C      if ( IEND.eq.-1 ) go to 260
233: C
234: C      ... "end of file" by "/" ...
235: C
236: C      if ( IEND.eq.1 ) go to 220
237: C
238: C      ... check "/" on the tail of VNM
239: C
240:      JJEND = 0
241:      if ( VNM(NLEN:NLEN).eq.'/' ) then
242:          JJEND = 1
243:          NLEN = NLEN - 1
244:      end if
245: C
246: C      ... change vname to capital letters (July 1996 M.Sasaki)
247: C
248:      call CAPITAL( VNM(:NLEN), NLEN )
249: C
250: C      ... check if input is VNM( data ) or only VNM
251: C
252:      IBR = 0
253:      call FPROBE( IBR, ' ', CH )
254:      if ( IBR.ne.0.and.CH.ne.'( ' ) IBR = 0
255: C
256: C-----
257: C      PAPER(1:3) : coordinates of a corner
258: C      PAPER(4:6) : drawing axis vector 1 - X axis (unit vector)
259: C      PAPER(7:9) : drawing axis vector 2 - Y axis (unit vector)
260: C

```

src/cgview/pablo1.f

```

261:         if ( VNM(:NLEN).eq.'PAPER'.and.IBR.ne.0 ) then
262:
263:             ICALL    = 1
264:             NB       = 10
265:             call R8READ( VNM, PAPER, NA, -NB, IERR )
266:             if ( NA.ne.NB ) IRR = IRR + 1
267: C
268:             write(IPR,'(1x,a/(3x,3e12.5))')
269:             &         ' PAPER (corner-xyz : X-axis : Y-axis ) ',
270:             &         (PAPER(J),J=1,NB)
271: C
272:             if ( PAPER(10).eq.0.0D0 ) then
273:                 write(IPR,*) 'XXX PAPER(10) (lines per cm) is zero!!'
274:                 NERR    = NERR + 1
275:             end if
276:             DDX    = SQRT(PAPER(4)**2+PAPER(5)**2+PAPER(6)**2)
277:             if ( DDX.eq.0.0D0 ) then
278:                 write(IPR,*) 'XXX PAPER(4:6) (X-vector) is zero vector!!'
279:                 NERR    = NERR + 1
280:             else
281:                 PAPER(4) = PAPER(4) /DDX
282:                 PAPER(5) = PAPER(5) /DDX
283:                 PAPER(6) = PAPER(6) /DDX
284:             end if
285:
286:             DDY    = SQRT(PAPER(7)**2+PAPER(8)**2+PAPER(9)**2)
287:             if ( DDY.eq.0.0D0 ) then
288:                 write(IPR,*) 'XXX PAPER(7:9) (Y-vector) is zero vector!!'
289:                 NERR    = NERR + 1
290:             else
291:                 PAPER(7) = PAPER(7) /DDY
292:                 PAPER(8) = PAPER(8) /DDY
293:                 PAPER(9) = PAPER(9) /DDY
294:             end if
295: C
296: C -----
297: C ... centering to arbitrary point (CENTER(1:3))
298: C
299:             else if ( IBR.ne.0.and.
300:             & (VNM(:NLEN).eq.'CENTER' .or. VNM(:NLEN).eq.'C') ) then
301:
302:                 ICALL    = 1
303:                 NB       = 3
304:                 XI(1)    = 0.0D0
305:                 XI(2)    = 0.0D0
306:                 XI(3)    = 0.0D0
307:                 call R8READ( VNM, XI, NA, -NB, IERR )
308:                 if ( NA.ne.NB ) IRR = IRR + 1
309: C
310:                 write(IPR,'(1x,a/(3x,3e12.5))') ' Center coordinates : ) ',
311:                 &         (XI(J),J=1,3)
312: C
313:                 if ( XMAX.eq.0.0D0 .or. YMAX.eq.0.0D0 ) then
314:                     write(IPR,*) 'XXX drawing area size (XMAX,YMAX) is ',
315:                     &         'not valid. so your CENTER() input is ignored !!'
316:                     NERR    = NERR + 1
317:                 else
318:                     PAPER(1) = XI(1) - 0.5*XMAX*PAPER(4) - 0.5*YMAX*PAPER(7)
319:                     PAPER(2) = XI(2) - 0.5*XMAX*PAPER(5) - 0.5*YMAX*PAPER(8)
320:                     PAPER(3) = XI(3) - 0.5*XMAX*PAPER(6) - 0.5*YMAX*PAPER(9)
321:                 end if
322: C
323: C -----
324: C ... X-AXIS vector (PAPER(4:6))
325: C

```

```

326:             else if ( (VNM(:NLEN).eq.'XAXIS' .or. VNM(:NLEN).eq.'XA')
327:             &         .and.IBR.ne.0 ) then
328:
329:                 ICALL    = 1
330:                 NB       = 3
331:                 XI(1)    = 1.0D0
332:                 XI(2)    = 0.0D0
333:                 XI(3)    = 0.0D0
334:                 call R8READ( VNM, XI, NA, -NB, IERR )
335:                 if ( NA.ne.NB ) IRR = IRR + 1
336: C
337:                 write(IPR,'(1x,a/(3x,3e12.5))') ' X-axis vector : ) ',
338:                 &         (XI(J),J=1,3)
339:                 DDX    = SQRT(XI(1)**2+XI(2)**2+XI(3)**2)
340:                 if ( DDX.eq.0.0D0 ) then
341:                     write(IPR,*) 'XXX X-axis vector is zero vector!!'
342:                     NERR    = NERR + 1
343:                 else
344:                     PAPER(4) = XI(1) /DDX
345:                     PAPER(5) = XI(2) /DDX
346:                     PAPER(6) = XI(3) /DDX
347:                 end if
348: C
349: C -----
350: C ... Y-AXIS vector (PAPER(7:9))
351: C
352:             else if ( (VNM(:NLEN).eq.'YAXIS' .or. VNM(:NLEN).eq.'YA')
353:             &         .and.IBR.ne.0 ) then
354:
355:                 ICALL    = 1
356:                 NB       = 3
357:                 XI(1)    = 1.0D0
358:                 XI(2)    = 0.0D0
359:                 XI(3)    = 0.0D0
360:                 call R8READ( VNM, XI, NA, -NB, IERR )
361:                 if ( NA.ne.NB ) IRR = IRR + 1
362: C
363:                 write(IPR,'(1x,a/(3x,3e12.5))') ' Y-axis vector : ) ',
364:                 &         (XI(J),J=1,3)
365:                 DDY    = SQRT(XI(1)**2+XI(2)**2+XI(3)**2)
366:                 if ( DDY.eq.0.0D0 ) then
367:                     write(IPR,*) 'XXX Y-axis vector is zero vector!!'
368:                     NERR    = NERR + 1
369:                 else
370:                     PAPER(7) = XI(1) /DDY
371:                     PAPER(8) = XI(2) /DDY
372:                     PAPER(9) = XI(3) /DDY
373:                 end if
374: C
375: C -----
376: C ... drawing range size
377: C
378:             else if ( (VNM(:NLEN).eq.'SIZE'.or.VNM(:NLEN).eq.'XMAX')
379:             &         .and.IBR.ne.0 ) then
380:
381:                 ICALL    = 1
382:                 NB       = 2
383:                 call R8READ( VNM, XI(1), NA, -NB, IERR )
384:                 if ( NA.ne.NB ) IRR = IRR + 1
385:                 if ( XI(1).gt.0.0D0 ) XMAX = XI(1)
386:                 if ( XI(2).gt.0.0D0 ) YMAX = XI(2)
387:                 write(IPR,*) ' XMAX (X-Axis length) ', XMAX
388:                 write(IPR,*) ' YMAX (Y-Axis length) ', YMAX
389: C
390: C -----

```

src/cgview/pablo1.f

```

391: C   ... X-Y scaling reatio
392: C
393:       else if ( VNM(:NLEN).eq.'SCALE' ) then
394:         ICALL = 1
395:         NB = 2
396:         call R8READ( VNM, XI(1), NA, -NB, IERR )
397:         if ( NA.ne.NB ) IRR = IRR + 1
398: C
399: C -----
400: C   ... line spacing (PAPER(10))
401: C
402:       else if ( VNM(:NLEN).eq.'LINES' ) then
403: C
404:         ICALL = 1
405:         if ( IBR.ne.0 ) then
406:           NB = 1
407:           call R8READ( VNM, PAPER(10), NA, -NB, IERR )
408:           if ( NA.ne.NB ) IRR = IRR + 1
409:         else
410:           PAPER(10) = -1.0D0
411:           NA = 1
412:         end if
413: C
414:         write(IPR,*)
415:         & ' LINES (PAPER(10)) lines per cm or pixels per line ',
416:         & PAPER(10)
417: C
418:         if ( PAPER(10).eq.0.0D0 ) then
419:           write(IPR,*) 'XXX PAPER(10) (lines per cm) is zero!!'
420:           NERR = NERR + 1
421:         end if
422: C
423: C -----
424: C   ... one touch drawing orientation command !!!
425: C
426:       else if ( VNM(:NLEN).eq.'XY' .or. VNM(:NLEN).eq.'XL'
427: & .or. VNM(:NLEN).eq.'YZ' .or.
428: & VNM(:NLEN).eq.'YX' .or. VNM(:NLEN).eq.'ZX'
429: & .or. VNM(:NLEN).eq.'ZY' ) then
430:         MXYZ = 1
431:         if ( VNM(:NLEN).eq.'YX' ) MXYZ = -1
432:         if ( VNM(:NLEN).eq.'XZ' ) MXYZ = 2
433:         if ( VNM(:NLEN).eq.'ZX' ) MXYZ = -2
434:         if ( VNM(:NLEN).eq.'YZ' ) MXYZ = 3
435:         if ( VNM(:NLEN).eq.'ZY' ) MXYZ = -3
436: C
437:         call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
438: C
439:         call GPCELL( IPR, MCELL, IMCELL, MXYZ, PAPER, XMAX, YMAX,
440: & GRANGE, DINF, A(LIBSDA), A(LGRBOD), NBODY, A(LIDCEL),
441: & A(LIPCEL), A(LICTYP), NCELL, A(LKZAA), NZONE, A(LKZDA),
442: & A(LKSFBD), NZDA )
443: C
444: C -----
445: C   ... window (paper) position commands ...
446: C -----
447: C
448: C -----
449: C -----
450: C   ... zooming : ZOOM( factor [cx [ cy ] ] )
451: C
452: C       factor > 0 : zoom-in
453: C       factor < 0 : zoom-out
454: C
455:       else if ( VNM(:NLEN).eq.'ZOOM' .or. VNM(:NLEN).eq.'ZM' ) then

```

```

456: C
457:       XI(1) = 2.0D0
458:       XI(2) = 0.5D0
459:       XI(3) = 0.5D0
460: C
461:       ICALL = 1
462:       if ( IBR.ne.0 ) then
463:         NB = 3
464:         call R8READ( VNM, XI, NA, -NB, IERR )
465:         if ( NA.gt.NB ) IRR = IRR + 1
466:       else
467:         XI(1) = ZMSAVE
468:         NA = 1
469:       end if
470: C
471:       write(IPR,*) ' ZOOM factor :', XI(1)
472:       write(IPR,*) ' center position : ', XI(2), XI(3)
473: C
474:       if ( NA.ge.1 ) then
475:         if ( XI(1).lt.0 ) XI(1) = 1.0D0/ABS(XI(1))
476:         ZMSAVE = XI(1)
477:         call ZOOMWIN( XI(1), XI(2), XI(3), XMAX, YMAX, PAPER )
478:       end if
479: C
480: C -----
481: C   ... zoom specified input-zone
482: C
483:       else if ( VNM(:NLEN).eq.'ZOOMZONE' .or. VNM(:NLEN).eq.'ZZ' ) then
484:         ICALL = 1
485:         if ( IBR.ne.0 ) then
486:           NB = 2
487:           call I4READ( VNM, ITEMP, NA, -NB, IERR )
488:           if ( NA.gt.0 ) then
489:             IZZZZ = ITEMP(1)
490:             MXYZ = 1
491:             if ( NA.ge.2 ) MXYZ = ITEMP(2)
492:             if ( MXYZ.gt.3 .or. MXYZ.lt.1 ) MXYZ = 1
493: C
494:             call GPZONE( IPR, IZZZZ, MXYZ, MCELL, IMCELL, PAPER,
495: & XMAX, YMAX, GRANGE, DINF, A(LIBSDA), A(LGRBOD),
496: & NBODY, NINPZ, A(LKINPZ), A(LKZAA), NZONE,
497: & A(LKZDA), A(LKSFBD), NZDA, A(LKCELL), A(LIPCEL),
498: & A(LIDCEL), NCELL )
499:           end if
500:         end if
501: C
502: C -----
503: C   ... zoom specified body
504: C
505:       else if ( VNM(:NLEN).eq.'ZOOMBODY' .or. VNM(:NLEN).eq.'ZB' ) then
506:         ICALL = 1
507:         if ( IBR.ne.0 ) then
508:           NB = 2
509:           call I4READ( VNM, ITEMP, NA, -NB, IERR )
510:           if ( NA.gt.0 ) then
511:             IBBBB = ITEMP(1)
512:             MXYZ = 1
513:             if ( NA.ge.2 ) MXYZ = ITEMP(2)
514:             if ( MXYZ.gt.3 .or. MXYZ.lt.1 ) MXYZ = 1
515: C
516:             call GPBODY( IPR, IBBBB, MXYZ, PAPER, XMAX, YMAX, GRANGE,
517: & DINF, A(LIBSDA), A(LGRBOD), NBODY )
518:           end if
519:         end if
520: C

```

src/cgview/pablo1.f

```

521: C -----
522: C     ... relative movement
523: C
524: C     else if ( VNM(:NLEN).eq.'MOVE' .or. VNM(:NLEN).eq.'M'
525: C     & .or. VNM(:NLEN).eq.'MX' .or. VNM(:NLEN).eq.'MY'
526: C     & .or. VNM(:NLEN).eq.'MZ' ) then
527: C
528: C         XI(1) = 0.0D0
529: C         XI(2) = 0.0D0
530: C         XI(3) = 0.0D0
531: C
532: C         ICALL = 1
533: C         if ( IBR.ne.0 ) then
534: C             if ( VNM(:NLEN).eq.'MX' ) then
535: C                 NB = -1
536: C                 call R8READ( VNM, XI(1), NA, NB, IERR )
537: C                 if ( NA.gt.ABS(NB) ) IRR = IRR + 1
538: C                 if ( NA.ge.1 ) XMVSAVE = XI(1)
539: C             else if ( VNM(:NLEN).eq.'MY' ) then
540: C                 NB = -1
541: C                 call R8READ( VNM, XI(2), NA, NB, IERR )
542: C                 if ( NA.gt.ABS(NB) ) IRR = IRR + 1
543: C                 if ( NA.ge.1 ) YMVSAVE = XI(2)
544: C             else if ( VNM(:NLEN).eq.'MZ' ) then
545: C                 NB = -1
546: C                 call R8READ( VNM, XI(3), NA, NB, IERR )
547: C                 if ( NA.gt.ABS(NB) ) IRR = IRR + 1
548: C                 if ( NA.ge.1 ) ZMVSAVE = XI(3)
549: C             else
550: C                 NB = -3
551: C                 call R8READ( VNM, XI, NA, NB, IERR )
552: C                 if ( NA.gt.ABS(NB) ) IRR = IRR + 1
553: C                 if ( NA.ge.1 ) XMVSAVE = XI(1)
554: C                 if ( NA.ge.2 ) YMVSAVE = XI(2)
555: C                 if ( NA.ge.3 ) ZMVSAVE = XI(3)
556: C             end if
557: C         else
558: C             NA = 1
559: C             if ( VNM(:NLEN).eq.'MX' ) then
560: C                 XI(1) = XMVSAVE
561: C             else if ( VNM(:NLEN).eq.'MY' ) then
562: C                 XI(2) = YMVSAVE
563: C             else if ( VNM(:NLEN).eq.'MZ' ) then
564: C                 XI(3) = ZMVSAVE
565: C             else
566: C                 XI(1) = XMVSAVE
567: C                 XI(2) = YMVSAVE
568: C                 XI(3) = ZMVSAVE
569: C             end if
570: C         end if
571: C
572: C         if ( NA.ge.1 ) then
573: C             call MOVEWIN( XI(1), XI(2), XI(3), XMAX, YMAX, PAPER )
574: C         end if
575: C -----
576: C     ... absolute movement
577: C
578: C     else if ( IBR.ne.0.and.(VNM(:NLEN).eq.'MOVEA'
579: C     & .or.VNM(:NLEN).eq.'MA'.or.VNM(:NLEN).eq.'MXA'
580: C     & .or.VNM(:NLEN).eq.'MYA'.or.VNM(:NLEN).eq.'MZA' ) ) then
581: C
582: C         XI(1) = 0.0D0
583: C         XI(2) = 0.0D0
584: C         XI(3) = 0.0D0
585: C

```

```

586: ICALL = 1
587:
588: if ( VNM(:NLEN).eq.'MXA' ) then
589:     NB = -1
590:     call R8READ( VNM, XI(1), NA, NB, IERR )
591:     if ( NA.gt.ABS(NB) ) IRR = IRR + 1
592: else if ( VNM(:NLEN).eq.'MYA' ) then
593:     NB = -1
594:     call R8READ( VNM, XI(2), NA, NB, IERR )
595:     if ( NA.gt.ABS(NB) ) IRR = IRR + 1
596: else if ( VNM(:NLEN).eq.'MZA' ) then
597:     NB = -1
598:     call R8READ( VNM, XI(3), NA, NB, IERR )
599:     if ( NA.gt.ABS(NB) ) IRR = IRR + 1
600: else
601:     NB = -3
602:     call R8READ( VNM, XI, NA, NB, IERR )
603:     if ( NA.gt.ABS(NB) ) IRR = IRR + 1
604: end if
605: if ( NA.ge.1 ) then
606:     call MOVEAWIN( XI(1), XI(2), XI(3), PAPER )
607: end if
608: C -----
609: C ... resize window : RESIZE|RS( dx [dy [ ic ]] )
610: C
611: C     else if ( IBR.ne.0
612: C     & .and.(VNM(:NLEN).eq.'RESIZE'.or.VNM(:NLEN).eq.'RS' ) ) then
613: C
614: C         ICALL = 1
615: C         NB = -3
616: C         XI(1) = 0.0D0
617: C         XI(2) = 0.0D0
618: C         XI(3) = 0.0D0
619: C
620: C         call R8READ( VNM, XI, NA, NB, IERR )
621: C
622: C         IC = NINT(XI(3))
623: C         JAR = 0
624: C         call RESIZEWIN( XI(1), XI(2), JAR, IC, XMAX, YMAX, PAPER )
625: C
626: C     ... rotation
627: C
628: C     else if ( VNM(:NLEN).eq.'ROT' .or. VNM(:NLEN).eq.'ROTX'
629: C     & .or. VNM(:NLEN).eq.'ROTY' ) then
630: C
631: C         XI(1) = 0.0D0
632: C         XI(2) = 0.5D0
633: C         XI(3) = 0.5D0
634: C
635: C         ICALL = 1
636: C         if ( IBR.ne.0 ) then
637: C             NB = -3
638: C             call R8READ( VNM, XI, NA, NB, IERR )
639: C             if ( NA.gt.ABS(NB) ) IRR = IRR + 1
640: C             if ( NA.ge.1 ) then
641: C                 if ( VNM(:NLEN).eq.'ROT' ) then
642: C                     ROTSAVE = XI(1)
643: C                 else if ( VNM(:NLEN).eq.'ROTX' ) then
644: C                     XROTSAVE = XI(1)
645: C                 else if ( VNM(:NLEN).eq.'ROTY' ) then
646: C                     YROTSAVE = XI(1)
647: C                 end if
648: C             end if
649: C         else
650: C             if ( NA.ge.1 ) then
651: C                 if ( VNM(:NLEN).eq.'ROT' ) then

```

src/cgview/pablo1.f

```

651:          XI(1) = ROTSAVE
652:          else if ( VNM(:NLEN).eq.'ROTX' ) then
653:            XI(1) = XROTSAVE
654:          else if ( VNM(:NLEN).eq.'ROTY' ) then
655:            XI(1) = YROTSAVE
656:          end if
657:        end if
658:      end if
659: C
660:      write(IPR,*) ' ', VNM(:NLEN), ' deg ', XI(1)
661:      write(IPR,*) ' Axis position : ', XI(2), XI(3)
662: C
663:      call ROTATEWIN( VNM(:NLEN), XI(1), XI(2), XI(3), XMAX, YMAX,
664: & PAPER )
665: C
666: C -----
667: C ... X-Y scan mode
668: C
669:      else if ( VNM(:NLEN).eq.'SCAN'.and.IBR.ne.0 ) then
670:        ICALL = 1
671:        NB = 2
672:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
673:        if ( NA.ne.NB ) IRR = IRR + 1
674: C
675:        JSCANX = ITEMP(1)
676:        JSCANY = ITEMP(2)
677:        write(IPR,*) 'X-SCAN MODE ( 1/2/3 = +y/-y/+y/ ) : ', JSCANX
678:        write(IPR,*) 'Y-SCAN MODE ( 1/2/3 = +x/-x/+x/ ) : ', JSCANY
679: C
680: C -----
681: C ... flooding particle mode
682: C
683:        FLOOD( flag number-of-particles )
684:        or
685:        FL( flag number-of-particles )
686: C
687:      else if ( VNM(:NLEN).eq.'FLOOD'.or.VNM(:NLEN).eq.'FL' ) then
688:        ICALL = 1
689:        NB = 2
690:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
691:        if ( NA.gt.0 ) JGFLOOD = ITEMP(1)
692:        if ( NA.gt.1 ) NGFLOOD = ITEMP(2)
693: C
694:        write(IPR,*) 'Flood particle : ', JGFLOOD,
695: & ', number of particle : ', NGFLOOD
696:        if ( ABS(GRANGE(1)).eq.DINF .or. ABS(GRANGE(2)).eq.DINF
697: & .or. ABS(GRANGE(3)).eq.DINF .or. ABS(GRANGE(4)).eq.DINF
698: & .or. ABS(GRANGE(5)).eq.DINF .or. ABS(GRANGE(6)).eq.DINF )
699: & then
700:          write(IPR,*)
701: & ' >>> Geometry range (GRANGE) has infinite bound.'
702:          write(IPR,*) ' >>> Cannot change to particle flooding mode.'
703:          JGFLOOD = 0
704:          write(IPR,*) 'Current geometry range : '
705:          write(IPR,7000) ' X ', GRANGE(1), GRANGE(2)
706:          write(IPR,7000) ' Y ', GRANGE(3), GRANGE(4)
707:          write(IPR,7000) ' Z ', GRANGE(5), GRANGE(6)
708:          if ( JGFLOOD.ne.0 ) then
709:            call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
710:          end if
711:        end if
712: C
713: C -----
714: C ... force GRANGE for flooding particle mode if necessary
715: C

```

```

716: C
717:      else if ( VNM(:NLEN).eq.'GRANGE' ) then
718:        ICALL = 1
719:        NB = 6
720:        if ( IBR.ne.0 ) then
721:          call R8READ( VNM, GRANGE, NA, -NB, IERR )
722:          if ( GRANGE(2).gt.GRANGE(1) ) then
723:            DGR = GRANGE(1)
724:            GRANGE(1) = GRANGE(2)
725:            GRANGE(2) = DGR
726:          end if
727:          if ( GRANGE(4).gt.GRANGE(3) ) then
728:            DGR = GRANGE(3)
729:            GRANGE(3) = GRANGE(4)
730:            GRANGE(4) = DGR
731:          end if
732:          if ( GRANGE(6).gt.GRANGE(5) ) then
733:            DGR = GRANGE(5)
734:            GRANGE(5) = GRANGE(6)
735:            GRANGE(6) = DGR
736:          end if
737:          write(IPR,*) 'Geometry range : '
738:          write(IPR,7000) ' X ', GRANGE(1), GRANGE(2)
739:          write(IPR,7000) ' Y ', GRANGE(3), GRANGE(4)
740:          write(IPR,7000) ' Z ', GRANGE(5), GRANGE(6)
741: C
742:          call GR2PAPER( GRANGE, MXYZ, PAPER, XMAX, YMAX, DINF )
743:        end if
744: 7000 format(1X,A,1P2E13.5)
745: C
746: C -----
747: C ... zone search order ...
748: C
749:      else if ( VNM(:NLEN).eq.'ZSEARCH' .or. VNM(:NLEN).eq.'ZS' ) then
750:        ICALL = 1
751:        NB = 1
752:        call I4READ( VNM, JZREVERSE, NA, -NB, IERR )
753:        if ( NA.gt.0 ) then
754:          write(IPR,*)
755: & ' Zone search order (0/1=forward/reverse,2=both) : ',
756: & JZREVERSE
757:          call FORGET( A(LKMEMO2), A(LMEMC2), A(LMEMZ2), NZONE, NMEMO
758: & )
759:        end if
760: C
761: C -----
762: C ... material/region overlap check ...
763: C
764:      else if ( VNM(:NLEN).eq.'OVERLAP' .or. VNM(:NLEN).eq.'OVL' ) then
765:        ICALL = 1
766:        if ( IBR.ne.0 ) then
767:          NB = 1
768:          call I4READ( VNM, JGMCK, NA, -NB, IERR )
769:        else
770:          JGMCK = 0
771:        end if
772:        write(IPR,*) ' Material/region overlap check: ', JGMCK
773: C
774: C -----
775: C ... overlap judgement torelance ...
776: C
777:      else if ( VNM(:NLEN).eq.'DOVERLAP' .or. VNM(:NLEN).eq.'DOVL' )
778: & then
779:        ICALL = 1
780:        NB = 1

```

src/cgview/pablo1.f

```

781:      call R8READ( VNM, DOVERLAP, NA, -NB, IERR )
782:      if ( NA.ne.NB ) IRR = IRR + 1
783:      write(IPR,*) ' Overlap check tolerance: ', DOVERLAP
784: C
785: C -----
786: C ... overlap message count limit per picture ...
787: C (negative MPLIM means no-limit, zero value means check
788: C for current value)
789: C
790:      else if ( VNM(:NLEN).eq.'MPLIM' .or. VNM(:NLEN).eq.'MPL' )
791:      & then
792:          ICALL = 1
793:          NB = 1
794:          call I4READ( VNM, MPLIMIT0, NA, -NB, IERR )
795:          if ( NA.ne.NB ) IRR = IRR + 1
796:          if ( MPLIMIT0.ne.0 ) MPLIMIT = MPLIMIT0
797:          write(IPR,*) ' Overlap check message count limit: ', MPLIMIT
798: C
799: C -----
800: C ... clear next zone memory ...
801: C
802:      else if ( VNM(:NLEN).eq.'FORGET' ) then
803:          ICALL = 1
804:          call FORGET( A(LKMEMO2), A(LMEMC2), A(LMEMZ2), NZONE, NMEMO )
805:          write(IPR,*) ' Next zone memory is cleared.'
806: C
807: C -----
808: C ... change geometry nesting level
809: C
810:      else if ( IBR.ne.0 .and.
811:      & ( VNM(:NLEN).eq.'LEVEL' .or. VNM(:NLEN).eq.'LV' ) ) then
812: C
813:          ICALL = 1
814:          NB = 1
815:          call I4READ( VNM, LEVEL, NA, -NB, IERR )
816:          if ( NA.ne.NB ) IRR = IRR + 1
817:          write(IPR,*) ' LEVEL (geometry nesting level) ', LEVEL
818: C
819: C -----
820: C ... toggle subframe boundary drawing mode
821: C
822:      else if ( VNM(:NLEN).eq.'SUBF' .or. VNM(:NLEN).eq.'SBF' ) then
823:          LEVEL = -ABS(LEVEL)
824:      else if ( VNM(:NLEN).eq.'NOSUBF' .or. VNM(:NLEN).eq.'NOSBF' ) then
825:          LEVEL = ABS(LEVEL)
826: C
827: C -----
828: C ... specify cell :
829: C MCELL > 0 : cell ID
830: C MCELL = 0 : global geometry
831: C MCELL < 0 : cell ID inquiry (do not set PAPER & XMAX/YMAX)
832: C
833:      else if ( VNM(:NLEN).eq.'CELL' ) then
834:          ICALL = 1
835:          MXYZ = 1
836:          if ( IBR.eq.0 ) then
837:              MCELL0 = -1
838:          else
839:              NB = 2
840:              call I4READ( VNM, ITEMP, NA, -NB, IERR )
841:              if ( NA.gt.NB ) IRR = IRR + 1
842:              if ( NA.gt.0 ) then
843:                  MCELL0 = ITEMP(1)
844:                  MXYZ = 1
845:                  if ( NA.ge.2 ) MXYZ = ITEMP(2)

```

```

846:          if ( MXYZ.gt.3 .or. MXYZ.lt.1 ) MXYZ = 1
847:          end if
848:      end if
849:      if ( JLATT.eq.0.and.MCELL0.eq.0 ) then
850:          write(IPR,*) ' >> CELL(0) for non-lattice geometry means',
851:      & ' reset to default scene.'
852:      end if
853:
854:      MCELL1 = MCELL0
855:
856:      call GPCELL( IPR, MCELL1, IMCELL1, MXYZ, PAPER, XMAX, YMAX,
857:      & GRANGE, DINF, A(LIBSDA), A(LGRBOD), NBODY, A(LIDCEL),
858:      & A(LIPCEL), A(LICTYP), NCELL, A(LKZAA), NZONE, A(LKZDA),
859:      & A(LKSFBDD), NZDA )
860:
861:      if ( MCELL0.gt.0.and.IMCELL1.gt.0 ) then
862:          MCELL = MCELL1
863:          IMCELL = IMCELL1
864:      else if ( MCELL0.eq.0 ) then
865:          MCELL = MCELL0
866:          IMCELL = IMCELL1
867:      end if
868: C
869: C -----
870: C ... change color index entry (who can use this ???)
871: C
872:      else if ( VNM(:NLEN).eq.'COLOR'.and.IBR.ne.0 ) then
873:
874:          ICALL = 1
875:          NB = 200
876:          call I4READ( VNM, CLDAT, NA, -NB, IERR )
877:
878:          do 130 I = 0, 199
879:              if ( CLDAT(I).le.0 ) CLDAT(I) = 11
880:          130 continue
881:
882:          do 150 K = 200, MXCLDAT, 200
883:              K2 = MIN(MXCLDAT,K+200-1)
884:              do 140 I = K, K2
885:                  CLDAT(I) = CLDAT(I-200)
886:              140 continue
887:          150 continue
888:
889:          WRITE(6,*) ' COLOR ',CLDAT
890:
891: C
892: C -----
893: C ... change zone color index entry : ZCOLOR( zone# color# )
894: C
895:      else if ( IBR.ne.0
896:      & .and.(VNM(:NLEN).eq.'ZCOLOR'.or.VNM(:NLEN).eq.'ZC' ) ) then
897:          ICALL = 1
898:          NB = 2
899:          call I4READ( VNM, ITEMP, NA, -NB, IERR )
900:          if ( NA.ne.NB ) IRR = IRR + 1
901: C
902:          if ( ITEMP(1).gt.0.and.ITEMP(1).le.NZONE ) then
903:              if ( ITEMP(2).ge.0.and.ITEMP(2).le.MXCLRIDX ) then
904:                  ZONCLR(ITEMP(1)) = ITEMP(2)
905:              end if
906:          end if
907: C
908: C -----
909: C ... reset zone color index entry to default
910: C

```

src/cgview/pablo1.f

```

911:      else if ( VNM(:NLEN).eq.'ZCR' ) then
912:        call ZCLR00( ZONCLR, NZONE, NCC )
913:        write(IPR,*) ' Zone color index is reset to default.'
914: C
915: C -----
916: C ... change input-zone color index entry : IZCOLOR( izeone# color# )
917: C
918:      else if ( IBR.ne.0
919:      &        .and.(VNM(:NLEN).eq.'IZCOLOR'.or.VNM(:NLEN).eq.'IZC' )
920:      &        then
921:        ICALL = 1
922:        NB = 2
923:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
924:        if ( NA.ne.NB ) IRR = IRR + 1
925: C
926:        if ( ITEMP(1).gt.0.and.ITEMP(1).le.NINPZ ) then
927:          if ( ITEMP(2).ge.0.and.ITEMP(2).le.MXCLRIDX ) then
928:            IZNCLR(ITEMP(1)) = ITEMP(2)
929:          and if
930:            end if
931: C
932: C -----
933: C ... reset input-zone color index entry to default
934: C
935:      else if ( VNM(:NLEN).eq.'IZCR' ) then
936:        call IZCLR00( IZNCLR, NINPZ, NCC )
937:        write(IPR,*) ' Input-Zone color index is reset to default.'
938: C
939: C -----
940: C ... change material color index entry : MCOLOR( mat# color# )
941: C
942:      else if ( IBR.ne.0
943:      &        .and.(VNM(:NLEN).eq.'MCOLOR'.or.VNM(:NLEN).eq.'MC' ) ) then
944:        ICALL = 1
945:        NB = 2
946:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
947:        if ( NA.ne.NB ) IRR = IRR + 1
948:        if ( ITEMP(1).ge.0.and.ITEMP(1).le.MXMAT ) then
949:          if ( ITEMP(2).ge.0.and.ITEMP(2).le.MXCLRIDX ) then
950:            MATCLR(ITEMP(1)) = ITEMP(2)
951:          end if
952:        end if
953: C
954: C -----
955: C ... reset material color index entry to default
956: C
957:      else if ( VNM(:NLEN).eq.'MCR' ) then
958:        call MCLR00( MATCLR, MXMAT, NCC )
959:        write(IPR,*) ' Material color index is reset to default.'
960: C
961: C -----
962: C ... change region color index entry : RCOLOR( reg# color# )
963: C
964:      else if ( IBR.ne.0.and.(VNM(:NLEN).eq.'RCOLOR'
965:      &        .or.VNM(:NLEN).eq.'IRC'.or.VNM(:NLEN).eq.'RCOLOR'
966:      &        .or.VNM(:NLEN).eq.'RC' ) ) then
967:        ICALL = 1
968:        NB = 2
969:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
970:        if ( NA.ne.NB ) IRR = IRR + 1
971:        if ( ITEMP(1).gt.0.and.ITEMP(1).le.NREG ) then
972:          if ( ITEMP(2).ge.0.and.ITEMP(2).le.MXCLRIDX ) then
973:            REGCLR(ITEMP(1)) = ITEMP(2)
974:          end if
975:        end if

```

```

976: C
977: C -----
978: C ... reset region color index entry to default
979: C
980:      else if ( VNM(:NLEN).eq.'RCR' ) then
981:        call RCLR00( REGCLR, NREG, NCC )
982:        write(IPR,*) ' Region color index is reset to default.'
983: C
984: C -----
985: C ... IKIND : unused ....
986: C
987:      else if ( VNM(:NLEN).eq.'IKIND' ) then
988: C
989:        ICALL = 1
990:        NB = 1
991:        call I4READ( VNM, IKIND, NA, NB, IERR )
992:        if ( NA.ne.NB ) IRR = IRR + 1
993:        write(IPR,*) ' IKIND (not used) ', IKIND
994: C
995: C -----
996: C ... MLOST : lost message limit per zone ....
997: C
998:      else if ( VNM(:NLEN).eq.'MLOST' ) then
999: C
1000:        ICALL = 1
1001:        NB = 1
1002:        call I4READ( VNM, MSGSLST, NA, -NB, IERR )
1003:        if ( NA.ne.NB ) IRR = IRR + 1
1004:        write(IPR,*) ' MSGSLST :', MSGSLST
1005: C
1006:        if ( MSGSLST.lt.0 ) MSGSLST = ABS(MSGSLST)
1007: C
1008: C -----
1009: C ... drawing style : paint or boundary
1010: C
1011:      else if ( IBR.ne.0.and.VNM(:NLEN).eq.'STYLE'
1012:      &        .or.VNM(:NLEN).eq.'ST' ) then
1013: C
1014:        ICALL = 1
1015:        NB = 2
1016:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
1017:        if ( NA.gt.NB ) IRR = IRR + 1
1018: C
1019:        STYLE = ITEMP(1)
1020:        if ( NA.ge.2 ) STYLE2 = ITEMP(2)
1021: C
1022:        write(IPR,*) ' STYLE (0=raster scan/1=boundary curve) ', STYLE
1023:        if ( STYLE.eq.0 ) write(IPR,*)
1024:        &        ' STYLE2 (0=color paint/1=mark boundary) ', STYLE2
1025: C
1026: C -----
1027: C ... drawing style (combined)
1028: C
1029: C
1030: C === boundary : bundary drawing by STYLE=0 and STYLE2 = 1
1031: C and line(-1) scan( 1 1 )
1032: C
1033:      else if ( VNM(:NLEN).eq.'BOUNDARY' .or. VNM(:NLEN).eq.'BDY' ) then
1034:        STYLE = 0
1035:        STYLE2 = 1
1036:        JSCANX = 1
1037:        JSCANY = 1
1038:        PAPER(10) = -1.0d0
1039: C
1040: C -----

```


src/cgview/pablo1.f

```

1041: C      === paint : paint drawing by STYLE=0 and STYLE2 = 0
1042: C      and line(-1) scan( 0 3 )
1043: C
1044: C      else if ( VNM(:NLEN).eq.'PAINT' .or. VNM(:NLEN).eq.'PNT' ) then
1045: C          STYLE = 0
1046: C          STYLE2 = 0
1047: C          JSCANX = 0
1048: C          JSCANY = 3
1049: C          PAPER(10) = -1.0d0
1050: C
1051: C -----
1052: C      ... what is drawn : 0/1/2 = zone/region/material
1053: C
1054: C      else if ( IBR.ne.0
1055: C      & .and.VNM(:NLEN).eq.'SPTYP' .or. VNM(:NLEN).eq.'WHAT' )
1056: C      & then
1057: C
1058: C          ICALL = 1
1059: C          NB = 1
1060: C          call I4READ( VNM, IS0, NA, -NB, IERR )
1061: C          if ( NA.ne.NB ) IRR = IRR + 1
1062: C          write(IPR,*) ' SPTYP(-1/0/1/2=zone/i=zone/region/material) '
1063: C          & IS0
1064: C          if ( IS0.ne.-1.and.IS0.ne.0.and.IS0.ne.1.and.IS0.ne.2 ) then
1065: C              write(IPR,*) ' !!! invalid SPTYP. Ignored.'
1066: C          else
1067: C              SPTYP = IS0
1068: C          end if
1069: C
1070: C -----
1071: C      ... title string
1072: C
1073: C      else if ( IBR.ne.0.and.VNM(:NLEN).eq.'TITLE' ) then
1074: C
1075: C          ICALL = 1
1076: C          NB = 59
1077: C          NE = 0
1078: C          TITLGR = ' '
1079: C
1080: C      160 continue
1081: C          NL = NB - NE
1082: C          if ( NL.gt.0 ) then
1083: C              call CHREAD( VNM, LINE1, ' ', NLEN, ITERM, IEND )
1084: C              NLEN = MIN(NL,NLEN)
1085: C              if ( NLEN.gt.0 ) TITLGR(NE+1:NE+NLEN) = LINE1(1:NLEN)
1086: C              NE = NLEN + NE + 1
1087: C              if ( ITERM.eq.0.and.IEND.eq.0 ) go to 160
1088: C          end if
1089: C
1090: C          TITLGR(NE:NE) = ' '
1091: C          call RESET
1092: C          write(IPR,*) ' TITLE ', TITLGR
1093: C
1094: C -----
1095: C      ... IDENT
1096: C
1097: C      else if ( VNM(:NLEN).eq.'IDENT' ) then
1098: C
1099: C          ICALL = 1
1100: C          NB = 1
1101: C          call I4READ( VNM, IDENT, NA, NB, IERR )
1102: C          if ( NA.ne.NB ) IRR = IRR + 1
1103: C          write(IPR,*) ' IDENT ', IDENT
1104: C
1105: C -----

```

```

1106: C      ... redraw current window ( specific to PIFFLIB/CALCOMP)
1107: C
1108: C      else if ( VNM(:NLEN).eq.'REDRAW' .or. VNM(:NLEN).eq.'RD' ) then
1109: C          ICALL = 1
1110: C          call PLOT( 0. , 0. , 888 )
1111: C
1112: C -----
1113: C      ... draw image (or output pixmap on display on X Winowd System)
1114: C      every INTREFRESH's call to GRAPHs routine.
1115: C
1116: C      else if ( IBR.ne.0
1117: C      & .and.(VNM(:NLEN).eq.'REFRESH' .or. VNM(:NLEN).eq.'RF' ) )
1118: C      & then
1119: C          ICALL = 1
1120: C          INTO = INTREFRESH
1121: C          NB = 1
1122: C          call I4READ( VNM, INTREFRESH, NA, NB, IERR )
1123: C          if ( INTREFRESH.lt.1 ) INTREFRESH = INTO
1124: C
1125: C -----
1126: C      ... enable/disable key event waiting in PIFF lib drawing window
1127: C
1128: C      else if ( VNM(:NLEN).eq.'WAITOFF' .or. VNM(:NLEN).eq.'WOFF' ) then
1129: C          ICALL = 1
1130: C          call PLOT( 0. , 0. , 111 )
1131: C          IWON = 0
1132: C      else if ( VNM(:NLEN).eq.'WAITON' .or. VNM(:NLEN).eq.'WON' ) then
1133: C          ICALL = 1
1134: C          call PLOT( 0. , 0. , 112 )
1135: C          IWON = 1
1136: C
1137: C -----
1138: C      ... display or do not display version ...
1139: C
1140: C      else if ( VNM(:NLEN).eq.'VERSION' ) then
1141: C          OVERSION = 1
1142: C      else if ( VNM(:NLEN).eq.'NOVERSION' ) then
1143: C          OVERSION = 0
1144: C
1145: C -----
1146: C      ... display or do not display lost symbol description ...
1147: C
1148: C      else if ( VNM(:NLEN).eq.'LOSTSYM' ) then
1149: C          OLOSTSYM = 1
1150: C          write(IPR,*) ' LOSTSYM (lost position symbol) : On'
1151: C      else if ( VNM(:NLEN).eq.'NOLOSTSYM' ) then
1152: C          OLOSTSYM = 0
1153: C          write(IPR,*) ' LOSTSYM (lost position symbol) : Off'
1154: C
1155: C -----
1156: C      ... display or do not display frame of image region ...
1157: C
1158: C      else if ( VNM(:NLEN).eq.'FRAME' ) then
1159: C          OFRAME = 1
1160: C      else if ( VNM(:NLEN).eq.'NOFRAME' ) then
1161: C          OFRAME = 0
1162: C
1163: C -----
1164: C      ... display as old version or not ...
1165: C
1166: C      else if ( VNM(:NLEN).eq.'OLDFACE' ) then
1167: C          OFRAME = 1
1168: C          OVERSION = 0
1169: C          OLOSTSYM = 1
1170: C          JHYAXIS = 0

```

src/cgview/pablo1.f

```

1171:      else if ( VNM(:NLEN).eq.'NEWFACE' ) then
1172:        OFRAME = 1
1173:        OVERSION = 1
1174:        OLOSTSYM = 0
1175:        JHYAXIS = 1
1176: C
1177: C -----
1178: C ... check mode ...
1179: C
1180:      else if ( VNM(:NLEN).eq.'CHECK' ) then
1181:        call ECHOPARAM( IPR, PAPER, XMAX, YMAX, JSCANX, JSCANY, SPTYP,
1182:          &      STYLE, STYLE2, LEVEL, INTREFRESH, MCELL, IMCELL,
1183:          &      JZREVERSE, GRANGE, JGFLOOD )
1184: C
1185: C -----
1186: C ... eval mode : only printout input item. can be used for "dentak"
1187: C
1188:      else if ( VNM(:NLEN).eq.'EVAL' .or. VNM(:NLEN).eq.'E' ) then
1189:        NB = 6
1190:        call I4READ( VNM, XI, NA, -NB, IERR )
1191:        do 170 I = 1, NA
1192:          write(IPR,7020) XI(I), XI(I)
1193:        170 continue
1194:        7020 format(1X,' Evaluated: ',1P,E20.9,0P,F16.8)
1195: C
1196: C -----
1197: C ... specify ID of scene parameter explicitly
1198: C ... SAVE[( id )] or S[( id )]
1199: C ( all scenes will be saved anyway if buffer size allows )
1200: C
1201:      else if ( VNM(:NLEN).eq.'SAVE' .or. VNM(:NLEN).eq.'S' ) then
1202:        IDSAVE = -1
1203:        if ( IBR.ne.0 ) then
1204:          NB = 1
1205:          call I4READ( VNM, IDSAVE, NA, NB, IERR )
1206:        end if
1207: C
1208: C -----
1209: C ... restore drawing parameter ... RESTORE[( id )] or R[( id )]
1210: C
1211:      else if ( VNM(:NLEN).eq.'RESTORE' .or. VNM(:NLEN).eq.'R' ) then
1212:        IDSAVE = -1
1213:        if ( IBR.ne.0 ) then
1214:          NB = 1
1215:          call I4READ( VNM, IDSAVE, NA, -NB, IERR )
1216:        end if
1217:        call RESTOREPARAM( IPR, 'GET', IDSAVE, IDRESTORE, TITLGR,
1218:          &      PAPER, XMAX, YMAX, JSCANX, JSCANY, SPTYP, STYLE,
1219:          &      STYLE2, LEVEL, INTREFRESH, OFRAME, OVERSION, OLOSTSYM, MCELL,
1220:          &      IMCELL, JZREVERSE, GRANGE, JGFLOOD )
1221:        JFWBK = 2
1222: C
1223: C -----
1224: C ... FORWARD|F BACK|B : move saved parameter list
1225: C
1226:      else if ( VNM(:NLEN).eq.'FORWARD' .or. VNM(:NLEN).eq.'F' ) then
1227:        JFWBK = 1
1228:        call RESTOREPARAM( IPR, 'FORWARD', 1, IDRESTORE, TITLGR, PAPER,
1229:          &      XMAX, YMAX, JSCANX, JSCANY, SPTYP, STYLE, STYLE2,
1230:          &      LEVEL, INTREFRESH, OFRAME, OVERSION, OLOSTSYM, MCELL,
1231:          &      IMCELL, JZREVERSE, GRANGE, JGFLOOD )
1232: C
1233:      else if ( VNM(:NLEN).eq.'BACK' .or. VNM(:NLEN).eq.'B' ) then
1234:        JFWBK = -1
1235:        call RESTOREPARAM( IPR, 'BACK', 1, IDRESTORE, TITLGR, PAPER,

```

```

1236:      &      XMAX, YMAX, JSCANX, JSCANY, SPTYP, STYLE, STYLE2,
1237:      &      LEVEL, INTREFRESH, OFRAME, OVERSION, OLOSTSYM, MCELL,
1238:      &      IMCELL, JZREVERSE, GRANGE, JGFLOOD )
1239: C
1240: C -----
1241: C ... echo input-zone ( zone ) composition
1242: C
1243:      else if ( ( VNM(:NLEN).eq.'ZONE'.or.VNM(:NLEN).eq.'Z' ).and.IBR.ne.0
1244:        &      ) then
1245:        ICALL = 1
1246:        NB = MXITMP
1247:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
1248:        if ( NA.gt.0 ) then
1249:          do 180 I = 1, NA
1250:            IZZZZ = ITEMP(I)
1251:            call SHOWZN( IPR, IZZZZ, NINPZ, NZONE, NBODY, NREG, NZDA,
1252:              &      NSDA, NCELL, A(LKINPZ), A(LKZDA), A(LKZAA),
1253:              &      A(LSDA), A(LKMAT), A(LKZMAT), A(LKCELL),
1254:              &      A(LIDCEL), A(LKSFBF), CHA(LIZNAM), A(LKBZL),
1255:              &      A(LIBZL), A(LIBSDA), A(LKREG), CHA(LKREGN),
1256:              &      A(LKREGI), CHA(LTNAMS) )
1257:          180 continue
1258:        end if
1259: C
1260: C -----
1261: C ... echo material composition
1262: C
1263:      else if ( VNM(:NLEN).eq.'MAT' ) then
1264:        ICALL = 1
1265:        NB = MXITMP
1266:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
1267:        if ( CODE.eq.'MVP'.and.NA.gt.0 ) then
1268:          do 190 I = 1, NA
1269:            IMMMM = ITEMP(I)
1270:            call SHOWMT( IPR, IMMMM, CODE, JNEUT, JPHOT, NMAT, NUC,
1271:              &      NPATOM, NINPZ, NZONE, A(LIDMAT), A(LMNUC),
1272:              &      A(LINUCT), A(LDENST), A(LLPDEN), A(LMPATM),
1273:              &      A(LNATMT), A(LIATMT), A(LDNSTP), A(LLPDNP),
1274:              &      CHA(LNCIDI), CHA(LNUCID), A(LTEMPN), A(LKZMAT),
1275:              &      A(LKMAT) )
1276:          190 continue
1277:          else if ( NA.gt.0.and.CODE.ne.'MVP' ) then
1278:            write(IPR,*) '!!! MAT command is only for MVPinput.'
1279:          end if
1280: C
1281: C -----
1282: C ... echo body
1283: C
1284:      else if ( ( VNM(:NLEN).eq.'BODY'.or.VNM(:NLEN).eq.'BD' )
1285:        &      .and.IBR.ne.0 ) then
1286:        ICALL = 1
1287:        NB = MXITMP
1288:        call I4READ( VNM, ITEMP, NA, -NB, IERR )
1289:        if ( NA.gt.0 ) then
1290:          do 200 I = 1, NA
1291:            IBBBB = ITEMP(I)
1292:            call SHOWBD( IPR, IBBBB, JTLLT, NINPZ, NZONE, NBODY,
1293:              &      NREG, NZDA, NSDA, NSURF, A(LDBODI), A(LIBODI),
1294:              &      A(LIPSDA), A(LKINPZ), A(LKZDA), A(LKZAA),
1295:              &      A(LSDA), A(LKMAT), A(LKZMAT), A(LKSFBF),
1296:              &      CHA(LIZNAM), A(LIBSDA), A(LKREG), CHA(LKREGN),
1297:              &      A(LKREGI), CHA(LTNAMS) )
1298:          200 continue
1299:        end if
1300: C

```

```

end if
endif
-----
KMEMO ...

else if ( VNM(:NLEN).eq.'KMEMO' ) then
    ICALL    = 1
    NB       = -2
    call I4READ( VNM, ITEMP, NA, NB, IERR )

    if ( NA.gt.0 ) then
        IZ = ITEMP(1)
        IR = 0
        if ( NA.gt.1 ) IR = ITEMP(2)
        write(IPR,*) ' KMEMO (zone [search dir.] : ', IZ, IR
        if ( IZ.lt.1.or.IZ.gt.NZONE ) then
            write(IPR,*) ' !!! zone # in KMEMO (=', IZ,
            & ' ) exceeds NZONE(=', MZONE ,') or less than 1.'
            goto 120
        end if
        if ( IR.lt.0.or.IR.gt.1 ) then
            write(IPR,*) ' !!! zone search order in KMEMO (=', IR,
            & ' must be 0 or 1.'
            goto 120
        end if

        call PKMEMO(IPR,IZ,IR,A(LKMEMO2),A(LMEMC2),A(LMEMZ2),
            & NZONE,NMEMO)

    end if
-----
MCNP ...

else if ( VNM(:NLEN).eq.'MCNP' ) then
    ICALL    = 1
    IMC       = 99
    call MCNOUT( IMC, IPR, IUG1, IUG2, IUB, IERR, LIMIT, A, A, A,
    & LIMITC, CHA, TITLE, CODE, A(LDBODI), A(LIBODI),
    & A(LIPSDA), A(LKINPZ), A(LKZDA), A(LKZAA), A(LSDA),
    & A(LKZMAT), A(LKMAT), A(LKSFB), CHA(LIZNAM), A(LIBSDA),
    & A(LKPRG), CHA(LKPRGN), A(LKPRGI), CHA(LTNAMS)

```

```

&      A(LIBSDA), NBODY, A(LIPSDA), NSURF, A(LSDA), NSDA
&      )
&      end if
&      end if
-----
.. KMEMO ...

else if ( VNM(:NLEN).eq.'KMEMO' ) then
  ICALL = 1
  NB = -2
  call I4READ( VNM, ITEMP, NA, NB, IERR )

  if ( NA.gt.0 ) then
    IZ = ITEMP(1)
    IR = 0
    if ( NA.gt.1 ) IR = ITEMP(2)
    write(IPR,*) ' KMEMO (zone [search dir.]) : ', IZ, IR
    if ( IZ.lt.1.or.IZ.gt.NZONE ) then
      write(IPR,*) ' !!! zone # in KMEMO (=', IZ,
&      ' ) exceeds NZONE(=', NZONE, ') or less than 1.'
      goto 120
    end if
    if ( IR.lt.0.or.IR.gt.1 ) then
      write(IPR,*) ' !!! zone search order in KMEMO (=', IR,
&      ' must be 0 or 1.'
      goto 120
    end if

    call PKMEMO(IPR, IZ, IR, A(LKMEMO2), A(LMEMC2), A(LMEMZ2),
&      NZONE, NMEMO)

  end if
-----
.. MCNP ...

else if ( VNM(:NLEN).eq.'MCNP' ) then
  ICALL = 1
  IMC = 99
  call MCNOUT( IMC, IPR, IUG1, IUG2, IUB, IERR, LIMIT, A, A, A,
&      LIMIT, CHA, TITLE, CODE, A(LDBODI), A(LIBODI),
&      A(LIPSDA), A(LKINPZ), A(LKZDA), A(LKZAA), A(LSDA),
&      A(LKZMAT), A(LKMAT), A(LKSFBDD), CHA(LIZNAM), A(LIBSDA),
&      A(LKREG), CHA(LKREGN), A(LKREGI), CHA(LTNAMS))

```

210

src/cgview/pablo1.f

```

1431: C
1432:          CWORK = PRNCOM(IPCOM)
1433: C
1434:          KK      = INDEX(CWORK,'%p')
1435:          if ( KK.gt.0 ) then
1436:             CWORK2 = CWORK(KK+3:)
1437:             write(CWORK(KK:KK+2),'(i3.3)') IPNUM0
1438:             CWORK(KK+3:) = CWORK2
1439:          end if
1440:
1441:          KK      = INDEX(CWORK,'%w')
1442:          if ( KK.gt.0 ) then
1443:             CWORK2 = CWORK(KK+3:)
1444:             write(CWORK(KK:KK+13),'(''0x'',z12.12)') IWID
1445:             CWORK(KK+14:) = CWORK2
1446:          end if
1447:
1448:          KK      = INDEX(CWORK,'%f')
1449:          if ( KK.gt.0.and.PRNNAM.ne.' ' ) then
1450:             CWORK2 = CWORK(KK+3:)
1451:             CWORK(KK:) = PRNNAM(:ICLEN2(PRNNAM)) //CWORK2
1452:          end if
1453:
1454:          LLL      = ICLEN2(CWORK)
1455:          write(IPR,*) '>>> Print: ', CWORK(:LLL)
1456:          call FTSYSTEM( ICODE, CWORK(:LLL), LLL )
1457:          write(IPR,*) '>>> Printed'
1458: C
1459:          if ( IPNUM0.gt.0 ) IPNUM = IPNUM0 + 1
1460:          else
1461:             write(IPR,*)
1462:             &          '>>> print command not defined or invalid command #'
1463:          end if
1464:          end if
1465: C
1466: C -----
1467: C .. image printout file name base
1468: C
1469:          else if ( VNM(:NLEN).eq.'PRNAME' ) then
1470:             if ( IBR.ne.0 ) then
1471:                call CHREAD( VNM, LINE1, ' ', NLEN, ITERM, IEND )
1472:                if ( NLEN.gt.0 ) PRNNAM = LINE1(:NLEN)
1473:                write(IPR,'(1X,' ' Print name base <'',A,'''>'')')
1474:                &          PRNNAM(:ICLEN2(PRNNAM))
1475:             end if
1476: C
1477: C -----
1478: C ... Debug ...
1479: C
1480:          else if ( VNM(:NLEN).eq.'DEBUG' .or. VNM(:NLEN).eq.'DB' ) then
1481:             ICALL = 1
1482:             NB    = -NCGVDB
1483:             call I4READ( VNM, JCGVDB(1), NA, NB, IERR )
1484: C
1485: C -----
1486: C ... Help mode ...
1487: C
1488:          else if ( VNM(:NLEN).eq.'HELP' .or. VNM(:NLEN).eq.'H'
1489:             &          .or. VNM(:NLEN).eq.'?' .or. VNM(:NLEN).eq.'HELP1'
1490:             &          .or. VNM(:NLEN).eq.'H1'
1491:             &          .or. VNM(:NLEN).eq.'HELP2' .or. VNM(:NLEN).eq.'H2'
1492:             &          .or. VNM(:NLEN).eq.'HELP3' .or. VNM(:NLEN).eq.'H3'
1493:             &          .or. VNM(:NLEN).eq.'HELP4' .or. VNM(:NLEN).eq.'H4'
1494:             &          .or. VNM(:NLEN).eq.'HELPA' .or. VNM(:NLEN).eq.'HW'
1495:             &          .or. VNM(:NLEN).eq.'HELP5' .or. VNM(:NLEN).eq.'H5' ) then

```

```

1496: C
1497:          ICALL = 1
1498:          call HELPPG( IPR, VNM(:NLEN) )
1499: C
1500: C -----
1501: C ... reset input record ...
1502: C
1503:          else if ( VNM(:NLEN).eq.'RESET' ) then
1504:             write(IPR,*) '=== RESET INPUT RECORD==='
1505:             call RESET
1506: C
1507: C -----
1508: C ... quit immediately ...
1509: C
1510:          else if ( VNM(:NLEN).eq.'QUIT' .or. VNM(:NLEN).eq.'Q' .or.
1511:             &          VNM(:NLEN).eq.'EXIT' ) then
1512:             write(IPR,*) '=== QUIT PROGRAM ==='
1513:             go to 260
1514: C
1515: C -----
1516: C ... ??? what is this ?
1517: C
1518:          else
1519:             write(IPR,*) '!!! unrecognizable input item <', VNM(:NLEN),
1520:             &          '>', ' or it lacks required "(...)"'
1521: C
1522: C ... check a vname is followed by '(' ...
1523: C
1524: CCCC call FPROBE( IBR, ' ', CH )
1525:          if ( IBR.ne.0 ) then
1526:             call DMREAD( VNM, IDUMMY, IDUMMY, IERR )
1527:          end if
1528:          write(IPR,*) '!!! skipped'
1529:          end if
1530: C
1531:          if ( JJEND.eq.0 ) go to 120
1532: C
1533: C
1534: C
1535:          220 continue
1536: C
1537: Ccc if ( ICALL.eq.0 ) go to 190
1538: C
1539: C
1540:          if ( IRR.ne.0 ) then
1541:             write(IPR,*) '!!! No. of input data is incorrect !!!'
1542:          end if
1543: C
1544: C ---- check scanmode ----
1545: C
1546: C JSCANX/Y = 3 ( bi-direction ) is not allowed when STYLE = 1
1547: C
1548:          JSCANX0 = JSCANX
1549:          if ( STYLE.eq.1.and.(JSCANX.eq.0.or.JSCANX.eq.3) ) then
1550:             JSCANX = 1
1551:          end if
1552: C
1553:          JSCANY0 = JSCANY
1554:          if ( STYLE.eq.1.and.(JSCANY.eq.0.or.JSCANY.eq.3) ) then
1555:             JSCANY = 1
1556:          end if
1557: C
1558:          XMIN = 0.0D0
1559:          YMIN = 0.0D0
1560: C

```

src/cgview/pablo1.f

```

1561: C ... normalize vectors
1562: C
1563:       X1      = SQRT(PAPER(4)**2+PAPER(5)**2+PAPER(6)**2)
1564:       if ( X1.ne.0.0D0 ) then
1565:           PAPER(4) = PAPER(4) /X1
1566:           PAPER(5) = PAPER(5) /X1
1567:           PAPER(6) = PAPER(6) /X1
1568:       end if
1569:
1570:       Y1      = SQRT(PAPER(7)**2+PAPER(8)**2+PAPER(9)**2)
1571:       if ( Y1.ne.0.0D0 ) then
1572:           PAPER(7) = PAPER(7) /Y1
1573:           PAPER(8) = PAPER(8) /Y1
1574:           PAPER(9) = PAPER(9) /Y1
1575:       end if
1576: C
1577:       if ( PAPER(10).ne.0.0D0 ) D1CG = 1.0D0/PAPER(10)
1578: C
1579: C ... coordinates of lower-right / upper-left corners
1580: C
1581:       PAPER(11) = PAPER(1) + PAPER(4)*XMAX
1582:       PAPER(12) = PAPER(2) + PAPER(5)*XMAX
1583:       PAPER(13) = PAPER(3) + PAPER(6)*XMAX
1584:
1585:       PAPER(14) = PAPER(1) + PAPER(7)*YMAX
1586:       PAPER(15) = PAPER(2) + PAPER(8)*YMAX
1587:       PAPER(16) = PAPER(3) + PAPER(9)*YMAX
1588: C
1589:       if ( JLATT.eq.0 ) LEVEL = 0
1590: C
1591:       ICHK1 = 0
1592: C
1593: C .....
1594: C
1595:       call ECHOPARAM( IPR, PAPER, XMAX, YMAX, JSCANX, JSCANY, SPTYP,
1596: & STYLE, STYLE2, LEVEL, INTREFRESH, MCELL, IMCELL,
1597: & JZREVERSE, GRANGE, JGFLOOD )
1598: C
1599: C ... save scene parameters if FORWARD/BACK command is not used
1600: C
1601:       if ( JFWBK.eq.0 ) then
1602:           call SAVEPARAM( IPR, IDSAVE, TITLGR, PAPER, XMAX, YMAX, JSCANX,
1603: & JSCANY, SPTYP, STYLE, STYLE2, LEVEL, INTREFRESH,
1604: & OFRAME, OVERSION, OLOSTSYM, MCELL, IMCELL, JZREVERSE,
1605: & GRANGE, JGFLOOD )
1606:       end if
1607: C
1608: C ... initalization for every drawing action
1609: C
1610:       do 230 I = 1, NZONE
1611:           IGMCK(I) = 0
1612:       230 continue
1613:       do 240 I = 1, NINPZ
1614:           NDEFSRC(I) = 0
1615:       240 continue
1616: C
1617:       NIPGMCK = 0
1618:       do 250 I = 1, MIPGMCK
1619:           XGMCK(4,I) = 0.0D0
1620:       250 continue
1621: C
1622: C
1623:       if ( IWON.eq.1 ) then
1624:           write(IPR,'(/lx,a/)')
1625:           & '#### Type any key in drawing window to draw next image ###'

```

```

1626:       end if
1627: C
1628:       return
1629: C
1630:       260 continue
1631:       ICHK1 = 1
1632:       return
1633:       end
1634: C
1635: C-----
1636: C
1637: C ===== SMALL TO CAPITAL CONVERSION =====
1638: C
1639:       subroutine CAPITAL( STR, LEN )
1640: C
1641:       character(*) STR
1642:       character*26 SMALL, CAPITL
1643:       data SMALL /'abcdefghijklmnopqrstuvwxyz'/
1644:       data CAPITL /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
1645: C
1646:       do 100 I = 1, LEN
1647:           K = INDEX(SMALL,STR(I:I))
1648:           if ( K.ne.0 ) STR(I:I) = CAPITL(K:K)
1649:       100 continue
1650: C
1651:       return
1652:       end
1653: C
1654: C=====
1655: C Routines to set default colormap
1656: C=====
1657: C
1658:       subroutine ZCLR00( ZONCLR,NZONE, NCC )
1659:       integer ZONCLR(NZONE)
1660:       do 100 I = 1, NZONE
1661:           ZONCLR(I) = MOD(I,NCC) + 1
1662:       100 continue
1663:       return
1664:       end
1665: C
1666:       subroutine IZCLR00( IZNCLR,NINPZ, NCC )
1667:       integer IZNCLR(NINPZ)
1668:       do 100 I = 1, NINPZ
1669:           IZNCLR(I) = MOD(I,NCC) + 1
1670:       100 continue
1671:       return
1672:       end
1673: C
1674:       subroutine RCLR00( REGCLR,NREG, NCC )
1675:       integer REGCLR(NREG)
1676:       do 100 I = 1, NREG
1677:           REGCLR(I) = MOD(I,NCC) + 1
1678:       100 continue
1679:       return
1680:       end
1681: C
1682:       subroutine MCLR00( MATCLR,MXMAT, NCC )
1683:       integer MATCLR(0:MXMAT)
1684:       MATCLR(0) = 1
1685:       do 100 I = 1, MXMAT
1686:           MATCLR(I) = MOD(I,NCC) + 1
1687:       100 continue
1688:       return
1689:       end
1690: C

```

src/cgview/pablo1.f

```
1691: C=====
1692: C
1693:      subroutine FORGET( KMEMO2, MEMC2, MEMZ2, NZONE, NMEMO )
1694: C
1695:      integer KMEMO2(NZONE,NMEMO,0:1)
1696:      integer MEMC2(NZONE,NMEMO,0:1)
1697:      integer MEMZ2(NZONE,0:1)
1698:      do 130 K = 0, 1
1699:          do 110 J = 1, NMEMO
1700:              do 100 I = 1, NZONE
1701:                  KMEMO2(I,J,K) = 0
1702:                  MEMC2(I,J,K) = 0
1703:              100 continue
1704:          110 continue
1705:          do 120 I = 1, NZONE
1706:              MEMZ2(I,K) = 0
1707:              MEMZ2(I,K) = 0
1708:          120 continue
1709:      130 continue
1710:      return
1711:      end
1712: C
1713: C=====
1714: C
1715:      subroutine PKMEMO(IPR, IZ, IR, KMEMO2, MEMC2, MEMZ2, NZONE, NMEMO )
1716: C
1717:      integer KMEMO2(NZONE,NMEMO,0:1)
1718:      integer MEMC2(NZONE,NMEMO,0:1)
1719:      integer MEMZ2(NZONE,0:1)
1720: C
1721:      write(IPR,7000) IZ,IR, MEMZ2(IZ,IR)
1722:      7000 format(lx,'>> Next zone memory : zone =',i6,' serach direction ',
1723:      & IZ/
1724:      & lx,'>> used entry so far for this zone ',I4/
1725:      & lx,'>> memorized-zone (count) ...')
1726:      do 100 K=1, MEMZ2(IZ,IR), 5
1727:          K2 = min( K+5-1, MEMZ2(IZ,IR))
1728:          write(IPR,7100) (KMEMO2(IZ,I,IR), MEMC2(IZ,I,IR), I=K, K2)
1729:      100 continue
1730:      7100 format(lx,'>>> ',5(I5,' (' ,i6,')':))
1731:      return
1732:      end
```

src/cgview/paper.f

```

1: C
2: C -----
3: C
4: C .... Zoom IN/OUT of drawing window.
5: C
6:       subroutine ZOOMWIN( ZOOM, X,      Y,      XMAX, YMAX, PAPER )
7: C
8:       implicit real*8(A-H,O-Z)
9:       real*8 PAPER(9)
10:      real*8 XMAX, YMAX
11: C
12:      call NORMVEC( PAPER(4), PAPER(5), PAPER(6) )
13:      call NORMVEC( PAPER(7), PAPER(8), PAPER(9) )
14: C
15:      ZOOM = ABS(ZOOM)
16:      if ( ZOOM.ne.0.0 ) then
17:          XMAX1 = XMAX/ZOOM
18:          YMAX1 = YMAX/ZOOM
19:          DX = XMAX*X - XMAX1*0.5
20:          DY = YMAX*Y - YMAX1*0.5
21:          PAPER(1) = PAPER(1) + DX*PAPER(4) + DY*PAPER(7)
22:          PAPER(2) = PAPER(2) + DX*PAPER(5) + DY*PAPER(8)
23:          PAPER(3) = PAPER(3) + DX*PAPER(6) + DY*PAPER(9)
24:          XMAX = XMAX1
25:          YMAX = YMAX1
26:      end if
27: C
28:      return
29:      end
30: C
31: C -----
32: C
33: C .... move drawing window.
34: C
35:       subroutine MOVEWIN( DX,      DY,      DZ,      XMAX, YMAX, PAPER )
36:       implicit real*8(A-H,O-Z)
37:       real*8 PAPER(9)
38:       real*8 XMAX, YMAX
39: C
40:       call NORMVEC( PAPER(4), PAPER(5), PAPER(6) )
41:       call NORMVEC( PAPER(7), PAPER(8), PAPER(9) )
42: C
43:       call VECXVEC( CX, CY, CZ, PAPER(4), PAPER(5), PAPER(6), PAPER(7),
44:       & PAPER(8), PAPER(9) )
45: C
46:       PAPER(1) = PAPER(1) + DX*XMAX*PAPER(4) + DY*YMAX*PAPER(7)
47:       & + DZ*XMAX*CX
48:       PAPER(2) = PAPER(2) + DX*XMAX*PAPER(5) + DY*YMAX*PAPER(8)
49:       & + DZ*XMAX*CY
50:       PAPER(3) = PAPER(3) + DX*XMAX*PAPER(6) + DY*YMAX*PAPER(9)
51:       & + DZ*XMAX*CZ
52: C
53:       return
54:       end
55: C
56: C -----
57: C
58: C .... move drawing window (absolute displacement).
59: C
60:       subroutine MOVEAWIN( DX,      DY,      DZ,      PAPER )
61:       implicit real*8(A-H,O-Z)
62:       real*8 PAPER(9)
63: C
64:       call NORMVEC( PAPER(4), PAPER(5), PAPER(6) )
65:       call NORMVEC( PAPER(7), PAPER(8), PAPER(9) )

```

```

66: C
67:       call VECXVEC( CX, CY, CZ, PAPER(4), PAPER(5), PAPER(6), PAPER(7),
68:       & PAPER(8), PAPER(9) )
69: C
70:       PAPER(1) = PAPER(1) + DX*PAPER(4) + DY*PAPER(7) + DZ*CX
71:       PAPER(2) = PAPER(2) + DX*PAPER(5) + DY*PAPER(8) + DZ*CY
72:       PAPER(3) = PAPER(3) + DX*PAPER(6) + DY*PAPER(9) + DZ*CZ
73: C
74:       return
75:       end
76: C
77: C -----
78: C
79: C ... resize window (relative size):
80: C
81:       subroutine RESIZEWIN( DX,      DY,      JAR,      IC,      XMAX, YMAX,
82:       & PAPER )
83: C
84:       dx, dy : resizing factor
85:       jar : 0 = dx/dy is relative size
86:       ic : corner to extend/shrink
87: C
88:       ic = 0 : center
89:       ic = 1 : lower-left corner
90:       ic = 2 : lower-right corner
91:       ic = 3 : upper-left corner
92:       ic = 4 : upper-right corner
93: C
94:       implicit real*8(A-H,O-Z)
95:       real*8 PAPER(9)
96: C
97:       call NORMVEC( PAPER(4), PAPER(5), PAPER(6) )
98:       call NORMVEC( PAPER(7), PAPER(8), PAPER(9) )
99: C
100:      XMAX2 = XMAX
101:      YMAX2 = YMAX
102:      if ( JAR.eq.0 ) then
103:          if ( DX.ne.0.0D0 ) XMAX2 = XMAX*ABS(DX)
104:          if ( DY.ne.0.0D0 ) YMAX2 = YMAX*ABS(DY)
105:      else
106:          if ( DX.ne.0.0D0 ) XMAX2 = ABS(DX)
107:          if ( DY.ne.0.0D0 ) YMAX2 = ABS(DY)
108:      end if
109: C
110:      DDX = 0
111:      DDY = 0
112:      if ( IC.eq.0 ) then
113:          DDX = (XMAX-XMAX2)*0.5D0
114:          DDY = (YMAX-YMAX2)*0.5D0
115:      else if ( IC.eq.1 ) then
116:          DDX = XMAX - XMAX2
117:          DDY = YMAX - YMAX2
118:      else if ( IC.eq.2 ) then
119:          DDX = 0
120:          DDY = YMAX - YMAX2
121:      else if ( IC.eq.3 .or. IC.eq.4 ) then
122:          DDX = 0
123:          DDY = 0
124:      end if
125: C
126:      PAPER(1) = PAPER(1) + DDX*PAPER(4) + DDY*PAPER(7)
127:      PAPER(2) = PAPER(2) + DDX*PAPER(5) + DDY*PAPER(8)
128:      PAPER(3) = PAPER(3) + DDX*PAPER(6) + DDY*PAPER(9)
129:      XMAX = XMAX2
130:      YMAX = YMAX2

```

src/cgview/paper.f

```

131: C
132:     return
133:     end
134: C
135: C-----
136: C
137: C ... rotate window
138: C
139:     subroutine ROTATEWIN( MODE, DEG, POS, POS2, XMAX, YMAX,
140: &                         PAPER )
141:     implicit real*8(A-H,O-Z)
142:     character*(*) MODE
143:     real*8 PAPER(9)
144:     real*8 XMAX, YMAX
145: C
146:     call NORMVEC( PAPER(4), PAPER(5), PAPER(6) )
147:     call NORMVEC( PAPER(7), PAPER(8), PAPER(9) )
148: C
149: C ... "deg" degree rotation round a line which passes (x0,y0,z0)
150: C and whose direction is (rx,ry,rz).
151: C
152: C ... x-rotation ...
153:     if ( MODE.eq.'ROTX' ) then
154:         RX = PAPER(4)
155:         RY = PAPER(5)
156:         RZ = PAPER(6)
157:         X0 = PAPER(1) + PAPER(7)*YMAX*POS
158:         Y0 = PAPER(2) + PAPER(8)*YMAX*POS
159:         Z0 = PAPER(3) + PAPER(9)*YMAX*POS
160: C ... y-rotation ...
161:     else if ( MODE.eq.'ROTY' ) then
162:         RX = PAPER(7)
163:         RY = PAPER(8)
164:         RZ = PAPER(9)
165:         X0 = PAPER(1) + PAPER(4)*XMAX*POS
166:         Y0 = PAPER(2) + PAPER(5)*XMAX*POS
167:         Z0 = PAPER(3) + PAPER(6)*XMAX*POS
168: C ... z-rotation ...
169:     else if ( MODE.eq.'ROT' ) then
170:         call VECXVEC( RX, RY, RZ, PAPER(4), PAPER(5), PAPER(6),
171: &                     PAPER(7), PAPER(8), PAPER(9) )
172:         call NORMVEC( RX, RY, RZ )
173:         X0 = PAPER(1) + PAPER(4)*XMAX*POS + PAPER(7)*YMAX*POS2
174:         Y0 = PAPER(2) + PAPER(5)*XMAX*POS + PAPER(8)*YMAX*POS2
175:         Z0 = PAPER(3) + PAPER(6)*XMAX*POS + PAPER(9)*YMAX*POS2
176:     end if
177: C
178:     TH = DEG/180.0D0*3.1415926535897932D0
179: C
180:     DX = PAPER(1) - X0
181:     DY = PAPER(2) - Y0
182:     DZ = PAPER(3) - Z0
183:     call ROTVEC( DX2, DY2, DZ2, DX, DY, DZ, RX, RY, RZ, TH )
184:     PAPER(1) = X0 + DX2
185:     PAPER(2) = Y0 + DY2
186:     PAPER(3) = Z0 + DZ2
187: C
188:     call ROTVEC( DX2, DY2, DZ2, PAPER(4), PAPER(5), PAPER(6), RX, RY,
189: &                 RZ, TH )
190:     PAPER(4) = DX2
191:     PAPER(5) = DY2
192:     PAPER(6) = DZ2
193: C
194:     call ROTVEC( DX2, DY2, DZ2, PAPER(7), PAPER(8), PAPER(9), RX, RY,
195: &                 RZ, TH )
196:     PAPER(7) = DX2
197:     PAPER(8) = DY2
198:     PAPER(9) = DZ2
199: C
200:     return
201:     end

```


src/cgview/pifext.c

```
1: /*****
2:  * Additional functions to extend cgview_slice 2.0 on piflib.
3:  *
4:  * From Nov 1995 : Mako Sasaki (J.R.I.)
5:  *****/
6: #include <X11/Xlib.h>
7: #include <X11/Xutil.h>
8: #include <stdio.h>
9: #include "piflib.h"
10:
11: /*****
12:  * Get pixel size in the same unit as plot()
13:  *****/
14:
15: #ifdef _NEED_UBAR
16: void pixsize( x_pm, y_pm )
17: #elif _UPPERCASE
18: void PIXSIZE( x_pm, y_pm )
19: #else
20: void pixsize( x_pm, y_pm )
21: #endif
22: {
23:     double *x_pm ;    /** X length per pixel */
24:     double *y_pm ;    /** Y length per pixel */
25:
26:     /** if output is PS/EPS */
27:     #define PS_PIXSIZE_MM (72.0/28.4)
28:     if ( output_device == 2 || output_device == 3 ) {
29:         *x_pm = 1.0/(double)(f_value * PS_PIXSIZE_MM * 2 ) ;
30:         *y_pm = 1.0/(double)(f_value * PS_PIXSIZE_MM * 2 ) ;
31:     } else {
32:         *x_pm = 1.0/(double)( f_value * x_pix_mm ) ;
33:         *y_pm = 1.0/(double)( f_value * y_pix_mm ) ;
34:     }
35: }
```

src/cgview/preinp.f

```

1:      subroutine PREINP( IPREIN,CSTRNG,IDEVICE,PSFILE, MLIMIT, CODE )
2: C=<CGVIEW>=====
3: C   Purpose: Interpret command string for file-name/I/O-unit coupling
4: C   or default-option overriding etc.
5: C   Probably called from main routine before calling of 'CENTER'
6: C   routine, or called before reading input to CGVIEW.
7: C
8: C=====
9: C   argument :
10: C
11: C i iprein : counter of calling for this routine given externally.
12: C   (when iprein = 1 , initialize data )
13: C i cstr : command string
14: C
15: C   /nn[rful][:file-name]
16: C
17: C   Assign I/O unit nn to a file.
18: C
19: C   xxx=yyy
20: C
21: C   Execution parameter setting.
22: C
23: C o  idevice : initial plot device (1/2/3= display/Postscript/EPS)
24: C o  psfile  : Postscript/EPS file name
25: C o  mlimit  : memory size limit (effective only in dynamic allocation
26: C   mode )
27: C
28: C=====
29: C
30: C   include '../shared/INC/_IOUNIT'
31: C
32: C   ... external data ...
33: C
34: C   character*(*) CSTRNG
35: C   character*(*) PSFILE
36: C   character*(*) CODE
37: C
38: C   include 'INC/_FLAGS'
39: C
40: C   ... local data ...
41: C
42: C   character*20 CTEMP
43: C
44: C-----
45: C
46: C   ... initialization on first call ...
47: C
48: C   if ( IPREIN.eq.1 ) then
49: C     JRPCPU = 0
50: C     code = 'MVP'
51: C   end if
52: C
53: C
54: C   write(IPR,'(1x,a,a,a)') ' PARAMETER : <', CSTRNG, '>'
55: C
56: C   IE      = 0
57: C
58: C   .... 'CSTRNG' may be broken into some commands separated by blank.
59: C
60: C   100 IS      = IE + 1
61: C   call NOBLNK( CSTRNG, IS, IE, LEN(CSTRNG) )
62: C
63: C   ... cstrng(is:ie) : Command .....
64: C
65: C   if ( IE.le.LEN(CSTRNG) ) then

```

```

66: C
67: C-----
68: C   File allocation
69: C-----
70: C   if ( CSTRNG(IS:IS).eq.'/' ) then
71: C
72: C
73: C       JJJJJ = JFOPEN(CSTRNG(IS:IE))
74: C
75: C-----
76: C   Postscript or EPS file name
77: C-----
78: C
79: C   else if ( CSTRNG(IS:IS+2).eq.'ps='
80: C   & .or. CSTRNG(IS:IS+2).eq.'PS=' ) then
81: C
82: C       if( ie.gt.is+2.and.cstrng(is+3:ie).ne.' ' ) then
83: C         psfile = cstrng(is+3:ie)
84: C       end if
85: C
86: C-----
87: C   output device type
88: C-----
89: C
90: C   else if ( CSTRNG(IS:IS+1).eq.'d='
91: C   & .or. CSTRNG(IS:IS+1).eq.'D=' ) then
92: C
93: C       ctemp = cstrng(is+2:ie)
94: C
95: C   ... display (currently X Window System)
96: C   if ( ctemp.eq.'X'.or.ctemp.eq.'x'.or.ctemp.eq.' ' ) then
97: C     idevice = 1
98: C
99: C   ... Postscript
100: C   else if ( ctemp(1:1).eq.'P'.or.ctemp(1:1).eq.'p' ) then
101: C     idevice = 2
102: C
103: C   ... Encapsulated Postscript (EPS)
104: C   else if ( ctemp(1:1).eq.'E'.or.ctemp(1:1).eq.'e' .or.
105: C   & ctemp(1:1).eq.'T'.or.ctemp(1:1).eq.'t' ) then
106: C     idevice = 3
107: C   else
108: C     write(IPR,'(1x,a)')
109: C     & 'xxx Unsupported output device : ',cstrng(is:ie)
110: C     write(IPR,'(1x,a)')
111: C     & ' Output on display.'
112: C     idevice = 1
113: C   end if
114: C
115: C-----
116: C   memory size
117: C-----
118: C
119: C   C/#IF DYNAMIC
120: C   else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
121: C     CTEMP = CSTRNG(IS+7:IE)
122: C     read(CTEMP(:20),'(bn,i20)') MLIMIT
123: C     go to 110
124: C   C/#ELSE
125: C   else if ( IMATCH('MEMORY=*',CSTRNG(IS:IE)).eq.1 ) then
126: C     write(IPR,'(1x,a)')
127: C     & '<MEMORY=...> has meanings in dynamic memory mode.'
128: C     go to 110
129: C   C/#ENDIF
130: C-----

```

src/cgview/preinp.f

```
131: C      code for which the input data is created ( MVP ot GMVP )
132: C
133: C  Unfortunately, there is no way to clearly distinguish input for
134: C  MVP and GMVP ...
135: C -----
136: C
137: C      else if ( CSTRNG(IS:IS+1).eq.'c='
138: C      &      .or. CSTRNG(IS:IS+1).eq.'C=' ) then
139: C          code = cstrng(is+2:ie)
140: C          write(IPR,*) 'input is specified as  input to <',code,
141: C      &      '> code.'
142: C -----
143: C      unsupported command
144: C -----
145: C
146: C      else
147: C          write(IPR,'(1x,a)') ' !!! This command is not supported.'
148: C      end if
149: C
150: C 110      continue
151: C      go to 100
152: C  end if
153: C
154: C      return
155: C  end
```

src/cgview/prgmck.f

```

1:      subroutine PRGMCK( IOW,IGMCK,IPGMCK,XGMCK,NIPGMCK, MIPGMCK,IZNAM,
2:      & KINPZ, NZONE, NINPZ )
3: C=====
4: C Purpose: output summary of geometry overlap check
5: C=====
6:      integer IGMCK(NZONE)
7:      integer IPGMCK(3,*)
8:      real*8 XGMCK(4,*)
9: C
10:     integer KINPZ(NZONE)
11:     character*12 IZNAM(NINPZ)
12: C
13: CC parameter( MZLINE = 2 )
14: CC integer ITEMP(2,MZLINE)
15: C
16: C-----
17: C
18:     KK = 0
19:     do I = 1, NZONE
20:         KK = KK + IGMCK(I)
21:     end do
22:     if ( KK.eq.0 ) return
23: C
24:     write(IOW,7000)
25: 7000 format(/1X,'>>> *** Overlapping input-zones are found ***'/
26: & 1X,'>>> Input-zone overlap is harmless when',
27: & ' identical material and region are'/
28: & 1X,'>>> assigned to overlapping input-zones.'/
29: & 1X,'>>> If not so, it indicates invalid geometry input!!'/
30: & 1X,'>>> Please check the following zone(input-zone) pairs. //)
31: C
32:     KOVMAX = 0
33:     do KOV = 0, 3
34:         KK0 = 0
35:         KK = 0
36:         do I = 1, min(NIPGMCK,MIPGMCK)
37:             if ( IPGMCK(3,I).eq.KOV ) then
38:                 if ( KK0.eq.0 ) then
39:                     KK0 = 1
40:                     if ( KOV.eq.0 ) then
41:                         write(IOW,*) '>>> === Input-zone overlap ==='
42:                     else if ( KOV.eq.1 ) then
43:                         write(IOW,*) '>>> === Region overlap ==='
44:                     else if ( KOV.eq.2 ) then
45:                         write(IOW,*) '>>> === Material overlap ==='
46:                     else if ( KOV.eq.3 ) then
47:                         write(IOW,*)
48: & '>>> === Material and region overlap ==='
49:                     end if
50:                 end if
51:             end if
52:         end do
53:         KOVMAX = max(KOV,KOVMAX)
54:
55:         write(IOW,7020)
56: & IPGMCK(1,I),KINPZ(IPGMCK(1,I)),
57: & IZNAM(KINPZ(IPGMCK(1,I)):(ICLEN2(IZNAM(KINPZ(IPGMCK(1,I)))))),
58: & IPGMCK(2,I),KINPZ(IPGMCK(2,I)),
59: & IZNAM(KINPZ(IPGMCK(2,I)):(ICLEN2(IZNAM(KINPZ(IPGMCK(2,I)))))),
60: & XGMCK(4,I),XGMCK(1,I),XGMCK(2,I),XGMCK(3,I)
61:     end if
62: end do
63: 7020 format(1X,'>>> ',
64: & '(',I4,'(',I4,'<',A,'>',I4,'(',I4,'<',A,'>)',',',
65: & ' Overlap length max.=' ,1p,e13.5/

```

```

66:     & 1X,'>>> position',1p,3e13.5)
67: C
68:     if ( KOVMAX.eq.0 ) then
69:         write(IOW,7040)
70: 7040 format(/1X,'>>> No material/region overlap is found.'/
71: & 1X,'>>> If you don't like to see the input-zone overlap',
72: & ' messages, you can use'/
73: & 1X,'>>> "OR" operator to combine input-zones',
74: & ' into a single input-zone.'/)
75:     end if
76: C
77:     return
78: end

```

src/cgview/prsize.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine PRSIZE
3: C=====
4: C  PURPOSE:  PRINTOUT OF COMMON /SIZES/
5: C  CALLED IN:  INTRO
6: C=====
7:   include  '../shared/INC/_SIZES'
8:   include  '../shared/INC/_IUNIT'
9: C
10:  write(IPR,7000)
11: 7000 format('1',130('='))' SIZE- OR LIMIT-PARAMETERS ',
12: & '(INCLUDES PARAMETERS WHOSE VALUES ARE GIVEN BY THE CODE)'
13: & '/1X ,130('='))
14: C
15:  write(IPR,7100)
16: & NPART , NHIST , NBANK , NZONE , NINPZ , NSURF , NSDA,
17: & NZDA , NMEMO
18: 7100 format(
19: &/3X,' NPART ',I9,3X,'TOTAL NUMBER OF PARTICLES '
20: &/3X,' NHIST ',I9,3X,'NUMBER OF PARTICLES RUN IN ONE BATCH '
21: &/3X,' NBANK ',I9,3X,'PARTICLE BANK LENGTH '
22: &/3X,' NZONE ',I9,3X,'ZONE NUMBER '
23: &/3X,' NINPZ ',I9,3X,'NUMBER OF INPUT ZONE '
24: &/3X,' NSURF ',I9,3X,'TOTAL SURFACE NUMBER '
25: &/3X,' NSDA ',I9,3X,'TOTAL NUMBER OF SURFACE DATA '
26: &/3X,' NZDA ',I9,3X,'TOTAL NUMBER OF ZONE DATA '
27: &/3X,' NMEMO ',I9,3X,'LENGTH OF NEXT-ZONE MEMORY ARRAY ' )
28: C
29:  write(IPR,7200)
30: & NREG
31: 7200 format(
32: &/3X,' NREG ',I9,3X,'EDIT REGION NUMBER ' )
33: C
34:  write(IPR,7300)
35: & NREFS , NMEMS ,
36: & NMEMOP
37: 7300 format(
38: &/3X,' NREFS ',I9,3X,'NUMBER OF REFLECTIVE SURFACES '
39: &/3X,' NMEMS ',I9,3X,'MAXIMUM NUMBER OF STARTING ZONES '
40: & ',FOR EACH PARTICLE SOURCE '
41: &/3X,' NMEMOP',I9,3X,'OPTIMIZE KMEMO ARRAY WHILE '
42: & ',NBATCH .LE. NMEMOP (DEFAULT=2)' )
43: C
44:  write(IPR,7400)
45: & NLATT , NCELL , NEST , NLBZ
46: 7400 format(
47: &/3X,' NLATT ',I9,3X,'NUMBER OF LATTICE '
48: &/3X,' NCELL ',I9,3X,'NUMBER OF CELLS '
49: &/3X,' NEST ',I9,3X,'MAXIMUM NESTING LEVEL IN '
50: & ',LATTICE GEOMETRY '
51: &/3X,' NLBZ ',I9,3X,'NUMBER OF LATTICE-BUFFER ZONES ')
52: C
53:  return
54:  end
```

src/cgview/psdraw.c

```

1: /*****
2: *
3: *   psdraw.c : Drawing Routines for Post Script File
4: *
5: *****/
6: /*****
7: update:
8:   04 Mar 2004: change exit() to exit(0) in psplot. (Y.Nagaya)
9:   21 Sep 1998: add include <string.h>
10: *****/
11:
12: #include <X11/Xlib.h>
13: #include <X11/Xutil.h>
14: #include <stdio.h>
15: #include <string.h>
16: #include "piflib.h"
17:
18: #define XORIG 20.0
19: #define YORIG 17.0
20:
21: int page_number ;
22: int begin_page ;
23:
24: psplots( ps_file )
25:   char *ps_file ;
26: {
27:   static char *outfile = { "piflib.PS" };
28:   static char *ps[] = {
29:     "%!PS-Adobe-2.0",
30:     "%%Creator: piflib",
31:     "%%EndComments",
32:     "/pifdict 30 dict def",
33:     "pifdict begin",
34:     "/Color true def",
35:     "/DC {Color {setrgbcolor} {pop pop pop 0 setgray} ifelse} def",
36:     "/LC0 { stroke 0 0 0 DC } def",
37:     "/LC1 { stroke 1 1 1 DC } def",
38:     "/LC2 { stroke 1 0.1 0.1 DC } def",
39:     "/LC3 { stroke 0.4 1 0.1 DC } def",
40:     "/LC4 { stroke 0.1 0.2 1 DC } def",
41:     "/LC5 { stroke 0.3 1 1 DC } def",
42:     "/LC6 { stroke 1 0.2 1 DC } def",
43:     "/LC7 { stroke 1 1 0.1 DC } def",
44:     "/LT1 { stroke [5 5] 0 setdash } def",
45:     "/LT2 { stroke [10 5] 0 setdash } def",
46:     "/LT3 { stroke [10 5 5 5] 0 setdash } def",
47:     "/LT4 { stroke [10 5 5 5 5 5] 0 setdash } def",
48:     "/LT5 { stroke [10 5 5 5 5 5 5] 0 setdash } def",
49:     "/LT6 { stroke [10 5 2 5] 0 setdash } def",
50:     "/LT7 { stroke [10 5 2 2 5] 0 setdash } def",
51:     "/LT8 { stroke [10 5 2 2 2 5] 0 setdash } def",
52:     "/LT9 { stroke [10 5 8 5 2 5 8 5] 0 setdash } def",
53:     "/LT10 { stroke [2 2] 0 setdash } def",
54:     "/LT11 { stroke [2 2 2 2 10] 0 setdash } def",
55:     "/M {moveto} def",
56:     "/L {lineto} def",
57:     "end",
58:     "%%EndInitialize",
59:     NULL
60:   };
61:
62:   long k;
63:   int i;
64:   char filename[81];
65:   char c;

```

```

66:
67: /* x_pix_mm = 2.834645676; */ /** 72 dpi */
68: /* y_pix_mm = 2.834645676; */
69: x_pix_mm = 11.3385827; /** 288 dpi */
70: y_pix_mm = 11.3385827;
71:
72: ps_xorg = XORIG;
73: ps_yorg = YORIG;
74:
75: ps_plot = 0;
76: f_value = 1.0;
77:
78: for( i = 0 ; i < 81 ; i++ ) {
79:   filename[i] = '\0';
80: }
81:
82: /*****
83:   if( !ifxpif ) {
84:     printf("Input filename to output( for PostScript )\n");
85:     printf("(Default : piflib.PS) >>:");
86:     i = 0;
87:     while((c = getchar()) != '\n') {
88:       filename[i] = c;
89:       i++;
90:     }
91:   }
92:   *****/
93:
94:   if( ps_file != NULL && strlen(ps_file) > 0 ) strcpy( filename, ps_file );
95:
96:   if(filename[0] == '\0'){
97:     if ( (fp = fopen(outfile, "w")) == NULL ) {
98:       printf("Can not Open File: %s !!\n",outfile);
99:       exit(1);
100:     }
101:   }else{
102:     if ( (fp = fopen(filename, "w")) == NULL ) {
103:       printf("Can not Open Output File: %s !!\n",filename);
104:       exit(1);
105:     }
106:   }
107:
108:   for ( k = 0 ; ps[k] != NULL ; k++ ) {
109:     fprintf( fp, "%s\n", ps[k] );
110:   }
111:   page_number = 1;
112:   fprintf( fp, "%%%Page: %d %d\n", 1, page_number );
113: }
114:
115: psplot( x_point1, y_point1, iflag )
116:   float x_point1, y_point1;
117:   int iflag;
118: {
119:   float xxx, yyy;
120:   /**float ps_x, ps_y; */
121:   /** M.Sasaki 951114 */
122:   static float ps_x, ps_y;
123:
124:   xxx = x_point1;
125:   yyy = y_point1;
126:
127:   x_store = xxx;
128:   y_store = yyy;
129:
130:   xxx *= f_value;

```

src/cgview/psdraw.c

```

131:     yyy *= f_value;
132:
133:     if( iflag == 999 ) {
134:         if(ps_plot) {
135:             fprintf(fp, "gsave\n");
136:             fprintf(fp, "stroke\n");
137:             fprintf(fp, "grestore\n");
138:             fprintf(fp, "end\n");
139:             fprintf(fp, "showpage\n");
140:         }
141:         fclose(fp);
142:         exit(0);
143:     }else if( iflag == 888 || iflag == 777 ) {
144:         ;
145:     }else if( iflag == 666 ) {
146:         fprintf(fp, "gsave\n");
147:         fprintf(fp, "stroke\n");
148:         fprintf(fp, "grestore\n");
149:         fprintf(fp, "end\n");
150:         fprintf(fp, "showpage\n");
151:         page_number++;
152:         fprintf(fp, "%%%Page: %d %d\n",1,page_number);
153:
154:         ifplot3 = 0;
155:         ps_plot = 0;
156:         ps_xorg = XORIG;
157:         ps_yorg = YORIG;
158:     }else if( iflag == 2 ) {
159:         ps_x = (ps_xorg + xxx)*x_pix_mm;
160:         ps_y = (ps_yorg + yyy)*y_pix_mm;
161:
162:         if(!ps_plot) {
163:             fprintf(fp, "pifdict begin\n");
164:             fprintf(fp, "gsave\n");
165:             fprintf(fp, "newpath\n");
166:             fprintf(fp, "grestore\n");
167:             fprintf(fp, "0 839 translate\n");
168:             fprintf(fp, "-90 rotate\n");
169:             fprintf(fp, ".25 .25 scale\n");
170:             if(!ifplot3) {
171:                 fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
172:                     (long)(ps_yorg*y_pix_mm) );
173:             }
174:         }
175:         fprintf(fp, "%d %d L\n", (long)ps_x, (long)ps_y);
176:         ps_plot = 1;
177:     }else if( iflag == 3 ) {
178:         ps_x = (ps_xorg + xxx)*x_pix_mm;
179:         ps_y = (ps_yorg + yyy)*y_pix_mm;
180:
181:         if(!ps_plot) {
182:             fprintf(fp, "pifdict begin\n");
183:             fprintf(fp, "gsave\n");
184:             fprintf(fp, "newpath\n");
185:             fprintf(fp, "grestore\n");
186:             fprintf(fp, "0 839 translate\n");
187:             fprintf(fp, "-90 rotate\n");
188:             fprintf(fp, ".25 .25 scale\n");
189:         }
190:         fprintf(fp, "%d %d M\n", (long)ps_x, (long)ps_y);
191:         ps_plot = 1;
192:         ifplot3 = 1;
193:     }else if( iflag == -2 ) {
194:         ps_xorg += xxx;
195:         ps_yorg += yyy;
196:
197:         ps_x = ps_xorg*x_pix_mm;
198:         ps_y = ps_yorg*y_pix_mm;
199:
200:         if(!ps_plot) {
201:             fprintf(fp, "pifdict begin\n");
202:             fprintf(fp, "gsave\n");
203:             fprintf(fp, "newpath\n");
204:             fprintf(fp, "grestore\n");
205:             fprintf(fp, "0 839 translate\n");
206:             fprintf(fp, "-90 rotate\n");
207:             fprintf(fp, ".25 .25 scale\n");
208:             if(!ifplot3) {
209:                 fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
210:                     (long)(ps_yorg*y_pix_mm) );
211:             }
212:         }
213:         fprintf(fp, "%d %d L\n", (long)ps_x, (long)ps_y);
214:         ps_plot = 1;
215:     }else if( iflag == -3 ) {
216:         ps_xorg += xxx;
217:         ps_yorg += yyy;
218:
219:         ps_x = ps_xorg*x_pix_mm;
220:         ps_y = ps_yorg*y_pix_mm;
221:
222:         if(!ps_plot) {
223:             fprintf(fp, "pifdict begin\n");
224:             fprintf(fp, "gsave\n");
225:             fprintf(fp, "newpath\n");
226:             fprintf(fp, "grestore\n");
227:             fprintf(fp, "0 839 translate\n");
228:             fprintf(fp, "-90 rotate\n");
229:             fprintf(fp, ".25 .25 scale\n");
230:         }
231:
232:         fprintf(fp, "%d %d M\n", (long)ps_x, (long)ps_y);
233:         ps_plot = 1;
234:         ifplot3 = 1;
235:     }
236:     ps_store_x = ps_x;
237:     ps_store_y = ps_y;
238: }
239:
240: psdot( x1, y1 )
241: float *x1, *y1;
242: {
243:     float x_dot, y_dot;
244:     float xxx, yyy;
245:
246:     xxx = *x1;
247:     yyy = *y1;
248:
249:     x_dot = (ps_xorg + xxx)*x_pix_mm;
250:     y_dot = (ps_yorg + yyy)*y_pix_mm;
251:
252:     ps_plot = 1;
253: }
254:
255: psplotlines( nml, xpoints, ypoints )
256: int nml;
257: float *xpoints, *ypoints;
258: {
259:     int i;
260:     psplot( xpoints[0], ypoints[0], 3 ) ;

```

src/cgview/psdraw.c

```
261:     for ( i=1 ; i < nml ; i++ ) {  
262:         psplot( xpoints[i], ypoints[i], 2 ) ;  
263:     }  
264: }  
265:
```

SAFE

src/cgview/pssetattr.c

```

1: /*****
2: /*
3: /*      pssetattr.c : Set Attribute Routines for PostScript Output      */
4: /*
5: /*****
6: #include <X11/Xlib.h>
7: #include <X11/Xutil.h>
8: #include <stdio.h>
9: #include "piflib.h"
10:
11: static unsigned int pscolors[MAX_NUM_OF_COLORS][3];
12:
13: pssetrgb( red, green, blue )
14:     unsigned int *red, *green, *blue;
15: {
16:     float rf, gf, bf;
17:
18:     rf = (float)(*red)/255;
19:     gf = (float)(*green)/255;
20:     bf = (float)(*blue)/255;
21:
22:     fprintf(fp, "stroke\n");
23:     fprintf(fp, "%.2f %.2f %.2f setrgbcolor\n", rf, gf, bf);
24:     fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
25:
26: }
27:
28: /*** pssetrgbnum & pssetclrs added by M.Sasaki (Nov 1996) ***/
29:
30: pssetrgbnum(color_number, red, green, blue )
31:     int *color_number;
32:     unsigned int *red, *green, *blue;
33: {
34:     if( *color_number >= MAX_NUM_OF_COLORS-1 ) {
35:         fprintf(stderr, "*** [Piflib error] too large color number in pssetrgbnum\n",
36:             *color_number ) ;
37:         return 1 ;
38:     }
39:     pscolors[*color_number][0] = *red;
40:     pscolors[*color_number][1] = *green;
41:     pscolors[*color_number][2] = *blue;
42:     return 0;
43: }
44:
45: pssetclrs( color_number )
46:     int *color_number;
47: {
48:
49:     float rf, gf, bf;
50:     rf = (float)(pscolors[*color_number][0])/255;
51:     gf = (float)(pscolors[*color_number][1])/255;
52:     bf = (float)(pscolors[*color_number][2])/255;
53:
54:     fprintf(fp, "stroke\n");
55:     fprintf(fp, "%.2f %.2f %.2f setrgbcolor\n", rf, gf, bf);
56:     /***
57:     fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
58:     *****/
59:
60: }
61:
62: pssetclr( color_number )
63:     int *color_number;
64: {
65:
66:     if(!ps_plot) {
67:         fprintf(fp, "pifdict begin\n");
68:         fprintf(fp, "gsave\n");
69:         fprintf(fp, "newpath\n");
70:         fprintf(fp, "grestore\n");
71:         fprintf(fp, "0 839 translate\n");
72:         fprintf(fp, "-90 rotate\n");
73:         fprintf(fp, ".25 .25 scale\n");
74:         if(!ifplot3) {
75:             fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
76:                 (long)(ps_yorg*y_pix_mm) );
77:         }
78:         fprintf(fp, "LC%d\n", *color_number);
79:         fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
80:         ps_plot = 1;
81:     }
82:
83: pssetltp( kind_line )
84:     int *kind_line;
85: {
86:     if(!ps_plot) {
87:         fprintf(fp, "pifdict begin\n");
88:         fprintf(fp, "gsave\n");
89:         fprintf(fp, "newpath\n");
90:         fprintf(fp, "grestore\n");
91:         fprintf(fp, "0 839 translate\n");
92:         fprintf(fp, "-90 rotate\n");
93:         fprintf(fp, ".25 .25 scale\n");
94:         if(!ifplot3) {
95:             fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
96:                 (long)(ps_yorg*y_pix_mm) );
97:         }
98:     }
99:
100:     if(*kind_line == 0) {
101:         fprintf(fp, "stroke\n");
102:         fprintf(fp, "[ 0 setdash\n");
103:         fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
104:     } else {
105:         fprintf(fp, "LT%d\n", *kind_line);
106:         fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
107:     }
108:     ps_plot = 1;
109: }
110:
111: pssetlwd( wd_line )
112:     int *wd_line;
113: {
114:     if(!ps_plot) {
115:         fprintf(fp, "pifdict begin\n");
116:         fprintf(fp, "gsave\n");
117:         fprintf(fp, "newpath\n");
118:         fprintf(fp, "grestore\n");
119:         fprintf(fp, "0 839 translate\n");
120:         fprintf(fp, "-90 rotate\n");
121:         fprintf(fp, ".25 .25 scale\n");
122:         if(!ifplot3) {
123:             fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
124:                 (long)(ps_yorg*y_pix_mm) );
125:         }
126:     }
127:
128:     fprintf(fp, "stroke\n");
129:     fprintf(fp, "%d setlinewidth\n", *wd_line);

```

src/cgview/pssetattr.c

```
130:     fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
131:     ps_plot = 1;
132: }
133:
134:
135: /** Added capline xontrol Sep 1999 by M.Sasaki
136:  : do nothing currently ***/
137:
138: int pscap_flag = 0 ;
139:
140: pscapline( iflag )
141: int iflag ;
142: {
143:     if ( pscap_flag != iflag ) {
144:         pscap_flag = iflag ;
145:     }
146: }
147:
148: psgetcapline( iflag )
149: int *iflag ;
150: {
151:     *iflag = pscap_flag ;
152: }
153:
154:
155: pscpfont( font_name )
156: char *font_name;
157: {
158:     ;
159: }
160:
161: psfactor( factor_value)
162: float *factor_value;
163: {
164:     f_value = *factor_value;
165: }
166:
167:
168: psnewpen(ipen)
169: int *ipen;
170: {
171:     if(!ps_plot) {
172:         fprintf(fp, "pifdict begin\n");
173:         fprintf(fp, "gsave\n");
174:         fprintf(fp, "newpath\n");
175:         fprintf(fp, "grestore\n");
176:         fprintf(fp, "0 839 translate\n");
177:         fprintf(fp, "-90 rotate\n");
178:         fprintf(fp, ".25 .25 scale\n");
179:         if(!ifplot3) {
180:             fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
181:                 (long)(ps_yorg*y_pix_mm) );
182:         }
183:     }
184:
185:     fprintf(fp, "stroke\n");
186:     fprintf(fp, "%d setlinewidth\n", (*ipen)*2 - 1);
187:     fprintf(fp, "%d %d M\n", (long)ps_store_x, (long)ps_store_y );
188:     ps_plot = 1;
189: }
```

src/cgview/pssymchar.c

```
1: /*****
2:  */
3:  /* pssymchar.c : Symbol Routines for PostScript */
4:  */
5:  *****/
6:  #include <X11/Xlib.h>
7:  #include <X11/Xutil.h>
8:  #include <stdio.h>
9:  #include <math.h>
10: #include "piflib.h"
11:
12: pssymfnt( x_str, y_str, height, string, angle, no_of_letters)
13:     float *x_str, *y_str;
14:     float *height, *angle;
15:     int *no_of_letters;
16:     /** long *no_of_letters; */
17:     char *string;
18: {
19:     float xxx, yyy;
20:
21:     xxx = *x_str;
22:     yyy = *y_str;
23:
24:     xxx *= x_pix_mm;
25:     yyy *= y_pix_mm;
26:
27:     xxx = ps_xorg + xxx;
28:     yyy = ps_yorg - yyy;
29:
30:     /*****/
31:
32:     ps_plot = 1;
33:
34: }
35:
36: pssymbol(x, y, hh, ibcda, angle, nchar )
37:
38:     float *x, *y, *hh, *angle;
39:     int *nchar;
40:     int *ibcda;
41:
42: {
43:     static float oldth = 9999.0;
44:     static float oldfct = 9999.0;
45:     static float rad = 0.017453;
46:     static float x0 = 0.0;
47:     static float y0 = 0.0;
48:
49:     static float DDV = 0.3;
50:
51:     static float eps = 0.01;
52:     static long kp8 = 256;
53:     static long kp16 = 65536;
54:     static int nxy = 0;
55:     static int ichar = 0;
56:
57:     static float xa[9], ya[9];
58:     static float fct, th, th1;
59:     static float sinth, costh;
60:
61:     unsigned long ibit8, ibit16, ibit24;
62:     unsigned int kpos, knum, ktab, kpt;
63:     unsigned int nnt;
64:     unsigned long idata;
65:     unsigned char ibcd[255];
```

```
66:
67:     long iebc;
68:     int ipen, icb, nt, iswv;
69:     int nx, ny;
70:
71:     float xt, yt;
72:     float div, h;
73:     float xx, yy;
74:
75:     int i, m, kk;
76:
77:     static long idxtab[128] = { 0x00080001, 0x030C0002, 0x00060016, 0x02070011
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
0x00070013, 0x02070005, 0x00070007, 0x02080008
0x010B000A, 0x0307000C, 0x010E000E, 0x020D0011
0x02060014, 0x02040011, 0x000C0016, 0x00050079
0x0302004F, 0x030E002E, 0x02030056, 0x0108007A
0x0105006F, 0x0211007C, 0x00080077, 0x03080080
0x0102007A, 0x01030082, 0x000200C1, 0x00080083
0x00060085, 0x03030068, 0x00060088, 0x010D0086
0x02090089, 0x0309008B, 0x0009008E, 0x01070090
0x000C0092, 0x000C0095, 0x020A009B, 0x03070097
0x01090099, 0x0006009E, 0x020E007C, 0x020C009F
0x020900A2, 0x030800A4, 0x00020074, 0x011200BB
0x03050072, 0x030E00A6, 0x01060029, 0x01060067
0x02040056, 0x02040043, 0x02040052, 0x00020063
0x0007002D, 0x02050071, 0x03090044, 0x030D0044
0x01050070, 0x010500AA, 0x0005006E, 0x0005006D
0x00000000, 0x01090032, 0x000C0035, 0x0008004D
0x02070034, 0x00070038, 0x00060038, 0x030C0039
0x01060048, 0x0306003C, 0x020B00AB, 0x00050055
0x0003002A, 0x01040066, 0x0305004F, 0x00020060
0x020A0074, 0x0106005D, 0x0006004B, 0x00030048
0x0105004A, 0x03040049, 0x020A004C, 0x00070060
0x030C004C, 0x00090060, 0x03070040, 0x020B0063
0x030B004F, 0x02040042, 0x020C0053, 0x0303007B
0x02020050, 0x01020062, 0x020D0057, 0x0304005A
0x0007005D, 0x0303005E, 0x0105005F, 0x01050062
0x02050063, 0x0107005B, 0x010E00AE, 0x00060055
```

```

104:                                0x020E0069, 0x030200B1, 0x00030068, 0x010E003E, 164:                                0x27, 0x20, 0x11, 0x31, 0x20, 0x27, 0x16, 0x36,
105:                                0x0309004C, 0x00050019, 0x0109001A, 0x010D0022, 165:                                0x27, 0x03, 0x43, 0x34, 0x32, 0x43, 0x03, 0x14,
106:                                0x0204001C, 0x020A001D, 0x000C001E, 0x00050021, 166:                                0x12, 0x03, 0x01, 0x12, 0x20, 0x27, 0x47, 0x50,
107:                                0x01110022, 0x020B0026, 0x020B0053, 0x010B00B2, 167:                                0x00, 0x24, 0x07, 0x57, 0x27, 0x25, 0x40, 0x15,
108:                                0x001000B5, 0x0004002C, 0x0305002A, 0x000900B9, 168:                                0x16, 0x27, 0x36, 0x03, 0x02, 0x11, 0x21, 0x42,
109:};                               169:                                0x04, 0x44, 0x70, 0x41, 0x01, 0x70, 0x10, 0x35,
110:static int itz[768] = {         170:                                0x13, 0x23, 0x24, 0x14, 0x13, 0x00, 0x40, 0x70,
111:                                0x24, 0x14, 0x03, 0x01, 0x10, 0x30, 0x41, 0x43, 171:                                0x46, 0x06, 0x70, 0x03, 0x43, 0x42, 0x52, 0x42,
112:                                0x34, 0x24, 0x22, 0x24, 0x02, 0x20, 0x42, 0x24, 172:                                0x33, 0x22, 0x12, 0x03, 0x04, 0x15, 0x25, 0x34,
113:                                0x22, 0x02, 0x24, 0x20, 0x24, 0x42, 0x22, 0x04, 173:                                0x33, 0x34, 0x45, 0x55, 0x64, 0x63, 0x52, 0x01,
114:                                0x44, 0x22, 0x00, 0x22, 0x40, 0x22, 0x12, 0x32, 174:                                0x41, 0x70, 0x22, 0x26, 0x24, 0x44, 0x04, 0x05,
115:                                0x22, 0x44, 0x04, 0x44, 0x00, 0x40, 0x00, 0x22, 175:                                0x01, 0x10, 0x20, 0x31, 0x36, 0x47, 0x57, 0x66,
116:                                0x04, 0x22, 0x44, 0x22, 0x20, 0x22, 0x44, 0x43, 176:                                0x02, 0x32, 0x43, 0x44, 0x35, 0x05, 0x16, 0x26,
117:                                0x13, 0x04, 0x13, 0x11, 0x00, 0x11, 0x31, 0x40, 177:                                0x44, 0x54, 0x65, 0x70, 0x63, 0x52, 0x42, 0x24,
118:                                0x31, 0x33, 0x22, 0x24, 0x20, 0x22, 0x02, 0x42, 178:                                0x14, 0x03, 0x00, 0x10, 0x21, 0x22, 0x33, 0x24,
119:                                0x22, 0x04, 0x40, 0x22, 0x00, 0x44, 0x22, 0x04, 179:                                0x25, 0x16, 0x06, 0x46, 0x36, 0x25, 0x24, 0x13,
120:                                0x44, 0x00, 0x40, 0x22, 0x22, 0x24, 0x01, 0x41, 180:                                0x22, 0x21, 0x30, 0x40, 0x00, 0x15, 0x12, 0x21,
121:                                0x24, 0x22, 0x23, 0x03, 0x20, 0x43, 0x23, 0x22, 181:                                0x31, 0x42, 0x45, 0x42, 0x51, 0x11, 0x15, 0x05,
122:                                0x10, 0x30, 0x20, 0x27, 0x16, 0x05, 0x16, 0x17, 182:                                0x45, 0x35, 0x31, 0x42, 0x21, 0x11, 0x02, 0x04,
123:                                0x37, 0x46, 0x43, 0x01, 0x00, 0x40, 0x30, 0x37, 183:                                0x15, 0x35, 0x44, 0x42, 0x31, 0x21, 0x20, 0x26,
124:                                0x02, 0x42, 0x47, 0x07, 0x04, 0x34, 0x43, 0x41, 184:                                0x01, 0x05, 0x16, 0x26, 0x35, 0x33, 0x03, 0x33,
125:                                0x30, 0x10, 0x01, 0x02, 0x06, 0x17, 0x37, 0x46, 185:                                0x31, 0x20, 0x10, 0x01, 0x45, 0x70, 0x41, 0x31,
126:                                0x03, 0x07, 0x47, 0x46, 0x20, 0x06, 0x17, 0x37, 186:                                0x15, 0x05, 0x02, 0x11, 0x22, 0x25, 0x22, 0x31,
127:                                0x46, 0x45, 0x34, 0x14, 0x34, 0x43, 0x41, 0x30, 187:                                0x42, 0x45, 0x04, 0x14, 0x12, 0x21, 0x32, 0x34,
128:                                0x10, 0x01, 0x03, 0x14, 0x05, 0x06, 0x01, 0x10, 188:                                0x44, 0x70, 0x26, 0x20, 0x00, 0x23, 0x41, 0x23,
129:                                0x30, 0x41, 0x46, 0x37, 0x17, 0x06, 0x04, 0x13, 189:                                0x15, 0x06, 0x24, 0x13, 0x12, 0x21, 0x31, 0x42,
130:                                0x43, 0x01, 0x41, 0x70, 0x42, 0x04, 0x46, 0x04, 190:                                0x43, 0x34, 0x24, 0x15, 0x26, 0x36, 0x41, 0x21,
131:                                0x44, 0x70, 0x41, 0x01, 0x37, 0x35, 0x27, 0x37, 191:                                0x12, 0x14, 0x25, 0x45, 0x70, 0x43, 0x03, 0x04,
132:                                0x06, 0x17, 0x26, 0x20, 0x26, 0x37, 0x46, 0x03, 192:                                0x15, 0x24, 0x12, 0x24, 0x35, 0x44, 0x30, 0x21,
133:                                0x13, 0x12, 0x14, 0x13, 0x23, 0x22, 0x24, 0x23, 193:                                0x31, 0x32, 0x22, 0x21, 0x70, 0x03, 0x53, 0x70,
134:                                0x33, 0x32, 0x34, 0x33, 0x44, 0x00, 0x03, 0x43, 194:                                0x24, 0x34, 0x35, 0x25, 0x24, 0x01, 0x45, 0x70,
135:                                0x03, 0x06, 0x17, 0x37, 0x46, 0x40, 0x37, 0x46, 195:                                0x05, 0x41, 0x42, 0x31, 0x11, 0x02, 0x04, 0x15,
136:                                0x41, 0x30, 0x00, 0x07, 0x37, 0x46, 0x45, 0x34, 196:                                0x35, 0x44, 0x70, 0x26, 0x20, 0x02, 0x11, 0x31,
137:                                0x04, 0x34, 0x43, 0x41, 0x47, 0x07, 0x04, 0x34, 197:                                0x42, 0x43, 0x34, 0x14, 0x05, 0x16, 0x36, 0x45,
138:                                0x04, 0x00, 0x40, 0x53, 0x33, 0x43, 0x41, 0x30, 198:                                0x70, 0x27, 0x20, 0x03, 0x63, 0x11, 0x15, 0x14,
139:                                0x10, 0x01, 0x06, 0x17, 0x37, 0x46, 0x45, 0x10, 199:                                0x04, 0x44, 0x34, 0x35, 0x31, 0x32, 0x42, 0x02,
140:                                0x30, 0x20, 0x27, 0x17, 0x37, 0x05, 0x06, 0x17, 200:                                0x42, 0x31, 0x11, 0x02, 0x05, 0x16, 0x36, 0x45,
141:                                0x37, 0x46, 0x45, 0x34, 0x24, 0x22, 0x70, 0x10, 201:                                0x43, 0x32, 0x22, 0x13, 0x14, 0x25, 0x35, 0x44,
142:                                0x30, 0x21, 0x10, 0x70, 0x22, 0x27, 0x20, 0x31, 202:                                0x37, 0x35, 0
```

src/cgview/pssymchar.c

```

228:         ipen = 2;
229:     }
230:     icb = 3;
231:     ichar = *ibcda + 1;
232:
233:     if( ichar <= 15 ) {
234:         div = 4.0;
235:     }
236: }
237:
238: fct = h / div;
239: th = *angle;
240:
241: if( th != oldth ) {
242:     oldth = th;
243:     th1 = th*rad;
244:     sinth = sin( th1 );
245:     costh = cos( th1 );
246:     oldfct = fct;
247:     xa[0] = 0.0;
248:     ya[0] = 0.0;
249:     xa[1] = fct*costh;
250:     ya[1] = fct*sinth;
251:     for ( i = 2 ; i < 8 ; i++ ) {
252:         xa[i] = xa[i-1] + xa[1];
253:         ya[i] = ya[i-1] + ya[1];
254:     }
255: }
256: if( fct != oldfct ) {
257:     oldfct = fct;
258:     xa[0] = 0.0;
259:     ya[0] = 0.0;
260:     xa[1] = fct*costh;
261:     ya[1] = fct*sinth;
262:     for ( i = 2 ; i < 8 ; i++ ) {
263:         xa[i] = xa[i-1] + xa[1];
264:         ya[i] = ya[i-1] + ya[1];
265:     }
266: }
267:
268: xx = *x;
269: yy = *y;
270: if( (float)sqrt( pow( (double)( xx - 999.0 ), 2.0 ) ) > eps ) x0 = xx;
271: if( (float)sqrt( pow( (double)( yy - 999.0 ), 2.0 ) ) > eps ) y0 = yy;
272: xx = x0;
273: yy = y0;
274:
275: /* VERTICAL DIRECTION CHECK. */
276: if( nt >= 0 ) {
277:     if( *hh < 0 ) {
278:         x0 = x0 + h*sinth;
279:         y0 = y0 - h*costh;
280:         xa[8] = ( 1.0 + DDV ) * h*sinth;
281:         ya[8] = -( 1.0 + DDV ) * h*costh;
282:         iswv = 1;
283:     }
284: }
285:
286: if( ( nt < 0 ) && ( ichar <= 15 ) ) {
287:     x0 = x0 - ( xa[2] - ya[2] );
288:     y0 = y0 - ( xa[2] + ya[2] );
289: }
290:
291: nnt = abs( nt );
292:

```

```

293: if ( icb != 3 ) {
294:     strcpy(ibcd, (char *)ibcda);
295:     ibcd[*nchar] = '\0' ;
296: }
297:
298: for ( kk = 0 ; kk < nnt ; kk++ ) {
299:
300:     if ( icb == 3 ) {
301:         ichar = *ibcda;
302:     }
303:     else {
304:         astoeb( &ibcd[kk], &iebc );
305:         ichar = iebc%128;
306:     }
307:     idata = idxtab[ichar];
308:
309:     if( idata != 0 ) {
310:         kpos = idata / ibit24;
311:         knum = ( idata / ibit16 ) % ibit8;
312:         ktab = idata % ibit16;
313:         kpt = ( ktab - 1 ) * 4 + kpos;
314:         for ( m = kpt ; m < kpt+knun ; m++ ) {
315:             nx = itz[m] / 16;
316:             ny = itz[m] % 16;
317:
318:             if( nx == 7 ) {
319:                 ipen = 3;
320:             }
321:             if( nx != 7 ) {
322:                 xt = x0 + xa[nx] - ya[ny];
323:                 yt = y0 + ya[nx] + xa[ny];
324:                 psplot( &xt, &yt, &ipen );
325:                 ipen = 2;
326:             }
327:         }
328:     }
329:
330:     x0 = x0 + xa[7];
331:     y0 = y0 + ya[7];
332:     ipen = 3 ;
333: }
334: /*
335: *
336: * END OF SYMB4
337: *
338: */
339: if( *hh < 0.0 ) {
340:     x0 = x0 - h*sinth;
341:     y0 = y0 + h*costh;
342: }
343:
344: if( *nchar < 0 ) {
345:     x0 = xx;
346:     y0 = yy;
347: }
348:
349: ipen = 3;
350: psplot(&x0, &y0, &ipen);
351: }

```

src/cgview/seaone.f

```

1:      subroutine SEAONE(IOW, JREVR,
2:      N      JGMCK, IGMCK, IPGMCK, XGMCK, NIPGMCK, MIPGMCK,
3:      N      DOVERLAP, MPLIMIT,
4:      N      NBANK, NINPZ, NZONE, NSDA, NZDA, IDEFER, NDEAD,
5:      N      NREFS, NREG, NLBZ, NEST, NSUZON, NSPACE, NSURF, DEPS,
6:      N      MSGST, NDEFSRC,
7:      B      XXX, YYY, ZZZ, AAA, BBB, CCC,
8:      B      IZZ, LEVL, LZZ,
9:      B      LPOS, LCRS, KLSF, IBREG, IBSPC,
10:     B      IZSS, IBNK, KIBNK, MIBNK,
11:     S      LSFFL, NFFL, IZFFL, LSSRC, NNXT, IZNXT,
12:     S      LSDED, LSREF, ISREF, IZREF, NBREF, KSREF,
13:     *      LSLAT, IZLAT, NALT,
14:     G      SDA, IPSDA, IBSDA, KREGN,
15:     G      KINPZ, IZNAM, KZMAT, KZREG, KZDA, KZAA,
16:     *      KCELL, IPCEL, MLBZZ, ISUSP, LPOS0,
17:     W      X, Y, Z, W, IBP, IFI,
18:     W      IZI, IWK, IFL, R, ISRF, FXYZ,
19:     P      LEVEL, MCELL, IMCELL,
20:     P      MZONE, NMEMO, KMEMO2, MEMC2, MEMZ2)
21: C
22: C=====
23: C PURPOSE:  FIND ZONES TO ENTER.  (ONE_ZONE LOGIC)
24: C CALLED IN: ACTION
25: C CALLS: JUDGE
26: C=====
27: C
28: C === INTER-STACK DATA FLOW ===
29: C
30: C   NEXT-ZONE SEARCH STACK---+---->  FREE-FLIGHT STACK
31: C   (LSSRC, IZNXT, NNXT)      !      !      (LSFFL, IZFFL, NFFL)
32: C   DEAD PARTICLE STACK  ---+---->  LATTICE-SEARCH STACK
33: C   (LSDED, NDEAD)          !      !      (LSLAT, IZLAT, NXLT)
34: C                           +---->  REFLECTION STACK
35: C                           !      !      (LSREF, ISREF, IZREF, NBREF)
36: C   DEAD-PARTICLE STACK  +---->  (LSDED, NDEAD)
37: C   (LSDED, NDEAD)
38: C === BANK DATA TO BE UPDATED ===
39: C
40: C   IZZ      : ZONE #
41: C   CCCC WWW  : WEIGHT (SPLITTING OR RUSSIAN ROULETTE)
42: C
43: C === BANK DATA ADDED NEWLY (SPLITTING) ===
44: C
45: C   (XXX, YYY, ZZZ), (AAA, BBB, CCC), IZZ : POSITION, DIRECTION & ZONE #
46: C   CCCC IGG, WWW      : ENERGY GROUP & WEIGHT
47: C   LEVL, LZZ, LPOS, LCRS      : LATTICE-PARAMETER
48: C
49: C=====
50:     implicit real*8(D)
51:     INCLUDE 'INC/_FLAGS'
52: C ..... BANK .....
53:     real*8 XXX(NBANK), YYY(NBANK), ZZZ(NBANK), AAA(NBANK), BBB(NBANK),
54:     & CCC(NBANK)
55:     integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK, NEST),
56:     & LPOS(NBANK, NEST), LCRS(NBANK, NEST), KLSF(NBANK)
57:     integer IBREG(NBANK), IBSPC(NBANK, 0:NST)
58:     integer LPOS0(NBANK)
59: C
60:     integer IZSS(NBANK)
61:     integer IBNK(NBANK, *), KIBNK(0:MIBNK)
62: C
63: C ... counter of error in geometry check
64:     integer IGMCK(NZONE)
65:     integer IPGMCK(3, MIPGMCK)

```

```

66:     real*8 XGMCK(4, MIPGMCK)
67:     real*8 DOVERLAP
68: C
69:     integer NDEFSRC(NINPZ)
70: C
71: C ..... STACK .....
72: C
73:     integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK), LSSRC(NBANK),
74:     & NNXT(NZONE+1), IZNXT(NBANK), LSREF(NBANK), ISREF(NBANK),
75:     & IZREF(NBANK), NBREF(1), LSLAT(NBANK), IZLAT(NBANK),
76:     & NXLT(NLBZ+1), LSDED(NBANK)
77: C
78: C ..... GEOMETRY .....
79: C
80:     real*8 SDA(NSDA)
81:     integer IPSDA(3, NSURF+1)
82:     integer IBSDA(3, *)
83: C
84:     integer KZMAT(NZONE), KZREG(NZONE), KSREF(NZDA), KZDA(2, NZDA),
85:     & KZAA(NZONE+1), KCELL(NZONE), IPCEL(1), MLBZZ(NZONE)
86: C
87:     integer KINPZ(NZONE)
88:     character*12 IZNAM(NINPZ)
89:     character*12 KREGN(NINPZ)
90: C
91:     integer KMEMO2(NZONE, NMEMO, 0:1), MEMC2(NZONE, NMEMO, 0:1),
92:     & MEMZ2(NZONE, 0:1)
93: C
94:     integer ISUSP(NSUZON, NSPACE)
95: C ..... VARIANCE REDUCTION .....
96: C
97: C ..... WORK AREA .....
98: C
99:     real*8 X(NBANK), Y(NBANK), Z(NBANK), W(NBANK)
100:    real R(NBANK), FXYZ(NBANK)
101:    integer IBP(NBANK), IZI(NBANK), IWK(NBANK), ISRF(NBANK)
102:    logical IFI(NBANK), IFL(NBANK), JRR
103:    CCC data JRR /.false./
104: C
105:    include '../shared/INC/_PMLATT'
106:    include '../shared/INC/_SFLATT'
107: C
108: C -----
109: C
110:     NZ1 = NZONE + 1
111:     NR1 = NREFS + 1
112: C
113:     LEVELA = ABS(LEVEL)
114: C
115:     DDD2 = DEPS*2.0D0
116:     if ( JGMCK.ne.0 ) then
117:       if ( DOVERLAP.gt.0 ) then
118:         DDD2 = DEPS*DOVERLAP
119:       else if ( DOVERLAP.lt.0 ) then
120:         DDD2 = -DOVERLAP
121:       end if
122:     end if
123: C
124: C .... MZONE:  > 0 : SEARCH FOR ZONE MZONE,  < 0 : DEFERRED PARTICLES
125:     if ( MZONE.lt.0 ) go to 380
126: C
127: C .... GATHER VECTORS .....
128: C
129:     IRCF = 0
130: C

```

src/cgview/seaone.f

```

131:      do 360 K SZ = 0, 1
132: C
133:      NIBP      = 0
134:      do 100 I = 1, NNXT(NZONE+1)
135:      if ( IZNXT(I).eq.MZONE.and.IZSS(LSSRC(I)).eq.KSZ ) then
136:      NIBP      = NIBP + 1
137:      IBP(NIBP) = LSSRC(I)
138:      end if
139: 100 continue
140: Check %%%%%%%%%%
141: C      write(*,*) '%%K SZ ',KSZ,
142: C      & ' MZONE ',MZONE,' NNXT(NZONE+1) ',NNXT(NZONE+1),
143: C      & ' NNXT(MZONE) ',NNXT(MZONE),' NIBP ',NIBP
144: C %%%%%%%%%%
145: C
146:      if ( NIBP.eq.0 ) go to 360
147: C
148:      do 110 I = 1, NIBP
149:      Y(I)      = XXX(IBP(I)) + DEPS*AAA(IBP(I))
150:      Y(I)      = YY(IBP(I)) + DEPS*BBB(IBP(I))
151:      Z(I)      = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
152: 110 continue
153: C
154: C .... SEARCH ZONE # KKZ+1 TO MMZ.
155: C
156:      if ( JLATT.ne.0 ) then
157:      ICEL      = KCELL(MZONE)
158:      MMZ      = IPCEL(ICEL+1) - 1
159:      if ( ICEL.gt.0 ) then
160:      KKZ      = IPCEL(ICEL) - 1
161:      else
162:      KKZ      = 0
163:      end if
164:      else
165:      ICEL      = 0
166:      KKZ      = 0
167:      MMZ      = NZONE
168:      end if
169: C
170:      MMX      = MIN(MMZ-KKZ,NMEMO)
171: C
172: C      if ( JREVRs.ne.0 ) then
173: C      if ( KSZ.eq.1 ) then
174: C      IK      = KKZ
175: C      KKZ      = MMZ + 1
176: C      MMZ      = IK + 1
177: C      end if
178: C
179: C ..... IMEMO = 1 MEANS THAT ALL ZONES IN KMEMO ARRAY HAS NOT BEEN
180: C SEARCHED.
181: C      IMEMO      = 1
182: C
183: C ..... BEGINS NEXT-ZONE-SEARCH .....
184: C      INX      = NNXT(MZONE)
185: C
186:      INX      = NIBP
187:      INFL      = NFFL(NZONE+1)
188: C
189: C      IF(JREFL.NE.0) IRCF = NBREF(NR1)
190: C      IRCF      = 0
191: C
192:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
193: C
194:      do 280 M = 1, MMX
195: C

```

```

196: C .... RETURN HERE IF KMEMO(MZONE,M)=0 AND THERE ARE NO PARTICLES WHICH
197: C BELONGS TO ZONE 'KZ'. DETERMINE NEXT 'KZ' WITHOUT INCREMENTING
198: C LOOP COUNTER 'M'.
199: C
200: 120      if ( INX.eq.0 ) go to 290
201: C
202: C .... DETERMINE ZONE TO BE CHECKED (KZ) .....
203: C      if ( KMEMO2(MZONE,M,KSZ).ne.0 ) then
204: C      KZ      = KMEMO2(MZONE,M,KSZ)
205: C      else
206: C      IMEMO      = 0
207: C      if ( JREVRs.eq.0 ) then
208: C      if ( KSZ.eq.0 ) then
209: C      KKZ      = KKZ + 1
210: C      if ( KKZ.eq.MZONE ) go to 130
211: C      do 140 MM = 1, MMX
212: C      if ( KKZ.eq.KMEMO2(MZONE,MM,KSZ) ) go to 130
213: C      continue
214: C      KZ      = KKZ
215: C      if ( KZ.gt.MMZ ) go to 290
216: C      else
217: C      KKZ      = KKZ - 1
218: C      if ( KKZ.eq.MZONE ) go to 150
219: C      do 160 MM = 1, MMX
220: C      if ( KKZ.eq.KMEMO2(MZONE,MM,KSZ) ) go to 150
221: C      continue
222: C      KZ      = KKZ
223: C      if ( KZ.lt.MMZ ) go to 290
224: C      end if
225: C      end if
226: C      MAT      = KZMAT(KZ)
227: C      IF(JREFL.NE.0) JRR = MAT.LE.-1001
228: C
229: C      JRR      = .false.
230: C
231: C      ... get distance from
232: C
233: C      if ( JGMCK.ne.0 ) then
234: C      JRR      = .true.
235: C      end if
236: C
237: C .... CHECK FOR EACH PARTICLE. IFI(I) = .FALSE./.TRUE. = OUT/IN
238: C
239: C      call JUDGE( 'SEAONE', KZ, INX, X, Y, Z, IFI, SDA, KZDA,
240: C      & KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, JRR,
241: C      & KSREF, ISRF, FXYZ )
242: C
243: C .... COMPRESS PARTICLE DATA & UPDATE FLIGHT STACK .....
244: C
245: C      KK      = 0
246: C      do 170 I = 1, INX
247: C      if ( IFI(I) ) KK      = KK + 1
248: C      170 continue
249: C
250: C      if ( KK.ne.0 ) then
251: C
252: C      ... check material/region overlap for input-zone
253: C
254: C      if ( JGMCK.ne.0.and.KZMAT(MZONE).ne.-999
255: C      & .and.KINPZ(KZ).ne.KINPZ(MZONE) ) then
256: C      check %%%%%%%%%%
257: C      write(*,*) '%%seaone ',KINPZ(KZ),KINPZ(MZONE)
258: C %%%%%%%%%%
259: C      CCCCCC      MPLIMIT = 10
260: C      call OVLPMMSG( IOW, JTLLT, IMCELL, MZONE, KZ,

```

src/cgview/seaone.f

```

261:      &          DDD2, MPLIMIT,
262:      &          NINPZ, NZONE, NSURF, NSUZON, NEST, NSPACE,
263:      &          IPSDA, IBSDA, IZNAM, KREGN, KZMAT,
264:      &          KZREG, KINPZ, ISUSP,
265:      &          INX, IBP, IFI, FXYZ, IGMCK,
266:      &          IPGMCK, XGMCK, NIPGMCK, MIPGMCK, XXX, YYY,
267:      &          ZZZ, AAA, BBB, CCC, KLSF, LEVL, IBSPC,
268:      &          IBNK, KIBNK, MIBNK,
269:      &          NBANK )
270:      end if
271: C
272:      II      = 0
273:      IFFL    = 0
274:      IDEAD   = NDEAD
275: C
276: C ..... SEND PARTICLES TO FLIGHT STACK .....
277: C
278:      if ( MAT.ge.0 ) then
279: *VOCL LOOP,NOVREC
280:      do 180 I = 1, INX
281:      if ( IFI(I) ) then
282:      IFFL = IFFL + 1
283:      LSFFL(INFL+IFFL) = IBP(I)
284:      end if
285: 180      continue
286: C
287: C ..... UPDATE REGION NUMBER FOR UNIVERSE DEPENDENT TALLY .....
288: C
289:      if ( JTLLT.ne.0 ) then
290: *VOCL LOOP,NOVREC
291:      do 190 I = 1, IFFL
292:      IP      = LSFFL(INFL+I)
293:      ILV     = LEVL(IP)
294:      if ( ILV.gt.0 ) then
295:      IBREG(IP) = ISUSP(KZREG(KZ),IBSPC(IP,ILV))
296:      else
297:      IBREG(IP) = KZREG(KZ)
298:      end if
299: 190      continue
300:      else
301: C
302: C ... always set IBREG
303: *VOCL LOOP,NOVREC
304:      do 200 I = 1, IFFL
305:      IP      = LSFFL(INFL+I)
306:      IBREG(IP) = KZREG(KZ)
307: 200      continue
308:      end if
309: C
310:      do 210 I = 1, IFFL
311:      IZFFL(INFL+I) = KZ
312: 210      continue
313:      NFFL(KZ) = NFFL(KZ) + IFFL
314:      INFL = INFL + IFFL
315: C
316: C ..... ENTERING LATTICE .....
317: C
318: CCCCC      else if ( MAT.le.-1.and.MAT.ge.-998 ) then
319:      else if ( ISLATT(MAT) ) then
320: *VOCL LOOP,NOVREC
321:      do 220 I = 1, INX
322:      if ( IFI(I).and.LEVL(IBP(I)).lt.LEVELA ) then
323:      ILAT = ILAT + 1
324:      LSLAT(ILAT) = IBP(I)
325:      IZLAT(ILAT) = MLBZZ(KZ)

```

```

326:      IZZ(IBP(I)) = KZ
327:      LPOS0(IBP(I)) = -1
328: C
329: C ... when LEVEL < NEST, some particles are sent to
330: C flight stack
331: C
332:      else if ( IFI(I).and.LEVL(IBP(I)).ge.LEVELA ) then
333:      IFFL = IFFL + 1
334:      LSFFL(INFL+IFFL) = IBP(I)
335: CCCC      if ( JTLLT.ne.0 ) IBREG(IBP(I)) = 0
336:      IBREG(IBP(I)) = 0
337:      end if
338: 220      continue
339: C
340:      if ( IFFL.gt.0 ) then
341:      do 230 I = 1, IFFL
342:      IZFFL(INFL+I) = KZ
343: 230      continue
344:      NFFL(KZ) = NFFL(KZ) + IFFL
345:      INFL = INFL + IFFL
346:      end if
347:      NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + KK - IFFL
348: C
349: C ..... ESCAPING CELL OR LATTICE .....
350: C
351:      else if ( MAT.eq.-999
352:      .and.
353:      (IMCELL.eq.0.or.(IMCELL.gt.0.and.KZ.ne.IMCELL)) )
354:      then
355: *VOCL LOOP,NOVREC
356:      do 240 I = 1, INX
357:      if ( IFI(I) ) then
358:      ILAT = ILAT + 1
359:      LSLAT(ILAT) = IBP(I)
360:      IZLAT(ILAT) = MLBZZ(KZ)
361:      IZZ(IBP(I)) = KZ
362:      end if
363: 240      continue
364:      NXLT(MLBZZ(KZ)) = NXLT(MLBZZ(KZ)) + KK
365: C
366: C ..... LEAKAGE & REFLECTION .....
367: C
368: C ( CGVIEW: Particles are sent to flight stack )
369: C
370: CCCC      else if ( (IMCELL.eq.KZ) .or. MAT.le.-1000 ) then
371:      else if ( (IMCELL.eq.KZ) .or. MAT.le.-1000
372:      .and..not.ISLATT(MAT) ) then
373: Cccc      if ( KZMAT(MZONE).le.-999 ) then
374: *VOCL LOOP,NOVREC
375:      do 250 I = 1, INX
376:      if ( IFI(I) ) then
377:      IFFL = IFFL + 1
378:      LSFFL(INFL+IFFL) = IBP(I)
379: CCCCCC      if ( JTLLT.ne.0 ) IBREG(IBP(I)) = 0
380:      IBREG(IBP(I)) = 0
381:      end if
382: 250      continue
383:      do 260 I = 1, IFFL
384:      IZFFL(INFL+I) = KZ
385: 260      continue
386:      NFFL(KZ) = NFFL(KZ) + IFFL
387:      INFL = INFL + IFFL
388: Cccc      else
389: *VOCL LOOP,NOVREC
390: Cccc      do 240 I = 1, INX

```


src/cgview/seaone.f

```

391: Cccc          if ( IFI(I) ) then
392: Cccc          IRCF = IRCF + 1
393: Cc            LSREF(IRCF) = IBP(I)
394: Cc            IZZ(IBP(I)) = KZ
395: Cc            NDEAD = NDEAD + 1
396: Cc            if ( JTLT.ne.0 ) IBREG(IBP(I)) = 0
397: Cc            end if
398: Cc240          continue
399: Cc            end if
400: Cc            end if
401: C
402: C ..... COMPRESS POINTERS & TEMPORALY ARRAYS .....
403: C
404: *VOCL LOOP,NOVREC
405:          do 270 I = 1, INX
406:            if ( .not.IFI(I) ) then
407:              II = II + 1
408:              IBP(II) = IBP(I)
409:              X(II) = X(I)
410:              Y(II) = Y(I)
411:              Z(II) = Z(I)
412:            end if
413: 270          continue
414:          INX = II
415: C
416: C ..... SETTING OF KMEMO ARRAY .....
417: C
418:          if ( IMEMO.eq.0 ) then
419:            KMEMO2(MZONE,M,KSZ) = KZ
420:            MEMC2(MZONE,M,KSZ) = MEMC2(MZONE,M,KSZ) + KK
421:            MEMZ2(MZONE,KSZ) = M
422:          else if ( M.le.NMEMO ) then
423:            MEMC2(MZONE,M,KSZ) = MEMC2(MZONE,M,KSZ) + KK
424:          end if
425: C
426:          else if ( IMEMO.eq.0 ) then
427:            go to 120
428:          end if
429: 280          continue
430: C
431: C .... UPDATE ZONE # IN BANK
432: C
433: 290          continue
434:          do 300 I = NFFL(NZONE+1) + 1, INFL
435:            IZZ(LSFFL(I)) = IZFFL(I)
436: 300          continue
437: C
438: C
439: C .... COMPRESS SEARCH STACK .....
440:          II = 0
441: *VOCL LOOP,NOVREC
442:          do 310 I = 1, NNXT(NZONE+1)
443:            if ( IZNXT(I).ne.MZONE .or. IZSS(LSSRC(I)).ne.KSZ ) then
444:              II = II + 1
445:              LSSRC(II) = LSSRC(I)
446:              IZNXT(II) = IZNXT(I)
447:            end if
448: 310          continue
449: C
450: C IF THERE ARE ANY UNFINISHED PARTICLES IN SEARCH STACK, I TREAT THEM
451: C AS DEFERRED PARTICLES OR LOST PARTICLES.
452: C
453: Check %%%%%%%%%%
454: C      write(*,*) '%%% INX ',INX
455: C %%%%%%%%%%%%%%%

```

```

456: C ..... DEFERRED PARTICLES .....
457: C          if ( IMEMO.eq.1 .or. IMEMO.eq.0.and.
458: C            & ((JREVRs.eq.0.and.KKZ.le.MMZ)
459: C            & .or.(JREVRs.ne.0.and.KKZ.ge.MMZ)) ) then
460: C
461:          if ( INX.ne.0 ) then
462:            if ( IMEMO.eq.1 .or. (IMEMO.eq.0
463: C            & .and.
464: C            & ((KSZ.eq.0.and.KKZ.le.MMZ).or.(KSZ.eq.1.and.KKZ.ge.MMZ)))
465: C            & ) then
466: C
467:            do 320 I = 1, INX
468:              LSSRC(II+I) = IBP(I)
469:              IZNXT(II+I) = -MZONE
470:            continue
471: 320          IDEFER = IDEFER + INX
472:              NNXT(NZONE+1) = NNXT(NZONE+1) + INX
473: C
474:              NDEFSRC(KINPZ(MZONE)) = NDEFSRC(KINPZ(MZONE)) + 1
475: C
476: C ..... LOST PARTICLES .....
477: C
478:          else
479: C
480:            if ( IMCELL.eq.MZONE .or. KZMAT(MZONE).ne.-999 ) then
481: C
482: C .... CHECK FOR PREVIOUS ZONE. IFI(I) = .FALSE./.TRUE. = OUT/IN
483: C
484: C << CGVIEW >>
485: C   paths in reflector material zone or "-999" material zone
486: C   will be rescued here,
487: C   because they may be given infinite flight distance in FLIONE
488: C   routine but not treated as "lost" so their position is
489: C   actually somewhere in zone MZONE.
490: C
491:          JRR = .false.
492:          call JUDGE( 'SEAONE', MZONE, INX, X, Y, Z, IFI, SDA,
493: C          & KZDA, KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL,
494: C          & JSIMP, JRR, KSREF, ISRF, FXYZ )
495: C
496:          IDEAD = 0
497:          IFFL = 0
498: C
499:          do 330 I = 1, INX
500: C
501: C ..... TANGENTIAL or from reflective material
502: C ==> sent (or re-sent) to flight stack
503: C
504:          if ( IFI(I).and.(.not.ISLATT(KZMAT(MZONE))) ) then
505: C          & .or.ISLATT(KZMAT(MZONE)).and.LEVL(IBP(I)).lt.LEVELA
506: C
507:            IFFL = IFFL + 1
508:            LSFFL(INFL+IFFL) = IBP(I)
509:            IZFFL(INFL+IFFL) = MZONE
510:            XXX(IBP(I)) = X(I)
511:            YYY(IBP(I)) = Y(I)
512:            ZZZ(IBP(I)) = Z(I)
513:            KLSF(IBP(I)) = 0
514: C ..... LOST
515: C
516:          else
517:            IFI(I) = .false.
518:            IDEAD = IDEAD + 1
519:            LSDED(IBP(I)) = 3
520: C
521:          end if
522: 330          continue
523: C

```

src/cgview/seaone.f

```

521:      NFFL(MZONE) = NFFL(MZONE) + IFFL
522:      INFL      = INFL + IFFL
523: C
524:      else
525: C
526:      do 340 I = 1, INX
527:      IFI(I) = .false.
528:      LSDDED(IBP(I)) = 3
529: 340      continue
530:      IDEAD = INX
531: C
532:      end if
533: C
534:      NDEAD = NDEAD + IDEAD
535:      if ( IDEAD.gt.0 ) then
536:      write(IOW,*) ' !! (SEAONE) ', IDEAD,
537:      & ' particles are LOST !! ZONE # = ', MZONE
538: C
539:      do 350 I = 1, MIN(INX,MSGLST)
540:      if (.not.IFI(I) ) then
541:      write(IOW,7000) IBP(I), MZONE, X(I), Y(I), Z(I),
542:      & AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I))
543:      &
544:      if ( KLSF(IBP(I)).ne.0 ) then
545:      call SDA2BD( KLSF(IBP(I)), ISRF, IBD, IPDA,
546:      & NSURE )
547:      if ( IBD.ne.0 ) then
548:      write(IOW,*) ' On body ', IBD
549:      end if
550:      end if
551:
552:      if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 ) then
553:      write(IOW,7020)
554:      & (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,
555:      & LEVL(IBP(I)))
556:      end if
557:      end if
558: 350      continue
559:      if ( INX.gt.MSGLST ) then
560:      write(IOW,*) ' >>> more ', INX - MSGLST,
561:      & ' particles are lost but message suppressed.'
562:      end if
563:      end if
564: 7000      format(1X,I5,' ZONE',I5,' XYZ=',1P,E12.5,E12.5,E12.5,
565:      & ' DIR=',E11.4,E11.4,E11.4)
566: 7020      format(8X,'LEVL = ',I3,' LZZ = ',I5,' LPOS = ',I5)
567:      end if
568:      end if
569: C
570: C .... Adjustment of number of particle in stack .....
571: C
572: C      NNXT(NZONE+1) = NNXT(NZONE+1) - NNXT(MZONE)
573: C      NNXT(MZONE) = 0
574: C
575: C      NNXT(NZONE+1) = NNXT(NZONE+1) - NIBP
576: C      NNXT(MZONE) = NNXT(MZONE) - NIBP
577: C
578: C      NFFL(NZONE+1) = INFL
579: C      if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
580: C
581: 360 continue
582: C
583:      if ( IRCF.gt.0 ) then
584:      INFL = NFFL(NZONE+1)
585:      do 370 I = 1, IRCF

```

```

586:      LSFFL(INFL+I) = LSREF(I)
587: 370      continue
588:      end if
589:      NBREF(1) = IRCF
590:      return
591: C
592: C =====
593: C .... TREATMENT OF DEFERRED PARTICLE .....
594: C =====
595: C
596: 380 continue
597: C
598:      IRCF = 0
599:      do 500 KSZ = 0, 1
600: C
601:      NIBP = 0
602:      do 390 I = 1, NNXT(NZONE+1)
603:      if ( IZNXT(I).lt.0.and.IZSS(LSSRC(I)).eq.KSZ ) then
604:      NIBP = NIBP + 1
605:      IBP(NIBP) = LSSRC(I)
606:      IZI(NIBP) = -IZNXT(I)
607:      end if
608: 390      continue
609: C
610:      if ( NIBP.eq.0 ) go to 500
611: C
612: C      if ( II.ne.IDEFER ) then
613: C      write(IOW,*) ' !! (SEAONE) NUMBER OF DEFERRED PARTICLES IN ',
614: C      & ' SEARCH STACK IS DIFFERENT FROM IDEFER !! ', II,
615: C      & IDEFER
616: C      stop 666
617: C      end if
618: C
619:      do 400 I = 1, NIBP
620:      X(I) = XXX(IBP(I)) + DEPS*AAA(IBP(I))
621:      Y(I) = YYY(IBP(I)) + DEPS*BBB(IBP(I))
622:      Z(I) = ZZZ(IBP(I)) + DEPS*CCC(IBP(I))
623: 400      continue
624: C
625: C .... CHECK ALL ZONES .....
626: C
627: C      INX = IDEFER
628: C      INX = NIBP
629: C      IDEAD = NDEAD
630: C
631: C      IF(JREFL.NE.0) IRCF = NBREF(NR1)
632: C      IRCF = 0
633: C
634:      if ( JLATT.ne.0 ) ILAT = NXLT(NLBZ+1)
635:      KKZ = 0
636:      MMZ = NZONE
637: C
638:      KKZ1 = 1
639:      KKZ2 = NZONE
640:      KINC = 1
641: C      ... for reversed order zone search
642: C      if ( JREVRS.ne.0 ) then
643: C      if ( KSZ.eq.1 ) then
644: C      KKZ1 = NZONE
645: C      KKZ2 = 1
646: C      KINC = -1
647: C      end if
648: C
649: C      JRR = .false.
650: C      if ( JGMCK.ne.0 ) JRR = .true.

```

src/cgview/seaone.f

```

651: C
652:       do 470 KZ = KKZ1, KKZ2, KINC
653:       if ( INX.eq.0 ) go to 480
654: C
655:       call JUDGE( 'SEARCH', KZ, INX, X, Y, Z, IFI, SDA, KZDA,
656: &               KZAA, NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, JRR,
657: &               KSREF, ISRF, FXYZ )
658: C
659:       II      = 0
660: C
661: C     .. set zone # range in a cell
662: C
663:       if ( JLATT.ne.0 ) then
664:         ICEL   = KCELL(KZ)
665:         MMZ    = IPCEL(ICEL+1) - 1
666:         if ( ICEL.gt.0 ) then
667:           KKZ   = IPCEL(ICEL) - 1
668:         else
669:           KKZ   = 0
670:         end if
671:       end if
672: C
673:       do 410 I = 1, INX
674:         if ( IZI(I).le.KKZ .or. IZI(I).gt.MMZ ) IFI(I) = .false.
675: 410      continue
676: C
677:       if ( JGMCK.ne.0 ) then
678: C
679:         CCCCCC      MPLIMIT = 10
680:         do 420 I = 1, INX
681:           if ( IFI(I).and.KINPZ(KZ).ne.KINPZ(IZI(I))
682: &           .and.KZMAT(IZI(I)).ne.-999 ) then
683:             &
684:             &           (KZMAT(KZ).ne.KZMAT(IZI(I))
685:             &           .or.KZREG(KZ).ne.KZREG(IZI(I))) ) then
686: C
687:               INX0    = 1
688:               MMMMZ   = IZI(I)
689:               call OVLPMMSG( IOW, JTLLT, IMCELL, MMMMZ, KZ,
690: &               DDD2,MPLIMIT,
691: &               NINPZ, NZONE, NSURF, NSUZON, NEST, NSPACE,
692: &               IPSDA, IBSDA, IZNAM, KREGN, KZMAT,
693: &               KZREG, KINPZ, ISUSP,
694: &               INX0, IBP(I), IFI(I),
695: &               FXYZ(I), IGMCK, IPGMCK, XGMCK, NIPGMCK,
696: &               MIPGMCK, XXX, YYY, ZZZ, AAA, BBB, CCC,
697: &               KLSF, LEVL, IBSPC,
698: &               IBNK, KIBNK, MIBNK, NBANK )
699:             end if
700: 420          continue
701:         end if
702: C
703: C     ... for reflector, lattice frame, lattice buffer (-999) zones ...
704: C
705:       if ( KZMAT(KZ).lt.0 ) then
706:         IFFL   = 0
707:       *VOCL LOOP,NOVREC
708:       do 430 I = 1, INX
709:         if ( (.not.IFI(I)) ) then
710:           II    = II + 1
711:           IBP(II) = IBP(I)
712:           X(II)  = X(I)
713:           Y(II)  = Y(I)
714:           Z(II)  = Z(I)
715:           IZI(II) = IZI(I)

```

```

716:       else
717: C
718: C     .... FREE-FLIGHT ....
719: C
720: C     &
721: C     &
722: C     &
723: C
724:       if ( LEVL(IBP(I)).ge.LEVELA.and.KZMAT(KZ).le.-1
725: &
726: &
727: &
728: &
729: &
730: &
731: &
732: &
733: &
734: &
735: &
736: C
737: C     .... LATTICE-SEARCH .....
738: C
739:       else if ( ISLATT(KZMAT(KZ)) .or. KZMAT(KZ).eq.-999
740: &
741: &
742: &
743: &
744: &
745: &
746: C
747: C     .... LEAKAGE & REFLECTION
748: C
749:       else if ( KZMAT(KZ).le.-1000 ) then
750:         IFFL   = IFFL + 1
751:         LSFFL(NFFL(NZONE+1)+IFFL) = IBP(I)
752:         IZZ(IBP(I)) = KZ
753:         IBREG(IBP(I)) = 0
754: C
755:         NDEAD   = NDEAD + 1
756:         IRCF    = IRCF + 1
757:         LSREF(IRCF) = IBP(I)
758:         IZZ(IBP(I)) = KZ
759:         CCCCCC
760:         if ( JTLLT.ne.0 ) IBREG(IBP(I)) = 0
761:       end if
762:     end if
763: 430    continue
764:     if ( IFFL.gt.0 ) then
765:       do 440 I = 1, IFFL
766:         IZFFL(NFFL(NZONE+1)+I) = KZ
767: 440    continue
768:         NFFL(NZONE+1) = NFFL(NZONE+1) + IFFL
769:         NFFL(KZ)      = NFFL(KZ) + IFFL
770:       end if
771: C
772: C     ... KZMAT(KZ) > 0
773: C
774:       else
775: C
776:         IFFL   = 0
777:       *VOCL LOOP,NOVREC
778:       do 450 I = 1, INX
779:         if ( .not.IFI(I) ) then
780:           II    = II + 1

```

src/cgview/seaone.f

```

781:      IBP(II) = IBP(I)
782:      X(II)   = X(I)
783:      Y(II)   = Y(I)
784:      Z(II)   = Z(I)
785:      IZI(II) = IZI(I)
786:      else
787: C
788: C ..... FREE-FLIGHT .....
789: C
790:      IFFL = IFFL + 1
791:      LSFFL(NFFL(NZONE+1)+IFFL) = IBP(I)
792:      IZZ(IBP(I)) = KZ
793: C
794: C ... SET REGION # FOR UNIVERSE-DEPENDENT TALLY MODE ...
795: C
796:      if ( JTLT.ne.0 ) then
797:      ILV = LEVL(IBP(I))
798:      if ( ILV.gt.0.and.KZREG(KZ).gt.0 ) then
799:      IBREG(IBP(I)) =
800: &      ISUSP(KZREG(KZ),IBSPC(IBP(I),ILV))
801:      else
802:      IBREG(IBP(I)) = KZREG(KZ)
803:      end if
804:      else
805:      IBREG(IBP(I)) = KZREG(KZ)
806:      end if
807: C
808: C ..... TANGENTIAL PARTICLES
809: C
810:      if ( IZI(I).eq.KZ ) then
811:      XXX(IBP(I)) = X(I)
812:      YYY(IBP(I)) = Y(I)
813:      ZZZ(IBP(I)) = Z(I)
814:      end if
815:      end if
816: 450      continue
817: C
818:      if ( IFFL.gt.0 ) then
819:      do 460 I = 1, IFFL
820:      IZFFL(NFFL(NZONE+1)+I) = KZ
821: 460      continue
822:      NFFL(NZONE+1) = NFFL(NZONE+1) + IFFL
823:      NFFL(KZ) = NFFL(KZ) + IFFL
824:      end if
825:      end if
826:      INX = II
827: 470      continue
828: C
829: 480      continue
830: C
831: C ..... LOST PARTICLES .....
832: C
833:      if ( INX.ne.0 ) then
834:      NDEAD = NDEAD + INX
835:      write(IOW,*) ' !! (SEAONE) ', INX,
836: &      ' particles are LOST in deferred search mode.'
837:      do 490 I = 1, INX
838:      LSDED(IBP(I)) = 3
839:      write(IOW,7040) IBP(I), IZZ(IBP(I)), X(I), Y(I), Z(I),
840: &      AAA(IBP(I)), BBB(IBP(I)), CCC(IBP(I))
841: 7040      format(1X,I5,' ZONE',I5,' XYZ=',1P,E12.5,E12.5,E12.5,
842: &      ' DIR=',E11.4,E11.4,E11.4)
843:      if ( KLSF(IBP(I)).ne.0 ) then
844:      call SDA2BD( KLSF(IBP(I)), ISRF, IBD, IPSDA, NSURF )
845:      if ( IBD.ne.0 ) then

```

```

846:      write(IOW,*) ' On body ', IBD
847:      end if
848:      end if
849:      if ( JLATT.ne.0.and.LEVL(IBP(I)).ne.0 ) then
850:      write(IOW,7020)
851: &      (L,LZZ(IBP(I),L),LPOS(IBP(I),L),L=1,
852: &      LEVL(IBP(I)))
853:      end if
854: 490      continue
855:      end if
856: C
857:      if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
858: C
859: 500      continue
860: C
861:      if ( IRCF.gt.0 ) then
862:      do 510 I = 1, IRCF
863:      LSFFL(NFFL(NZONE+1)+I) = LSREF(I)
864: 510      continue
865:      end if
866:      NBREF(1) = IRCF
867: C
868: C .... COMPRESS SEARCH STACK .....
869: C
870:      II = 0
871: *VOCL LOOP,NOVREC
872:      do 520 I = 1, NNXT(NZONE+1)
873:      if ( IZNXT(I).ge.0 ) then
874:      II = II + 1
875:      LSSRC(II) = LSSRC(I)
876:      IZNXT(II) = IZNXT(I)
877:      end if
878: 520      continue
879:      NNXT(NZONE+1) = NNXT(NZONE+1) - IDEFER
880: C      if ( JLATT.ne.0 ) NXLT(NLBZ+1) = ILAT
881:      IDEFER = 0
882: C
883:      return
884:      end
885: C
886: C =====
887: C
888:      subroutine SDA2BD( KL, ISRF, IBD, IPSDA, NSURF )
889:      integer IPSDA(3,NSURF+1)
890: C
891:      do 100 I = 1, NSURF
892:      if ( KL.eq.IPSDA(1,I) ) then
893:      ISRF = I
894:      IBD = IPSDA(2,I)
895:      return
896:      end if
897: 100      continue
898:      ISRF = 0
899:      IBD = 0
900:      return
901:      end

```

src/cgview/selone.f

```

1:      subroutine SELONE( MACT, MZONE, NZONE, NBANK, NHIST, NTGEN,
2:      &                  NPART, NGENE, NFFL, NNXT, NBREF, NREFS,
3:      &                  JREFL, IDEFER, NCOLS, NDEAD, JLATT, NXLT, NLBZ,
4:      &                  JSTOP )
5: C=====
6: C  PURPOSE: SELECT NEXT ACTION (ZONE-SELECTION GEOMETRY TRACKING)
7: C  CALLED IN: ACTION
8: C=====
9:      integer NFFL(NZONE+1), NNXT(NZONE+1), NBREF(NREFS+1)
10:     integer NXLT(NLBZ+1)
11: C
12:     integer MMM(6), MZZ(6)
13:     integer JSTOP
14: C
15:     include 'INC/_MVIEW'
16: C
17: C ..... LOCAL DATA FOR ENDLESS RANDOM-WALK DETECTION ....
18: C
19:     real*8 DESUM, DESUM2
20:     integer JLOOP, NLOOP
21: C
22:     data MZZ(3), MZZ(4), MMM(5), MMM(6) /0, -1, 0, 0/
23: C
24:     data DESUM /0.0D0/, DESUM2 /0.0D0/, JLOOP /0/, NLOOP /0/
25: C
26: C .... IF ALL PARTICLES ARE DEAD, GENERATE NEXT PARTICLES ....
27: C
28:     if ( JSTOP.eq.0.and.NBANK.eq.NDEAD ) then
29:         NGENE = MIN(NPART-NTGEN,NHIST)
30:         if ( NGENE.eq.0 ) go to 100
31:         MACT = 0
32:         MMAX = 0
33:         MZONE = 0
34:         if ( NTGEN+NGENE.eq.NPART ) JSTOP = 1
35:         go to 140
36:     end if
37: C
38: C .... CHECK FLIGHT STACK .....
39:     100 MM = 0
40:     MZ = 0
41:     if ( NFFL(NZONE+1).gt.0 ) then
42: *VOCL LOOP,NOVREC
43:         do 110 KZ = 1, NZONE
44:             if ( NFFL(KZ).gt.MM ) then
45:                 MM = NFFL(KZ)
46:                 MZ = KZ
47:             end if
48:         110 continue
49:     end if
50:     MMM(1) = MM
51:     MZZ(1) = MZ
52: C
53: C .... CHECK NEXT ZONE SEARCH STACK .....
54:     MM = 0
55:     MZ = 0
56:     if ( NNXT(NZONE+1).gt.0 ) then
57: *VOCL LOOP,NOVREC
58:         do 120 KZ = 1, NZONE
59:             if ( NNXT(KZ).gt.MM ) then
60:                 MM = NNXT(KZ)
61:                 MZ = KZ
62:             end if
63:         120 continue
64:     end if
65:     MMM(2) = MM

```

```

66:     MZZ(2) = MZ
67: C
68: C .... CHECK COLLISION STACK .....
69:     MMM(3) = 0
70:     if ( NCOLS.gt.(NBANK-NDEAD)*0.5 .or. MMM(1).eq.0.and.MMM(2).eq.0 )
71:         & MMM(3) = NCOLS
72: C
73: C .... CHECK DEFERRED PARTICLE .....
74:     MMM(4) = IDEFER
75: C
76: C .... CHECK REFLECTION STACK .....
77:     if ( JREFL.ne.0 ) then
78:         MMM(5) = 0
79:         if ( NBREF(NREFS+1).gt.MMM(3) ) MMM(5) = NBREF(NREFS+1)
80:     end if
81: C
82: C .... CHECK LATTICE-SEARCH STACK .....
83:     if ( JLATT.ne.0 ) then
84:         MMM(6) = 0
85:         if ( NXLT(NLBZ+1).gt.MMM(3) ) MMM(6) = NXLT(NLBZ+1)
86:     end if
87: C
88: C ....
89:     MMAX = 0
90:     MACT = 0
91: C
92: *VOCL LOOP,SCALAR
93:     do 130 M = 1, 6
94:         if ( MMM(M).gt.MMAX ) then
95:             MMAX = MMM(M)
96:             MACT = M
97:             MZONE = MZZ(M)
98:         end if
99:     130 continue
100:     if ( JSTOP.eq.1.and.MMAX.eq.0 ) MACT = 99
101:     if ( MACT.eq.4 ) MACT = 2
102: C
103: C-----
104: C DETECTION OF ENDLESS RANDOM-WALK
105: C-----
106: C
107: C JLOOP = 0 : NON-DETECTION MODE
108: C
109: C     NUMBER OF REMAINING PARTICLES ARE MORE THAN 1% OF 'NHIST'.
110: C
111: C JLOOP = 1 : DETECTION MODE
112: C
113: C     IF IN A BATCH
114: C     NUMBER OF REMAINING PARTICLES ARE LESS THAN 2 OR 1% OF 'NHIST'.
115: C
116: C     SUM OF PROCESSED-PARTICLES IN DETECTION MODE
117: C     WHEN ----- > 20 %
118: C     SUM OF PROCESSED-PARTICLES IN THE BATCH
119: C
120: C     ENTER MONITORING-MODE.
121: C
122: C JLOOP = 2 : MONITORING MODE
123: C
124: C     PRINT SELECTED EVENT, ZONE ETC. UPTO 30 TIMES.
125: C     AND TERMINATE THE BATCH.
126: C
127: C
128: C
129:     if ( MACT.ne.0.and.MACT.ne.99 ) then
130: C

```

src/cgview/selone.f

```

131: C      .... SUMMATION OF NUMBER OF PROCESSED PARTICLES IN THIS BATCH ..
132: C
133: C      DESUM      = DESUM + MMAX
134: C
135: C
136: C      if ( JLOOP.eq.0.and.NBANK-NDEAD.le.MAX(INT(NGENE*0.01),2) )
137: C      &      then
138: C      @      NBANK-NDEAD .LE. MAX( INT(NHIST*0.01) ,2) ) THEN
139: C      DESUM2 = 0.0
140: C      JLOOP = 1
141: C      end if
142: C
143: C      if ( JLOOP.eq.1 ) then
144: C      DESUM2 = DESUM2 + MMAX
145: C      CCCCCCCCCC if ( DESUM2/DESUM.gt.0.2 ) then
146: C      if ( DESUM2/DESUM.gt.10.0 ) then
147: C      write(6,'(1X,A,I4,A,I3,A/)') 'XXX PROBABLY ', NBANK
148: C      &      - NDEAD, ' PARTICLES ARE '//
149: C      &      'IN ENDLESS LOOP. ( MONITOR ', 30, ' EVENTS )'
150: C      JLOOP = 2
151: C      NLOOP = 0
152: C      end if
153: C      end if
154: C
155: C      .... PROBABLE ENDLESS LOOP .....
156: C
157: C      ( MONITOR PRINT UP TO 30 EVENTS )
158: C
159: C      if ( JLOOP.eq.2 ) then
160: C      NLOOP = NLOOP + 1
161: C
162: C      if ( NLOOP.lt.30 ) then
163: C      write(6,*) ' EVENT :', MACT, ' MZONE ', MZONE, ' MMAX ',
164: C      &      MMAX
165: C      else
166: C
167: C      .... MORE THAN 30 EVENTS MONITORED. TERMINATE BATCH. ....
168: C
169: C      write(6,'(1X,A,I4,A,I3,A/)') 'XXX PROBABLY ', NBANK
170: C      &      - NDEAD, ' PARTICLES ARE '//
171: C      &      'IN ENDLESS LOOP AFTER MONITORING ', 30,
172: C      &      ' EVENTS.'
173: C
174: C      .... CLEAR EVENT STACKS HERE ....
175: C
176: C      NDEAD = NBANK
177: C      call PUTV( NFFL, NZONE+1, 0 )
178: C      call PUTV( NNXT, NZONE+1, 0 )
179: C      if ( JREFL.ne.0 ) call PUTV( NBREF, NREFS+1, 0 )
180: C      if ( JLATT.ne.0 ) call PUTV( NXLT, NLBZ+1, 0 )
181: C
182: C      NCOLS = 0
183: C      NCOLP = 0
184: C      IDEFER = 0
185: C      MACT = 0
186: C      if ( JSTOP.eq.1 ) MACT = 99
187: C      end if
188: C      end if
189: C      end if
190: C
191: C      .... END OF SELONE ...
192: C
193: C      140 continue
194: C
195: C      .... RESET ENDLESS-LOOP MONITORING PARAMETERS ....

```

```

196: C
197: C      if ( MACT.eq.0 .or. MACT.eq.99 ) then
198: C      DESUM = 0.0
199: C      DESUM2 = 0.0
200: C      JLOOP = 0
201: C      NLOOP = 0
202: C      end if
203: C
204: C      if (JCGVDB(2).gt.0) then
205: C      write(6,*) '%% EVENT :', MACT, ' MZONE ', MZONE, ' MMAX ',MMAX,
206: C      &      ' NDEAD ',NDEAD
207: C      end if
208: C      if (JCGVDB(2).ge.2) then
209: C      write(6,'(1x,3(a,i7))') 'NCOLS', NCOLS, 'NCOLP', NCOLP,
210: C      &      'NDEAD', NDEAD
211: C      write(6,7140) 'NFFL', NFFL
212: C      write(6,7140) 'NNXT', NNXT
213: C      if ( JLATT.ne.0 ) write(6,7140) 'NXLT', NXLT
214: C      if ( JREFL.ne.0 ) write(6,7140) 'NBREF', NBREF
215: C      7140      format(1X,'<',A,'>',10I7/(5X,10I7))
216: C      end if
217: C
218: C      return
219: C      end

```

src/cgview/showbd.f

```

1:      subroutine SHOWBD( IPR,   IBBBB, JTLLT, NINPZ, NZONE, NBODY, NREG,
2:      &                  NZDA, NSDA, NSURF, DBODI, IBODI, IPSDA,
3:      &                  KINPZ, KZDA, KZAA, SDA, KZMAT, KMAT,
4:      &                  KSFB, IZNAM, IBSDA, KREG, KREGN, KREGI, TNAMS
5:      &                  )
6: C=====
7: C Purpose: print geometry information of specified body (IBBBB).
8: C
9: C <<attention>> array KMAT and KZMAT contain material ID as input
10: C when called in CGVIEW code which does not convert
11: C material IDs to material numbers.
12: C
13: C-----
14: include '../shared/INC/ LNAM'
15: C
16:      real*8 DBODI(*)
17:      integer IBODI(NBODY+1)
18:      integer IPSDA(3,NSURF+1)
19: C
20:      integer KINPZ(NZONE+1), KZDA(2,NZDA), KZAA(NZONE+1)
21:      real*8 SDA(NSDA)
22:      integer KZMAT(NZONE,2), KMAT(NINPZ)
23:      integer KSFB(NZDA)
24:      integer IBSDA(3,NBODY)
25:      integer KREG(NINPZ)
26:      character*12 KREGN(NINPZ)
27:      integer KREGI(NINPZ)
28:      character*12 IZNAM(*)
29:      character*(LNAM) TNAMS(*)
30: C
31: C ... local data
32: C
33:      character*3 TYPE
34:      character*80 LIN
35: C
36: C-----
37: C
38:      KB = 0
39:      do 100 KB = 1, NBODY
40:      if ( IBSDA(2,KB).eq.IBBBB ) then
41:      go to 120
42:      end if
43: 100 continue
44: C
45:      if ( IBBBB.ne.0 ) then
46:      write(IPR,*) 'xxx body ', IBBBB, ' does not exist.'
47:      end if
48: C
49:      write(IPR,7000)
50: 7000 format(1X,'>> Possible body IDs')
51:      NB = 1
52:      LIN = ' '
53:      do 110 I = 1, NBODY
54:      call TYPBOD( TYPE, '<', IBSDA(3,I) )
55:      write(LIN(NB+11),'(I6,'('',A,'') ''') IBSDA(2,I),
56:      &          TYPE(1:3)
57:      NB = NB + 12
58:      if ( NB+11.gt.LEN(LIN) ) then
59:      write(IPR,7020) LIN(:ICLEN2(LIN))
60: 7020 format(1X,'>> ',A)
61:      NB = 1
62:      LIN = ' '
63:      end if
64: 110 continue
65:      if ( NB.gt.1 ) write(IPR,7020) LIN(:ICLEN2(LIN))

```

```

66:      return
67: C
68: 120 continue
69: C
70: C ... get body type
71: C
72:      call TYPBOD( TYPE, '<', IBSDA(3,KB) )
73: C
74:      write(IPR,*)
75: C
76:      write(IPR,*) '>> Body ', IBBBB, ' : type <', TYPE, '>' (# ', KB,
77:      &          ')
78: C
79:      if ( TYPE.eq.'BOX' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'V1X',
80:      & 'V1Y', 'V1Z', 'V2X', 'V2Y', 'V2Z', 'V3X', 'V3Y', 'V3Z'
81:      if ( TYPE.eq.'RPP' ) write(IPR,7040) 'X1', 'X2', 'Y1', 'Y2', 'Z1',
82:      & 'Z2'
83:      if ( TYPE.eq.'SPH' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'R'
84:      if ( TYPE.eq.'RCC' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'HX', 'HY',
85:      & 'HZ', 'R'
86:      if ( TYPE.eq.'RCL' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'HX', 'HY',
87:      & 'HZ', 'R', 'VX', 'VY', 'VZ'
88:      if ( TYPE.eq.'CYL' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'H', 'R'
89:      if ( TYPE.eq.'ELL' ) write(IPR,7040) 'F1X', 'F1Y', 'F1Z', 'F2X',
90:      & 'F2Y', 'F2Z', 'D'
91:      if ( TYPE.eq.'GEL' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'V1X',
92:      & 'V1Y', 'V1Z', 'V2X', 'V2Y', 'V2Z', 'L1', 'L2', 'L3'
93:      if ( TYPE.eq.'TRC' ) write(IPR,7040) 'C1X', 'C1Y', 'C1Z', 'C2X',
94:      & 'C2Y', 'C2Z', 'R1', 'R2'
95:      if ( TYPE.eq.'RHP' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'H', 'DX'
96:      if ( TYPE.eq.'HEX' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'HX', 'HY',
97:      & 'HZ', 'VX', 'VY', 'VZ', 'D'
98:      if ( TYPE.eq.'ARB' ) write(IPR,7040) 'C1X', 'C1Y', 'C1Z', 'C2X',
99:      & 'C2Y', 'C2Z', 'C3X', 'C3Y', 'C3Z', 'C4X', 'C4Y', 'C4Z', 'C5X',
100:      & 'C5Y', 'C5Z', 'C6X', 'C6Y', 'C6Z', 'C7X', 'C7Y', 'C7Z', 'C8X',
101:      & 'C8Y', 'C8Z', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6'
102:      if ( TYPE.eq.'WED'.or. TYPE.eq.'RAW' ) write(IPR,7040) 'CX',
103:      & 'CY', 'CZ', 'V1X', 'V1Y', 'V1Z', 'V2X', 'V2Y', 'V2Z', 'V3X',
104:      & 'V3Y', 'V3Z'
105:      if ( TYPE.eq.'HAF' ) write(IPR,7040) 'NX', 'NY', 'NZ', 'D'
106:      if ( TYPE.eq.'ELT' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'NX', 'NY',
107:      & 'NZ', 'RM', 'R1', 'R2', 'TLT'
108:      if ( TYPE.eq.'TEC' ) write(IPR,7040) 'CX', 'CY', 'CZ', 'HX', 'HY',
109:      & 'HZ', 'AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'RT'
110:      if ( TYPE.eq.'GQS' ) write(IPR,7040) 'X2', 'Y2', 'Z2', 'XY', 'YZ',
111:      & 'ZX', 'X ', 'Y ', 'Z ', 'C '
112:      if ( TYPE.eq.'BBC' ) write(IPR,7040) 'Bod1', 'Bod2', '...'
113: C
114:      if ( TYPE.eq.'BBC' ) then
115:      write(IPR,7080) (
116:      &          INT(SIGN(1D0,DBODI(I))*IBSDA(2,NINT(ABS(DBODI(I))))),
117:      &          I=IBODI(KB),IBODI(KB+1)-1)
118:      else
119:      write(IPR,7060) (DBODI(I),I=IBODI(KB),IBODI(KB+1)-1)
120:      end if
121: C
122: 7040 format(1X,'>',6(' <',A4,'>' :)/1X,'>',6(' <',A4,'>' :),/
123:      & 1X,'>',6(' <',A4,'>' :)/1X,'>',6
124:      & (' <',A4,'>' :)/1X,'>',6(' <',A4,'>' :))
125: 7060 format(1X,'>',1P,6(E13.5)/1X,'>',1P,6(E13.5)/1X,'>',1P,6
126:      & (E13.5)/1X,'>',1P,6(E13.5)/1X,'>',1P,6(E13.5))
127: 7080 format(1X,'>',1P,6(2X,I8,3X:)/1X,'>',1P,6(2X,I8,3X:)/1X,'>',1P,6
128:      & (2X,I8,3X:)/1X,'>',1P,6(2X,I8,3X:)/1X,'>',1P,6
129:      & (2X,I8,3X:))
130: C

```

src/cgview/showbd.f

```
131:      if ( TYPE.eq.'BBC' ) then
132:        do 140 IB = IBODI(KB), IBODI(KB+1) - 1
133:          KB      = NINT(DBODI(IB))
134:          KB      = ABS(KB)
135:          write(IPR,*) '> --> Body ', IBSDA(2,ABS(KB))
136:          do 130 ISP = IBSDA(1,KB), IBSDA(1,KB+1) - 1
137:            call SHOWSF( ISP, IPR, IPSDA, SDA, NSURF, NSDA )
138:          130 continue
139:        140 continue
140:      else
141:        do 150 ISP = IBSDA(1,KB), IBSDA(1,KB+1) - 1
142:          call SHOWSF( ISP, IPR, IPSDA, SDA, NSURF, NSDA )
143:        150 continue
144:      end if
145: C
146:      return
147: end
148: C
149: C=====
150: C
151:      subroutine SHOWSF( ISP, IPR, IPSDA, SDA, NSURF, NSDA )
152: C
153:      integer IPSDA(3,NSURF+1)
154:      real*8 SDA(NSDA)
155: C
156:      IS      = IPSDA(1,ISP)
157:      KSF      = NINT(SDA(IS))
158: 7000 format(1X,'>> Surface #',I4,' Type <',A,'>')
159: 7020 format(1X,'>> ',1P,5E13.5)
160: C
161:      if ( KSF.eq.1 ) then
162:        write(IPR,7000) ISP, 'Slab'
163:        write(IPR,7020) (SDA(IS+I),I=1,5)
164:      else if ( KSF.eq.2 ) then
165:        write(IPR,7000) ISP, 'Sphere'
166:        write(IPR,7020) (SDA(IS+I),I=1,4)
167:      else if ( KSF.eq.3 ) then
168:        write(IPR,7000) ISP, 'Cylinder'
169:        write(IPR,7020) (SDA(IS+I),I=1,7)
170:      else if ( KSF.eq.4 ) then
171:        write(IPR,7000) ISP, 'Quadratic'
172:        write(IPR,7020) (SDA(IS+I),I=1,10)
173:      else if ( KSF.eq.5 ) then
174:        write(IPR,7000) ISP, 'Half space'
175:        write(IPR,7020) (SDA(IS+I),I=1,4)
176:      else if ( KSF.eq.6 ) then
177:        write(IPR,7000) ISP, 'Torus'
178:        write(IPR,7020) (SDA(IS+I),I=1,10)
179:      else
180:        write(IPR,*) 'xx unknown surface type !!! : ', KSF
181:      end if
182: C
183:      return
184: end
```


src/cgview/showmt.f

```
1:      subroutine SHOWMT( IPR,   IMMMM, CODE,  JNEUT, JPHOT, NMAT,  NUC,
2:      &                  NPATOM,NINPZ, NZONE, IDMAT, MNUC,  INUCT,
3:      &                  DENST, LPDEN, MPATM, NATMT, IATMT, DNSTP,
4:      &                  LPDNP, NCIDI, NUCID, TEMPN, KZMAT, KMAT )
5: C=====
6: C Purpose: print MVP material composiotion information (IMMMM).
7: C
8: C      IMMMM > 0 : material ID
9: C      IMMMM < 0 : all material
10: C
11: C <<attention>> array KMAT and KZMAT contain material ID as input
12: C               when called in CGVIEW code which does not convert
13: C               material IDs to material numbers.
14: C
15: C=====
16:      character*(*) CODE
17: C
18:      integer IDMAT(NMAT), MNUC(NMAT), LPDEN(NMAT+1)
19:      integer INUCT(*)
20:      real DENST(*)
21:      character*16 NCIDI(NUC), NUCID(NUC)
22:      real*8 TEMPN(NUC)
23: C
24:      integer MPATM(NMAT), LPDNP(NMAT+1), NATMT(NPATOM)
25:      integer IATMT(*)
26:      real DNSTP(*)
27: C
28: C
29:      integer KZMAT(NZONE,2), KMAT(NINPZ)
30: C
31: C ... local data
32: C
33: C-----
34: C
35:      if ( IMMMM.gt.0 ) then
36:      do 100 IM = 1, NMAT
37:      if ( IDMAT(IM).eq.IMMMM ) then
38:      go to 110
39:      end if
40: 100 continue
41:      write(IPR,*) 'xxx material with ID ', IMMMM, ' does not exist.'
42:      write(IPR,'(1X, ''=== List of material ID =='')')
43:      write(IPR,'(1X,10I7)') (IDMAT(I),I=1,NMAT)
44:      return
45: 110 continue
46:      IM1 = IM
47:      IM2 = IM
48:      else
49:      IM1 = 1
50:      IM2 = NMAT
51:      end if
52: C
53:      write(IPR,*) '>>'
54:      do 130 IM = IM1, IM2
55:      write(IPR,*) '>> == Material ', IDMAT(IM), ' : ', MNUC(IM),
56:      &          ' nuclides'
57:      write(IPR,7000)
58:      do 120 N = LPDEN(IM), LPDEN(IM+1) - 1
59:      K = INUCT(N)
60:      call MCNNID( NCIDI(K), MMMM, IE )
61:      write(IPR,7020) NCIDI(K), NUCID(K), DENST(N), TEMPN(K), MMMM
62: 120 continue
63: 7000 format(1X,'>> Nuc ID(input)      Nuc ID(internal) Num density',
64:      &          ' Temperature')
65: 7020 format(1X,'>> ',A,2X,A,E15.7,E15.7,I7)
66:      130 continue
67: C
68:      return
69:      end
```

src/cgview/showzn.f

```

1:      subroutine SHOWZN( IPR,  IZZZZ, NINPZ, NZONE, NBODY, NREG,  NZDA,
2:      &                  NSDA, NCELL, KINPZ, KZDA, KZAA, SDA,  KMAT,
3:      &                  KZMAT, KCELL, IDCEL, KSFBD, IZNAM, KBZL,  IBZL,
4:      &                  IBSDA, KREG, KREGN, KREGI, TNAMS )
5: C=====
6: C Purpose: print geometry information of specified zone (IZZZZ).
7: C
8: C      IZZZZ > 0 : input-zone IZZZZ
9: C      IZZZZ < 0 : zone abs(IZZZZ)
10: C
11: C <<attention>> array KMAT and KZMAT contain material ID as input
12: C      when called in CGVIEW code which does not convert
13: C      material IDs to material numbers.
14: C
15: C-----
16:      include 'INC/_FLAGS'
17:      include '../shared/INC/_LNAME'
18: C
19:      integer KINPZ(NZONE), KZDA(2,NZDA), KZAA(NZONE+1)
20:      real*8 SDA(NSDA)
21:      integer KZMAT(NZONE,2), KMAT(NINPZ)
22:      integer KCELL(NZONE)
23:      integer IDCEL(NCELL)
24:      integer KSFBD(NZDA)
25:      integer KBZL(*), IBZL(NZONE+1)
26:      integer IBSDA(3,NBODY)
27:      integer KREG(NINPZ)
28:      character*12 KREGN(NINPZ)
29:      integer KREGI(NINPZ)
30:      character*12 IZNAM(NINPZ)
31:      character*(LNAME) TNAMS(*)
32: C
33: C ... local data
34: C
35: C      parameter( NLB = 8 )
36: C      parameter( NLB = 5 )
37: C      integer IBN(NLB)
38: C      character*4 BDTYP(NLB)
39: C      character*8 CWK
40: C
41: C-----
42: C
43:      KZ      = IZZZZ
44:      if ( IZZZZ.gt.0 ) then
45:          if ( IZZZZ.gt.NINPZ ) then
46:              write(IPR,*) 'xxx input zone ', IZZZZ,
47:              &          ' does not exist (ninpz=', NINPZ, ' )'
48:              return
49:          end if
50:      else if ( IZZZZ.lt.0 ) then
51:          KZ      = -IZZZZ
52:          if ( KZ.gt.NZONE ) then
53:              write(IPR,*) 'xxx zone ', KZ, ' does not exist (nzone=',
54:              &          NZONE, ' )'
55:              return
56:          end if
57:      else
58:          return
59:      end if
60: C
61: C
62:      write(IPR,*) '>>'
63:      if ( IZZZZ.gt.0 ) then
64: C
65:          K1      = 0

```

```

66:          K2      = NZONE
67:          do 100 I = 1, NZONE
68:              if ( K1.eq.0.and.KINPZ(I).eq.KZ ) then
69:                  K1      = I
70:              else if ( K1.ne.0.and.KINPZ(I).ne.KZ ) then
71:                  K2      = I - 1
72:                  go to 110
73:              end if
74:          100 continue
75: C
76:          110 continue
77: C
78:          if ( JTLT.eq.0 ) then
79:              write(IPR,*) '>>Input zone ', KZ, ' name=<',
80:              &          IZNAM(KZ) (:ICLEN2(IZNAM(KZ))), '>' material= ',
81:              &          KMAT(KZ), ' region#=', KREG(KZ), '<',
82:              &          TNAMS(KREGI(KZ)) (:ICLEN2(TNAMS(KREGI(KZ)))), '>'
83:          else
84:              write(IPR,*) '>>Input zone ', KZ, ' name=<',
85:              &          IZNAM(KZ) (:ICLEN2(IZNAM(KZ))), '>' material= ',
86:              &          KMAT(KZ)
87:          end if
88:          if ( JLATT.ne.0.and.KCELL(K1).ne.0 ) then
89:              write(IPR,*) '>> from zone ', K1, ' to ', K2, ', cell ID ',
90:              &          IDCEL(KCELL(K1))
91:          else
92:              write(IPR,*) '>> from zone ', K1, ' to ', K2
93:          end if
94: C
95:      else
96:          write(IPR,*) '>>Zone ', KZ, ' input-zone=', KINPZ(KZ), ' <',
97:          &          IZNAM(KINPZ(KZ)) (:ICLEN2(IZNAM(KINPZ(KZ)))),
98:          &          '>' material= ', KZMAT(KZ,1), ' (mat2 = ', KZMAT(KZ,2),
99:          &          ' )'
100:          if ( JLATT.ne.0.and.KCELL(KZ).ne.0 ) then
101:              write(IPR,*) '>> cell ID ', IDCEL(KCELL(KZ))
102:          end if
103:          K1      = KZ
104:          K2      = KZ
105:      end if
106: C
107: C
108: C
109: C
110:      do 130 IZ = K1, K2
111:          if ( K2.gt.K1 ) write(IPR,*) '>>* Zone ', IZ, ':'
112: C
113:          ILB      = 0
114:          IB        = 0
115:          CWK      = '>> Body:'
116: C
117:          KEND      = IBZL(IZ+1) - 1
118:          do 120 J = IBZL(IZ), KEND
119:              ILB      = ILB + 1
120:              KB        = KBZL(J)
121:              KBA        = ABS(KB)
122:              call TYPBOD( BDTYP(ILB), '<', IBSDA(3,KBA) )
123:              IBN(ILB)    = IBSDA(2,KBA)*SIGN(1,KB)
124:              if ( ILB.eq.NLB .or. (ILB.gt.0.and.J.eq.KEND) ) then
125:                  write(IPR,7000) CWK,
126:                  &          (IBN(I),BDTYP(I)(1:ICLEN2(BDTYP(I))),I=1,ILB)
127:              ILB      = 0
128:              CWK      = '>>'
129:          end if
130:      120 continue

```

src/cgview/showzn.f

```
131: 7000    format(1X,A,5(I6,'(',A3,')':))
132: C
133: 130 continue
134: C
135:      return
136:      end
```

SAFE

src/cgview/source.f

```

1:      subroutine SOURCE( IOW,  IRAND, JREVR, PAPER, GRANGE, XXX,  YYY,
2:      &                   ZZZ,  AAA,  BBB,  CCC,  IZZ,  NBANK,
3:      &                   NZONE, NEST,  LEVL,  LZZ,  LPOS,  LCRS,  KLSF,
4:      &                   IBREG, IBSPC, IZSS,  AAAG, BBBG, CCCG,
5:      &                   GDIST, KGCNT, DBNK, KDBNK, MDBNK, IBNK,
6:      &                   KIBNK, MIBNK, NMEMS, LSFFL, NFFL,  IZFFL,
7:      &                   LSDED, NDEAD, KZMAT, KINPZ, IZNAM, KZREG, KREGN,
8:      &                   NINPZ, NMAT, NREG,  SDA,  KZDA,  KZAA,  NSDA,
9:      &                   NZDA, KCELL, IPCEL, NLBZ,  MLBZZ, DINF,  DEPS,
10:     &                   KSREF, IGMCK, IPGMCK, XGMCK, LSLAT, IZLAT, NXLT,
11:     &                   XPLT, YPLT,  LSCAN, ILOOP, NGENE, NGENE0,
12:     &                   NTGEN, NPART, IWK1,  X,  Y,  Z,  IFL,
13:     &                   IFI,  R,  IWK2,  IZW2,  FXYZ2 )
14: C=====
15: C  PURPOSE: GENERATE PARTICLES
16: C  CALLED IN: ACTION
17: C  CALLS:
18: C=====
19:
20:      implicit real*8(D)
21:
22: C/#IF MS_VISUAL
23: C
24: C ... This interface block is for CUTIL & MS-Visual tools. ...
25: C
26: *      interface
27: *          subroutine CAPLINE( IIII )
28: *              integer      IIII
29: *CDEC$      ATTRIBUTES C :: CAPLINE
30: *CDEC$      ATTRIBUTES REFERENCE :: IIII
31: *          end subroutine
32: *      end interface
33: C/#ENDIF
34:
35:      include 'INC/_FLAGS'
36:      include 'INC/_MVIEW'
37: C
38: C .... PARTICLE BANK .....
39:
40:      real*8 XXX(NBANK), ZZZ(NBANK), YYY(NBANK), AAA(NBANK), BBB(NBANK),
41:      &      CCC(NBANK)
42:      integer IZZ(NBANK), LEVL(NBANK), LZZ(NBANK,NEST),
43:      &      LPOS(NBANK,NEST), LCRS(NBANK,NEST), KLSF(NBANK),
44:      &      IBREG(NBANK), IBSPC(NBANK,0:NEST)
45: C
46: C
47:      real*8 DBNK(NBANK,*)
48:      integer KDBNK(0:MDBNK)
49:      integer IBNK(NBANK,*)
50:      integer KIBNK(0:MIBNK)
51:
52: C .... STACKS ...(LATTICE SEARCH) .....
53:
54:      integer LSFFL(NBANK), NFFL(NZONE+1), IZFFL(NBANK)
55:      integer LSLAT(NBANK), NXLT(NLBZ+1), IZLAT(NBANK), LSDED(NBANK)
56:
57:
58: C .... GEOMETRY DATA
59:
60:      integer KZMAT(NZONE), KINPZ(NZONE), KZREG(NZONE), KZDA(2,NZDA),
61:      &      KZAA(NZONE+1), MLBZZ(NZONE), IPCEL(1), KCELL(NZONE)
62:      integer KSREF(NZDA)
63:      character*12 IZNAM(NINPZ)
64:      character*12 KREGN(NINPZ)
65:      real*8 SDA(NSDA)

```

```

66:      real*8 DINF, DEPS
67:      real*8 GRANGE(6)
68: C
69: C .... CGVIEW specific
70: C
71:      real*8 PAPER(*)
72: CCC      real*8 DX, DY, XMAX, YMAX
73:      real*8 XPLT(NBANK), YPLT(NBANK)
74:      integer IZSS(NBANK)
75:      real*8 AAAG(NBANK), BBBG(NBANK), CCCG(NBANK), GDIST(NBANK)
76:      integer KGCNT(NBANK)
77:      integer LSCAN(NBANK)
78: CCC      integer LEVEL, SPTYP
79: C
80:      integer IGMCK(NZONE), IPGMCK(3,MIPGMCK)
81:      real*8 XGMCK(4,MIPGMCK)
82: C
83: C .... working array
84: C
85:      integer IWK1(NBANK)
86:      logical*4 IFI(NBANK), IFL(NBANK)
87:      real*8 X(NBANK), Y(NBANK), Z(NBANK)
88:      real R(4*NBANK)
89:      integer IWK2(NBANK)
90:      integer IZW2(NBANK,0:1)
91:      real FXYZ2(NBANK,0:1)
92: C
93:      parameter( PAI = 3.1415926535897932D0 )
94:      parameter( DPAI = 2D0*PAI )
95: C
96:      include '../shared/INC/_PMLATT'
97:      include '../shared/INC/_SFLATT'
98: C
99: C=====
100: C  create sources on starting points of line scanning.
101: C=====
102: C
103:      NN      = NGENE
104:      NDEAD    = NBANK - NGENE
105: C
106:      JREVR    = 0
107:      if ( JZREVERSE.ne.0 ) then
108:          JREVR = 1
109:      end if
110: C
111: C-----
112: C  ... particle flooding mode .....
113: C-----
114: C
115:      if ( JGFLOOD.ne.0 ) then
116: C
117:          DG1      = GRANGE(1) - 0.05*(GRANGE(2)-GRANGE(1))
118:          DG2      = GRANGE(2) + 0.05*(GRANGE(2)-GRANGE(1))
119:          DG3      = GRANGE(3) - 0.05*(GRANGE(4)-GRANGE(3))
120:          DG4      = GRANGE(4) + 0.05*(GRANGE(4)-GRANGE(3))
121:          DG5      = GRANGE(5) - 0.05*(GRANGE(6)-GRANGE(5))
122:          DG6      = GRANGE(6) + 0.05*(GRANGE(6)-GRANGE(5))
123: C
124:      call RANU2( IRAND, R, 3*NN, ICODE )
125:      do I = 1, NN
126:          IWK1(I) = I
127:          LSDED(I) = 0
128:          KLSF(I) = 0
129: C
130:          IZZ(I) = 0

```

src/cgview/source.f

```

131: C
132:       if ( JLATT.ne.0 ) LEVL(I) = 0
133: C
134: C       ... JGFLOOD = 2: cosine distribution
135: C             (dense in center)
136: C
137:       if ( ABS(JGFLOOD).eq.2 ) then
138:         XXX(I) = DG1 + (DG2-DG1)*(ASIN(2D0*R(I)-1D0)/PAI+0.5D0)
139:         YYY(I) = DG3 + (DG4-DG3)*
140: &           (ASIN(2D0*R(I+NN)-1D0)/PAI+0.5D0)
141:         ZZZ(I) = DG5 + (DG6-DG5)*
142: &           (ASIN(2D0*R(I+2*NN)-1D0)/PAI+0.5D0)
143: C
144: C       ... JGFLOOD = 1: flat distribution
145: C
146:       else
147:         XXX(I) = DG1 + (DG2-DG1)*R(I)
148:         YYY(I) = DG3 + (DG4-DG3)*R(I+NN)
149:         ZZZ(I) = DG5 + (DG6-DG5)*R(I+2*NN)
150:       end if
151: C
152: C       ... XPLT,YPLT : coordinates on drawing area
153: C
154:       if ( MXYZ.eq.1 ) then
155:         XPLT(I) = XXX(I) - DG1 + (XMAX-DG2+DG1)*0.5
156:         YPLT(I) = YYY(I) - DG3 + (YMAX-DG4+DG3)*0.5
157:       else if ( MXYZ.eq.-1 ) then
158:         XPLT(I) = YYY(I) - DG3 + (XMAX-DG4+DG3)*0.5
159:         YPLT(I) = XXX(I) - DG1 + (YMAX-DG2+DG1)*0.5
160:       else if ( MXYZ.eq.2 ) then
161:         XPLT(I) = XXX(I) - DG1 + (XMAX-DG2+DG1)*0.5
162:         YPLT(I) = ZZZ(I) - DG5 + (YMAX-DG6+DG5)*0.5
163:       else if ( MXYZ.eq.-2 ) then
164:         XPLT(I) = ZZZ(I) - DG5 + (XMAX-DG6+DG5)*0.5
165:         YPLT(I) = XXX(I) - DG1 + (YMAX-DG2+DG1)*0.5
166:       else if ( MXYZ.eq.3 ) then
167:         XPLT(I) = YYY(I) - DG3 + (XMAX-DG4+DG3)*0.5
168:         YPLT(I) = ZZZ(I) - DG5 + (YMAX-DG6+DG5)*0.5
169:       else if ( MXYZ.eq.-3 ) then
170:         XPLT(I) = ZZZ(I) - DG5 + (XMAX-DG6+DG5)*0.5
171:         YPLT(I) = YYY(I) - DG3 + (YMAX-DG4+DG3)*0.5
172:       end if
173: C
174:       KGCNT(I) = 0
175: C
176:       IZSS(I) = 0
177:       if ( JZREVERSE.eq.1 ) then
178:         IZSS(I) = 1
179:       else if ( JZREVERSE.eq.2 ) then
180:         IZSS(I) = MOD(NTGEN+I,2)
181:       end if
182: C
183:     end do
184: C
185:     call RANU2( IRAND, R, 3*NN, ICODE )
186: C
187: *VOCL LOOP,NOVREC
188:     do I = 1, NN
189:       DMU = 2.0D0*R(I) - 1.0D0
190:       DSN = SQRT(1.0D0-DMU*DMU)
191:       DFI = DPAI*R(I+NN)
192: C
193:       AAA(I) = DMU
194:       BBB(I) = DSN*COS(DFI)
195:       CCC(I) = DSN*SIN(DFI)

```

```

196: C
197:       DD1 = DINF
198:       if ( AAA(I).gt.0.0D0 ) then
199:         DD1 = (DG2-XXX(I)) /AAA(I)
200:       else if ( AAA(I).lt.0.0D0 ) then
201:         DD1 = (DG1-XXX(I)) /AAA(I)
202:       end if
203:       DD2 = DINF
204:       if ( BBB(I).gt.0.0D0 ) then
205:         DD2 = (DG4-YYY(I)) /BBB(I)
206:       else if ( BBB(I).lt.0.0D0 ) then
207:         DD2 = (DG3-YYY(I)) /BBB(I)
208:       end if
209:       DD3 = DINF
210:       if ( CCC(I).gt.0.0D0 ) then
211:         DD3 = (DG6-ZZZ(I)) /CCC(I)
212:       else if ( CCC(I).lt.0.0D0 ) then
213:         DD3 = (DG5-ZZZ(I)) /CCC(I)
214:       end if
215:       GDIST(I) = MIN(DD1,DD2,DD3)
216: C
217:       if ( MXYZ.eq.1 ) then
218:         AAAG(I) = AAA(I)
219:         BBBG(I) = BBB(I)
220:         CCCG(I) = CCC(I)
221:       else if ( MXYZ.eq.-1 ) then
222:         AAAG(I) = BBB(I)
223:         BBBG(I) = AAA(I)
224:         CCCG(I) = CCC(I)
225:       else if ( MXYZ.eq.2 ) then
226:         AAAG(I) = AAA(I)
227:         BBBG(I) = CCC(I)
228:         CCCG(I) = BBB(I)
229:       else if ( MXYZ.eq.-2 ) then
230:         AAAG(I) = CCC(I)
231:         BBBG(I) = AAA(I)
232:         CCCG(I) = BBB(I)
233:       else if ( MXYZ.eq.3 ) then
234:         AAAG(I) = BBB(I)
235:         BBBG(I) = CCC(I)
236:         CCCG(I) = AAA(I)
237:       else if ( MXYZ.eq.-3 ) then
238:         AAAG(I) = CCC(I)
239:         BBBG(I) = BBB(I)
240:         CCCG(I) = AAA(I)
241:       end if
242:     end do
243: C
244: C -----
245: C ... line scan mode .....
246: C -----
247: C
248:       else
249: C
250:       do 100 I = 1, NN
251: C       IWK1(I) = I
252:         LSDED(I) = 0
253:         KLSF(I) = 0
254: C
255:         IZZ(I) = 0
256: C
257:         if ( JLATT.ne.0 ) LEVL(I) = 0
258: C
259: C =====
260: C

```

src/cgview/source.f

```

261: C      JSCANX = 1  JSCANX = 2  JSCANX = 3
262: C
263: C      ^^^^^^^^^^  |||||  ^|^|^|^
264: C      |||||  |||||  |||||
265: C      |||||  |||||  |||||
266: C      |||||  |||||  |||||
267: C      |||||  |||||  |||||
268: C      |||||  |||||  |||||
269: C      |||||  vvvvvvvvvv  v|v|v|v|v|
270: C
271: C =====
272: C
273: C      IIX      = NTGEN + I
274: C
275: C      IZSS(I) = 0
276: C      if ( JZREVERSE.eq.1 ) then
277: C          IZSS(I) = 1
278: C      else if ( JZREVERSE.eq.2 ) then
279: C          IZSS(I) = MOD(IIX,2)
280: C      end if
281: C
282: C      if ( ILOOP.eq.1 ) then
283: C          J      = IIX
284: C          if ( JSCANX.eq.3 ) then
285: C              if ( IIX.gt.NPART/2 ) then
286: C                  J      = 2*(IIX-NPART/2) - 1
287: C              else
288: C                  J      = 2*IIX
289: C              end if
290: C          end if
291: C          IMOD      = MOD(J,2)
292: C          if ( JSCANX.eq.1 .or. JSCANX.eq.3.and.IMOD.eq.0 ) then
293: C              XXX(I) = PAPER(1) + PAPER(4)*DXCG*(J-1)
294: C              YYY(I) = PAPER(2) + PAPER(5)*DXCG*(J-1)
295: C              ZZZ(I) = PAPER(3) + PAPER(6)*DXCG*(J-1)
296: C              AAA(I) = PAPER(7)
297: C              BBB(I) = PAPER(8)
298: C              CCC(I) = PAPER(9)
299: C
300: C              XPLT(I) = DXCG*(J-1)
301: C              YPLT(I) = 0.0
302: C
303: C              GDIST(I) = YMAX
304: C              AAAG(I) = 0.0
305: C              BBBG(I) = 1.0
306: C              CCCG(I) = 0.0
307: C              KGCNT(I) = 0
308: C
309: C              LSCAN(I) = +2
310: C          else if ( JSCANX.eq.2 .or. JSCANX.eq.3.and.IMOD.eq.1 )
311: C              &      then
312: C              XXX(I) = PAPER(1) + PAPER(7)*YMAX + PAPER(4)*DXCG*
313: C              &      (J-1)
314: C              YYY(I) = PAPER(2) + PAPER(8)*YMAX + PAPER(5)*DXCG*
315: C              &      (J-1)
316: C              ZZZ(I) = PAPER(3) + PAPER(9)*YMAX + PAPER(6)*DXCG*
317: C              &      (J-1)
318: C              AAA(I) = -PAPER(7)
319: C              BBB(I) = -PAPER(8)
320: C              CCC(I) = -PAPER(9)
321: C
322: C              XPLT(I) = DXCG*(J-1)
323: C              YPLT(I) = YMAX
324: C
325: C              GDIST(I) = YMAX

```

```

326: C      AAAG(I) = 0.0
327: C      BBBG(I) = -1.0
328: C      CCCG(I) = 0.0
329: C      KGCNT(I) = 0
330: C
331: C      LSCAN(I) = -2
332: C      end if
333: C=====
334: C
335: C      JSCANY = 1      JSCANY = 2      JSCANY = 3
336: C      ----->      <----->      <----->
337: C      ----->      <----->      <----->
338: C      ----->      <----->      <----->
339: C      ----->      <----->      <----->
340: C      ----->      <----->      <----->
341: C      ----->      <----->      <----->
342: C      ----->      <----->      <----->
343: C      ----->      <----->      <----->
344: C
345: C=====
346: C      else
347: C          J      = IIX
348: C          if ( JSCANY.eq.3 ) then
349: C              if ( IIX.gt.NPART/2 ) then
350: C                  J      = 2*(IIX-NPART/2) - 1
351: C              else
352: C                  J      = 2*IIX
353: C              end if
354: C          end if
355: C          IMOD      = MOD(J,2)
356: C          if ( JSCANY.eq.1 .or. JSCANY.eq.3.and.IMOD.eq.0 ) then
357: C              XXX(I) = PAPER(1) + PAPER(7)*DYCG*(J-1)
358: C              YYY(I) = PAPER(2) + PAPER(8)*DYCG*(J-1)
359: C              ZZZ(I) = PAPER(3) + PAPER(9)*DYCG*(J-1)
360: C              AAA(I) = PAPER(4)
361: C              BBB(I) = PAPER(5)
362: C              CCC(I) = PAPER(6)
363: C
364: C              XPLT(I) = 0.0
365: C              YPLT(I) = DYCG*(J-1)
366: C
367: C              GDIST(I) = XMAX
368: C              AAAG(I) = 1.0
369: C              BBBG(I) = 0.0
370: C              CCCG(I) = 0.0
371: C              KGCNT(I) = 0
372: C
373: C              LSCAN(I) = +1
374: C          else if ( JSCANY.eq.2 .or. JSCANY.eq.3.and.IMOD.eq.1 )
375: C              &      then
376: C              XXX(I) = PAPER(1) + PAPER(4)*XMAX + PAPER(7)*DYCG*
377: C              &      (J-1)
378: C              YYY(I) = PAPER(2) + PAPER(5)*XMAX + PAPER(8)*DYCG*
379: C              &      (J-1)
380: C              ZZZ(I) = PAPER(3) + PAPER(6)*XMAX + PAPER(9)*DYCG*
381: C              &      (J-1)
382: C              AAA(I) = -PAPER(4)
383: C              BBB(I) = -PAPER(5)
384: C              CCC(I) = -PAPER(6)
385: C
386: C          CCCCCCCCCCCC XPLT(I) = 0.0
387: C              XPLT(I) = XMAX
388: C              YPLT(I) = DYCG*(J-1)
389: C
390: C              GDIST(I) = XMAX

```

src/cgview/source.f

```

391:          AAAG(I) = -1.0
392:          BBBG(I) = 0.0
393:          CCCG(I) = 0.0
394:          KGCNT(I) = 0
395:
396:          LSCAN(I) = -1
397:        end if
398:      end if
399:    100 continue
400: C
401:  end if
402: C
403: C-----
404: C ... set flight path counter to zero or negative
405: C-----
406: C
407: C << CGVIEW specific >> set flight path counter to zero
408: C in single cell display mode because particles may
409: C enter lattice cell zone bypassing LATICE routine.
410: C
411:   if ( KIBNK(5).ne.0 ) then
412: C
413: C     << CGVIEW >>
414: C
415:     if ( IMCELL.eq.0 ) then
416:       do 110 I = 1, NN
417:         IBNK(I,KIBNK(5)) = -1
418:       110 continue
419:     else
420:       do 120 I = 1, NN
421:         IBNK(I,KIBNK(5)) = 0
422:       120 continue
423:     end if
424:   end if
425: C
426: C
427: C-----
428: C ... check zone to start ...
429: C-----
430: C ... KSZ=0/1 = forward/backward zone search ....
431: C
432:   do 230 KSZ = 0, 1
433: C
434:     do 130 I = 1, NN
435:       IZW2(I,KSZ) = 0
436:     130 continue
437: C
438:     NNS = 0
439:     do 140 I = 1, NN
440: CCCCCC   if ( IZSS(I).eq.KSZ ) then
441: C          NNS = NNS + 1
442: C          X(NNS) = XXX(I) + 1.0D-8*AAA(I)
443: C          Y(NNS) = YYY(I) + 1.0D-8*BBB(I)
444: C          Z(NNS) = ZZZ(I) + 1.0D-8*CCC(I)
445: C          IWK1(NNS) = I
446: CCCCCC   end if
447:     140 continue
448:
449:     if ( NNS.eq.0 ) go to 230
450:
451:     IPP = 0
452: CCCC II = NN
453:     II = NNS
454:     MMZ1 = 1
455:     MMZ = NZONE

```

```

456:     MMINC = 1
457:   if ( JLATT.ne.0 ) then
458:     MMZ = IPCEL(1) - 1
459: C
460: C ... single cell display mode
461: C
462:   if ( IMCELL.gt.0 ) then
463:     MMZ1 = IPCEL(KCELL(IMCELL))
464:     MMZ = IPCEL(KCELL(IMCELL)+1) - 1
465:   end if
466: end if
467: C
468: C
469: if ( KSZ.eq.1 ) then
470:   MMT = MMZ1
471:   MMZ1 = MMZ
472:   MMZ = MMT
473:   MMINC = -1
474: end if
475:
476: do 200 M = MMZ1, MMZ, MMINC
477:   IZ = M
478: C
479: C .... JUDGE WHETHER PARTICLES BELONG TO ZONE IZ OR NOT. ....
480: C (save distance to boundary on R(*))
481: C
482:   call JUDGE( 'SOURCE', IZ, II, X, Y, Z, IFI, SDA, KZDA, KZAA,
483: &             NSDA, NZDA, NZONE+1, JVMNT, IFL, JSIMP, .true.,
484: &             KSREF, IWK2, R )
485: C
486: C .... memorize found zone and distance from boundary
487: C for both zone-search order to check zone overlap
488: C
489:   do 150 I = 1, II
490:     if ( IFI(I) ) then
491:       IZW2(IWK1(I),KSZ) = IZ
492:       FXYZ2(IWK1(I),KSZ) = R(I)
493:     end if
494:   150 continue
495: C
496: C
497: C ..... SEND PARTICLES TO FLIGHT STACK .....
498: C
499:   INN = 0
500:
501:   if ( KZMAT(IZ).ge.0 .or. KZMAT(IZ).eq.-1000
502: &     .or. KZMAT(IZ).eq.-2000 .or. KZMAT(IZ).eq.-3000
503: &     .or. KZMAT(IZ).eq.-4000
504: &     .or. (LEVEL.eq.0.and.ISLATT(KZMAT(IZ)))
505: &     .or. (IMCELL.ne.0.and.KZMAT(IZ).eq.-999) ) then
506:
507: CCCC & (LEVEL.eq.0.and.KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998)
508:
509:     IFFL = NFFL(NZONE+1)
510: *VOCL LOOP,NOVREC
511:     do 160 I = 1, II
512:       if ( IFI(I).and.IZSS(IWK1(I)).eq.KSZ ) then
513:         IZZ(IWK1(I)) = IZ
514:         INN = INN + 1
515:         LSFFL(IFFL+INN) = IWK1(I)
516:         IZFFL(IFFL+INN) = IZ
517:         IREG = KZREG(IZ)
518: C
519:         ... always set IBREG (from Jan 2000)
520:         IBREG(IWK1(I)) = IREG
521:         if ( JTLLT.ne.0 ) then

```

src/cgview/source.f

```

521:          IBSPC(IWK1(I),0) = 0
522:        end if
523:      end if
524: 160      continue
525:
526:          NFFL(IZ) = NFFL(IZ) + INN
527:          NFFL(NZONE+1) = NFFL(NZONE+1) + INN
528: C
529: C ..... IF PARTICLES ARE IN A LATTICE, SEND THEM TO LATTICE STACK.
530:
531: CCCC      else if ( KZMAT(IZ).le.-1.and.KZMAT(IZ).ge.-998 ) then
532: C          else if ( ISLATT(KZMAT(IZ)) ) then
533:
534: C              NXLT(NLBZ+1) = NXLT(NLBZ+1)
535:
536: C              ILAT = NXLT(NLBZ+1)
537: *VOCL LOOP,NOVREC
538:          do 170 I = 1, II
539:            if ( IFI(I).and.IZSS(IWK1(I)).eq.KSZ ) then
540:              IZZ(IWK1(I)) = IZ
541:              INN = INN + 1
542:              LSLAT(ILAT+INN) = IWK1(I)
543:              IZLAT(ILAT+INN) = MLBZZ(IZ)
544:              IREG = KZREG(IZ)
545: C              ... always set IBREG (from Jan 2000)
546:              IBREG(IWK1(I)) = IREG
547:              if ( JTLAT.ne.0 ) then
548:                IBSPC(IWK1(I),0) = 0
549:              end if
550:            end if
551: 170      continue
552:          NXLT(MLBZZ(IZ)) = NXLT(MLBZZ(IZ)) + INN
553:          NXLT(NLBZ+1) = ILAT + INN
554: C
555: C ..... GENERATING PARTICLES IN INVALID MATERIAL ? .....
556: C
557:      else
558: *VOCL LOOP,NOVREC
559:          do 180 I = 1, II
560:            if ( IFI(I).and.IZSS(IWK1(I)).eq.KSZ ) then
561:              INN = INN + 1
562:              LSDDED(IWK1(I)) = 1
563:            end if
564: 180      continue
565:          IPP = IPP + INN
566:          if ( INN.gt.0 ) write(IOW,7000) KZMAT(IZ), IZ, INN
567: 7000      format(/1X,' !!! CAUTION: YOU ARE GOING TO',
568: &              ' GENERATE PARTICLES IN A ZONE WHICH ',
569: &              ' PARTICLES SHOULD NOT BE IN !!! '/1X,' MAT = ',
570: &              I5,' ZONE = ',I5,' ',I8,
571: &              ' PARTICLES. THEY ARE TREATED AS LOST!!!')
572:          end if
573: C
574: C ..... COMPRESS .....
575:
576:          if ( INN.lt.II ) then
577:            INNN = 0
578: *VOCL LOOP,NOVREC
579:          do 190 I = 1, II
580:            if ( .not.IFI(I) ) then
581:              INNN = INNN + 1
582:              IWK1(INNN) = IWK1(I)
583:              X(INNN) = X(I)
584:              Y(INNN) = Y(I)
585:              Z(INNN) = Z(I)

```

```

586:          end if
587: 190      continue
588:          II = INNN
589:        else
590:          II = 0
591:          go to 210
592:        end if
593: 200      continue
594: 210      continue
595: C
596: C ..... ZONE NOT FOUND !! (LOST) .....
597: C
598:          if ( II.ne.0 ) then
599:
600:            write(IOW,7020) II, (XXX(IWK1(I)),YYY(IWK1(I)),ZZZ(IWK1(I))),
601: &              AAA(IWK1(I)),BBB(IWK1(I)),CCC(IWK1(I)),I=1,
602: &              MIN(II,MSGSLST))
603: 7020      format(/1X,' **** ',I5,
604: &              ' PARTICLES ARE LOST IN SOURCE ROUTINE!'/1X,
605: &              ' X = ',1P,E12.5,' Y = ',E12.5,' Z = ',E12.5,
606: &              ' MU = ',E12.5,' ETA = ',1P,E12.5,' XI = ',E12.5))
607:
608:          if ( II.gt.MSGSLST ) then
609:            write(IOW,*) ' >>> more ', II - MSGSLST,
610: &              ' particles are lost but message suppressed.'
611:          end if
612:
613:          do 220 I = 1, II
614:            LSDDED(IWK1(I)) = 1
615: 220      continue
616:          end if
617: C
618:          NDEAD = NDEAD + II + IPP
619: C
620: 230 continue
621: C
622: C-----
623: C check zone overlap using saved zone# and distance to boundary
624: C for forward/backward order of zone search.
625: C-----
626: C
627:          DDD2 = DEPS*2.0D0
628:          if ( JGMCK.ne.0 ) then
629:            if ( DOVERLAP.gt.0 ) then
630:              DDD2 = DEPS*DOVERLAP
631:            else if ( DOVERLAP.lt.0 ) then
632:              DDD2 = -DOVERLAP
633:            end if
634:          end if
635: C
636: CCC      MPLIMIT = 10
637:          MM = 0
638: C
639:          do 260 I = 1, NN
640:            if ( IZW2(I,0).ne.0.and.IZW2(I,1).ne.0 ) then
641:              IZ0 = IZW2(I,0)
642:              IZ1 = IZW2(I,1)
643:              KZ0 = KINPZ(IZ0)
644:              KZ1 = KINPZ(IZ1)
645:              if ( KZ0.ne.KZ1
646: &              .and.(FXYZ2(I,0).gt.DDD2.or.FXYZ2(I,1).gt.DDD2) ) then
647: C
648:              IGMCK(IZ0) = IGMCK(IZ0) + 1
649:              IGMCK(IZ1) = IGMCK(IZ1) + 1
650: C

```


src/cgview/source.f

```

651: C      ... judgement of REGION identity :
652: C      In TALLY-LATTICE mode and CELL-wise specified,
653: C      KZREG(IZ*) does not give the REGION # itself,
654: C      so name of input zone (KREGN) is used for
655: C      region overlap judgement.
656: C
657: C      JSAMEREG = 1
658: C      if ( JTLT.ne.0 ) then
659: C          if( KREGN(KZ0).ne.KREGN(KZ1) ) JSAMEREG = 0
660: C      else
661: C          if( KZREG(IZ0).ne.KZREG(IZ1) ) JSAMEREG = 0
662: C      end if
663: C
664: C      if ( IGMCK(IZ0).lt.MPLIMIT .or. MPLIMIT.lt.0 ) then
665: C
666: C          if ( KZMAT(IZ0).ne.KZMAT(IZ1).and.JSAMEREG.eq.0 ) then
667: C              if ( JTLT.eq.0 ) then
668: C                  write(IOW,7040) IZ0, KZ0,
669: C                      IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))),
670: C                      KZMAT(IZ0), KZREG(IZ0), IZ1, KZ1,
671: C                      IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
672: C                      KZMAT(IZ1), KZREG(IZ1)
673: C              else
674: C                  write(IOW,7042) IZ0, KZ0,
675: C                      IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))),
676: C                      KZMAT(IZ0), KREGN(KZ0), IZ1, KZ1,
677: C                      IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
678: C                      KZMAT(IZ1), KREGN(KZ1)
679: C              end if
680: C              KOV = 3
681: C          else if ( KZMAT(IZ0).ne.KZMAT(IZ1) ) then
682: C              write(IOW,7060) IZ0, KZ0,
683: C                  IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))),
684: C                  KZMAT(IZ0), IZ1, KZ1,
685: C                  IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
686: C                  KZMAT(IZ1)
687: C              KOV = 2
688: C          else if ( JSAMEREG.eq.0 ) then
689: C              if ( JTLT.eq.0 ) then
690: C                  write(IOW,7080) IZ0, KZ0,
691: C                      IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))),
692: C                      KZREG(IZ0), IZ1, KZ1,
693: C                      IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
694: C                      KZREG(IZ1)
695: C              else
696: C                  write(IOW,7082) IZ0, KZ0,
697: C                      IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))),
698: C                      KREGN(KZ0), IZ1, KZ1,
699: C                      IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
700: C                      KREGN(KZ1)
701: C              end if
702: C              KOV = 1
703: C          else
704: C              write(IOW,7100) IZ0, KZ0,
705: C                  IZNAM(KZ0) (:ICLEN2(IZNAM(KZ0))), IZ1, KZ1,
706: C                  IZNAM(KZ1) (:ICLEN2(IZNAM(KZ1))),
707: C                  KZMAT(IZ0), KZREG(IZ0)
708: C              KOV = 0
709: C          end if
710: C          write(IOW,7120) IZ0, IZ1, XXX(I), YYY(I), ZZZ(I),
711: C              AAA(I), BBB(I), CCC(I),
712: C              MIN(FXYZ2(I,0),FXYZ2(I,1))
713: C      else
714: C          if ( IGMCK(IZ0).eq.MPLIMIT ) then
715: C              write(IOW,*) 'XXX(SOURCE) more than ', MPLIMIT,

```

```

716: C      &      ' point of overlapping material and/or region',
717: C      &      ' for zone(inp-zone) ', IZ0, '(', IZ1, 'KZ0, '))'
718: C      end if
719: C      if ( KZMAT(IZ0).ne.KZMAT(IZ1).and.JSAMEREG.eq.0 ) then
720: C          KOV = 3
721: C      else if ( KZMAT(IZ0).ne.KZMAT(IZ1) ) then
722: C          KOV = 2
723: C      else if ( JSAMEREG.eq.0 ) then
724: C          KOV = 1
725: C      else
726: C          KOV = 0
727: C      end if
728: C      end if
729: C
730: C
731: C      ... memorize overlapping pair
732: C
733: C      MM = 0
734: C      IZZ1 = MIN(IZ0,IZ1)
735: C      IZZ2 = MAX(IZ0,IZ1)
736: C      do 240 M = 1, MIN(NIPGMCK,MIPGMCK)
737: C          if ( IPGMCK(1,M).eq.IZZ1.and.IPGMCK(2,M).eq.IZZ2 )
738: C              &      then
739: C                  go to 250
740: C              end if
741: C          continue
742: C          NIPGMCK = NIPGMCK + 1
743: C          if ( NIPGMCK.le.MIPGMCK ) then
744: C              M = NIPGMCK
745: C              IPGMCK(1,NIPGMCK) = IZZ1
746: C              IPGMCK(2,NIPGMCK) = IZZ2
747: C          else
748: C              M = 0
749: C          end if
750: C          continue
751: C          MM = M
752: C
753: C      if ( MM.ne.0 ) then
754: C          IPGMCK(3,MM) = KOV
755: C          DM = MIN(FXYZ2(I,0),FXYZ2(I,1))
756: C          if ( ABS(DM).gt.XGMCK(4,MM) ) then
757: C              XGMCK(4,MM) = DM
758: C              XGMCK(1,MM) = XXX(I)
759: C              XGMCK(2,MM) = YYY(I)
760: C              XGMCK(3,MM) = ZZZ(I)
761: C          end if
762: C          end if
763: C          end if
764: C          end if
765: C      260 continue
766: C
767: C      7040 format(1X,'xxx Overlapping material & region at start point?:/3X,
768: C          &      ' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6,' region',I5/
769: C          &      3X,' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6,' region',I5
770: C          &      )
771: C      7042 format(1X,'xxx Overlapping material & region at start point?:/3X,
772: C          &      ' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6,
773: C          &      ' region <'A,'>' /
774: C          &      3X,' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6,
775: C          &      ' region <'A,'>' /
776: C          &      )
777: C      7060 format(1X,'xxx Overlapping material found at start point :/3X,
778: C          &      ' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6/3X,
779: C          &      ' zone(izone)',I5,'(',I5,'')<'A,'> mat',I6)
780: C      7080 format(1X,'xxx Overlapping region found at start point:/3X,

```

src/cgview/source.f

```
781:      &      ' zone(izone)',I5,'(',I5,')<',A,'> region',I5/3X,
782:      &      ' zone(izone)',I5,'(',I5,')<',A,'> region',I5)
783: 7082 format(1X,'xxx Overlapping region found at start point:'/3X,
784:      &      ' zone(izone)',I5,'(',I5,')<',A,'> region <',A,'>/3X,
785:      &      ' zone(izone)',I5,'(',I5,')<',A,'> region <',A,'>')
786: 7100 format(1X,'!!! Overlapping input-zone found at start point:'/3X,
787:      &      ' zone(izone)',I5,'(',I5,')<',A,'>', ' zone(izone)',I5,'(',
788:      &      I5,')<',A,'>/3X,'mat',I6,' region',I5)
789: 7120 format(3X,' ZONE',2I5,' XYZ=',1P,3E13.5/3X,' DIR=',3E12.4,
790:      &      ' DIST=',E11.4)
791: C
792: C -----
793: C
794:      NTGEN = NTGEN + NGENE
795: C      NTGEN = NTGEN + NGENE - 1
796: CCC write(IOW,*) ' GENERATED PARTICLES = ', NTGEN + 1
797: C
798: C
799:      if ( STYLE.eq.0.and.STYLE2.eq.0 ) then
800:          call CAPLINE( 2 )
801:      else
802:          call CAPLINE( 1 )
803:      end if
804: C
805:      return
806:      end
```

src/cgview/texdraw.c

```

1:  /*****
2:  /*
3:  /* texdraw.c : Drawing Routines for EPS file Output
4:  /*
5:  /*****
6:
7:  /*****
8:  update:
9:    04 Mar 2004: change exit() to exit(0) in psplot. (Y.Nagaya)
10:   21 Sep 1998: add include <string.h>
11:  *****/
12: #include <X11/Xlib.h>
13: #include <X11/Xutil.h>
14: #include <stdio.h>
15: #include <string.h>
16: #include "piflib.h"
17:
18: #define XORIG 20.0
19: #define YORIG 17.0
20:
21: texplots ( ps_file )
22:   char * ps_file ;
23: {
24:   /*** static char *outfile = { "piflib" }; ***/
25:   static char outfile[256];
26:   static char *fileID = { ".EPS" };
27:   static char *ps[] = { "%!PS-Adobe-2.0 EPSF-2.0",
28:     "%Creator: piflib",
29:     "%DocumentFonts: Courier",
30:     "%BoundingBox: 50 50 410 310", **/
31:     "%BoundingBox: 20 50 550 800",
32:     "%EndComments",
33:     "/pifdict 30 dict def",
34:     "pifdict begin",
35:     "/Color true def",
36:     "/DC {Color {setrgbcolor} {pop pop pop 0 setgray} if
else} def",
37:     "/LC0 { stroke 0 0 0 DC } def",
38:     "/LC1 { stroke 1 1 1 DC } def",
39:     "/LC2 { stroke 1 0 0 DC } def",
40:     "/LC3 { stroke 0 1 0 DC } def",
41:     "/LC4 { stroke 0 0 1 DC } def",
42:     "/LC5 { stroke 0 1 1 DC } def",
43:     "/LC6 { stroke 1 0 1 DC } def",
44:     "/LC7 { stroke 1 1 0 DC } def",
45:     "/LT1 { stroke [5 5] 0 setdash } def",
46:     "/LT2 { stroke [10 5] 0 setdash } def",
47:     "/LT3 { stroke [10 5 5 5] 0 setdash } def",
48:     "/LT4 { stroke [10 5 5 5 5] 0 setdash } def",
49:     "/LT5 { stroke [10 5 5 5 5 5 5] 0 setdash } def",
50:     "/LT6 { stroke [10 5 2 5] 0 setdash } def",
51:     "/LT7 { stroke [10 5 2 2 5] 0 setdash } def",
52:     "/LT8 { stroke [10 5 2 2 2 2 5] 0 setdash } def",
53:     "/LT9 { stroke [10 5 8 5 2 5 8 5] 0 setdash } def",
54:     "/LT10 { stroke [2 2] 0 setdash } def",
55:     "/LT11 { stroke [2 2 2 2 2 10] 0 setdash } def",
56:     "/M {moveto} def",
57:     "/L {lineto} def",
58:     "/N {moveto lineto} def",
59:     "end",
60:     "%EndInitialize",
61:     NULL
62:   };
63:
64:   int k;
65:   int i,j;
66:   char c;
67:
68:   /* x_pix_mm = 2.834645676; */
69:   /* y_pix_mm = 2.834645676; */
70:   x_pix_mm = 11.3385827;
71:   y_pix_mm = 11.3385827;
72:
73:   ps_xorg = XORIG;
74:   ps_yorg = YORIG;
75:
76:   ps_plot = 0;
77:   f_value = 1.0;
78:
79:   /*****
80:   if(!ifxpif) {
81:     for( i = 0 ; i < 81 ; i++) {
82:       filename[i] = NULL;
83:     }
84:
85:     printf("Input filename to output( for PostScript )\n");
86:     printf("(Default : piflib??.EPS) >>:");
87:     i = 0;
88:     while((c = getchar()) != '\n') {
89:       filename[i] = c;
90:       i++;
91:     }
92:     for ( j = 0 ; j < i ; j++){
93:       if ( filename[j] == '.' ) {
94:         dpoint = j;
95:       }
96:     }
97:     strcpy(fileext, filename+dpoint);
98:     ifxpif = 1;
99:   }
100:   *****/
101:
102:   filename[0] = '\0';
103:   if( ps_file != NULL && strlen(ps_file) > 0 ) strcpy( filename, ps_file );
104:
105:   if(!ifxpif) {
106:     ifxpif = 1;
107:     i = strlen( filename );
108:     dpoint = 0 ;
109:     for ( j = 0 ; j < i ; j++){
110:       if ( filename[j] == '.' ) {
111:         dpoint = j;
112:       }
113:     }
114:     if( i > 0 ) strcpy(fileext, filename+dpoint);
115:   }
116:
117:   if(filename[0] == '\0'){
118:     sprintf(outfile,"piflib%d",EPS_page);
119:     strcat(outfile, fileID);
120:     if ( (fp = fopen(outfile, "w")) == NULL ) {
121:       printf("***(PIFLIB) Can not Open File : %s !!\n",outfile);
122:       exit(0);
123:     }
124:   }else{
125:     sprintf(filename+dpoint,"%d",EPS_page);
126:     strcat(filename, fileext);
127:     if ( (fp = fopen(filename, "w")) == NULL ) {
128:       printf("***(PIFLIB) Can not Open Output File: %s !!\n,filename");
129:       exit(0);

```

src/cgview/texdraw.c

```

130:     }
131: }
132:
133: /**** output EPS header on file ****/
134:
135: for ( k = 0 ; ps[k] != NULL ; k++ ) {
136:     fprintf( fp, "%s\n", ps[k] );
137: }
138: }
139:
140: texplot( x_point1, y_point1, iflag )
141: float x_point1, y_point1;
142: int iflag;
143: {
144:     float xxx, yyy;
145:     /**** float ps_x, ps_y ****/
146:     /**** M.Sasaki 951114 : make ps_x, ps_y to static variable
147:         to avoid setting garbage value on ps_store_[x|y]
148:     ****/
149:     static float ps_x, ps_y;
150:
151:     xxx = x_point1;
152:     yyy = y_point1;
153:
154:     x_store = xxx;
155:     y_store = yyy;
156:
157:     xxx *= f_value;
158:     yyy *= f_value;
159:
160:     if( iflag == 999 ) {
161:         if(ps_plot) {
162:             fprintf(fp, "gsave\n");
163:             fprintf(fp, "stroke\n");
164:             fprintf(fp, "grestore\n");
165:             fprintf(fp, "end\n");
166:             fprintf(fp, "showpage\n");
167:         }
168:         fclose(fp);
169:         exit(0);
170:     }else if( iflag == 888 || iflag == 777 ) {
171:         ;
172:     }else if( iflag == 666 ) {
173:         fprintf(fp, "gsave\n");
174:         fprintf(fp, "stroke\n");
175:         fprintf(fp, "grestore\n");
176:         fprintf(fp, "end\n");
177:         fprintf(fp, "showpage\n");
178:
179:         fclose(fp);
180:         EPS_page++;
181:         texplots( filename );
182:
183:         ifplot3 = 0;
184:         ps_plot = 0;
185:         ps_xorg = XORIG;
186:         ps_yorg = YORIG;
187:     }else if( iflag == 2 ) {
188:         ps_x = (ps_xorg + xxx)*x_pix_mm;
189:         ps_y = (ps_yorg + yyy)*y_pix_mm;
190:
191:         if(!ps_plot) {
192:             fprintf(fp, "pifdict begin\n");
193:             fprintf(fp, "gsave\n");
194:
195:             fprintf(fp, "grestore\n");
196:             fprintf(fp, "50 50 translate\n");
197:             fprintf(fp, ".1062 .1062 scale\n");
198:             if(!ifplot3) {
199:                 fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
200:                     (long)(ps_yorg*y_pix_mm) );
201:             }
202:         }
203:         fprintf(fp, "%d %d L\n", (long)ps_x, (long)ps_y);
204:         ps_plot = 1;
205:     }else if( iflag == 3 ) {
206:         ps_x = (ps_xorg + xxx)*x_pix_mm;
207:         ps_y = (ps_yorg + yyy)*y_pix_mm;
208:
209:         if(!ps_plot) {
210:             fprintf(fp, "pifdict begin\n");
211:             fprintf(fp, "gsave\n");
212:             fprintf(fp, "newpath\n");
213:             fprintf(fp, "grestore\n");
214:             fprintf(fp, "50 50 translate\n");
215:             fprintf(fp, ".1062 .1062 scale\n");
216:         }
217:         fprintf(fp, "%d %d M\n", (long)ps_x, (long)ps_y);
218:         ps_plot = 1;
219:         ifplot3 = 1;
220:     }else if( iflag == -2 ) {
221:         ps_xorg += xxx;
222:         ps_yorg += yyy;
223:
224:         ps_x = ps_xorg*x_pix_mm;
225:         ps_y = ps_yorg*y_pix_mm;
226:
227:         if(!ps_plot) {
228:             fprintf(fp, "pifdict begin\n");
229:             fprintf(fp, "gsave\n");
230:             fprintf(fp, "newpath\n");
231:             fprintf(fp, "grestore\n");
232:             fprintf(fp, "50 50 translate\n");
233:             fprintf(fp, ".1062 .1062 scale\n");
234:             if(!ifplot3) {
235:                 fprintf(fp, "%d %d M\n", (long)(ps_xorg*x_pix_mm),
236:                     (long)(ps_yorg*y_pix_mm) );
237:             }
238:         }
239:         fprintf(fp, "%d %d L\n", (long)ps_x, (long)ps_y);
240:         ps_plot = 1;
241:     }else if( iflag == -3 ) {
242:         ps_xorg += xxx;
243:         ps_yorg += yyy;
244:
245:         ps_x = ps_xorg*x_pix_mm;
246:         ps_y = ps_yorg*y_pix_mm;
247:
248:         if(!ps_plot) {
249:             fprintf(fp, "pifdict begin\n");
250:             fprintf(fp, "gsave\n");
251:             fprintf(fp, "newpath\n");
252:             fprintf(fp, "grestore\n");
253:             fprintf(fp, "50 50 translate\n");
254:             fprintf(fp, ".1062 .1062 scale\n");
255:         }
256:
257:         fprintf(fp, "%d %d M\n", (long)ps_x, (long)ps_y);
258:         ps_plot = 1;
259:         ifplot3 = 1;

```

src/cgview/texdraw.c

```
260:     }
261:
262:     ps_store_x = ps_x;
263:     ps_store_y = ps_y;
264: }
265:
266: texdot( x1, y1 )
267:     float *x1, *y1;
268: {
269:     float x_dot, y_dot;
270:     float xxx, yyy;
271:
272:     xxx = *x1;
273:     yyy = *y1;
274:
275:     x_dot = (ps_xorg + xxx)*x_pix_mm;
276:     y_dot = (ps_yorg + yyy)*y_pix_mm;
277:
278:     ps_plot = 1;
279: }
280:
281: texplotlines( nml, xpoints, ypoints )
282:     int nml;
283:     float *xpoints, *ypoints;
284: {
285:     int i;
286:     psplot( xpoints[0], ypoints[0], 3 ) ;
287:     for ( i=1 ; i < nml ; i++ ) {
288:         texplot( xpoints[i], ypoints[i], 2 ) ;
289:     }
290: }
291:
292:
```

src/cgview/texsymchar.c

```
1: /*****  
2:  */  
3: /* texsymchar.c : Symbol Routines for EPS file Output */  
4:  */  
5: /*****/  
6: #include <X11/Xlib.h>  
7: #include <X11/Xutil.h>  
8: #include <stdio.h>  
9: #include <math.h>  
10: #include "piflib.h"  
11:  
12: texsymbol( x_str, y_str, height, string, angle, no_of_letters)  
13:     float *x_str, *y_str;  
14:     float *height, *angle;  
15:     int *no_of_letters;  
16:     /** long *no_of_letters; **/  
17:     char *string;  
18: {  
19:     float xxx, yyy;  
20:  
21:     xxx = *x_str;  
22:     yyy = *y_str;  
23:  
24:     xxx *= x_pix_mm;  
25:     yyy *= y_pix_mm;  
26:  
27:     xxx = ps_xorg + xxx;  
28:     yyy = ps_yorg - yyy;  
29:  
30:     /*****/  
31:  
32:     ps_plot = 1;  
33:  
34: }  
35:  
36: texsymbol(x, y, hh, ibcda, angle, nchar )  
37:  
38:     float *x, *y, *hh, *angle;  
39:     int *nchar;  
40:     int *ibcda;  
41:  
42: {  
43:     static float oldth = 9999.0;  
44:     static float oldfct = 9999.0;  
45:     static float rad = 0.017453;  
46:     static float x0 = 0.0;  
47:     static float y0 = 0.0;  
48:  
49:     static float DDV = 0.3;  
50:  
51:     static float eps = 0.01;  
52:     static long kp8 = 256;  
53:     static long kp16 = 65536;  
54:     static int nxy = 0;  
55:     static int ichar = 0;  
56:  
57:     static float xa[9], ya[9];  
58:     static float fct, th, th1;  
59:     static float sinth, costh;  
60:  
61:     unsigned long ibit8, ibit16, ibit24;  
62:     unsigned int kpos, knum, ktab, kpt;  
63:     unsigned int nnt;  
64:     unsigned long idata;  
65:     unsigned char ibcd[255];  
66:  
67:     long iebc;  
68:     int ipen, icb, nt, iswv;  
69:     int nx, ny;  
70:  
71:     float xt, yt;  
72:     float div, h;  
73:     float xx, yy;  
74:  
75:     int i, m, kk;  
76:  
77:     static long idxtab[128] = { 0x00080001, 0x030C0002, 0x00060016, 0x02070011  
78:                                0x00070013, 0x02070005, 0x00070007, 0x02080008  
79:                                0x010B000A, 0x0307000C, 0x010E000E, 0x020D0011  
80:                                0x02060014, 0x02040011, 0x000C0016, 0x00050079  
81:                                0x0302004F, 0x030E002E, 0x02030056, 0x0108007A  
82:                                0x0105006F, 0x0211007C, 0x00080077, 0x03080080  
83:                                0x0102007A, 0x01030082, 0x000200C1, 0x00080083  
84:                                0x00060085, 0x03030068, 0x00060088, 0x010D0086  
85:                                0x02090089, 0x0309008B, 0x0009008E, 0x01070090  
86:                                0x000C0092, 0x000C0095, 0x020A009B, 0x03070097  
87:                                0x01090099, 0x0006009E, 0x020E007C, 0x020C009F  
88:                                0x020900A2, 0x030800A4, 0x00020074, 0x011200BB  
89:                                0x03050072, 0x030E00A6, 0x01060029, 0x01060067  
90:                                0x02040056, 0x02040043, 0x02040052, 0x00020063  
91:                                0x0007002D, 0x02050071, 0x03090044, 0x030D0044  
92:                                0x01050070, 0x010500AA, 0x0005006E, 0x0005006D  
93:                                0x00000000, 0x01090032, 0x000C0035, 0x0008004D  
94:                                0x02070034, 0x00070038, 0x00060038, 0x030C0039  
95:                                0x01060048, 0x0306003C, 0x020B00AB, 0x00050055  
96:                                0x0003002A, 0x01040066, 0x0305004F, 0x00020060  
97:                                0x020A0074, 0x0106005D, 0x0006004B, 0x00030048  
98:                                0x0105004A, 0x03040049, 0x020A004C, 0x00070060  
99:                                0x030C004C, 0x00090060, 0x03070040, 0x020B0063  
100:                               0x030B004F, 0x02040042, 0x020C0053, 0x0303007B  
101:                               0x02020050, 0x01020062, 0x020D0057, 0x0304005A  
102:                               0x0007005D, 0x0303005E, 0x0105005F, 0x01050062  
103:                               0x02050063, 0x0107005B, 0x010E00AE, 0x00060055
```

src/cgview/texsymchar.c

```

104:          0x020E0069, 0x030200B1, 0x00030068, 0x010E003E    164:          0x27, 0x20, 0x11, 0x31, 0x20, 0x27, 0x16, 0x36,
105:          0x0309004C, 0x00050019, 0x0109001A, 0x010D0022    165:          0x27, 0x03, 0x43, 0x34, 0x32, 0x43, 0x03, 0x14,
106:          0x0204001C, 0x020A001D, 0x000C001E, 0x00050021    166:          0x12, 0x03, 0x01, 0x12, 0x20, 0x27, 0x47, 0x50,
107:          0x01110022, 0x020B0026, 0x020B0053, 0x010B00B2    167:          0x00, 0x24, 0x07, 0x57, 0x27, 0x25, 0x40, 0x15,
108:          0x001000B5, 0x0004002C, 0x0305002A, 0x000900B9    168:          0x16, 0x27, 0x36, 0x03, 0x02, 0x11, 0x21, 0x42,
};          169:          0x04, 0x44, 0x70, 0x41, 0x01, 0x70, 0x10, 0x35,
109:          170:          0x13, 0x23, 0x24, 0x14, 0x13, 0x00, 0x40, 0x70,
110:          171:          0x46, 0x06, 0x70, 0x03, 0x43, 0x42, 0x52, 0x42,
111:          172:          0x33, 0x22, 0x12, 0x03, 0x04, 0x15, 0x25, 0x34,
112:          173:          0x33, 0x34, 0x45, 0x55, 0x64, 0x63, 0x52, 0x01,
113:          174:          0x41, 0x70, 0x22, 0x26, 0x24, 0x44, 0x04, 0x05,
114:          175:          0x01, 0x10, 0x20, 0x31, 0x36, 0x47, 0x57, 0x66,
115:          176:          0x02, 0x32, 0x43, 0x44, 0x35, 0x05, 0x16, 0x26,
116:          177:          0x44, 0x54, 0x65, 0x70, 0x63, 0x52, 0x42, 0x24,
117:          178:          0x14, 0x03, 0x00, 0x10, 0x21, 0x22, 0x33, 0x24,
118:          179:          0x25, 0x16, 0x06, 0x46, 0x36, 0x25, 0x24, 0x13,
119:          180:          0x22, 0x21, 0x30, 0x40, 0x00, 0x15, 0x12, 0x21,
120:          181:          0x31, 0x42, 0x45, 0x42, 0x51, 0x11, 0x15, 0x05,
121:          182:          0x45, 0x35, 0x31, 0x42, 0x21, 0x11, 0x02, 0x04,
122:          183:          0x15, 0x35, 0x44, 0x42, 0x31, 0x21, 0x20, 0x26,
123:          184:          0x01, 0x05, 0x16, 0x26, 0x35, 0x33, 0x03, 0x33,
124:          185:          0x31, 0x20, 0x10, 0x01, 0x45, 0x70, 0x41, 0x31,
125:          186:          0x15, 0x05, 0x02, 0x11, 0x22, 0x25, 0x22, 0x31,
126:          187:          0x42, 0x45, 0x04, 0x14, 0x12, 0x21, 0x32, 0x34,
127:          188:          0x44, 0x70, 0x26, 0x20, 0x00, 0x23, 0x41, 0x23,
128:          189:          0x15, 0x06, 0x24, 0x13, 0x12, 0x21, 0x31, 0x42,
129:          190:          0x43, 0x34, 0x24, 0x15, 0x26, 0x36, 0x41, 0x21,
130:          191:          0x12, 0x14, 0x25, 0x45, 0x70, 0x43, 0x03, 0x04,
131:          192:          0x15, 0x24, 0x12, 0x24, 0x35, 0x44, 0x30, 0x21,
132:          193:          0x31, 0x32, 0x22, 0x21, 0x70, 0x03, 0x53, 0x70,
133:          194:          0x24, 0x34, 0x35, 0x25, 0x24, 0x01, 0x45, 0x70,
134:          195:          0x05, 0x41, 0x42, 0x31, 0x11, 0x02, 0x04, 0x15,
135:          196:          0x35, 0x44, 0x70, 0x26, 0x20, 0x02, 0x11, 0x31,
136:          197:          0x42, 0x43, 0x34, 0x14, 0x05, 0x16, 0x36, 0x45,
137:          198:          0x70, 0x27, 0x20, 0x03, 0x63, 0x11, 0x15, 0x14,
138:          199:          0x04, 0x44, 0x34, 0x35, 0x31, 0x32, 0x42, 0x02,
139:          200:          0x42, 0x31, 0x11, 0x02, 0x05, 0x16, 0x36, 0x45,
140:          201:          0x43, 0x32, 0x22, 0x13, 0x14, 0x25, 0x35, 0x44,
141:          202:          0x37, 0x35, 0x27, 0x37, 0x70, 0x17, 0x15, 0x07,
142:          203:          0x17, 0x00, 0x10, 0x36, 0x46, 0x70, 0x44, 0x35,
143:          204:          0x15, 0x04, 0x13, 0x33, 0x42, 0x31, 0x11, 0x02,
144:          205:          0x70, 0x12, 0x34, 0x00, 0x00, 0x00, 0x00, 0x00 };
145:
146:          206:
147:          207:
148:          208:
149:          209:
150:          210:
151:          211:
152:          212:
153:          213:
154:          214:
155:          215:
156:          216:
157:          217:
158:          218:
159:          219:
160:          220:
161:          221:
162:          222:
163:          223:
164:          224:
165:          225:
166:          226:
167:          227:

```

```

for ( i = 0 ; i < 256 ; i++ ){
    ibcd[i] = 0;
}

ibit8 = 256;
ibit16 = 65536;
ibit24 = 16777216;

h = *hh;
h = (float)sqrt( pow( (double)h, 2.0 ) );
ichar = 0;
ipen = 3;
nt = *nchar;
icb = 0;
div = 7.0;

if( nt == 0 ) {
    icb = 3;
}
if( nt < 0 ) {
    if( nt != -1 ) {

```

src/cgview/texsymchar.c

```

228:         ipen = 2;
229:     }
230:     icb = 3;
231:     ichar = *ibcda + 1;
232:
233:     if( ichar <= 15 ) {
234:         div = 4.0;
235:     }
236: }
237:
238: fct = h / div;
239: th = *angle;
240:
241: if( th != oldth ) {
242:     oldth = th;
243:     th1 = th*rad;
244:     sinth = sin( th1 );
245:     costh = cos( th1 );
246:     oldfct = fct;
247:     xa[0] = 0.0;
248:     ya[0] = 0.0;
249:     xa[1] = fct*costh;
250:     ya[1] = fct*sinth;
251:     for ( i = 2 ; i < 8 ; i++ ) {
252:         xa[i] = xa[i-1] + xa[1];
253:         ya[i] = ya[i-1] + ya[1];
254:     }
255: }
256: if( fct != oldfct ) {
257:     oldfct = fct;
258:     xa[0] = 0.0;
259:     ya[0] = 0.0;
260:     xa[1] = fct*costh;
261:     ya[1] = fct*sinth;
262:     for ( i = 2 ; i < 8 ; i++ ) {
263:         xa[i] = xa[i-1] + xa[1];
264:         ya[i] = ya[i-1] + ya[1];
265:     }
266: }
267:
268: xx = *x;
269: yy = *y;
270: if( (float)sqrt( pow( (double)( xx - 999.0 ), 2.0 ) ) > eps ) x0 = xx;
271: if( (float)sqrt( pow( (double)( yy - 999.0 ), 2.0 ) ) > eps ) y0 = yy;
272: xx = x0;
273: yy = y0;
274:
275: /* VERTICAL DIRECTION CHECK. */
276: if( nt >= 0 ) {
277:     if( *hh < 0 ) {
278:         x0 = x0 + h*sinth;
279:         y0 = y0 - h*costh;
280:         xa[8] = ( 1.0 + DDV ) * h*sinth;
281:         ya[8] = -( 1.0 + DDV ) * h*costh;
282:         iswv = 1;
283:     }
284: }
285:
286: if( ( nt < 0 ) && ( ichar <= 15 ) ) {
287:     x0 = x0 - ( xa[2] - ya[2] );
288:     y0 = y0 - ( xa[2] + ya[2] );
289: }
290:
291: nnt = abs( nt );
292:

```

```

293: if ( icb != 3 ) {
294:     strcpy(ibcd, (char *)ibcda);
295:     ibcd[*nchar] = '\0' ;
296: }
297:
298: for ( kk = 0 ; kk < nnt ; kk++ ) {
299:
300:     if ( icb == 3 ) {
301:         ichar = *ibcda;
302:     }
303:     else {
304:         astoeb( &ibcd[kk], &iebc );
305:         ichar = iebc%128;
306:     }
307:     idata = idxtab[ichar];
308:
309:     if( idata != 0 ) {
310:         kpos = idata / ibit24;
311:         knum = ( idata / ibit16 ) % ibit8;
312:         ktab = idata % ibit16;
313:         kpt = ( ktab - 1 ) * 4 + kpos;
314:         for ( m = kpt ; m < kpt+knun ; m++ ) {
315:             nx = itz[m] / 16;
316:             ny = itz[m] % 16;
317:
318:             if( nx == 7 ) {
319:                 ipen = 3;
320:             }
321:             if( nx != 7 ) {
322:                 xt = x0 + xa[nx] - ya[ny];
323:                 yt = y0 + ya[nx] + xa[ny];
324:                 texplot( &xt, &yt, &ipen );
325:                 ipen = 2;
326:             }
327:         }
328:     }
329:
330:     x0 = x0 + xa[7];
331:     y0 = y0 + ya[7];
332:     ipen = 3 ;
333: }
334: /*
335: *
336: * END OF SYMB4
337: *
338: */
339: if( *hh < 0.0 ) {
340:     x0 = x0 - h*sinth;
341:     y0 = y0 + h*costh;
342: }
343:
344: if( *nchar < 0 ) {
345:     x0 = xx;
346:     y0 = yy;
347: }
348:
349: ipen = 3;
350: texplot(&x0, &y0, &ipen);
351: }

```


src/cgview/vector.f

```

1: C
2: C -----
3: C .... normalize 3D vector ....
4: C
5:       subroutine NORMVEC( VX,   VY,   VZ )
6:       real*8 VX, VY, VZ
7:       real*8 XL
8:       XL = SQRT(VX**2+VY**2+VZ**2)
9:       if ( XL.gt.0.0 ) then
10:        VX = VX/XL
11:        VY = VY/XL
12:        VZ = VZ/XL
13:      end if
14:      return
15:    end
16: C
17: C -----
18: C .... outer product of two vectors ....
19: C   ( v{x,y,z} can overlap with {a,b}{x,y,z} on memory)
20: C
21:       subroutine VECXVEC( VX,   VY,   VZ,   AX,   AY,   AZ,   BX,
22: & BY,   BZ )
23:       real*8 VX, VY, VZ, AX, AY, AZ, BX, BY, BZ
24:       real*8 VXX, VYY, VZZ
25:       VXX = AY*BZ - BY*AZ
26:       VYY = AZ*BX - BZ*AX
27:       VZZ = AX*BY - BX*AY
28:       VX = VXX
29:       VY = VYY
30:       VZ = VZZ
31:       return
32:     end
33: C
34: C -----
35: C .... Rotate vector v{x,y,z} round a{x,y,z} by th radian.
36: C   v{x,y,z}2 is the result.
37: C
38: C   a{x,y,z} must be a unit vector.
39: C
40:       subroutine ROTVEC( V2X,   V2Y,   V2Z,   VX,   VY,   VZ,   AX,
41: & AY,   AZ,   TH )
42: C
43:       implicit real*8(A-H,O-Z)
44: C
45: C
46: C
47: C   ..... {a}
48: C   .
49: C   .
50: C   . {p} = {v} - ({v},{a}){a}
51: C   {v} .
52: C   .
53: C   .
54: C   .
55: C
56:       P = VX*AX + VY*AY + VZ*AZ
57:       PX = VX - P*AX
58:       PY = VY - P*AY
59:       PZ = VZ - P*AZ
60:       XL = SQRT(PX**2+PY**2+PZ**2)
61: C
62: C ... {v} is nearly parallel to {a}
63: C
64:       if ( XL.lt.1.0D-10 ) then
65:         V2X = VX
66:         V2Y = VY
67:         V2Z = VZ
68:         return
69:       end if
70: C
71: C ... {q} = {a} x {p} : vertical to both {a} and {p}
72: C
73:       call VECXVEC( QX, QY, QZ, AX, AY, AZ, PX, PY, PZ )
74:       call NORMVEC( QX, QY, QZ )
75:       QX = XL*QX
76:       QY = XL*QY
77:       QZ = XL*QZ
78: C
79: C ... and {v2} = ({a},{v}){a} + cos(th)*{p} + sin(th)*{q}
80: C
81:       SINTH = SIN(TH)
82:       COSTH = COS(TH)
83: C
84:       V2X = P*AX + PX*COSTH + QX*SINTH
85:       V2Y = P*AY + PY*COSTH + QY*SINTH
86:       V2Z = P*AZ + PZ*COSTH + QZ*SINTH
87: C
88:       return
89:     end

```

src/cgview/verstr.f

```
1:      subroutine VERSTR
2: C=====
3: C CGVIEW version string setting
4: C=====
5: C
6: C   ... 20 Jan 1999
7: C     call HVERSN( 'CGVIEW V3.0 beta1' )
8: C   ... 30 Jan 1999
9: C     call HVERSN( 'CGVIEW V3.0 beta2','30 January 1999' )
10: C
11: C
12: C     call HVERSN( 'CGVIEW V3.0 beta3','10 February 1999' )
13: C
14: C     call HVERSN( 'CGVIEW V3.0 beta4','18 April 1999' )
15: C
16: C     call HVERSN( 'CGVIEW V3.0 beta5','27 April 1999' )
17: C     call HVERSN( 'CGVIEW V3.0 beta6','25 May 1999' )
18: C     call HVERSN( 'CGVIEW V3.0 beta7','21 June 1999' )
19: C     call HVERSN( 'CGVIEW V3.0 beta8','7 July 1999' )
20: C     call HVERSN( 'CGVIEW V3.0 beta9','9 August 1999' )
21: C     call HVERSN( 'CGVIEW V3.0 beta10','5 October 1999' )
22: C     call HVERSN( 'CGVIEW V3.0 beta11','22 October 1999' )
23: C     call HVERSN( 'CGVIEW V3.0 beta12pre','22 October 1999' )
24: C     call HVERSN( 'CGVIEW V3.0 beta12','9 November 1999' )
25: C     call HVERSN( 'CGVIEW V3.0 beta13pre','9 November 1999' )
26: C     call HVERSN( 'CGVIEW V3.0 beta13','22 November 1999' )
27: C     call HVERSN( 'CGVIEW V3.0 beta14','24 December 1999' )
28: C     call HVERSN( 'CGVIEW V3.0 beta15pre','24 December 1999' )
29: C     call HVERSN( 'CGVIEW V3.0 beta15pre1','20 February 2000' )
30: C     call HVERSN( 'CGVIEW V3.0 beta15pre2','25 February 2000' )
31: C     call HVERSN( 'CGVIEW V3.0 beta15pre3','27 February 2000' )
32: C     call HVERSN( 'CGVIEW V3.0 beta15','2 April 2000' )
33: C     call HVERSN( 'CGVIEW V3.0 beta16pre1','3 April 2000' )
34: C     call HVERSN( 'CGVIEW V3.0 beta16','30 April 2000' )
35: C
36: C   ... revision beta17 is given for CGVIEW on JHET
37: C
38: C     call HVERSN( 'CGVIEW V3.0 beta18','21 September 2000' )
39: C     call HVERSN( 'CGVIEW V3.0 beta19','10 December 2001' )
40: C
41:      return
42:      end
```

src/cgview/vmntr.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine VMNTR( NS,   IOW )
3: C=====
4: C  PURPOSE:   DUMMY SUBROUTINE IN SLICE
5: C  CALLED IN: LATICE
6: C=====
7: C
8:   return
9: C
10: C ..... CPU MONITORING .....
11: C
12:   entry VMNTR0(NS,IOW)
13:   return
14: C
15:   entry VMNTR2(NS)
16: C
17:   return
18: C
19: C ..... CPU MONITORING OF ROUTINE-NS CALLED FROM ROUTINE-NU .....
20: C
21: C
22:   entry VMNTR00(NS,NU)
23: C
24:   return
25: C
26:   entry VMNTR22(NS,NU)
27: C
28:   return
29: C
30: C*****... VECTOR LENGTH MONITORING .....
31: C ..... PARTICLES-PER-CALL MONITORING .....
32: C
33:   entry VMNTR1(NS,NN)
34:   return
35: C
36: C
37: C ***** CLEAR ALL MONITOR ARRAYS *****
38: C
39:   entry VMNTRC
40: C
41:   return
42: C
43: C ***** GIVE SCPU A VALUE *****
44: C
45:   entry VMNTRG(TT,NS,NU)
46:   return
47: C
48:   entry VMNTRH(TT2,NS)
49:   return
50: C
51: C ***** TAKE A VALUE FROM CPU *****
52: C
53:   entry VMNTRT(TT,NS,NU)
54:   return
55: C
56: C
57: C
58: C ..... PRINT OUT MONITORING INFORMATION .....
59: C
60: C
61:   entry VMNTRF(IOW)
62: C
63: C ..... CPU SUM FOR EACH ROTINE ....
64: C
65: C
66:   return
67:   end
```

src/cgview/wrkary.f

```

1:      subroutine WRKARY( LWORK, LAST,  LIMIT, MAXSF )
2: C=====
3: C  PURPOSE : CALCULATE WORKIG-ARRAY-POINTERS AND SIZE OF MEMORY
4: C          FOR EACH EVENT-PROCESSING ROUTINE.
5: C  CALLED IN : INTRO
6: C  CALLS : KEEP
7: C=====
8:      include 'INC/_FLAGS'
9:      include '../shared/INC/_SIZES'
10: C
11:      include 'INC/_WKSOU'
12:      include 'INC/_WKFL1'
13:      include 'INC/_WKSE1'
14:      include 'INC/_WKLAT'
15:      include 'INC/_WKGRA'
16: C
17:      include '../shared/INC/_IOUNIT'
18: C
19: C
20: CCCC call KEPP( 'WDUMY', LWK, 1, 'R8', LAST, 1.234D-12, JDEBG )
21: C
22:      call GTLAST( LWORK )
23: C
24: C ..... WORKING ARRAY OVERFLOW COUNTER .....
25: C
26:      IOVFL = 0
27:      MAXWK = 0
28: C
29: C
30: C ===== FOR SOURCE =====
31: C
32: C      LASTP = LWORK
33:      call STLAST( 'SOURCE', LWORK, LASTP )
34:      call LKEEP( 'IWK1 ', P1(1), NBANK, 'I4', LASTP, JDEBG )
35:      call LKEEP( 'X ', P1(2), NBANK, 'R8', LASTP, JDEBG )
36:      call LKEEP( 'Y ', P1(3), NBANK, 'R8', LASTP, JDEBG )
37:      call LKEEP( 'Z ', P1(4), NBANK, 'R8', LASTP, JDEBG )
38:      call LKEEP( 'IFL ', P1(5), NBANK, 'I4', LASTP, JDEBG )
39:      call LKEEP( 'IFI ', P1(6), NBANK, 'I4', LASTP, JDEBG )
40:      call LKEEP( 'R ', P1(7), 4*NBANK, 'R4', LASTP, JDEBG )
41:      call LKEEP( 'IWK2 ', P1(8), NBANK, 'I4', LASTP, JDEBG )
42:      call LKEEP( 'ISRF2 ', P1(9), 2*NBANK, 'I4', LASTP, JDEBG )
43:      call LKEEP( 'FXYZ2 ', P1(10), 2*NBANK, 'R4', LASTP, JDEBG )
44: C
45:      if ( LSIZL(LASTP).gt.LIMIT ) IOVFL = IOVFL + 1
46:      MAXWK = MAX(MAXWK,LASTP)
47:      write(IPR,7000) 'SOURCE', LASTP - LWORK, LASTP
48: C
49: C
50: C ===== FOR FLIONE =====
51: C
52: C      LASTP = LWORK
53:      call STLAST( 'FLIONE', LWORK, LASTP )
54:      call LKEEP( 'X ', P0(1), NBANK, 'R8', LASTP, JDEBG )
55:      call LKEEP( 'Y ', P0(2), NBANK, 'R8', LASTP, JDEBG )
56:      call LKEEP( 'Z ', P0(3), NBANK, 'R8', LASTP, JDEBG )
57:      call LKEEP( 'AI ', P0(4), NBANK, 'R8', LASTP, JDEBG )
58:      call LKEEP( 'BI ', P0(5), NBANK, 'R8', LASTP, JDEBG )
59:      call LKEEP( 'CI ', P0(6), NBANK, 'R8', LASTP, JDEBG )
60:      call LKEEP( 'W ', P0(7), NBANK, 'R8', LASTP, JDEBG )
61:      call LKEEP( 'DI ', P0(8), NBANK, 'R8', LASTP, JDEBG )
62:      call LKEEP( 'IGI ', P0(9), NBANK, 'I4', LASTP, JDEBG )
63:      call LKEEP( 'IBP ', P0(10), NBANK, 'I4', LASTP, JDEBG )
64:      call LKEEP( 'IFC ', P0(11), NBANK, 'I4', LASTP, JDEBG )
65:      call LKEEP( 'R ', P0(12), 3*NBANK, 'R4', LASTP, JDEBG )

```

```

66:      call LKEEP( 'DLOC1 ', P0(13), NBANK, 'R8', LASTP, JDEBG )
67:      call LKEEP( 'DLOC2 ', P0(14), NBANK, 'R8', LASTP, JDEBG )
68:      call LKEEP( 'IKL ', P0(15), NBANK, 'I4', LASTP, JDEBG )
69:      call LKEEP( 'IKL2 ', P0(16), NBANK, 'I4', LASTP, JDEBG )
70:      if ( JFISX.ne.0 .or. JHLAT.ne.0 ) then
71:          call LKEEP( 'XUP ', P0(17), NBANK, 'R8', LASTP, JDEBG )
72:          call LKEEP( 'YUP ', P0(18), NBANK, 'R8', LASTP, JDEBG )
73:          call LKEEP( 'ZUP ', P0(19), NBANK, 'R8', LASTP, JDEBG )
74:          call LKEEP( 'AUP ', P0(20), NBANK, 'R8', LASTP, JDEBG )
75:          call LKEEP( 'BUP ', P0(21), NBANK, 'R8', LASTP, JDEBG )
76:          call LKEEP( 'CUP ', P0(22), NBANK, 'R8', LASTP, JDEBG )
77:      end if
78:      call LKEEP( 'TI', P0(23), NBANK, 'R8', LASTP, JDEBG )
79:      call LKEEP( 'ISTG ', P0(24), NBANK, 'I4', LASTP, JDEBG )
80:      call LKEEP( 'ILUP ', P0(25), NBANK, 'I4', LASTP, JDEBG )
81:      call LKEEP( 'ILST ', P0(26), NBANK, 'I4', LASTP, JDEBG )
82:
83:      if ( JFISX.ne.0 ) then
84:          call LKEEP( 'XYZ1 ', P0(27), 3*NBANK, 'R8', LASTP, JDEBG )
85:          call LKEEP( 'ABC1 ', P0(28), 3*NBANK, 'R8', LASTP, JDEBG )
86:          call LKEEP( 'XYZ2 ', P0(29), 3*NBANK, 'R8', LASTP, JDEBG )
87:          call LKEEP( 'ABC2 ', P0(30), 3*NBANK, 'R8', LASTP, JDEBG )
88:          call LKEEP( 'DDI ', P0(31), NBANK, 'R8', LASTP, JDEBG )
89:          call LKEEP( 'IBP0 ', P0(32), NBANK, 'I4', LASTP, JDEBG )
90:          call LKEEP( 'IBP1 ', P0(33), NBANK, 'I4', LASTP, JDEBG )
91:          call LKEEP( 'IBP2 ', P0(34), NBANK, 'I4', LASTP, JDEBG )
92:          call LKEEP( 'KPLT ', P0(35), NLBZ+1, 'I4', LASTP, JDEBG )
93:          call LKEEP( 'JKSF ', P0(36), NLBZ, 'I4', LASTP, JDEBG )
94:      end if
95: C
96:
97:      if ( LSIZL(LASTP).gt.LIMIT ) IOVFL = IOVFL + 1
98:      MAXWK = MAX(MAXWK,LASTP)
99:      write(IPR,7000) 'FLIONE', LASTP - LWORK, LASTP
100: C
101: C ===== FOR SEAOONE =====
102: C
103: C      LASTP = LWORK
104:      call STLAST( 'SEAOONE', LWORK, LASTP )
105:      call LKEEP( 'X ', P2(1), NBANK, 'R8', LASTP, JDEBG )
106:      call LKEEP( 'Y ', P2(2), NBANK, 'R8', LASTP, JDEBG )
107:      call LKEEP( 'Z ', P2(3), NBANK, 'R8', LASTP, JDEBG )
108:      call LKEEP( 'W ', P2(4), NBANK, 'R8', LASTP, JDEBG )
109:      call LKEEP( 'IBP ', P2(5), NBANK, 'I4', LASTP, JDEBG )
110:      call LKEEP( 'IFI ', P2(6), NBANK, 'I4', LASTP, JDEBG )
111:      call LKEEP( 'IZI ', P2(7), NBANK, 'I4', LASTP, JDEBG )
112:      call LKEEP( 'IWK ', P2(8), NBANK, 'I4', LASTP, JDEBG )
113:      call LKEEP( 'IFL ', P2(9), NBANK, 'I4', LASTP, JDEBG )
114:      call LKEEP( 'R ', P2(10), NBANK, 'R4', LASTP, JDEBG )
115:      call LKEEP( 'ISRF ', P2(11), NBANK, 'I4', LASTP, JDEBG )
116:      call LKEEP( 'FXYZ ', P2(12), NBANK, 'R4', LASTP, JDEBG )
117:      if ( LSIZL(LASTP).gt.LIMIT ) IOVFL = IOVFL + 1
118:      MAXWK = MAX(MAXWK,LASTP)
119:      write(IPR,7000) 'SEAOONE', LASTP - LWORK, LASTP
120: C
121: C ===== FOR GRAINT =====
122: C
123: C      LASTP = LWORK
124:      call STLAST( 'GRAINT', LWORK, LASTP )
125:      call LKEEP( 'X01 ', P3(1), NBANK, 'R4', LASTP, JDEBG )
126:      call LKEEP( 'Y01 ', P3(2), NBANK, 'R4', LASTP, JDEBG )
127:      call LKEEP( 'IZZx ', P3(3), NBANK, 'I4', LASTP, JDEBG )
128:      call LKEEP( 'IZ0 ', P3(4), NBANK, 'I4', LASTP, JDEBG )
129:      call LKEEP( 'IZ01 ', P3(5), NBANK, 'I4', LASTP, JDEBG )
130:      call LKEEP( 'CLR ', P3(6), NBANK, 'I4', LASTP, JDEBG )

```

src/cgview/wrkary.f

```

131:      call LKEEP( 'X02  ', P3(7), NBANK, 'R4', LASTP, JDEBG )
132:      call LKEEP( 'Y02  ', P3(8), NBANK, 'R4', LASTP, JDEBG )
133:      call LKEEP( 'BCLR ', P3(9), NBANK, 'I4', LASTP, JDEBG )
134:      call LKEEP( 'XC01 ', P3(10), NBANK, 'R4', LASTP, JDEBG )
135:      call LKEEP( 'YC01 ', P3(11), NBANK, 'R4', LASTP, JDEBG )
136:      if ( LSIZL(LASTP).gt.LIMIT ) IOVFL = IOVFL + 1
137:      MAXWK = MAX(MAXWK,LASTP)
138:      write(IPR,7000) 'GRAXX ', LASTP - LWORK, LASTP
139: C
140: C
141: C ===== FOR LATTICE =====
142: C
143:      if ( JLATT.ne.0 ) then
144: C      LASTP = LWORK
145:          call STLAST( 'LATTICE', LWORK, LASTP )
146:          call LKEEP( 'X    ', P5(1), NBANK, 'R8', LASTP, JDEBG )
147:          call LKEEP( 'Y    ', P5(2), NBANK, 'R8', LASTP, JDEBG )
148:          call LKEEP( 'Z    ', P5(3), NBANK, 'R8', LASTP, JDEBG )
149:          call LKEEP( 'AI   ', P5(4), NBANK, 'R8', LASTP, JDEBG )
150:          call LKEEP( 'BI   ', P5(5), NBANK, 'R8', LASTP, JDEBG )
151:          call LKEEP( 'CI   ', P5(6), NBANK, 'R8', LASTP, JDEBG )
152:          call LKEEP( 'IWK  ', P5(7), NBANK, 'I4', LASTP, JDEBG )
153:          call LKEEP( 'JKSF ', P5(8), NZONE+1, 'I4', LASTP, JDEBG )
154:          call LKEEP( 'IPZONE', P5(9), NZONE+1, 'I4', LASTP, JDEBG )
155:          call LKEEP( 'MEM   ', P5(10), NZONE+1, 'I4', LASTP, JDEBG )
156: C ... added for ver3.0
157:          if ( JFISX.ne.0 .or. JSTGR.ne.0 ) then
158:              call LKEEP( 'DI   ', P5(11), NBANK, 'R8', LASTP, JDEBG )
159:              call LKEEP( 'DLOC1 ', P5(12), NBANK, 'R8', LASTP, JDEBG )
160:              call LKEEP( 'DLOC2 ', P5(13), NBANK, 'R8', LASTP, JDEBG )
161:              call LKEEP( 'IKL   ', P5(14), NBANK, 'I4', LASTP, JDEBG )
162:              call LKEEP( 'IKL2  ', P5(15), NBANK, 'I4', LASTP, JDEBG )
163:              call LKEEP( 'R     ', P5(16), 3*NBANK, 'I4', LASTP, JDEBG )
164:          end if
165:          if ( LSIZL(LASTP).gt.LIMIT ) IOVFL = IOVFL + 1
166:          MAXWK = MAX(MAXWK,LASTP)
167:          write(IPR,7000) 'LATTICE', LASTP - LWORK, LASTP
168:      end if
169: C
170: C
171: C
172: C
173: 7000 format(/' === SUBROUTINE <'A6,> NEEDS WORKING AREA OF ' ,I10,
174:      &      ' WORDS. ==== ( LAST POSITION ',I10,')')
175: C
176: C
177:      LAST = MAXWK
178:      if ( LSIZL(LAST).gt.LIMIT ) then
179:          write(IPR,7020) LAST, LIMIT
180: 7020      format(' XXX INSUFFICIENT MEMORY TO KEEP WORKING AREA',
181:      &      ' ( NECESSARY ',I10,' WORDS. LIMIT ',I10,'WORDS)')
182:          call CNTERR( 'FATAL' )
183:      end if
184: C
185:      return
186:      end

```

src/cgview/xdraw.c

```

1: /*****
2:  */
3:  /* xdraw.c : Drawing Routines on X Window System */
4:  /*
5:  /* Modified for use in CGVIEW */
6:  /* 20 Nov 1996: exit when failed to open display. */
7:  /* 19 Sep 1998: Enable backing store */
8:  /* 4 Oct 1998: Enable specification of window width (pixel) */
9:  /* thru an environment variable CGVWIDTH. */
10: /* 6 Apr 1999: draw(); copy pixmap on (0,0) of window. */
11: /*
12: /*****
13: #include <X11/Xlib.h>
14: #include <X11/Xutil.h>
15: #include <stdio.h>
16: #include <stdlib.h>
17: #include <string.h>
18: #include "piflib.h"
19:
20:  */
21:  ** Paper size in mili meter (A4 landscape)
22:  */
23:
24: #define WIDTH_MM 290.0
25: #define HEIGHT_MM 210.0
26:
27:  */
28:  ** Paper margin in mili meter (A4 landscape)
29:  */
30:
31: #define XMARGIN_MM 20.0
32: #define YMARGIN_MM 17.0
33:
34: /*** Drawing origin in paper **/
35:
36: #define XORIG XMARGIN_MM
37: #define YORIG YMARGIN_MM
38:
39:  */
40:  ** Display drawing is performed for
41:  * XMARGIN_MM - xmargin_WIN < x < WIDTH_MM-(XMARGIN_MM-xmargin_WIN)
42:  * YMARGIN_MM - ymargin_WIN < y < HEIGHT_MM-(YMARGIN_MM-ymargin_WIN)
43:  *
44:  */
45:
46: /*** 5.0mm display margin
47: double xmargin_WIN = 5.0 ;
48: double ymargin_WIN = 5.0 ;
49: *****/
50:
51: /*** no display margin ***/
52:
53: double xmargin_WIN = 0.0;
54: double ymargin_WIN = 0.0;
55:
56: double xmargin;
57: double ymargin;
58:
59: /*** add a flag : this is to enable no-wait drawing ***/
60: /*** Jul 1996 M.Sasaki ****/
61: /*** Set by "call plot(0,0,111)" ****/
62: /*** Disabled by "call plot(0,0,112)" ****/
63:
64: static wait_key_event = 1;
65:
66:  */
67:  * Window width and height in pixel
68:  */
69: float win_width, win_height;
70:
71: #define UINT unsigned int
72:
73:
74: unsigned long GetColor(color_name)
75: char *color_name;
76: {
77:     XColor c0, c1;
78:
79:     XAllocNamedColor(d, cmap, color_name, &c1, &c0);
80:
81:     return (c1.pixel);
82: }
83:
84:
85: /*****
86:  * draw screen by copying pixmap "p" on window "w"
87:  *****/
88:
89: draw()
90: {
91:     int x, y;
92:     unsigned int width, height, border_width, depth;
93:
94:     XGetGeometry(d, w, &root, &x, &y, &width, &height,
95:                 &border_width, &depth);
96:     /***
97:     * XCopyArea(d, p, w, gc, 0, DisplayHeight(d,0) - height,
98:     ***/
99:     XCopyArea(d, p, w, gc, 0, 0,
100:              width, height, 0, 0);
101:     XFlush(d);
102: }
103:
104:
105: redraw()
106: {
107:
108:     if (ifdraw == 1) {
109:         draw();
110:     }
111:     if (wait_key_event) {
112:         while (1) {
113:             XNextEvent(d, &e);
114:             if (e.type == Expose) {
115:                 draw();
116:             }
117:             if (e.type == ConfigureNotify) {
118:                 draw();
119:             }
120:             if (e.type == KeyPress) {
121:                 break;
122:             }
123:         }
124:     }
125: }
126:
127: #define MAX_W_NAME 128
128: char w_name[MAX_W_NAME+1];
129:
130: xwinname( name, length )

```

src/cgview/xdraw.c

```
131: char * name;
132: int length ;
133: {
134:     int n;
135:     if ( length > MAX_W_NAME ) {
136:         n = MAX_W_NAME ;
137:     } else {
138:         n = length ;
139:     }
140:     strncpy( w_name, name, n ) ;
141: }
142:
143: xplots()
144: {
145:     int x, y;
146:     unsigned int width, height, border_width, depth;
147:
148:     char *cgv_width;
149:
150:     if ((d = XOpenDisplay(NULL)) == NULL) {
151:         fprintf(stderr, "***PIFFLIB(xplots) Failed to open display\n");
152:         exit(1);
153:     }
154:     /*
155:      * .... get display width (pixel) from environment variable CGVWIDTH ...
156:      */
157:     cgv_width = getenv("CGVWIDTH");
158:
159:     if (cgv_width != NULL) {
160:         if (1 != sscanf(cgv_width, "%f", &win_width)) {
161:             fprintf(stderr,
162:                 "***PIFFLIB(xplots) Failed to convert environment var. CGV
_WIDTH : %s\n", cgv_width);
163:             exit(1);
164:         }
165:         win_height = HEIGHT_MM / WIDTH_MM * win_width;
166:     } else {
167:
168:
169:     /*
170:      * .... Using DisplayWidthMM, DisplayHeightMM is ridiculous ...
171:      *      (in what way the machine knows the actual mili-meter size of display?
172:      */
173:         win_width =
174:             (float) DisplayWidth(d,0) * (WIDTH_MM/(float)DisplayWidthMM(d,0));
175:         win_height =
176:             (float) DisplayHeight(d,0) * (HEIGHT_MM/(float)DisplayHeightMM(d,0))
177:     }
178:
179:     w = XCreateSimpleWindow(d, RootWindow(d, 0), 100, 100, (UINT) win_width,
180:         (UINT) win_height, 0, BlackPixel(d, 0), WhitePixel(d, 0));
181:     /*
182:         (UINT) win_height, 2, BlackPixel(d, 0), WhitePixel(d, 0));
183:     */
184:
185:     if ( strlen(w_name) == 0 ) {
186:         strcpy( w_name, "Piflib" ) ;
187:     }
188:
189:     XStoreName(d, w, w_name);
190:
191:     XMapWindow(d, w);
192:
193:     XSelectInput(d, w, ExposureMask | KeyPressMask);
194:
195:     p = XCreatePixmap(d, w, (UINT) win_width, (UINT) win_height,
196:         DefaultDepth(d, 0));
197:
198:     gc = XCreateGC(d, p, 0, 0);
199:     /**
200:         XSetLineAttributes(d, gc, 2, LineSolid, CapRound, JoinRound);
201:     */
202:
203:     XSetLineAttributes(d, gc, 1, LineSolid, CapRound, JoinRound);
204:
205:     XSetForeground(d, gc, WhitePixel(d, 0));
206:
207:     XFillRectangle(d, p, gc, 0, 0, (UINT) win_width, (UINT) win_height);
208:
209:     XSetForeground(d, gc, BlackPixel(d, 0));
210:
211:     /**
212:         Enable backing store (if possible)
213:     */
214:
215:     attributes.backing_store = Always;
216:     XChangeWindowAttributes(d, w,
217:         CWBackingStore, &attributes);
218:
219:     fs = XLoadQueryFont(d, "6x12");
220:     XSetFont(d, gc, fs->fid);
221:
222:     wmhints.flags = InputHint;
223:     wmhints.input = True;
224:     XSetWMHints(d, w, &wmhints);
225:
226:     xmargin = XORIG - xmargin_WIN;
227:     ymargin = YORIG - ymargin_WIN;
228:
229:     x_pix_mm = (double) win_width / (double) (WIDTH_MM - 2 * xmargin);
230:     y_pix_mm = (double) win_height / (double) (HEIGHT_MM - 2 * ymargin);
231:
232:     x_org = XORIG - xmargin;
233:     y_org = HEIGHT_MM - 2 * ymargin - (YORIG - ymargin);
234:
235:     /*
236:         printf( "plots  %f %f %f %f \n", xmargin, ymargin, x_org, y_org) ;
237:         printf( "plots  %f %f \n", x_pix_mm, y_pix_mm) ;
238:     */
239:
240:     ifdraw = 0;
241:     f_value = 1.0;
242:     x_point0 = x_org;
243:     y_point0 = y_org;
244:
245:     cmap = DefaultColormap(d, 0);
246:
247:     color[0] = GetColor("black");
248:     color[1] = GetColor("white");
249:     color[2] = GetColor("red");
250:     color[3] = GetColor("green");
251:     color[4] = GetColor("blue");
252:     color[5] = GetColor("cyan");
253:     color[6] = GetColor("magenta");
254:     color[7] = GetColor("yellow");
255: }
256:
257: /**
```

src/cgview/xdraw.c

```

258: * return window ID
259: *****/
260: xwinid( windowid )
261: int *windowid ;
262: {
263:     *windowid=(int)w ;
264: }
265:
266:
267: xplot(x_point1, y_point1, iflag)
268: float x_point1, y_point1;
269: int iflag;
270: {
271:     float x_draw0, y_draw0, x_draw1, y_draw1;
272:     float xxx, yyy;
273:
274:     xxx = x_point1;
275:     yyy = y_point1;
276:
277:     x_store = xxx;
278:     y_store = yyy;
279:
280:     xxx *= f_value;
281:     yyy *= f_value;
282: /**/ temp /**/
283: /**/
284:     switch( iflag ) {
285:         case 2 : plot_cnt[0]++ ;
286:             break;
287:         case 3 : plot_cnt[1]++ ;
288:             break;
289:         case 999 : plot_cnt[2]++ ;
290:             break;
291:         case 666 : plot_cnt[3]++ ;
292:             break;
293:         case 777 : plot_cnt[4]++ ;
294:             break;
295:         case 888 : plot_cnt[5]++ ;
296:             break;
297:     }
298: /**/
299:
300:
301: /**/
302: * Close window and exit
303: *****/
304:
305:     if (iflag == 999) {
306:         if (ifdraw == 1) {
307:             redraw();
308:         }
309:         XClearWindow(d, w);
310:         XFreePixmap(d, p);
311:         exit(0);
312:     }
313: /**/
314: * Redraw
315: *****/
316:
317:     if (iflag == 888 || iflag == 777) {
318:         draw();
319:     }
320: /**/
321: * Clear window
322: *****/

```

```

323:
324:     if ( iflag == 666 ) {
325:         redraw();
326:         XClearWindow(d, w);
327:         XSetForeground(d, gc, WhitePixel(d, 0));
328:         XFillRectangle(d, p, gc, 0, 0, (UINT) win_width, (UINT) win_height);
329:         XSetForeground(d, gc, BlackPixel(d, 0));
330: /**/
331:         XSetLineAttributes(d, gc, 2, LineSolid, CapRound, JoinRound);
332: /**/
333:
334:         XSetLineAttributes(d, gc, 1, LineSolid, CapRound, JoinRound);
335:
336:         xmargin = XORIG - xmargin_WIN;
337:         ymargin = YORIG - ymargin_WIN;
338:
339:         x_org = XORIG - xmargin;
340:         y_org = HEIGHT_MM - 2 * ymargin - (YORIG - ymargin);
341:
342:         x_point0 = x_org;
343:         y_point0 = y_org;
344:
345:         ifdraw = 0;
346: /**/ f_value = 1.0; /**/
347: /**/ temp /**/
348: /**/
349:         printf("draw counts monitor :%d \n",draw_cnt) ;
350:         printf("(2)%d (3)%d (999)%d (666)%d (777)%d (888)%d\n",
351:             plot_cnt[0],
352:             plot_cnt[1],
353:             plot_cnt[2],
354:             plot_cnt[3],
355:             plot_cnt[4],
356:             plot_cnt[5]);
357: /**/
358:     }
359: /**/
360:     printf( "xplot  %f %f %f %f %f %f\n", xmargin, ymargin, x_org, y_org,
361:         *x_point1, *y_point1 ) ;
362: /**/
363:
364: /**/
365: * Draw line
366: *****/
367:
368:     if (iflag == 2) {
369:         x_draw0 = x_point0 * x_pix_mm;
370:         y_draw0 = y_point0 * y_pix_mm;
371:         x_draw1 = (x_org + xxx) * x_pix_mm;
372:         y_draw1 = (y_org - yyy) * y_pix_mm;
373:
374: /**/
375:         printf( "draw  %f %f %f %f %f %f\n",x_draw0,y_draw0,x_draw1,y_draw1,
376:             x_pix_mm,y_pix_mm ) ;
377: /**/
378:
379:         XDrawLine(d, p, gc, (int) x_draw0, (int) y_draw0,
380:             (int) x_draw1, (int) y_draw1);
381:
382:         x_point0 = x_org + xxx;
383:         y_point0 = y_org - yyy;
384:
385:         ifdraw = 1;
386:     }
387: /**/

```


src/cgview/xdraw.c

```
388:  * Move position
389:  *****/
390:
391:  if (iflag == 3) {
392:      x_point0 = x_org + xxx;
393:      y_point0 = y_org - yyy;
394:  }
395:  /***/
396:  * Draw line and move origin
397:  *****/
398:
399:  if (iflag == -2) {
400:      x_draw0 = x_point0 * x_pix_mm;
401:      y_draw0 = y_point0 * y_pix_mm;
402:      x_draw1 = (x_org + xxx) * x_pix_mm;
403:      y_draw1 = (y_org - yyy) * y_pix_mm;
404:
405:      XDrawLine(d, p, gc, (int) x_draw0, (int) y_draw0,
406:                 (int) x_draw1, (int) y_draw1);
407:      x_point0 = x_org + xxx;
408:      y_point0 = y_org - yyy;
409:
410:      x_org = x_point0;
411:      y_org = y_point0;
412:
413:      ifdraw = 1;
414:  }
415:  /***/
416:  * move origin
417:  *****/
418:
419:  if (iflag == -3) {
420:      x_point0 = x_org + xxx;
421:      y_point0 = y_org - yyy;
422:      x_org = x_point0;
423:      y_org = y_point0;
424:  }
425:
426:  /***/
427:  * disable key event
428:  *****/
429:
430:  if (iflag == 111) {
431:      wait_key_event = 0;
432:  }
433:  /***/
434:  * enable key event
435:  *****/
436:
437:  if (iflag == 112) {
438:      wait_key_event = 1;
439:  }
440:  /***/
441:  XFlush(d);
442:  /***/
443: }
444:
445: xdot(x1, y1)
446: float *x1, *y1;
447: {
448:     float x_dot, y_dot;
449:     float xxx, yyy;
450:
451:     xxx = *x1;
452:     yyy = *y1;
453:
454:     x_dot = (x_org + xxx) * x_pix_mm;
455:     y_dot = (y_org - yyy) * y_pix_mm;
456:
457:     XDrawPoint(d, p, gc, (int) x_dot, (int) y_dot);
458:
459:     ifdraw = 1;
460:
461:     XFlush(d);
462: }
463:
464: #define SIZE_PPBUF 1024
465:
466: XPoint pp[SIZE_PPBUF];
467:
468: xplotlines( nml, xpoints, ypoints )
469:     int nml;
470:     float *xpoints, *ypoints;
471: {
472:     int i;
473:
474:     for ( i=0 ; i < nml ; i++ ) {
475:         pp[i].x = (x_org + xpoints[i]*f_value) * x_pix_mm;
476:         pp[i].y = (y_org - ypoints[i]*f_value) * y_pix_mm;
477:     }
478:     XDrawLines( d, p, gc, pp, nml, CoordModeOrigin );
479:     /***/
480:     plot( xpoints[0], ypoints[0], 3 );
481:     for ( i=1 ; i < nml ; i++ ) {
482:         plot( xpoints[i], ypoints[i], 2 );
483:     }
484:     /***/
485:     ifdraw = 1;
486:     XFlush(d);
487: }
488:
```

src/cgview/xsdumy.f

```
1: C
2: C === Dummy entry for MVP cross section input routines
3: C
4: C      9 Jun 1999
5: C
6:      subroutine xsec()
7:      return
8:      end
9: C
10: C
11:      subroutine xsecp()
12:      return
13:      end
14: C
15:      subroutine xsece()
16:      return
17:      end
18: C
19:      subroutine xsecd()
20:      return
21:      end
```

src/cgview/xsetattr.c

```

1: /*****
2:  */
3:  /* xsetattr.c : Set Attributes Routines for Window */
4:  */
5:  *****/
6:  #include <X11/Xlib.h>
7:  #include <X11/Xutil.h>
8:  #include <stdio.h>
9:  #include "piflib.h"
10:
11: xsetrgb( red, green, blue )
12: unsigned int *red, *green, *blue;
13: {
14:     XColor color;
15:
16:     unsigned long rrr, ggg, bbb;
17:
18:     rrr = *red;
19:     ggg = *green;
20:     bbb = *blue;
21:
22:     rrr <= 8;
23:     ggg <= 8;
24:     bbb <= 8;
25:
26:     rrr |= *red;
27:     ggg |= *green;
28:     bbb |= *blue;
29:
30:     color.red = rrr;
31:     color.green = ggg;
32:     color.blue = bbb;
33:
34:     XAllocColor(d, cmap, &color);
35:     XSetForeground(d, gc, color.pixel);
36: }
37:
38: /*****
39:  set a global data colorGCs[color_number] to
40:  a specified RGB value
41:  ***
42:  ***
43:  Added by M.Sasaki (9 Nov 1996)
44:  **/
45:
46: xsetrgbnum( color_number, red, green, blue )
47: unsigned int *color_number;
48: unsigned int *red, *green, *blue;
49: {
50:
51:     XColor color0 ;
52:
53:     unsigned long rrr, ggg, bbb;
54:
55:     rrr = *red;
56:     ggg = *green;
57:     bbb = *blue;
58:
59:     rrr <= 8;
60:     ggg <= 8;
61:     bbb <= 8;
62:
63:     rrr |= *red;
64:     ggg |= *green;
65:     bbb |= *blue;

```

```

66:
67:     if( *color_number >= MAX_NUM_OF_COLORS-1 ) {
68:         fprintf(stderr, "*** [Piflib error] too large color number in xsetrgbnu
m %d.\n",
69:             *color_number ) ;
70:         return 1 ;
71:     }
72:     color0.red = rrr;
73:     color0.green = ggg;
74:     color0.blue = bbb;
75:
76:     XAllocColor(d, cmap, &color0);
77:
78:     colors[*color_number]= color0.pixel;
79:     /**
80:     if( (colorGCs[*color_number] = XCreateGC( d, p, 0, 0 ))==NULL) {
81:         fprintf(stderr, "***PIFLIB(xsetrgbnum) Failed to create new GC\n");
82:         exit(1) ;
83:     }
84:     XCopyGC( d, gc, -1 , colorGCs[*color_number] ) ;
85:     XSetForeground(d, colorGCs[*color_number], color0.pixel);
86:     **/
87:
88:     return 0 ;
89: }
90:
91: xsetclrs( color_number )
92: int *color_number;
93: {
94:     XSetForeground(d, gc, colors[*color_number]);
95:     /**
96:     gc = colorGCs[*color_number] ;
97:     **/
98: }
99:
100: xsetclr( color_number )
101: int *color_number;
102: {
103:     int i;
104:
105:     i = *color_number;
106:     XSetForeground(d, gc, color[i]);
107: }
108:
109: xsetltp( kind_line )
110: int *kind_line;
111: {
112:     int k_line;
113:     XGCValues gv;
114:
115:     static char dash1[] = { 5, 5};
116:     static char dash2[] = {10, 5};
117:     static char dash3[] = {10, 5, 5, 5};
118:     static char dash4[] = {10, 5, 5, 5, 5};
119:     static char dash5[] = {10, 5, 5, 5, 5, 5, 5};
120:     static char dash6[] = {10, 5, 2, 5};
121:     static char dash7[] = {10, 5, 2, 2, 2, 5};
122:     static char dash8[] = {10, 5, 2, 2, 2, 2, 2, 5};
123:     static char dash9[] = {10, 5, 8, 5, 2, 5, 8, 5};
124:     static char dash10[] = { 2, 2};
125:     static char dash11[] = { 2, 2, 2, 2, 2, 10};
126:
127:     k_line = *kind_line;
128:     k_line++;
129:

```

src/cgview/xsetattr.c

```
130:     if( k_line != 1 ) {
131:         gv.line_style = LineOnOffDash;
132:         XChangeGC(d, gc, GCLineStyle, &gv);
133:     }
134:
135:     switch( k_line ) {
136:     case 1:
137:         gv.line_style = LineSolid;
138:         XChangeGC(d, gc, GCLineStyle, &gv);
139:
140:     case 2:
141:         break;
142:     case 3:
143:         XSetDashes(d, gc, 0, dash1, 2);
144:         break;
145:     case 4:
146:         XSetDashes(d, gc, 0, dash2, 2);
147:         break;
148:     case 5:
149:         XSetDashes(d, gc, 0, dash3, 4);
150:         break;
151:     case 6:
152:         XSetDashes(d, gc, 0, dash4, 6);
153:         break;
154:     case 7:
155:         XSetDashes(d, gc, 0, dash5, 8);
156:         break;
157:     case 8:
158:         XSetDashes(d, gc, 0, dash6, 4);
159:         break;
160:     case 9:
161:         XSetDashes(d, gc, 0, dash7, 6);
162:         break;
163:     case 10:
164:         XSetDashes(d, gc, 0, dash8, 8);
165:         break;
166:     case 11:
167:         XSetDashes(d, gc, 0, dash9, 8);
168:         break;
169:     case 12:
170:         XSetDashes(d, gc, 0, dash10, 2);
171:         break;
172:     case 13:
173:         XSetDashes(d, gc, 0, dash11, 6);
174:         break;
175:     }
176: }
177:
178: xsetlwd( wd_line )
179: int *wd_line;
180: {
181:     XGCValues gv;
182:
183:     gv.line_width = *wd_line;
184:     /* gv.line_width++; */
185:
186:     XChangeGC( d, gc, GCLineWidth, &gv);
187:
188: }
189:
190:
191: /** Added Sep 1999 by M.Sasaki */
192:
193: xcapline( iflag )
194: int iflag ;
195: {
196:     XGCValues gv ;
197:
198:     XGetGCValues(d, gc, GCCapStyle, &gv) ;
199:     switch ( iflag ) {
200:     case 1:
201:         if ( gv.cap_style != CapRound ) {
202:             gv.cap_style = CapRound ;
203:             XChangeGC( d, gc, GCCapStyle, &gv);
204:         }
205:         break;
206:     case 2:
207:         if ( gv.cap_style != CapButt ) {
208:             gv.cap_style = CapButt ;
209:             XChangeGC( d, gc, GCCapStyle, &gv);
210:         }
211:         break;
212:     }
213:
214:     /**
215:      printf("--- xcapline : %d\n", iflag ) ;
216:      */
217:
218:     XFlush(d) ;
219: }
220:
221: xgetcapline( iflag )
222: int *iflag;
223: {
224:     XGCValues gv ;
225:
226:     XGetGCValues(d, gc, GCCapStyle, &gv) ;
227:     *iflag = 0 ;
228:     switch ( gv.cap_style ) {
229:     case CapRound:
230:         *iflag = 1 ;
231:         break;
232:     case CapButt:
233:         *iflag = 2 ;
234:         break;
235:     }
236: }
237:
238:
239: xcpfont( font_name )
240: char *font_name;
241: {
242:     fs = XLoadQueryFont(d, font_name);
243:     XSetFont(d, gc, fs->fid);
244: }
245:
246: xfactor( factor_value)
247: float *factor_value;
248: {
249:     f_value = *factor_value;
250:
251: }
252:
253: xnewpen(ipen)
254: int *ipen;
255: {
256:     if ( *ipen == 1 ) {
257:         XSetForeground(d, gc, color[6]);
258:     } else if ( *ipen == 2 ) {
259:         XSetForeground(d, gc, color[0]);
260:     }
261: }
```

src/cgview/xsetattr.c

```
260:     }else if ( *ipen == 3 ) {
261:         XSetForeground(d, gc, color[3]);
262:     }
263: }
264:
265: xwhere( x, y, fact )
266:     float *x, *y, *fact;
267: {
268:     *x = x_store;
269:     *y = y_store;
270:     *fact = f_value;
271: }
272:
```

WAVE

src/cgview/xsymchar.c

```
1: /*****  
2:  */  
3:  /* xsymchar.c : Symbol Routines for Window */  
4:  */  
5:  /***/  
6:  #include <X11/Xlib.h>  
7:  #include <X11/Xutil.h>  
8:  #include <stdio.h>  
9:  #include <math.h>  
10: #include "piflib.h"  
11:  
12: xsymfnt( x_str, y_str, height, string, angle, no_of_letters)  
13: float *x_str, *y_str;  
14: float *height, *angle;  
15: int *no_of_letters;  
16: /** long *no_of_letters; */  
17: char *string;  
18: {  
19:     float xxx, yyy;  
20:  
21:     xxx = *x_str;  
22:     yyy = *y_str;  
23:  
24:     xxx *= x_pix_mm;  
25:     yyy *= y_pix_mm;  
26:  
27:     xxx = x_org + xxx;  
28:     yyy = y_org - yyy;  
29:  
30:     XDrawString( d, p, gc, (long)xxx, (long)yyy, string, *no_of_letters);  
31:  
32:     ifdraw = 1;  
33:  
34:     XFlush(d);  
35: }  
36:  
37: xsymbol(x, y, hh, ibcda, angle, nchar )  
38:  
39:     float *x, *y, *hh, *angle;  
40:     int *nchar;  
41:     int *ibcda;  
42:  
43: {  
44:     static float oldth = 9999.0;  
45:     static float oldfct = 9999.0;  
46:     static float rad = 0.017453;  
47:     static float x0 = 0.0;  
48:     static float y0 = 0.0;  
49:  
50:     static float DDV = 0.3;  
51:  
52:     static float eps = 0.01;  
53:     static long kp8 = 256;  
54:     static long kpl6 = 65536;  
55:     static int nxy = 0;  
56:     static int ichar = 0;  
57:  
58:     static float xa[9], ya[9];  
59:     static float fct, th, thl;  
60:     static float sinth, costh;  
61:  
62:     unsigned long ibit8, ibit16, ibit24;  
63:     unsigned int kpos, knum, ktab, kpt;  
64:     unsigned int nnt;  
65:     unsigned long idata;
```

```
66:     unsigned char ibcd[255];  
67:  
68:     long iebc;  
69:     int ipen, icb, nt, iswv;  
70:     int nx, ny;  
71:  
72:     float xt, yt;  
73:     float div, h;  
74:     float xx, yy;  
75:  
76:     int i, m, kk;  
77:  
78:     static long idxtab[128] = { 0x00080001, 0x030C0002, 0x00060016, 0x02070011  
79:                                0x00070013, 0x02070005, 0x00070007, 0x02080008  
80:                                0x010B000A, 0x0307000C, 0x010E000E, 0x020D0011  
81:                                0x02060014, 0x02040011, 0x000C0016, 0x00050079  
82:                                0x0302004F, 0x030E002E, 0x02030056, 0x0108007A  
83:                                0x0105006F, 0x0211007C, 0x00080077, 0x03080080  
84:                                0x0102007A, 0x01030082, 0x000200C1, 0x00080083  
85:                                0x00060085, 0x03030068, 0x00060088, 0x010D0086  
86:                                0x02090089, 0x0309008B, 0x0009008E, 0x01070090  
87:                                0x000C0092, 0x000C0095, 0x020A009B, 0x03070097  
88:                                0x01090099, 0x0006009E, 0x020E007C, 0x020C009F  
89:                                0x020900A2, 0x030800A4, 0x00020074, 0x011200BB  
90:                                0x03050072, 0x030E00A6, 0x01060029, 0x01060067  
91:                                0x02040056, 0x02040043, 0x02040052, 0x00020063  
92:                                0x0007002D, 0x02050071, 0x03090044, 0x030D0044  
93:                                0x01050070, 0x010500AA, 0x0005006E, 0x0005006D  
94:                                0x00000000, 0x01090032, 0x000C0035, 0x0008004D  
95:                                0x02070034, 0x00070038, 0x00060038, 0x030C0039  
96:                                0x01060048, 0x0306003C, 0x020B00AB, 0x00050055  
97:                                0x0003002A, 0x01040066, 0x0305004F, 0x00020060  
98:                                0x020A0074, 0x0106005D, 0x0006004B, 0x00030048  
99:                                0x0105004A, 0x03040049, 0x020A004C, 0x00070060  
100:                               0x030C004C, 0x00090060, 0x03070040, 0x020B0063  
101:                               0x030B004F, 0x02040042, 0x020C0053, 0x0303007B  
102:                               0x02020050, 0x01020062, 0x020D0057, 0x0304005A  
103:                               0x0007005D, 0x0303005E, 0x0105005F, 0x01050062  
104:                               0x02050063, 0x0107005B, 0x010E00AE, 0x00060055
```

src/cgview/xsymchar.c

```
105:                0x020E0069, 0x030200B1, 0x00030068, 0x010E003E
106:                0x0309004C, 0x00050019, 0x0109001A, 0x010D0022
107:                0x0204001C, 0x020A001D, 0x000C001E, 0x00050021
108:                0x01110022, 0x020B0026, 0x020B0053, 0x010B00B2
109:                0x001000B5, 0x0004002C, 0x0305002A, 0x000900B9
};
110:
111: static int itz[768] = { 0x22, 0x24, 0x04, 0x00, 0x40, 0x44, 0x24, 0x22,
112:                        0x24, 0x14, 0x03, 0x01, 0x10, 0x30, 0x41, 0x43,
113:                        0x34, 0x24, 0x22, 0x24, 0x02, 0x20, 0x42, 0x24,
114:                        0x22, 0x02, 0x24, 0x20, 0x24, 0x42, 0x22, 0x04,
115:                        0x44, 0x22, 0x00, 0x22, 0x40, 0x22, 0x12, 0x32,
116:                        0x22, 0x44, 0x04, 0x44, 0x00, 0x40, 0x00, 0x22,
117:                        0x04, 0x22, 0x44, 0x22, 0x20, 0x22, 0x44, 0x33,
118:                        0x13, 0x04, 0x13, 0x11, 0x00, 0x11, 0x31, 0x40,
119:                        0x31, 0x33, 0x22, 0x24, 0x20, 0x22, 0x02, 0x42,
120:                        0x22, 0x04, 0x40, 0x22, 0x00, 0x44, 0x22, 0x04,
121:                        0x44, 0x00, 0x40, 0x22, 0x22, 0x24, 0x01, 0x41,
122:                        0x24, 0x22, 0x23, 0x03, 0x20, 0x43, 0x23, 0x22,
123:                        0x10, 0x30, 0x20, 0x27, 0x16, 0x05, 0x06, 0x17,
124:                        0x37, 0x46, 0x45, 0x41, 0x00, 0x40, 0x30, 0x37,
125:                        0x02, 0x42, 0x47, 0x07, 0x04, 0x34, 0x43, 0x41,
126:                        0x30, 0x10, 0x01, 0x02, 0x06, 0x17, 0x37, 0x46,
127:                        0x06, 0x07, 0x47, 0x46, 0x20, 0x06, 0x17, 0x37,
128:                        0x46, 0x45, 0x34, 0x14, 0x34, 0x43, 0x41, 0x30,
129:                        0x10, 0x01, 0x03, 0x14, 0x05, 0x06, 0x01, 0x10,
130:                        0x30, 0x41, 0x46, 0x37, 0x17, 0x06, 0x04, 0x13,
131:                        0x43, 0x01, 0x41, 0x70, 0x42, 0x04, 0x46, 0x04,
132:                        0x44, 0x70, 0x41, 0x01, 0x37, 0x35, 0x27, 0x37,
133:                        0x06, 0x17, 0x26, 0x20, 0x26, 0x37, 0x46, 0x03,
134:                        0x13, 0x12, 0x14, 0x13, 0x24, 0x22, 0x24, 0x23,
135:                        0x33, 0x32, 0x34, 0x33, 0x43, 0x00, 0x03, 0x43,
136:                        0x03, 0x06, 0x17, 0x37, 0x46, 0x40, 0x37, 0x46,
137:                        0x41, 0x30, 0x00, 0x07, 0x37, 0x46, 0x45, 0x34,
138:                        0x04, 0x34, 0x43, 0x41, 0x47, 0x07, 0x04, 0x34,
139:                        0x04, 0x00, 0x40, 0x53, 0x33, 0x43, 0x41, 0x30,
140:                        0x10, 0x01, 0x06, 0x17, 0x37, 0x46, 0x45, 0x13,
141:                        0x30, 0x20, 0x27, 0x17, 0x37, 0x05, 0x06, 0x17,
142:                        0x37, 0x46, 0x45, 0x34, 0x24, 0x22, 0x70, 0x10,
143:                        0x30, 0x21, 0x10, 0x70, 0x22, 0x27, 0x20, 0x31,
144:                        0x36, 0x27, 0x30, 0x10, 0x17, 0x37, 0x10, 0x20,
145:                        0x21, 0x11, 0x31, 0x21, 0x26, 0x25, 0x15, 0x35,
146:                        0x25, 0x23, 0x13, 0x33, 0x40, 0x00, 0x07, 0x04,
147:                        0x44, 0x47, 0x40, 0x00, 0x07, 0x40, 0x47, 0x24,
148:                        0x07, 0x00, 0x03, 0x47, 0x25, 0x40, 0x35, 0x46,
149:                        0x41, 0x30, 0x10, 0x01, 0x06, 0x17, 0x37, 0x46,
150:                        0x70, 0x22, 0x40, 0x21, 0x25, 0x23, 0x03, 0x43,
151:                        0x23, 0x01, 0x45, 0x23, 0x05, 0x41, 0x17, 0x37,
152:                        0x30, 0x10, 0x25, 0x24, 0x34, 0x35, 0x25, 0x70,
153:                        0x31, 0x21, 0x22, 0x32, 0x31, 0x20, 0x01, 0x25,
154:                        0x41, 0x01, 0x02, 0x01, 0x10, 0x30, 0x41, 0x43,
155:                        0x34, 0x14, 0x05, 0x06, 0x17, 0x37, 0x46, 0x20,
156:                        0x27, 0x07, 0x47, 0x00, 0x40, 0x70, 0x14, 0x34,
157:                        0x07, 0x02, 0x01, 0x10, 0x30, 0x41, 0x47, 0x07,
158:                        0x20, 0x47, 0x40, 0x24, 0x00, 0x07, 0x37, 0x46,
159:                        0x45, 0x34, 0x04, 0x24, 0x40, 0x00, 0x47, 0x70,
160:                        0x07, 0x40, 0x07, 0x24, 0x20, 0x24, 0x47, 0x70,
161:                        0x33, 0x13, 0x70, 0x11, 0x31, 0x20, 0x11, 0x16,
162:                        0x27, 0x01, 0x41, 0x70, 0x02, 0x44, 0x06, 0x05,
163:                        0x21, 0x45, 0x40, 0x30, 0x31, 0x41, 0x40, 0x70,
164:
165:                0x00, 0x47, 0x70, 0x06, 0x07, 0x17, 0x16, 0x06,
166:                0x27, 0x20, 0x11, 0x31, 0x20, 0x27, 0x16, 0x36,
167:                0x27, 0x03, 0x43, 0x34, 0x32, 0x43, 0x03, 0x14,
168:                0x12, 0x03, 0x01, 0x12, 0x20, 0x27, 0x47, 0x50,
169:                0x00, 0x24, 0x07, 0x57, 0x27, 0x25, 0x40, 0x15,
170:                0x16, 0x27, 0x36, 0x03, 0x02, 0x11, 0x21, 0x42,
171:                0x04, 0x44, 0x70, 0x41, 0x01, 0x70, 0x10, 0x35,
172:                0x13, 0x23, 0x24, 0x14, 0x13, 0x00, 0x40, 0x70,
173:                0x46, 0x06, 0x70, 0x03, 0x43, 0x42, 0x52, 0x42,
174:                0x33, 0x22, 0x12, 0x03, 0x04, 0x15, 0x25, 0x34,
175:                0x33, 0x34, 0x45, 0x55, 0x64, 0x63, 0x52, 0x01,
176:                0x41, 0x70, 0x22, 0x26, 0x24, 0x44, 0x04, 0x05,
177:                0x01, 0x10, 0x20, 0x31, 0x36, 0x47, 0x57, 0x66,
178:                0x02, 0x32, 0x43, 0x44, 0x35, 0x05, 0x16, 0x26,
179:                0x44, 0x54, 0x65, 0x70, 0x63, 0x52, 0x42, 0x24,
180:                0x14, 0x03, 0x00, 0x10, 0x21, 0x22, 0x33, 0x24,
181:                0x25, 0x16, 0x06, 0x46, 0x36, 0x25, 0x24, 0x13,
182:                0x22, 0x21, 0x30, 0x40, 0x00, 0x15, 0x12, 0x21,
183:                0x31, 0x42, 0x45, 0x42, 0x51, 0x11, 0x15, 0x05,
184:                0x45, 0x35, 0x31, 0x42, 0x21, 0x11, 0x02, 0x04,
185:                0x15, 0x35, 0x44, 0x42, 0x31, 0x21, 0x20, 0x26,
186:                0x01, 0x05, 0x16, 0x26, 0x35, 0x33, 0x03, 0x33,
187:                0x31, 0x20, 0x10, 0x01, 0x45, 0x70, 0x41, 0x31,
188:                0x15, 0x05, 0x02, 0x11, 0x22, 0x25, 0x22, 0x31,
189:                0x42, 0x45, 0x04, 0x14, 0x12, 0x21, 0x32, 0x34,
190:                0x44, 0x70, 0x26, 0x20, 0x00, 0x23, 0x41, 0x23,
191:                0x15, 0x06, 0x24, 0x13, 0x12, 0x21, 0x31, 0x42,
192:                0x43, 0x34, 0x24, 0x15, 0x26, 0x36, 0x41, 0x21,
193:                0x12, 0x14, 0x25, 0x45, 0x70, 0x43, 0x03, 0x04,
194:                0x15, 0x24, 0x12, 0x24, 0x35, 0x44, 0x30, 0x21,
195:                0x31, 0x32, 0x22, 0x21, 0x70, 0x03, 0x53, 0x70,
196:                0x24, 0x34, 0x35, 0x25, 0x24, 0x01, 0x45, 0x70,
197:                0x05, 0x41, 0x42, 0x31, 0x11, 0x02, 0x04, 0x15,
198:                0x35, 0x44, 0x70, 0x26, 0x20, 0x02, 0x11, 0x31,
199:                0x42, 0x43, 0x34, 0x14, 0x05, 0x16, 0x36, 0x45,
200:                0x70, 0x27, 0x20, 0x03, 0x63, 0x11, 0x15, 0x14,
201:                0x04, 0x44, 0x34, 0x35, 0x31, 0x32, 0x42, 0x02,
202:                0x42, 0x31, 0x11, 0x02, 0x05, 0x16, 0x36, 0x45,
203:                0x43, 0x32, 0x22, 0x13, 0x14, 0x25, 0x35, 0x44,
204:                0x37, 0x35, 0x27, 0x37, 0x70, 0x17, 0x15, 0x07,
205:                0x17, 0x00, 0x10, 0x36, 0x46, 0x70, 0x44, 0x35,
206:                0x15, 0x04, 0x13, 0x33, 0x42, 0x31, 0x11, 0x02,
207:                0x70, 0x12, 0x34, 0x00, 0x00, 0x00, 0x00, 0x00 };
208:
209: for ( i = 0 ; i < 256 ; i++ ){
210:     ibcd[i] = '\0';
211: }
212:
213: ibit8 = 256;
214: ibit16 = 65536;
215: ibit24 = 16777216;
216:
217: h = *hh;
218: h = (float)sqrt( pow( (double)h, 2.0 ) );
219: ichar = 0;
220: ipen = 3;
221: nt = *nchar;
222: icb = 0;
223: div = 7.0;
224:
225: if( nt == 0 ) {
226:     icb = 3;
227: }
228:
229: if( nt < 0 ) {
```

src/cgview/xsymchar.c

```

228:         if( nt != -1 ) {
229:             ipen = 2;
230:         }
231:         icb = 3;
232:         ichar = *ibcda + 1;
233:
234:         if( ichar <= 15 ) {
235:             div = 4.0;
236:         }
237:     }
238:
239:     fct = h / div;
240:     th = *angle;
241:
242:     if( th != oldth ) {
243:         oldth = th;
244:         thl = th*rad;
245:         sinth = sin( thl );
246:         costh = cos( thl );
247:         oldfct = fct;
248:         xa[0] = 0.0;
249:         ya[0] = 0.0;
250:         xa[1] = fct*costh;
251:         ya[1] = fct*sinth;
252:         for ( i = 2 ; i < 8 ; i++ ) {
253:             xa[i] = xa[i-1] + xa[1];
254:             ya[i] = ya[i-1] + ya[1];
255:         }
256:     }
257:     if( fct != oldfct ) {
258:         oldfct = fct;
259:         xa[0] = 0.0;
260:         ya[0] = 0.0;
261:         xa[1] = fct*costh;
262:         ya[1] = fct*sinth;
263:         for ( i = 2 ; i < 8 ; i++ ) {
264:             xa[i] = xa[i-1] + xa[1];
265:             ya[i] = ya[i-1] + ya[1];
266:         }
267:     }
268:
269:     xx = *x;
270:     yy = *y;
271:     if( (float)sqrt( pow( (double)( xx - 999.0 ), 2.0 ) ) > eps ) x0 = xx;
272:     if( (float)sqrt( pow( (double)( yy - 999.0 ), 2.0 ) ) > eps ) y0 = yy;
273:     xx = x0;
274:     yy = y0;
275:
276:     /* VERTICAL DIRECTION CHECK. */
277:     if( nt >= 0 ) {
278:         if( *hh < 0 ) {
279:             x0 = x0 + h*sinth;
280:             y0 = y0 - h*costh;
281:             xa[8] = ( 1.0 + DDV )*h*sinth;
282:             ya[8] = -( 1.0 + DDV )*h*costh;
283:             iswv = 1;
284:         }
285:     }
286:
287:     if( ( nt < 0 ) && ( ichar <= 15 ) ) {
288:         x0 = x0 - ( xa[2] - ya[2] );
289:         y0 = y0 - ( xa[2] + ya[2] );
290:     }
291:
292:     nnt = abs( nt );

```

```

293:
294:     if ( icb != 3 ) {
295:         strcpy(ibcd, (char *)ibcda);
296:         ibcd[*nchar] = '\0';
297:     }
298:
299:     for ( kk = 0 ; kk < nnt ; kk++ ) {
300:
301:         if ( icb == 3 ) {
302:             ichar = *ibcda;
303:         }
304:         else {
305:             astoeb( &ibcd[kk], &iebc );
306:             ichar = iebc%128;
307:         }
308:         idata = idxtab[ichar];
309:
310:         if( idata != 0 ) {
311:             kpos = idata / ibit24;
312:             knum = ( idata / ibit16 ) % ibit8;
313:             ktab = idata % ibit16;
314:             kpt = ( ktab - 1 )*4 + kpos;
315:             for ( m = kpt ; m < kpt+knun ; m++ ) {
316:                 nx = itz[m] / 16;
317:                 ny = itz[m] % 16;
318:
319:                 if( nx == 7 ) {
320:                     ipen = 3;
321:                 }
322:                 if( nx != 7 ) {
323:                     xt = x0 + xa[nx] - ya[ny];
324:                     yt = y0 + ya[nx] + xa[ny];
325:                     xplot( &xt, &yt, &ipen );
326:                     ipen = 2;
327:                 }
328:             }
329:         }
330:
331:         x0 = x0 + xa[7];
332:         y0 = y0 + ya[7];
333:         ipen = 3 ;
334:     }
335:     /*
336:     *
337:     * END OF SYMB4
338:     *
339:     */
340:     if( *hh < 0.0 ) {
341:         x0 = x0 - h*sinth;
342:         y0 = y0 + h*costh;
343:     }
344:
345:     if( *nchar < 0 ) {
346:         x0 = xx;
347:         y0 = yy;
348:     }
349:
350:     ipen = 3;
351:     xplot(&x0, &y0, &ipen);
352: }
353:
354:
355:
356:
357:

```


src/cgview/xsymchar.c

358:
359:
360:
361:



src/cgview/xyplot.f

```

1: CC/#IF .NOT.PIFFLIB
2: *      SUBROUTINE ROTATE(THETA)
3: C
4: *      COMMON /XYWORK/  XLR,YLR,XPN,YPN,XOR,YOR,XW,YW,XWOR,YWOR,IPN,FACO,
5: *      *      XDOT,YDOT,XDOT1,YDOT1,RSYS,XSLR,YSLR,SIR,COR,
6: *      *      IMETA,IPLOT,VXL,VYL,VXU,VYU,RSYSP,
7: *      *      AXSTEP,H1,H2,H3,D1,D2,D3,
8: *      *      HMRK
9: C
10: *      TH=THETA*0.0174533
11: *      SIR=SIN(TH)
12: *      COR=COS(TH)
13: C
14: *      RETURN
15: *      END
16: C
17: *      SUBROUTINE SYMBO2(XP,YP,HEIGHT,IBC,THETA,N,FAI,SPACE)
18: C
19: *      INTEGER*4      IPAT
20: *      INTEGER*4      IADSYM,IDAT
21: *      CHARACTER*127  IBC
22: *      CHARACTER*1    IBC1(127),C1
23: C      CHARACTER*1    IBC1(127)
24: *      DIMENSION      IBCI(127)
25: *      DIMENSION      IPAT(3400)
26: C
27: *      COMMON /XYWORK/  XLR,YLR,XPN,YPN,XOR,YOR,XW,YW,XWOR,YWOR,IPN,FACO,
28: *      *      XDOT,YDOT,XDOT1,YDOT1,RSYS,XSLR,YSLR,SIR,COR,
29: *      *      IMETA,IPLOT,VXL,VYL,VXU,VYU,RSYSP,
30: *      *      AXSTEP,H1,H2,H3,D1,D2,D3,
31: *      *      HMRK
32: C
33: *      DATA IC/0/,NKANF/3386/
34: C
35: *      INCLUDE 'INC/SYMBOL'
36: C
37: *      IF(N .EQ. 0) RETURN
38: C
39: *      NABS=IABS(N)
40: C
41: *      IC=IC+1
42: *      IF(IC .EQ. 1) THEN
43: *      call DTIME
44: *      ENDIF
45: C
46: *      IF(N .GT. 0) THEN
47: *      READ(IBC,'(127A1)') (IBC1(I),I=1,NABS)
48: *      DO 1100 I=1,NABS
49: *1100   IBCI(I)=IACHAR(IBC1(I))
50: *      ELSE
51: *      READ(IBC,'(127A1)') (IBC1(I),I=1,2*NABS)
52: C      READ(IBC,'(127A1)') (IBC1(I),I=1,NABS)
53: *      DO 1200 I=1,NABS
54: *1200   IBCI(I)=IACHAR(IBC1(2*I-1))
55: *      ENDIF
56: C
57: *      XPS=XP
58: *      YPS=YP
59: *      XPP=XP
60: *      YPP=YP
61: *      DD=12
62: *      IF(XP .EQ. 999.0) XPP=XSLR
63: *      IF(YP .EQ. 999.0) YPP=YSLR
64: *      HRSYM=ABS(HEIGHT)/DD
65: *      IF(N .LT. 0) THEN

```

```

66: *      IXS=-4
67: *      IYS=-6
68: *      ELSE
69: *      IXS=0
70: *      IYS=0
71: *      ENDIF
72: C
73: *      SI=SIN(THETA/180.0*3.141592)
74: *      CO=COS(THETA/180.0*3.141592)
75: *      FAID=FAI
76: *      IF(ABS(FAI) .GT. 60.0) FAID=0.0
77: *      TA=TAN(FAID /180.0*3.141592)
78: C
79: *      DO 2000 I=1,NABS
80: *      XMG=FLOAT(I-1)*(DD+SPACE)
81: *      IXSD=IXS
82: *      IF(IBC1(I) .LT. 16) IXSD=IXS-3
83: *      IADSYM=IPAT(IBC1(I)*2+1)*256+IPAT(IBC1(I)*2+1+1)
84: *      INPSYM=IPAT(IADSYM+1)
85: *      DO 2100 J=1,INPSYM
86: *      IDAT=IPAT(IADSYM+J*2+1)
87: *      IPEN=IPAT(IADSYM+J*2 )+2
88: *      YSYM=MOD(IDAT,16)-2 +IYS
89: *      XSYM= IDAT/16 -1 +IXSD
90: *      XSYM=XSYM + TA*YSYM
91: *      XPS=((XSYM+XMG)*CO-YSYM*SI)*HRSYM+XPP
92: *      YPS=((XSYM+XMG)*SI+YSYM*CO)*HRSYM+YPP
93: *2100   CALL PLOT(XPS,YPS,IPEN)
94: *2000   CONTINUE
95: C
96: *      XSLR=((DD+XMG)*CO)*HRSYM+XPP
97: *      YSLR=((DD+XMG)*SI)*HRSYM+YPP
98: C
99: *8000   RETURN
100: *      END
101: C
102: *      SUBROUTINE SYMBOL(XP,YP,HEIGHT,IBC,THETA,N)
103: C
104: *      CHARACTER*127 IBC
105: C
106: *      FAI=0
107: *      SPACE=0
108: *      CALL SYMBO2(XP,YP,HEIGHT,IBC,THETA,N,FAI,SPACE)
109: C
110: *      RETURN
111: *      END
112: C
113: *      SUBROUTINE NUMB2(XP,YP,HEIGHT,FN,THETA,N,FAI,SPACE)
114: C
115: *      CHARACTER C*127,C1(19)
116: C
117: *      COMMON /XYWORK/  XLR,YLR,XPN,YPN,XOR,YOR,XW,YW,XWOR,YWOR,IPN,FACO,
118: *      *      XDOT,YDOT,XDOT1,YDOT1,RSYS,XSLR,YSLR,SIR,COR,
119: *      *      IMETA,IPLOT,VXL,VYL,VXU,VYU,RSYSP,
120: *      *      AXSTEP,H1,H2,H3,D1,D2,D3,
121: *      *      HMRK
122: C
123: *      XPP=XP
124: *      YPP=YP
125: *      HH =HEIGHT
126: *      NNS=N
127: *      IF(XPP .EQ. 999.0) XPP=XSLR
128: *      IF(YPP .EQ. 999.0) YPP=YSLR
129: C
130: *      WRITE(C,'(F19.8)') FN

```

src/cgview/xyplot.f

```

131: *      READ(C,'(19A1)')      (C1(I),I=1,19)
132: C
133: *      NEF=0
134: *      DO 1000 I=1,19
135: *      IF(NEF.EQ. 0 .AND. C1(I) .NE. ' ') NEF=I
136: *      IF(C1(I) .EQ. ' ') NPI=I
137: *1000 CONTINUE
138: C
139: *      IF(NEF .EQ. NPI) THEN
140: *          NEF=NEF-1
141: *          C1(NEF)='0'
142: *      ENDIF
143: C
144: *      NNS=(NPI-NEF+1)+NNS
145: *      IF(NNS.LT. 1 .AND. FN.EQ. 0.0) NNS=1
146: *      IF(NNS.LE. 0) THEN
147: *          NNS=IABS(NNS)+2
148: *          NEF=NEF-(NNS-1)
149: *      ENDIF
150: C
151: *      WRITE(C,'(19A1)') (C1(I),I=NEF,NEF+NNS-1)
152: C
153: *      IF(HEIGHT .GT. 0.0) THEN
154: *          CALL SYMBO2(XPP,YPP,HEIGHT,C,THETA,NNS,FAI,SPACE)
155: *      ELSE
156: *          XSTEP=(FLOAT(NPI-NEF))*HEIGHT
157: *          IF(XSTEP.GT. 0.0) XSTEP=0
158: *          CNUM=cos(3.141592*THETA/180.0)
159: *          SNUM=sin(3.141592*THETA/180.0)
160: *          XPP=CNUM*XSTEP+XPP
161: *          YPP=SNUM*XSTEP+YPP
162: *          HH=-HEIGHT
163: *          CALL SYMBO2(XPP,YPP,HH,C,THETA,NNS,FAI,SPACE)
164: *      ENDIF
165: C
166: *      RETURN
167: *      END
168: C
169: *      SUBROUTINE NUMBER(XP,YP,HEIGHT,FN,THETA,N)
170: C
171: *      FAI=0
172: *      SPACE=0
173: *      CALL NUMB2(XP,YP,HEIGHT,FN,THETA,N,FAI,SPACE)
174: C
175: *      RETURN
176: *      END
177: CC/#ENDIF
178: C
179: C
180: C-----
181: C
182: *      subroutine FRAME( XL,      YL,      XU,      YU )
183: C
184: C/#IF MS_VISUAL
185: C
186: C ... This interface block is for CUTIL & MS-Visual tools. ...
187: C
188: *      interface
189: *          subroutine PLOT( RRR1, RRR2, III1 )
190: *              integer      III1
191: *              real         RRR1, RRR2
192: *CDEC$      ATTRIBUTES C :: PLOT
193: *CDEC$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
194: *          end subroutine
195: *      end interface

```

```

196: C/#ENDIF
197: C
198: *      call PLOT( XL, YL, 3 )
199: *      call PLOT( XU, YL, 2 )
200: *      call PLOT( XU, YU, 2 )
201: *      call PLOT( XL, YU, 2 )
202: *      call PLOT( XL, YL, 2 )
203: C
204: *      return
205: *      end
206: C
207: C-----
208: C
209: *      subroutine SCALE( DAT,      SIZE, N,      K )
210: C
211: *      common /XYWORK/ XLR,      YLR,      XPN,      YPN,      XOR,      YOR,
212: *      &      XW,      YW,      XWOR,      YWOR,      IPN,      FAC,      XDOT,
213: *      &      YDOT,      XDOT1,      YDOT1,      RSYS,      XSLR,      YSLR,      SIR,
214: *      &      COR,      IMETA,      IPLOT,      VXL,      VYL,      VXU,      VYU,
215: *      &      RSYSP,      AXSTEP, H1,      H2,      H3,      D1,      D2,
216: *      &      D3,      HMRK
217: C
218: *      integer IKK
219: *      dimension DAT(1)
220: *      dimension FL(7)
221: *      data FL /8.0, 1.0, 2.0, 4.0, 5.0, 8.0, 1.0/
222: C
223: *      if ( SIZE.le.1.0 ) return
224: *      if ( N.eq.0 ) return
225: *      if ( K.eq.0 ) K = 1
226: *      NK = ABS(N*K)
227: C
228: *      DMIN = 1.E35
229: *      DMAX = 1.E-35
230: C
231: *      IKK = ABS(K)
232: C
233: *      do 100 I = 1, NK, ABS(K)
234: *          if ( DAT(I).lt.DMIN ) DMIN = DAT(I)
235: *          if ( DAT(I).gt.DMAX ) DMAX = DAT(I)
236: *      100 continue
237: C
238: *      DDEL = (DMAX-DMIN) /SIZE*AXSTEP
239: *      SDEL = SIGN(1.0,DDEL)
240: C
241: *      NT = INT(ALOG10(ABS(DDEL)))
242: *      E = ABS(DDEL) - REAL(NT)*10.0
243: C
244: *      do 110 I = 2, 6
245: *          if ( E.lt.FL(I) ) go to 120
246: *      110 continue
247: C
248: *      I = 7
249: C
250: *      120 NTG = NT
251: *      NTL = NT
252: *      EG = FL(I)
253: *      EL = FL(I-1)
254: *      if ( I.eq.7 ) NTG = NT + 1
255: *      if ( I.eq.2 ) NTL = NT - 1
256: *      DDEL = EG*10**NTG
257: *      DDEL = DDEL/AXSTEP
258: *      DDEL = SDEL*DDEL
259: *      DMIN = SIGN(((ABS(DMIN)/DDEL))*DDEL,DMIN)
260: *      DMAX = DMIN + SIZE*DDEL

```

src/cgview/xyplot.f

```

261: C
262:   if ( N.gt.0 ) then
263:     if ( K.gt.0 ) then
264:       DAT(NK+1) = DMIN
265:       DAT(NK+ABS(K)+1) = DDEL
266:       IKK = ABS(K)
267:     else
268:       DAT(NK+1) = DMAX
269:       DAT(NK+ABS(K)+1) = -DDEL
270:       IKK = ABS(K)
271:     end if
272:   else if ( K.gt.0 ) then
273:     DAT(NK+1) = 0.0
274:     DAT(NK+ABS(K)+1) = DDEL
275:     IKK = ABS(K)
276:   else
277:     DAT(NK+1) = 0.0
278:     DAT(NK+ABS(K)+1) = -DDEL
279:     IKK = ABS(K)
280:   end if
281:
282:   return
283: end
284: C
285: C-----
286: C
287: C === AXIS routine : numbers are drawn parallel to axis
288: C
289:   subroutine AXIS2( XP, YP, LABEL, N, SIZE, THETA, DMIN,
290: & DDEL, NT )
291:   character*(*) LABEL
292: C
293:   JNDIR = 0
294:   call AXISX( JNDIR, XP,YP, LABEL, N, SIZE, THETA, DMIN,
295: & DDEL, NT )
296:   return
297: end
298: C
299: C-----
300: C
301: C === AXIS routine : numbers are drawn vertical to axis
302: C (drawing label is not recommended !!!)
303: C
304:   subroutine AXIS3( XP, YP, LABEL, N, SIZE, THETA, DMIN,
305: & DDEL, NT )
306:   character*(*) LABEL
307: C
308:   JNDIR = 1
309:   call AXISX( JNDIR, XP,YP, LABEL, N, SIZE, THETA, DMIN,
310: & DDEL, NT )
311:   return
312: end
313: C
314: C-----
315: C
316:   subroutine AXISX( JNDIR, XP,YP, LABEL, N, SIZE, THETA, DMIN,
317: & DDEL, NT )
318: C
319: C .....
320: C JNDIR = 0 : numbers are drawn parallel to axis
321: C JNDIR = 1 : numbers are drawn vertical to axis
322: C .....
323: C
324: C/#IF MS_VISUAL
325: C

```

```

326: C ... This interface block is for CUTIL & MS-Visual tools. ...
327: C
328: *   interface
329: *     subroutine PLOT( RRR1, RRR2, III1 )
330: *       integer III1
331: *       real RRR1, RRR2
332: *CDEC$ ATTRIBUTES C :: PLOT
333: *CDEC$ ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
334: *     end subroutine
335: *   end interface
336: C/#ENDIF
337: C
338:   character*(*) LABEL
339: C
340:   common /XYWORK/ XLR, YLR, XPN, YPN, XOR, YOR,
341: & XW, YW, XWOR, YWOR, IPN, FAC, XDOT,
342: & YDOT, XDOT1, YDOT1, RSYS, XSLR, YSLR, SIR,
343: & COR, IMETA, IPLOT, VKL, VYL, VXU, VYU,
344: & RSYSP, AXSTEP, H1, H2, H3, D1, D2,
345: & D3, HMRK
346: C
347:   real NN, NA
348: C
349: C-----
350: C
351:   TH = THETA*.0174533
352:   CO = COS(TH)
353:   SI = SIN(TH)
354:   SABS = ABS(SIZE)
355:   SIC = SABS*CO
356:   SIS = SABS*SI
357:   AXC = AXSTEP*CO
358:   AXS = AXSTEP*SI
359: C
360:   NN = SABS/AXSTEP + 1.0
361:   NA = NN/2.0
362: C
363:   if ( N.ge.0 ) then
364:     KN = N
365:     A = 1
366:   else
367:     KN = -N
368:     A = -1
369:   end if
370: C
371:   FDD = AXSTEP*DDEL
372:   D11 = D3*A
373: C
374:   if ( JNDIR.eq.1 ) then
375:     D22 = 0.0
376:     D21 = (H2*1.0)*A
377:     XN = XP + D22*CO - D21*SI
378:     YN = YP + D21*CO + D22*SI
379:   else
380:     D22 = -H2*.5
381:     D21 = D1*A - H2*.5
382:     XN = XP + D22*CO - D21*SI - 0.1*AXSTEP*CO
383:     YN = YP + D21*CO + D22*SI - 0.1*AXSTEP*SI
384:   end if
385: C
386:   D31 = D2*A - H3*.5
387:   D32 = D2*A + H3*.5
388: C
389:   AF = 1.
390:   N2 = NT

```

src/cgview/xyplot.f

```

391: C
392:     FS      = ABS(DMIN)
393:     if ( FS.le.0.1 ) then
394:         FE      = ABS(DMIN+FDD*(NN-1))
395:         if ( FE.le.0.1 ) then
396:             FA      = FS
397:             if ( FS.lt.FE ) FA = FE
398:             if ( FA.ne.0.0 ) then
399:                 N1      = 1 - INT(LOG10(FA))
400:                 AF      = 10.**N1
401:                 N2      = NT - N1
402:                 if ( N2.le.0 ) N2 = N2 - 1
403:                 BN      = -N1
404:                 go to 100
405:             end if
406:         end if
407:     end if
408: C
409:     if ( NT.lt.-9 ) then
410:         AF      = 10.***(NT+1)
411:         N2      = -1
412:     end if
413: Check ...
414: C     write(*,*) 'axis2: NT N2 ',NT,N2
415: C
416:     100 NDEL      = 0
417:     XVAL      = DMIN*AF
418:
419:     do 130 I = 1, NN
420: C
421:         if ( MOD(NDEL,2).eq.0 ) then
422:             if ( JNDIR.eq.0 ) then
423:                 call JNUMBER( XN, YN, H2, AXSTEP*REAL(NDEL), THETA, N2,
424:                     &          0, 0 )
425:             else
426:                 if ( A.gt.0.0 ) then
427:                     call JNUMBER( XN,YN,H2,AXSTEP*REAL(NDEL), THETA-90.0,
428:                         &          N2,
429:                         &          2, 1 )
430:                 else
431:                     call JNUMBER( XN,YN,H2,AXSTEP*REAL(NDEL), THETA-90.0,
432:                         &          N2,
433:                         &          0, 1 )
434:                 end if
435:             end if
436:         end if
437:         NDEL      = NDEL + 1
438: C
439:         XVAL      = XVAL + FDD*AF
440:         XN        = XN + AXC
441:         YN        = YN + AXS
442: C
443:         if ( NA.eq.0 ) then
444:             if ( H1.ne.0. ) then
445:                 if ( AF.le.1. ) then
446:                     if ( NT.ge.-1 ) then
447:                         if ( N.ne.0 ) go to 110
448:                         go to 120
449:                     end if
450:                 end if
451:             end if
452:             if ( NT.lt.-1 ) BN = -REAL(NT) - 1.
453: C
454:             AN      = REAL(KN+6)*.5
455:             XD3      = -D31*SI - AN*H3*CO

```

```

456:             XP1      = XP + SIC*.5 + XD3
457:             YP1      = YP + SIS*.5 + YD3
458:             call JSYMBOL( XP1, YP1, H3, LABEL, THETA, KN )
459:             call JSYMBOL( 999.0, 999.0, H3, ' *10', THETA, 6 )
460: C
461:             if ( BN.ne.1. ) then
462:                 XD4      = D32*SI - AN*H3*CO
463:                 YD4      = -D32*CO - AN*H3*SI
464:                 XP2      = XP + SIC*.5 - XD4
465:                 YP2      = YP + SIS*.5 - YD4
466:                 call JNUMBER( XP2, YP2, H1, BN, THETA, -1, 0, 0 )
467:             end if
468:             go to 120
469:         end if
470: C
471:     110     AN      = REAL(KN)*.5
472:             XD3      = -D31*SI - AN*H3*CO
473:             YD3      = D31*CO - AN*H3*SI
474:             XP1      = XP + SIC*.5 + XD3
475:             YP1      = YP + SIS*.5 + YD3
476:             call JSYMBOL( XP1, YP1, H3, LABEL, THETA, KN )
477:         end if
478:     120     NA      = NA - 1
479:     130 continue
480: C
481:     XA      = XP + SIC
482:     YA      = YP + SIS
483:     call PLOT( XA, YA, 3 )
484: C
485:     A      = NN - 1.
486:     XN      = XP + A*AXC
487:     YN      = YP + A*AXS
488:     XD1      = -D11*SI
489:     YD1      = D11*CO
490: C
491:     D1SGN      = 1.
492:     if ( SIZE.lt.0. ) D1SGN = -1.
493:
494:     if ( SIZE.lt.0.0 ) then
495:         D1SGN      = -1
496:     else
497:         D1SGN      = 1
498:     end if
499:
500:     do 140 I = 1, NN
501:         call PLOT( XN, YN, 2 )
502:         call PLOT( XN+XD1*D1SGN, YN+YD1*D1SGN, 2 )
503:         call PLOT( XN, YN, 2 )
504:         XN      = XN - AXC
505:         YN      = YN - AXS
506:     140 continue
507: C
508:     return
509: end
510: C
511: C-----
512: C
513:     subroutine AXIS( XP,      YP,      LABEL, N,      SIZE, THETA, DMIN,
514:         &          DDEL )
515: C
516:     character*127 LABEL
517: C
518:     NT      = 2
519:     call AXIS2( XP, YP, LABEL, N, SIZE, THETA, DMIN, DDEL, NT )
520: C

```

src/cgview/xyplot.f

```

521:      return
522:      end
523: C
524: C-----
525: C
526:      subroutine AXSPRM( AXSTEP,H1,      H2,      H3,      D1,      D2,      D3 )
527: C
528:      common /XYWORK/ XLR,      YLR,      XPN,      YPN,      XOR,      YOR,
529:      &      XW,      YW,      XWOR,      YWOR,      IPN,      FAC,      XDOT,
530:      &      YDOT,      XDOT1,      YDOT1,      RSYS,      XSLR,      YSLR,      SIR,
531:      &      COR,      IMETA,      IPLOT,      VXL,      VYL,      VXU,      VYU,
532:      &      RSYSP,      AXS,      H10,      H20,      H30,      D10,      D20,
533:      &      D30,      HMRK
534:      AXS      = AXSTEP
535:      H10      = H1
536:      H20      = H2
537:      H30      = H3
538:      D10      = D1
539:      D20      = D2
540:      D30      = D3
541:      return
542:      end
543: C
544: C-----
545: C
546:      subroutine LINE( XDAT, YDAT, N,      K,      J,      L,      XMIN,
547:      &      YMIN,      XDELTA,YDELTA )
548: C
549:      integer N, K, J, L
550:      dimension XDAT(N), YDAT(N)
551: C
552:      LTYPE = 1
553:      call LINE2( XDAT, YDAT, N, K, J, L, LTYPE, XMIN, YMIN, XDELTA,
554:      &      YDELTA )
555: C
556:      return
557:      end
558: C
559: C-----
560: C
561:      subroutine LINE2( XDAT, YDAT, N,      K,      J,      L,      LTYPE,
562:      &      XMIN,      YMIN,      XDELTA, YDELTA )
563: C=====
564: C Draw lines on 2D plane:
565: C
566: C << Argument descriptionis decoded by M.Sasaki, and
567: C it may not be correct guess :-). >>
568: C
569: C XDAT((N+1)*K), YDAT((N+1)*K) : coordinate of points on line.
570: C N : N points are connected as a line
571: C K : connect every K points in points (XDAT,YDAT)
572: C J : J < 0 : move to (XDAT(1),YDAT(1)). Nothing drawn.
573: C      J = 0 : draw lines
574: C      J > 0 : draw symbol marks on each points also
575: C L : symbol mark # (??)
576: C=====
577: C/#IF MS_VISUAL
578: C
579: C ... This interface block is for CUTIL & MS-Visual tools. ...
580: C
581: C      interface
582: C      subroutine PLOT( RRR1, RRR2, III1 )
583: C      integer      III1
584: C      real          RRR1, RRR2
585: CDEC$      ATTRIBUTES C :: PLOT

```

```

586: *CDEC$      ATTRIBUTES REFERENCE :: RRR1, RRR2, III1
587: *      end subroutine
588: *      end interface
589: C/#ENDIF
590: C
591:      common /XYWORK/ XLR,      YLR,      XPN,      YPN,      XOR,      YOR,
592:      &      XW,      YW,      XWOR,      YWOR,      IPN,      FAC,      XDOT,
593:      &      YDOT,      XDOT1,      YDOT1,      RSYS,      XSLR,      YSLR,      SIR,
594:      &      COR,      IMETA,      IPLOT,      VXL,      VYL,      VXU,      VYU,
595:      &      RSYSP,      AXSTEP, H1,      H2,      H3,      D1,      D2,
596:      &      D3,      HMRK
597: C
598:      integer N, K, J, L, SYMB
599:      character*127 MIBCD
600:      character*16 WORK
601:      character      IDS*4
602:      dimension XDAT(1), YDAT(1)
603:      data SYMB /-1/
604: C
605: C-----
606: C
607:      if ( N.lt.2 ) return
608: C
609:      IABSK = ABS(K)
610:      IABSK = ABS(K)
611:      if ( IABSK.eq.0 ) IABSK = 1
612:      XDMIN = XMIN
613:      YDMIN = YMIN
614:      XDDEL = XDELTA/2.0
615:      YDDEL = YDELTA/2.0
616:      if ( XDDEL.eq.0.0 ) XDDEL = 1.0
617:      if ( YDDEL.eq.0.0 ) YDDEL = 1.0
618: C
619:      NDAT = N*IABSK
620:      KSTEP = IABSK
621: C
622:      X = (XDAT(1)-XDMIN) /XDDEL
623:      Y = (YDAT(1)-YDMIN) /YDDEL
624:      call PLOT( X, Y, 3 )
625: C
626:      if ( J.ge.0 ) then
627: C
628:      do 100 I = KSTEP + 1, NDAT, KSTEP
629:      X = (XDAT(I)-XDMIN) /XDDEL
630:      Y = (YDAT(I)-YDMIN) /YDDEL
631:      call PLOT( X, Y, 2 )
632: 100 continue
633: C
634:      if ( J.eq.0 ) return
635: C
636:      write(WORK,7020) MOD(ABS(L),128)
637:      write(MIBCD,7000) WORK(16:16)
638: 7000 format(A1)
639: 7020 format(A16)
640:      end if
641: C
642:      J1 = 1
643: C      IDS = MOD(ABS(L),16)
644:      write(IDS,7040) MOD(ABS(L),16)
645: 7040 format(A4)
646: C
647:      do 110 I = 1, NDAT, KSTEP
648:      X = (XDAT(I)-XDMIN) /XDDEL
649:      Y = (YDAT(I)-YDMIN) /YDDEL
650:      MSW = MOD(J1,J)

```

src/cgview/xyplot.f

```

651:
652:         if ( MSW.eq.0 ) call JSYMBOL( X, Y, HMRK, IDS, 0.0, SYMB )
653:
654:         J1      = J1 + 1
655:         110 continue
656:
657:         return
658:         end
659: C
660: C-----
661: C
662:       subroutine LINMRK( HMRK )
663: C
664:       common /XYWORK/  XLR,   YLR,   XPN,   YPN,   XOR,   YOR,
665:       &              XW,    YW,    XWOR,  YWOR,  IPN,   FAC,  XDOT,
666:       &              YDOT,  XDOT1, YDOT1, RSYS, XSLR, YSLR, SIR, COR,
667:       &              COR,   IMETA, IPLOT, VXL,  VYL,  VXU,  VYU,
668:       &              RSYSP, AXSTEP, H1,   H2,   H3,   D1,   D2,
669:       &              D3,    HMRKO
670:       HMRKO = HMRK
671:       return
672:       end
673: C
674: CC/#IF .NOT.PIFFLIB
675: *      SUBROUTINE VTOM(XP,YP,IXP,IYP)
676: C
677: *      REAL IXP,IYP
678: C
679: *      COMMON /XYWORK/  XLR,YLR,XPN,YPN,XOR,YOR,XW,YW,XWOR,YWOR,IPN,FAC,
680: *      *              XDOT,YDOT,XDOT1,YDOT1,RSYS,XSLR,YSLR,SIR,COR,
681: *      *              IMETA,IPLOT,VXL,VYL,VXU,VYU,RSYSP,
682: *      *              AXSTEP,H1,H2,H3,D1,D2,D3,
683: *      *              HMRK
684: *      IXP=(XDOT1*(XP-XWOR)/XW)
685: *      IYP=(YDOT1*(YP-YWOR)/YW)
686: C
687: *      RETURN
688: *      END
689: C
690: CC/#ENDIF
691: C
692: C-----
693: C
694:       subroutine XINT( PID,   STYLE, CLDAT, IKIND )
695: C
696: CD      INCLUDE '/usr/include/phigs/phigs77.h'
697: C
698:       common /XYWORK/  XLR,   YLR,   XPN,   YPN,   XOR,   YOR,
699:       &              XW,    YW,    XWOR,  YWOR,  IPN,   FAC,  XDOT,
700:       &              YDOT,  XDOT1, YDOT1, RSYS, XSLR, YSLR, SIR, COR,
701:       &              COR,   IMETA, IPLOT, VXL,  VYL,  VXU,  VYU,
702:       &              RSYSP, AXSTEP, H1,   H2,   H3,   D1,   D2,
703:       &              D3,    HMRK
704:       character PID*72
705:       common /COLOR/  CLR
706:       integer CLR(0:999)
707:       integer STYLE, CLDAT(0:999), IKIND
708:       integer CANVASID, WSID
709: C
710: CC/#IF .NOT.PIFFLIB
711: *      DO 132 IYY = 0,999
712: *          CLR(IYY) = CLDAT(IYY)
713: * 132 CONTINUE
714: CC/#ENDIF
715:       IMETA      = 0

```

```

716:       IPLOT      = 0
717:       AXSTEP     = 1.0
718:       H1         = 0.14
719:       H2         = 0.21
720:       H3         = 0.28
721:       D1         = 0.38
722:       D2         = 0.825
723:       D3         = 0.178
724:       HMRK       = 0.2
725: C
726: CC/#IF .NOT.PIFFLIB
727: *      wsid = canvasid(PID,STYLE,IKIND)
728: *      call popph(7,0)
729: *      call popwk(1,wsid,phigswstcanvas)
730: *      call popst(1)
731: *      call psici(8)
732: *      call psiasf(0,1)
733: *      call psiasf(1,1)
734: *      call psiasf(2,1)
735: *      call pslwsc(0.5)
736: C92.8.7 =====>
737: C IKIND = 1 .... PLOT TYPE IS NOMAL
738: C IKIND = 2 .... PLOT TYPE IS RESULT
739: *      IF (IKIND .EQ. 2 ) THEN
740: *          call pscr(1,  0, 0.0 , 0.0 , 0.0 )
741: *          call pscr(1,  1, 1.0 , 1.0 , 1.0 )
742: *          call pscr(1,  2, 1.0 , 0.0 , 0.0 )
743: *          call pscr(1,  3, 1.0 , 0.25, 0.0 )
744: *          call pscr(1,  4, 1.0 , 0.5 , 0.0 )
745: *          call pscr(1,  5, 1.0 , 0.75, 0.3 )
746: *          call pscr(1,  6, 1.0 , 0.8 , 0.7 )
747: *          call pscr(1,  7, 0.7 , 0.8 , 1.0 )
748: *          call pscr(1,  8, 0.3 , 0.75, 1.0 )
749: *          call pscr(1,  9, 0.0 , 0.5 , 1.0 )
750: *          call pscr(1, 10, 0.0 , 0.25, 1.0 )
751: *          call pscr(1, 11, 0.0 , 0.0 , 1.0 )
752: *          call pscr(1, 12, 0.0 , 1.0 , 0.0 )
753: *      ELSE
754: *          call pscr(1,  0, 0.0, 0.0, 0.0)
755: *          call pscr(1,  1, 1.0, 1.0, 1.0)
756: *          call pscr(1,  2, 1.0, 0.0, 0.0)
757: *          call pscr(1,  3, 0.0, 1.0, 0.0)
758: *          call pscr(1,  4, 0.0, 0.0, 1.0)
759: *          call pscr(1,  5, 1.0, 1.0, 0.0)
760: *          call pscr(1,  6, 1.0, 0.0, 1.0)
761: *          call pscr(1,  7, 0.0, 1.0, 1.0)
762: *          call pscr(1,  8, 0.5, 0.5, 0.5)
763: *          call pscr(1,  9, 0.5, 0.0, 0.0)
764: *          call pscr(1, 10, 0.0, 0.5, 0.0)
765: *          call pscr(1, 11, 0.0, 0.0, 0.5)
766: *          call pscr(1, 12, 0.5, 0.5, 0.0)
767: *          call pscr(1, 13, 0.5, 0.0, 0.5)
768: *          call pscr(1, 14, 0.0, 0.5, 0.5)
769: *          call pscr(1, 15, 0.8, 0.8, 1.0)
770: *          call pscr(1, 16, 1.0, 0.8, 0.8)
771: *      ENDIF
772: C92.8.7 <=====
773: CC/#ENDIF
774:       return
775:       end

```

src/cgview/INC/_FLAGS

```
1:      parameter (MAXJDB = 16)
2:      parameter (MKPAR = 64)
3:      COMMON /FLAGS /
4:      & JNEUT, JPHOT, JTIME, JFISS, JEIGN,
5:      & JALLZ, JONEZ, JSIMP, JREFL, JLATT, JHLAT , JTLLT ,
6:      & JRRLT, JIMPT, JWWND, JPSTR, JWTIM,
7:      & JVMNT,
8:      & JDEBG(MAXJDB),
9:      & JMCHK, JFISX, JSTGR, JKPAR(MKPAR)
10: C
11: C .... OPTION FLAGS. DEFAULT VALUES ARE GIVEN IN 'FLAGIN'
12: C      INPUT-SYMBOL          DEFAULT    INCOMPATIBLE OPTION
13: C JALLZ I4    ALL-ZONE          YES
14: C JONEZ I4    ONE-ZONE         NO
15: C JSIMP I4    (GEOMETRY IS SIMPLE (NO (-) SIGNED BODY HAVING MORE THAN ONE
16: C              SURFACE)) ... SET BY CODE IN SUBROUTINE 'ZONEIN')
17: C JREFL I4    (REFLECTIVE SURFACE) NO (TO BE SET INTERNALLY)
18: C JLATT I4    LATTICE GEOMETRY  NO
19: C JHLAT I4    (PROBLEM WITH HEXAGONAL LATTICE)
20: C JTLLT I4    TALLY-LATTICE     : NO
21: C*** added Apr 2000
22: C JKPAR(MKPAR) I4 specified particles are treated in current
23: C      calculation if non-zero.
24: C
25:      COMMON /CGVFLG/ JXPS
26: C
27: C ... CGVIEW Specific flags ...
28: C
29: C JXPS I4    PIFFLIB OUTPUT TYPE
30: C
```


src/cgview/INC/_KPIDS

```
1: C
2: C-----
3: C Particle ID's and particle species number limit for MVP
4: C
5: C Please modify particle symbol information in file _KPSYM
6: C and symbol initialization routine (KPINIT) when particle species
7: C are added or removed from this file.
8: C
9: C
10: C ... neutron and photon
11: C     parameter ( KPNEUT=1 )
12: C     parameter ( KPPHOT=2 )
13: C ... electron and proton
14: C     parameter ( KPELEC=3 )
15: C     parameter ( KPPROT=4 )
16: C ... pi mesons (pi-0 pi+ pi- )
17: C     parameter ( KPPI0=5 )
18: C     parameter ( KPPIP=6 )
19: C     parameter ( KPPIM=7 )
20: C ... mu mesons (mu+ mu- )
21: C     parameter ( KPMUP =8 )
22: C     parameter ( KPMUM =9 )
23: C
24: C-----
25: C ... number of particles possible to treat in this code
26: C
27: C * this should be maximum of possible KP*
28: C * flag array JKPAR must have element size >= KPLIM.
29: C
30: C     parameter ( KPLIM = 3 )
31: C     parameter ( KPLIM = 9 )
32: C
```

src/cgview/INC/_KPSYMS

```
1: C
2: C Particle ID's and symbol matching table used in symbol/ID conversion
3: C routine and its initialization routine of MVP.
4: C
5: C -----
6: C
7: C << "Must-be" 's of this file !!! >>
8: C
9: C * Particle ID definition file _KPIDS *must be* included before
10: C this file
11: C
12: C * When contents of particle ID definition file _KPIDS are changed,
13: C this file *must be* modified to follow the change.
14: C
15: C -----
16: C
17: C ... number of particles possible to treat in this code
18: C (this should be maximum of possible KP* )
19: C
20: C parameter ( KPLIM = ? )
21: C
22: C
23: C ... particle symbol string and their alias (or short form)
24: C
25: C character*32 KPSYM
26: C common /CKPSYM/ KPSYM(2,KPLIM)
27: C
28: C === possible initialization
29: C
30: C ... neutron and photon
31: C KPSYM(1,KPNEUT) = 'NEUTRON'
32: C KPSYM(2,KPNEUT) = 'N'
33: C KPSYM(1,KPPHOT) = 'PHOTON'
34: C KPSYM(2,KPPHOT) = 'P'
35: C ... electron and proton
36: C KPSYM(1,KPELEC) = 'ELECTRON'
37: C KPSYM(2,KPELEC) = 'EL'
38: C KPSYM(1,KPPROT) = 'PROTON'
39: C KPSYM(2,KPPROT) = 'PR'
40: C ... pi mesons (pi-0 pi+ pi- )
41: C KPSYM(1,KPPIO) = 'PIO'
42: C KPSYM(2,KPPIO) = 'PI0'
43: C KPSYM(1,KPPIP) = 'PI-PLUS'
44: C KPSYM(2,KPPIP) = 'PIP'
45: C KPSYM(1,KPPIM) = 'PI-MINUS'
46: C KPSYM(2,KPPIM) = 'PIM'
47: C ... mu mesons (mu+ mu- )
48: C KPSYM(1,KPMUP) = 'MU-PLUS'
49: C KPSYM(2,KPMUP) = 'MUP'
50: C KPSYM(1,KPMUM) = 'MU-MINUS'
51: C KPSYM(2,KPMUM) = 'MUM'
52: C
```

src/cgview/INC/_MVGEOM

```
1: C*
2: C*.... GEOMETRY ARRAY POINTERS (specific to CGVIEW ).....
3: C*
4:      COMMON /MGEOML/ LKMEMO2, LMEMC2, LMEMZ2
5: C*
6: C KMEMO2(NZONE,NMEMO,2) I4  Memory of next-entering zones for each zone.
7: C MEMC2(NZONE,NMEMO,2)  I4  Counter of entering events for zones$
8: C      $ memorized in 'KMEMO' array.
9: C MEMZ2(NZONE,2)    I4  Used elements in 'KMEMO' array for each zone.
```

SAFE

src/cgview/INC/_MVIEW

```

1: C
2: C ... common data for viewer (separated from PGEOM : May 1996)
3: C
4: C
5: C     parameter ( NCGVDB = 10 )
6: C
7: C ... part 1: floating point variables
8: C
9: C     common /PGEOM3/ XMAX , XMIN , YMAX , YMIN , XXMAX , YYMAX,
10: C * DXCG, DYCG , DOVERLAP,
11: C 8 ZMSAVE,XMVSAVE,YMVSAVE,ZMVSAVE,
12: C 8 ROTSAVE,XROTSAVE,YROTSAVE,
13: C 8 XLENG, WFRAME,UFRM,CDT,FACT,XFRM,YFRM
14: C
15: C ... part 2: integer variables
16: C
17: C     common /PGEOM3/ LEVEL , SPTYP ,
18: C A CLDAT , STYLE , STYLE2, IKIND,
19: C B IDENT
20: C C JSCANX, JSCANY, JSCANX0,JSCANY0, MXYZ,
21: C C JGFLOOD, NGFLOOD, NIPGMCK,
22: C C INTREFRESH,
23: C C OFRAME, OVERSION, OLOSTSYM, JHYAXIS,
24: C D JCGVDB(NCGVDB),
25: C E MCELL, IMCELL,
26: C F JZREVERSE, MSGLST
27: C F JGMCK, MPLIMIT
28: C
29: C ... part 3: array pointer variables
30: C
31: C     common /PGEOM3/ LIGMCK, LIPGMCK, LXGMCK, LNDEFSRC,
32: C & LXPLT , LYPLT , LZPLT , LLSCAN,
33: C & LIBPWK, LLPOS0, LLEVL0,
34: C O LCOLOR, LBCOLOR,LBXPLT,LBYPLT
35: C
36: C     parameter (MIPGMCK = 1000)
37: C
38: C     real*8 XMAX,XMIN,YMAX,YMIN,XXMAX,YYMAX
39: C     real*8 DXCG, DYCG
40: C     real*8 DOVERLAP
41: C     real*8 ZMSAVE,XMVSAVE,YMVSAVE,ZMVSAVE
42: C     real*8 ROTSAVE,XROTSAVE,YROTSAVE
43: C
44: C     real XLENG,WFRAME,UFRM,CDT,FACT,XFRM,YFRM
45: C
46: C     integer LEVEL,SPTYP, STYLE,STYLE2,IKIND
47: C
48: C     integer IDENT, JSCANX, JSCANY, JSCANX0,JSCANY0, INTREFRESH
49: C     integer OFRAME, OVERSION, OLOSTSYM, JHYAXIS
50: C     integer JCGVDB
51: C     integer JZREVERSE
52: C
53: C     common /PGEOM4/ TITLGR
54: C     character TITLGR*72
55: C
56: C     common /CGVCOM/ LZONCLR, LIZNCLR, LREGCLR, LMATCLR
57: C     parameter (MXCLDAT=1000)
58: C     integer CLDAT(0:MXCLDAT)
59: C     parameter (MXMAT=10000)
60: C     parameter (MXCLRIDX=16)
61: C
62: C XPLT R8 GRAPHICS WORK
63: C YPLT R8 GRAPHICS WORK
64: C ZPLT R8 GRAPHICS WORK
65: C IBPWK I4 ZONE WORK
66: C XMAX R8 GRAPHICS WORK
67: C XMIN R8 GRAPHICS WORK
68: C YMAX R8 GRAPHICS WORK
69: C YMIN R8 GRAPHICS WORK
70: C ZMAX R8 GRAPHICS WORK
71: C ZMIN R8 GRAPHICS WORK
72: C LSDED I4 LOST PARTICLES
73: C
74: C ***** DISTAANCE LIMITS /PGEOM2/
75: C DEPS R8 RECOGNIZABLE MINIMUM DISTANCE (DEFAULT = 1.0D-08)
76: C DINP R8 DISTANCE TREATED AS INFINITY (DEFAULT = 1.0D+30)
77: C DY R8
78: C
79: C ZMSAVE R8 save buffer for zooming ratio.
80: C XMVSAVE R8 save buffer for x-movement ratio.
81: C YMVSAVE R8 save buffer for y-movement ratio.
82: C ZMVSAVE R8 save buffer for z-movement ratio.
83: C ROTSAVE R8 save buffer for z-rotation.(ROT(...))
84: C XROTSAVE R8 save buffer for x-rotation.(ROTX(...))
85: C YROTSAVE R8 save buffer for y-rotation.(ROTY(...))
86: C
87: C LEVEL I4 LEVEL OF WORLD
88: C SPTYP I4 SELECT ZONE
89: C 1 : INPUT ZONE
90: C 2 : MATERIAL
91: C 3 : REGION
92: C TITLGR C72 GRAPH TITLE
93: C
94: C JSCANX I4 scanning mode in X-direction for drawing
95: C 1|2|3 = +X direction| -X direction| +X/-X alternative
96: C 0 = no scan in X direction
97: C -1 = grid (POINT)
98: C -2 = random (POINT)
99: C JSCANY I4 scanning mode in Y-direction for drawing
100: C (meanings are similar to JSCANX)
101: C
102: C JSCANX0, JSCANY0 : to save older value of "JSCANX/Y"
103: C
104: C LSCAN(NBANK) I4 flight direction in screen
105: C 0 : not specified
106: C +1/-1 : in +X/-X direction of screen
107: C +2/-2 : in +Y/-Y direction of screen
108: C
109: C JGFLOOD : "flood" mode geometry check. Particles are generated
110: C randomly in a square region with random directions.
111: C
112: C INTREFRESH : call plot(0,0,777) (refresh window) every INTFRESH
113: C call of GRAINT
114: C
115: C OFRAME : draw outer frame if non-zero
116: C OVERSION : draw version string of CGVIEW if non-zero
117: C OLOSTSYM : draw lost symbol description if non-zero
118: C
119: C JHYAXIS : draw y-axis numbers horizontally.
120: C
121: C JCGVDB(NCGVDB) : CGVIEW DEBUG FLAG
122: C
123: C MCELL : cell ID displayed ( display global geometry when MCELL<=0)
124: C IMCELL : first zone # of cell MCELL (material must be -999)
125: C ( display global geometry when IMCELL<=0)
126: C
127: C JZREVERSE : control of zone search order in SEAONE.
128: C
129: C 0 = ordinary (from lower zone#)
130: C 1 = reversed (from higher zone#)

```

src/cgview/INC/_MVIEW

```
131: C      2 = mixed (change order line by line)
132: C
133: C ZONCLR(NZONE) : color #for each zone
134: C IZNCLR(NINPZ) : color #for each input-zone
135: C REGCLR(NREG) : color #for each region
136: C MATCLR(0:MXMAT) : color #for each material
137: C MXMAT : limit for number of materials in problem
138: C
139: C MXCLRIDX : maximum of color index
140: C
141: C MSGLST : max pf lost particle message per zone.
142: C JMGCK : flag to check overlapping input-zone definition with
143: C      different material etc.
144: C IGMCK(NINPZ) : counter of overlapped point for input-zones
145: C IPGMCK(2,MPGMCK) : pair list of overlapping input-zones
146: C XGMCK(4,MPGMCK) : overlapping point (1:3) and max. distance(4).
147: C NIPGMCK : number of memorized pairs of overlapping input-zones
148: C DOVERLAP : tolerance factor of overlap judgement
149: C      ( > 0 : times DEFS, < 0 : |DOVERLAP| is absolute distance)
150: C NDEFSRC(NINPZ) : number of "deffered search events"
151: C
152: C      parameter ( MPRNCOM = 8 )
153: C      common /CGVPRINT/ PRNCOM(MPRNCOM), PRNNAM
154: C      character*256 PRNCOM
155: C      character*64 PRNNAM
156: C
157: C
158: C
```

src/cgview/INC/_NGPS

```
1: C
2: C   ... INC/_KPIDS must be included before this file to refer KPLIM
3: C
4: C   COMMON /NGPCOM/  NGP(KPLIM), KNGP(KPLIM+1), KENGP(KPLIM+1)
5: C
6: C NGP(KPLIM)   I4   number of (common) energy groups for each particle
7: C               species.
8: C KNGP(KPLIM+1) I4   KNGP(J) is < Sum of NGP(i),i=1,J-1> + 1.
9: C               It is also used to calcualte energy group # for each
10: C               particle species (from 1 to NGP(particleID)) from
11: C               "unified" energy group # (from 1 to NGROUP).
12: C KENGP(KPLIM+1) I4   KNGP(J) is < Sum of (NGP(i)+1),i=1,J-1> + 1.
13: C               It is used to point arrays on which data for each particle
14: C               species having length of <NGP(*)+1> (such as energy group
15: C               boundaries) are packed in the order of particle species ID.
16: C
```

src/cgview/INC/_PROGV

```
1: C*          $PROGV          LEVEL=4          DATE=92.04.07
2: C*..... DESCRIPTION OF PROGRAM VERSION ETC. ....
3: CHARACTER*60 PROGV(40)
4: DATA (PROGV(I),I=1,15)/
5: 1'          << CGVIEW-SLICE FOR JAERI MONTE CARLO CODES  >>
6: 2'
7: 3'          VERSION 91.1          ( NOV. 1994 )
8: 4'
9: 5'          *** MAIN CAPABILITY OF CGVIEW-SLICE ***
10: 6' DRAW A CROSS SECTIONAL VIEW ON ANY PLANE
11: 7'
12: 8'
13: 9'
14: A' GEOMETRY          : COMBINATORIAL GEOMETRY,
15: 1'                    RECTANGULAR & HEXAGONAL LATTICE.
16: 2'
17: 3'
18: 4'
19: 5'
20: C*
21: DATA (PROGV(I),I=16,30)/
22: 6'          POINT ESTIMATOR
23: 7' BOUNDARY CONDITION : VACUUM, PERFECT REFLECTION &
24: 8' WHITE REFLECTION.
25: 9'
26: A' MATERIAL NUMBER ASSIGNMENT :
27: 1'
28: 2'          .GE. 1 : REAL MEDIUM.
29: 3'          0 : INNER VOID.
30: 4'          -1 TO -999 : LATTICE.
31: 5'          -1000 : OUTER VOID.
32: 6'          -1001 TO -1999 : ALBEDO MATERIAL (NOT IN USE)
33: 7'          -2000 : PERFECT REFLECTOR (MIRROR).
34: 8'          -3000 : WHITE REFLECTOR.
35: 9'
36: A' FILES :
37: C*
38: DATA (PROGV(I),I=31,40)/
39: 1'
40: 2'
41: 3'
42: 4'
43: 5'
44: 6'
45: 7'
46: 8'
47: 9'
48: A'
```

src/cgview/INC/_PTLSP

```

1: C*
2: C* --- DATA FOR COMBINATION OF GEOMETRY & TALLY ----
3: C*
4:     COMMON /PTLSP/
5:     &  NNAMES, NLTSPC, NSMSPC,
6:     &  LTNAMS, LKLTSP, LILTSP, LKCLSP,
7:     &  LISPNM, LISUSP, LKSPSU, LKTCSP, LIRGSP, LKSUZN,
8:     &  LIPTRG, LLPTRG, LITRNM, LKUNV, LKZUNV, NKTCSP
9: C*
10: C NNAMES I4 NUMBER OF "REGION" OR "SPACE" NAMES GIVEN IN GEOMETRY$
11: C     $ INPUT.
12: C NLTSPC I4 NUMBER OF COMBINATIONS OF (SPACE-FAMILY NAME, UNIVERSE)
13: C     $ "SPACE-FAMILY-NAME" IS A NAME GIVEN TO A ZONE WHICH IS $
14: C     $ USED AS THE FRAME OF A SUB-UNIVERSE (LATTICE). <JLTTL>
15: C NSMSPC I4 TOTAL NUMBER OF DATA GIVEN AS "SPACE" NAMES OF EACH CELL $
16: C     $ IN LATTICES OR UNIVERSES.
17: C TNAMS(NNAMES) CH12 LIST OF NAMES$
18: C     $ GIVEN AS "REGION" OR "SPACE" NAMES IN GEOMETRY DATA.$
19: C     $( IN AN ORDER JUDGED BY FUNCTION 'ISTRNM' )
20: C KLTSP(2,NLTSPC) I4 LIST OF (SPACE-FAMILY-NAME: UNIVERSE) COMBINATIONS.
21: C
22: C     (1, ) : SPACE-FAMILY-NAME AS POSITION IN 'TNAMS' LIST.
23: C     (2, ) : UNIVERSE (LATTICE) ID.
24: C*
25: C ILTSP(NLTSPC+1) I4 POINTERS TO 'KCLSP' DATA FOR EACH $
26: C     $(SPACE-FAMILY-NAME: UNIVERSE) COMBINATION.
27: C KCLSP(ILTSP(NLTSPC+1)-1) I4 "SPACE" NAME DATA (POSITION IN 'TNAMS'$
28: C     $ LIST) FOR (SPACE-FAMILY-NAME: UNIVERSE) COMBINATIONS.
29: C ISPNM(2,0:NEST,NSPACE) I4 SPACE NAME LIST.
30: C
31: C     (1,J,) : SPACE-FAMILY NAME (POSITION IN 'TNAMS' LIST) FOR $
32: C     $ J'TH GEOMETRY NESTING LEVEL.
33: C     (2,J,) : SPACE NAME ("GIVEN" NAME) (POSITION IN 'TNAMS' LIST)$
34: C     $ FOR J'TH GEOMETRY NESTING LEVEL.
35: C     (1,0) : GEOMETRY NESTING LEVELS FOR EACH SPACE.
36: C*
37: C ISUSP(NSUZON,NSPACE) I4 REGION #'S FOR EACH SUB-UNIVERSE ZONE $
38: C     $(ZONES FOR WHICH TALLIES ARE TAKEN AS BELONGING REGIONS IN $
39: C     $SUB-UNIVERSES) AND FOR EACH SPACE.
40: C*
41: C KSPSU(NUNV,0:NSPACE) I4 POINTERS TO 'KTCSP' DATA (SPACE #'S IN $
42: C     $UNIVERSES) FOR EACH UNIVERSE-FRAME-ZONE AND SPACES (INCLUDES$
43: C     $ SPACE 0).
44: C*
45: C KSUZN(NINPZ,2)          I4
46: C     (,1) : INPUT-ZONE # FOR EACH SUB-UNIVERSE ZONE(INPUT-ZONE).
47: C     (,2) : SUB-UNIVERSE ZONE # FOR EACH INPUT-ZONE.
48: C*
49: C KTCSP(NKTCSP) I4 SPACE #'S FOR EACH CELL IN UNIVERSES IN SPACES. $
50: C     $( POINTED BY 'KSPSU' )
51: C     $(ZONES FOR WHICH TALLIES ARE TAKEN AS BELONGING REGIONS IN $
52: C     $SUB-UNIVERSES) AND FOR EACH SPACE.
53: C NKTCSP I4 NUMBER OF DATA IN 'KTCSP' ARRAY.
54: C*
55: C IIRGSP(2,NREG) I4
56: C     (1,I) : SPACE # OF I'TH REGION.
57: C     (2,I) : "REGION"-NAME NUMBER IN 'TNAMS' TABLE.
58: C*
59: C IPTRG(NTREG+1) I4 POINTER TO REGION=TALLY-REGION TABLE $
60: C     $'LPTRG'.
61: C*
62: C LPTRG(*) I4 REGION = TALLY-REGION TABLE.
63: C*
64: C ITRNM(NTREG) I4 TALLY-REGION NAME OR REGION NUMBER.
65: C     $ FOR TALLY-REGION SPECIFIED '#REGION' SECTION IN INPUT:

```

```

66: C     $ NAME (POSITION IN 'TNAMS' TABLE * (-1) )
67: C     $ OTHER TALLY-REGION : REGION NUMBER.
68: C*
69: C KUNV(NINPZ) I4 UNIVERSE NUMBER IF AN INPUT-ZONE IS A UNIVERSE-FRAME.
70: C KZUNV(NZONE) I4 UNIVERSE NUMBER IF A ZONE BELONGS A UNIVERSE-FRAME.

```


src/cgview/INC/_STACK

```
1: C .... PARTICLE STACK POINTERS & LENGTHS
2:   COMMON /STACK/
3:     @ LLSFFL, LNFFL , LIZFFL , LLSCOL, NCOLS , LLSSRC, LNNXT , LIZNXT,
4:     @ IDEFER, LLSDED, NDEAD , NFISB , LLSREF, LIZREF, LISREF, LNBREF,
5:     @ LLSLAT, LIZLAT, LNXLT
6: C
7: C ***** FREE FLIGHT STACK *****
8: C LSFFL(NBANK)      I4   BANK INDEX OF FREE-FLIGHT PARTICLE
9: C NFFL(NZONE+1)    I4   NUMBER OF PARTICLES FOR EACH ZONE
10: C IZFFL(NBANK)     I4   ZONE #
11: C ***** COLLISION STACK *****
12: C LSCOL(NBANK)     I4   BANK INDEX OF COLLISION PARTICLE
13: C NCOLS            I4   LENGTH OF COLLISION STACK
14: C ***** SEARCH (NEXT ZONE SEARCH) STACK *****
15: C LSSRC(NBANK)     I4   BANK INDEX OF PARTICLES SEARCHING NEXT ZONE
16: C NNXN(NZONE+1)    I4   LENGTH OF SEARCH STACK
17: C IZNXT(NBANK)     I4   NUMBER OF PARTICLES FOR EACH ZONE
18: C IDEFER           I4   NUMBER OF DEFERRED PARTICLES
19: C***** MORGUE (DEAD PARTICLE STACK) *****
20: C LDED(NBANK)      I4   BANK INDEX OF DEAD PARTICLES
21: C NDEAD            I4   NUMBER OF DEAD PARTICLES
22: C ***** REFLECTION STACK *****
23: C LSREF(NBANK)     I4   BANK INDEX OF PARTICLES TO BE REFLECTED
24: C IZREF(NBANK)     I4   ZONE # FOR REFLECTED PARTICLES TO ENTER
25: C ISREF(NBANK)     I4   REFLECTIVE SURFACE # FOR EACH PARTICLE
26: C NBREF(NREFS+1)   I4   NUMBER OF PARTICLES FOR EACH REFL. SURFACE
27: C
28: C ***** LATTICE STACK *****
29: C LSLAT(NBANK)     I4   BANK INDEX OF PARTICLES WAITING FOR LATTICE SEARCH
30: C IZLAT(NBANK)     I4   LATTICE-BUFFER ZONE #
31: C NXLT(NLBZ+1)     I4   NUMBER OF PARTICLES IN EACH LATTICE-BUFFER-ZONE
32: C
```

src/cgview/INC/_SYMBOL

```

1: C      Produced data from .symbol.cod on SUN-4
2:      DATA (IPAT(I),I= 1, 100)/
3:      @ 2, 1, 2, 12, 2, 31, 2, 40, 2, 49,
4:      @ 2, 58, 2, 69, 2, 80, 2, 89, 2, 102,
5:      @ 2, 0, 2, 107, 2, 124, 2, 135, 2, 140,
6:      @ 2, 157, 2, 162, 2, 167, 2, 176, 2, 183,
7:      @ 2, 196, 2, 0, 2, 207, 2, 220, 2, 233,
8:      @ 2, 0, 2, 238, 2, 243, 3, 4, 3, 17,
9:      @ 3, 24, 3, 37, 2, 0, 3, 62, 3, 75,
10:     @ 3, 100, 3, 117, 3, 146, 3, 185, 3, 212,
11:     @ 3, 225, 3, 238, 3, 251, 4, 8, 4, 17,
12:     @ 4, 30, 4, 35, 4, 46, 4, 51, 4, 70/
13:     DATA (IPAT(I),I= 101, 200)/
14:     @ 4, 81, 4, 100, 4, 131, 4, 140, 4, 163,
15:     @ 4, 190, 4, 203, 4, 238, 5, 11, 5, 32,
16:     @ 5, 55, 5, 62, 5, 71, 5, 78, 5, 103,
17:     @ 5, 142, 5, 157, 5, 182, 5, 203, 5, 218,
18:     @ 5, 231, 5, 242, 6, 11, 6, 24, 6, 37,
19:     @ 6, 50, 6, 63, 6, 70, 6, 81, 6, 90,
20:     @ 6, 123, 6, 128, 6, 153, 6, 172, 6, 199,
21:     @ 6, 208, 6, 221, 6, 228, 6, 247, 7, 0,
22:     @ 7, 11, 7, 20, 7, 29, 7, 48, 7, 57,
23:     @ 7, 64, 7, 69, 7, 74, 7, 103, 7, 124/
24:     DATA (IPAT(I),I= 201, 300)/
25:     @ 7, 141, 7, 162, 7, 183, 7, 198, 7, 225,
26:     @ 7, 240, 7, 249, 8, 10, 8, 21, 8, 28,
27:     @ 8, 55, 8, 72, 8, 91, 8, 112, 8, 133,
28:     @ 8, 148, 8, 173, 8, 188, 8, 205, 8, 216,
29:     @ 8, 235, 8, 244, 8, 255, 9, 8, 9, 27,
30:     @ 9, 32, 9, 51, 2, 0, 2, 0, 2, 0,
31:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
32:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
33:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
34:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0/
35:     DATA (IPAT(I),I= 301, 400)/
36:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
37:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
38:     @ 2, 0, 9, 56, 9, 75, 9, 82, 9, 89,
39:     @ 9, 96, 9, 107, 9, 126, 9, 145, 9, 158,
40:     @ 9, 179, 9, 192, 9, 209, 9, 222, 9, 233,
41:     @ 9, 246, 10, 13, 10, 18, 10, 37, 10, 50,
42:     @ 10, 67, 10, 80, 10, 97, 10, 118, 10, 139,
43:     @ 10, 158, 10, 179, 10, 188, 10, 209, 10, 228,
44:     @ 10, 243, 11, 10, 11, 27, 11, 54, 11, 73,
45:     @ 11, 96, 11, 113, 11, 126, 11, 143, 11, 152/
46:     DATA (IPAT(I),I= 401, 500)/
47:     @ 11, 175, 11, 202, 11, 213, 11, 230, 11, 251,
48:     @ 12, 10, 12, 19, 12, 46, 12, 65, 12, 90,
49:     @ 12, 107, 12, 124, 12, 141, 12, 158, 12, 169,
50:     @ 12, 182, 12, 197, 12, 212, 12, 231, 12, 244,
51:     @ 12, 255, 13, 14, 13, 29, 13, 38, 2, 0,
52:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
53:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
54:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
55:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
56:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0/
57:     DATA (IPAT(I),I= 501, 600)/
58:     @ 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
59:     @ 2, 0, 0, 5, 1, 30, 0, 18, 0, 210,
60:     @ 0, 222, 0, 30, 9, 1, 62, 0, 28, 0,
61:     @ 20, 0, 50, 0, 178, 0, 212, 0, 220, 0,
62:     @ 190, 0, 62, 4, 1, 126, 0, 20, 0, 212,
63:     @ 0, 126, 4, 1, 126, 0, 114, 1, 24, 0,
64:     @ 216, 4, 1, 30, 0, 210, 1, 18, 0, 222,
65:     @ 5, 1, 126, 0, 24, 0, 114, 0, 216, 0,

```

```

66:     @ 126, 5, 1, 114, 0, 126, 0, 24, 0, 216,
67:     @ 0, 126, 4, 1, 18, 0, 222, 0, 30, 0/
68:     DATA (IPAT(I),I= 601, 700)/
69:     @ 210, 6, 1, 30, 0, 222, 0, 18, 0, 210,
70:     @ 1, 168, 0, 72, 2, 1, 222, 0, 171, 8,
71:     @ 1, 126, 0, 114, 1, 18, 0, 222, 1, 216,
72:     @ 0, 24, 1, 30, 0, 210, 5, 1, 222, 0,
73:     @ 30, 0, 210, 0, 18, 0, 222, 2, 1, 126,
74:     @ 0, 114, 8, 1, 126, 0, 20, 0, 212, 0,
75:     @ 126, 1, 28, 0, 114, 0, 220, 0, 28, 2,
76:     @ 1, 120, 0, 216, 2, 1, 24, 0, 18, 4,
77:     @ 1, 26, 0, 82, 0, 154, 0, 26, 3, 1,
78:     @ 20, 0, 91, 0, 148, 6, 1, 30, 0, 158/
79:     DATA (IPAT(I),I= 701, 800)/
80:     @ 1, 152, 0, 24, 1, 18, 0, 146, 5, 1,
81:     @ 24, 0, 152, 0, 122, 0, 118, 0, 152, 6,
82:     @ 1, 27, 0, 155, 1, 149, 0, 21, 1, 34,
83:     @ 0, 142, 6, 1, 26, 0, 154, 1, 94, 0,
84:     @ 85, 1, 18, 0, 146, 2, 1, 18, 0, 146,
85:     @ 2, 1, 30, 0, 158, 8, 1, 125, 0, 110,
86:     @ 0, 94, 0, 77, 0, 67, 0, 50, 0, 34,
87:     @ 0, 19, 6, 1, 27, 0, 123, 0, 153, 0,
88:     @ 151, 0, 117, 0, 21, 3, 1, 20, 0, 92,
89:     @ 0, 148, 6, 1, 24, 0, 41, 0, 73, 0/
90:     DATA (IPAT(I),I= 801, 900)/
91:     @ 103, 0, 135, 0, 152, 12, 1, 26, 0, 43,
92:     @ 0, 75, 0, 105, 0, 137, 0, 154, 1, 150,
93:     @ 0, 133, 0, 101, 0, 71, 0, 39, 0, 22,
94:     @ 6, 1, 62, 0, 53, 1, 51, 0, 34, 0,
95:     @ 66, 0, 51, 12, 1, 61, 0, 45, 0, 46,
96:     @ 0, 62, 0, 60, 0, 43, 1, 125, 0, 109,
97:     @ 0, 110, 0, 126, 0, 124, 0, 107, 8, 1,
98:     @ 25, 0, 156, 1, 150, 0, 19, 1, 49, 0,
99:     @ 62, 1, 126, 0, 113, 14, 1, 155, 0, 125,
100:    @ 0, 61, 0, 27, 0, 26, 0, 56, 0, 120/
101:    DATA (IPAT(I),I= 901,1000)/
102:    @ 0, 150, 0, 148, 0, 114, 0, 50, 0, 20,
103:    @ 1, 81, 0, 94, 19, 1, 17, 0, 158, 1,
104:    @ 62, 0, 46, 0, 29, 0, 28, 0, 43, 0,
105:    @ 59, 0, 76, 0, 77, 0, 62, 1, 116, 0,
106:    @ 99, 0, 98, 0, 113, 0, 129, 0, 146, 0,
107:    @ 147, 0, 132, 13, 1, 150, 0, 148, 0, 97,
108:    @ 0, 49, 0, 19, 0, 22, 0, 106, 0, 109,
109:    @ 0, 94, 0, 62, 0, 45, 0, 42, 0, 145,
110:    @ 6, 1, 61, 0, 45, 0, 46, 0, 62, 0,
111:    @ 60, 0, 43, 6, 1, 111, 0, 76, 0, 57/
112:    DATA (IPAT(I),I=1001,1100)/
113:    @ 0, 54, 0, 67, 0, 96, 6, 1, 47, 0,
114:    @ 76, 0, 89, 0, 86, 0, 67, 0, 32, 6,
115:    @ 1, 93, 0, 81, 1, 19, 0, 155, 1, 147,
116:    @ 0, 27, 4, 1, 24, 0, 152, 1, 93, 0,
117:    @ 82, 6, 1, 50, 0, 34, 0, 35, 0, 51,
118:    @ 0, 49, 0, 32, 2, 1, 24, 0, 152, 5,
119:    @ 1, 33, 0, 34, 0, 50, 0, 49, 0, 33,
120:    @ 2, 1, 17, 0, 158, 9, 1, 62, 0, 28,
121:    @ 0, 19, 0, 49, 0, 113, 0, 147, 0, 156,
122:    @ 0, 126, 0, 62, 5, 1, 60, 0, 94, 0/
123:    DATA (IPAT(I),I=1101,1200)/
124:    @ 81, 0, 65, 0, 97, 9, 1, 26, 0, 28,
125:    @ 0, 62, 0, 126, 0, 156, 0, 154, 0, 18,
126:    @ 0, 17, 0, 145, 15, 1, 27, 0, 28, 0,
127:    @ 62, 0, 126, 0, 156, 0, 154, 0, 120, 0,
128:    @ 72, 0, 120, 0, 150, 0, 147, 0, 113, 0,
129:    @ 49, 0, 19, 0, 21, 4, 1, 148, 0, 20,
130:    @ 0, 126, 0, 113, 11, 1, 158, 0, 30, 0,

```

src/cgview/INC/_SYMBOL

```
131: @ 24, 0, 58, 0, 122, 0, 152, 0, 147, 0,
132: @ 113, 0, 49, 0, 19, 0, 20, 13, 1, 155,
133: @ 0, 156, 0, 126, 0, 62, 0, 28, 0, 19/
134: DATA (IPAT(I),I=1201,1300)/
135: @ 0, 49, 0, 113, 0, 147, 0, 151, 0, 121,
136: @ 0, 57, 0, 23, 6, 1, 27, 0, 30, 0,
137: @ 158, 0, 155, 0, 87, 0, 81, 17, 1, 126,
138: @ 0, 62, 0, 28, 0, 26, 0, 56, 0, 120,
139: @ 0, 150, 0, 147, 0, 113, 0, 49, 0, 19,
140: @ 0, 22, 0, 56, 0, 120, 0, 154, 0, 156,
141: @ 0, 126, 14, 1, 152, 0, 118, 0, 54, 0,
142: @ 24, 0, 28, 0, 82, 0, 126, 0, 156, 0,
143: @ 147, 0, 113, 0, 49, 0, 19, 0, 19, 0,
144: @ 20, 10, 1, 90, 0, 74, 0, 75, 0, 91/
145: DATA (IPAT(I),I=1301,1400)/
146: @ 0, 90, 1, 83, 0, 82, 0, 66, 0, 67,
147: @ 0, 83, 11, 1, 90, 0, 74, 0, 75, 0,
148: @ 91, 0, 90, 1, 82, 0, 66, 0, 67, 0,
149: @ 83, 0, 81, 0, 64, 3, 1, 142, 0, 40,
150: @ 0, 120, 4, 1, 25, 0, 153, 1, 150, 0,
151: @ 22, 3, 1, 30, 0, 120, 0, 18, 12, 1,
152: @ 26, 0, 28, 0, 62, 0, 126, 0, 156, 0,
153: @ 154, 0, 86, 0, 84, 1, 82, 0, 65, 0,
154: @ 97, 0, 82, 19, 1, 117, 0, 100, 0, 68,
155: @ 0, 53, 0, 58, 0, 75, 0, 107, 0, 122/
156: DATA (IPAT(I),I=1401,1500)/
157: @ 0, 117, 0, 132, 0, 149, 0, 156, 0, 123,
158: @ 0, 62, 0, 28, 0, 19, 0, 49, 0, 129,
159: @ 0, 146, 7, 1, 18, 0, 26, 0, 94, 0,
160: @ 154, 0, 146, 1, 151, 0, 23, 12, 1, 18,
161: @ 0, 30, 0, 126, 0, 156, 0, 154, 0, 120,
162: @ 0, 24, 0, 120, 0, 150, 0, 148, 0, 114,
163: @ 0, 18, 10, 1, 155, 0, 156, 0, 126, 0,
164: @ 62, 0, 28, 0, 20, 0, 50, 0, 114, 0,
165: @ 148, 0, 150, 7, 1, 18, 0, 30, 0, 126,
166: @ 0, 156, 0, 148, 0, 114, 0, 18, 6, 1/
167: DATA (IPAT(I),I=1501,1600)/
168: @ 158, 0, 30, 0, 18, 0, 146, 1, 120, 0,
169: @ 24, 5, 1, 158, 0, 30, 0, 18, 1, 24,
170: @ 0, 120, 12, 1, 156, 0, 126, 0, 62, 0,
171: @ 28, 0, 20, 0, 50, 0, 114, 0, 148, 0,
172: @ 152, 1, 104, 0, 152, 0, 146, 6, 1, 30,
173: @ 0, 18, 1, 146, 0, 158, 1, 152, 0, 24,
174: @ 6, 1, 78, 0, 110, 0, 94, 0, 82, 0,
175: @ 66, 0, 98, 6, 1, 126, 0, 116, 0, 82,
176: @ 0, 50, 0, 20, 0, 21, 6, 1, 30, 0,
177: @ 18, 1, 23, 0, 142, 1, 40, 0, 130, 3/
178: DATA (IPAT(I),I=1601,1700)/
179: @ 1, 30, 0, 18, 0, 146, 5, 1, 18, 0,
180: @ 30, 0, 90, 0, 158, 0, 146, 4, 1, 18,
181: @ 0, 30, 0, 146, 0, 158, 11, 1, 62, 0,
182: @ 28, 0, 20, 0, 50, 0, 114, 0, 148, 0,
183: @ 156, 0, 126, 0, 62, 1, 124, 0, 158, 7,
184: @ 1, 24, 0, 120, 0, 154, 0, 156, 0, 126,
185: @ 0, 30, 0, 18, 12, 1, 50, 0, 20, 0,
186: @ 28, 0, 62, 0, 126, 0, 156, 0, 148, 0,
187: @ 114, 0, 50, 0, 84, 0, 116, 0, 146, 9,
188: @ 1, 24, 0, 120, 0, 154, 0, 156, 0, 126/
189: DATA (IPAT(I),I=1701,1800)/
190: @ 0, 30, 0, 18, 1, 56, 0, 146, 13, 1,
191: @ 156, 0, 126, 0, 62, 0, 28, 0, 26, 0,
192: @ 56, 0, 120, 0, 150, 0, 148, 0, 114, 0,
193: @ 50, 0, 20, 0, 21, 4, 1, 30, 0, 158,
194: @ 1, 94, 0, 82, 6, 1, 30, 0, 20, 0,
195: @ 50, 0, 114, 0, 148, 0, 158, 3, 1, 30,
```

```
196: @ 0, 82, 0, 158, 9, 1, 30, 0, 20, 0,
197: @ 50, 0, 84, 0, 90, 0, 84, 0, 114, 0,
198: @ 148, 0, 158, 4, 1, 30, 0, 146, 1, 18,
199: @ 0, 158, 5, 1, 30, 0, 87, 0, 82, 0/
200: DATA (IPAT(I),I=1801,1900)/
201: @ 87, 0, 158, 4, 1, 30, 0, 158, 0, 18,
202: @ 0, 146, 4, 1, 127, 0, 95, 0, 80, 0,
203: @ 112, 9, 1, 30, 0, 88, 0, 81, 0, 88,
204: @ 0, 158, 1, 136, 0, 40, 1, 37, 0, 133,
205: @ 4, 1, 63, 0, 95, 0, 80, 0, 48, 3,
206: @ 1, 44, 0, 95, 0, 140, 2, 1, 18, 0,
207: @ 146, 2, 1, 63, 0, 110, 14, 1, 24, 0,
208: @ 58, 0, 106, 0, 136, 0, 131, 0, 146, 1,
209: @ 131, 0, 114, 0, 50, 0, 20, 0, 21, 0,
210: @ 55, 0, 103, 0, 133, 10, 1, 30, 0, 18/
211: DATA (IPAT(I),I=1901,2000)/
212: @ 1, 20, 0, 50, 0, 98, 0, 132, 0, 136,
213: @ 0, 106, 0, 58, 0, 24, 8, 1, 136, 0,
214: @ 106, 0, 58, 0, 24, 0, 20, 0, 50, 0,
215: @ 98, 0, 132, 10, 1, 142, 0, 130, 1, 132,
216: @ 0, 98, 0, 50, 0, 20, 0, 24, 0, 58,
217: @ 0, 106, 0, 136, 10, 1, 22, 0, 134, 0,
218: @ 136, 0, 106, 0, 58, 0, 24, 0, 20, 0,
219: @ 50, 0, 98, 0, 132, 7, 1, 125, 0, 110,
220: @ 0, 94, 0, 77, 0, 66, 1, 42, 0, 106,
221: @ 13, 1, 136, 0, 106, 0, 58, 0, 24, 0/
222: DATA (IPAT(I),I=2001,2100)/
223: @ 21, 0, 51, 0, 99, 0, 133, 1, 138, 0,
224: @ 130, 0, 96, 0, 48, 0, 18, 7, 1, 30,
225: @ 0, 18, 1, 24, 0, 58, 0, 106, 0, 136,
226: @ 0, 130, 4, 1, 77, 0, 75, 1, 73, 0,
227: @ 66, 8, 1, 109, 0, 107, 1, 105, 0, 97,
228: @ 0, 80, 0, 48, 0, 33, 0, 34, 5, 1,
229: @ 30, 0, 18, 1, 114, 0, 22, 0, 106, 3,
230: @ 1, 62, 0, 78, 0, 66, 13, 1, 26, 0,
231: @ 41, 0, 34, 0, 41, 0, 58, 0, 74, 0,
232: @ 89, 0, 82, 0, 89, 0, 106, 0, 122, 0/
233: DATA (IPAT(I),I=2101,2200)/
234: @ 137, 0, 130, 8, 1, 26, 0, 40, 0, 34,
235: @ 0, 40, 0, 74, 0, 106, 0, 136, 0, 130,
236: @ 9, 1, 58, 0, 24, 0, 20, 0, 50, 0,
237: @ 98, 0, 132, 0, 136, 0, 106, 0, 58, 10,
238: @ 1, 26, 0, 16, 1, 21, 0, 51, 0, 99,
239: @ 0, 133, 0, 136, 0, 106, 0, 58, 0, 24,
240: @ 10, 1, 138, 0, 128, 1, 133, 0, 99, 0,
241: @ 51, 0, 21, 0, 24, 0, 58, 0, 106, 0,
242: @ 136, 7, 1, 42, 0, 34, 0, 40, 0, 74,
243: @ 0, 106, 0, 121, 0, 120, 12, 1, 136, 0/
244: DATA (IPAT(I),I=2201,2300)/
245: @ 106, 0, 58, 0, 24, 0, 23, 0, 38, 0,
246: @ 118, 0, 133, 0, 132, 0, 98, 0, 50, 0,
247: @ 20, 7, 1, 42, 0, 106, 1, 76, 0, 67,
248: @ 0, 82, 0, 98, 0, 115, 8, 1, 26, 0,
249: @ 20, 0, 50, 0, 82, 0, 116, 0, 122, 0,
250: @ 116, 0, 146, 5, 1, 26, 0, 21, 0, 66,
251: @ 0, 117, 0, 122, 9, 1, 26, 0, 20, 0,
252: @ 50, 0, 84, 0, 89, 0, 84, 0, 114, 0,
253: @ 148, 0, 154, 4, 1, 26, 0, 114, 1, 18,
254: @ 0, 122, 5, 1, 26, 0, 84, 1, 138, 0/
255: DATA (IPAT(I),I=2301,2400)/
256: @ 84, 0, 32, 4, 1, 26, 0, 122, 0, 18,
257: @ 0, 114, 9, 1, 111, 0, 95, 0, 78, 0,
258: @ 73, 0, 56, 0, 71, 0, 65, 0, 80, 0,
259: @ 96, 2, 1, 90, 0, 82, 9, 1, 47, 0,
260: @ 63, 0, 78, 0, 73, 0, 88, 0, 71, 0,
```

src/cgview/INC/_SYMBOL

```
261: @ 65, 0, 48, 0, 32, 2, 1, 31, 0, 159,
262: @ 9, 1, 35, 0, 18, 0, 17, 0, 32, 0,
263: @ 48, 0, 65, 0, 66, 0, 51, 0, 35, 3,
264: @ 1, 143, 0, 47, 0, 36, 3, 1, 32, 0,
265: @ 112, 0, 122, 3, 1, 18, 0, 49, 0, 64/
266: DATA (IPAT(I),I=2401,2500)/
267: @ 5, 1, 71, 0, 72, 0, 88, 0, 87, 0,
268: @ 71, 9, 1, 29, 0, 157, 0, 156, 0, 137,
269: @ 0, 119, 0, 67, 0, 33, 1, 25, 0, 137,
270: @ 9, 1, 26, 0, 138, 0, 137, 0, 119, 0,
271: @ 86, 1, 72, 0, 70, 0, 51, 0, 17, 6,
272: @ 1, 139, 0, 104, 0, 70, 0, 21, 1, 87,
273: @ 0, 81, 10, 1, 22, 0, 25, 0, 153, 0,
274: @ 152, 0, 133, 0, 99, 0, 82, 0, 49, 1,
275: @ 89, 0, 91, 0, 42, 0, 138, 1, 90,
276: @ 0, 82, 1, 18, 0, 146, 8, 1, 25, 0/
277: DATA (IPAT(I),I=2501,2600)/
278: @ 137, 1, 91, 0, 81, 0, 65, 1, 19, 0,
279: @ 53, 0, 89, 0, 1, 23, 0, 153, 0, 134,
280: @ 0, 117, 1, 59, 0, 65, 5, 1, 41, 0,
281: @ 89, 0, 82, 1, 18, 0, 130, 6, 1, 26,
282: @ 0, 138, 0, 130, 0, 18, 1, 38, 0, 134,
283: @ 11, 1, 26, 0, 41, 0, 39, 1, 74, 0,
284: @ 89, 0, 87, 1, 154, 0, 152, 0, 116, 0,
285: @ 82, 0, 49, 2, 1, 24, 0, 152, 9, 1,
286: @ 29, 0, 157, 0, 155, 0, 121, 0, 88, 1,
287: @ 74, 0, 70, 0, 51, 0, 17, 6, 1, 22/
288: DATA (IPAT(I),I=2601,2700)/
289: @ 0, 71, 0, 105, 0, 158, 1, 105, 0, 97,
290: @ 8, 1, 24, 0, 28, 0, 156, 0, 151, 0,
291: @ 132, 0, 65, 1, 92, 0, 94, 6, 1, 44,
292: @ 0, 140, 1, 92, 0, 82, 1, 18, 0, 146,
293: @ 8, 1, 27, 0, 155, 1, 110, 0, 97, 0,
294: @ 81, 1, 19, 0, 70, 0, 107, 10, 1, 27,
295: @ 0, 155, 0, 148, 0, 113, 0, 97, 1, 17,
296: @ 0, 51, 0, 69, 0, 91, 0, 94, 10, 1,
297: @ 26, 0, 42, 0, 123, 0, 139, 1, 150, 0,
298: @ 134, 0, 37, 0, 21, 1, 62, 0, 97, 9/
299: DATA (IPAT(I),I=2701,2800)/
300: @ 1, 78, 0, 76, 0, 41, 0, 24, 1, 76,
301: @ 0, 156, 0, 153, 0, 100, 0, 33, 1,
302: @ 62, 0, 59, 0, 41, 0, 24, 1, 59, 0,
303: @ 155, 1, 123, 0, 118, 0, 83, 0, 49, 0,
304: @ 1, 28, 0, 140, 0, 130, 0, 18, 10, 1,
305: @ 26, 0, 42, 0, 139, 0, 155, 1, 62, 0,
306: @ 54, 1, 126, 0, 119, 0, 83, 0, 49, 9,
307: @ 1, 45, 0, 75, 1, 25, 0, 71, 1, 156,
308: @ 0, 153, 0, 117, 0, 66, 0, 17, 7, 1,
309: @ 29, 0, 141, 0, 122, 0, 86, 0, 17, 1/
310: DATA (IPAT(I),I=2801,2900)/
311: @ 86, 0, 145, 11, 1, 26, 0, 42, 0, 156,
312: @ 0, 137, 0, 120, 1, 46, 0, 60, 0, 52,
313: @ 0, 66, 0, 130, 0, 147, 8, 1, 28, 0,
314: @ 58, 0, 72, 1, 157, 0, 155, 0, 135, 0,
315: @ 83, 0, 49, 13, 1, 78, 0, 76, 0, 57,
316: @ 0, 23, 1, 76, 0, 156, 0, 153, 0, 117,
317: @ 0, 83, 0, 33, 1, 57, 0, 88, 0, 118,
318: @ 9, 1, 27, 0, 92, 0, 142, 1, 152, 0,
319: @ 24, 1, 92, 0, 86, 0, 67, 0, 33, 11,
320: @ 1, 29, 0, 43, 0, 40, 1, 77, 0, 91/
321: DATA (IPAT(I),I=2901,3000)/
322: @ 0, 88, 1, 157, 0, 154, 0, 135, 0, 100,
323: @ 0, 33, 8, 1, 45, 0, 141, 1, 153, 0,
324: @ 25, 1, 89, 0, 85, 0, 67, 0, 33, 6,
325: @ 1, 46, 0, 33, 1, 41, 0, 57, 0, 104,
```

```
326: @ 0, 119, 8, 1, 25, 0, 41, 0, 138, 0,
327: @ 154, 1, 94, 0, 87, 0, 67, 0, 33, 4,
328: @ 1, 44, 0, 140, 1, 18, 0, 146, 11, 1,
329: @ 28, 0, 44, 0, 157, 0, 155, 0, 120, 0,
330: @ 68, 0, 17, 1, 41, 0, 72, 0, 132, 0,
331: @ 147, 13, 1, 27, 0, 43, 0, 140, 0, 121/
332: DATA (IPAT(I),I=3001,3100)/
333: @ 0, 86, 0, 19, 1, 102, 0, 133, 0, 148,
334: @ 1, 86, 0, 81, 1, 62, 0, 92, 5, 1,
335: @ 141, 0, 139, 0, 119, 0, 84, 0, 17, 8,
336: @ 1, 75, 0, 73, 0, 53, 0, 18, 1, 109,
337: @ 0, 123, 0, 147, 0, 146, 10, 1, 29, 0,
338: @ 43, 0, 36, 0, 50, 0, 114, 0, 147, 1,
339: @ 139, 0, 122, 0, 89, 0, 40, 7, 1, 28,
340: @ 0, 44, 0, 157, 0, 155, 0, 135, 0, 83,
341: @ 0, 33, 4, 1, 24, 0, 41, 0, 76, 0,
342: @ 146, 13, 1, 26, 0, 42, 0, 139, 0, 155/
343: DATA (IPAT(I),I=3101,3200)/
344: @ 1, 94, 0, 81, 1, 19, 0, 36, 0, 55,
345: @ 0, 56, 1, 121, 0, 133, 0, 147, 9, 1,
346: @ 28, 0, 44, 0, 157, 0, 155, 0, 119, 0,
347: @ 85, 1, 39, 0, 70, 0, 130, 12, 1, 30,
348: @ 0, 46, 0, 93, 0, 123, 1, 25, 0, 41,
349: @ 0, 88, 0, 118, 1, 20, 0, 36, 0, 83,
350: @ 0, 129, 8, 1, 94, 0, 73, 0, 18, 0,
351: @ 50, 0, 132, 1, 103, 0, 132, 0, 146, 8,
352: @ 1, 26, 0, 57, 0, 119, 0, 149, 1, 142,
353: @ 0, 123, 0, 69, 0, 17, 8, 1, 29, 0/
354: DATA (IPAT(I),I=3201,3300)/
355: @ 141, 1, 136, 0, 24, 1, 77, 0, 67, 0,
356: @ 82, 0, 146, 8, 1, 25, 0, 41, 0, 155,
357: @ 0, 154, 0, 136, 1, 46, 0, 60, 0, 97,
358: @ 5, 1, 45, 0, 109, 0, 98, 1, 18, 0,
359: @ 146, 6, 1, 29, 0, 141, 0, 130, 0, 18,
360: @ 1, 40, 0, 136, 7, 1, 45, 0, 125, 1,
361: @ 25, 0, 153, 0, 151, 0, 99, 0, 33, 7,
362: @ 1, 29, 0, 22, 1, 142, 0, 136, 0, 117,
363: @ 0, 83, 0, 33, 9, 1, 61, 0, 55, 0,
364: @ 34, 0, 17, 1, 94, 0, 81, 0, 114, 0/
365: DATA (IPAT(I),I=3301,3400)/
366: @ 131, 0, 149, 6, 1, 30, 0, 17, 0, 33,
367: @ 0, 82, 0, 116, 0, 152, 5, 1, 28, 0,
368: @ 18, 0, 130, 0, 140, 0, 28, 7, 1, 24,
369: @ 0, 29, 0, 157, 0, 154, 0, 117, 0, 83,
370: @ 0, 33, 7, 1, 28, 0, 59, 0, 74, 1,
371: @ 17, 0, 50, 0, 117, 0, 152, 4, 1, 47,
372: @ 0, 60, 1, 79, 0, 92, 9, 1, 63, 0,
373: @ 46, 0, 45, 0, 60, 0, 76, 0, 93, 0,
374: @ 94, 0, 79, 0, 63, 26, 0, 0, 0, 0,
375: @ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0/
```

src/cgview/INC/_WKFL1

```
1: C*
2: C* ..... POINTER TO WORKING ARRAYS  : FLIONE (FLION0)
3: C*
4:      COMMON /WKFL1/ P0(40)
5:      INTEGER  P0
6: C*
7: C P0(40) I4  POINTERS TO WORKING ARRAYS  : FLIONE
```

SAFE

src/cgview/INC/_WKGRA

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEAOE .....
3: C*
4:      COMMON /WKGRA/ P3(20)
5:      INTEGER    P3
6: C*
7: C P3(10) I4    POINTERS TO WORKING ARRAYS : GRAPHICS
```

SEA

src/cgview/INC/_WKLAT

```
1: C*
2: C*..... POINTERS TO WORKING ARRAYS : LATICE
3: C*
4:      COMMON /WKLAT/ P5(20)
5:      INTEGER      P5
6: C*
7: C P5(20) I4  POINTERS TO WORKING ARRAYS : LATICE
```

SAFE

src/cgview/INC/_WKSE1

```
1: C*
2: C* ..... POINTERS TO WORKING ARRAYS : SEAONE .....
3: C*
4:      COMMON /WKSE1/ P2(12)
5:      INTEGER      P2
6: C*
7: C P2(12) I4   POINTERS TO WORKING ARRAYS : SEAONE
```

SEAONE

src/cgview/INC/_WKSOU

```
1: C*
2: C*  .... POINTERS TO WORKING ARRAYS FOR SOURCE
3: C*
4:      COMMON /WKSOU/  P1(15)
5:      INTEGER      P1
6: C*
7: C P1(15) I4  POINTERS TO WORKING ARRAYS : SOURCE
```

SAFE

src/cgview/INC/_XBANK

```

1: C*****
2: C bank parameter specific to CGVIEW
3: C*****
4: COMMON /MVBANK/ LIZSS, LAAAG, LBBBG, LCCCG, LGDIST, LKGCNT
5: C* ***** Bank for CGVIEW *****
6: C IZSS(NBANK) I4 zone search direction (0/1=forward/backward)
7: C AAAG(NBANK) R8 X-direction of scan line.
8: C BBBG(NBANK) R8 Y-direction of scan line.
9: C CCCG(NBANK) R8 Z-direction of scan line.
10: C GDIST(NBANK) R8 flight distance (remaining one until termination
11: C in most case)
12: C KGCNT(NBANK) R8 drawing fragment counter in a scanline
13: C
14: C*****
15: C Commonly used bank parameter in MVP/GMVP/CGVIEW (probably)
16: C*****
17: C*
18: C*... PARTICLE BANK POINTERS
19: C*
20: integer MDBNK, MIBNK
21: integer MDFBK, MIFBK
22: parameter ( MDBNK =20 , MIBNK = 20 )
23: parameter ( MDFBK =20 , MIFBK = 20 )
24:
25: COMMON /XBANK/
26: & LXXX, LYYY, LZZZ, LAAAG, LBBBG, LCCCG, LWWW, LIZZ, LIGG, LTTT
27: & LXXXF,LYYF, LZZZF, LIZZF, LLEVL, LLZZ, LLPOS, LXIM,
28: & LIBSPC, LIBREG,
29: & LLEVLF, LLZZF, LLPOSF, LLCRS, LLCRSF, LKLSF, LKLSFF,
30: & LIBSPF, LIBRGF,
31: & LDBNK, KDBNK(0:MDBNK), LIBNK, KIBNK(0:MIBNK)
32: C*
33: C*.... IMAGINARY PARTICLES ...
34: COMMON /XIBANK/
35: & LXXXI, LYYYI, LZZZI, LAAAI, LBBBI, LCCCI,
36: & LWWWI, LIZZI, LIGGI, LTTTI,
37: & LLEVI, LLZZI, LLPOSI,
38: & LLCRSI, LKLSFI, LKDETP, LOPTI, LPATH
39: C*
40: C XXX(NBANK) R8 X-POSITION (CM)
41: C YYY(NBANK) R8 Y-POSITION (CM)
42: C ZZZ(NBANK) R8 Z-POSITION (CM)
43: C AAA(NBANK) R8 X-DIRECTION
44: C BBB(NBANK) R8 Y-DIRECTION
45: C CCC(NBANK) R8 Z-DIRECTION
46: C WWW(NBANK) R4 WEIGHT
47: C IZZ(NBANK) I4 ZONE NUMBER
48: C IGG(NBANK) I4 ENERGY GROUP
49: C TTT(NBANK) R4 TIME (MSEC) <JTIME>
50: C XIM(NBANK) R4 OLD IMPORTANCE <JIMPT>
51: C***** FISSION PARTICLE BANK *****
52: C XXXF(NFBANK) R8 X-POSITION (CM)
53: C YYYF(NFBANK) R8 Y-POSITION (CM)
54: C ZZZF(NFBANK) R8 Z-POSITION (CM)
55: C IZZF(NFBANK) I4 ZONE NUMBER
56: C***** LATTICE PARAMETER BANK < JLATT > *****
57: C LEVL(NBANK) I4 CURRENT LATTICE LEVEL NUMBER
58: C LZZ (NBANK,NEST) I4 ZONE NUMBER TO RETURN
59: C LPOS(NBANK,NEST) I4 POSITION NUMBER IN LATTICE
60: C LCRS(NBANK,NEST) I4 FLAG TO PARTICLES WHOSE FLIGHT-TRACK MAY CROSS$
61: C $ LATTICE-FRAME. (HEXAGONAL LATTICE)
62: C* ***** FISSION BANK ***** <JLATT> & <JFISS>
63: C LEVLF(NFBANK) I4 CURRENT LATTICE LEVEL NUMBER FOR
64: C LZZF (NFBANK,NEST) I4 ZONE NUMBER TO RETURN
65: C LPOSF(NFBANK,NEST) I4 POSITION NUMBER IN LATTICE

```

```

66: C LCRSF(NFBANK,NEST) I4 FLAG TO PARTICLES WHOSE FLIGHT-TRACK MIGHT$
67: C $ CROSS LATTICE-FRAME. (HEXAGONAL LATTICE)
68: C*
69: C KLSF(NBANK) I4 SURFACE DATA POINTER IF PARTICLE IS ON-SURFACE
70: C KLSFF(NFBANK) I4 SURFACE DATA POINTER IF PARTICLE IS ON-SURFACE
71: C*
72: C*5/27/91 === BANK DATA FOR IMGINARY PATICLES FOR NEXT EVENT ESTIMATOR
73: C KDETP(IMPMAX) I4 DETECTOR NUMBERS FOR EACH IMAGINARY PARTICLES.
74: C OPTI(IMPMAX) R8 OPTICAL PATH FOR IMAGINARY PARTICLES.
75: C PATH(IMPMAX) R8 PATH LENGTH FOR IMAGINARY PARTICLES.
76: C*
77: C

```

src/cgview/INC/_XWORK

```
1: C ..... VECTOR WORK AREA POINTERS .....
2:      COMMON /XWORK/ NWORK,LWORK,NWORKB,LWORKB
3: C
4: C NWORK          I4  NUMBER OF WORK AREA WHOSE LENGTH IS NBANK*4 BYTE.
5: C WORK(NBANK,NWORK)      R4/I4/R8  WORK AREA
6: C NWORKB         I4  NUMBER OF WORK AREA WHOSE LENGTH IS (NZONE+1)*4
7: C                                BYTE.
8: C WORKB(NZONE+1,NWORKB)  R4/I4/R8  WORK AREA
9: C
```

SAFE

```
src/mvpart/cnterr.f
```

[illegible][illegible]

[illegible][illegible]

src/mvpart/getinp.f

```

1:      subroutine GETINP( PATH, LPATH, FILE, NUCIN, RTEMP, NUCOUT, IEND,
2:      &                  NSYSI, NSYSO )
3: c=<mv>=====
4: c  purpose: get a nucid & file name data & requested temperature
5: c-----
6: c  path   : path name of output index & mv material file
7: c  nucin  : nuclide id requested
8: c  file   : nuclide file name requested
9: c
10: c
11: CKSK 72 columns is too short for input & output file name(72=>144)
12:      character*144 PATH
13: CKSK  character*72 LPATH
14:      character*256 FILE
15:      character*12 NUCIN
16:      character*16 NUCOUT
17: c
18: c  .... local variables
19: c
20: CKSK  character*72 OFILE
21: CKSK  character*72 FINDEX
22: CKSK  character*72 LINE
23:      character*144 OFILE
24:      character*144 FINDEX
25:      character*144 LINE
26: CKSK
27: c
28: CKSK add MXCOL
29:      MXCOL      = 144
30:      NIDXO      = 25
31: c
32:      write(NSYSO,*) ' '
33:      write(NSYSO,*) ' == getting nuclide-wise input data of path name'
34:      write(NSYSO,*) ' '
35: c
36:      IEND      = 0
37: c
38: c***** read index file name **** if blank, file by file conversion ****
39: c
40: c ** request path name if not defined
41: c
42:      if ( PATH.eq.' ' ) then
43: c
44:          write(NSYSO,*) 'name of output directory path name ->'
45: c
46: 100      read(NSYSI,'(a)',end =140) LINE
47:          if ( LINE(1:1).eq.''' ' ) go to 100
48:          if ( LINE(1:1).eq.' ' ' ) go to 100
49:          if ( LINE(1:1).eq.'*' ' ) go to 100
50: c
51: c  .... 'path' string ....
52: c
53:      IS      = 1
54:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
55: CKSK      if ( IE.le.0 .or. IE.gt.72 ) IE = 72
56:          if ( IE.le.0 .or. IE.gt.MXCOL ) IE = MXCOL
57: c
58: c
59: c  .... path name ....
60: c
61: c  .... read next line
62:      if ( LINE(IS:IE).ne.'path' ) go to 100
63: c
64: c  .... get path name ...
65: c

```

```

66:      IS      = IE + 1
67:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
68:      IE      = IS + MIN(LEN(PATH)-1,IE-IS)
69:      PATH    = LINE(IS:IE)
70:      LPATH   = IE - IS + 1
71:      write(NSYSO,*) ' output directory name : ', PATH(:LPATH)
72: c
73: c
74: c=====
75: c          open output index file
76: c          logical unit no is 25 (fiexd)
77: c          index file name is findex
78: c=====
79: c
80:      write(NSYSO,*) 'name of output index file -> '
81: c
82: 110      read(NSYSI,'(a)',end =140) LINE
83:          if ( LINE(1:1).eq.''' ' ) go to 110
84:          if ( LINE(1:1).eq.' ' ' ) go to 110
85:          if ( LINE(1:1).eq.'*' ' ) go to 110
86: c
87: c  .... 'index' string ....
88: c
89:      IS      = 1
90:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
91: CKSK      if ( IE.le.0 .or. IE.gt.72 ) IE = 72
92:          if ( IE.le.0 .or. IE.gt.MXCOL ) IE = MXCOL
93: c
94: c
95: c  .... path name ....
96: c
97: c  .... read next line
98:      if ( LINE(IS:IE).ne.'index' ) go to 110
99: c
100: c  .... get findex name ...
101: c
102:      IS      = IE + 1
103:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
104:      IE      = IS + MIN(LEN(FINDEX)-1,IE-IS)
105:      FINDEX   = LINE(IS:IE)
106:      LINDEX   = IE - IS + 1
107: c
108:      write(NSYSO,*) 'index: ', FINDEX(:LINDEX)
109: c
110:      open( NIDXO, file =FINDEX, iostat =IOS, status ='UNKNOWN', form
111: &          ='formatted' )
112: c
113:      if ( IOS.ne.0 ) then
114:          write(NSYSO,*) ' == output index file open error : code ',
115: &          IOS
116:          stop 666
117:      end if
118: c
119:      call rwind(NIDXO)
120:      write(NIDXO,7000) '*****'
121:      write(NIDXO,7000) '** this file is neutron index file **'
122:      write(NIDXO,7000) '** for mv code created by mvdpod code **'
123:      write(NIDXO,7000) '*****'
124:      write(NIDXO,7000) '** this file is used for temporary library**'
125:      write(NIDXO,7000) '*****'
126:      write(NIDXO,7020) 'PATH ', PATH(:LPATH)
127:      write(NIDXO,7000) '*****'
128:      write(NIDXO,7000) '* following nuclide library is calculated *'
129:      write(NIDXO,7000) '* by automatic doppler broadening. *'
130:      write(NIDXO,7000) '*****'

```

src/mvpart/getinp.f

```
131: c
132: 7000      format(A)
133: 7020      format(A5,A)
134:      end if
135: c
136: c
137: c
138: c      .... id ....
139: c
140: 120 LINE   = ' '
141:      read(NSYSI,fmt ='(a)',end =140) LINE
142:      if ( LINE(1:1).eq.'*' .or. LINE.eq.' ' ) go to 120
143: c
144:      NUCIN  = ' '
145: c
146: c      .... get nuclide id character ....
147: c
148:      IS     = 1
149:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
150:      IE     = IS + MIN(LEN(NUCIN)-1,IE-IS)
151:      NUCIN  = LINE(IS:IE)
152: c
153:      call SCTOLC( NUCIN, 12 )
154: c
155: c      .... get temperature
156: c
157:      IS     = IE + 1
158:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
159: c
160:      RTEMP  = 0.0
161:      read(LINE(IS:IE),*) RTEMP
162:      if ( RTEMP.lt.0.0 ) RTEMP = 0.0
163:      write(NSYSO,*) ' ** requested temperature is ', RTEMP, ' kelvin'
164: c
165: 130 LINE   = ' '
166:      read(NSYSI,fmt ='(a)',end =140) LINE
167:      if ( LINE(1:1).eq.'*' .or. LINE.eq.' ' ) go to 130
168: c
169:      NUCOUT = ' '
170: c
171: c      .... get output nuclide id character ....
172: c
173:      IS     = 1
174:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
175:      IE     = IS + MIN(LEN(NUCOUT)-1,IE-IS)
176:      NUCOUT = LINE(IS:IE)
177:      LNUCOU = IE - IS + 1
178:      if ( LNUCOU.lt.8 ) LNUCOU = 8
179:      if ( LNUCOU.gt.12 ) then
180:          LNUCOU = 12
181:          NUCOUT(13:LEN(NUCOUT)) = ' '
182:      end if
183:      call SCTOLC( NUCOUT, 12 )
184: c
185: c      .... get output mvp material file name ...
186: c
187:      IS     = IE + 1
188:      call NOBLNK( LINE, IS, IE, LEN(LINE) )
189:      IE     = IS + MIN(LEN(OFILE)-1,IE-IS)
190:      OFILE  = LINE(IS:IE)
191:      LOFILE = IE - IS + 1
192: c
193:      FILE   = ' '
194:      FILE   = PATH(1:LPATH) //LINE(IS:IE)
195: c
```

```
196:      write(NSYSO,*) ' ** nucout name          : ',
197:      &      NUCOUT(:LNUCOU)
198:      write(NSYSO,*) ' ** output file name      : ',
199:      &      OFILE(:ICLEN2(OFILE))
200:      write(NSYSO,*) ' ** full output file name : ',
201:      &      FILE(:ICLEN2(FILE))
202: c
203:      write(NID XO,'(a,2x,a)') NUCOUT(:LNUCOU), OFILE(:LOFILE)
204: c
205:      IEND   = 0
206:      return
207: c
208: c
209: c
210: 140 close( NID XO )
211:      IEND   = 1
212: c
213:      return
214:      end
```

src/mvpart/gtcmdl.f

```
1:      subroutine GTCMDL
2: C=<MVPART>=====
3: C Purpose : Get arguments in the command line.
4: C
5: C=====
6: C/#IF MS_VISUAL
7: C
8: C ... This interface block is for CUTIL & MS-Visual tools. ...
9: C
10: *      use dflib
11: *      use dfport
12: C/#ENDIF
13: C
14: C      character*128 CSTRNG
15: C
16: C-----
17: C Program control information from command line arguments.
18: C-----
19: C
20: C ... ARGNONE & FAT flag indicating no means to get command argument.
21: C
22: C/#IF ARGV
23: C
24: C/# IF ARGV( GETARG2 )
25: *      KJ = IARGC() - 1
26: C/# ELSEIF ARGV( MSF )
27: *      KJ = NARGS() - 1
28: C/# ELSE
29: *      KJ = IARGC()
30: C/# ENDIF
31:
32:      do 100 I = 1, KJ
33:
34:          IPREIN = IPREIN + 1
35: C/# IF SYSTEM( CRAY )
36: *      call PXFGETARG(I, CSTRNG, LLLLL, ierr )
37: C/# ELSEIF ARGV( IGETARG )
38: *      LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
39: C/# ELSE
40: C/# IF ARGV( GETARG2 )
41: *      call GETARG(I+1, CSTRNG )
42: C/# ELSEIF ARGV( MSF )
43: C ... LL2 should be integer*2
44: *      call GETARG(I, CSTRNG, LL2 )
45: C/# ELSE
46: *      call GETARG( I, CSTRNG )
47: C/# ENDIF
48: *      LLLLL = MIN( INDEX( CSTRNG, ' ' ) - 1, LEN( CSTRNG ) )
49: *      if ( LLLLL.eq.0 ) LLLLL = LEN( CSTRNG )
50: C/# ENDIF
51:
52: C
53: C ... quit command line check if '@@' is encountered ...
54: C
55:
56:      if ( CSTRNG(1:LLLLL) .eq. '@@' ) goto 105
57: c Nagaya
58: c      write(6,*) ' Nagaya (gtcmdl) : CSTRNG(1:LLLLL) = ',
59: c          & CSTRNG(1:LLLLL)
60: c Nagaya
61:      call PREINA( CSTRNG(1:LLLLL) )
62: 100 continue
63: C
64: 105 continue
65: C
```

```
66:      if ( KJ.gt.0 ) write(6,(''1''))
67:
68: C ... END OF C/#IF ARGV ...
69: C/#ENDIF
70: c
71:      return
72:      end
```


src/mvpart/gtvval.f

```
1:      subroutine GTVVAL( VNAME, VALUE, IERR )
2: C=<MVP>=====
3: C  PURPOSE: RETURN THE VALUE OF AN INTERNAL VARIABLE NAMED VNAME:
4: C  CALLED IN:  DENTAK, PARA & OTHERS
5: C  ARGUMENTS:
6: C      VNAME : VARIABLE NAME (NOT VARIABLE ITSELF !)
7: C      VALUE : VALUE (EXPANDED TO DOUBLE PRECISION IF NECESSARY)
8: C      IERR  : ERROR CODE
9: C              0 : SUCCESSFUL
10: C             1 : NON-EXISTENT VARIABLE NAME OR VALUE FETCHING
11: C              NOT ALLOWED.
12: C
13: C             2 : PROBABLY VALUE UNDEFINED YET
14: C
15: C CAUTION: THIS ROUTINE IS CALLED FROM UTILITY ROUTINE DENTAK & PARAM
16: C          BUT CODE-DEPENDENT (MVP & GMVP USE DIFFRENT SOURCE FOR
17: C          THIS ROUTINE.
18: C=====
19:      character*(*) VNAME
20:      real*8 VALUE
21:      integer IERR
22: C
23: C -- Common data -----
24: C
25: C -----
26: C
27: C
28:      real*8 V0
29: C
30: C      .... VJ : A JUNK VALUE TO CONFIRM VALUE ASSIGNMENT ....
31: C
32:      real*8 VJ
33:      parameter( VJ  = -9.9876543D38 )
34: C
35:      IERR    = 0
36:      V0      = VJ
37: C
38: C      if ( VNAME.eq.'NPATOM' ) V0 = NPATOM
39: C      if ( VNAME.eq.'ETOP' ) V0  = ETOP
40: C      if ( VNAME.eq.'EBOT' ) V0  = EBOT
41: C      if ( VNAME.eq.'ETOPP' ) V0 = ETOPP
42: C      if ( VNAME.eq.'EBOTP' ) V0 = EBOTP
43: C      if ( V0.ne.VJ ) go to 100
44: C
45: C .... VERIFICATION ...
46: C
47: 100 continue
48:      if ( V0.ne.VJ ) then
49:          VALUE = V0
50:          IERR   = 0
51:      else
52:          IERR   = 1
53:      end if
54:      return
55:      end
```

src/mvpart/inpidx.f

```

1:      subroutine INPIDX( NONUC, FILE, NUCNM, NSYSI, NSYSO, MXNUC )
2:      C=<MVPART>=====
3:      C PURPOSE: get a nuclid & file name data from cross section index file.
4:      C CALLED IN: MAIN
5:      C CALLS:
6:      C=====
7:      c   nonuc : no of nuclides fetched.
8:      c   nucnm : nuclide id fetched.
9:      c   file  : nuclide file name fetched.
10:     c-----
11:     c
12:     c
13:     character*256 FILE(MXNUC)
14:     character*144 PATH
15:     character*144 FINDEX
16:     character*144 LINE
17:     character*12  NUCNM(MXNUC)
18:     c
19:     c
20:     c ... nout00 is the default output i/o unit.
21:     c
22:     write(NSYSO,*) ' '
23:     write(6,*) ' = getting nuclide & file name from index file =='
24:     write(NSYSO,*) ' '
25:     c
26:     c**** read index file name **** if blank, file by file conversion ****
27:     c
28:     NOCUM   = 0
29:     c
30:     FINDEX  = ' '
31:     write(NSYSO,*) 'name of index file -> '
32:     c
33:     100 read(NSYSI,'(a)') FINDEX
34:     if ( FINDEX(1:1).eq.' ' ) go to 100
35:     c
36:     write(NSYSO,*) 'index: ', FINDEX
37:     c
38:     c=====
39:     c convert data described in index file .
40:     c=====
41:     c
42:     NIDX    = 25
43:     c
44:     C/#IF READONLY(DEC)
45:     *      open( nidx , file=findex, iostat=ios,
46:     *          #      form= 'FORMATTED', readonly )
47:     C/#ELSEIF READONLY(ACTION)
48:     *      open( nidx , file=findex, iostat=ios,
49:     *          #      form= 'FORMATTED', action='READ' )
50:     C/#ELSE
51:     *      open( NIDX, file =FINDEX, iostat =IOS, status = 'OLD', form =
52:     *          &      'FORMATTED' )
53:     C/#ENDIF
54:     c
55:     c
56:     110 continue
57:     c
58:     if ( IOS.ne.0 ) then
59:         write(NSYSO,*) ' == index file open error : code ', IOS
60:         stop 666
61:     end if
62:     c
63:     call GTPATH( NIDX, PATH, LPATH )
64:     c
65:     120 LINE  = ' '

```

```

66:     read(NIDX,fmt = '(a)',end =130,iostat =IOS) LINE
67:     c
68:     if ( IOS.ne.0 ) go to 110
69:     c
70:     if ( LINE(1:1).eq.'*' .or. LINE.eq.' ' ) go to 120
71:     c
72:     c .... get id character or 'path' string ....
73:     c
74:     IS      = 1
75:     call NOBLNK( LINE, IS, IE, LEN(LINE) )
76:     IE      = IS + MIN(LEN(NUCNM(1))-1,IE-IS)
77:     c
78:     c
79:     c .... path name ....
80:     c
81:     if ( LINE(IS:IE).eq.'PATH' ) then
82:     c
83:         c .... get path name ...
84:         c
85:         IS      = IE + 1
86:         call NOBLNK( LINE, IS, IE, LEN(LINE) )
87:         IE      = IS + MIN(LEN(PATH)-1,IE-IS)
88:         PATH    = LINE(IS:IE)
89:         LPATH   = IE - IS + 1
90:         go to 140
91:     c .... read next line
92:     else
93:         go to 120
94:     end if
95:     c
96:     130 continue
97:     call rwind(NIDX)
98:     c
99:     c .... id ....
100:    c
101:    c
102:    140 LINE  = ' '
103:    read(NIDX,fmt = '(a)',end =150) LINE
104:    if ( LINE(1:1).eq.'*' .or. LINE.eq.' ' ) go to 140
105:    c
106:    NONUC    = NONUC + 1
107:    if ( NONUC.gt.MXNUC ) then
108:        write(NSYSO,*) ' XXX(INPIDX) Fatal programing error :',
109:        &      ' Number of nuclides exceeds limit(', MXNUC, ') !!!'
110:        write(NSYSO,*) ' Make MXNUC larger and re-compile!'
111:        stop 999
112:    end if
113:    c
114:    NUCNM(NONUC) = ' '
115:    c
116:    c .... get nuclide id character ....
117:    c
118:    IS      = 1
119:    call NOBLNK( LINE, IS, IE, LEN(LINE) )
120:    IE      = IS + MIN(LEN(NUCNM(1))-1,IE-IS)
121:    NUCNM(NONUC) = LINE(IS:IE)
122:    c
123:    if ( LINE(IS:IE).eq.'PATH' ) then
124:        IS      = IE + 1
125:        call NOBLNK( LINE, IS, IE, LEN(LINE) )
126:        IE      = IS + MIN(LEN(PATH)-1,IE-IS)
127:        PATH    = LINE(IS:IE)
128:        LPATH   = IE - IS + 1
129:        NONUC    = NONUC - 1
130:        goto 140
131:    end if

```

src/mvpart/inpidx.f

```

131: c
132: c .... get filename
133: c
134:       IS       = IE + 1
135:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
136:       IE       = IS + MIN(LEN(FILE(1))-1,IE-IS)
137: c
138:       FILE(NONUC) = ' '
139: c
140:       if ( LPATH.gt.0 ) then
141:         FILE(NONUC) = PATH(1:LPATH) //LINE(IS:IE)
142:       else
143:         FILE(NONUC) = LINE(IS:IE)
144:       end if
145: c
146:       write(NSYSO,*) ' ** nonuc & nucm : ', NONUC, NUCNM(NONUC)
147:       write(NSYSO,*) ' ** infile name : ',
148: & FILE(NONUC) (:ICLEN2(FILE(NONUC)))
149:       go to 140
150: c
151: c
152: c ..... end of index file ...
153: c
154: 150 continue
155:       close( NIDX )
156: c
157:       write(NSYSO,*) ' ** total nuclide is ', NONUC
158: c
159:       if ( NONUC.lt.MXNUC ) then
160:         FILE(NONUC+1) = ' '
161:         NUCNM(NONUC+1) = ' '
162:       end if
163: c
164:       return
165:       end
166:
167:
168: C
169: C =====
170: C subroutine gtpath gets directory path name of file opened on I/O
171: C unit IOU
172: C
173:       subroutine GTPATH( IOU,  PATH,  LPATH )
174: C
175:       character*(*) PATH
176: C
177:       ... local data ...
178: C
179:       character*256 FILE
180:       logical NMD
181: C
182: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
183: C
184:       PATH       = ' '
185:       LPATH      = 0
186:       FILE       = ' '
187:       inquire(IOU,name =FILE,named =NMD,iostat =IOS)
188: C
189:       if ( IOS.eq.0 ) then
190:         if ( NMD ) then
191:           K       = 0
192:           do 100 K = LEN(FILE), 1, -1
193: C/#IF FILENAME(DOS)
194: C
195: C CHAR(92) is '\ ' in ASC-II character code.

```

```

196: C  '\ ' may be taken as "escape code" characters in character string
197: C  on some Fortran compilers. so such an expression is used.
198: C
199: *           if ( FILE(K:K).eq.char(92) ) go to 110
200: C/#ELSEIF FILENAME(MACOS)
201: *           if ( FILE(K:K).eq.':' ) go to 110
202: C/#ELSE
203:           if ( FILE(K:K).eq.'/' ) go to 110
204: C/#ENDIF
205: 100         continue
206:           K       = 0
207: 110         continue
208:           if ( K.gt.0 ) then
209:             PATH   = FILE(:K)
210:             LPATH   = K
211:           end if
212:         end if
213:       end if
214: C/#ENDIF
215:       return
216:       end

```

```
src/mvpart/logoma.f
```

```

1:      subroutine LOGOMA( IPR )
2: C=<MVPART>=====
3: C Purpose: Output a logo 'MVPART'.
4: C
5: C=====
6: C
7:      character*46 MVP(7)
8:      character*44 ART(7)
9:      character*16 BLANK
10: C
11:      data MVP /
12:      &
13:      &
14:      &
15:      &
16:      &
17:      &'
18:      &'/_
19:      data ART /
20:      &'
21:      &'
22:      &'
23:      &'
24:      &'
25:      &'/_
26:
27: C
28: C
29: C
30: C
31: C
32: C
33: C
34: C/_
35: C
36: C
37: C
38: C
39: C
40: C
41: C
42: C/_
43: C
44: C
45:      BLANK = ' '
46:      write(IPR,'((20X,A,A))') (BLANK(:8-I),MVP(i),I=1,7)
47: C KSK write(IPR,'((5X,A,A,A))') (BLANK(:8-I),MVP(i),ART(I),I=1,7)
48: C
49:      return
50:      end

```

src/mvpart/main000.f

```

1: C/#IF CMAIN
2: *      subroutine FTMAIN
3: C/#ELSE
4:      program MVPART
5: C/#ENDIF
6: C=====
7: C      *****
8: C      main routine for mvp arbitrary temperature library processor
9: C      *****
10: C=====
11: C      include '../artcore/INC/_PARAM'
12: C
13: C      parameter( MXNUC      = 3000 )
14: C      parameter( MEMORY     = 50000000 )
15: C
16: C      common /AARRAY/ A
17: C      real A(MEMORY)
18: C
19: C      integer OTAPE
20: C
21: C      common /UNITS/  INP,      OUTP,      ITAPE,      OTAPE,      SCR
22: C      common /HEADR/ C1H,      C2H,      L1H,      L2H,      N1H,      N2H,
23: C      &      MATH,      MFH,      MTH,      NOSEQ
24: C
25: C      common /MAINIO/ NSYSI,      NSYSO,      NSYSO2
26: C
27: C      common /CONTRL/ MATD,      ZA,      ELUNR,      EHUNR,      CRSMIN,      JEMORY,
28: C      &      IUNRES,      INTUNR,      IDUMP,      IDBG
29: C      common /THMCNT/ METHDT,      NATOM,      MODF4T
30: C
31: C      common /HEAD1/  NPTS,      NTDATA,      NUNR,      NUNR2,      NLEV,      NBINA,
32: C      &      NBINE,      NNU,      NMT,      NNK,      IGFLAG,      LFI,      LNU,
33: C      &      IOTHERM,      ISTU,      IENDU,      LSUNR,      LSNU,      NBINA1,      NBINE1,
34: C      &      LNUD,      NNUD,      NNF,      LSNUD,      LASYM,      NOTDOP,
35: C      &      KDUMMY(68),
36: C      &      ATW,      TLAB,      ETHERM,      ESAB,      ELOW,
37: C      &      EHI,      TSTAR,      RDUMMY(43)
38: C      common /HEAD2/  MTINFO(MTMAX),      MTPAR(MTMAX),      ISTMT(MTMAX),
39: C      &      IENDMT(MTMAX),      NEUMT(MTMAX),      LCTMT(MTMAX),
40: C      &      NEANG(MTMAX),      NKF5(MTMAX),      NEF5(MTMAX),
41: C      &      MTHR(MTMAX),      LSTF3(MTMAX),      LSTF4(MTMAX),
42: C      &      LSTF5(MTMAX),      NEF5S(NKMAX,MTMAX),
43: C      &      LFF5S(NKMAX,MTMAX),      LSTF5S(NKMAX,MTMAX),
44: C      &      QVAL(MTMAX),      QVAL2(MTMAX),      IORDMT(MTMAX)
45: C
46: C      common /HEAD3/  NOGAM,      NOGAM3,      MTGAM(MTMAX),      NGTYPE(MTMAX),
47: C      &      NKGAM(MTMAX),      N14GAM(MTMAX),      N15GAM(MTMAX),
48: C      &      LSTGAM(MTMAX),      MTTOG(MTMAX),      NEGYLD(MTMAX),
49: C      &      NEF5G(MTMAX),      NKF5G(MTMAX),      LCTMTG(MTMAX),      NCAP,
50: C      &      NCAPG,      MTCAP(MTMAX),      MTCAPG(MTMAX),      EG1L(MTMAX),
51: C      &      EG2U(MTMAX)
52: C
53: C      common /F6CONT/ LF6,      LSTF6,      MTTOF6(MTMAX),      EF61L(MTMAX),
54: C      &      EF62U(MTMAX)
55: C
56: C      character*8 HDATE
57: C      character*8 MATT
58: C      character*120 PTITLE
59: C      character*256 INFILE
60: C      character*256 IFILE(MXNUC)
61: C      character*256 OTFILE
62: C      character*12 NUCNM(MXNUC)
63: C      character*12 NUCIN
64: C      character*16 NUCOUT
65: C      character*144 PATH

```

```

66:      character TIME8*8, XDATE*12
67: C
68: C      common /ARTCHA/ IFILE,NUCNM
69: C
70: C      logical OPD
71: C
72: C/#IF FLOATHANDLER( ONSTMT )
73: C
74: C      == Floating point error handler specification by "ON" statement
75: C      (Specific to HP fortran)
76: C      ---- TO PREVENT UNDERFLOW ABORT in trapping mode (+T option)---
77: C
78: C      ON REAL*4 UNDERFLOW CALL UNDERF4
79: C      ON REAL*8 UNDERFLOW CALL UNDERF8
80: C/#ENDIF
81: C
82: C-----
83: C
84: C-----initial set
85: C
86: C      call CPUTM(TCP00)
87: C      call TOKEI(T00,1)
88: C
89: C      *** define logical unit for ART ***
90: C
91: C      OTAPE : 0 : for debug in ENDF format
92: C      (=0 means no print, cf. =11 in previous version)
93: C
94: C      NSYSI : 5 : standard input
95: C      NSYSO : 6 : standard output
96: C
97: C      NIN : 10 : ART master library
98: C      NOUT : 20 : Doppler broaden MVP library
99: C
100: C      NSYSO2 : 99 : output for check ( effective if IDBG>1)
101: C
102: C      NSYSI = 5
103: C      NSYSO = 6
104: C      NSYSO2 = 99
105: C      OTAPE = 0
106: C
107: C      NATOM = 0
108: C      MODF4T = 0
109: C      NIN = 10
110: C      NOUT = 20
111: C
112: C      NATOM = 0
113: C      MODF4T = 0
114: C
115: C      .... Get arguments in the command line ....
116: C
117: C      call GTCMDL
118: C
119: C      .... Print logo 'MVPART' & date & time ....
120: C
121: C      call LOGOMA( NSYSO )
122: C      call HIZUKE( XDATE )
123: C      call JIKAN( TIME8 )
124: C
125: C      write(NSYSO,8000) XDATE,TIME8
126: C      8000 format(/20X,' EXECUTION DATE ',A/
127: C      &      20X,' STARTUP TIME ',A/)
128: C
129: C      write(NSYSO,*) ' ** Enter IVPPOP & IDBG in free format '
130: C      read(NSYSI,*) IVPPOP, IDBG

```

src/mvpart/main000.f

```

131: C
132:   if ( IVPOP.ne.0 ) IVPOP = 1
133: C
134:   if ( IDBG.lt.2 ) then
135:     OTAPE = 0
136:     NSYSO2 = 0
137:   endif
138: C
139:   call SETCON
140: C
141:   call READIN
142: C
143: C ... for consistency to ART in MVP
144:   IDBG = IDBG+1
145: C
146:   if ( OTAPE.gt.0 ) then
147: C ***** open otape *****
148:     IOS = 0
149:     open( OTAPE, iostat=IOS, status='UNKNOWN', form='FORMATTED'
150:   &
151: C
152:     if ( IOS.ne.0 ) then
153:       write(NSYSO,*) ' == ENDF output file open error : code '
154:     &
155:       stop 666
156:     end if
157: C
158:     call rwind(OTAPE)
159:     write(OTAPE,7000) 777, 0, 0, 0
160:     NOSEQ = 1
161:   end if
162: C
163:   7000 format(' ENDF output from MVPART',16X,
164:   &
165:   I4,I2,I3,I5)
166: C *** read index file of original mvp library
167: C
168:   NONUC = 0
169:   call INPIDX( NONUC, IFILE, NUCNM, NSYSI, NSYSO, MXNUC )
170:   LPATH = 0
171:   PATH = ' '
172: C
173: C loop of nuclide
174: C
175:   100 continue
176:   RTEMP = 0.0
177:   IEND = 0
178:   call GETINP( PATH, LPATH, OTFILE, NUCIN, RTEMP, NUCOUT, IEND,
179:   &
180:   NSYSI, NSYSO )
181: C
182:   if ( IEND.eq.1 ) go to 130
183: C
184:   INFILE = ' '
185:   MPOS = 0
186:   LL1 = INDEX(NUCIN,' ') - 1
187:   if ( LL1.le.0 ) LL1 = LEN(NUCIN)
188:   do 110 M = 1, NONUC
189:     LL2 = INDEX(NUCNM(M),' ') - 1
190:     if ( LL2.le.0 ) LL2 = LEN(NUCNM(1))
191: C
192:   write(6,*) ' ** m 111 112 : ',m,111,112
193:   write(6,*) ' ** nucin = ',nucin (:111)
194:   write(6,*) ' ** nucnm = ',nucnm(m):(112)
195: C
196:   if ( LL1.eq.LL2 ) then

```

```

196:   if ( NUCIN(:LL1).eq.NUCNM(M):(LL2) ) then
197:     MPOS = M
198:     go to 120
199:   end if
200:   end if
201:   110 continue
202: C
203:   120 continue
204:   if ( MPOS.le.0 ) then
205:     write(NSYSO,*) ' ** ', NUCIN(:LL1),
206:   &
207:     ' is not found in index file '
208:     write(NSYSO,*) ' ** mvp doppler broadening is stopped !!! '
209:     close( 25 )
210:     stop 999
211:   end if
212: C
213:   INFILE = IFILE(MPOS)
214: C
215:   LL = INDEX(INFILE,' ') - 1
216:   if ( LL.lt.0 ) LL = LEN(INFILE)
217: C
218:   write(NSYSO,('( ' input file : ',a)') INFILE(:LL)
219: C
220: C/#IF READONLY(DEC)
221: *   open( nin, file=infile(:ll), iostat=IOS,
222: *   &
223: *   form='UNFORMATTED', readonly )
224: C/#ELSEIF READONLY(ACTION)
225: *   open( NIN, file=INFILE(:LL), iostat=IOS,
226: *   &
227: *   form='UNFORMATTED', action='READ' )
228: C/#ELSE
229: *   open( nin, file=infile(:ll), iostat=IOS, status='OLD',
230: *   &
231: *   form='UNFORMATTED' )
232: C/#ENDIF
233: C
234:   if ( IOS.ne.0 ) then
235:     write(NSYSO,*) ' == input file open error : code ', IOS
236:     stop 666
237:   end if
238: C
239:   ESAB = 0
240:   TSTAR = 0.0
241:   IRC = 0
242:   NPTS = 0
243:   NTDATA = 0
244:   NUNR = 0
245:   NUNR2 = 0
246:   NLEV = 0
247:   NBINA = 0
248:   NBINE = 0
249:   NNU = 0
250:   NMT = MTMAX
251:   NNK = NKMAX
252:   IGFLAG = 0
253:   LFI = 0
254:   LNU = 0
255:   ITERM = 0
256:   ISTU = 0
257:   IENDU = 0
258:   LSUNR = 0
259:   LSNU = 0
260:   NBINA1 = 0
261:   NBINE1 = 0
262:   ATW = 0.0
263:   TLAB = 0.0
264:   ETHERM = 0.0

```

src/mvpart/main000.f

```

261:      ESAB      = 0.0
262:      ELOW      = 0.0
263:      EHI       = 0.0
264: C
265:      LSNUD     = 0
266:      NNUD     = 0
267:      LNUD     = 0
268:      NNF      = 0
269:      NOGAM     = 0
270:      NOGAM3    = 0
271:      NCAP      = 0
272:      NCAPG     = 0
273:      LF6       = 0
274:      LSTF6     = 0
275: C
276:      call CLEA( RDUMMY, 13, 0.0 )
277:      call ICLEA( KDUMMY, 10, 0 )
278:      LENG      = MTMAX*(13+NKMAX*3)
279:      call ICLEA( MTINFO, LENG, 0 )
280:      call CLEA( QVAL, MTMAX, 0.0 )
281:      call CLEA( QVAL2, MTMAX, 0.0 )
282:      LENG      = MTMAX*10
283:      call ICLEA( MTGAM, LENG, 0 )
284:      call ICLEA( LCTMTG, MTMAX, 1 )
285:      call ICLEA( MTCAP, MTMAX, 0 )
286:      call ICLEA( MTCAPG, MTMAX, 0 )
287:      call CLEA( EG1L, MTMAX, 0.0 )
288:      call CLEA( EG2U, MTMAX, 0.0 )
289:      call ICLEA( MTTOF6, MTMAX*3, 0 )
290: C
291:      call rwind(NIN)
292:      read(NIN) MATT, HDATE, PTITLE, NPTS, NTDATA
293:      call rwind(NIN)
294: C
295:      if ( PTITLE(1:9).eq.'VERSION 3' ) then
296:        IVERSN = 3
297:      else if ( INDEX(PTITLE(1:120),'VERSION 2').ne.0 ) then
298:        IVERSN = 2
299:      end if
300: C
301:      if ( NSYSO2.gt.0 ) write(NSYSO2,*) ' *** npts = ', NPTS
302: C
303:      IDUMP     = 0
304:      LOC1      = 1
305:      LOC2      = LOC1 + NPTS
306:      LOC3      = LOC2 + NPTS
307:      LOC4      = LOC3 + NPTS
308:      LOC5      = LOC4 + NPTS
309: CM LOC6       = LOC5 + NTDATA
310:      LOC6      = LOC5 + NTDATA + NPTS ! extend for OK elastic XS
311:      if ( MOD(LOC6,2).eq.0 ) LOC6     = LOC6 + 1
312: C
313:      LEMORY    = MEMORY - LOC6 + 1
314:      call DIMCHK( 'main ', LOC5, MEMORY )
315: C
316:      call ARTPRC( A(LOC1), A(LOC2), A(LOC3), A(LOC4), A(LOC5), A(LOC5),
317: &      A(LOC6), A(LOC6), LEMORY, PTITLE, IVPOP, RTEMP, NIN, IVERSN )
318: C
319:      close( NIN )
320: C
321:      if ( OTAPE.ge.1 ) call FEND
322:      if ( OTAPE.ge.1 ) call MEND
323: C
324: C
325:      inquire(NOUT,opened=OPD)

```

```

326:      if ( OPD ) close( NOUT )
327: C
328:      LL        = INDEX(OTFILE,' ') - 1
329:      if ( LL.lt.0 ) LL = LEN(OTFILE)
330: C
331:      open( NOUT, file =OTFILE(:LL), iostat =IOS, status ='unknown',
332: &      form ='unformatted' )
333: C
334:      if ( IOS.ne.0 ) then
335:        write(NSYSO,*) ' == output file open error : code ', IOS
336:        stop 666
337:      end if
338: C
339: C
340:      JREC3 = 1
341: C
342:      call LIBOUT( A(LOC5), A(LOC5), NOUT, JREC3, PTITLE, NUCOUT,
343: &      IVERSN )
344: C
345:      close( NOUT )
346: C
347: C
348: C
349:      go to 100
350: C
351: C *** end of process
352: C
353: 130 continue
354: C
355:      if ( OTAPE.gt.0 ) then
356:        call TEND
357:        call rwind(OTAPE)
358:        close( OTAPE )
359:      end if
360: C
361:      call CPUTM(TCP01)
362:      call TOKEI(T01,0)
363: C
364:      write(NSYSO,7020) TCP01 - TCP00, T01 - T00
365: 7020 format(/1x,' *** MVPART code finished *** CPU time ',E12.5,
366: &      ' Elapsed time ',E12.5)
367: C
368:      stop
369:      end
370: C
371: C == Floating point error handler specified by "ON" statement
372: C (Specific to HP fortran)
373: C
374: C
375: C ... underflow handling ...
376: C
377:      subroutine UNDERF4( R4 )
378:      real R4
379:      R4 = 0.0
380:      return
381:      end
382: C
383: C
384:      subroutine UNDERF8( R8 )
385:      real*8 R8
386:      R8 = 0.0D0
387:      return
388:      end

```

src/mvpart/preina.f

```
1:      subroutine PREINA( CSTRNG )
2: C=<MVPART>=====
3: C Purpose: Interpret command string for file-name/I/O-unit coupling.
4: C
5: C=====
6: C argument :
7: C
8: C i  cstr : command string
9: C
10: C      /nn[rfu][:file-name]
11: C
12: C      Assign I/O unit nn to a file.
13: C
14: C      xxx=yyy
15: C
16: C      Execution parameter setting.
17: C
18: C-----
19: C
20: C ... external data ...
21: C
22: C      character*(*) CSTRNG
23: C      common /MAINIO/  NSYSI,  NSYSO, NSYSO2
24: C
25: C-----
26: C
27: C      write(NSYSO,'(1x,a,a)') ' PARAMETER : <', CSTRNG, '>'
28: C
29: C      IE      = 0
30: C
31: C .... 'CSTRNG' may be broken into some commands separated by blank.
32: C
33: C
34: C 100 IS      = IE + 1
35: C      call NOBLNK( CSTRNG, IS, IE, LEN(CSTRNG) )
36: C
37: C
38: C ... cstrng(is:ie) : Command .....
39: C
40: C      if ( IE.le.LEN(CSTRNG) ) then
41: C
42: C-----
43: C      File allocation
44: C-----
45: C      if ( CSTRNG(IS:IS).eq.'/' ) then
46: C
47: C
48: C      JJJJJ   = JFOPEN(CSTRNG(IS:IE))
49: C
50: C-----
51: C      unsupported command
52: C-----
53: C
54: C      else
55: C      write(NSYSO,'(1x,a)') ' !!! This command is not supported.'
56: C      end if
57: C
58: C      go to 100
59: C      end if
60: C
61: C      return
62: C      end
```


src/mvpart/prstop.f

```
1:      subroutine PRSTOP( NERR, MESSG )
2: C=====
3: C Purpose: To print summary input-data error when terminating
4: C   before completing input data processing etc.
5: C
6: C   << This routine is necessary to use "FREAD" routines,
7: C   from MVP-BURN v2.0 and after>>
8: C
9: C   nerr : increment count fatal error 'nerr' times.
10: C
11: C CALLED IN: anywhere
12: C CALLS PRNERR
13: C=====
14:      character*(*) MESSG
15: C
16:      common /MAINIO/ NSYSI, NSYSO
17: C-----
18: C
19:      IPR = NSYSO
20: C
21:      if ( NERR.gt.0 ) then
22:         do 100 I = 1, NERR
23:            call CNTERR( 'FATAL' )
24:         100 continue
25:      end if
26: C
27:      call PRNERR( IPR, ' ' )
28: C
29:      write(IPR,7000)
30: 7000 format('1'/5X,5('XXXXXXXXXXXX')//6X,
31: &      ' Your input data has one or more fatal errors which make '
32: &      '//6X,' it impossible to process all of your input data. '//
33: &      5X,5('XXXXXXXXXXXX')//)
34: C
35:      if ( MESSG.ne.' ' ) then
36:         write(IPR,7020) MESSG
37: 7020 format(/5X,' MESSAGE FROM SUBROUTINE : '//5X,A/)
38:      end if
39:
40: C/#IF PARA( PVM )
41: *      call pvmfexit(info)
42: C/#ELSEIF PARA( MPI )
43: *      call mpi_finalize(info)
44: C/#ENDIF
45:
46: C/#IF UNIX
47: *      call flushstd()
48: C/#ENDIF
49:      return
50:      end
```

src/artcore/bmcmkd.f

```
1:      subroutine BMCMKD( F,      Q,      IANS, N1 )
2: c=====
3: c  purpose: preparation for discrete sampling (double precision)
4: c  called in: jnput,prsour,pkkai
5: c      f ... probability density.
6: c=====
7: cm      real*4      f(n1),q(n1)
8:      real*8 F(N1), Q(N1)
9:      integer IANS(2,N1)
10: c
11:      real*8 S, FMAX, FMIN, FN1, SAVE
12: c
13:      S      = 0.0
14:      do 100 K = 1, N1
15:          S      = S + F(K)
16:      100 continue
17: c
18:      if ( S.le.0.0 ) S      = 1.000000D+0
19: c
20:      do 110 K = 1, N1
21:          F(K)      = F(K) /S
22:      110 continue
23: c
24: cm      fn1 = float(n1)
25:      FN1      = DBLE(N1)
26: c
27:      do 130 K = 1, N1
28:          FMAX      = -0.5
29:          FMIN      = 2.0
30: c
31:          do 120 K1 = 1, N1
32:              if ( F(K1).gt.FMAX ) then
33:                  I      = K1
34:                  FMAX      = F(K1)
35:              end if
36:              if ( F(K1).ge.0. ) then
37:                  if ( F(K1).lt.FMIN ) then
38:                      J      = K1
39:                      FMIN      = F(K1)
40:                  end if
41:              end if
42:          120 continue
43:          Q(J)      = 1.0 - FN1*F(J)
44:          F(J)      = -1.0
45:          F(I)      = F(I) - Q(J) /FN1
46:          IANS(1,J)      = I
47:          IANS(2,J)      = J
48:      130 continue
49: c --- check negative q-value
50:      do 140 J = 1, N1
51:          SAVE      = Q(J)
52:          if ( SAVE.lt.1.00000D-9 ) SAVE      = 0.0
53:          Q(J)      = SAVE
54:      140 continue
55: c
56:      return
57:      end
```

src/artcore/broadh.f

```
1: cm      subroutine broadh(y,sigy,xcy,heatm1,heatm2,iwrkmx)
2:      &      subroutine BROADH( Y,      SIGY,  XCY,   HEATM1,HEATM2,IWRKMx,
3:      &      NPMAX, IMAXD, IMAXX, DATOP, XCCOLD,DCOLD,
4:      &      YCOLD, B,    R,      A,      BMY,   EXBMY,
5:      &      ERFCBM,F2B,   F4A,   F4B,   SNEW,  DCNEW,
6:      &      YYMAA, BB,    BPY )
7: c
8: c      high energy doppler broadening routine. this routine will
9: c      be used to doppler broaden all cross sections at energies
10: c      where ae/kt is greater than 16. any point with a lower
11: c      energy will already have been doppler broadened by
12: c      routine broadl.
13: c
14: c      for ae/kt less than or equal to 16 both exponentials in the
15: c      doppler broadening kernel must be considered. for higher energies
16: c      the second exponential may be ignored, which simplifies the
17: c      doppler broadening.
18: c
19: c      the routine has been designed with no subroutine calls
20: c      in order to minimize running time. the arithmetic
21: c      statement functions ration(a) and erfc(r*experf) will
22: c      be compiled as in line coding by virtually any fortran
23: c      compiler, and as such do not represent function calls.
24: c
25: c      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HEATME, LOOPH,
26: c      &      HIHEAT, HEATM1, HEATM2, HOT3, HOT3M1
27: c***** double *****
28: c      real*8 A1, A2, A3, A4, A5, Y, YY, YYA, YYB, YYC, YYD,
29: c      &      DUMMY1, DUMMY2, ZERO, HALF, QUART3, ONE,
30: c      &      ONEP5, TWO, TWOP5, CONA
31: c
32: c      double precision a1,a2,a3,a4,a5,y,a,b,r,bmy,bpy,yma,ypa,
33: c      1 yy,aalp5,bb1p5,yya,yyb,yyc,yyd,dummy1,dummy2,f2b,f4b,f2a,f4a,
34: c      2 expperf,zexp,ration,erfc,ovpi,ovpi2,expbm,expam,
35: c      3 erfcbm,erfcam,zd,ymaa,ymbb,zero,half,quart3,one,onep5,two,
36: c      4 twop5,ycold,cona
37: c***** double *****
38: c      common /INDEX/  COLD1,  COLD2,  COLD1P, COLD2P, HOT1,   HOT2,
39: c      &      HOT3,   HOT3M1, N2IN,   N2TOT,  N2SCR
40: c      common /INDICE/ HEATME, LOOPH, LOHEAT, HIHEAT
41: c
42: c      real*8 OVPI,OVPI2
43: ccccc common /CONTAC/  OVPI,   OVPI2,  ATOP
44: c      include 'INC/_CONST1'
45: c
46: cm      common/comm2/xccold(6012)
47: cm      common/comm3/dcold(6012)
48: cm      common/comm4/ycold(6012)
49: c
50: c      real XCCOLD(NPMAX)
51: c      real DCOLD(NPMAX)
52: c      real*8 YCOLD(NPMAX)
53: c
54: c
55: c
56: c      real*8 B(IMAXD), R(IMAXD), BMY(IMAXD), BPY(IMAXD)
57: c      real*8 EXBMY(IMAXD), ERFCBM(IMAXD), A(IMAXD)
58: c      real*8 F2B(IMAXD), F4B(IMAXD), BB(IMAXD)
59: c      real*8 SNEW(IMAXD), DCNEW(IMAXD)
60: c      real*8 YYMAA(IMAXD), F4A(IMAXD)
61: c
62: c      real*8 DATOP, DEL
63: c
64: c      complementary error function definition.
65: c
66: c-----define constants for complementary error function
67: c***** double *****
68: c      data A1 /0.254829592D+00/
69: c      data A2 /-0.284496736D+00/
70: c      data A3 /1.421413741D+00/
71: c      data A4 /-1.453152027D+00/
72: c      data A5 /1.061405429D+00/
73: c      data CONA /3.275911D-01/
74: c      data ZERO /0.0D+00/
75: c      data HALF /0.5D+00/
76: c      data QUART3 /0.75D+00/
77: c      data ONE /1.0D+00/
78: c      data ONEP5 /1.5D+00/
79: c      data TWO /2.0D+00/
80: c      data TWOP5 /2.5D+00/
81: c
82: c      parameter( A1 =0.254829592D+00)
83: c      parameter( A2 =-0.284496736D+00)
84: c      parameter( A3 =1.421413741D+00)
85: c      parameter( A4 =-1.453152027D+00)
86: c      parameter( A5 =1.061405429D+00)
87: c      parameter( CONA =3.275911D-01)
88: c      parameter( ZERO =0.0D+00)
89: c      parameter( HALF =0.5D+00)
90: c      parameter( QUART3 =0.75D+00)
91: c      parameter( ONE =1.0D+00)
92: c      parameter( ONEP5 =1.5D+00)
93: c      parameter( TWO =2.0D+00)
94: c      parameter( TWOP5 =2.5D+00)
95: c
96: cdel zexp(zd)=exp(zd)
97: c***** double *****
98: c***** single *****
99: c      data a1/0.254829592e+00/
100: c      data a2/-0.284496736e+00/
101: c      data a3/1.421413741e+00/
102: c      data a4/-1.453152027e+00/
103: c      data a5/1.061405429e+00/
104: c      data cona/3.275911e-01/
105: c      data zero/0.0/
106: c      data half/0.5/
107: c      data quart3/0.75/
108: c      data one/1.0/
109: c      data onep5/1.5/
110: c      data two/2.0/
111: c      data twop5/2.5/
112: c      zexp(zd)=exp(zd)
113: c***** single *****
114: c-----define arithmetic statement function for rational argument
115: c-----of complementary error function.
116: c      cdek ration(a)=one/(one+cona*a)
117: c-----define arithmetic statment function for complementary
118: c-----statment function.
119: cdel erfc(r,experf)=(((a5*r+a4)*r+a3)*r+a2)*r+a1)*r*experf
120: c
121: c
122: c      set up loop to doppler broaden cross sections.
123: c
124: c-----define all required constants for point.
125: c      YY = Y*Y
126: c      YYA = YY + HALF
127: c      YYB = TWOP5*YY + QUART3
128: c      YYC = OVPI2*Y
129: c      YYD = YYC*(YY+TWO)
130: c-----initialize integrals.
```

src/artcore/broadh.f

```

131:      DUMMY1 = ZERO
132:      DUMMY2 = ZERO
133:      IMAX = 0
134: c*
135: c*   integrate over all energy intervals above the current energy
136: c*   point.
137: c*
138: c-----no intervals above the last point.
139:       if ( HEATM2.ge.COLD2P ) then
140: c-----y is at upper end of tabulated range. continue cross section as
141: c-----1/v to ycold=infinity.
142:       DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*(Y+OVPI)
143:       go to 160
144:     end if
145: c
146: c-----initialize integrals.
147: c
148: cdel  a      = y
149: cdel  siga   = sigy
150: cdel  f2a    = yya + yyb
151: cdel  f4a    = yyb + yyd
152: c
153:       A(1) = Y
154:       SNEW(1) = SIGY
155:       BB(1) = YY + ONEP5
156:       F2B(1) = YYA + YYC
157:       F4A(1) = YYB + YYD
158: c
159: c-----set up loop over intervals above current one.
160: c   imax = 0
161: c   do 20 looph = heatm2 , hiheat
162: c     b = ycold(looph+1)
163: c     bmy = b-y
164: c-----only extend range of integration up to atop units above y.
165: cm   if(bmy.le.atop) go to 10
166: c     if(bmy.gt.atop) then
167: c       bmy=      atop
168: c       b = y + atop
169: c     endif
170:       IMAX = 0
171:       do 100 LOOPH = HEATM2, HIHEAT
172:         IMAX = IMAX + 1
173:         DEL = YCOLD(LOOPH+1) - Y
174:         BMY(IMAX) = DEL
175: c-----only extend range of integration up to datop units above y.
176:       if ( DEL.ge.DATOP ) then
177:         BMY(IMAX) = DATOP
178:         go to 110
179:       end if
180: c
181: c   100 continue
182: c
183: c   110 continue
184:       if ( IMAX.gt.IMAXX ) stop 999
185: c
186:       LAST = HEATM2 + IMAX - 1
187:       JMAX = 0
188: c
189:       do 120 LOOPH = HEATM2, LAST
190:         JMAX = JMAX + 1
191:         B(JMAX) = BMY(JMAX) + Y
192:         A(JMAX+1) = B(JMAX)
193:         SNEW(JMAX+1) = XCCOLD(LOOPH+1)
194:         DCNEW(JMAX) = DCOLD(LOOPH)
195: c   120 continue

```

```

196: c-----define constants for this point.
197: c   bblp5 = b*b+onep5
198: c   bpy = b+y
199: c   r = ration(bmy)
200: c   expbm = zexp(-bmy*bmy)
201: c
202:       YYMAA(1) = ZERO
203: c
204:       do 130 I = 1, IMAX
205:         BB(I+1) = B(I)*B(I) + ONEP5
206:         BPY(I) = B(I) + Y
207:         R(I) = ONE/(ONE+CONA*BMY(I))
208:         EXBMY(I) = EXP(-BMY(I)*BMY(I))
209:         YYMAA(I+1) = (Y-A(I+1))*(Y+A(I+1))
210: c   130 continue
211: c
212: c   erfcbm = erfc(r,expbm)
213: c   f2b = yya*erfcbm + ovpi*bpy*expbm
214: c   yymaa = (y-a)*(y+a)
215: c   f4b = (yya*yymaa+yyb)*erfcbm+ovpi*(bpy*(bblp5+yymaa)+y)*expbm
216: c
217:       do 140 I = 1, IMAX
218:         ERFCBM(I) = (((A5*R(I)+A4)*R(I)+A3)*R(I)+A2)*R(I)+A1)*R(I)*
219:         & EXBMY(I)
220:         F2B(I+1) = YYA*ERFCBM(I) + OVPI*BPY(I)*EXBMY(I)
221:         F4B(I) = (YYA*YYMAA(I)+YYB)*ERFCBM(I)
222:         & OVPI*(BPY(I)*(BB(I+1)+YYMAA(I))+Y)*EXBMY(I)
223:         F4A(I+1) = (YYA*YYMAA(I+1)+YYB)*ERFCBM(I)
224:         & OVPI*(BPY(I)*(BB(I+1)+YYMAA(I+1))+Y)*EXBMY(I)
225: c   140 continue
226: c
227: c-----add contribution from current interval.
228: c   dummy1 = dummy1 + siga*(f2a-f2b)
229: c   dummy2 = dummy2 + dcold(looph)*(f4a-f4b)
230: c
231:       do 150 I = 1, IMAX
232:         DUMMY1 = DUMMY1 + SNEW(I)*(F2B(I)-F2B(I+1))
233:         DUMMY2 = DUMMY2 + DCNEW(I)*(F4A(I)-F4B(I))
234: c   150 continue
235: c-----test for end of range of integration.
236: c   if(bmy.ge.atop) go to 40
237: c-----save values from last intergal.
238: c   a = b
239: c   siga = xccold(looph+1)
240: c   f2a = f2b
241: c   yymaa = (y-a)*(y+a)
242: c   f4a = (yya*yymaa+yyb)*erfcbm+ovpi*(bpy*(bblp5+yymaa)+y)*expbm
243: c   imax = imax + 1
244: c   20 continue
245: c-----continue cross section as 1/v to ycold=infinity.
246: cm   dummy1 = dummy1+xccold(cold2p)*ycold(cold2p)*(y*erfcbm+ovpi*expbm)
247: c
248:       if ( BMY(IMAX).lt.DATOP ) then
249:         DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*
250:         & (Y*ERFCBM(IMAX)+OVPI*EXBMY(IMAX))
251:       end if
252: c
253: cdel  go to 40
254: c*
255: c*   integrate over all energy intervals below the current energy
256: c*   point.
257: c*
258: c-----no intervals below current first point.
259: c   160 continue
260:       if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX

```

src/artcore/broadh.f

```

261: c
262:   if ( HEATM1.le.COLD1P ) then
263: c-----y is at lower end of tabulated range. continue cross section as
264: c-----1/v to ycold=0.0
265:       DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*(Y-OVPI)
266:       XCY    = HALF*(DUMMY1+DUMMY2) /YY
267:       return
268:   end if
269: c
270: c-----re-initialize integrals to zero distance values
271: c
272:   B(1) = Y
273: cm   sigb = sigy
274: cm   f2b  = yya - yyc
275: cm   f4b  = yyb - yyd
276:   BB(1) = YY + ONEP5
277:   SNEW(1) = SIGY
278:   F2B(1) = YYA - YYC
279:   F4B(1) = YYB - YYD
280: c-----set up loop over intervals below current point.
281:   LOOPH = HEATM1
282:   IMAX = 0
283: c
284: c   do 160 ll = loheat,heatm1
285: c   looph = looph-1
286: c   a     = ycold(looph)
287: c   yma = y - a
288: c-----only extend range of integration down to atop units below y.
289: cm   if(yma.le.atop) go to 150
290: c   if(yma.gt.atop) then
291: c       yma = atop
292: c       a   = y - atop
293: c   endif
294: c
295:   do 170 LL = LOHEAT, HEATM1
296:       IMAX = IMAX + 1
297:       LOOPH = LOOPH - 1
298:       DEL = Y - YCOLD(LOOPH)
299:       BMY(IMAX) = DEL
300: c-----only extend range of integration up to datop units above y.
301:   if ( DEL.ge.DATOP ) then
302:       BMY(IMAX) = DATOP
303:       go to 180
304:   end if
305: 170 continue
306: c
307: 180 continue
308:   if ( IMAX.gt.IMAXX ) stop 999
309: c
310:   LOOPH = HEATM1
311:   JMAX = 0
312: c
313:   LAST = LOHEAT + IMAX - 1
314:   do 190 LL = LOHEAT, LAST
315:       JMAX = JMAX + 1
316:       LOOPH = LOOPH - 1
317:       A(JMAX) = Y - BMY(JMAX)
318:       B(JMAX+1) = A(JMAX)
319:       SNEW(JMAX+1) = XCCOLD(LOOPH)
320:       DCNEW(JMAX) = DCOLD(LOOPH)
321: 190 continue
322: c
323: c-----define constants for this point.
324: c   aalp5 = a*a + onep5
325: c   ypa = y + a

```

```

326: c   r = ration(yma)
327: c   expam = zexp(-yma*yma)
328: c
329:   YYMAA(1) = ZERO
330:   do 200 I = 1, IMAX
331:       BB(I+1) = A(I)*A(I) + ONEP5
332:       BPY(I) = A(I) + Y
333:       R(I) = ONE/(ONE+CONA*BMY(I))
334:       EXBMY(I) = EXP(-BMY(I)*BMY(I))
335:       YYMAA(I+1) = (Y-B(I+1))*(Y+B(I+1))
336: 200 continue
337: c
338: c   erfcam = erfc(r,expam)
339: c   f2a = yya*erfcam - ovpi*ypa*expam
340: c   yymbb = (y-b)*(y+b)
341: c   f4a = (yya*yymbb+yyb)*erfcam-ovpi*(ypa*(aalp5+yymbb)+y)*expam
342: c
343:   do 210 I = 1, IMAX
344:       ERFCBM(I) = (((A5*R(I)+A4)*R(I)+A3)*R(I)+A2)*R(I)+A1)*R(I)*
345:       & EXBMY(I)
346:       F2B(I+1) = YYA*ERFCBM(I) - OVPI*BPY(I)*EXBMY(I)
347:       F4A(I) = (YYA*YYMAA(I)+YYB)*ERFCBM(I)
348:       & - OVPI*(BPY(I)*(BB(I+1)+YYMAA(I))+Y)*EXBMY(I)
349:       F4B(I+1) = (YYA*YYMAA(I+1)+YYB)*ERFCBM(I)
350:       & - OVPI*(BPY(I)*(BB(I+1)+YYMAA(I+1))+Y)*EXBMY(I)
351: 210 continue
352: c
353: c-----add contribution from current interval.
354: c   dummy1 = dummy1 + sigb*(f2b-f2a)
355: c   dummy2 = dummy2 + dcold(looph)*(f4b-f4a)
356: c
357:   do 220 I = 1, IMAX
358:       DUMMY1 = DUMMY1 + SNEW(I)*(F2B(I)-F2B(I+1))
359:       DUMMY2 = DUMMY2 + DCNEW(I)*(F4B(I)-F4A(I))
360: 220 continue
361: c
362: c-----test for end of range of integration.
363: cm   if(yma.ge.atop) go to 190
364: cm   if(bmy(imax).ge.atop) go to 190
365: c-----save values from last integral.
366: c   b = a
367: c   sigb = xccold(looph)
368: c   f2b = f2a
369: c   yymbb = (y-b)*(y+b)
370: c   f4b = (yya*yymbb+yyb)*erfcam-ovpi*(ypa*(aalp5+yymbb)+y)*expam
371: c   imax = imax + 1
372: c 160 continue
373: c
374: c-----continue cross section as 1/v to ycold=0.0
375: c
376: cm   dummy1=dummy1+xccold(cold1p)*ycold(cold1p)*(y*erfcam-ovpi*expam)
377: c
378:   if ( BMY(IMAX).lt.DATOP ) then
379:       DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*
380:       & (Y*ERFCBM(IMAX)-OVPI*EXBMY(IMAX))
381:   end if
382: ccel go to 190
383: c
384: c-----define broadened cross section.
385: c
386: c 190 continue
387: c
388:   XCY = HALF*(DUMMY1+DUMMY2) /YY
389: c
390:   if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX

```

src/artcore/broadh.f

```
391: c
392:   return
393: end
```

SAE

src/artcore/broad1.f

```

1:  cm      subroutine broad1(y,sigy,xcy,heatm1,heatm2,iwrkmx)
2:      subroutine BROADL( Y,      SIGY,  XCY,  HEATM1,HEATM2,IWRKMX,
3:      &                NPMAX, XCCOLD,YCOLD )
4:  c
5:  c      low energy doppler broadening routine. this routine will
6:  c      be used to doppler broaden all cross sections at energies
7:  c      where ae/kt is less than or equal to 16. any point with a
8:  c      higher energy will be passed on to routine broadh.
9:  c
10: c      for ae/kt less than or equal to 16 both exponentials in the
11: c      doppler broadening kernel must be considered. for higher energies
12: c      the second exponential may be ignored, which simplifies the
13: c      doppler broadening.
14: c
15: c      the routine has been designed with no subroutine calls
16: c      in order to minimize running time. the arithmetic
17: c      statement functions ration(a) and erfc(r,experf) will
18: c      be compiled as in line coding by virtually any fortran
19: c      compiler, and as such do not represent function calls.
20: c
21: c      caddb6th
22: c      save IMAX
23: c
24: c      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HEATME, LOOPL,
25: c      &      HIHEAT, HEATM1, HEATM2, HOT3, HOT3M1
26: c      c***** double *****
27: c      real*8 A1, A2, A3, A4, A5, Y, R, B, R, BMY, BPY, YMA,
28: c      &      YPA, YY, YYA, DUMMY1, F2B, F2A, Y2, ERFCY2, RBY, RBPY,
29: c      &      RYMA, RYPA, EXPERF, EXPBM, EXPBP, EXPAM, EXPAP,
30: c      &      RATION, ERFC, ERFCBM, ERFCBP, ERFCAM, ERFCAP,
31: c      &      ZD, ZERO, HALF, ONE, TWO, FOUR, CONA, EXPY2
32: c      c***** double *****
33: c      common /INDEX/  COLD1,  COLD2,  COLD1P, COLD2P, HOT1,  HOT2,
34: c      &      HOT3,    HOT3M1, N2IN,  N2TOT,  N2SCR
35: c      common /INDICE/  HEATME, LOOPL,  LOHEAT, HIHEAT
36: c
37: c      real*8 OVPI, OVPI2
38: c      common /CONTAC/  OVPI,   OVPI2,  ATOP
39: c
40: c      include 'INC/_CONST1'
41: c
42: c      common/comm2/xccold(6012)
43: c      common/comm4/ycold(6012)
44: c
45: c      real XCCOLD(NPMAX)
46: c      real*8 YCOLD(NPMAX)
47: c
48: c      complementary error function definition.
49: c
50: c      c-----define constants for complementary error function
51: c      c***** double *****
52: c      data A1 /0.254829592D+00/
53: c      data A2 /-0.284496736D+00/
54: c      data A3 /1.421413741D+00/
55: c      data A4 /-1.453152027D+00/
56: c      data A5 /1.061405429D+00/
57: c      data CONA /3.275911D-01/
58: c      data ZERO /0.0D+00/
59: c      data HALF /0.5D+00/
60: c      data ONE /1.0D+00/
61: c      data TWO /2.0D+00/
62: c      data FOUR /4.0D+00/
63: c
64: c      parameter( A1 =0.254829592D+00)
65: c      parameter( A2 =-0.284496736D+00)

```

```

66: c      parameter( A3 =1.421413741D+00)
67: c      parameter( A4 =-1.453152027D+00)
68: c      parameter( A5 =1.061405429D+00)
69: c      parameter( CONA =3.275911D-01)
70: c      parameter( ZERO =0.0D+00)
71: c      parameter( HALF =0.5D+00)
72: c      parameter( ONE =1.0D+00)
73: c      parameter( TWO =2.0D+00)
74: c      parameter( FOUR =4.0D+00)
75: c***** double *****
76: c***** single *****
77: c      data a1/0.254829592e+00/
78: c      data a2/-0.284496736e+00/
79: c      data a3/1.421413741e+00/
80: c      data a4/-1.453152027e+00/
81: c      data a5/1.061405429e+00/
82: c      data cona/3.275911e-01/
83: c      data zero/0.0/
84: c      data half/0.5/
85: c      data one/1.0/
86: c      data two/2.0/
87: c      data four/4.0/
88: c      zexp(zd)=exp(zd)
89: c***** single *****
90: c-----define arithmetic statement function for rational argument
91: c-----of complementary error function.
92: c      RATION(A)  = ONE/(ONE+CONA*A)
93: c-----define arithmetic statement function for complementary
94: c-----error function.
95: c      ERFC(R,EXPERF)  = (((A5*R+A4)*R+A3)*R+A2)*R+A1)*R*EXPERF
96: c
97: c      set up loop to doppler broaden cross sections.
98: c
99: c-----define all required constants for point.
100: c      Y2      = Y + Y
101: c      YY      = Y*Y
102: c      YYA     = YY + HALF
103: c-----initialize integrals.
104: c      DUMMY1  = ZERO
105: c      if ( Y2.gt.ATOP ) then
106: c          EXPY2  = ZERO
107: c          ERFCY2 = ZERO
108: c      else
109: c          R      = RATION(Y2)
110: c          EXPY2  = EXP(-Y2*Y2)
111: c          ERFCY2 = ERFC(R,EXPY2)
112: c      end if
113: c*
114: c*      integrate over all energy intervals above the current energy
115: c*      point.
116: c*
117: c-----no intervals above the last point.
118: c      if ( HEATM2.ge.COLD2P ) then
119: c-----continue cross section as 1/v to ycold=infinity.
120: c          DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*TWO*
121: c      &      (Y*(ONE+ERFCY2)+OVPI*(ONE-EXPY2))
122: c          go to 110
123: c      end if
124: c
125: c
126: c
127: c      F2A      = YYA*(ONE-ERFCY2) + OVPI2*Y
128: c      SIGA      = SIGY
129: c
130: c-----set up loop over intervals above current one.

```

src/artcore/broad1.f

```

131: c
132:     IMAX      = 0
133: c
134:     do 100 LOOPL = HEATM2, HIHEAT
135:         B      = YCOLD(LOOPL+1)
136:         SIGB   = XCCOLD(LOOPL+1)
137: c-----only extend range of integration up to atop units above y.
138:         BMY    = B - Y
139:         if ( BMY.gt.ATOP ) then
140:             XXA = YCOLD(LOOPL)**2
141:             XXB = B*B
142:             BMY = ATOP
143:             B   = Y + ATOP
144:             BS  = B*B
145:             SIGB = ((BS-XXB)*SIGA+(XXA-BS)*SIGB) / (XXA-XXB)
146:         end if
147: c-----define contribution of first integral.
148:         REMY    = RATION(BMY)
149:         EXPBM   = EXP(-BMY*BMY)
150:         ERFCBM  = ERFC(REMY,EXPBM)
151: c-----define contribution of second integral.
152:         BPY     = B + Y
153:         if ( BPY.gt.ATOP ) then
154:             EXPBP = ZERO
155:             ERFCBP = ZERO
156:             F2B   = YYA*ERFCBM + OVPI*BPY*EXPBM
157: c
158:         else
159:             RBPY  = RATION(BPY)
160:             EXPBP = EXP(-BPY*BPY)
161:             ERFCBP = ERFC(RBPY,EXPBP)
162: c-----add contribution from current interval.
163:             F2B   = YYA*(ERFCBM-ERFCBP)
164:             &      + OVPI*(B*(EXPBM-EXPBP)+Y*(EXPBM+EXPBP))
165:         end if
166: c
167:         DUMMY1 = DUMMY1 + (SIGA+SIGB)*(F2A-F2B)
168: c-----test for end of range of integration.
169:         if ( BMY.ge.ATOP ) go to 110
170: c-----save values from last integral.
171:         F2A   = F2B
172:         SIGA   = SIGB
173:         IMAX   = IMAX + 1
174:     100 continue
175: c
176: c-----continue cross section as 1/v to ycold=infinity.
177: c
178:         DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*TWO*
179:         &      (Y*(ERFCBM+ERFCBP)+OVPI*(EXPBM-EXPBP))
180: c
181: c-----y is at upper end of tabulated range. continue cross section as
182: c-----1/v to ycold=infinity.
183: c*
184: c*   integrate over all energy intervals below the current energy
185: c*   point.
186: c*
187: c-----no intervals below first point.
188:     110 continue
189: c
190:         if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
191: c
192: c
193:         if ( HEATM1.le.COLD1P ) then
194: c-----y is at lower end of tabulated range. continue cross section as
195: c-----1/v to ycold=zero

```

```

196: c-----continue cross section as 1/v to ycold=0.0
197:         DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*TWO*
198:         &      (Y*(ONE-ERFCY2)-OVPI*(ONE-EXPY2))
199:         XCY    = DUMMY1/(FOUR*YY)
200:         return
201:     end if
202: c
203: c-----re-initialize integrals to zero distance values
204: c
205:         F2B   = YYA*(ONE+ERFCY2) - OVPI2*Y
206:         SIGB   = SIGY
207: c-----set up loop over intervals below current point.
208:         LOOPL = HEATM1
209:         IMAX   = 0
210: c
211:         do 130 LL = LOHEAT, HEATM1
212:             LOOPL = LOOPL - 1
213:             A     = YCOLD(LOOPL)
214:             SIGA   = XCCOLD(LOOPL)
215: c-----only extend range of integration down to atop units below y.
216:             YMA   = Y - A
217: c
218:             if ( YMA.gt.ATOP ) then
219:                 XXB = YCOLD(LOOPL+1)**2
220:                 XXA = A*A
221:                 YMA = ATOP
222:                 A   = Y - ATOP
223:                 AS  = A*A
224:                 SIGA = ((AS-XXB)*SIGA+(XXA-AS)*SIGB) / (XXA-XXB)
225:             end if
226: c-----define contribution of first integral.
227:             RYMA   = RATION(YMA)
228:             EXPAM  = EXP(-YMA*YMA)
229:             ERFCAM = ERFC(RYMA,EXPAM)
230: c-----define contribution of second integral.
231:             YPA    = Y + A
232:             if ( YPA.gt.ATOP ) then
233:                 EXPAP = ZERO
234:                 ERFCAP = ZERO
235:                 F2A    = YYA*ERFCAM - OVPI*YPA*EXPAM
236: c
237:             else
238:                 RYPA  = RATION(YPA)
239:                 EXPAP = EXP(-YPA*YPA)
240:                 ERFCAP = ERFC(RYPA,EXPAP)
241: c-----add contribution from current interval.
242:                 F2A   = YYA*(ERFCAM+ERFCAP)
243:                 &      - OVPI*(A*(EXPAM-EXPAP)+Y*(EXPAM+EXPAP))
244:             end if
245:             120 DUMMY1 = DUMMY1 + (SIGA+SIGB)*(F2B-F2A)
246: c-----test for end of range of integration.
247:             if ( YMA.ge.ATOP ) go to 140
248: c-----save values from last integral.
249:             F2B   = F2A
250:             SIGB   = SIGA
251:             IMAX   = IMAX + 1
252:         130 continue
253: c
254: c-----continue cross section as 1/v to ycold=0.0
255: c
256:         DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*TWO*
257:         &      (Y*(ERFCAM-ERFCAP)-OVPI*(EXPAM-EXPAP))
258: c-----define broadened cross section.
259:         140 continue
260:         if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX

```


src/artcore/broadl.f

```
261:      XCY      = DUMMY1/(FOUR*YY)
262: c
263:      return
264:      end
```

SAFE

src/artcore/broadn.f

```

1:      subroutine BROADN( NPMAX, IMAXD, IMAXX, IADDMX,XCCOLD,DCOLD,
2:      &                  XCHOT, XCSAVE,ECOLD, YCOLD, EHOT,  ESAVE,
3:      &                  WBROAD,IVPOP )
4: C=<ARTCORE>=====
5: C Purpose :
6: C
7: C   given a cross section that is described by a table of cross
8: C   section vs. e and linear-linear interpolation between points
9: C   this routine will exactly doppler broadening the cross
10: C   section at a portion of the energies (one, two or three pages).
11: C
12: C   input parameters
13: C   cold1=index to first point loaded in core.
14: C   cold2=index to last point loaded in core.
15: C   cold1p=index to first point to use in doppler integration
16: C   cold2p=index to last point to use in doppler integration
17: C   the pairs (cold1,cold2) and (cold1p,cold2p) will differ
18: C   only if there is an unresolved resonance present.
19: C   hot1 =index to first point to broaden.
20: C   hot2 =index to last point to broaden.
21: C   ycold =table of speeds corresponding to xcold.
22: C   xccold =table of cross sections at initial temperature.
23: C   dcold=slope between points in (ycold,xcold) table.
24: C   ehot =table of energies corresponding to xchot (and xcold).
25: C   xchot =table of cross sections broadened to temp.
26: C   alpha =doppler width (11505.3*awr/tempef)
27: C
28: C   the table of points (cold1p,cold2p) is used to broaden the
29: C   table of points (hot1,hot2).
30: C
31: C   if there are 6012 (3 pages) or fewer data points all doppler
32: C   broadened cross sections will be calculated in one pass through
33: C   this routine. if there are over 6012 points the data will be
34: C   doppler broadened a page at a time. an exception is that at the
35: C   beginning of the section the first two pages will be broadened
36: C   in one pass and at the end of the section the last two pages will
37: C   be doppler broadened.
38: C
39: C   normally all of the points loaded into core will be used in the
40: C   doppler integration, i.e. cold1p=cold1 and cold2p=cold1p. however
41: C   if the three pages of data that are in core contain any portion of
42: C   the unresolved resonance region the doppler integration will be
43: C   cut off at the edge of the unresolved region and the cross section
44: C   will be extended from that point as 1/v. in this case for data
45: C   points at energies less than the unresolved resonance region
46: C   cold2p will point to the lower energy limit of the unresolved
47: C   region and cold1p will point to the first point in core. similarly
48: C   for points above the upper energy limit of the unresolved region
49: C   cold1p will point to the upper energy limit of the unresolved
50: C   region and cold2p will point to the last point loaded in core.
51: C   within the unresolved energy range the original data point will
52: C   just be copied without doppler broadening and cold1p and cold2p
53: C   are not used.
54: C Called in : FILE3
55: C Calls :
56: C=====
57:      include 'INC/_MAINIO'
58:
59:      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HOT3, HOT3M1,
60:      &      HEATME, HEATER, HIHEAT, UNRES1, UNRES2, HEATM1, HEATM2,
61:      &      HOTEND
62: c***** double *****
63:      real*8 EMID, ARG, Y, YMID, ENEXT, EIH,
64:      &      EIHm1, DEMAX
65:      real*8 DIFF ! work variable for absolute difference

```

```

66:      real*8 RDIF ! work variable for relative difference
67:      real*8 RTOL ! relative tolerance for single precision
68:      parameter(RTOL = 1.d-7) ! 32 bit precision
69: c***** double *****
70:      common /INDEX/   COLD1,  COLD2,  COLD1P, COLD2P, HOT1,  HOT2,
71:      &                HOT3,  HOT3M1, N2IN,  N2TOT,  N2SCR
72:      common /HOTS/    ALPHA,  HOTSY,  TEMPK,  TEMPEF, N2TAPI, N2TAPO
73:      common /INDICE/  HEATME, HEATER, LOHEAT, HIHEAT
74:      common /EXTEND/  MESS,    DTMAX
75:      common /MAXIE/   DEMAX
76:      common /RESOLV/  UREVAL,  UREACT, UNRES1, UNRES2, EULOW,  EUHIGH,
77:      &                IL,      IH
78:      common /OKERRC/  ERXC3C,  ERXC30, KERR3C, MAXERC, ENER3C(21),
79:      &                ER3C(21)
80:      common /SLIM/    ISTART,  NOTHIN, ITHIN1, ITHIN2, ITHIN3, MTEND
81:
82:      real XCCOLD(NPMAX), DCOLD(NPMAX), XCHOT(NPMAX), XCSAVE(IADDMX)
83:      real*8 ECOLD(NPMAX), YCOLD(NPMAX), EHOT(NPMAX), ESAVE(IADDMX)
84:      real*8 WBROAD(IMAXD,14), DATOP, ZERO
85:
86:      include 'INC/_CONST1'
87:
88: c-----define minimum cross section of interest.
89:      data XCMIN /1.00000E-10/
90:      data EMEV /1.00000E+06/
91:      data ZERO /0.00000D+00/
92: c
93: c   set up loop to select points from which to begin iteration.
94: c
95: c-----turn off beginning of energy range flag.
96:      COLD1  = 1
97:      COLD2  = HOT2
98:      COLD1P = COLD1
99:      COLD2P = COLD2
100: c
101:      IWRKMX = 0
102:      KWRKMX = 0
103:      DATOP  = DBLE(ATOP)
104: c
105: c-----if unresolved resonance region data is in core truncate range
106: c-----of integration at unresolved resonance region energy boundary.
107: c-----for energies below unresolved region set upper integration limit
108: c-----to lower energy limit of unresolved region.
109: c
110:      if ( HOT1.le.UNRES1.and.UNRES1.le.COLD2 ) COLD2P = UNRES1
111: c
112:      HOTEND = 0
113:      NLOW   = HOT1
114:      if ( NLOW.gt.1 ) NLOW = NLOW - 1
115: c-----initialize indices to interior cold points in energy range to
116: c-----use for integration.
117:      LOHEAT = COLD1P + 1
118:      HIHEAT = COLD2P - 1
119: c
120: cm  write(6,*) ' ** hot1 hot2 (broadn) : ',hot1,hot2
121: c
122:      do 260 IHEAT = HOT1, HOT2
123:          IHEATM = IHEAT - 1
124: c-----define cold energy and cross section in scalar form.
125:          EIH    = ECOLD(IHEAT)
126:          XCIH   = XCCOLD(IHEAT)
127: c
128: c   unresolved region will be copied and not broadened. when the
129: c   upper energy limit of the unresolved region is reached reset
130: c   integration limits to extend from upper energy of unresolved

```

src/artcore/broadn.f

```

131: c      region to upper energy of data in core.
132: c
133:       if ( IHEAT.gt.UNRES1 ) then
134:       if ( IHEAT.lt.UNRES2 ) then
135: c-----copy points within unresolved region.
136:         HOT3M1 = HOT3
137:         HOT3    = HOT3 + 1
138:         EHOT(HOT3) = EIH
139:         XCHOT(HOT3) = XCIH
140: c-----if over 2 pages of broadened data thin it.
141:       if ( HOT3.ge.NPMAX ) call THINIT( NPMAX, EHOT, XCHOT )
142:       if ( HOT3.gt.NPMAX ) then
143:         write(NSYSO,*) ' ** hot3 npmax (broadn) : ',
144:           & HOT3, NPMAX
145:         stop 999
146:       end if
147:       go to 240
148:       else if ( IHEAT.eq.UNRES2 ) then
149: c-----upper limit of unresolved region reached. define range of
150: c-----of integration from upper limit of unresolved range up to last
151: c-----point in core.
152:         COLD1P = UNRES2
153:         COLD2P = COLD2
154: c-----re-define indices to interior cold points in energy range to
155: c-----use for integration.
156:         LOHEAT = COLD1P + 1
157:         HIHEAT = COLD2P - 1
158:       end if
159:     end if
160: c
161: c      always use first point as node.
162: c
163: c-----is this first point in energy range.
164:       if ( IHEAT.ne.COLD1P ) then
165: c
166: c      always use last point as node.
167: c
168:       if ( IHEAT.ge.HOT2 .or. IHEAT.ge.COLD2P ) go to 110
169: c
170: c      use preceding point if it was maximum or minimum and energy has
171: c      not yet been used.
172: c
173:       if ( DSIGN*DCOLD(IHEATM).lt.0.0 ) then
174:         DSIGN = -DSIGN
175: c-----only use point if same energy has not yet been used.
176:       if ( EIH1.le.EHOT(HOT3) ) go to 250
177: c
178: c      define parameters at energy point to broaden.
179: c
180: c-----define broadened cross section at same energy that cold cross
181: c-----section is given (preceding point).
182:       HOT3M1 = HOT3
183:       HOT3    = HOT3 + 1
184:       EHOT(HOT3) = EIH1
185:       Y        = YCOLD(IHEATM)
186:       SIGY     = XCIHM1
187:       HEATM1   = IHEATM
188:       HEATM2   = IHEATM
189:       NHIGH    = IHEATM
190:       go to 130
191:     end if
192:   else
193: c-----initialize sign of slope.
194:     DSIGN = 1.0
195:     if ( DCOLD(COLD1P).lt.0.0 ) DSIGN = -1.0

```

```

196: c-----set flag to indicate beginning of energy range.
197:       HOTEND = 1
198:       go to 110
199:     end if
200: c
201: c      only use point if same energy has not yet been used (broadening
202: c      eliminates all discontinuities).
203: c
204: 100   if ( EHOT(HOT3).ge.EIH ) go to 250
205: c
206: c      use all points above 1 MeV.
207: c
208:       if ( EIH.lt.EMEV ) then
209: c
210: c      use all non-positive cross section points.
211: c
212:       if ( XCIH.gt.0.0 ) then
213: c
214: c      use all points where cold cross section changes by more than a
215: c      factor of two.
216: c
217:       if ( XCIH.le.2.0*SIGY.and.SIGY.le.2.0*XCIH ) then
218: c
219: c      if point is not within allowable energy spacing interpolate to
220: c      insert energy point.
221: c
222:       if ( EIH.gt.ENEXT ) go to 120
223: c
224: c      use at least every tenth point.
225: c
226: c
227: c      do not use point as node.
228: c
229:       if ( (IHEAT-NLOW).lt.10 ) go to 250
230: c-----cold cross section has changed by factor of two. use either cross
231: c-----section or energy interval.
232:       else if ( EIH.gt.ENEXT ) then
233:         go to 120
234:       end if
235:     end if
236:   end if
237: c-----define broadened cross section at same energy that cold cross
238: c-----section is given (current point).
239: 110   HOT3M1 = HOT3
240:       HOT3    = HOT3 + 1
241:       EHOT(HOT3) = EIH
242:       Y        = YCOLD(IHEAT)
243:       SIGY     = XCIH
244:       HEATM1   = IHEAT
245:       HEATM2   = IHEAT
246:       NHIGH    = IHEAT
247:       go to 130
248: c-----define broadened cross section at energy at which cold cross
249: c-----section not given (define cross section by energy interpolations).
250: 120   HOT3M1 = HOT3
251:       HOT3    = HOT3 + 1
252:       EHOT(HOT3) = ENEXT
253:       ARG      = ALPHA*EHOT(HOT3)
254:       Y        = SQRT(ARG)
255:       SIGY     = ((EIH-ENEXT)*XCIHM1+(ENEXT-EIHM1)*XCIH) / (EIH-EIHM1)
256:       HEATM1   = IHEAT
257:       HEATM2   = IHEATM
258:       NHIGH    = IHEATM
259: c
260: c      broaden data.

```

src/artcore/broadn.f

```

261: c
262: c-----initialize count of saved data points.
263: 130  ISAVE = 0
264: c-----select doppler broadening method.
265:   if ( Y.le.ATOP ) then
266:     call BROADL( Y, SIGY, XCHOT(HOT3), HEATM1, HEATM2, IWRKMX,
267:   &      NPMAX, XCCOLD, YCOLD )
268:   else if ( Y.gt.HOTSYS ) then
269:     if ( IVPOP.eq.1 ) then
270:       call DCLEA( WBROAD, 12*IMAXD, ZERO )
271:       call BROADS( Y, SIGY, XCHOT(HOT3), HEATM1, HEATM2,
272:   &      IWRKMX, NPMAX, IMAXD, IMAXX, DATOP, XCCOLD,
273:   &      DCOLD, YCOLD, WBROAD(1,1), WBROAD(1,2),
274:   &      WBROAD(1,3), WBROAD(1,4), WBROAD(1,5),
275:   &      WBROAD(1,6), WBROAD(1,7), WBROAD(1,8),
276:   &      WBROAD(1,9), WBROAD(1,10), WBROAD(1,11),
277:   &      WBROAD(1,12) )
278:     else
279:       call SROADS( Y, SIGY, XCHOT(HOT3), HEATM1, HEATM2,
280:   &      IWRKMX, NPMAX, XCCOLD, DCOLD, YCOLD )
281:     end if
282:   else if ( IVPOP.eq.1 ) then
283:     call DCLEA( WBROAD, 14*IMAXD, ZERO )
284:     call BROADH( Y, SIGY, XCHOT(HOT3), HEATM1, HEATM2, IWRKMX,
285:   &      NPMAX, IMAXD, IMAXX, DATOP, XCCOLD, DCOLD, YCOLD,
286:   &      WBROAD(1,1), WBROAD(1,2), WBROAD(1,3), WBROAD(1,4),
287:   &      WBROAD(1,5), WBROAD(1,6), WBROAD(1,7), WBROAD(1,8),
288:   &      WBROAD(1,9), WBROAD(1,10), WBROAD(1,11),
289:   &      WBROAD(1,12), WBROAD(1,13), WBROAD(1,14) )
290:   else
291:     call SROADH( Y, SIGY, XCHOT(HOT3), HEATM1, HEATM2, IWRKMX,
292:   &      NPMAX, XCCOLD, DCOLD, YCOLD )
293:   end if
294: c-----check for beginning of energy range (only broaden one point and
295: c-----then go to end of loop to set up for first energy interval).
296:   if ( HOTEND.gt.0 ) go to 240
297: c-----do not interpolate zero length intervals.
298: 140  if ( EHOT(HOT3).le.EHOT(HOT3M1) ) go to 230
299: c
300: c   check for convergence.
301: c
302: c-----do not interpolate if cross sections at both ends of interval
303: c-----are absolutely less than minimum cross section of interest
304:   if ( ABS(XCHOT(HOT3)).lt.XCMIN.and.ABS(XCHOT(HOT3M1)).lt.XCMIN
305:   &      ) go to 230
306: c
307: c   perform calculation at midpoint.
308: c
309: c-----define mid-point energy and speed-like term.
310:   EMID = 0.5*(EHOT(HOT3)+EHOT(HOT3M1))
311: c   if ( EMID.le.EHOT(HOT3M1) ) go to 230
312:   RDIF = abs(EMID - EHOT(HOT3M1))/EHOT(HOT3M1)
313:   if(RDIF.lt.RTOL) go to 230
314:   ARG = ALPHA*EMID
315:   YMID = SQRT(ARG)
316: c-----define indices to energy interval or point.
317:   do 150 NUSE = NLOW, IHEAT
318: c   if ( YMID.lt.YCOLD(NUSE) ) go to 170
319: c   if ( YMID.eq.YCOLD(NUSE) ) go to 160
320:   DIFF = YMID - YCOLD(NUSE)
321:   RDIF = abs(DIFF)/YCOLD(NUSE)
322:   if(RDIF.lt.RTOL) then
323:     go to 160
324:   else if(DIFF.lt.0.d0) then
325:     go to 170

```

```

326:   end if
327: 150  continue
328:   NUSE = IHEAT
329: c-----energy point.
330: 160  SIGMID = XCCOLD(NUSE)
331:   HEATM1 = NUSE
332:   HEATM2 = NUSE
333:   go to 180
334: c-----energy interval.
335: 170  HEATM1 = NUSE
336:   HEATM2 = NUSE - 1
337:   SIGMID = ((ECOLD(NUSE)-EMID)*XCCOLD(HEATM2)
338:   &      +(EMID-ECOLD(HEATM2))*XCCOLD(NUSE)) /
339:   &      (ECOLD(NUSE)-ECOLD(HEATM2))
340: c-----select broadening method.
341: 180  if ( YMID.le.ATOP ) then
342:     call BROADL( YMID, SIGMID, XCMID, HEATM1, HEATM2, IWRKMX,
343:   &      NPMAX, XCCOLD, YCOLD )
344:   else if ( YMID.gt.HOTSYS ) then
345:     if ( IVPOP.eq.1 ) then
346:       call DCLEA( WBROAD, 12*IMAXD, ZERO )
347:       call BROADS( YMID, SIGMID, XCMID, HEATM1, HEATM2, IWRKMX,
348:   &      NPMAX, IMAXD, IMAXX, DATOP, XCCOLD, DCOLD, YCOLD,
349:   &      WBROAD(1,1), WBROAD(1,2), WBROAD(1,3),
350:   &      WBROAD(1,4), WBROAD(1,5), WBROAD(1,6),
351:   &      WBROAD(1,7), WBROAD(1,8), WBROAD(1,9),
352:   &      WBROAD(1,10), WBROAD(1,11), WBROAD(1,12) )
353:     else
354:       call SROADS( YMID, SIGMID, XCMID, HEATM1, HEATM2, IWRKMX,
355:   &      NPMAX, XCCOLD, DCOLD, YCOLD )
356:     end if
357:   else if ( IVPOP.eq.1 ) then
358:     call DCLEA( WBROAD, 14*IMAXD, ZERO )
359:     call BROADH( YMID, SIGMID, XCMID, HEATM1, HEATM2, IWRKMX,
360:   &      NPMAX, IMAXD, IMAXX, DATOP, XCCOLD, DCOLD, YCOLD,
361:   &      WBROAD(1,1), WBROAD(1,2), WBROAD(1,3), WBROAD(1,4),
362:   &      WBROAD(1,5), WBROAD(1,6), WBROAD(1,7), WBROAD(1,8),
363:   &      WBROAD(1,9), WBROAD(1,10), WBROAD(1,11),
364:   &      WBROAD(1,12), WBROAD(1,13), WBROAD(1,14) )
365:   else
366:     call SROADH( YMID, SIGMID, XCMID, HEATM1, HEATM2, IWRKMX,
367:   &      NPMAX, XCCOLD, DCOLD, YCOLD )
368:   end if
369: c
370: c   check for convergence.
371: c
372: c-----do not further sub-divide energy ranges with negative cross
373: c-----sections (such data has no physical meaning and additional
374: c-----time should not be spend on it).
375:   if ( XCMID.lt.0.0 ) go to 220
376: c-----no convergence if cross section changes by more than 40 per-cent.
377:   if ( XCHOT(HOT3M1).gt.1.4*XCHOT(HOT3)
378:   &      .or. XCHOT(HOT3).gt.1.4*XCHOT(HOT3M1) ) go to 210
379: c-----define cross section at mid-point by linear interpolation.
380:   XCLIN =
381:   &      ((EHOT(HOT3)-EMID)*XCHOT(HOT3M1)+(EMID-EHOT(HOT3M1))*
382:   &      XCHOT(HOT3)) / (EHOT(HOT3)-EHOT(HOT3M1))
383: c-----if requested, define energy dependent convergence criteria.
384:   if ( KERR3C.ne.0 ) call ERROKC( EMID )
385: c-----use more stringent convergence criteria if iterating toward....
386:   if ( XCMID.lt.XCHOT(HOT3) ) then
387: c-----minimum.
388:     if ( XCMID.lt.XCHOT(HOT3M1) ) go to 190
389:     else if ( XCMID.ne.XCHOT(HOT3) ) then
390: c-----maximum.

```

src/artcore/broadn.f

```
391:          if ( XCMID.gt.XCHOT(HOT3M1) ) go to 190
392:          end if
393: c-----standard linear interpolation.
394:          if ( ABS(XCMID-XCLIN).gt.ABS(ERXC3C*XCMID) ) go to 200
395:          go to 220
396: c-----more stringent linear interpolation toward minimum/maximum.
397: 190      if ( ABS(XCMID-XCLIN).le.ABS(ERXC30*XCMID) ) go to 220
398: c-----low cross section convergence test.
399: 200      if ( ABS(XCMID).lt.XCMIN.and.ABS(XCHOT(HOT3M1)).lt.XCMIN )
400:          & go to 220
401: c-----no convergence. save point and half interval.
402: 210      if ( ISAVE.lt.IADDMX ) ISAVE = ISAVE + 1
403:          ESAVE(ISAVE) = EHOT(HOT3)
404:          XCSAVE(ISAVE) = XCHOT(HOT3)
405:          EHOT(HOT3) = EMID
406:          XCHOT(HOT3) = XCMID
407:          go to 140
408: c-----convergence. if thinning will be performed save midpoint and
409: c-----end point. if no thinning only save end point.
410: 220      if ( NOTHING.le.0 ) then
411:          EHOT(HOT3+1) = EHOT(HOT3)
412:          XCHOT(HOT3+1) = XCHOT(HOT3)
413:          EHOT(HOT3) = EMID
414:          XCHOT(HOT3) = XCMID
415:          HOT3M1 = HOT3
416:          HOT3 = HOT3 + 1
417:      end if
418: c-----convergence. if over 2 pages of broadened data thin it.
419: 230      if ( HOT3.ge.NPMAX ) call THINIT( NPMAX, EHOT, XCHOT )
420:          if ( HOT3.gt.NPMAX ) then
421:              write(NSYSO,*) ' ** hot3 npmax (broadn) : ', HOT3, NPMAX
422:              stop 999
423:          end if
424: c-----if end of interval proceed to next interval. otherwise use last
425: c-----point generated.
426:          if ( ISAVE.gt.KWRKMX ) KWRKMX = ISAVE
427: c
428:          if ( ISAVE.gt.0 ) then
429:              HOT3M1 = HOT3
430:              HOT3 = HOT3 + 1
431:              EHOT(HOT3) = ESAVE(ISAVE)
432:              XCHOT(HOT3) = XCSAVE(ISAVE)
433:              ISAVE = ISAVE - 1
434:              go to 140
435:          end if
436: c-----end of interval. set up for next interval by placing end of last
437: c-----interval at beginning of next interval.
438:          NLOW = HEATM1
439:          ENEXT = DEMAX*EHOT(HOT3)
440: c-----continue in loop if preceding (not current) point was used or
441: c-----if still iterating in energy range due to energy point spacing.
442:          if ( NHIGH.ne.IHEAT ) go to 100
443: c-----define beginning of next tabulated energy interval.
444: 240      HOTEND = 0
445:          NLOW = IHEAT
446:          SIGY = XCIH
447:          ENEXT = DEMAX*EHOT(HOT3)
448: c-----save last cold energy and cross section.
449: 250      EIHM1 = EIH
450:          XCIHM1 = XCIH
451: c
452: 260      continue
453: cm      write(6,*) ' *** iwrkmx kwrkmx : ',iwrkmx,kwrkmx
454: cm      write(6,*) ' *** kerr3c hot3 : ',kerr3c,hot3
455: c
456:          return
457:          end
```

src/artcore/broads.f

```

1: cm      subroutine broads(y,sigy,xcy,heatm1,heatm2,iwrkmx)
2:      subroutine BROADS( Y,      SIGY,  XCY,   HEATM1,HEATM2,IWRKMx,
3:      &      NPMAX, IMAXD, IMAXX, DATOP, XCCOLD,DCOLD,
4:      &      YCOLD, B,    R,    A,    BMY,  EXBMY,
5:      &      ERFCAM,F2B,   F4A,   F4B,   SNEW, DCNEW, YYMAA
6:      &      )
7: c
8: c      high energy doppler broadening routine. this routine will
9: c      be used to doppler broaden all cross sections at energies
10: c      where ae/kt is greater than one million. any point with a lower
11: c      energy will already have been doppler broadened by routine
12: c      broadl or broadh.
13: c
14: c      for ae/kt less than or equal to 36 both exponentials in the
15: c      doppler broadening kernel must be considered. for higher energies
16: c      the second exponential may be ignored, which simplifies the
17: c      doppler broadening.
18: c
19: c      for ae/kt greater than 1,000,000 it is possible to assume the
20: c      term (x/y)**2 in the doppler integral is just unity (since x only
21: c      varies from y-4 to y+4, for y greater than 1000 this term is
22: c      essentially constant since it varies from (996/1000)**2 to
23: c      (1004/1000)**2.= 1.0 +/- 0.008.
24: c
25: c      the routine has been designed with no subroutine calls
26: c      in order to minimize running time. the arithmetic
27: c      statement functions ration(a) and erfc(r,experf) will
28: c      be compiled as in line coding by virtually any fortran
29: c      compiler, and as such do not represent function calls.
30: c
31: caddb6th
32:      save IMAX
33: cend
34:      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HEATME, LOOPS,
35:      &      HIHEAT, HEATM1, HEATM2, HOT3, HOT3M1
36: c***** double *****
37: c      double precision a1,a2,a3,a4,a5,y,a,b,r,bmy,yma,yya,dummy1,
38: c      1 dummy2,f2a,f2b,f4a,f4b,expam,expbm,erfcam,erfcbm,zd,
39: c      2 expperf,zexp,ration,erfc,ovpi,ovpi2,zero,half,one,twop5,
40: c      3 ycold,cona
41: c***** double *****
42:      real*8 A1, A2, A3, A4, A5, Y, YYA, DUMMY1, DUMMY2,
43:      &      ZERO, HALF, ONE, TWOP5, CONA
44: c***** double *****
45:      common /INDEX/  COLD1,  COLD2,  COLD1P, COLD2P, HOT1,  HOT2,
46:      &      HOT3,    HOT3M1, N2IN,  N2TOT, N2SCR
47:      common /INDICE/  HEATME, LOOPS,  LOHEAT, HIHEAT
48: c
49: c      common/comm2/xccold(6012)
50: c      common/comm3/dcold(6012)
51: c      common/comm4/ycold(6012)
52: c
53:      real XCCOLD(NPMAX)
54:      real DCOLD(NPMAX)
55:      real*8 YCOLD(NPMAX)
56: c
57: c
58: c
59:      real*8 B(IMAXD), R(IMAXD), A(IMAXD), BMY(IMAXD)
60:      real*8 EXBMY(IMAXD), ERFCAM(IMAXD)
61:      real*8 F2B(IMAXD), F4B(IMAXD)
62:      real*8 SNEW(IMAXD), DCNEW(IMAXD)
63:      real*8 YYMAA(IMAXD), F4A(IMAXD)
64: c
65:      real*8 DATOP, DEL

```

```

66: c
67: c      real*8 OVPI, OVPI2
68: c      common /CONTAC/  OVPI,   OVPI2,  ATOP
69: c
70: c      include 'INC/_CONST1'
71: c
72: c      complementary error function definition.
73: c
74: c-----define constants for complementary error function
75: c***** double *****
76: c      data A1 /0.254829592D+00/
77: c      data A2 /-0.284496736D+00/
78: c      data A3 /1.421413741D+00/
79: c      data A4 /-1.453152027D+00/
80: c      data A5 /1.061405429D+00/
81: c      data CONA /3.275911D-01/
82: c      data ZERO /0.0D+00/
83: c      data HALF /0.5D+00/
84: c      data ONE /1.0D+00/
85: c      data TWOP5 /2.5D+00/
86: c
87: c      parameter( A1 =0.254829592D+00)
88: c      parameter( A2 =-0.284496736D+00)
89: c      parameter( A3 =1.421413741D+00)
90: c      parameter( A4 =-1.453152027D+00)
91: c      parameter( A5 =1.061405429D+00)
92: c      parameter( CONA =3.275911D-01)
93: c      parameter( ZERO =0.0D+00)
94: c      parameter( HALF =0.5D+00)
95: c      parameter( ONE =1.0D+00)
96: c      parameter( TWOP5 =2.5D+00)
97: c***** double *****
98: c***** single *****
99: c      data a1/0.254829592e+00/
100: c      data a2/-0.284496736e+00/
101: c      data a3/1.421413741e+00/
102: c      data a4/-1.453152027e+00/
103: c      data a5/1.061405429e+00/
104: c      data cona/3.275911e-01/
105: c      data zero/0.0/
106: c      data half/0.5/
107: c      data one/1.0/
108: c      data twop5/2.5/
109: c      zexp(zd)=exp(zd)
110: c***** single *****
111: c-----define arithmetic statement function for rational argument
112: c-----of complementary error function.
113: cm      ration(a)=one/(one+cona*a)
114: c-----define arithmetic statment function for complementary
115: c-----statment function.
116: cm      erfc(r,experf)=((((a5*r+a4)*r+a3)*r+a2)*r+a1)*r*experf
117: c
118: c
119: c      set up loop to doppler broaden cross sections.
120: c
121: c-----define all required constants for point.
122: c      YYA      = OVPI2*Y
123: c-----initialize integrals.
124: c      DUMMY1 = ZERO
125: c      DUMMY2 = ZERO
126: c*
127: c*      integrate over all energy intervals above the current energy
128: c*      point.
129: c*
130: c-----no intervals above the last point.

```

src/artcore/broads.f

```

131:      if ( HEATM2.ge.COLD2P ) then
132: c-----y is at upper end of tabulated range. continue cross section as
133: c-----1/v to ycold=infinity.
134:      DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*(Y+OVPI) /(Y*Y)
135:      go to 160
136:      end if
137: c
138: c-----initialize integrals.
139: c
140: cm    a      = y
141: cm    siga   = sigy
142: cm    erfcam = one
143: cm    f4a    = twop5 + yya
144: cm    A(1)   = Y
145: cm    SNEW(1) = SIGY
146: cm    ERFCAM(1) = ONE
147: cm    F4A(1) = TWOP5 + YYA
148: c
149: c-----set up loop over intervals above current one.
150: c
151:      IMAX = 0
152: c
153: c do 20 loops = heatm2,hiheat
154: c    b      = ycold(loops+1)
155: c    bmy    = b - y
156: c-----only extend range of integration up to atop units above y.
157: c    if(bmy.gt.atop) then
158: c      bmy=atop
159: c      b=y+atop
160: c    endif
161:      IMAX = 0
162:      do 100 LOOPS = HEATM2, HIHEAT
163:      IMAX = IMAX + 1
164:      DEL = YCOLD(LOOPS+1) - Y
165:      BMY(IMAX) = DEL
166: c-----only extend range of integration up to datop units above y.
167: c    if ( DEL.ge.DATOP ) then
168: c      BMY(IMAX) = DATOP
169: c      go to 110
170: c    end if
171: c
172: c 100 continue
173: c
174: c 110 continue
175: c    if ( IMAX.gt.IMAXX ) stop 999
176: c
177: c    LAST = HEATM2 + IMAX - 1
178: c    JMAX = 0
179: c
180: c do 120 LOOPS = HEATM2, LAST
181: c   JMAX = JMAX + 1
182: c   B(JMAX) = BMY(JMAX) + Y
183: c   A(JMAX+1) = B(JMAX)
184: c   SNEW(JMAX+1) = XCCOLD(LOOPS+1)
185: c   DCNEW(JMAX) = DCOLD(LOOPS)
186: c 120 continue
187: c
188: c-----define constants for this point.
189: c    r      = ration(bmy)
190: c    expbm  = zexp(-bmy*bmy)
191: c
192: c    YYMAA(1) = ZERO
193: c
194: c do 130 I = 1, IMAX
195: c   R(I) = ONE/(ONE+CONA*BMY(I))

```

```

196:      EXBMY(I) = EXP(-BMY(I)*BMY(I))
197:      YYMAA(I+1) = (Y-A(I+1))*(Y+A(I+1))
198: c 130 continue
199: c    erfcbm = erfc(r,expbm)
200: c    f2b = ovpi*(b+y)*expbm
201: c    f4b = ((y-a)*(y+a)+twop5)*erfcbm + f2b
202: c do 140 I = 1, IMAX
203: c   ERFCAM(I+1) = (((A5*R(I)+A4)*R(I)+A3)*R(I)+A2)*R(I)+A1)*R(I)*
204: c   & EXBMY(I)
205: c   F2B(I) = OVPI*(B(I)+Y)*EXBMY(I)
206: c   F4B(I) = F2B(I) + (YYMAA(I)+TWOP5)*ERFCAM(I+1)
207: c   F4A(I+1) = F2B(I) + (YYMAA(I+1)+TWOP5)*ERFCAM(I+1)
208: c 140 continue
209: c-----add contribution from current interval.
210: c    dummy1 = dummy1 + siga*(erfcam-erfcbm)
211: c    dummy2 = dummy2 + dcold(loops)*(f4a-f4b)
212: c
213: c do 150 I = 1, IMAX
214: c   DUMMY1 = DUMMY1 + SNEW(I)*(ERFCAM(I)-ERFCAM(I+1))
215: c   DUMMY2 = DUMMY2 + DCNEW(I)*(F4A(I)-F4B(I))
216: c 150 continue
217: c-----test for end of range of integration.
218: c    if(bmy.ge.atop) go to 40
219: c-----save values from last interval.
220: c    a = b
221: c    siga = xccold(loops+1)
222: c    erfcam = erfcbm
223: c    f4a = ((y-a)*(y+a)+twop5)*erfcbm+f2b
224: c    imax = imax + 1
225: c 20 continue
226: c-----continue cross section as 1/v to ycold=infinity.
227: c    dummy1=dummy1+xccold(cold2p)*ycold(cold2p)*(y*erfcbm+ovpi*expbm)/
228: c    1 (y*y)
229: c
230: c    if ( BMY(IMAX).lt.DATOP ) then
231: c      DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*
232: c      & (Y*ERFCAM(IMAX+1)+OVPI*EXBMY(IMAX)) /(Y*Y)
233: c    end if
234: c
235: c
236: c del go to 40
237: c*
238: c* integrate over all energy intervals below the current energy
239: c* point.
240: c*
241: c-----no intervals below current first point.
242: c 160 continue
243: c
244: c    if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
245: c
246: c    if ( HEATM1.le.COLD1P ) then
247: c-----y is at lower end of tabulated range. continue cross section as
248: c-----1/v to ycold=0.0
249: c      DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*(Y-OVPI) /(Y*Y)
250: c-----define broadened cross section.
251: c      XCY = HALF*(DUMMY1+DUMMY2)
252: c      return
253: c    end if
254: c
255: c-----re-initialize integrals to zero distance values
256: c
257: c    b = y
258: c    sigb = sigy
259: c    erfcbm = one
260: c    f4b = twop5-yya

```

src/artcore/broads.f

```

261:      B(1)      = Y
262:      SNEW(1) = SIGY
263:      ERFCAM(1) = ONE
264:      F4B(1)   = TWOP5 - YYA
265: c-----set up loop over intervals below current point.
266:      LOOPS     = HEATM1
267:      IMAX      = 0
268: c
269: c      do 60 ll = loheat , heatm1
270: c      loops    = loops - 1
271: c      a        = ycold(loops)
272: c      yma      = y - a
273: c-----only extend range of integration down to atop units below y.
274: c      if(yma.ge.atop) then
275: c          yma = atop
276: c          a   = y - atop
277: c      endif
278: c      do 170 LL = LOHEAT, HEATM1
279: c          IMAX = IMAX + 1
280: c          LOOPS = LOOPS + 1
281: c          DEL   = Y - YCOLD(LOOPS)
282: c          BMY(IMAX) = DEL
283: c-----only extend range of integration up to datop units above y.
284: c          if ( DEL.ge.DATOP ) then
285: c              BMY(IMAX) = DATOP
286: c              go to 180
287: c          end if
288: c      170 continue
289: c
290: c      180 continue
291: c          if ( IMAX.gt.IMAXX ) stop 999
292: c
293: c          LOOPS = HEATM1
294: c          JMAX  = 0
295: c
296: c          LAST  = LOHEAT + IMAX - 1
297: c          do 190 LL = LOHEAT, LAST
298: c              JMAX = JMAX + 1
299: c              LOOPS = LOOPS + 1
300: c              A(JMAX) = Y - BMY(JMAX)
301: c              B(JMAX+1) = A(JMAX)
302: c              SNEW(JMAX+1) = XCCOLD(LOOPS)
303: c              DCNEW(JMAX) = DCOLD(LOOPS)
304: c          190 continue
305: c
306: c-----define constants for this point.
307: c      50 r      = ration(yma)
308: c      expam     = zexp(-yma*yma)
309: c
310: c      YYMAA(1)  = ZERO
311: c      do 200 I = 1, IMAX
312: c          R(I)   = ONE/(ONE+CONA*BMY(I))
313: c          EXBMY(I) = EXP(-BMY(I)*BMY(I))
314: c          YYMAA(I+1) = (Y-B(I+1))*(Y+B(I+1))
315: c      200 continue
316: c
317: c      erfcam    = erfc(r,expam)
318: c      f2a       = ovpi*(y+a)*expam
319: c      f4a       = ((y-b)*(y+b)+twop5)*erfcam-f2a
320: c      do 210 I = 1, IMAX
321: c          ERFCAM(I+1) = (((A5*R(I)+A4)*R(I)+A3)*R(I)+A2)*R(I)+A1)*R(I)*
322: c      &          EXBMY(I)
323: c          F2B(I)      = OVPI*(Y+A(I))*EXBMY(I)
324: c          F4A(I)      = (YYMAA(I)+TWOP5)*ERFCAM(I+1) - F2B(I)
325: c          F4B(I+1)    = (YYMAA(I+1)+TWOP5)*ERFCAM(I+1) - F2B(I)

```

```

326:      210 continue
327: c-----add contribution from current interval.
328: c      dummy1 = dummy1 + sigb*(erfcam-erfcam)
329: c      dummy2 = dummy2 + dcold(loops)*(f4b-f4a)
330: c
331: c      do 220 I = 1, IMAX
332: c          DUMMY1 = DUMMY1 + SNEW(I)*(ERFCAM(I)-ERFCAM(I+1))
333: c          DUMMY2 = DUMMY2 + DCNEW(I)*(F4B(I)-F4A(I))
334: c      220 continue
335: c
336: c-----test for end of range of integration.
337: c      if(yma.ge.atop) go to 80
338: c-----save values from last integral.
339: c      b      = a
340: c      sigb    = xccold(loops)
341: c      erfcbm  = erfcam
342: c      f4b     = ((y-b)*(y+b)+twop5)*erfcam-f2a
343: c      imax    = imax + 1
344: c      60 continue
345: c-----continue cross section as 1/v to ycold=0.0
346: c      dummy1=dummy1+xccold(coldlp)*ycold(coldlp)*(y*erfcam-ovpi*expam)/
347: c      1      (y*y)
348: c
349: c      if ( BMY(IMAX).lt.DATOP ) then
350: c          DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*
351: c      &          (Y*ERFCAM(IMAX+1)-OVPI*EXBMY(IMAX)) / (Y*Y)
352: c      end if
353: c
354: c-----define broadened cross section.
355: c
356: c      80 xcy=half*(dummy1+dummy2)
357: c
358: c      XCY      = HALF*(DUMMY1+DUMMY2)
359: c      if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
360: c
361: c      return
362: c      end

```


src/artcore/cardo.f

```
1:      subroutine CARDO( C1H,  C2H,  L1H,  L2H,  N1H,  N2H )
2: c
3: c      write endf/b card.
4: c
5: c***** character *****
6:      character*4 KSIGN
7: c***** character *****
8:      integer OTAPE
9:      common /UNITS/      INP,      OUTP,      ITAPE,  OTAPE,  SCR
10:     common /NORMF/      XNORM(6),      KEXP(6)
11:     common /NORMC/      KSIGN(6)
12:     common /HEADR/      DUMMY(6),      MATH,      MFH,      MTH,      NOSEQ
13:     if ( OTAPE.le.0 ) return
14:     if ( MTH.le.0 ) then
15:         call SENDO
16:         return
17:     end if
18: c-----convert floating point numbers to standard output form.
19:     call NORMX( C1H, XNORM(1), KSIGN(1), KEXP(1) )
20:     call NORMX( C2H, XNORM(2), KSIGN(2), KEXP(2) )
21: c-----eliminate -0
22: c     if ( IABS(L1H).le.0 ) L1H = 0
23: c     if ( IABS(L2H).le.0 ) L2H = 0
24: c     if ( IABS(N1H).le.0 ) N1H = 0
25: c     if ( IABS(N2H).le.0 ) N2H = 0
26: c-----output card image.
27:     write(OTAPE,7000) (XNORM(I),KSIGN(I),KEXP(I),I=1,2), L1H, L2H,
28:     &      N1H, N2H, MATH, MFH, MTH, NOSEQ
29:     NOSEQ = NXTSEQ(NOSEQ)
30:     return
31: 7000 format(2(F8.5,A1,I2),4I11,I4,I2,I3,I5)
32:     end
```

src/artcore/clea.f

```
1:      subroutine CLEA( A,      LENG,  ASET )
2: c
3:      real A(LENG)
4: c
5:      if ( LENG.le.0 ) return
6: c
7:      do 100 I = 1, LENG
8:          A(I)      = ASET
9:      100 continue
10:     return
11: end
12: c
13:     subroutine ICLEA( IA,      LENG,  IASET )
14:     integer IA(LENG)
15: c
16:     if ( LENG.le.0 ) return
17: c
18:     do 100 I = 1, LENG
19:         IA(I)      = IASET
20:     100 continue
21:     return
22: end
```

```
src/artcore/copout.f
```

```

1: subroutine COPOUT( NPMAX, EHOT, XCHOT )
2: c
3: c      this routine is designed to copy the section of broadened and/or
4: c      thinned data from the core or scratch file to the result file in
5: c      the endf/b format.
6: c
7: c***** character *****
8: cm      character*4 messag,fmthol,ksign
9:      character*4 FMTHOL, KSIGN
10: c***** character *****
11: c***** integer *****
12: c      integer fmthol
13: c***** integer *****
14:      integer OTAPE, OUTP, SCR, COLD1, COLD2, COLD1P, COLD2P, HOT1,
15:      & HOT2, HOT3, HOT3M1
16: c***** double *****
17: cdel double precision ehot
18: c***** double *****
19:      common /UNITS/ INP, OUTP, ITAPE, OTAPE, SCR
20:      common /HEADR/ C1H, C2H, L1H, L2H, N1H, N2H,
21:      & MATH, MFH, MTH, NOSEQ
22:      common /LEADER/ C1, Q, L1, L2, N1, N2
23:      common /WHATZA/ IZA
24:      common /INDEX/ COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2,
25:      & HOT3, HOT3M1, N2IN, N2TOT, N2SCR
26:      common /PAGER/ NPAGE, NPT2, NPT3, NP1P1, NP2P1
27:      common /NORMF/ XNORM(6), KEXP(6)
28:      common /NORMC/ KSIGN(6)
29:      common /FILLER/ N2LEFT, TOOHI, ITHRES, LOAD1, LOAD2
30:      common /THRESH/ ETHRES, EMIN
31:      common /FLAGS/ MINUS3
32:      common /MATTOT/ MATIN, MATOUT
33:      common /TEMPO/ TEMP3, IVERSE
34:      common /EXTEND/ MESS, DTMAX
35:      common /HOLFMT/ FMTHOL
36: c
37: cm      common/comm5/ehot(6012)
38: cm      common/comm6/xchot(6012)
39: c
40: c
41:      real*8 EHOT(NPMAX)
42:      real XCHOT(NPMAX)
43: c
44: c
45: cm      dimension messag(3,2),nbto(1),into(1)
46:      integer NBTO(1), INTO(1)
47: c
48: c-----define cross section extension message.
49: cm      data messag/' ',' ',' ','exte','nsio','n'/
50: c-----define linear-linear interpolation law.
51:      data INTO /2/
52: c
53: c      define table size, output beginning of tabl record and define
54: c      where data is (i.e. on scratch or in core).
55: c
56: c-----initialize negative cross section flag off.
57:      MINUS3 = 0
58: c-----output tabl lead card (section head card already output in main)
59:      N1 = 1
60: c
61:      call CARDO( C1, Q, L1, L2, N1, N2TOT )
62: c-----output interpolation law.
63:      NBTO(1) = N2TOT
64:      call TERPO( NBTO, INTO, 1 )
65: c-----define number of data points in core (all if core resident, or

```

```

66: c-----three pages if data is on scratch).
67:      IOUT      = N2TOT
68: c
69: c      set up loop over pages of data.
70: c
71: c-----if threshold replaced due to doppler broadening insure that if
72: c-----first tabulated energy is above minimum energy of interest the
73: c-----cross section is zero at the new threshold.
74: c
75:      if ( ITHRES.gt.0 ) then
76:          ITHRES = 0
77:          if ( EHOT(1).gt.1.1*EMIN ) XCHOT(1) = 0.0
78:      end if
79: c
80: c-----output points in endf/b format.
81: c
82:      call POINTV( EHOT, XCHOT, IOUT )
83: c
84: c      output section report and increment point counts for material.
85: c      print warning if output contains any negative cross sections.
86: c
87:      call normx(temp3,xnorm(1),ksign(1),kexp(1))
88:      call normx(q,xnorm(2),ksign(2),kexp(2))
89: c
90:      write(outp,5900) iza,math,mth,fmthol,
91:      1 (xnorm(1),ksign(1),kexp(1),l=1,2),n2in,n2tot,
92:      2 messag(1,mess),messag(2,mess)
93: c
94:      matin = matin + n2in
95:      matout =matout+ n2tot
96: c
97: c-----print warning message if this section contains negative cross
98: c-----sections.
99: c      if(minus3.gt.0) write(outp,5910)
100: c
101: c5900 format(2x,i6,2i5,2x,a2,3x,2(1x,f8.5,a1,i2),2i7,2x,2a4,a1)
102: c5910 format(19x,45hwarning...the above section contains negative,
103: c      1 16h cross sections.)
104: c
105: c *** end of process
106: c
107:      return
108:      end

```

src/artcore/dclea.f

```
1:      subroutine DCLEA( A,      LENG, B )
2: c*****
3: c      clea is used for value set of real      1-dimension array.
4: c*****
5:      real*8 A(LENG), B
6: c
7:      do 100 I = 1, LENG
8:          A(I)      = B
9:      100 continue
10:      return
11:      end
```

DATA

src/artcore/dimchk.f

```
1:      subroutine DIMCHK( SUBNAM,LMAX,  LIMIT )
2: C=<ARTCORE>=====
3: C Purpose : dimension size check for variable dimension method
4: C Called in : ARTPRC
5: C=====
6:      include 'INC/_MAINIO'
7: c
8:      character*6 SUBNAM
9: c
10:     if ( LMAX.le.LIMIT ) return
11: c
12:     MNEED = (LMAX-LIMIT)
13:     write(NSYSO,7000) SUBNAM, LIMIT, MNEED
14:     stop 999
15: c
16: 7000 format(/1X,' error stop at subroutine(',A6,') !!!'/
17: &          1X,' memory over !!!'/
18: &          /1X,' memory reserved ----- ',I18,' words'/
19: &          /1X,' lack of memory ----- ',I18,' words'/)
20: c
21:     end
```

src/artcore/dmain.f

```

1:      subroutine ARTPRC( ENERGY,XMESH, YMESH, XSANS, ADATA, IDATA, A,
2:      &                  IA,          MEMORY,TITLE, IVPOP, RTEMP, NIN,
3:      &                  IVERSN )
4: C=<ARTCORE>=====
5: C MVPART: MVP arbitrary temperature neutron library processor
6: C-----
7: C Purpose : Input single library file and process it.
8: C Called in : ARTLIB(MVP), MAIN(MVPART)
9: C Calls : FILE3, UNRESR, LISU3R, THERMJ, LISTHS, THERM6, LISTH6, INTEFF
10: C=====
11: C
12:      include 'INC/_PARAM'
13: C
14:      character*4 FMTHOL
15: C
16:      common /HOLFM/ FMTHOL
17: C
18:      include 'INC/_MAINIO'
19: C
20:      common /CONTRL/ MATD, ZA, ELUNR, EHUNR, CRSMIN, LEMORY,
21:      & IUNRES, INTUNR, IDUMP, IDBG
22: C
23:      common /HEAD1/ NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA,
24:      & NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITHERM,
25:      & ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1,
26:      & LNUD, NNUD, NNF, LSNUD, LASYM, NOTDOP,
27:      & KDUMMY(68),
28:      & ATW, TLAB, ETHERM, ESAB, ELOW,
29:      & EHI, TSTAR, RDUMMY(43)
30: C
31:      common /HEAD2/ MTINFO(MTMAX), MTPAR(MTMAX), ISTMT(MTMAX),
32:      & IENDMT(MTMAX), NEUMT(MTMAX), LCTMT(MTMAX),
33:      & NEANG(MTMAX), NKF5(MTMAX), NEF5(MTMAX),
34:      & MTTHR(MTMAX), LSTF3(MTMAX), LSTF4(MTMAX),
35:      & LSTF5(MTMAX), NEF5S(NKMAX,MTMAX),
36:      & LFF5S(NKMAX,MTMAX), LSTF5S(NKMAX,MTMAX),
37:      & QVAL(MTMAX), QVAL2(MTMAX)
38: C
39:      common /HEAD3/ NOGAM, NOGAM3, MTGAM(MTMAX), NGTYPE(MTMAX),
40:      & NKGAM(MTMAX), N14GAM(MTMAX), N15GAM(MTMAX),
41:      & LSTGAM(MTMAX), MTTOG(MTMAX), NEGYLD(MTMAX),
42:      & NEF5G(MTMAX), NKF5G(MTMAX), LCTMTG(MTMAX), NGREC,
43:      & LENGAM(MXREC), NCAP, NCAPG, MTCAP(MTMAX),
44:      & MTCAPG(MTMAX), EG1L(MTMAX), EG2U(MTMAX),
45:      & EGRANG(2,MTMAX)
46: C
47:      common /F6CONT/ LF6, LSTF6, MTTOF6(MTMAX), EF61L(MTMAX),
48:      & EF62U(MTMAX), F6RANG(2,MTMAX), LSSF, LSTDOP
49: C
50:      real A(MEMORY)
51:      integer IA(MEMORY)
52: C
53:      real ADATA(*)
54:      integer IDATA(*)
55:
56:      real ENERGY(NPTS)
57: C
58:      real XMESH(NPTS), YMESH(NPTS), XSANS(NPTS)
59: C
60:      character*16 MATT
61:      character*120 TITLE
62:      character*8 HDATE
63: C
64: C
65: C

```

```

66:      call CPUTM( TCP0 )
67:      call TOKEI( T00, 0 )
68: C
69:      if ( IVPOP.eq.0 ) then
70:      write(NSYSO,*) ' >>> Doppler broadening in scalar mode <<<'
71:      else
72:      write(NSYSO,*) ' >>> Doppler broadening in vector mode <<<'
73:      end if
74: C
75: C
76: C/#IF INLINEEXP
77:      call IEXPIN( NSYSO )
78: C/#ENDIF
79: C
80: C ... read library
81: C
82:      rewind NIN
83: C
84: C *****
85: C **** Read record 1 *****
86: C *****
87: C
88:      if ( IVERSN.eq.2 ) then
89:      MATT = ' '
90:      read(NIN) MATT(1:8), HDATE, TITLE,
91:      & NPTS, NTDATA, NUNR, NUNR2, NLEV,
92:      & NBINA, NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITHERM,
93:      & ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF,
94:      & LSNUD, NCAP, NCAPG, LF6, LSTF6, LSSF, LSTDOP, LASYM,
95:      & NOTDOP,
96:      & (KDUMMY(I),I=1,2), ATW, TLAB, ETHERM, ESAB, ELOW, EHI,
97:      & TSTAR, (RDUMMY(I),I=1,12)
98:      else if ( IVERSN.eq.3 ) then
99:      read(NIN) MATT(1:16), TITLE,
100:      & NPTS, NTDATA, NUNR, NUNR2, NLEV,
101:      & NBINA, NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITHERM,
102:      & ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF,
103:      & LSNUD, NCAP, NCAPG, LF6, LSTF6, LSSF, LSTDOP, LASYM,
104:      & NOTDOP,
105:      & (KDUMMY(I),I=1,68), ATW, TLAB, ETHERM, ESAB, ELOW, EHI,
106:      & TSTAR, (RDUMMY(I),I=1,43)
107:      HDATE = TITLE(113:120)
108:      end if
109: C
110: C *****
111: C **** Read record 2 *****
112: C *****
113:      read(NIN) MTINFO, MTPAR, ISTMT, IENDMT, NEUMT, LCTMT, NEANG, NKF5,
114:      & NEF5, MTTHR, LSTF3, LSTF4, LSTF5, QVAL, QVAL2, LSTGAM,
115:      & MTTOG, NEGYLD, NEF5G, NKF5G, LCTMTG, MTCAP, MTCAPG, EG1L,
116:      & EG2U
117: C
118:      TBASE = TLAB ! This sentence must be here for doppler-scattering.
119:      read(NIN) (ADATA(I),I=1,NTDATA)
120:      rewind NIN
121: C
122:      ZA = RDUMMY(1)
123:      MATD = RDUMMY(2)
124:      FMTHOL = 'v'
125:      if ( LF6.gt.0 ) FMTHOL = 'vi'
126: C
127:      IST = 0
128:      IEND = 0
129:      *VOCL LOOP, NOVREC
130:      do 100 I = 1, NPTS

```

src/artcore/dmain.f

```

131:      ENERGY(I) = ADATA(I)
132: C
133: C      .... keep elastic scattering cross section
134: C
135:      ADATA(NTDATA+I) = ADATA(LSTF3(2)+I-1) ! dopp-scat
136: 100 continue
137: C
138:      if ( LF6.eq.1 ) then
139:      ISW = LSTF6 - 1
140:      do 110 I = 1, NMT
141:      ISW = ISW + 1
142:      MTTOF6(I) = IDATA(ISW)
143: 110 continue
144: C
145:      do 120 I = 1, NMT
146:      ISW = ISW + 1
147:      EF61L(I) = ADATA(ISW)
148: 120 continue
149: C
150:      do 130 I = 1, NMT
151:      ISW = ISW + 1
152:      EF62U(I) = ADATA(ISW)
153: 130 continue
154: C
155:      end if
156: C
157: C
158: C ***** added for t-free version
159: C
160:      ITFREE = 0
161:      LENTHS = 0
162:      LENU3R = 0
163:      LSTTHS = 0
164:      LSTU3R = 0
165:      LSTU3 = 0
166:      LSTTSC = 0
167:      KDOP = 0
168: C
169:      if ( LSTDOP.gt.0 ) then
170:      ITFREE = 1
171:      LENTHS = IDATA(LSTDOP)
172:      LENU3R = IDATA(LSTDOP+1)
173:      LSTTHS = 0
174:      if ( LENTHS.gt.0 ) LSTTHS = LSTDOP + 2
175:      LSTU3R = 0
176:      if ( LENU3R.gt.0 ) then
177:      LSTU3R = LSTDOP + 2
178:      if ( LENTHS.gt.0 ) LSTU3R = LSTTHS + LENTHS
179:      end if
180:      end if
181: C
182:      NTU3R = 0
183:      NURT = 0
184:      NBNU3R = 0
185:      IFISUN = 0
186:      if ( LSTU3R.gt.0 ) then
187:      NTU3R = IDATA(LSTU3R)
188:      NURT = IDATA(LSTU3R+1)
189:      NBNU3R = IDATA(LSTU3R+2)
190:      IFISUN = IDATA(LSTU3R+3)
191:      end if
192: C
193:      NTTHSC = 0
194:      NPTHE = 0
195:      LCTTHE = 0

```

```

196:      LTCTHE = 0
197:      NRETHE = 0
198:      NETHE = 0
199:      NBNTHE = 0
200:      NPPTHIN = 0
201:      NIGTH = 0
202:      NFGTH = 0
203:      MXNGTH = 0
204:      NATOMT = 0
205:      NPFCUT = 0
206:      EMAXTH = 0.0
207:      EHITHE = 0.0
208:      LOCTHE = 1
209:      ISWB6T = 0
210:      LTHR = 0
211:      NTHELS = 0
212:      NEBRG = 0
213:      NPDBY = 0
214:      SBMT2 = 0
215: C
216:      if ( LSTTHS.gt.0 ) then
217:      NTTHSC = IDATA(LSTTHS)
218:      NPTHE = IDATA(LSTTHS+1)
219:      LCTTHE = IDATA(LSTTHS+2)
220:      NRETHE = IDATA(LSTTHS+3)
221:      NETHE = IDATA(LSTTHS+4)
222:      NBNTHE = IDATA(LSTTHS+5)
223:      NPPTHIN = IDATA(LSTTHS+6)
224:      NIGTH = IDATA(LSTTHS+7)
225:      NFGTH = IDATA(LSTTHS+8)
226:      MXNGTH = IDATA(LSTTHS+9)
227:      NATOMT = IDATA(LSTTHS+10)
228:      NPFCUT = IDATA(LSTTHS+11)
229:      EMAXTH = ADATA(LSTTHS+12)
230:      EHITHE = ADATA(LSTTHS+13)
231: C
232:      if ( LCTTHE.lt.0 ) then
233:      ISWB6T = 1
234:      NTHELS = IDATA(LSTTHS+14)
235:      LTHR = IDATA(LSTTHS+15)
236:      NEBRG = IDATA(LSTTHS+16)
237:      NPDBY = IDATA(LSTTHS+17)
238:      SBMT2 = ADATA(LSTTHS+20)
239:      LCTTHE = -LCTTHE
240: C
241: C
242: C
243: C
244: C
245: C
246: C
247: C
248: C
249: C
250: C
251: C
252: C
253: C
254: C
255: C
256: C
257: C
258: C
259: C
260: C

```

src/artcore/dmain.f

```

261:         end if
262:     end if
263: end if
264: C
265: C ... print out library information
266: C
267: C the folloing part is always print out in MVPART.
268: C if ( IDBG.gt.0 ) then
269: C
270:     if ( IVERS.N.eq.2 ) then
271:         write(NSYSO,7000) MATT(1:8), HDATE, TITLE
272:     else
273:         write(NSYSO,7010) MATT(1:16), HDATE, TITLE(1:112)
274:     end if
275:     write(NSYSO,7020) NPTS, NTDATA, NUNR, NUNR2, NLEV,
276: &     NBINA, NBINE, NNU, NMT
277:     write(NSYSO,7040) NNK, IGFLAG, LFI, LNU, ITHRM, ISTU, IENDU,
278: &     LSUNR, LSNU, NBINA1, NBINE1, LF6, LSTF6, LSSF, LSTDOP,
279: &     LASYM, NOTDOP
280:     write(NSYSO,7060) NNUD, LNUD, NNF, LSNUD, NCAP, NCAPG
281:     write(NSYSO,7080) ATW, TLAB, ETHERM, ESAB, ELOW, PHI,
282: &     TSTAR, RTEMP
283: C
284:     if ( NLEV.gt.0 ) write(NSYSO,7100) (MTTHR(I),I=1,NLEV)
285:     if ( NCAP.gt.0 ) write(NSYSO,7120) (MTCAP(I),I=1,NCAP)
286:     if ( NCAPG.gt.0 ) write(NSYSO,7140) (MTCAPG(I),I=1,NCAPG)
287: C
288: 7000 format('1',20X,'*****'/'1X,
289: & 20X,'* mvp material library information *'/'1X,20X,
290: & ' * ( t-free original library ) *'/'1X,20X,
291: & '*****'/'1X,20X,
292: & ' * material id = ',A8,' *'/'1X,20X,
293: & '*****'/'1X,20X,
294: & ' * production date : ',A8,' *'/'1X,20X,
295: & '*****'/'1X,10X,
296: & ' comment : ',A60/1X,10X,' : ',A60//)
297: 7010 format('1',20X,'*****'/'1X,
298: & 20X,'* mvp material library information *'/'1X,20X,
299: & ' * ( t-free original library ) *'/'1X,20X,
300: & '*****'/'1X,20X,
301: & ' * material id = ',A16,' *'/'1X,20X,
302: & '*****'/'1X,20X,
303: & ' * production date : ',A8,' *'/'1X,20X,
304: & '*****'/'1X,10X,
305: & ' comment : ',A60/1X,10X,' : ',A52//)
306: 7020 format(
307: & 1X,15X,'npts : no of smooth data grid record.....',I10/
308: & 1X,15X,'ntdata: total record of #3 record (words)....',I10/
309: & 1X,15X,'nunr : no of unresolved prob. tables .....',I10/
310: & 1X,15X,'nunr2 : length of unresolved prob. table ...',I10/
311: & 1X,15X,'nlev : no of threshold reactions .....',I10/
312: & 1X,15X,'nbina : no of bins in angular dis. table....',I10/
313: & 1X,15X,'nbine : no of bins in energy dis. table....',I10/
314: & 1X,15X,'nnu : length of nu-data .....',I10/
315: & 1X,15X,'nmt : total no of reaction considered.....',I10)
316: 7040 format(
317: & 1X,15X,'nnk : max no of sub-sections in file-5....',I10/
318: & 1X,15X,'igflag: gamma production data flag .....',I10/
319: & 1X,15X,'lfi : flag of fission reaction .....',I10/
320: & 1X,15X,'lnu : nu-data flag (1/2=poly/tab).....',I10/
321: & 1X,15X,'itherm: flag of thermal scattering .....',I10/
322: & 1X,15X,'istu : upper energy mesh point of u.r. ....',I10/
323: & 1X,15X,'iendu : lower energy mesh point of u.r. ....',I10/
324: & 1X,15X,'lsunr : start position of u.r. table .....',I10/
325: & 1X,15X,'lsnu : start position of nu-data.....',I10/

```

```

326: & 1X,15X,'nbinal: nbina + 1 .....',I10/
327: & 1X,15X,'nbinel: nbine + 1 .....',I10/
328: & 1X,15X,'lf6 : flag of ddx data (file6) .....',I10/
329: & 1X,15X,'lstf6 : start posiotn for ddx control .....',I10/
330: & 1X,15X,'lssf : unresolved lssf flag .....',I10/
331: & 1X,15X,'lstdop: start posiotn for doppler data.....',I10/
332: & 1X,15X,'lasym : a flag of S(a,b) asymmetry .....',I10/
333: & 1X,15X,'notdop: a flag of doppler broadening .....',I10)
334: 7060 format(
335: & 1X,15X,'nnud : length of delayed nu-data.....',I10/
336: & 1X,15X,'lnud : delayed nu-data flag(1/2=poly/tab)....',I10/
337: & 1X,15X,'nnf : no of precursor families .....',I10/
338: & 1X,15X,'lsnud : start position of delayed nu .....',I10/
339: & 1X,15X,'ncap : no of capture reactions .....',I10/
340: & 1X,15X,'ncapg : no of gamma yield reac. by capture..',I10)
341: 7080 format(
342: & 1X,15X,'atw : atomic weight in n.m.u. ....',F10.3
343: & /1X,15X,'tlab : laboratory temperature in kelvin ...',
344: & F10.3/1X,15X,
345: & 'etherm: upper energy for thermal ela. mod...',F10.3/1X,
346: & 1X,15X,'esab : highest energy for thermal inela ...',F10.3/
347: & 1X,15X,'elow : lower energy defined (ev) .....',
348: & F10.5/1X,15X,
349: & 'ehi : upper energy defined (ev) .....',F10.1/1X,
350: & 15X,'tstar : effective temperature in kelvin ....',F10.3//
351: & 1X,20X,' << input requested temperaure list >>'/'1X,15X,
352: & 'rtemp : requested temperature in kelvin ....',F10.3/1X,
353: & 15X,'
354: & rtemp must be greater than equal tlab + 2.0'
355: & //)
356: 7100 format(1X,15X,'mtthr : threshold reaction mt id .....',I10I4)
357: 7120 format(1X,15X,'mtcap : reaction id for capture .....',I10I4)
358: 7140 format(1X,15X,'mtcapg : reaction id for (capture,gamma).....',I10I4)
359: C
360: write(NSYSO,7160)
361: write(NSYSO,7180)
362: write(NSYSO,7200)
363: do 140 MT = 1, NMT
364:     if ( MTINFO(MT).gt.0 ) then
365:         write(NSYSO,7220) MT, MTINFO(MT), ISTMT(MT), IENDMT(MT),
366: &         MTPAR(MT), NEUMT(MT), QVAL(MT), QVAL2(MT)
367:     end if
368: 140 continue
369: write(NSYSO,7200)
370: C
371: 7160 format('1',
372: & 20X,'*****'/'1X,
373: & 20X,'* #2 record infromation (part 1) *'/'1X,
374: & 20X,'*****'/'1X,
375: & 1X,10X,' mt mtinfo istmt iendmt mtpar neumt ',
376: & ' qval(ev) qval2(ev) ')
377: 7180 format(1X,9X,7('-----'))
378: 7200 format(1X,10X,I3,5I8,1P2E12.5)
379: C
380: write(NSYSO,7240)
381: write(NSYSO,7260)
382: write(NSYSO,7280)
383: C
384: do 150 MT = 1, NMT
385:     ISAVE = IABS(NKF5(MT))
386:     if ( MTINFO(MT).le.0.and.ISAVE.eq.0 ) go to 150
387:     write(NSYSO,7300) MT, LSTF3(MT), LCTMT(MT), NEANG(MT),
388: &     LSTF4(MT), NKF5(MT), NEF5(MT), LSTF5(MT)
389: 150 continue
390: write(NSYSO,7280)

```


src/artcore/dmain.f

```

391: 7240 format('1',
392: & 20X, '*****' //1X,
393: & 20X, ' #2 record infromation (part 2) *' //1X,
394: & 20X, '*****' //)
395: 7260 format(1X,10X,' mt lstf3 lctmt neang lstf4 nkf5 ',
396: & 'nef5 lstf5 ')
397: 7280 format(1X,9X,7('-----'))
398: 7300 format(1X,10X,I3,8I8)
399: 7320 format(1X,10X,' lff5s : ',10I8)
400: 7340 format(1X,10X,' nef5s : ',10I8)
401: 7360 format(1X,10X,' lstf5s: ',10I8)
402: C
403: if ( LF6.eq.1 ) then
404: write(NSYSO,7380)
405: write(NSYSO,7400)
406: do 160 MT = 1, NMT
407: if ( MTTOF6(MT).le.0 ) go to 160
408: write(NSYSO,7420) MT, MTTOF6(MT), EF61L(MT), EF62U(MT)
409: 160 continue
410: write(NSYSO,7400)
411: end if
412: C
413: 7380 format('1',
414: & 20X, '*****' //1X,
415: & 20X, ' #2 record infromation (ddx data) *' //1X,
416: & 20X, '*****' //)
417: & 10X,
418: & ' mt mttof6 ef61l(ev) ef62u(ev)')
419: 7400 format(1X,9X,4('-----'))
420: 7420 format(1X,10X,I3,I8,3X,1P2E12.5)
421: C
422: C ... print tfree control data
423: C
424: if ( ITFREE.gt.0 ) then
425: write(NSYSO,7440) LSTDOP, LSTTHS, LSTU3R, LENTHS, LENU3R
426: C
427: if ( LSTU3R.gt.0 ) then
428: write(NSYSO,7460) NTU3R, NURT, NBNU3R, IFISUN
429: LSTTU3 = LSTU3R + 3
430: ISW0 = LSTTU3 + NTU3R
431: if(NTU3R.eq.12) then
432: write(NSYSO,7481) (ADATA(LSTTU3+I),I=1,NTU3R)
433: write(NSYSO,7501) (ADATA(ISW0+I),I=1,NTU3R)
434: else
435: write(NSYSO,7480) (ADATA(LSTTU3+I),I=1,NTU3R)
436: write(NSYSO,7500) (ADATA(ISW0+I),I=1,NTU3R)
437: end if
438: end if
439: C
440: if ( LSTTHS.gt.0 ) then
441: write(NSYSO,7520) NTTHSC, NPTHE, LCTTHE, NRETHE, NETHE,
442: & NBNTHE, NPTHIN, NIGTH, NFGTH, MXNGTH, NATOMT,
443: & NPTCUT, EMAXTH, EHITHE, LOCTHE,
444: & ISWB6T, LASYM, NTHELS, LTHR, NEBRG, NPDBY, SBMT2
445: C
446: if( ISWB6T.eq.0 ) then
447: LSTTSC = LSTTHS + 13
448: ISW1 = LSTTSC + NTTHSC
449: ISW2 = LSTTSC + NTTHSC*2
450: write(NSYSO,7540) (ADATA(LSTTSC+I),I=1,NTTHSC)
451: write(NSYSO,7560) (ADATA(ISW1+I),I=1,NTTHSC)
452: if ( NPTHE.gt.0 )
453: & write(NSYSO,7580) (IDATA(ISW2+I),I=1,NTTHSC)
454: else
455: LSTTSC = LSTTHS + 20

```

```

456: ISW1 = LSTTSC + NTTHSC
457: ISW2 = LSTTSC + NTTHSC*2
458: C
459: if( IDBG.gt.2 ) then
460: write(NSYSO,*) ' NTTHSC LSTTHS LSTTSC ISW1 ISW2 '
461: write(NSYSO,*) NTTHSC,LSTTHS,LSTTSC,ISW1,ISW2
462: end if
463: C
464: write(NSYSO,7540) (ADATA(LSTTSC+I),I=1,NTTHSC)
465: write(NSYSO,7560) (ADATA(ISW1+I),I=1,NTTHSC)
466: write(NSYSO,7580) (IDATA(ISW2+I),I=1,NTTHSC)
467: end if
468: end if
469: end if
470: C
471: C
472: 7440 format('1',
473: & 20X, '*****' //1X,
474: & 20X, ' #2 record infromation (ddx data) *' //1X,
475: & 20X, '*****' //)
476: & 15X, 'lstdop: start address of temp. dep. data ...',I10/1X,
477: & 15X, 'lstths: start address of thermal scattering.',I10/1X,
478: & 15X, 'lstu3r: start address of u.r.p. table .....',I10/1X,
479: & 15X, 'lenths: record length of thermal scattering.',I10/1X,
480: & 15X, 'lenu3r: record length of u.r.p. table .....',I10/)
481: 7460 format('0',15X, '***** unresolved probability data control ****/'
482: & 1X,15X, 'ntu3r : no of temperature in u.r.p. data ...',I10/
483: & 1X,15X, 'nurt : no of energy in file-2 u.r. eval....',I10/
484: & 1X,15X, 'nbnu3r: no of brobaility bin .....',I10/
485: & 1X,15X, 'ifisun: fission reaction flag(0/1:no/yes)...',I10)
486: C7480 format(' temperatures .... ',11F10.3)
487: 7480 format(1X,79(' ')1X, ' temperatures (K) : node points ' /
488: & 1X,79(' ')1X,7F10.3/1X,4F10.3/1X,79(' '))
489: C7500 format(' log(temp.) .... ',11F10.6)
490: 7500 format(1X,79(' ')1X, ' log(temp.) : node points ' /
491: & 1X,79(' ')1X,7F10.6/1X,4F10.6/1X,79(' '))
492: 7481 format(1X,79(' ')1X, ' temperatures (K) : node points ' /
493: & 1X,79(' ')1X,7F10.3/1X,5F10.3/1X,79(' '))
494: 7501 format(1X,79(' ')1X, ' log(temp.) : node points ' /
495: & 1X,79(' ')1X,7F10.6/1X,5F10.6/1X,79(' '))
496: C
497: 7520 format('0',15X, '***** thermal scattering data control ****/'
498: & 1X,15X, 'ntthsc: no of temperature in th.sc. data ...',I10/
499: & 1X,15X, 'npthe : no of thermal elastic energy mesh...',I10/
500: & 1X,15X, 'lctthe: flame of elastic scattering angle...',I10/
501: & 1X,15X, 'nrethe: interpolation scheme for inci. eng...',I10/
502: & 1X,15X, 'nethe : no of incident energy in els. scat...',I10/
503: & 1X,15X, 'nbnthe: no of angle bins for elastic scat...',I10/
504: & 1X,15X, 'npthin: no of thermal inel. energy mesh 1...',I10/
505: & 1X,15X, 'nigth : no of thermal inel. energy mesh 2...',I10/
506: & 1X,15X, 'nfgth : no of thermal inel. energy mesh 3...',I10/
507: & 1X,15X, 'mxngth: =(nfg)*(nfg+1)/2 or nfg*nfg.....',I10/
508: & 1X,15X, 'natomt: no of atoms in molecule .....',I10/
509: & 1X,15X, 'nptcut: no of cut data in origianl mt=2 ....',I10/
510: & 1X,15X, 'emaxth: max. energy of thermal inelastic....',F10.5/
511: & 1X,15X, 'etherm: max. energy of thermal elastic ....',F10.5/
512: & 1X,15X, 'locthe: location of original thermal els. sc',I10/
513: & 1X,15X, 'iswb6t: thermal endf flag (0/1:b3/b6) .....',I10/
514: & 1X,15X, 'lasym : S(a,b) asymmetry (0/1:sym/asym)....',I10/
515: & 1X,15X, 'nthels: no of temperature in coh/incoh data.',I10/
516: & 1X,15X, 'lthr : elastic type (0/1/2:no/coh./incoh)...',I10/
517: & 1X,15X, 'negrg : no of Bragg edges in coh. scattering',I10/
518: & 1X,15X, 'npdby : no of debye-waller integrals data ..',I10/
519: & 1X,15X, 'sdbdy : bound cross section of incoh. scatt.',F10.5/)
520: C

```

src/artcore/dmain.f

```

521: 7540 format(' temperatures .... ',11F10.3)
522: 7560 format(' effective temp... ',11F10.3)
523: 7580 format(' interpol. method. ',11I10)
524: C
525:     else
526: C
527:         write(NSYSO,7590) TLAB, RTEMP
528: 7590 format( '1X,10X','tlab : laboratory temperature in kelvin ...',
529: &          F10.3/
530: &          1X,15X,' << input requested temperaure list >> '/1X,10X,
531: &          'rtemp : requested temperature in kelvin ....',F10.3/1X,
532: &          10X,'          rtemp must be greater than equal tlab + 2.0'
533: &          /)
534:         if ( LSTU3R.gt.0 ) then
535:             LSTTU3 = LSTU3R + 3
536:         end if
537:         if ( LSTTHS.gt.0 ) then
538:             LSTTSC = LSTTHS + 13
539:             if ( ISWB6T.eq.1 ) LSTTSC = LSTTHS + 20
540:         end if
541: C
542:     end if
543: C
544: C ... output process
545: C
546:     if ( NOTDOP.eq.1 ) then
547:         write(NSYSO,262) MATF(:ICLEN2(MATF))
548:         go to 261
549:     end if
550: C
551: 262 format(' This library('',A,'') has no dependency on temperature.',
552: &          '// So , doppler broadening process is skipped !!!')
553: C
554:     EUHIGH = 0.0
555:     EULOW = 0.0
556:     if ( NUNR.gt.0 ) then
557:         EUHIGH = ENERGY(ISTU-1)
558:         EULOW = ENERGY(IENDU)
559:     end if
560: C
561:     if ( (RTEMP-TLAB).lt.2.000 ) go to 230
562: C
563:     call FISTPR( RTEMP, IDBG )
564: C
565:     IST001 = ISTMT(1)
566:     IEND01 = IENDMT(1)
567:     MST001 = LSTF3(1) - 1
568: C
569:     IST002 = ISTMT(101)
570:     IEND02 = IENDMT(101)
571:     MST002 = LSTF3(101) - 1
572:     NP0101 = MTINFO(101)
573:     KDOP = 0
574: C
575:     ALPHA = ATW/(8.6164E-5*(RTEMP-TLAB))
576:     ELIM = 1.00E+6/ALPHA
577: C
578:     if ( NSYSO2.gt.0 ) then
579:         write(NSYSO2,*) ' ** euhigh eulow elim : ',
580: &          EUHIGH, EULOW, ELIM
581:     end if
582: C
583:     do 220 MT = 1, NMT
584:         if ( MTINFO(MT).le.0 ) go to 220
585:         if ( MT.eq.2 ) go to 170

```

```

586:         if ( MT.eq.18 ) go to 170
587:         if ( MT.eq.27 ) go to 170
588:         if ( MT.eq.100 ) go to 170
589: C
590:         if ( MTPAR(MT).lt.0 ) go to 220
591: C
592:         if ( MT.ge.102 ) go to 170
593: C
594:         NP = MTINFO(MT)
595:         IST = ISTMT(MT)
596:         IEND = IENDMT(MT)
597:         E1 = ENERGY(IEND)
598: C
599:         if ( NSYSO2.gt.0 ) then
600:             write(NSYSO2,*) ' ** doppler calculation(mt=', MT,
601: &          ' ) is skipped !! '
602:             write(NSYSO2,*) ' ** np ist iend e-first elim : ', NP, IST,
603: &          IEND, E1, ELIM
604:         end if
605:         go to 220
606: C
607: 170 continue
608: C
609:         MTH = MT
610:         QVALMT = QVAL(MT)
611:         NP = MTINFO(MT)
612:         call CLEA( XMESH, NPTS, 0.0 )
613:         call CLEA( YMESH, NPTS, 0.0 )
614:         IST = ISTMT(MT)
615:         IEND = IENDMT(MT)
616:         NP1 = NP + 1
617:         IS1 = IST - 1
618:         MST = LSTF3(MT) - 1
619: C
620:         do 180 I = 1, NP
621:             XX = ENERGY(IS1+I)
622:             YY = ADATA(MST+I)
623:             XMESH(NP1-I) = ENERGY(IS1+I)
624:             YMESH(NP1-I) = ADATA(MST+I)
625:             if ( NSYSO2.gt.0 ) then
626:                 if ( I.le.10 ) write(NSYSO2,*) ' mt i x y : ', MTH, NP1
627:                 &          - I, XX, YY
628:             if ( (NP
629:                 &          -I).le.10 ) write(NSYSO2,*) ' mt i x y : ', MTH, NP1
630:                 &          - I, XX, YY
631:             end if
632: 180 continue
633: C
634:         IMAXX = 1000
635:         IMAXD = IMAXX + 1
636:         NPMAX = 7*NP/4
637:         NPMAX = 3*NP
638:         IADDMX = 8002
639: C
640:         if ( NPMAX.lt.IADDMX ) NPMAX = IADDMX
641: C
642:         LOC01 = 1
643:         LOC02 = LOC01 + 2*NPMAX
644:         LOC03 = LOC02 + 2*NPMAX
645:         LOC04 = LOC03 + 2*NPMAX
646:         LOC05 = LOC04 + 2*IADDMX
647:         LOC06 = LOC05 + 2*14*IMAXD
648:         LOC07 = LOC06 + NPMAX
649:         LOC08 = LOC07 + NPMAX
650:         LOC09 = LOC08 + NPMAX

```

src/artcore/dmain.f

```

651:      LOC10 = LOC09 + IADDMX
652:      call DIMCHK( 'ARTPRC', LOC10, MEMORY )
653: C
654:      IDOP = 0
655: C
656:      call FILE3( ATW, TLAB, RTEMP, MTH, MATD, QVALMT, EUHIGH, EULOW,
657: &              NP, XMesh, YMesh, XSANS, IDOP, NPMAX, IMAXD, IMAXX,
658: &              IADDMX, A(LOC01), A(LOC02), A(LOC03), A(LOC04),
659: &              A(LOC05), A(LOC06), A(LOC07), A(LOC08), A(LOC09),
660: &              NPCTUT, ADATA(LOCTHE), ZA, IVPOP, IDBG )
661: C
662:      if ( IDOP.gt.0 ) then
663: C
664:          IMOD = 1
665:          if ( MT.eq.100 ) IMOD = 0
666:          if ( NP0101.gt.0.and.MT.eq.27 ) IMOD = 0
667:          if ( NP0101.gt.0.and.MT.gt.101 ) IMOD = 2
668: C ***** modify total x-section *****
669:          if ( IMOD.ge.1 ) then
670:              IDOP = 1
671:              IADD = IST - IST001
672:              do 190 I = 1, NP
673:                  ADATA(MST001+IADD+I) = ADATA(MST001+IADD+I)
674:                  & - ADATA(MST+I) + XSANS(NP1-I)
675: 190          continue
676:          end if
677: C ***** modify total capture x-section *****
678:          if ( IMOD.eq.2 ) then
679:              IADD = IST - IST002
680:              do 200 I = 1, NP
681:                  ADATA(MST002+IADD+I) = ADATA(MST002+IADD+I)
682:                  & - ADATA(MST+I) + XSANS(NP1-I)
683: 200          continue
684:          end if
685: C
686:          do 210 I = 1, NP
687:              XX = ENERGY(IS1+I)
688:              YY = ADATA(MST+I)
689:              YY2 = XSANS(NP1-I)
690:              ADATA(MST+I) = YY2
691:              if ( NSYSO2.gt.0 ) then
692:                  if ( I.le.10 ) write(NSYSO2,*) ' mt i x y y2 : ', MTH,
693: &              NP1 - I, XX, YY, YY2
694:                  if ( (NP-I).le.10 )
695:                      & write(NSYSO2,*) ' mt i x y y2 : ', MTH,
696:                      & NP1 - I, XX, YY, YY2
697:                  end if
698: 210          continue
699: C
700:          end if
701: C
702: 220 continue
703:      call LASTPR ( IDBG )
704: C
705: C ... print temperature dependent data of u.r.p.t & thermal data
706: C
707: 240 continue
708: C
709:      IPRTH = 0
710:      if ( IDBG.gt.2 ) IPRTH = 1
711: C
712: C
713:      if ( LSTU3R.gt.0 ) then
714: C
715: C ... print temp. dependent unresolved probability data

```

```

716: C
717: C      tmpu3r(ntu3r)
718:      LOC1 = LSTTU3 + 1
719: C      tmplog(ntu3r)
720:      LOC2 = LOC1 + NTU3R
721: C      eunrtb(nurt)
722:      LOC3 = LOC2 + NTU3R
723: C      sigeav(nurt)
724:      LOC4 = LOC3 + NURT
725: C      sigfav(nurt)
726:      LOC5 = LOC4 + NURT
727: C      sigtav(nurt)
728:      LOC6 = LOC5 + NURT
729: C      probun(nbnu3r,nurt)
730:      LOC7 = LOC6 + NURT
731: C      sigetb(ntu3r,nbnu3r,nurt)
732:      LOC8 = LOC7 + NURT*NBNU3R
733: C      sigftb(ntu3r,nbnu3r,nurt)
734:      LOC9 = LOC8 + NURT*NBNU3R*NTU3R
735: C      sigttb(ntu3r,nbnu3r,nurt)
736:      LOC10 = LOC9 + NURT*NBNU3R*NTU3R
737: C
738: C
739: C      anse (nbnu3r,nurt)
740:      MLOC01 = 1
741: C      ansf (nbnu3r,nurt)
742:      MLOC02 = MLOC01 + NBNU3R*NURT
743: C      anst (nbnu3r,nurt)
744:      MLOC03 = MLOC02 + NBNU3R*NURT
745: C      q (nbnu3r,nurt)
746:      MLOC04 = MLOC03 + NBNU3R*NURT
747: C      iupos (2,nbnu3r,nurt)
748:      MLOC05 = MLOC04 + NBNU3R*NURT
749: C      work (nbnu3r)
750:      MLOC06 = MLOC05 + NBNU3R*NURT*2
751:      if ( MOD(MLOC06,2).eq.0 ) MLOC06 = MLOC06 + 1
752: C      workq (nbnu3r)
753:      MLOC07 = MLOC06 + NBNU3R*2
754:      MLOC08 = MLOC07 + NBNU3R*2
755: C
756:      call DIMCHK( 'ARTPRC', MLOC08, MEMORY )
757:      LENU3R = NURT*(2+NBNU3R*6)
758:      if ( LSSF.eq.1 ) LENU3R = NURT*(5+NBNU3R*6)
759: C
760:      if ( NSYSO2.gt.0 ) then
761:          write(NSYSO2,*) ' ** loc1 to loc9 for subr.(lisu3r) ** '
762:          write(NSYSO2,'(9i10)') LOC1, LOC2, LOC3, LOC4, LOC5, LOC6,
763: &              LOC7, LOC8, LOC9
764:      end if
765: C
766: C      NPRT not used in unresr
767:      NPRT = NSYSO
768:      if ( IDBG.le.1 ) NPRT = NSYSO2
769: C
770:      call UNRESR( NPRT, NURT, NBNU3R, NTU3R, IFISUN, ADATA(LOC1),
771: &              ADATA(LOC2), ADATA(LOC3), ADATA(LOC4), ADATA(LOC5),
772: &              ADATA(LOC6), ADATA(LOC7), ADATA(LOC8), ADATA(LOC9),
773: &              ADATA(LOC10), A(MLOC01), A(MLOC02), A(MLOC03),
774: &              A(MLOC04), A(MLOC05), A(MLOC06), A(MLOC07),
775: &              ADATA(LSUNR), IDATA(LSUNR), LENU3R, RTEMP, LSSF, IDBG,
776: &              MATD )
777: C
778:      if ( IPRTH.gt.0 ) then
779:          call LISU3R( NSYSO, NURT, NBNU3R, NTU3R, IFISUN,
780: &              ADATA(LOC1), ADATA(LOC3), ADATA(LOC4), ADATA(LOC5),

```

src/artcore/dmain.f

```

781:      &          ADATA(LOC6), ADATA(LOC7), ADATA(LOC8), ADATA(LOC9),
782:      &          ADATA(LOC10) )
783:      end if
784: C
785:      end if
786:
787:      if ( LSTTHS.gt.0.and. ISWB6T.eq.0 ) then
788:          if(ITHERM.eq.1.or.ITHERM.eq.-1) then
789: C
790: C ... interpolates temp. dependent thermal scattering data
791: C tmpths(ntthsc)
792:      LOC01 = LSTTSC + 1
793: C tmpeff(ntthsc)
794:      LOC2 = LOC01 + NTTHSC
795: C intthe(ntthsc)
796:      LOC3 = LOC2 + NTTHSC
797: C emesh3(npthe)
798:      LOC02 = LOC01 + 3*NTTHSC
799: C crsth3(ntthsc,npthe)
800:      LOC03 = LOC02 + NPTHE
801: C nbttth4(nrethe)
802:      LOC04 = LOC03 + NPTHE*NTTHSC
803: C intth4(nrethe)
804:      LOC05 = LOC04 + NRETHE
805: C angth4(ntthsc,nbnthe,nethe)
806:      LOC06 = LOC05 + NRETHE
807: C emesh7(npthin)
808:      LOC07 = LOC06 + NTTHSC*NBNTHE*NETHE
809:      if ( NPTHE.le.0 ) LOC07 = LOC01 + 2*NTTHSC
810: C crsth7(ntthsc,npthin)
811:      LOC08 = LOC07 + NPTHIN
812: C emesh5(nfgth)
813:      LOC09 = LOC08 + NPTHIN*NTTHSC
814: C etran7(ntthsc,nfgth,nigth)
815:      LOC10 = LOC09 + NFGTH
816: C angth7(ntthsc,5,mxngth)
817:      LOC11 = LOC10 + NTTHSC*NFGTH*NIGTH
818: C crsmt2(nptcut)
819:      LOC12 = LOC11 + NTTHSC*5*MXNGTH
820: C
821: C ans1 (npthe)
822:      MLOC01 = 1
823: C ans2 (nbnthe,nethe)
824:      MLOC02 = MLOC01 + NPTHE
825: C ans3 (npthin)
826:      MLOC03 = MLOC02 + NBNTHE*NETHE
827: C ans4 (nfgth,nfgth)
828:      MLOC04 = MLOC03 + NPTHIN
829: C ans5 (5,mxngth)
830:      MLOC05 = MLOC04 + NFGTH*NFGTH
831: C ians (nfgth*nfgth*2)
832:      MLOC06 = MLOC05 + 5*MXNGTH
833: C work1 (nfgth)
834:      MLOC07 = MLOC06 + NFGTH*NFGTH*2
835: C
836:      LENWRK = 2*NTTHSC
837: C
838:      if ( 2*NPTHE .gt.LENWRK ) LENWRK = 2*NPTHE
839:      if ( 2*NPTHIN.gt.LENWRK ) LENWRK = 2*NPTHIN
840: C
841:      LENWRK = 2*LENWRK
842: C
843:      if ( NFGTH*NFGTH.gt.LENWRK ) LENWRK = NFGTH*NFGTH
844: C work2 (lenwrk)
845:      MLOC08 = MLOC07 + NFGTH

```

```

846: C      workp (nfgth)
847:      MLOC09 = MLOC08 + LENWRK
848:      if ( MOD(MLOC09,2).eq.0 ) MLOC09 = MLOC09 + 1
849: C      workq(nfgth)
850:      MLOC10 = MLOC09 + 2*NFGTH
851:      MLOC11 = MLOC10 + 2*NFGTH
852: C
853:      call DIMCHK( 'ARTPRC', MLOC11, MEMORY )
854: C
855:      if ( NSYSO2.gt.0 ) then
856:          write(NSYSO2,*) ' ** loc01 to loc12 for subr.(lisths) ** '
857:          write(NSYSO2,'(9i10)') LOC01, LOC02, LOC03, LOC04, LOC05,
858:          &          LOC06
859:          write(NSYSO2,'(9i10)') LOC07, LOC08, LOC09, LOC10, LOC11,
860:          &          LOC12
861:      end if
862: C
863: C NPRT is not used in thermj
864:      NPRT = NSYSO
865:      if ( IDBG.le.1 ) NPRT = NSYSO2
866: C
867:      call THERMJ( NPRT, NTTHSC, NPTHE, NRETHE, NETHE,
868:          &          NBNTHE, NPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT, ADATA,
869:          &          IDATA, ENERGY, XMESH, ADATA(LOC01), ADATA(LOC02),
870:          &          ADATA(LOC03), ADATA(LOC04), ADATA(LOC05), ADATA(LOC06),
871:          &          ADATA(LOC07), ADATA(LOC08), ADATA(LOC09), ADATA(LOC10),
872:          &          ADATA(LOC11), ADATA(LOC12), ADATA(LOC2), IDATA(LOC3),
873:          &          A(MLOC01), A(MLOC02), A(MLOC03), A(MLOC04), A(MLOC05),
874:          &          A(MLOC06), A(MLOC07), A(MLOC08), A(MLOC09), A(MLOC10),
875:          &          LENWRK, RTEMP, NATOMT )
876: C
877: C ... print temp. dependent thermal scattering data
878: C
879:      if ( IPRTH.gt.0 ) then
880:          call LISTHS( NSYSO, NTTHSC, NPTHE, NRETHE, NETHE, NBNTHE,
881:          &          NPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT, ENERGY, NPTS,
882:          &          ADATA(LOC01), ADATA(LOC02), ADATA(LOC03),
883:          &          ADATA(LOC04), ADATA(LOC05), ADATA(LOC06),
884:          &          ADATA(LOC07), ADATA(LOC08), ADATA(LOC09),
885:          &          ADATA(LOC10), ADATA(LOC11), ADATA(LOC12) )
886:      end if
887:      end if
888: C
889:      end if
890:
891:      if ( LSTTHS.gt.0 .and. ISWB6T.eq.1 ) then
892:
893:          if ( IITHERM.eq.1 .or. IITHERM.eq.-1 ) then
894: C
895: C *** interpolates temp. dependent thermal scattering data
896: C tmpths(ntthsc)
897:      LOC01 = LSTTSC + 1
898: C tmpeff(ntthsc)
899:      LOC2 = LOC01 + NTTHSC
900: C inttmp(ntthsc)
901:      LOC03 = LOC2 + NTTHSC
902: C
903: C tmpels(nthels)
904:      LOC04 = LOC03 + NTTHSC
905: C emesh3(npthe)
906:      LOC05 = LOC04 + NTHELS
907: C intels(nthels)
908:      LOC06 = LOC05 + NPTHE
909: C engbrg(nebgr)
910:      LOC07 = LOC06 + NTHELS

```

src/artcore/dmain.f

```

911: C      sbg(nthels,negbrg)
912:         LOC08 = LOC07 + NEBRG
913: C      dby(npdby)
914:         LOC09 = LOC08 + NEBRG*NTHELS
915: C      emesh7(npthin)
916:         LOC10 = LOC09 + NPDBY
917:         if ( NEBRG.le.0.and.NPDBY.eq.0 ) LOC07 = LOC04
918: C      crsth7(ntthsc,npthin)
919:         LOC11 = LOC10 + NPTHIN
920: C      emesh5(nfgth)
921:         LOC12 = LOC11 + NPTHIN*NTTHSC
922: C      etran7(ntthsc,nfgth,nigth)
923:         LOC13 = LOC12 + NFGTH
924: C      angth7(ntthsc,4,mxngth)
925:         LOC14 = LOC13 + NTHSC*NFGTH*NIGTH
926: C      sin(mxngth)
927:         LOC15 = LOC14 + NTHSC*4*MXNGTH
928: C      crsmc2(nptcut)
929:         LOC16 = LOC15 + MXNGTH
930: C
931: C      ans1 (npthe)
932:         MLOC01 = 1
933: C      ans2 (nbinal,nethe)
934:         MLOC02 = MLOC01 + NPTHE
935: C      ans3 (npthin)
936:         MLOC03 = MLOC02 + NBNTHE*NBINAL
937: C      ans4 (nfgth,nfgth)
938:         MLOC04 = MLOC03 + NPTHIN
939: C      ans5 (5,mxngth)
940:         MLOC05 = MLOC04 + NFGTH*NFGTH
941: C      nbinbg(nebrg)
942:         MLOC06 = MLOC05 + 5*MXNGTH
943: C      anssb(nebrg)
944:         MLOC07 = MLOC06 + NEBRG
945: C      pije(nfgth,nfgth)
946:         MLOC08 = MLOC07 + NEBRG
947:         if( NEBRG.eq.0) MLOC08 = MLOC06
948: C      angnew(4,mxngth)
949:         MLOC09 = MLOC08 + NFGTH*NFGTH
950: C
951:         IBCMAX = NFGTH
952:         if( IBCMAX .lt. NEBRG ) IBCMAX = NEBRG
953: C      ians (nfgth*nfgth*2)
954:         MLOC10 = MLOC09 + 4*MXNGTH
955: C      work1 (nfgth)
956:         MLOC11 = MLOC10 + IBCMAX*IBCMAX*2
957: C
958:         LENWRK = 2*NTTHSC
959:         if ( 2*NPTHE .gt.LENWRK ) LENWRK = 2*NPTHE
960:         if ( 2*NPTHIN.gt.LENWRK ) LENWRK = 2*NPTHIN
961:         if ( 2*NEBRG .gt.LENWRK ) LENWRK = 2*NEBRG
962: C
963:         LENWRK = 2*LENWRK
964:         if ( NFGTH*NFGTH.gt.LENWRK ) LENWRK = NFGTH*NFGTH
965:         if ( NEBRG*NEBRG.gt.LENWRK ) LENWRK = NEBRG*NEBRG
966: C      work2 (lenwrk)
967:         MLOC12 = MLOC11 + NFGTH
968: C      workp (ibcmx)
969:         MLOC13 = MLOC12 + LENWRK
970:         if ( MOD(MLOC13,2).eq.0 ) MLOC13 = MLOC13 + 1
971: C      workq (ibcmx)
972:         MLOC14 = MLOC13 + 2*IBCMAX
973:         MLOC15 = MLOC14 + 2*IBCMAX
974: C
975:         call DIMCHK( 'ARTPRC', MLOC15, MEMORY )

```

```

976: C
977:         if ( NSYSO2.gt.0 ) then
978:           write(NSYSO2,*) ' ** loc01 to loc16 for subr.(lisths) ** '
979:           write(NSYSO2,'(9i10)') LOC01, LOC02, LOC03, LOC04, LOC05,
980:             & LOC06
981:           write(NSYSO2,'(9i10)') LOC07, LOC08, LOC09, LOC10, LOC11,
982:             & LOC12
983:           write(NSYSO2,'(9i10)') LOC13, LOC14, LOC15, LOC16
984:         end if
985: C
986: C      NPRT is not used in thermj
987:         NPRT = NSYSO
988:         if ( IDBG.le.1 ) NPRT = NSYSO2
989: C
990:         call THERM6( NPRT, NTTHSC, NTHELS,IBCMAX,
991:           & NPTHE, NPDBY , NEBRG ,SBMT2 ,
992:           & NPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT, ADATA,
993:           & IDATA, ENERGY, XMesh,
994:           & ADATA(LOC01), ADATA(LOC02),IDATA(LOC03),
995:           & ADATA(LOC04), IDATA(LOC06),
996:           & ADATA(LOC05), ADATA(LOC09), ADATA(LOC07), ADATA(LOC08),
997:           & ADATA(LOC10), ADATA(LOC11), ADATA(LOC12), ADATA(LOC13),
998:           & ADATA(LOC14), ADATA(LOC15), ADATA(LOC16),
999:           & A(MLOC01), A(MLOC02), A(MLOC03), A(MLOC04), A(MLOC05),
1000:          & IA(MLOC06), A(MLOC07), A(MLOC08), A(MLOC09),IA(MLOC10),
1001:          & A(MLOC11), A(MLOC12), A(MLOC13), A(MLOC14),
1002:          & LENWRK, RTEMP, NATOMT )
1003: C
1004: C      ... print temp. dependent thermal scattering data
1005: C
1006:         if ( IDBG.gt.2 ) then
1007:           call LISTH6( NSYSO, NTTHSC, NPTHE, NTHELS, NEBRG, NPDBY ,
1008:             & NPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT,
1009:             & ENERGY, NPTS, IDBG, LASYM,
1010:             & ADATA(LOC01), ADATA(LOC04), ADATA(LOC05),
1011:             & IDATA(LOC06), ADATA(LOC07), ADATA(LOC08),
1012:             & ADATA(LOC10), ADATA(LOC11), ADATA(LOC12),
1013:             & ADATA(LOC13), ADATA(LOC14), ADATA(LOC15),
1014:             & ADATA(LOC16), ADATA(LOC09), SBMT2 )
1015:         end if
1016:       end if
1017: C
1018: C      ... interpolates effective temperature for S.C.T approximation
1019: C      of free atom
1020: C
1021:         if(ITHERM.eq.-2) then
1022: C      tmpths(ntthsc)
1023:           LOC01 = LSTTSC + 1
1024: C      tmpeff(ntthsc)
1025:           LOC02 = LOC01 + NTTHSC
1026: C      inttmp(ntthsc)
1027:           LOC03 = LOC02 + NTTHSC
1028: C
1029:           TSTAR = 0.0
1030:           call INTEFF( RTEMP , TSTAR , NTTHSC ,NSYSO , IDBG ,
1031:             & ADATA(LOC01),ADATA(LOC02),IDATA(LOC03) )
1032:         end if
1033: C
1034:       end if
1035: C
1036: C      ... modify mt=3 x-section if exists
1037: C
1038:         if ( MTINFO(3).gt.0.and.KDOP.eq.1 ) then
1039:           IST = ISTMT(3)
1040:           IEND = IENDMT(3)

```

src/artcore/dmain.f

```
1041:      IST001 = ISTMT(1)
1042:      IEND01 = IENDMT(1)
1043:      IST002 = ISTMT(2)
1044:      IEND02 = IENDMT(2)
1045:      NP      = MTINFO(3)
1046:      NP0002  = MTINFO(2)
1047:      IADD01  = IST - IST001
1048:      IADD02  = IST - IST002
1049:      MST1    = LSTF3(3) - 1
1050:      MST001  = LSTF3(1) - 1
1051:      MST002  = LSTF3(2) - 1
1052: C
1053:      do 250 I = 1, NP
1054:      &      ADATA(MST1+I) = ADATA(MST001+IADD01+I)
1055:      250 continue
1056: C
1057:      if ( NP0002.gt.0 ) then
1058:      do 260 I = 1, NP
1059:      &      SAVEE = 0.0
1060:      &      IPOS  = IADD02 + I
1061:      &      if ( IPOS.ge.1.and.IPOS.le.NP0002 ) SAVEE =
1062:      &      &      ADATA(MST002+IPOS)
1063:      &      ADATA(MST1+I) = ADATA(MST1+I) - SAVEE
1064:      260 continue
1065:      end if
1066:      end if
1067:
1068:      261 continue
1069: C
1070: C ... modify TLAB
1071: C
1072:      if( NOTDOP.eq.0 ) then
1073:      &      TLAB = RTEMP
1074:      end if
1075:
1076:      230 continue
1077: C
1078: C ... modify NTDATA ! dopp-scat
1079: C
1080:      NTDATA0 = NTDATA
1081:      if(LSTDOP.gt.0) NTDATA = LSTDOP - 1
1082:      LSTDOP = 0
1083:
1084:      if(TBASE.eq.0.) then
1085: C      .... save elastic scattering cross section at original T
1086:      do I = 1, NPTS
1087:      &      ADATA(NTDATA+I) = ADATA(NTDATA0+I)
1088:      end do
1089:
1090:      MT      = 120
1091:      MTINFO(MT) = NPTS
1092:      MTPAR(MT) = MT
1093:      ISTMT(MT) = 1
1094:      IENDMT(MT) = NPTS
1095:      NEUMT(MT) = 1
1096:      LCTMT(MT) = 2
1097:      NEANG(MT) = 0
1098:      NKF5(MT) = 0
1099:      NEF5(MT) = 0
1100:      LSTF3(MT) = NTDATA + 1
1101:      LSTF4(MT) = 0
1102:      LSTF5(MT) = 0
1103:      QVAL(MT) = 0.0
1104:      QVAL2(MT) = 0.0
1105:      LSTGAM(MT) = 0
1106:
1107:      NTDATA = NTDATA + NPTS
1108:      end if
1109: C ... end of process
1110: C
1111:      call CPUTM( TCP1 )
1112:      call TOKEI( T01, 0 )
1113: C
1114:      write(NSYSO,7600) TCP1 - TCP0, T01 - T00
1115:      7600 format(1X,' *** CPU TIME USED FOR THIS NUCLIDE : ',E12.5/1X,
1116:      &      ' *** ELAPSED TIME FOR THIS NUCLIDE : ',E12.5/)
1117: C
1118:      return
1119:      end
```

src/artcore/errokc.f

```
1:      subroutine ERROKC( E )
2:      c
3:      c      define allowable error for file 3 reconstructed cross sections.
4:      c      the error law can be energy independent (constant) or energy
5:      c      dependent (given by a linearly interpolable table in energy
6:      c      vs. error).
7:      c
8:      c***** double *****
9:      real*8 E
10:     c***** double *****
11:     common /OKERRC/ ERXC3C, ERXC30, KERR3C, MAXERC, ENER3C(21),
12:     & ER3C(21)
13:     c-----initialize index to interpolation table.
14:     data MINERD /2/
15:     c-----energy dependent. within energy range of error law use linear
16:     c-----interpolation. outside range extend error as constant from
17:     c-----closest end of table.
18:     if ( E.le.ENER3C(1) ) then
19:     c-----extend to lower energies as constant.
20:     MINERD = 2
21:     ERXC3C = ER3C(1)
22:     ERXC30 = 0.1*ERXC3C
23:     return
24:     else
25:     do 100 NOWERD = MINERD, MAXERC
26:     if ( E.lt.ENER3C(NOWERD) ) go to 110
27:     if ( E.eq.ENER3C(NOWERD) ) go to 150
28: 100 continue
29:     c-----extend to higher energies as constant.
30:     go to 160
31:     end if
32: 110 NM1 = NOWERD - 1
33:     if ( E.lt.ENER3C(NM1) ) then
34:     do 120 NOWERD = 2, MAXERC
35:     if ( E.lt.ENER3C(NOWERD) ) go to 130
36:     if ( E.eq.ENER3C(NOWERD) ) go to 150
37: 120 continue
38:     go to 160
39:     else if ( E.eq.ENER3C(NM1) ) then
40:     c-----exact energy match.
41:     MINERD = NM1
42:     if ( MINERD.le.1 ) MINERD = 2
43:     ERXC3C = ER3C(NM1)
44:     ERXC30 = 0.1*ERXC3C
45:     return
46:     else
47:     go to 140
48:     end if
49:     c-----interpolate between energies.
50: 130 NM1 = NOWERD - 1
51: 140 MINERD = NOWERD
52:     ERXC3C =
53:     & ((ENER3C(NOWERD)-E)*ER3C(NM1)+(E-ENER3C(NM1))*ER3C(NOWERD)
54:     & )/(ENER3C(NOWERD)-ENER3C(NM1))
55:     ERXC30 = 0.1*ERXC3C
56:     return
57:     c-----exact energy match.
58: 150 MINERD = NOWERD
59:     ERXC3C = ER3C(NOWERD)
60:     ERXC30 = 0.1*ERXC3C
61:     return
62:     c-----extend to higher energies as constant.
63: 160 MINERD = MAXERC
64:     ERXC3C = ER3C(MAXERC)
65:     ERXC30 = 0.1*ERXC3C
```

```
66:      return
67:      end
```

src/artcore/errokt.f

```
1:      subroutine ERROKT( E )
2:      c
3:      c      define allowable error for file 3 linearized cross sections.
4:      c      the error law can be energy independent (constant) or energy
5:      c      dependent (given by a linearly interpolable table in energy
6:      c      vs. error).
7:      c
8:      c***** double *****
9:      real*8 E
10:     c***** double *****
11:     common /OKERT/  ERXC3T, KERR3T, MAXERT, ENER3T(21),      ER3T(21)
12:     c-----initialize index to interpolation table.
13:     data MINER3 /2/
14:     c-----energy dependent. within energy range of error law use linear
15:     c-----interpolation. outside range extend error as constant from
16:     c-----closest end of table.
17:     if ( E.le.ENER3T(1) ) then
18:     c-----extend as constant to lower energies.
19:         MINER3 = 2
20:         ERXC3T = ER3T(1)
21:         return
22:     else
23:         do 100 NOWER3 = MINER3, MAXERT
24:             if ( E.lt.ENER3T(NOWER3) ) go to 110
25:             if ( E.eq.ENER3T(NOWER3) ) go to 150
26:         100 continue
27:     c-----extend as constant to higher energies.
28:         go to 160
29:     end if
30:     110 NM1      = NOWER3 - 1
31:     if ( E.lt.ENER3T(NM1) ) then
32:         do 120 NOWER3 = 2, MAXERT
33:             if ( E.lt.ENER3T(NOWER3) ) go to 130
34:             if ( E.eq.ENER3T(NOWER3) ) go to 150
35:         120 continue
36:         go to 160
37:     else if ( E.eq.ENER3T(NM1) ) then
38:     c-----exact energy match.
39:         MINER3 = NM1
40:         if ( MINER3.le.1 ) MINER3 = 2
41:         ERXC3T = ER3T(NM1)
42:         return
43:     else
44:         go to 140
45:     end if
46:     c-----interpolate between energies.
47:     130 NM1      = NOWER3 - 1
48:     140 MINER3 = NOWER3
49:     ERXC3T =
50:     &      ((ENER3T(NOWER3)-E)*ER3T(NM1)+(E-ENER3T(NM1))*ER3T(NOWER3)
51:     &      )/(ENER3T(NOWER3)-ENER3T(NM1))
52:     return
53:     c-----exact energy match.
54:     150 MINER3 = NOWER3
55:     ERXC3T = ER3T(NOWER3)
56:     return
57:     c-----extend as constant to higher energies.
58:     160 MINER3 = MAXERT
59:     ERXC3T = ER3T(MAXERT)
60:     return
61:     end
```


src/artcore/fend.f

```
1:      subroutine FEND
2: c
3:      integer OTAPE, OUTP, SCR
4: c
5:      common /HEADR/  C1H,    C2H,    L1H,    L2H,    N1H,    N2H,
6: &      MATH,    MFH,    MTH,    NOSEQ
7:      common /UNITS/  INP,    OUTP,    ITAPE,    OTAPE,    SCR
8: c
9:      MTH      = 0
10:     MFH      = 0
11:     write(OTAPE,7000) MATH, MFH, MTH, NOSEQ
12:     NOSEQ    = NXTSEQ(NOSEQ)
13: c
14: 7000 format(66X,I4,I2,I3,I5)
15: c
16:     return
17:     end
```

src/artcore/fil5te.f

```

1:      subroutine FIL5TE( EMESH ,SB      ,Q      ,IEPOS, NBIN ,NEBRG ,ADATA,
2:      &                  IDATA,NSYSO ,LENGTH,SBDATA,WORKP, WORKQ,IDBG )
3:      c
4:      c      this subroutine was added for endf/B6 thermal coherent angular &
5:      c      energy distribution data interpolation by k.kaneko ITIRO 26/June/2001
6:      c
7:      real      Q(NEBRG,NEBRG), SB(NEBRG), EMESH(NEBRG), ADATA(LENGTH)
8:      real      SBDATA(NEBRG)
9:      integer    NBIN(NEBRG),IEPOS(2,NEBRG,NEBRG), IDATA(LENGTH)
10:     c
11:     real*8 WORKP(NEBRG), WORKQ(NEBRG), SUMP, DSAVE
12:     c
13:     c-----initial set
14:     c
15:     NEBRG2 = NEBRG*NEBRG
16:     NEBRG4 = NEBRG*NEBRG*2
17:     c
18:     call DCLEA( Q      , NEBRG2 , 0.0 )
19:     call DCLEA( IEPOS, NEBRG4 , 0 )
20:     call DCLEA( NBIN , NEBRG , 0 )
21:     c
22:     NBIN(1) = 2
23:     Q(1,1) = 0.500
24:     Q(2,1) = 0.500
25:     IEPOS(1,1,1) = 1
26:     IEPOS(2,1,1) = 1
27:     IEPOS(1,2,1) = 1
28:     IEPOS(2,2,1) = 1
29:     c
30:     LIM1 = 5
31:     LIM2 = NEBRG -4
32:     c
33:     c-----define SB data
34:     c
35:     do 10 I = NEBRG , 2 , -1
36:       DSAVE = DBLE( SB(I) - SB(I-1) )
37:       if( DSAVE.lt.0.0) DSAVE = 0.0
38:       SB(I) = DSAVE
39:     10 continue
40:     c
41:     c-----loop of incident energy
42:     c
43:     do 150 LOP = 2, NEBRG
44:       call DCLEA( WORKP, NEBRG, 0.0d0 )
45:       call DCLEA( WORKQ, NEBRG, 0.0d0 )
46:     c
47:       NBIN(LOP) = LOP
48:       SUMP = 0.0
49:     c
50:       do 100 J = 1, LOP
51:         WORKP(J) = DBLE( SB(J) )
52:         SUMP = SUMP + WORKP(J)
53:       100 continue
54:     c
55:       if ( SUMP.gt.0.0 ) then
56:         do 110 J = 1, LOP
57:           WORKP(J) = WORKP(J) /SUMP
58:         110 continue
59:       end if
60:     c
61:       call BMCMD( WORKP, WORKQ, IEPOS(1,1,LOP), LOP )
62:     c
63:       do 120 J = 1, LOP
64:         Q(J,LOP) = WORKQ(J)
65:       120 continue

```

```

66:     c
67:     if ( LOP.le.LIM1 .or. LOP.ge.LIM2 ) then
68:     c
69:       if ( IDBG.gt.1 ) then
70:         write(NSYSO,7000) LOP, EMESH(LOP)
71:         write(NSYSO,7020)
72:       c
73:         do 130 J = 1 , LOP
74:           WORKP(J) = SB(J)
75:         130 continue
76:       c
77:         do 140 I = 1, NBIN(LOP)
78:           write(NSYSO,7040) I, EMESH(I), WORKP(I), Q(I,LOP),
79:             & (IEPOS(J,I,LOP),J=1,2)
80:         140 continue
81:         write(NSYSO,7020)
82:       end if
83:     end if
84:   150 continue
85:   c
86:   7000 format(/
87:     & /1X,10X,'*****'/'
88:     & 1X,10X,'* thermal coherent elastic eng. distri. * '/'
89:     & 1X,10X,'* ( bmc sampling ) * '/'
90:     & 1X,10X,'*****'/'
91:     & /1X,15X,' lop = ',I3,10X,'incident energy= ',1PE12.5,' ev '/'
92:     & /1X,5X,'no energy(ev) struc. func. q-answer ians1 ians2')
93:   7020 format(1X,4X,5('-----'),'--')
94:   7040 format(1X,4X,I3,1P,3E12.5,2I6)
95:   7050 format(' NO ENERGY OLD-SB NEW-SB : ',I3,1P3E12.5)
96:   c
97:   c-----output data to adata or idata array
98:   c
99:   c + ( nbin(i), (q(j,i),j=1,nbin(i)) ,
100:  c + (iepos(1,j,i),iepos(2,j,i),j=1,nbin(i)) , i=1,nebrg )
101:   c
102:   ISW = 0
103:   do 180 I = 1, NEBRG
104:     if ( IDBG.gt.1 ) then
105:       write(NSYSO,7050) I,EMESH(I),SBDATA(I),SB(I)
106:     endif
107:   c
108:     SBDATA(I) = SB(I)
109:     ISW = ISW + 1
110:     IDATA(ISW) = NBIN(I)
111:     do 160 J = 1, NBIN(I)
112:       ISW = ISW + 1
113:       ADATA(ISW) = Q(J,I)
114:     160 continue
115:     do 170 J = 1, NBIN(I)
116:       ISW = ISW + 1
117:       IDATA(ISW) = IEPOS(1,J,I)
118:       ISW = ISW + 1
119:       IDATA(ISW) = IEPOS(2,J,I)
120:     170 continue
121:   180 continue
122:   c
123:   return
124:   end

```

src/artcore/file3.f

```

1:      subroutine FILE3( ATW,  TLAB,  RTEMP, MT,    MATNO, QVALMT,UHIGH,
2:      &                ULOW, NP,    XMESH, YMESH, XSANS, IDOP,  NPMAX,
3:      &                IMAXD, IMAXX, IADDMX,ECOLD, YCOLD, EHOT,  ESAVE,
4:      &                WBROAD,XCCOLD,DCOLD, XCHOT, XCSAVE,NPTCUT,
5:      &                CRSTHE,ZAPASS,IVPOP,IDBG )
6: C=<ARTCORE>=====
7: C Purpose :
8: C   This routine is designed to doppler broaden one endf/b section
9: C   of data (i.e., one reaction). Data is read, doppler broadened and
10: C   output in the endf/b format. If the section contains 6012 or fewer
11: C   points the entire operation is performed in core. If the section
12: C   contains more than 6012 points the data will be broadened a page
13: C   (1 page = 2004 points) at a time, written to scratch and after the
14: C   entire section has been broadened it will be read back from
15: C   scratch and output in the endf/b format.
16: C Called in : ARTPRC
17: C Calls :
18: C=====
19: c
20: c***** double *****
21:      real*8 ELAST
22: c***** double *****
23:      integer OUTP, SCR, TOOHI, COLD1, COLD2, COLD1P, COLD2P, HOT1,
24:      &          HOT2, HOT3, HOT3M1, UNRES1, UNRES2, UREVAL, UREACT
25: c***** character *****
26:      character*4 FMTHOL, KSIGN
27: c
28:      include 'INC/_MAINIO'
29: c
30:      common /INDEX/  COLD1,  COLD2,  COLD1P, COLD2P, HOT1,  HOT2,
31:      &              HOT3,  HOT3M1, N2IN,  N2TOT,  N2SCR
32:      common /PAGER/  NPAGE,  NPT2,  NPT3,  NP1P1,  NP2P1
33:      common /HOTS/   ALPHA,  HOTSY,  TEMPK,  TEMPEF, N2TA1, N2TAPO
34:      common /HEADR/  ZA,  AWR,  L1H,  L2H,  N1H,  N2H,
35:      &              MATH,  MFH,  MTH,  NOSEQ
36:      common /LEADER/  C1,  Q,  L1,  L2,  N1,  N2
37:      common /UNITS/  INP,  OUTP,  ITAPE,  OTAPE,  SCR
38:      common /FILLER/ N2LEFT, TOOHI, ITHRES, LOAD1,  LOAD2
39:      common /EXTEND/ MESS,  DTMAX
40:      common /RESOLV/ UREVAL, UREACT, UNRES1, UNRES2, EULOW,  EUHIGH,
41:      &              IL,  IH
42:      common /LASTE/  ELAST
43:      common /TEMPO/  TEMP3,  IVERSE
44:      common /SLIM/   ISTART, NOTHIN, ITHIN1, ITHIN2, ITHIN3, MTEND
45:      common /NORMF/  XNORM(6),  KEXP(6)
46:      common /NORMC/  KSIGN(6)
47:      common /HOLFM/  FMTHOL
48:      common /MATTOT/ MATIN,  MATOUT
49: c
50:      real XMESH(NP), YMESH(NP), XSANS(NP), CRSTHE(NPTCUT)
51: c
52:      real XCCOLD(NPMAX), DCOLD(NPMAX), XCHOT(NPMAX), XCSAVE(IADDMX)
53:      real*8 ECOLD(NPMAX), YCOLD(NPMAX), EHOT(NPMAX), ESAVE(IADDMX)
54:      real*8 WBROAD(IMAXD,14)
55: c
56:      character*4 MESSAG(3,2)
57: c
58: c-----define cross section extension message.
59:      data MESSAG /' ', ' ', ' ', 'exte', 'nsio', 'n'/
60: c
61: c
62: c-----define boltzmann constant in ev/degree kelvin
63: c----- (assuming energy will be expressed in ev and the atomic
64: c----- weight ratio will be expressed in neutron mass units,
65: c----- as opposed to atomic mass units (amu)).

```

```

66:      data BOLTZM /8.6164E-05/
67: c-----initialize temperature read from total cross section (mt=1).
68:      data TEMP1 /0.0/
69: c
70: c   read tab1 leader card and interpolation law. check interpolation
71: c   law and if data is not linearly interpolable terminate execution.
72: c
73: c
74: c   initialize all counts and flags for section.
75: c
76: c-----initialize flag not to print extension message with section.
77:      MESS = 1
78: c-----initialize flag to doppler broaden section.
79:      TOOHI = -1
80: c-----turn off doppler broadening flag if section is mu-bar, xi or
81: c-----gamma.
82:      if ( MTH.ge.251.and.MTH.le.253 ) TOOHI = 1
83: c
84:      TEMPK = RTEMP
85:      TEMP1 = TLAB
86:      TEMP3 = TLAB
87:      TEMPEF = TEMPK - TEMP3
88:      MTH = MT
89:      MFH = 3
90:      MATH = MATNO
91:      AWR = ATW
92:      ZA = ZAPASS
93:      IZA = ZAPASS
94:      EULOW = ULOW
95:      EUHIGH = UHIGH
96:      UREVAL = 0
97:      if ( EULOW.gt.0.0 ) UREVAL = 1
98: c
99:      C1 = RTEMP
100:      Q = QVALMT
101: c
102:      if ( OTAPE.gt.0 ) call CARDO( ZA, AWR, 0, 99, 0, 0 )
103: c
104: c-----should data be doppler broadened...
105:      if ( TEMPEF.le.1.999 ) then
106: c-----no.
107:          TOOHI = 1
108:      else
109: c-----yes.
110:          ALPHA = AWR/(BOLTZM*TEMPEF)
111: c
112:          ELIMIT = 1.00E+6/ALPHA
113:          if( NSYSO2.gt.0 ) then
114:              if ( MTH.eq.2 ) write(NSYSO2,*)
115:              &          ' ** thresh energy for doppler is ', ELIMIT
116:          end if
117:      end if
118: c-----initialize flag to indicate beginning of section.
119:      ISTART = 1
120: c-----initialize total number of points in section and number of points
121: c-----left to read.
122:      N2IN = NP
123:      N2LEFT = NP
124: c-----initialize broadening indices to first two pages.
125:      HOT1 = 1
126:      HOT2 = NPMAX
127: c-----initialize end of section flag.
128:      MTEND = 0
129: c-----initialize thinning indices.
130:      ITHIN1 = 1

```

src/artcore/file3.f

```

131:      ITHIN2 = 2
132:      ITHIN3 = 2
133: c-----initialize count of points on scratch.
134:      N2SCR = 0
135: c-----initialize last energy read for ascending energy test.
136:      ELAST = 0.0
137: c
138: c      initialize unresolved resonance region parameters for this section
139: c
140: c-----if there is an unresolved resonance region and this section is
141: c-----total, elastic, fission or capture initial indices.
142:      UREACT = UREVAL
143:      UNRES1 = 10000000
144:      UNRES2 = 10000000
145:      IL = 0
146:      IH = 0
147:      if ( UREACT.gt.0 ) then
148:      & if ( MTH.ne.1.and.MTH.ne.2.and.MTH.ne.18.and.MTH.ne.102 )
149:      & UREACT = 0
150: c
151: c *** check write
152: c
153:      if( NSYSO2.gt.0 ) then
154:      & if ( MTH.eq.2 ) write(NSYSO2,*) ' ** ureval eulow,euhig : ',
155:      & UREVAL, EULOW, EUHIGH
156:      & end if
157:      end if
158: c
159: c      load data.
160: c
161: c ----load next page of data at original temperature.
162:      IDOP = 0
163:      KDOP = 0
164: c
165:      if ( MT.eq.2.and.NPTCUT.gt.0 ) then
166:      & if( NSYSO2.gt.0 ) then
167:      & write(NSYSO2,*) ' ** nptcut e1 e2 : ', NPTCUT, XMESH(1),
168:      & XMESH(NPTCUT)
169:      & write(NSYSO2,*) ' ** orignal x1 x2 x3 : ',
170:      & YMESH(1), YMESH(NPTCUT), YMESH(NPTCUT+1)
171:      & write(NSYSO2,*) ' **new x1 x2 : ', CRSTHE(1), CRSTHE(NPTCUT)
172:      & end if
173: c
174:      do 100 I = 1, NPTCUT
175:      & YMESH(I) = CRSTHE(I)
176:      100 continue
177:      end if
178: c
179:      call FILLUP( XMESH, YMESH, NP, NPMAX, KDOP, XCCOLD, DCOLD, XCHOT,
180:      & ECOLD, YCOLD, EHOT )
181: c-----should data be doppler broadened...
182:      if ( TOOHI.le.0 ) then
183:      & IDOP = 1
184:      & call BROADN( NPMAX, IMAXD, IMAXX, IADDMX, XCCOLD, DCOLD, XCHOT,
185:      & XCSAVE, ECOLD, YCOLD, EHOT, ESAVE, WBROAD, IVPOP )
186:      & end if
187: c
188:      MTEND = 1
189:      if( NSYSO2.gt.0 ) then
190:      & write(NSYSO2,*) ' *** np hot3 : ', NP, HOT3
191:      & end if
192:      call THINIT( NPMAX, EHOT, XCHOT )
193:
194:
195:      MINUS3 = 0

```

```

196: *VOCL LOOP,SCALAR
197:      do 110 I = 1, HOT3
198:      & if ( XCHOT(I).lt.0.0 ) MINUS3 = 1
199:      110 continue
200: c
201:      call NORMX( TEMP3, XNORM(1), KSIGN(1), KEXP(1) )
202:      call NORMX( Q, XNORM(2), KSIGN(2), KEXP(2) )
203: c
204:      N2TOT = HOT3
205: c
206:      if ( IDBG.gt.0 ) then
207:      & if ( KDOP.eq.1 ) then
208:      & write(OUTP,7000) IZA, MATH, MTH, FMTHOL,
209:      & (XNORM(L),KSIGN(L),KEXP(L),L=1,2), N2IN, N2TOT,
210:      & MESSAG(1,MESS), MESSAG(2,MESS)
211:      & else
212:      & write(OUTP,7020) IZA, MATH, MTH, FMTHOL,
213:      & (XNORM(L),KSIGN(L),KEXP(L),L=1,2), N2IN, N2TOT
214:      & end if
215:      end if
216: c
217:      MATIN = MATIN + N2IN
218:      MATOUT = MATOUT + N2TOT
219: c
220: c-----print warning message if this section contains negative cross
221: c-----sections.
222:      if ( MINUS3.gt.0 ) write(OUTP,7040)
223: c
224:      7000 format(2X,I6,2I5,2X,A2,3X,2(1X,F8.5,A1,I2),2I7,2X,2A4,A1)
225:      7020 format(2X,I6,2I5,2X,A2,3X,2(1X,F8.5,A1,I2),2I7,2X,
226:      & 'not doppler broaden')
227:      7040 format(19X,'warning...the above section contains negative',
228:      & ' cross sections.')
229: c
230:      if ( OTAPE.gt.0 ) then
231:      & call COPOUT( NPMAX, EHOT, XCHOT )
232:      & call SENDO
233:      & end if
234: c
235:      if( NSYSO2.gt.0 ) then
236:      & write(NSYSO2,*) ' *** mt idop kdop : ', MT, IDOP, KDOP
237:      & write(NSYSO2,*) ' *** np hot3 : ', NP, HOT3
238:      & end if
239: c
240:      if ( IDOP.eq.0 ) then
241:      & do 120 I = 1, NP
242:      & XSANS(I) = YMESH(I)
243:      & continue
244:      & if ( MT.eq.2.and.NPTCUT.gt.0 ) then
245:      & do 130 I = 1, NPTCUT
246:      & XSANS(I) = 0.0
247:      & continue
248:      & end if
249:      & return
250:      & end if
251: c
252: c
253: c
254:      call CLEA( XSANS, NP, 0.0 )
255:      JST = 1
256:      IST = 1
257:      if ( MT.eq.2.and.NPTCUT.gt.0 ) IST = NPTCUT + 1
258: c
259:      do 160 I = IST, NP
260:      & ENOW = XMESH(I)

```

src/artcore/file3.f

```
261: c
262:     if ( I.gt.1.and.ENOW.eq.XMESH(I-1) ) then
263:         XSANS(I) = XSANS(I-1)
264:         go to 160
265:     end if
266: c
267:     do 140 J = JST, HOT3 - 1
268:         ISW = J
269:         if ( ENOW.ge.EHOT(J).and.ENOW.lt.EHOT(J+1) ) go to 150
270: 140 continue
271:     ISW = HOT3 - 1
272: 150 JST = ISW
273:     XX1 = EHOT(ISW)
274:     XX2 = EHOT(ISW+1)
275:     YY1 = XCHOT(ISW)
276:     YY2 = XCHOT(ISW+1)
277:     DELX = XX2 - XX1
278:     if ( ABS(DELX).gt.1.000E-30 ) then
279:         ANS = (YY2*(ENOW-XX1)+YY1*(XX2-ENOW)) / (XX2-XX1)
280:     else
281:         if( NSYSO2.gt.0 ) then
282:             write(NSYSO2,*) ' ** mt i xx1 xx2 yy1 yy2 : ',
283: & MT, I, XX1, XX2, YY1, YY2
284:         end if
285:         ANS = (YY1+YY2)*0.5000
286:     end if
287: c
288: c
289:     ANSO = ANS
290: c
291:     if ( ANS.lt.1.00E-33 ) then
292:         ANSO = 0.0
293:         if ( UHIGH.gt.0.0 ) then
294:             if ( ENOW.ge.ULOW.and.ENOW.le.UHIGH ) ANSO = ANS
295:         end if
296:     end if
297: c
298:     XSANS(I) = ANSO
299: 160 continue
300: c
301: 7060 format(// ' execution terminated'///)
302: c
303: c *** end of process
304: c
305:     return
306: end
```

src/artcore/file5t.f

```

1:      subroutine FILE5T( SDATA, Q,      IEPOS, WORK,  NFG,  NIG,
2:      &                  ADATA, IDATA, NSYSO, EMESH, LENGTH,WORKP, WORKQ
3:      &                  , IDBG )
4: c    &
5: c
6:      real Q(NFG,NFG), SDATA(NFG,NFG), EMESH(NFG), ADATA(LENGTH)
7:      integer IEPOS(2,NFG,NFG), IDATA(LENGTH)
8:      real WORK(NFG)
9:      real*8 WORKP(NFG), WORKQ(NFG), SUMP, DSAVE
10: c
11: c-----initial set
12: c
13: c      NFG2      = NFG*NFG
14: c      NFG4      = NFG*NFG*2
15: c
16: c      call CLEA( Q, NFG2, 0.0 )
17: c      call ICLEA( IEPOS, NFG4, 0 )
18: c
19: c      LIM1      = 5
20: c      LIM2      = NIG - 4
21: c
22: c-----loop of incident energy
23: c
24: c      do 150 LOP = 1, NIG
25: cdel  call clea( work , nfg , 0.0 )
26: c      call DCLEA( WORKP, NFG, 0.0d0 )
27: c      call DCLEA( WORKQ, NFG, 0.0d0 )
28: cdel  work(1) = sdata(1,lop)
29: c      WORKP(1) = DBLE(SDATA(1,LOP))
30: c      SUMP      = WORKP(1)
31: c
32: c      do 100 J = 2, NFG
33: cdel  save      = sdata(j,lop) - sdata(j-1,lop)
34: cdel  if(save.le.0.0) save = 0.0
35: c      DSAVE    = DBLE(SDATA(J,LOP)) - DBLE(SDATA(J-1,LOP))
36: c      if ( DSAVE.le.0.0 ) DSAVE = 0.0
37: cdel  work(j) = save
38: c      WORKP(J) = DSAVE
39: c      SUMP      = SUMP + WORKP(J)
40: c      100      continue
41: c
42: c      if ( SUMP.gt.0.0 ) then
43: c        do 110 J = 1, NFG
44: c          WORKP(J) = WORKP(J) /SUMP
45: c        110      continue
46: c      end if
47: c
48: c      call bmcml( work ,q(1,lop) , iepos(1,1,lop) , nfg )
49: c      call BMCMD( WORKP, WORKQ, IEPOS(1,1,LOP), NFG )
50: c
51: c      do 120 J = 1, NFG
52: c        Q(J,LOP) = WORKQ(J)
53: c      120      continue
54: c
55: c      if ( LOP.gt.LIM1.and.LOP.lt.LIM2 ) go to 150
56: c      if ( LOP.le.LIM1 .or. LOP.ge.LIM2 ) then
57: c
58: c        call CLEA( WORK, NFG, 0.0 )
59: c        WORK(1) = SDATA(1,LOP)
60: c        do 130 J = 2, NFG
61: c          SAVE = SDATA(J,LOP) - SDATA(J-1,LOP)
62: c          if ( SAVE.le.0.0 ) SAVE = 0.0
63: c          WORK(J) = SAVE
64: c        130      continue
65: c

```

```

66:      if ( IDBG.gt.1 ) then
67:        write(NSYSO,7000) LOP, EMESH(LOP)
68:        write(NSYSO,7020)
69:        do 140 I = 1, NFG
70:          write(NSYSO,7040) I, EMESH(I), WORK(I), Q(I,LOP),
71:          &                  (IEPOS(J,I,LOP),J=1,2)
72:        140      continue
73:        write(NSYSO,7020)
74:      end if
75:    end if
76:  150 continue
77: c
78:  7000 format(/
79:    & /1X,10X,'*****'
80:    & 1X,10X,'* thermal inelastic energy distribution * '
81:    & 1X,10X,'*          ( bmc sampling )          * '
82:    & 1X,10X,'*****'
83:    & /1X,15X,' lop = ',I3,10X,'incident energy= ',1PE12.5,' ev'/
84:    & /1X,5X,'no energy(ev) probability q-answer   ians1 ians2')
85:  7020 format(1X,4X,5('-----'),'---')
86:  7040 format(1X,4X,I3,1P,3E12.5,2I6)
87: c
88: c-----output data to adata or idata array
89: c
90: c      +      ( (q(j,i),j=1,nfg) ,
91: c      +      (iepos(1,j,i),iepos(2,j,i),j=1,nfg) , i=1,nfg )
92: c
93: c      ISW      = 0
94: c      cmod  do 180 I = 1, NFG
95: c        do 180 LOP = 1, NFG
96: c          I      = LOP
97: c          if( LOP.gt.NIG ) I = NIG
98: c          do 160 J = 1, NFG
99: c            ISW      = ISW + 1
100: c            ADATA(ISW) = Q(J,I)
101: c          160      continue
102: c          do 170 J = 1, NFG
103: c            ISW      = ISW + 1
104: c            IDATA(ISW) = IEPOS(1,J,I)
105: c            ISW      = ISW + 1
106: c            IDATA(ISW) = IEPOS(2,J,I)
107: c          170      continue
108: c          180      continue
109: c
110: c      return
111: c      end

```

src/artcore/fillup.f

```

1:      subroutine FILLUP( EIN,  XCIN,  NP,    NPMAX, IDOP,  XCCOLD,
2:      &                  DCOLD, XCHOT, ECOLD, YCOLD, EHOT )
3: C=<ARTCORE>=====
4: C Purpose :
5: C
6: C      load next page or pages of cross sections at the original
7: C      temperature into core. insure that the maximum energy
8: C      spacing required for linear-linear interpolatable doppler
9: C      broadened data is not exceeded.
10: C
11: C      if there is an unresolved resonance region insure that there is
12: C      at least two points at the lower and upper energy limits of the
13: C      unresolved region and define indices to lower and upper energy
14: C      limits of the unresolved region. do not add additional energy
15: C      points within the unresolved resonance region.
16: C
17: C Called in : artcore/file3.f
18: C
19: C History :
20: C=====
21:      integer TOOHI, COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HOT3,
22:      &          HOT3M1, UNRES1, UNRES2, UREVAL, UREACT
23: C***** double *****
24:      real*8 ENEXT, ARG, EKM1, DX, DEMAX
25: C***** double *****
26:      common /HEADR/  C1H,  AWR,  L1H,  L2H,  N1H,  N2H
27:      &          MATH,  MFH,  NOSEQ
28:      common /HOTS/   ALPHA,  HOTSY,  TEMPK,  TEMPEF,  N2TAPI,  N2TAPO
29:      common /SLIM/   ISTART,  NOTHIN,  ITHIN1,  ITHIN2,  ITHIN3,  MTEND
30:      common /INDEX/  COLD1,  COLD2,  COLD1P,  COLD2P,  HOT1,  HOT2,
31:      &          HOT3,  HOT3M1,  N2IN,  N2TOT,  N2SCR
32:      common /PAGER/  NPAGE,  NPT2,  NPT3,  NP1P1,  NP2P1
33:      common /FILLER/ N2LEFT,  TOOHI,  ITHRES,  LOAD1,  LOAD2
34:      common /THRESH/ ETHRES,  EMIN
35:      common /RESOLV/ UREVAL,  UREACT,  UNRES1,  UNRES2,  EULOW,  EUHIGH,
36:      &          IL,  IH
37:      common /MAXIE/  DEMAX
38: C
39:      include 'INC/_MAINIO'
40: C
41: C      real*8 OVPI, OVPI2
42: C      common /CONTAC/  OVPI,  OVPI2,  ATOP
43: C
44:      include 'INC/_CONST1'
45: C
46: Cm      common/fillxy/ein(1002),xcin(1002)
47: Cm      common/comm1/ecold(6012)
48: Cm      common/comm2/xccold(6012)
49: Cm      common/comm3/dcold(6012)
50: Cm      common/comm4/ycold(6012)
51: Cm      common/comm5/ehot(6012)
52: Cm      common/comm6/xchot(6012)
53: C
54: C
55:      real XCCOLD(NPMAX)
56:      real DCOLD(NPMAX)
57:      real XCHOT(NPMAX)
58:      real*8 ECOLD(NPMAX)
59:      real*8 YCOLD(NPMAX)
60:      real*8 EHOT(NPMAX)
61: C
62:      real EIN(NP), XCIN(NP)
63: C
64: C      at beginning of section set indices to force immediate read and to
65: C      load up to three pages (up to 6012 points) of data. if this is not

```

```

66: C      the beginning of the section set indices to only load the third
67: C      page (2004 points) of data.
68: C
69: C
70: C-----beginning of section. set indices to force reading when first data
71: C-----point is requested from endf/b data file.
72:      if ( N2IN.gt.NPMAX ) then
73:          write(NSYSO,*) ' ** n2in npmax (fillup) : ', N2IN, NPMAX
74:          stop 999
75:      end if
76: C-----initialize indices to read up to three pages of data points.
77:      LOAD1 = 1
78:      LOAD2 = NPMAX
79:      IDOP = 0
80: Cend
81: C-----initialize sign of cross section to positive.
82:      SIGN = 1.0
83: C-----initialize index to first point to broaden.
84:      HOT3 = 0
85:      HOT3M1 = -1
86: C-----initialize load index to first location.
87:      K = LOAD1
88: C-----if no more points in core read next page (if any).
89:      NFILL = N2LEFT
90:      N2LEFT = N2LEFT - NFILL
91: Cdel call pointi(ein,xcin,nfill)
92:      IFILL = 1
93: C
94:      HOT1 = 1
95:      COLD1 = 1
96: C
97: C-----section will only be copied. turn off unresolved region flag.
98: C
99: C
100:      100 if ( TOOHI.gt.0 ) then
101:          UREACT = 0
102:          do 110 III = 1, N2IN
103:              EHOT(III) = EIN(III)
104:              XCHOT(III) = XCIN(III)
105:          110 continue
106:          N2SCR = 0
107:          MTEND = -1
108:          HOT3 = N2IN
109:          return
110:      end if
111: C
112: C      at the beginning of section decide whether or not to doppler
113: C      broaden. do not doppler broaden any section in which the
114: C      threshold is higher than 1,000,000* kt/a (i.e. sections which
115: C      have a high energy threshold, above which broadening will have
116: C      a negligible effect).
117: C
118: Cdel 10 if(toohi) 60,170,110
119: C-----skip points below threshold.
120:      if ( TOOHI.lt.0 ) then
121:          do 120 IFILL = 1, NFILL
122:              if ( ABS(XCIN(IFILL)).gt.0.0 ) go to 130
123:          120 continue
124:          UREACT = 0
125:          EHOT(1) = EIN(1)
126:          XCHOT(1) = XCIN(1)
127:          EHOT(2) = EIN(2)
128:          XCHOT(2) = XCIN(2)
129:          N2SCR = 0
130:          MTEND = -1

```

src/artcore/fillup.f

```

131:      HOT3      = 2
132:      IDOP      = 1
133:      return
134:    end if
135: C
136: C
137: C
138:      130 IFILL  = IFILL - 1
139:      if ( IFILL.le.0 ) IFILL = 1
140: C-----no doppler broadening if threshold over 1,000,000*kt/a.
141:      AX        = ALPHA*EIN(IFILL)
142:      TOOHI     = 1
143:      ITHRES     = 0
144: C-----never treat total, elastic, fission or capture as a threshold
145: C-----reaction.
146:      if ( MTH.ne.1.and.MTH.ne.2.and.MTH.ne.18.and.MTH.ne.102 ) then
147: Cadd
148:      if( NSYSO2.gt.0 ) then
149:        write(NSYSO2,*)
150:      &      '*** mt( ', MTH, ' ) : doppler broadening is skipped'
151:      end if
152: Cend
153: Cmod if(ax.ge.1000000.0) go to 100
154:      if ( AX.ge.1000000.0 ) go to 100
155:    end if
156: C
157: C-----if cross section is merely incomplete evaluation and not real
158: C-----threshold reaction continue cross section to lower energies as
159: C-----1/v by ignoring starting zero value.
160: C
161:      if ( EIN(IFILL).lt.ETHRES.and.ABS(XCIN(IFILL)).le.0.0 ) then
162:        if ( IFILL.ne.NFILL ) then
163:          if ( EIN(IFILL).eq.EIN(IFILL+1) ) IFILL = IFILL + 1
164:        end if
165:      end if
166: C
167: C-----section will be doppler broadened.
168: C
169:      TOOHI     = 0
170: C-----if first point is negative (e.g. background cross section) reverse
171: C-----sign of cross section.
172:      if ( XCIN(IFILL).lt.0.0 ) SIGN = -1.0
173: C
174: C      define first data point. use either first tabulated point or if
175: C      this is a threshold reaction calculate position of new laboratory
176: C      effective threshold due to doppler broadening and use this as
177: C      first point.
178: C
179:      if ( EIN(IFILL).gt.ETHRES ) then
180: C-----insert one or two (if first point cross section is not zero)
181: C-----points below threshold.
182:      ITHRES     = 1
183:      V          = SQRT(AX) - ATOP
184:      ENEXT      = V*V/ALPHA
185:      if ( ENEXT.lt.ETHRES .or. V.le.0.0 ) ENEXT = EMIN
186:      ECOLD(K)   = ENEXT
187:      ARG        = ALPHA*ECOLD(K)
188:      YCOLD(K)   = SQRT(ARG)
189:      XCCOLD(K)  = 0.0
190:      K          = K + 1
191:      if ( XCIN(IFILL).ne.0 ) then
192: C
193:      ECOLD(K)   = EIN(IFILL)
194:      ARG        = ALPHA*ECOLD(K)
195:      YCOLD(K)   = SQRT(ARG)

```

```

196:      XCCOLD(K)  = 0.0
197:      go to 140
198:    end if
199:  end if
200: C-----use first tabulated point.
201:      ECOLD(K)   = EIN(IFILL)
202:      XCCOLD(K)  = XCIN(IFILL)
203:      ARG        = ALPHA*ECOLD(K)
204:      YCOLD(K)   = SQRT(ARG)
205:      IFILL      = IFILL + 1
206: C
207: C      first point defined. re-initialize unresolved resonance region
208: C      indices if section starts above lower energy of unresolved region.
209: C
210:      140 if ( UREACT.gt.0 ) then
211: C-----if section starts above upper energy limit of unresolved region
212: C-----turn off unresolved region.
213:      if ( ECOLD(LOAD1).ge.EUHIGH ) then
214:        UREACT = 0
215: C-----if section starts above lower energy limit of unresolved region
216: C-----indicate that additional points are not required at the lower
217: C-----energy limit of unresolved region.
218:      else if ( ECOLD(LOAD1).ge.EULOW ) then
219:        IL      = 2
220:        UNRES1  = 1
221:      end if
222:    end if
223:    go to 170
224: C-----if cross section has changed sign interpolate and insert energy
225: C-----point at zero cross section unless cross section at last point
226: C-----was zero.
227:      150 if ( SIGN*XCIN(IFILL).lt.0.0 ) then
228:        SIGN = -SIGN
229:        if ( XCKM1.ne.0 ) then
230:          IPATH = 0
231:          XCCOLD(K) = 0.0
232:          ECOLD(K) = (XCIN(IFILL)*EKM1-XCKM1*EIN(IFILL)) /
233:          &          (XCIN(IFILL)-XCKM1)
234:          if ( ECOLD(K).lt.EKM1 ) ECOLD(K) = EKM1
235:          if ( ECOLD(K).gt.EIN(IFILL) ) ECOLD(K) = EIN(IFILL)
236:          ARG = ALPHA*ECOLD(K)
237:          YCOLD(K) = SQRT(ARG)
238: C-----next input point is acceptable.
239:          IFILL = IFILL + IPATH
240:          go to 170
241:        end if
242:      end if
243: C-----check maximum energy spacing.
244:      160 if ( EIN(IFILL).le.ENEXT ) then
245: C-----energy spacing is o.k. accept next tabulated point.
246:      IPATH = 1
247:      ECOLD(K) = EIN(IFILL)
248:      XCCOLD(K) = XCIN(IFILL)
249:      ARG = ALPHA*ECOLD(K)
250:      YCOLD(K) = SQRT(ARG)
251:    else
252:      IPATH = 0
253:      ECOLD(K) = ENEXT
254:      if ( ECOLD(K).lt.EKM1 ) ECOLD(K) = EKM1
255:      if ( ECOLD(K).gt.EIN(IFILL) ) ECOLD(K) = EIN(IFILL)
256:      XCCOLD(K) =
257:      &      ((EIN(IFILL)-ENEXT)*XCKM1+(ENEXT-EKM1)*XCIN(IFILL)) /
258:      &      (EIN(IFILL)-EKM1)
259:      ARG = ALPHA*ECOLD(K)
260:      YCOLD(K) = SQRT(ARG)

```


src/artcore/fillup.f

```

261:      end if
262:      IFILL = IFILL + IPATH
263: C
264: C      define speed-like terms (see ucrl-50400, vol. 17, part c) and
265: C      slope between points.
266: C
267: 170 if ( K.gt.1 ) then
268:      DX = YCOLD(K) - YCOLD(K-1)
269:      DSCOLD = XCCOLD(K) - XCCOLD(K-1)
270:      if ( DX.ne.0 ) then
271:          DCOLD(K-1) = DSCOLD/(DX*(YCOLD(K)+YCOLD(K-1)))
272:      else
273:          DCOLD(K-1) = 0.0
274:      end if
275: end if
276: C
277: C      define next allowable energy interval. if there is an unresolved
278: C      resonance region insure that there are at least two points at the
279: C      lower and upper energy limits of the unresolved region. do not add
280: C      energy points within the unresolved energy region.
281: C
282: C-----save last point for interpolation.
283:      EKML = ECOLD(K)
284:      XCKML = XCCOLD(K)
285:      ENEXT = DEMAX*ECOLD(K)
286:      K = K + 1
287: Cmod
288: Cmod if(k.le.load2) go to 30
289:      if ( K.gt.LOAD2 ) then
290: C
291:          write(NSYSO,*) ' ** k load2 npmax (fillup) : ', N2IN, LOAD2,
292:          &      NPMAX
293:          stop 999
294: C
295: C      first point defined. load all other points.
296: C
297: C      check for unresolved resonance region. if there is an unresolved
298: C      resonance region define indices to the last point below the
299: C      unresolved resonance region (unres1) and the first point above the
300: C      unresolved resonance region (unres2). insure that there are at
301: C      least two points at the lower and upper energy limits of the
302: C      unresolved resonance region. when the lower limit of the
303: C      unresolved region is found set the index to the upper energy limit
304: C      to a large number to indicate that the upper energy limit is still
305: C      beyond the upper limit of the points loaded into core so far. once
306: C      the upper energy limit of the unresolved region is found the index
307: C      to the upper energy limit will be proper defined.
308: C
309:      else if ( IFILL.le.NFILL ) then
310: C
311:          if ( UREACT.le.0 ) go to 150
312: C-----insure there are at least two points at lower energy limit of
313: C-----unresolved region.
314:          if ( EIN(IFILL).lt.EULOW ) go to 150
315:          if ( EIN(IFILL).eq.EULOW ) then
316:              if ( IL.eq.0 ) UNRES1 = K
317:              IPATH = 1
318:              XCULOW = XCIN(IFILL)
319:          else
320:              if ( IL.lt.1 ) then
321: C-----there are no points at lower energy limit. interpolate and
322: C-----insert point.
323:              UNRES1 = K
324:              XCULOW =
325:          &      ((EIN(IFILL)-EULOW)*XCKML+(EULOW-EKML)*

```

```

326:          &      XCIN(IFILL)) /(EIN(IFILL)-EKML)
327:          else if ( IL.ne.1 ) then
328: C-----insure there are at least two points at upper energy limit of
329: C-----unresolved region.
330:          if ( EIN(IFILL).lt.EUHIGH ) go to 160
331:          if ( EIN(IFILL).eq.EUHIGH ) then
332:              XCUHI = XCIN(IFILL)
333:              IPATH = 1
334:          else
335:              if ( IH.lt.1 ) then
336: C-----there are no points at upper energy limit. interpolate and
337: C-----insert first point.
338:              XCUHI =
339:          &      ((EIN(IFILL)-EUHIGH)*XCKML+(EUHIGH
340:          &      -EKML)*XCIN(IFILL)) /(EIN(IFILL)-EKML)
341:          else if ( IH.ne.1 ) then
342:              go to 150
343:          end if
344:          IPATH = 0
345:          end if
346:          ECOLD(K) = EUHIGH
347:          XCCOLD(K) = XCUHI
348:          ARG = ALPHA*ECOLD(K)
349:          YCOLD(K) = SQRT(ARG)
350:          UNRES2 = K
351:          IH = IH + 1
352:          IFILL = IFILL + IPATH
353:          go to 170
354:          end if
355:          IPATH = 0
356:          end if
357:          ECOLD(K) = EULOW
358:          XCCOLD(K) = XCULOW
359:          ARG = ALPHA*ECOLD(K)
360:          YCOLD(K) = SQRT(ARG)
361:          IL = IL + 1
362:          IFILL = IFILL + IPATH
363:          go to 170
364:          end if
365: C
366: C      all points requested, or all remaining points, have been loaded.
367: C
368: C-----this is the end of the data table.
369: C
370:      LOAD2 = K - 1
371:      HOT2 = LOAD2
372:      MTEND = -1
373: C-----define index to last data point in core.
374:      COLD2 = LOAD2
375:      DCOLD(COLD2) = 0.0
376: C
377:      IDOP = 1
378: C
379: C *** end of process
380: C
381:      return
382:      end

```

src/artcore/fistpr.f

```
1:      subroutine FISTPR( RTEMP, IDBG )
2:      c
3:      c
4:      c
5:      integer OTAPE, OUTP, SCR
6:      c
7:      common /UNITS/  INP,      OUTP,      ITAPE,      OTAPE,      SCR
8:      common /HOTS/   ALPHA,    HOTSY,    TEMPK,    TEMPEF,    N2TAPI,    N2TAPO
9:      common /MATTOT/ MATIN,    MATOUT
10:     common /OKERRT/  ERXC3T,    KERR3T,    MAXERT,    ENER3T(21),    ER3T(21)
11:     common /OKERRC/  ERXC3C,    ERXC30,    KERR3C,    MAXERC,    ENER3C(21),
12:     &      ER3C(21)
13:     common /NORMF/   XNORM(6),    KEXP(6)
14:     character*4 KSIGN
15:     common /NORMC/   KSIGN(6)
16:      c
17:     if ( IDBG.gt.0 ) then
18:       write(OUTP,7000)
19:       PERCNT = 100.0*ER3T(1)
20:       PERCNC = 100.0*ER3C(1)
21:       call NORMX( ENER3T(1), XNORM(1), KSIGN(1), KEXP(1) )
22:       call NORMX( ER3T(1), XNORM(2), KSIGN(2), KEXP(2) )
23:       call NORMX( ENER3C(1), XNORM(3), KSIGN(3), KEXP(3) )
24:       call NORMX( ER3C(1), XNORM(4), KSIGN(4), KEXP(4) )
25:      c
26:       write(OUTP,7020) (XNORM(I),KSIGN(I),KEXP(I),I=3,4), PERCNC,
27:     &      (XNORM(I),KSIGN(I),KEXP(I),I=1,2), PERCNT, RTEMP
28:      c
29:      c
30:       write(OUTP,7040)
31:     endif
32:      c
33:     N2TAPI = 0
34:     N2TAPO = 0
35:     MATIN  = 0
36:     MATOUT = 0
37:      c
38:     7000 format('1',' doppler broaden endf/b cross sections',
39:     &      ' (sigmal 89-1)',' iterative solution')
40:     7020 format(2X,88('-')/
41:     &      2X,'accuracy criteria'/
42:     &      2X,88('-')/
43:     &      14X,'calculation',28X,'thinning'/
44:     &      6X,'energy',3X,'accuracy',3X,
45:     &      'per-cent',10X,'energy',3X,'accuracy',3X,'per-cent'/
46:     &      2X,88('-')/
47:     &      F9.5,A1,I2,F8.5,A1,I2,F11.3,5X,F8.5,A1,I2,F8.5,A1,
48:     &      I2,F11.3/2X,88('-')/
49:     &      3X,'requested target temperature is ',F9.3,' kelvin')
50:     7040 format(2X,88('-')/
51:     &      2X,'      za', '      mat', '      mt', '      endf/b',6X,
52:     &      'kelvin',5X,'q-value', ' points', ' points',7X,
53:     &      'unresolved region'/
54:     &      18X,' format',10X,'in',10X,'ev',
55:     &      '      in', '      out',7X,'e-low',6X,'e-high'/
56:     &      2X,88('-'))
57:      c
58:     return
59:     end
```

src/artcore/iexpn.f

```
1:      subroutine IEXPIN( IPR )
2: C=====
3: C Purpose: initialize exponential value table for inlined exponential
4: C          function used in doppler broadning procedures.
5: C-----
6: C arguments
7: C
8: C i  IPR : printout I/O unit
9: C
10: C=====
11:      include 'INC/_EXPPRM'
12: C
13:      data IEXPI /0/
14: C
15: C-----
16: C
17: C store values : exp(0) to exp(NEXPMX+1) by step 1/NEXPDV
18: C
19:      if ( IEXPI.eq.0 ) then
20:          IEXPI = 1
21:          do 100 I = 0, NEXPDV*(NEXPMX+1)
22:              EXPNN(I) = EXP(DBLE(I)/DBLE(NEXPDV))
23:          100 continue
24:      end if
25: C
26:      return
27:      end
```

src/artcore/intcoh.f

```

1:      subroutine INTCOH( MT,   NP,   NTEMP, INTBRG, EMESH, ENGBRG,SBG,
2:      &                  A,    TEMP,  RTEMP, NSYSO, NSYSO2, ANSXS,ANSSB,
3:      &                  NEBRG,LENWRK,IDBG )
4: c
5: c      this subroutine was added for endf/B6 thermal coherent
6: c      elastic cross section interpolation by k.kaneko ITIRO 26/June/2001
7: c
8:      integer INTBRG(NTEMP)
9:      real    EMESH(NP),  ENGBRG(NEBRG),SBG(NTEMP,NEBRG)
10:     real    A(LENWRK),  TEMP(NTEMP)
11:     real    ANSXS(NP),  ANSSB(NEBRG)
12: c
13: c
14: c
15:     if( NSYSO2.gt.0 ) then
16:       write(NSYSO2,*) ' *** np ntemp mt nebrg : ',
17:       &                NP, NTEMP, MT,NEBRG
18:     end if
19: c
20: c
21: c
22:     IPOSNT = 0
23:     XREQ    = RTEMP
24: c
25:     do 10 NT = 2, NTEMP
26:       if ( XREQ.ge.TEMP(MT-1).and.XREQ.le.TEMP(NT) ) IPOSNT = NT - 1
27:     10 continue
28: c
29:     IPOS1 = IPOSNT
30:     IPOS2 = IPOSNT+1
31:     INTTMP = INTBRG(IPOS2)
32:     X1     = TEMP(IPOS1)
33:     X2     = TEMP(IPOS2)
34: c
35:     if( NSYSO2.gt.0 ) then
36:       write(NSYSO2,*) ' ** iposnt inttmp : ',IPOSNT,INTTMP
37:       write(NSYSO2,*) ' ** t1 rtemp t2 : ',X1,RTEMP,X2
38:     end if
39: c
40: c *** uisng endf/b6 interpolation scheme
41: c
42:     DO 20 I = 1, NEBRG
43:       Y1 = SBG (IPOS1,I)
44:       Y2 = SBG (IPOS2,I)
45:       SBGNOW = 0.0
46:       CALL TERP1(X1,Y1,X2,Y2,XREQ,SBGNOW,INTTMP,NSYSO)
47:       ANSSB(I) = SBGNOW
48: c
49:     if( NSYSO2.gt.0 ) then
50:       write(NSYSO2,'(a,i3,lp4e12.5)')
51:       &                ' ** i enow y1 ans y2 : ',
52:       &                I,ENGBRG(I),Y1, SBGNOW,Y2
53:     end if
54:   20 CONTINUE
55: c
56:     if ( IDBG.gt.1) write(NSYSO,7020) MT, NP, NTEMP, RTEMP
57: c
58: c *** CALCULATE CROSS SECTION
59: c
60:     ANSXS(1) = ANSSB(1)
61:     ISET      = 1
62:     ENEXT     = ENGBRG(ISET+1)
63:     DO 100 I = 2, NP-1
64:       ENOW    = EMESH(I)
65:       IF(ENOW.GE.ENEXT) THEN

```

```

66:                                     ISET = ISET + 1
67:       IF(ISET.GE.NEBRG) ISET = NEBRG-1
68:       ENEXT = ENGBRG(ISET+1)
69:     ENDIF
70: c
71:     if( NSYSO2.gt.0 ) then
72:       write(NSYSO2,'(a,2i4,lp4e12.5)')
73:       &                ' ** I ISET ENOW EBRG (INTCOH) : ',I,ISET,ENOW,
74:       &                ENGBRG(ISET),ENEXT
75:     endif
76: c
77:     ANSXS(I) = ANSSB(ISET)
78:   100 CONTINUE
79:     ANSXS(NP)= ANSSB(NEBRG)
80: c
81:     do 120 I = 1, NP
82:       A(I) = EMESH(I)
83:       ANSXS(I)= ANSXS(I)/EMESH(I)
84:       A(I+NP) = ANSXS(I)
85:   120 continue
86: c
87:     if ( IDBG.gt.1 ) then
88:       write(NSYSO,7000) MT, NP, RTEMP
89:       call WOT6( A, 2, NP, 1, 'grp.','e-xs',' ', NSYSO )
90:     endif
91: c
92:   7000 format(///,31X,'*****'/ ' ',
93:   &          30X,'** thermal coherent elastic cross section  '/// ' ',
94:   &          30X,'*****'/ ' ',
95:   &          30X,'**      mt      : ',I6,'      '/// ' ',
96:   &          30X,'**      np      : ',I6,'      '/// ' ',
97:   &          30X,'**      temperature : ',F9.2,' kelvin      '/// ' ',30X,
98:   &          '*****'/)
99: c
100:   7020 format('1',' ## mt np ntemp rtemp at sub(intcoh) ## '/// ' ',3I6,
101:   &          3F12.5)
102: c
103: c
104: c
105:   return
106:   end

```

```

1:      subroutine INTDBY( MT,      NP,      NTEMP, INTDWA, EMESH, DBY, A,
2:      &                  TEMP,      RTEMP, NSYSO, NSYSO2, ANSXS, ANGDAT,
3:      &                  NPDBY, SBMT2, LENWRK,NBINA1, IDBG )
4:      c
5:      c      this subroutine was added for endf/B6 thermal incoherent
6:      c      elastic cross section interpolation by k.kaneko ITIRO 26/June/2001
7:      c
8:      integer INTDWA(NTEMP)
9:      real EMESH(NP), DBY(NPDBY), A(LENWRK), TEMP(NTEMP)
10:     real ANSXS(NP), ANGDAT(NBINA1,NP)
11:     c
12:     real*8 DWA,E,C2,U,RC2,X1,R1X1,XSEC,C1,X2,UNOW,ANSNOW
13:     c
14:     c
15:     if( NSYSO2.gt.0 ) then
16:       write(NSYSO2,*) ' ** np ntemp mt npdby sbmt2 : ',
17:       &                  NP, NTEMP, MT,NPDBY,SBMT2
18:     end if
19:     c
20:     c
21:     c
22:     IPOSNT = 0
23:     XREQ = RTEMP
24:     c
25:     do 10 NT = 2, NPDBY
26:       if ( XREQ.ge.TEMP(MT-1).and.XREQ.le.TEMP(NT) ) IPOSNT = NT - 1
27:     10 continue
28:     c
29:     INTTMP = INTDWA(IPOSNT+1)
30:     T1 = TEMP(IPOSNT)
31:     T2 = TEMP(IPOSNT+1)
32:     NBIN = NBINA1 - 1
33:     c
34:     if( NSYSO2.gt.0 ) then
35:       write(NSYSO2,*) ' ** iposnt inttmp nbin : ',IPOSNT,INTTMP,NBIN
36:     end if
37:     c
38:     c *** using endf/b6 interpolation scheme
39:     c
40:     Y1 = DBY(IPOSNT)
41:     Y2 = DBY(IPOSNT+1)
42:     INTNOW = INTTMP
43:     DBYNOW = 0.0
44:     c
45:     call TERP1( T1, Y1, T2, Y2, XREQ, DBYNOW, INTNOW, NSYSO )
46:     c
47:     if( NSYSO2.gt.0 ) then
48:       write(NSYSO2,*) ' ** t1 temp t2 : ', T1, XREQ , T2
49:       write(NSYSO2,*) ' ** y1 ans y2 : ', Y1, DBYNOW, Y2
50:     end if
51:     c
52:     if ( IDBG.gt.1) write(NSYSO,7020) MT, NP, NTEMP, RTEMP
53:     c
54:     c *** CALCULATE CROSS SECTION AND ANGULAR DISTRIBUTION
55:     c
56:     C1 = SBMT2*0.500000000
57:     DWA = DBYNOW
58:     DO 100 I = 1 , NP
59:       E = EMESH(I)
60:       C2 = 2*E*DWA
61:       U = -1
62:       RC2 = 1/C2
63:       X1 = EXP(-2*C2)
64:       R1X1 = 1/(1-X1)
65:       XSEC = C1*RC2*(1-X1)

```

```

66:      ANXSX(I) = XSEC
67:      ANGDAT(1,I) = -1
68: C
69:      DO 20 IU=1,NBIN
70:      X2 = EXP(-C2*(1-U))
71:      UNOW = 1 + RC2*LOG((1-X1)/NBIN+X2)
72:      ANSNOW = NBIN*RC2*(EXP(-C2*(1-UNOW))*(C2*UNOW-1)-X2*(C2*U-1))*R1X1
73:      U = UNOW
74:      ANGDAT(IU+1,I) = U
75:      A(IU) = ANSNOW
76: 20 CONTINUE
77:      ANGSAV = A(1) - (ANGDAT(2,I)-A(1))
78:      IF(ANGSAV.LE.-1.0) ANGSAV = -0.9999999
79:      ANGDAT(1,I) = ANGSAV
80:      SAVE = ANGDAT(NBIN+1,I)
81:      ANGSAV = A(NBIN) + (A(NBIN) -ANGDAT(NBIN,I))
82:      IF(ANGSAV.GE.+1.0) ANGSAV = +0.9999999
83:      ANGDAT(NBIN+1,I) = ANGSAV
84: C
85:      if( NSYSO2.gt.0 ) then
86:      WRITE(NSYSO2,105) I,E,XSEC,ANGDAT(1,I),A(1),ANGDAT(2,I)
87:      WRITE(NSYSO2,106) I,E,ANGDAT(NBIN,I),A(NBIN),ANGDAT(NBIN+1,I),SAVE
88:      endif
89: 100 CONTINUE
90: C
91: 105 FORMAT(' I E XSEC ANGDAT(1) : ',I4,6F10.6)
92: 106 FORMAT(' I E ANGDAT(L) : ',I4,6F10.6)
93: C
94: C
95:      do 120 I = 1, NP
96:      A(I) = EMESH(I)
97:      A(I+NP) = ANXSX(I)
98: 120 continue
99: C
100:      if ( IDBG.gt.1 ) then
101:      write(NSYSO,7000) MT, NP, RTEMP
102:      call WOT6( A, 2, NP, 1, 'grp.', 'e-xs', ' ', NSYSO )
103:      write(NSYSO,7010) MT, NP, RTEMP
104:      do 150 LOP = 1, NP
105:      write(NSYSO,7011) LOP,EMESH(LOP), (ANGDAT(J,LOP),J=1,15)
106:      write(NSYSO,7012) (ANGDAT(J,LOP),J=16,NBIN+1)
107: 150 continue
108:      endif
109: C
110: 7000 format(///,31X,'*****'/' ' ,
111: & 30X,'* thermal incoherent elastic cross section *'/' ' ,
112: & 30X,'*****'/' ' ,
113: & 30X,'* mt : ',I6,' *'/' ' ,
114: & 30X,'* np : ',I6,' *'/' ' ,
115: & 30X,'* temperature : ',F9.2,' kelvin *'/' ' ,30X,
116: & '*****'//)
117: C
118: 7010 format(///,31X,'*****'/' ' ,
119: & 30X,'* thermal incoherent elastic angular data *'/' ' ,
120: & 30X,'*****'/' ' ,
121: & 30X,'* mt : ',I6,' *'/' ' ,
122: & 30X,'* np : ',I6,' *'/' ' ,
123: & 30X,'* temperature : ',F9.2,' kelvin *'/' ' ,30X,
124: & '*****'//)
125: C
126: 7011 FORMAT(1H , ' : INC-E > ',I4,1PE12.5,0P,15F7.4)
127: 7012 FORMAT(1H , ' & ANG-D > ',2X,20F7.4)
128: C
129: 7020 format('1',' ## mt np ntemp rtemp at sub(intdby) ## '/' ' ,3I6,
130: & 3F12.5)

```

src/artcore/intdbby.f

```
131: c
132: c
133: c
134:      return
135:      end
```

SAE

src/artcore/inteff.f

```
1:      subroutine INTEFF( RTEMP0,TSTAR ,NTEMP ,NSYSO , IDBG ,
2:      &                  TMPTHS,TMPEFF,INTTMP )
3: C
4: C *** interpolates effective temperature for S.C.T approximation
5: C
6:      real    TMPTHS(NTEMP), TMPEFF(NTEMP)
7:      integer INTTMP(NTEMP)
8: C
9: C *** start of process
10: C
11:      RTEMP   = RTEMP0
12: C
13:      if ( RTEMP.lt.TMPTHS(1) ) then
14:        write(NSYSO,*) ' ** warning at sub(inteff) !!'
15:        write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
16:      &                ' is less than the lowest energy in',
17:      &                ' thermal scattering data !!'
18:      &
19:        write(NSYSO,*) ' ** so use ', TMPTHS(1), ' K data !!'
20:        RTEMP   = TMPTHS(1)
21:      end if
22: C
23:      if ( RTEMP.gt.TMPTHS(NTEMP) ) then
24:        write(NSYSO,*) ' ** warning at sub(inteff) !!'
25:        write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
26:      &                ' is greater than the highest energy in',
27:      &                ' thermal scattering data !!'
28:      &
29:        write(NSYSO,*) ' ** so use ', TMPTHS(NTEMP), ' K data !!'
30:        RTEMP   = TMPTHS(NTEMP)
31:      end if
32: C
33:      TSTAR   = 0.0
34:      NT1     = 1
35:      do 10 NT = 2, NTEMP
36:        if ( RTEMP.gt.TMPTHS(NT-1).and.RTEMP.le.TMPTHS(NT) ) NT1 = NT - 1
37:      10 continue
38:      NT2     = NT1 + 1
39:      X1      = TMPTHS(NT1)
40:      X2      = TMPTHS(NT2)
41:      Y1      = TMPEFF(NT1)
42:      Y2      = TMPEFF(NT2)
43:      INTNOW   = INTTMP(NT2)
44: C
45:      CALL   TERP1(X1,Y1,X2,Y2,RTEMP,TSTAR,INTNOW,NSYSO)
46: C
47:      if( IDBG.gt.0 ) then
48:        write(NSYSO,*) ' ** rtemp tstar : ', RTEMP, TSTAR
49:      end if
50: C
51: C *** end of process
52: C
53:      return
54:      end
```

src/artcore/intrpl.f

```

1:      subroutine INTRPL( IU,    L,    X,    Y,    N,    U,    V )
2: c
3: c interpolation of a single-valued function
4: c this subroutine interpolates, from values of the function
5: c given as ordinates of input data points in an x-y plane
6: c and for a given set of x values (abscissas), the values of
7: c a single-valued function  $y = y(x)$ .
8: c the input parameters are
9: c   iu = logical unit number of standard output unit
10: c   l  = number of input data points
11: c       (must be 2 or greater)
12: c   x  = array of dimension l storing the x values
13: c       (abscissas) of input data points
14: c       (in ascending order)
15: c   y  = array of dimension l storing the y values
16: c       (ordinates) of input data points
17: c   n  = number of points at which interpolation of the
18: c       y value (ordinate) is desired
19: c       (must be 1 or greater)
20: c   u  = array of dimension n storing the x values
21: c       (abscissas) of desired points
22: c the output parameter is
23: c   v  = array of dimension n where the interpolated y
24: c       values (ordinates) are to be displayed
25:
26: c declaration statements
27:      real X(L), Y(L), U(N), V(N)
28:      equivalence(P0,X3), (Q0,Y3), (Q1,T3)
29:      real M1, M2, M3, M4, M5
30:      equivalence(UK,DX), (IMN,X2,A1,M1), (IMX,X5,A5,M5), (J,SW,SA),
31:      & (Y2,W2,W4,Q2), (Y5,W3,Q3)
32: c preliminary processing
33:      L0      = L
34:      LM1     = L0 - 1
35:      LM2     = LM1 - 1
36:      LP1     = L0 + 1
37:      N0      = N
38:      if ( LM2.lt.0 ) then
39:
40: c error exit
41:
42:      write(IU,7000)
43:      go to 200
44:      else
45:      if ( N0.le.0 ) then
46:      write(IU,7020)
47:      go to 200
48:      else
49:      do 100 I = 2, L0
50:      if ( X(I-1).lt.X(I) ) then
51:      else if ( X(I-1).eq.X(I) ) then
52:      go to 170
53:      else
54:      go to 180
55:      end if
56: 100      continue
57:      IPV     = 0
58: c main do-loop
59:      do 160 K = 1, N0
60:      UK      = U(K)
61: c routine to locate the desired point
62:      if ( LM2.eq.0 ) then
63:      I       = 2
64:      go to 120
65:      else if ( UK.ge.X(L0) ) then

```

```

66:      I       = LP1
67:      go to 120
68:      else if ( UK.lt.X(1) ) then
69:      I       = 1
70:      go to 120
71:      else
72:      IMN     = 2
73:      IMX     = L0
74:      end if
75: 110      I     = (IMN+IMX) /2
76:      if ( UK.ge.X(I) ) then
77:      IMN     = I + 1
78:      else
79:      IMX     = I
80:      end if
81:      if ( IMX.gt.IMN ) go to 110
82:      I       = IMX
83: c check if i = ipv
84: 120      if ( I.eq.IPV ) go to 150
85:      IPV     = I
86: c routines to pick up necessary x and y values and
87: c to estimate them if necessary
88:      J       = I
89:      if ( J.eq.1 ) J = 2
90:      if ( J.eq.LP1 ) J = L0
91:      X3      = X(J-1)
92:      Y3      = Y(J-1)
93:      X4      = X(J)
94:      Y4      = Y(J)
95:      A3      = X4 - X3
96:      M3      = (Y4-Y3) /A3
97:      if ( LM2.eq.0 ) then
98:      M2      = M3
99:      M4      = M3
100:      else
101:      if ( J.ne.2 ) then
102:      X2      = X(J-2)
103:      Y2      = Y(J-2)
104:      A2      = X3 - X2
105:      M2      = (Y3-Y2) /A2
106:      if ( J.eq.L0 ) then
107:      M4      = M3 + M3 - M2
108:      go to 130
109:      end if
110:      end if
111:      X5      = X(J+1)
112:      Y5      = Y(J+1)
113:      A4      = X5 - X4
114:      M4      = (Y5-Y4) /A4
115:      if ( J.eq.2 ) M2 = M3 + M3 - M4
116:      end if
117: 130      if ( J.le.3 ) then
118:      M1      = M2 + M2 - M3
119:      else
120:      A1      = X2 - X(J-3)
121:      M1      = (Y2-Y(J-3)) /A1
122:      end if
123:      if ( J.ge.LM1 ) then
124:      M5      = M4 + M4 - M3
125:      else
126:      A5      = X(J+2) - X5
127:      M5      = (Y(J+2)-Y5) /A5
128:      end if
129: c numerical differentiation
130:      if ( I.ne.LP1 ) then

```


src/artcore/intrpl.f

```

131:      W2      = ABS(M4-M3)
132:      W3      = ABS(M2-M1)
133:      SW      = W2 + W3
134:      if ( SW.eq.0.0 ) then
135:         W2     = 0.5
136:         W3     = 0.5
137:         SW     = 1.0
138:      end if
139:      T3      = (W2*M2+W3*M3) /SW
140:      if ( I.eq.1 ) then
141:         T4     = T3
142:         SA     = A3 + A4
143:         T3     = 0.5*(M1+M2-A4*(A3-A4)*(M3-M4)/(SA*SA))
144:         X3     = X3 - A4
145:         Y3     = Y3 - M2*A4
146:         A3     = A4
147:         M3     = M2
148:         go to 140
149:      end if
150:   end if
151:      W3      = ABS(M5-M4)
152:      W4      = ABS(M3-M2)
153:      SW      = W3 + W4
154:      if ( SW.eq.0.0 ) then
155:         W3     = 0.5
156:         W4     = 0.5
157:         SW     = 1.0
158:      end if
159:      T4      = (W3*M3+W4*M4) /SW
160:      if ( I.eq.LP1 ) then
161:         T3     = T4
162:         SA     = A2 + A3
163:         T4     = 0.5*(M4+M5-A2*(A2-A3)*(M2-M3)/(SA*SA))
164:         X3     = X4
165:         Y3     = Y4
166:         A3     = A2
167:         M3     = M4
168:      end if
169: c determination of the coefficients
170:   140      Q2      = (2.0*(M3-T3)+M3-T4) /A3
171:      Q3      = (-M3-M3+T3+T4) /(A3*A3)
172: c computation of the polynomial
173:   150      DX      = UK - P0
174:      V(K)     = Q0 + DX*(Q1+DX*(Q2+DX*Q3))
175:   160      continue
176:      return
177:   end if
178:   170      write(IU,7040)
179:      go to 190
180:   end if
181:   180      write(IU,7060)
182:   190      write(IU,7080) I, X(I)
183:   200      write(IU,7100) L0, N0
184:      return
185: c format statements
186:   7000      format(1X/'   ***(INTRPL)   L = 1 or less.'/)
187:   7020      format(1X/'   ***(INTRPL)   N = 0 or less.'/)
188:   7040      format(1X/'   ***(INTRPL)   identical x values.'/)
189:   7060      format(1X/'   ***(INTRPL)   x values out of sequence.'/)
190:   7080      format('      I =',I7,10X,'X(I) =',E12.3)
191:   7100      format('      L =',I7,10X,'N =',I7/
192:      &      ' error detected in routine INTRPL')
193:      end

```

src/artcore/intth3.f

```

1:      subroutine INTTH3( MT,      NP,      NTEMP, INT,      EMESH, CROSS, A,
2:      &      TEMP,      RTEMP, NSYSO, NSYSO2, ANS,      LENWRK, IDBG )
3:      &      TEMP,      RTEMP, NSYSO, NSYSO2, ANS,      LENWRK )
4:      c
5:      c
6:      c
7:      integer INT(NTEMP)
8:      real EMESH(NP), CROSS(NTEMP,NP), A(LENWRK), TEMP(NTEMP)
9:      real ANS(NP)
10:     c
11:     c
12:     if( NSYSO2.gt.0 ) then
13:       write(NSYSO2,*) ' ** np ntemp mt : ', NP, NTEMP, MT
14:     end if
15:     c
16:     c
17:     c
18:     IPOSNT = 0
19:     INTTMP = 2
20:     XREQ = RTEMP
21:     c
22:     do 100 NT = 2, NTEMP
23:       if ( XREQ.ge.TEMP(NT-1).and.XREQ.le.TEMP(NT) ) IPOSNT = NT - 1
24:     100 continue
25:     c
26:     INTTMP = INT(IPOSNT)
27:     X1 = TEMP(IPOSNT)
28:     X2 = TEMP(IPOSNT+1)
29:     c
30:     if( NSYSO2.gt.0 ) then
31:       write(NSYSO2,*) ' ** iposnt inttmp : ', IPOSNT, INTTMP
32:       write(NSYSO2,*) ' ** x1 temp x2 : ', X1, XREQ, X2
33:     end if
34:     c
35:     *** uisng endf/b3 interpolation scheme
36:     c
37:     do 110 I = 1, NP
38:       Y1 = CROSS(IPOSNT,I)
39:       Y2 = CROSS(IPOSNT+1,I)
40:       INTNOW = INTTMP
41:       if ( Y1.le.0.0 ) INTNOW = 2
42:       if ( Y2.le.0.0 ) INTNOW = 2
43:     c
44:     ANS(I) = 0.0
45:     call TERPl( X1, Y1, X2, Y2, XREQ, ANS(I), INTNOW, NSYSO )
46:   110 continue
47:   c
48:   if ( IDBG.gt.1 ) write(NSYSO,7020) MT, NP, NTEMP, RTEMP
49:   c
50:   do 120 I = 1, NP
51:     A(I) = EMESH(I)
52:     A(I+NP) = ANS(I)
53:   120 continue
54:   c
55:   if ( IDBG.gt.1 ) then
56:     write(NSYSO,7000) MT, NP, RTEMP
57:     call WOT6( A, 2, NP, 1, 'grp.', 'e-xs', ' ', NSYSO )
58:   endif
59:   c
60:   7000 format('0',30X,'*****'/' ' ,
61:   &      30X,' thermal elastic cross section '/' ' ,
62:   &      30X,'*****'/' ' ,
63:   &      30X,' mt : ',I6,' '/' ' ,
64:   &      30X,' np : ',I6,' '/' ' ,
65:   &      30X,' temperature : ',F9.2,' kelvin '/' ' ,30X,

```

```

66:   &      '*****'/'/)
67:   c
68:   7020 format('1',' ## mt np ntemp rtemp at sub(intth3) ## '/' ' ,3I6,
69:   &      3F12.5)
70:   c
71:   c
72:   c
73:   return
74:   end

```

src/artcore/intth4.f

```

1:      subroutine INTTH4( MT,      NRE,      NEANG, NBINF4,NTEMP, NBT,      INT,
2:      &                  TEMP,  ANGDAT,ANSANG,A,      RTEMP,
3:      &                  NSYSO, NSYSO2, LENWRK, IDBG )
4: c    &                  NSYSO, NSYSO2, LENWRK )
5: c
6: c
7: c
8: c
9:      integer INT(NRE), NBT(NRE)
10:     real ANGDAT(NTEMP,NBINF4,NEANG), A(LENWRK), TEMP(NTEMP)
11:     real ANSANG(NBINF4,NEANG)
12: c
13: c
14:     if( NSYSO2.gt.0 ) then
15:       write(NSYSO2,*) ' ** mt nre neang nbinf4 : ',
16:       &                MT, NRE, NEANG, NBINF4
17:     end if
18: c
19: c *** interpolates angular data using akima-method
20: c
21:     do 120 I = 1, NEANG
22:       ENOW      = ANGDAT(1,1,I)
23:       ANSANG(1,I) = ENOW
24: c
25:       do 110 J = 2, NBINF4
26: c
27:         call CLEA( A, NTEMP, 0.0 )
28:         do 100 K = 1, NTEMP
29:           A(K)      = ANGDAT(K,J,I)
30:         100 continue
31: c
32:         ANS      = 0.0
33:         call INTRPL( NSYSO, NTEMP, TEMP, A, 1, RTEMP, ANS )
34: c
35:         if ( ANS.le.-1.0 .or. ANS.ge.+1.00 ) then
36:           write(NSYSO,*)
37:           &      ' ** invalid interpolation data is found !! '
38:           write(NSYSO,*) ' ** ans is ', ANS,
39:           &      ' for file-4 & e-int is ', ENOW
40:         end if
41: c
42:         if ( ANS.le.-1.0 ) ANS = -0.99999
43:         if ( ANS.ge.+1.0 ) ANS = +0.99999
44: c
45:         if ( J.gt.2 ) then
46:           BANGD      = ANSANG(J-1,I)
47:           if ( ANS.le.BANGD ) then
48:             ANSN      = BANGD + 1.00E-6
49:             write(NSYSO,*)
50:             &      ' ** invalid interpolation data is found !! '
51:             write(NSYSO,*) ' ** ang(' , J, ', ', I, ') is reset ',
52:             &      ANSN, ' form ', ANS
53:             ANS      = ANSN
54:           end if
55:         end if
56: c
57:         ANSANG(J,I) = ANS
58:       110 continue
59:     120 continue
60: c
61:     if ( IDBG.gt.1 ) then
62:       write(NSYSO,7000) MT, NRE, NEANG, NBINF4, RTEMP
63:       write(NSYSO,7020) NBT
64:       write(NSYSO,7040) INT
65:       write(NSYSO,7060) (ANSANG(1,J),J=1,NEANG)

```

```

66:       do 130 LOP = 1, NEANG
67:         ISW      = MOD(LOP,5)
68:         if ( ISW.eq.1 .or. LOP.eq.NEANG ) then
69:           write(NSYSO,7080) LOP, -1.0,
70:           &      (ANSANG(J,LOP),J=2,NBINF4),
71:           &      +1.0
72:         end if
73:       130 continue
74:     end if
75: c
76:     7000 format('1',' ## mt nre neang nbinf4 rtemp (intth4) ## '/' ',4I6,
77:     &      F12.5)
78:     7020 format(' ',' : nbt > ',20I6)
79:     7040 format(' ',' : int > ',20I6)
80:     7060 format(' ',' : e-int > ',1P10E11.4)
81:     7080 format(' ',' : ang-d > ',I6,16F7.4/( ' ',17F7.4))
82: c
83: c
84: c *** end of process
85: c
86:     return
87:     end

```

src/artcore/intth6.f

```

1:      subroutine INTTH6( MT,      NTEMP, NFG,      NIG,      NGMAX, EMESH,
2:      &                  ENGDAT,ANGDAT,SIN,      TWORK, TEMP,      RTEMP,
3:      &                  ANSENG,ANSANG,PIJE,      ANGNEW,NSYSO,  NSYSO2,
4:      &                  A,          LENWRK,IDBG  ,LASYM )
5: c
6: C=====
7: c      this subroutine was added for endf/B6 thermal inelastic cross
8: c      section interpolation by k.kaneko ITIRO 26/June/2001
9: C=====
10:      real  EMESH(NFG), TEMP(NTEMP)
11:      real  ENGDAT(NTEMP,NFG,NIG), ANGDAT(NTEMP,4,NGMAX)
12:      real  TWORK(NTEMP), A(LENWRK)
13:      real  ANSENG(NFG,NFG), ANSANG(5,NGMAX)
14:      real  SIN(NGMAX)
15:      real  PIJE(NFG,NFG),ANGNEW(4,NGMAX)
16: C
17: C
18: C
19:      XMIN = 1.0E-15
20: C
21:      if ( NSYSO2.gt.0 ) then
22:          write(NSYSO2,*) ' ** (INTHS) ntemp mt nfg nig ngmax lasym : '
23:      &      NTEMP, MT, NFG, NIG, NGMAX, LASYM
24:      end if
25: C
26: C
27: C
28:      NT1      = 1
29:      do 100 NT = 2, NTEMP
30:          if ( RTEMP.gt.TEMP(NT-1).and.RTEMP.le.TEMP(NT) ) NT1 = NT - 1
31:      100 continue
32:      NT2      = NT1 + 1
33:      X1       = TEMP(NT1)
34:      X2       = TEMP(NT2)
35: C
36:      if ( IDBG.gt.1 ) then
37:          write(NSYSO,7020) MT, NTEMP, NFG, NIG, NGMAX, RTEMP
38: C
39:          write(NSYSO,7000) NIG, NFG, RTEMP
40:          call WOT6( EMESH, 1, NFG, 1, 'grp.', ' ', ' ', NSYSO )
41:      end if
42: C
43: 7000 format('0',30X,'*****')
44:      &      1X,30X,'* thermal inelastic energy mesh for prob.*'
45:      &      1X,30X,'*****')
46:      &      1X,30X,'*      nig      : ',I6,'*'
47:      &      1X,30X,'*      nfg      : ',I6,'*'
48:      &      1X,30X,'* temperature : ',F9.2,' kelvin  *'
49:      &      1X,30X,'*****')
50: C
51: 7020 format('1',' ## mt ntemp nfg nig ngmax rtemp sub(intths) ## '/1X,
52:      &      5I6,F12.5)
53: C
54: C *** interpolate energy distribution data using akima-method
55: C
56:      call CLEA( ANSENG, NFG*NFG, 0.0 )
57:      call CLEA( PIJE , NFG*NFG, 0.0 )
58: C
59:      do 160 I = 1, NIG
60:          call CLEA( A, NFG, 0.0 )
61: C
62:          do 120 J = 1, NFG
63:              call CLEA( TWORK, NTEMP, 0.0 )
64:              do 110 K = 1, NTEMP
65:                  TWORK(K) = ENGDAT(K,J,I)

```

```

66:      110      continue
67: C
68:      ANS      = 0.0
69: C
70:      Y1       = TWORK(NT1)
71:      Y2       = TWORK(NT2)
72:      IF(NTEMP.GT.2.AND.Y1*Y2.GT.0.0) then
73:          call INTRPL( NSYSO, NTEMP, TEMP, TWORK, 1, RTEMP, ANS )
74:      else
75:          ANS = -1.0
76:      endif
77: C
78:      ANSO      = (Y2*(RTEMP-X1)+Y1*(X2-RTEMP)) /(X2-X1)
79: C
80:      if ( ANS.lt.0.0 ) ANS = ANSO
81: C
82:      A(J)      = ANS
83:      120      continue
84: C *** check interpolated data
85:      do 130 J = NFG, 2, -1
86:          A(J)   = DBLE(A(J)) - DBLE(A(J-1))
87:      130      continue
88: C
89:      SUMP      = 0.0
90:      do 140 J = 1, NFG
91:          SAVE   = A(J)
92:          if ( SAVE.gt.0.0 ) then
93:              SUMP = SUMP + SAVE
94:          else
95:              A(J) = 0.0
96:          end if
97:          ANSENG(J,I) = SUMP
98:      140      continue
99:      FACT      = 0.0
100:      if ( SUMP.gt.0.0 ) FACT = 1.00/SUMP
101: C
102:      do 150 J = 1, NFG
103:          ANSENG(J,I) = ANSENG(J,I)*FACT
104: C
105:          if(J.gt.1) then
106:              PIJE(J,I) = ANSENG(J,I) - ANSENG(J-1,I)
107:          else
108:              PIJE(J,I) = ANSENG(J,I)
109:          endif
110:      150      continue
111: C
112:      if ( SUMP.le.0.0 .and. IDBG.gt.0 ) then
113:          write(NSYSO,*)
114:          &      ' ** zero data for this energy ditribution data'
115:          write(NSYSO,*) ' ** source group no & energy is ', I,
116:          &      EMESH(I)
117:      end if
118:      160 continue
119: C
120: C
121:      if ( IDBG.gt.1 ) then
122:          write(NSYSO,7040) NIG, NFG, RTEMP
123:          call WOT6( ANSENG, NIG, NFG, 1, 'down', 'sour', ' ', NSYSO )
124:          write(NSYSO,7050) NIG, NFG, RTEMP
125:          call WOT6( PIJE , NIG, NFG, 1, 'down', 'sour', ' ', NSYSO )
126: C
127: C *** interpolate angular distribution data using akima-method
128: C
129:          write(NSYSO,('1',A))
130:          &      ' ** start of angular data interpolation'

```

src/artcore/intth6.f

```

131:      endif
132: C
133:      J          = 0
134:      call CLEA( ANSANG, 5*NGMAX, 0.0 )
135:      call CLEA( ANGNEW, 4*NGMAX, 0.0 )
136: C
137:      do 200 II = 1, NFG
138:          LAST    = II
139:          if(LASYM.eq.1) LAST = NFG
140:          do 190 JJ = 1, LAST
141: C
142:             J      = J + 1
143:             if( SIN(J).le.0.0 ) go to 190
144: C
145:             A(1)    = 0.0
146:             A(2)    = 0.0
147:             A(3)    = 0.0
148:             A(4)    = 0.0
149: C
150:             do 180 I = 1, 4
151:                 call CLEA( TWORK, NTEMP, 0.0 )
152: C
153:                 do 170 K = 1, NTEMP
154:                     TWORK(K) = ANGDAT(K,I,J)
155:                 170 continue
156: C
157:                 ANS      = 0.0
158:                 Y1       = TWORK(NT1)
159:                 Y2       = TWORK(NT2)
160:                 IF(NTEMP.GT.2.AND.Y1*Y2.GT.0.0) then
161:                     call INTRPL( NSYSO, NTEMP, TEMP, TWORK, 1, RTEMP, ANS )
162:                 else
163:                     ans      = -1.0
164:                 endif
165: C
166:                 ANSO      = (Y2*(RTEMP-X1)+Y1*(X2-RTEMP)) / (X2-X1)
167: C
168:                 if ( ANS.lt.0.0 ) ANS = ANSO
169: C
170:                 A(I)      = ANS
171:             180 continue
172: C *** CHECK INTERPOLATED DATA
173:             IF(A(1).LT.XMIN) THEN
174:                 A(1) = 0.0
175:                 A(2) = 0.0
176:                 A(3) = 0.0
177:                 A(4) = 0.0
178:             ENDIF
179: C
180:             IF(A(1).GT.0.0) THEN
181:                 IF(A(2).LT.XMIN) THEN
182:                     A(2) = 0.0
183:                     A(3) = 0.0
184:                 ENDIF
185:                 IF(ABS(A(2)-1.0).LT.XMIN) THEN
186:                     A(2) = 1.0
187:                     A(4) = 0.0
188:                 ENDIF
189:             ENDIF
190: C
191:             DO 185 I = 1 , 4
192:                 ANGNEW(I,J) = A(I)
193:             185 CONTINUE
194: C
195:             190 CONTINUE

```

```

196: C
197:      if ( NSYSO2.gt.0 ) then
198:          WRITE(NSYSO2,*) ' ** NGMAX II J : ', NGMAX,II,J
199:      endif
200: 200 CONTINUE
201: C
202: C *** CALCULATE ANGULAR DATA
203: C
204:      J          = 0
205:      DO 300 II= 1 , NFG
206:          LAST    = II
207:          IF(LASYM.EQ.1) LAST = NFG
208: C
209:          DO 290 JJ= 1 , LAST
210: C
211:             J      = J + 1
212:             DO 210 K = 1 , 5
213:                 ANSANG(K,J) = 0.0
214:             210 CONTINUE
215:             IF(SIN(J).LE.0.0) GO TO 290
216:             XSNOW      = SIN(J)*ANGNEW(1,J)
217:             IF(XSNOW .LE.0.0) GO TO 290
218: C
219:             XSP        = ANGNEW(2,J)
220:             XSM        = 1.00 - ANGNEW(2,J)
221:             UPLUS      = ANGNEW(3,J)
222:             UMINS      = ANGNEW(4,J)
223: C
224:             IF(UMPLUS.GE.0.50) THEN
225:                 P1 = XSP*2.00*(1.0-UMPLUS)
226:                 P3 = XSP*(2.00*UMPLUS-1.00)
227:             ELSE
228:                 P1 = XSP*2.00*UMPLUS
229:                 P3 = 0.0
230:             ENDIF
231: C
232:             IF(UMINS.GE.0.50) THEN
233:                 P2 = XSM*2.00*(1.0-UMINS)
234:                 P4 = XSM*(2.00*UMINS-1.00)
235:             ELSE
236:                 P2 = XSM*2.00*UMINS
237:                 P4 = 0.0
238:             ENDIF
239: C
240:             P5 = 0.0
241: C
242:             IF(UMPLUS.LE.0.500.AND.UMINS.LE.0.500) THEN
243:                 P5 = 1.0 - 2.0*(XSP*UMPLUS+XSM*UMINS)
244:             ENDIF
245: C
246:             IF(UMPLUS.LE.0.500.AND.UMINS.GT.0.500) THEN
247:                 P5 = XSP*(1.00-2.00*UMPLUS)
248:             ENDIF
249: C
250:             IF(UMPLUS.GT.0.500.AND.UMINS.LE.0.500) THEN
251:                 P5 = XSM*(1.00-2.00*UMINS)
252:             ENDIF
253: C
254:             P5X = 0.0
255:             IF(UMPLUS.LE.0.500) THEN
256:                 P5X = XSP*(1.00-2.0*UMPLUS)
257:             ENDIF
258: C
259:             PSUM      = P1 + P2 + P3 + P4 + P5
260: C

```

src/artcore/intth6.f

```

261:      IF(IDBG.gt.1) THEN
262:      WRITE(NSYSO, '(A,3I5,1P8E12.5)') ' ** J II JJ Pn PSUM : ',
263:      +      J,II,JJ,P1,P2,P3,P4,P5,P5X,PSUM
264:      ENDIF
265: C
266:      RNORM      = 1.0
267:      IF(PSUM.GT.1.0) RNORM = 1.0/PSUM
268: C
269:      P1 = P1 * RNORM
270:      P2 = P2 * RNORM
271:      P3 = P3 * RNORM
272:      P4 = P4 * RNORM
273:      P5 = P5 * RNORM
274:      P5X = P5X * RNORM
275: C
276:      ANSANG(1,J) = P1
277:      ANSANG(2,J) = P1 + P2
278:      ANSANG(3,J) = P1 + P2 + P3
279:      ANSANG(4,J) = P1 + P2 + P3 + P4
280:      ANSANG(5,J) = P5X
281: 290 CONTINUE
282: 300 CONTINUE
283: C
284: C *** CHECK INTERPOLATED ANGULAR DATA
285: C
286:      IF(LASYM.EQ.0) THEN
287:      ISUM      = 0
288:      DO 400 I = 1 , NFG
289:      ISUM      = ISUM + I-1
290:      IPOS      = ISUM + I
291:      SUMP      = ANSANG(4,IPOS) + ANSANG(5,IPOS)
292: C
293:      IF(IDBG.gt.1) then
294:      WRITE(NSYSO,*) ' ** I IPOS SUMP (400) : ',I,IPOS,SUMP
295:      endif
296: C
297:      IF(SUMP.LE.0.0.OR.I.EQ.1) GO TO 390
298: C
299:      DO 380 J = I-1 , 1 , -1
300:      JPOS = ISUM + J
301:      SUMP = ANSANG(4,JPOS) + ANSANG(5,JPOS)
302:      IF(SUMP.GT.0.0) THEN
303:      IPOS = JPOS
304:      GO TO 370
305:      ENDIF
306: C
307:      PSAVE = 0.0
308: C
309:      IF(I.LE.NIG) THEN
310:      PSAVE = PIJE(J,I)
311:      ELSE
312:      IF(J.LE.NIG) PSAVE = PIJE(I,J)
313:      ENDIF
314: C
315:      IF(PSAVE.GT.0.0) THEN
316: C
317: C
318:      IF(IDBG.gt.1) then
319:      WRITE(NSYSO,*) ' MODIFY ANG DATA : J I JPOS IPOS = ',J,I,JPOS,IPOS
320:      endif
321: C
322:      ANSANG(1,JPOS) = ANSANG(1,IPOS)
323:      ANSANG(2,JPOS) = ANSANG(2,IPOS)
324:      ANSANG(3,JPOS) = ANSANG(3,IPOS)
325:      ANSANG(4,JPOS) = ANSANG(4,IPOS)

```

```

326:      ANSANG(5,JPOS) = ANSANG(5,IPOS)
327:      ENDIF
328: 370 CONTINUE
329: 380 CONTINUE
330: 390 CONTINUE
331: 400 CONTINUE
332: C
333:      ELSE
334:      ISUM      = -NFG
335:      DO 450 I = 1 , NIG
336:      ISUM      = ISUM + NFG
337:      IPOS      = ISUM + I
338:      SUMP      = ANSANG(4,IPOS) + ANSANG(5,IPOS)
339: C
340:      IF(IDBG.gt.1) then
341:      WRITE(NSYSO,*) ' ** I IPOS SUMP (450) : ',I,IPOS,SUMP
342:      endif
343: C
344:      IF(SUMP.LE.0.0) GO TO 450
345: C
346:      IF(I.GT.1) THEN
347:      DO 420 J = I-1 , 1 , -1
348:      JPOS = ISUM + J
349:      SUMP = ANSANG(4,JPOS) + ANSANG(5,JPOS)
350:      IF(SUMP.GT.0.0) THEN
351:      IPOS = JPOS
352:      GO TO 410
353:      ENDIF
354: C
355:      PSAVE = PIJE(J,I)
356: C
357:      IF(PSAVE.GT.0.0) THEN
358: C
359:      IF(IDBG.gt.1) then
360:      WRITE(NSYSO,*) ' MODIFY ANG DATA : J I JPOS IPOS = ',J,I,JPOS,IPOS
361:      endif
362: C
363:      ANSANG(1,JPOS) = ANSANG(1,IPOS)
364:      ANSANG(2,JPOS) = ANSANG(2,IPOS)
365:      ANSANG(3,JPOS) = ANSANG(3,IPOS)
366:      ANSANG(4,JPOS) = ANSANG(4,IPOS)
367:      ANSANG(5,JPOS) = ANSANG(5,IPOS)
368:      ENDIF
369: 410 CONTINUE
370: 420 CONTINUE
371:      ENDIF
372: C
373:      IF(I.GE.NFG) GO TO 450
374: C
375:      IPOS      = ISUM + I
376:      DO 440 J = I+1 , NFG
377:      JPOS      = ISUM + J
378:      SUMP      = ANSANG(4,JPOS) + ANSANG(5,JPOS)
379:      IF(SUMP.GT.0.0) THEN
380:      IPOS = JPOS
381:      GO TO 430
382:      ENDIF
383: C
384:      PSAVE = PIJE(I,J)
385: C
386:      IF(PSAVE.GT.0.0) THEN
387: C
388:      IF(IDBG.gt.1) then
389:      WRITE(NSYSO,*) ' MODIFY ANG DATA : J I JPOS IPOS = ',J,I,JPOS,IPOS
390:      endif

```

src/artcore/intth6.f

```
391: c
392:                                ANSANG(1,JPOS) = ANSANG(1,IPOS)
393:                                ANSANG(2,JPOS) = ANSANG(2,IPOS)
394:                                ANSANG(3,JPOS) = ANSANG(3,IPOS)
395:                                ANSANG(4,JPOS) = ANSANG(4,IPOS)
396:                                ANSANG(5,JPOS) = ANSANG(5,IPOS)
397:                                ENDIF
398: 430      CONTINUE
399: 440      CONTINUE
400: 450      CONTINUE
401: C
402:                                ENDIF
403: C
404: C *** print out
405: C
406:   if ( IDEB.gt.1 ) then
407:     write(NSYSO,7060) NFG, RTEMP
408:     JJ = 0
409:     do 510 I = 1, NFG
410:       LAST = I
411:       if(LASYM.eq.1) LAST = NFG
412:       do 510 J = 1, LAST
413:         JJ = JJ + 1
414:         if ( ANSANG(4,JJ).gt.0.0.or. ANSANG(5,JJ).gt.0.0 ) then
415:           write(NSYSO,7080) JJ, J, I, (ANSANG(K,JJ),K=1,5)
416:           write(NSYSO,7090) EMESH(I),EMESH(J),SIN(JJ),
417:             & (ANGNEW(K,JJ),K=1,4)
418:         endif
419:       510      continue
420:     end if
421: C
422: 7040 format('1',30X,'*****')
423:   & 1x,30X,'* thermal inelastic scattering transfer */'
424:   & 1x,30X,'* cumulative probability table listing */'
425:   & 1x,30X,'*****')
426:   & 1x,30X,'* nig : ',I6,' */'
427:   & 1x,30X,'* nfg : ',I6,' */'
428:   & 1x,30X,'* temperature : ',F9.2,' kelvin */'
429:   & 1x,30X,'*****')
430: 7050 format('1',30X,'*****')
431:   & 1x,30X,'* thermal inelastic scattering transfer */'
432:   & 1x,30X,'* probability table listing */'
433:   & 1x,30X,'*****')
434:   & 1x,30X,'* nig : ',I6,' */'
435:   & 1x,30X,'* nfg : ',I6,' */'
436:   & 1x,30X,'* temperature : ',F9.2,' kelvin */'
437:   & 1x,30X,'*****')
438: 7060 format('1',30X,'*****')
439:   & 1x,30X,'* thermal inelastic angular table */'
440:   & 1x,30X,'*****')
441:   & 1x,30X,'* nfg : ',I6,' */'
442:   & 1x,30X,'* temperature : ',F9.2,' kelvin */'
443:   & 1x,30X,'*****')
444: 7080 format(1X,5X,'pa(1,',I4,',',I2,',',I2,') : ',1P,9E11.4)
445: 7090 format(1X,5X,' E-S E-D SIN FIT-D : ',1P8E12.5)
446: C
447: C *** end of process
448: C
449:   return
450:   end
```

src/artcore/intth7.f

```

1:      subroutine INTTH7( MT,      NP,      NTEMP, EMESH, CROSS, TEMP,
2:      &      ANSCRS,A,      RTEMP, LENWRK,NSYSO, NSYSO2, IDBG )
3:      c      &      ANSCRS,A,      RTEMP, LENWRK,NSYSO, NSYSO2 )
4:      c
5:      c
6:      c
7:      c
8:      real EMESH(NP), CROSS(NTEMP,NP), A(LENWRK), TEMP(NTEMP)
9:      real ANSCRS(NP)
10:     c
11:     c
12:     c
13:     if( NSYSO2.gt.0 ) then
14:       write(NSYSO2,*) ' ** np ntemp mt : ', NP, NTEMP, MT
15:     end if
16:     c
17:     c
18:     NT1      = 1
19:     do 100 NT = 2, NTEMP
20:       if ( NTEMP.gt.TEMP(NT-1).and.RTEMP.le.TEMP(NT) ) NT1 = NT - 1
21:     100 continue
22:     NT2      = NT1 + 1
23:     X1       = TEMP(NT1)
24:     X2       = TEMP(NT2)
25:     c
26:     c *** interpolate akima-method
27:     c
28:     call CLEA( ANSCRS, NP, 0.0 )
29:     c
30:     do 120 I = 1, NP
31:       call CLEA( A, NTEMP, 0.0 )
32:       do 110 K = 1, NTEMP
33:         A(K)      = CROSS(K,I)
34:       110 continue
35:     c
36:     ANS       = 0.0
37:     Y1        = A(NT1)
38:     Y2        = A(NT2)
39:     cmodb6th call INTRPL( NSYSO, NTEMP, TEMP, A, 1, RTEMP, ANS )
40:     c
41:     IF(NTEMP.GT.2.AND.Y1*Y2.GT.0.0) then
42:       call INTRPL( NSYSO, NTEMP, TEMP, A, 1, RTEMP, ANS )
43:     c
44:       else
45:       ANS = -1.0
46:     endif
47:   cend
48:   c
49:   ANSO       = (Y2*(RTEMP-X1)+Y1*(X2-RTEMP)) /(X2-X1)
50:   c
51:   if ( ANS.lt.0.0 ) ANS      = ANSO
52:   c
53:   ANSCRS(I)  = ANS
54: 120 continue
55: c
56: if ( IDBG.gt.1) write(NSYSO,7020) MT, NP, NTEMP, RTEMP
57: c
58: c
59: do 130 I = 1, NP
60:   A(I)      = EMESH(I)
61:   A(I+NP)   = ANSCRS(I)
62: 130 continue
63: c
64: if ( IDBG.gt.1) then
65:   write(NSYSO,7000) MT, NP, RTEMP

```

```

66:       call WOT6( A, 2, NP, 1, 'grp.', 'e-xs', ' ', NSYSO )
67:     end if
68:   c
69:   7000 format('0',30X,'*****'/
70:     &      1X,30X,'* thermal inelastic cross section */'
71:     &      1X,30X,'*****'/
72:     &      1X,30X,'*      mt      : ',I6,' */'
73:     &      1X,30X,'*      np      : ',I6,' */'
74:     &      1X,30X,'*      temperature : ',F9.2,' kelvin */'
75:     &      1X,30X,'*****'//)
76:   c
77:   7020 format('1', ' ## mt np ntemp rtemp sub(intth7) ## ',3I6,F12.5)
78:   c
79: c *** end of process
80: c
81:   return
82:   end

```


src/artcore/intths.f

```

1:      subroutine INTTHS( MT,      NTEMP, NFG,      NIG,      NGMAX, EMESH,
2:      &      ENGDAT,ANGDAT,IANG,      TWORK, TEMP,      RTEMP,
3:      &      ANSENG,ANSANG,NSYSO, NSYSO2, A,      LENWRK, IDBG )
4: c      &      ANSENG,ANSANG,NSYSO, NSYSO2, A,      LENWRK )
5: C=====
6: C
7: C=====
8:      integer IANG(NGMAX)
9:      real EMESH(NFG), TEMP(NTEMP)
10:     real ENGDAT(NTEMP,NFG,NIG), ANGDAT(NTEMP,5,NGMAX)
11:     real TWORK(NTEMP), A(LENWRK)
12:     real ANSENG(NFG,NFG), ANSANG(5,NGMAX)
13: C
14: C
15: C
16:     if ( NSYSO2.gt.0 ) then
17:       write(NSYSO2,*) ' ** (INTHS) ntemp mt nfg nig ngmax : ',
18: &      NTEMP, MT, NFG, NIG, NGMAX
19:     end if
20: C
21: C
22: C
23:     NT1      = 1
24:     do 100 NT = 2, NTEMP
25:       if ( RTEMP.gt.TEMP(NT-1).and.RTEMP.le.TEMP(NT) ) NT1 = NT - 1
26:     100 continue
27:     NT2      = NT1 + 1
28:     X1       = TEMP(NT1)
29:     X2       = TEMP(NT2)
30: C
31:     if ( IDBG.gt.1 ) then
32:       write(NSYSO,7020) MT, NTEMP, NFG, NIG, NGMAX, RTEMP
33: C
34:       write(NSYSO,7000) NIG, NFG, RTEMP
35:       call WOT6( EMESH, 1, NFG, 1, 'grp.', ' ', ' ', NSYSO )
36:     end if
37: C
38: 7000 format('0',30X,'*****')
39: &      1X,30X,' thermal inelastic energy mesh for prob. *'
40: &      1X,30X,'*****')
41: &      1X,30X,' nig      : ',I6,'      *'
42: &      1X,30X,' nfg      : ',I6,'      *'
43: &      1X,30X,' temperature : ',F9.2,' kelvin *'
44: &      1X,30X,'*****')
45: C
46: 7020 format('1', ' ## mt ntemp nfg nig ngmax rtemp sub(intths) ## '/1X,
47: &      5I6,F12.5)
48: C
49: C *** interpolate energy distribution data using akima-method
50: C
51:     call CLEA( ANSENG, NFG*NFG, 0.0 )
52: C
53:     do 160 I = 1, NIG
54:       call CLEA( A, NFG, 0.0 )
55: C
56:       do 120 J = 1, NFG
57:         call CLEA( TWORK, NTEMP, 0.0 )
58:         do 110 K = 1, NTEMP
59:           TWORK(K) = ENGDAT(K,J,I)
60:         110 continue
61: C
62:         ANS      = 0.0
63:         call INTRPL( NSYSO, NTEMP, TEMP, TWORK, 1, RTEMP, ANS )
64: C
65:         Y1      = TWORK(NT1)

```

```

66:         Y2      = TWORK(NT2)
67:         ANS0     = (Y2*(RTEMP-X1)+Y1*(X2-RTEMP)) / (X2-X1)
68: C
69:         if ( ANS.lt.0.0 ) ANS = ANS0
70: C
71:         A(J)     = ANS
72:       120 continue
73: C *** check interpolated data
74:       do 130 J = NFG, 2, -1
75:         A(J)     = DBLE(A(J)) - DBLE(A(J-1))
76:       130 continue
77: C
78:       SUMP      = 0.0
79:       do 140 J = 1, NFG
80:         SAVE     = A(J)
81:         if ( SAVE.gt.0.0 ) then
82:           SUMP    = SUMP + SAVE
83:         else
84:           A(J)    = 0.0
85:         end if
86:         ANSENG(J,I) = SUMP
87:       140 continue
88:       FACT      = 0.0
89:       if ( SUMP.gt.0.0 ) FACT = 1.00/SUMP
90: C
91:       do 150 J = 1, NFG
92:         ANSENG(J,I) = ANSENG(J,I)*FACT
93:       150 continue
94: C
95:       if ( SUMP.le.0.0 .and. IDBG.gt.0 ) then
96:         write(NSYSO,*)
97: &      ' ** zero data for this energy ditribution data'
98:         write(NSYSO,*) ' ** source group no & energy is ', I,
99: &      EMESH(I)
100:       end if
101: 160 continue
102: C
103: C
104:     if ( IDBG.gt.1 ) then
105:       write(NSYSO,7040) NIG, NFG, RTEMP
106:       call WOT6( ANSENG, NIG, NFG, 1, 'down', 'sour', ' ', NSYSO )
107: C
108: C *** interpolate angular distribution data using akima-method
109: C
110:       write(NSYSO,('1',A))
111: &      ' ** start of angular data interpolation'
112:     endif
113: C
114:     J          = 0
115:     call ICLEA( IANG, NGMAX, 0 )
116:     call CLEA( ANSANG, 5*NGMAX, 0.0 )
117: C
118:     do 280 II = 1, NFG
119:       do 270 JJ = 1, II
120: C
121:         J      = J + 1
122:         call CLEA( A, 5, 0.0 )
123: C
124:         XS1    = 0.0
125:         if ( II.le.NIG ) then
126:           XS1   = ANSENG(JJ,II)
127:           if ( JJ.gt.1 ) XS1 = XS1 - ANSENG(JJ-1,II)
128:         end if
129: C
130:         XS2    = 0.0

```

src/artcore/intths.f

```

131:         if ( JJ.le.NIG ) then
132:             XS2 = ANSENG(II,JJ)
133:             if ( II.gt.1 ) XS2 = XS2 - ANSENG(II-1,JJ)
134:         end if
135: C
136:         if ( XS1.le.0.0.and.XS2.le.0.0 ) then
137:             SUM = 1.0
138: C
139:             write(NSYSO2,*) ' ** matrix(' ,II,',',JJ,') is zero !!'
140:             write(NSYSO2,*) ' ** ei ej is ', EMESH(II), EMESH(JJ)
141:             write(NSYSO2,*) ' ** skip ang. data for htis data !!'
142: C
143:             if ( IDBG.gt.1 ) then
144:                 write(NSYSO,*) ' ** matrix(' ,II,',',JJ,') is zero !!'
145:                 write(NSYSO,*) ' ** ei ej is ', EMESH(II), EMESH(JJ)
146:                 write(NSYSO,*) ' ** skip ang. data for this data !!'
147:             end if
148:             go to 250
149:         end if
150: C *** check all zero data case
151:         SUM1 = ANG DAT(NT1,4,J) + ANG DAT(NT1,5,J)
152:         SUM2 = ANG DAT(NT2,4,J) + ANG DAT(NT2,5,J)
153: C
154:         if ( SUM1.le.0.0.or.SUM2.le.0.0 ) then
155:             if ( IDBG.gt.1 ) then
156:                 write(NSYSO,*) ' ** special case is found for '
157:                 & J, JJ, II
158:                 write(NSYSO,*) ' ** sum1 = ', SUM1,
159:                 & ' & sum2 = ', SUM2
160:             end if
161: C
162:             if ( SUM1.le.0.0 ) then
163:                 do 170 I = 1, 5
164:                     A(I) = ANG DAT(NT2,I,J)
165:                 170 continue
166:             end if
167:             if ( SUM2.le.0.0 ) then
168:                 do 180 I = 1, 5
169:                     A(I) = ANG DAT(NT1,I,J)
170:                 180 continue
171:             end if
172:             SUM = A(4) + A(5)
173:             go to 250
174:         end if
175: C
176:         do 210 I = 1, 5
177:             call CLEA( TWORK, NTEMP, 0.0 )
178:             do 190 K = 1, NTEMP
179:                 TWORK(K) = ANG DAT(K,I,J)
180:             190 continue
181: C
182:             if ( I.eq.5 ) then
183:                 do 200 K = 1, NTEMP
184:                     TWORK(K) = TWORK(K) + ANG DAT(K,4,J)
185:                 200 continue
186:             end if
187: C
188:             ANS = 0.0
189:             call INTRPL( NSYSO, NTEMP, TEMP, TWORK, 1, RTEMP, ANS )
190: C
191:             Y1 = TWORK(NT1)
192:             Y2 = TWORK(NT2)
193:             ANSO = (Y2*(RTEMP-X1)+Y1*(X2-RTEMP)) / (X2-X1)
194: C
195:             if ( ANS.lt.0.0 ) ANS = ANSO

```

```

196: C
197:         A(I) = ANS
198:         210 continue
199: C *** check interpolated data
200:         do 220 I = 5, 2, -1
201:             A(I) = DBLE(A(I)) - DBLE(A(I-1))
202:         220 continue
203:         SUMP = 0.0
204:         do 230 I = 1, 5
205:             if ( A(I).lt.0.0 ) A(I) = 0.0
206:             SUMP = SUMP + A(I)
207:         230 continue
208:         FACT = 1.0
209:         if(sump.gt.1.0) fact = 1.00 / sump
210: C         if ( SUMP.gt.0.0 ) FACT = 1.00/SUMP
211: Cmodified by itiro k.kaneko 7/28/1998 ****
212: C         p1 + p2 + p3 + p4 + p5 is not always 1.0
213: C
214:         if(SUMP.gt.1.0) FACT = 1.00 / SUMP
215: Cend *****
216:
217:         SUM = 0.0
218:         do 240 I = 1, 4
219:             SUM = SUM + A(I)*FACT
220:             A(I) = SUM
221:         240 continue
222:         A(5) = A(5)*FACT
223:         SUM = SUM + A(5)
224: C
225:         250 continue
226:         do 260 I = 1, 5
227:             ANSANG(I,J) = A(I)
228:         260 continue
229:         if ( SUM.le.0.0 ) then
230:             IANG(J) = 1
231:             if ( IDBG.gt.1 ) then
232:                 write(NSYSO,*)
233:                 & ' > zero data for this angular ditribution data'
234:                 write(NSYSO,*) ' > source & sink energy group : ',
235:                 & II, JJ, J
236:                 write(NSYSO,*) ' > source & sink energy: ', EMESH(II),
237:                 & EMESH(JJ)
238:             end if
239:         end if
240: C
241:         270 continue
242:         if ( IDBG.gt.2)
243:             & write(NSYSO,*) ' ** ngmax ii j : ', NGMAX, II, J
244:         280 continue
245: C
246: C *** reset data if iang(j).eq.1
247: C
248:         J2 = 1
249:         do 340 II = 2, NFG
250:             J1 = J2 + 1
251:             J2 = J1 + II - 1
252:             JSUM = 0
253:             do 290 J = J1, J2
254:                 JSUM = JSUM + IANG(J)
255:             290 continue
256: C
257:             if ( JSUM.gt.0 ) then
258:                 do 310 J = J1 + 1, J2
259:                     if ( IANG(J).eq.1.and.IANG(J-1).eq.0 ) then
260:                         if ( IDBG.gt.1 )

```

src/artcore/intths.f

```

261:      &          write(NSYSO,*) ' ** ii j j-1 : ', II, J, J - 1
262:      do 300 K = 1, 5
263:          ANSANG(K,J) = ANSANG(K,J-1)
264:      300      continue
265:          IANG(J) = 0
266:      end if
267:      310      continue
268: C
269:      do 330 J = J2 - 1, J1, -1
270:          if ( IANG(J).eq.1.and.IANG(J+1).eq.0 ) then
271:              if ( IDBG.gt.1 )
272:                  &          write(NSYSO,*) ' ** ii j j+1 : ', II, J, J + 1
273:              do 320 K = 1, 5
274:                  ANSANG(K,J) = ANSANG(K,J+1)
275:              320      continue
276:                  IANG(J) = 0
277:              end if
278:              330      continue
279:          end if
280:      340      continue
281: C
282:      if ( IANG(1).eq.1 ) then
283:          JPOS = 0
284:          do 350 J = 2, NGMAX
285:              if ( IANG(J).eq.0 ) then
286:                  JPOS = J
287:                  go to 360
288:              end if
289:          350      continue
290:          stop 999
291:          360      continue
292:          do 370 K = 1, 5
293:              ANSANG(K,1) = ANSANG(K,JPOS)
294:          370      continue
295:      end if
296: C
297: C
298: C *** print out
299: C
300:      if ( IDBG.gt.1 ) then
301:          write(NSYSO,7060) NFG, RTEMP
302:          JJ = 0
303:          do 380 I = 1, NFG
304:              do 380 J = 1, I
305:                  JJ = JJ + 1
306:                  if ( ANSANG(4,JJ).gt.0.0 .or. ANSANG(5,JJ).gt.0.0 )
307:                      &          write(NSYSO,7080) JJ, J, I, (ANSANG(K,JJ),K=1,5)
308:              380      continue
309:          end if
310: C
311:      7040 format('1',30X,'*****'/
312:      &          1x,30X,'* thermal inelastic scattering transfer */'
313:      &          1x,30X,'* cumulative probability table listing */'
314:      &          1x,30X,'*****'/
315:      &          1x,30X,'* nfg : ',I6,' */'
316:      &          1x,30X,'* nfg : ',I6,' */'
317:      &          1x,30X,'* temperature : ',F9.2,' kelvin */'
318:      &          1x,30X,'*****//')
319:      7060 format('1',30X,'*****'/
320:      &          1x,30X,'* thermal inelastic angular table */'
321:      &          1x,30X,'*****'/
322:      &          1x,30X,'* nfg : ',I6,' */'
323:      &          1x,30X,'* temperature : ',F9.2,' kelvin */'
324:      &          1x,30X,'*****//')
325:      7080 format(1X,5X,'pa(1,',I4,',',I2,',',I2,') : ',1P,9E11.4)

```

```

326: C
327: C *** end of process
328: C
329:      return
330:      end

```

src/artcore/lastpr.f

```
1:      subroutine LASTPR (IDBG)
2: c
3:      integer OTAPE, OUTP, SCR, UREVAL
4: c
5:      common /COPI/      MFIELD(3)
6:      common /UNITS/     INP,      OUTP,      ITAPE,  OTAPE,  SCR
7:      common /HOTS/      ALPHA,   HOTSY,   TEMPK,   TEMPEF, N2TAPI, N2TAPO
8:      common /NORMF/     XNORM(6),      KEXP(6)
9:      character*4 KSIGN
10:     common /NORMC/     KSIGN(6)
11:     common /EXTEND/     MESS,      DTMAX
12:     common /MATTOT/     MATIN,     MATOUT
13:     common /RESOLV/     UREVAL,     UREACT, UNRES1, UNRES2, EULOW,  EUHIGH,
14:     &                   IL,        IH
15: c
16: c-----list total number of file 3 points read and written.
17: c
18:     if ( IDBG.gt.0 ) then
19:         if ( UREVAL.gt.0 ) then
20:             call NORMX( EULOW, XNORM(1), KSIGN(1), KEXP(1) )
21:             call NORMX( EUHIGH, XNORM(2), KSIGN(2), KEXP(2) )
22:             write(OUTP,7000) MATIN, MATOUT
23:             &         (XNORM(L), KSIGN(L), KEXP(L), L=1,2)
24: c
25:         else
26:             write(OUTP,7020) MATIN, MATOUT
27:         end if
28:     endif
29: c-----increment total point counts for tape.
30:     N2TAPI = N2TAPI + MATIN
31:     N2TAPO = N2TAPO + MATOUT
32: c
33: c-----write warning message if cross section extension used.
34: c
35:     if ( DTMAX.gt.0.0 ) then
36:         call NORMX( DTMAX, XNORM(1), KSIGN(1), KEXP(1) )
37:         write(OUTP,7040) XNORM(1), KSIGN(1), KEXP(1)
38:     end if
39: c
40:     N2TAPI = 0
41:     N2TAPO = 0
42:     MATIN = 0
43:     MATOUT = 0
44: c
45:     7000 format(2X,88('-')/
46:     &         39X,'mat totals',2I7,1X,F8.5,A1,I2,1X,F8.5,A1,I2,' ext'/
47:     &         2X,88('-'))
48:     7020 format(2X,88('-')/
49:     &         39X,'mat totals',2I7,12X,'none'/
50:     &         2X,88('-'))
51: c
52:     7040 format(///' extension'/
53:     &         2X,9('-')/
54:     &         2x,'cross section extension can be avoided by'/
55:     &         2x,'thinning data or doppler broadening in steps'/
56:     &         2x,'of less than',F8.5,A1,I2,' kelvin')
57: c
58:     return
59:     end
```

src/artcore/libout.f

```

1:      subroutine LIBOUT( ADATA, IDATA, NOUT, JREC3,  TITLE, MATT,
2:      &
3:      &
4:      C=<ARTCORE>=====
5:      C Purpose : output doppler-broaden cross sections.
6:      C Called in : main000.f
7:      C Calls : UDATE
8:      C=====
9:      C
10:     include 'INC/_PARAM'
11:     include 'INC/_MAINIO'
12:  C
13:     common /CONTRL/  MATD,  ZA,  ELUNR,  EHUNR,  CRSMIN,  LEMORY,
14:     &
15:     &
16:     common /HEAD1/  NPTS,  NTDATA,  NUNR,  NUNR2,  NLEV,  NBINA,
17:     &
18:     &
19:     &
20:     &
21:     &
22:     &
23:     common /HEAD2/  MTINFO(MTMAX),  MTPAR(MTMAX),  ISTMT(MTMAX),
24:     &
25:     &
26:     &
27:     &
28:     &
29:     &
30:  C
31:     common /HEAD3/  NOGAM,  NOGAM3,  MTGAM(MTMAX),  NGTYPE(MTMAX),
32:     &
33:     &
34:     &
35:     &
36:     &
37:     &
38: Cadded for version-2 by jais k.kaneko 10/30/1991
39:  C
40:     common /F6CONT/  LF6,  LSTF6,  MTTOF6(MTMAX),  EF61L(MTMAX),
41:     &
42:     &
43:  C
44:     real ADATA(*)
45:     integer IDATA(*)
46:  C
47:     character*16 MATT
48:     character*120 TITLE
49:     character*8 HDATE
50:  C
51: C----- read library
52:  C
53:     call UDATE( HDATE )
54:  C
55:     rewind NOUT
56:     if ( IVERSN.eq.2 ) then
57:         write(NOUT) MATT(1:8), HDATE, TITLE,
58:         &
59:         &
60:         &
61:         &
62:         &
63:         &
64:         &
65:     else if ( IVERSN.eq.3 ) then

```

```

66:         TITLE(113:120) = HDATE
67:         write(NOUT) MATT(1:16), TITLE,
68:         &
69:         &
70:         &
71:         &
72:         &
73:         &
74:         &
75:     end if
76:  C
77:     write(NOUT) MTINFO, MTPAR, ISTMT, IENDMT, NEUMT, LCTMT, NEANG,
78:     &
79:     &
80:     &
81:  C
82:     if( JREC3.ne.0 ) then
83:         if(MTINFO(120).eq.0) then
84:             write(NOUT) (ADATA(I),I=1,NTDATA)
85:         else if(MTINFO(120).gt.0) then
86:             write(NOUT) (ADATA(I),I=1,NTDATA+NPTS) ! dopp-scat
87:         else
88:             write(NSYSO,*) ' XXX(LIBOUT) Wrong MTINFO(120) value.',
89:             &
90:             ' MTINFO(120)=' ,MTINFO(120)
91:         end if
92:     end if
93:  C
94:     rewind NOUT
95:  C
96:     ZA      = RDUMMY(1)
97:     MATD     = RDUMMY(2)
98:  C
99:     if ( NSYSO2.gt.0 ) then
100:         write(NSYSO2,*) ' ** tlab za matd : ', TLAB, ZA, MATD
101:     end if
102:  C
103:     if ( LF6.eq.1 ) then
104:         ISW      = LSTF6 - 1
105:         do 100 I = 1, NMT
106:             ISW      = ISW + 1
107:             MTTOF6(I) = IDATA(ISW)
108:         continue
109:     C
110:         do 110 I = 1, NMT
111:             ISW      = ISW + 1
112:             EF61L(I) = ADATA(ISW)
113:         continue
114:     C
115:         do 120 I = 1, NMT
116:             ISW      = ISW + 1
117:             EF62U(I) = ADATA(ISW)
118:         continue
119:     C
120:     end if
121:  C ***** added for t-free version
122:  C
123:     ITFREE      = 0
124:     LENTHS      = 0
125:     LENU3R      = 0
126:     LSTTHS      = 0
127:     LSTU3R      = 0
128:  C
129:     if ( LSTDOP.gt.0 ) then
130:         ITFREE = 1

```

src/artcore/libout.f

```

131:      LENTHS = IDATA(LSTDOP)
132:      LENU3R = IDATA(LSTDOP+1)
133:      LSTTHS = 0
134:      if ( LENTHS.gt.0 ) LSTTHS = LSTDOP + 2
135:      LSTU3R = 0
136:      if ( LENU3R.gt.0 ) then
137:        LSTU3R = LSTDOP + 2
138:        if ( LENTHS.gt.0 ) LSTU3R = LSTTHS + LENTHS
139:      end if
140:    end if
141: C
142:      NTU3R = 0
143:      NURT = 0
144:      NBNU3R = 0
145:      IFISUN = 0
146:      if ( LSTU3R.gt.0 ) then
147:        NTU3R = IDATA(LSTU3R)
148:        NURT = IDATA(LSTU3R+1)
149:        NBNU3R = IDATA(LSTU3R+2)
150:        IFISUN = IDATA(LSTU3R+3)
151:      end if
152: C
153:      NTTHSC = 0
154:      NPTHE = 0
155:      LCTTHE = 0
156:      LTCTHE = 0
157:      NRETHE = 0
158:      NETHE = 0
159:      NBNTHE = 0
160:      NPTHIN = 0
161:      NIGTH = 0
162:      NFGTH = 0
163:      MXNGTH = 0
164:      NATOMT = 0
165:      NPTCUT = 0
166:      EMAXTH = 0.0
167:      EHITHE = 0.0
168:      LOCTHE = 1
169: C
170:      if ( LSTTHS.gt.0 ) then
171:        NTTHSC = IDATA(LSTTHS)
172:        NPTHE = IDATA(LSTTHS+1)
173:        LCTTHE = IDATA(LSTTHS+2)
174:        NRETHE = IDATA(LSTTHS+3)
175:        NETHE = IDATA(LSTTHS+4)
176:        NBNTHE = IDATA(LSTTHS+5)
177:        NPTHIN = IDATA(LSTTHS+6)
178:        NIGTH = IDATA(LSTTHS+7)
179:        NFGTH = IDATA(LSTTHS+8)
180:        MXNGTH = IDATA(LSTTHS+9)
181:        NATOMT = IDATA(LSTTHS+10)
182:        NPTCUT = IDATA(LSTTHS+11)
183:        EMAXTH = ADATA(LSTTHS+12)
184:        EHITHE = ADATA(LSTTHS+13)
185: C
186:        LOCTHE = LSTTHS + 14 + 3*NTTHSC + NPTHE + NPTHE*NTTHSC
187:        &      + NRETHE + NRETHE + NTTHSC*NBNTHE*NETHE + NPTHIN
188:        &      + NPTHIN*NTTHSC + NFGTH + NTTHSC*NFGTH*NIGTH + NTTHSC*5
189:        &      *MXNGTH
190:      end if
191: C
192: C      print out library information
193: C
194: C      the following is always print out in MVPART.
195: C      if ( IDBG.gt.0 ) then

```

```

196: C
197:      if ( IVERSN.eq.2 ) then
198:        write(NSYSO,7000) MATT(1:8), HDATE, TITLE
199:      else
200:        write(NSYSO,7010) MATT(1:16), HDATE, TITLE(1:112)
201:      end if
202:      write(NSYSO,7020) NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA, NBINE,
203:      &      NNU, NMT
204:      write(NSYSO,7040) NNK, IGFLAG, LFI, LNU, ITHRM, ISTU, IENDU,
205:      &      LSUNR, LSNU, NBINA1, NBINE1, LF6, LSTF6, LSSF, LSTDOP
206:      write(NSYSO,7060) NNUD, LNUD, NNF, LSNUD, NCAP, NCAPG
207:      write(NSYSO,7080) ATW, TLAB, ETHERM, ESAB, ELOW, EHI, TSTAR
208:      if ( NLEV.gt.0 ) write(NSYSO,7100) (MTTHR(I),I=1,NLEV)
209:      if ( NCAP.gt.0 ) write(NSYSO,7120) (MTCAP(I),I=1,NCAP)
210:      if ( NCAPG.gt.0 ) write(NSYSO,7140) (MTCAPG(I),I=1,NCAPG)
211: C
212: 7000 format('1',20X,'*****')
213:      &      1X,20X,'* new mvp material library information */'
214:      &      1X,20X,'*      (after doppler broadenning) */'
215:      &      1X,20X,'*****'
216:      &      1X,20X,'*      material id = ',A8,' */'
217:      &      1X,20X,'*****'
218:      &      1X,20X,'*      production date : ',A8,' */'
219:      &      1X,20X,'*****'
220:      &      /1X,10X,' comment : ',A60/1X,10X,'      : ',A60//)
221: 7010 format('1',20X,'*****')
222:      &      1X,20X,'* new mvp material library information */'
223:      &      1X,20X,'*      (after doppler broadenning) */'
224:      &      1X,20X,'*****'
225:      &      1X,20X,'*      material id = ',A16,' */'
226:      &      1X,20X,'*****'
227:      &      1X,20X,'*      production date : ',A8,' */'
228:      &      1X,20X,'*****'
229:      &      /1X,10X,' comment : ',A60/1X,10X,'      : ',A52//)
230: 7020 format(1X,15X,'npts : no of smooth data grid record.....',I10/
231:      &      1X,15X,'ntdata: total record of #3 record (words)...',I10/
232:      &      1X,15X,'nunr : no of unresolved prob. tables .....',I10/
233:      &      1X,15X,'nunr2 : length of unresolved prob. table ...',I10/
234:      &      1X,15X,'nlev : no of threshold reactions .....',I10/
235:      &      1X,15X,'nbina : no of bins in angular dis. table....',I10/
236:      &      1X,15X,'nbine : no of bins in energy dis. table....',I10/
237:      &      1X,15X,'nnu : length of nu-data .....',I10/
238:      &      1X,15X,'nmt : total no of reaction considered.....',I10)
239: 7040 format(1X,15X,'nnk : max no of sub-sections in file-5.....',I10/
240:      &      1X,15X,'igflag: gamma production data flag .....',I10/
241:      &      1X,15X,'lfi : flag of fission reaction .....',I10/
242:      &      1X,15X,'lnu : nu-data flag (1/2=poly/tab).....',I10/
243:      &      1X,15X,'ithrm: flag of thermal scattering .....',I10/
244:      &      1X,15X,'istu : upper energy mesh point of u.r. ....',I10/
245:      &      1X,15X,'iendu : lower energy mesh point of u.r. ....',I10/
246:      &      1X,15X,'lsunr : start position of u.r. table .....',I10/
247:      &      1X,15X,'lsnu : start position of nu-data.....',I10/
248:      &      1X,15X,'nbina1: nbina + 1 .....',I10/
249:      &      1X,15X,'nbine1: nbine + 1 .....',I10/
250:      &      1X,15X,'lf6 : flag of ddx data (file6) .....',I10/
251:      &      1X,15X,'lstf6 : start posiotn for ddx control .....',I10/
252:      &      1X,15X,'lssf : unresolved lssf flag .....',I10/
253:      &      1X,15X,'lstdop: start posiotn for doppler data.....',I10)
254: 7060 format(1X,15X,'nnud : length of delayed nu-data.....',I10/
255:      &      1X,15X,'lnud : delayed nu-data flag(1/2=poly/tab)...',I10/
256:      &      1X,15X,'nnf : no of precursor families .....',I10/
257:      &      1X,15X,'lsnud : start position of delayed nu .....',I10/
258:      &      1X,15X,'ncap : no of capture reactions .....',I10/
259:      &      1X,15X,'ncapg : no of gamma yield reac. by capture..',I10)
260: 7080 format(1X,15X,'atw : atomic weight in n.m.u. ....',F12.3

```

src/artcore/libout.f

```

261:      &      /1X,15X,'tlab : laboratory temperature in kelvin ...',F12.3
262:      &      /1X,15X,'etherm: upper energy for thermal ela. mod...',F12.3
263:      &      /1X,15X,'esab : highest energy for thermal inela ...',F12.3
264:      &      /1X,15X,'elow : lower energy defined (ev) .....',F12.5
265:      &      /1X,15X,'ehi : upper energy defined (ev) .....',F12.1
266:      &      /1X,15X,'tstar : effective temperature in kelvin ....',F12.3
267:      & )
268:      7100 format(1X,15X,'mtthr : threshold reaction mt id .....',10I4)
269:      7120 format(1X,15X,'mtcap : reaction id for capture .....',10I4)
270:      7140 format(1X,15X,'mtcapg: reaction id for (capture,gamma)....',10I4)
271: C
272:      write(NSYSO,7160)
273:      write(NSYSO,7180)
274:      write(NSYSO,7200)
275:      do 130 MT = 1, NMT
276:      if ( MTINFO(MT).le.0 ) go to 130
277:      write(NSYSO,7220) MT, MTINFO(MT), ISTMT(MT), IENDMT(MT),
278:      &      MTPAR(MT), NEUMT(MT), QVAL(MT), QVAL2(MT)
279:      130 continue
280:      write(NSYSO,7200)
281: C
282:      7160 format('1',20X,'*****'//
283:      &      1X,20X,'* #2 record information (part 1) **'//
284:      &      1X,20X,'*****'//)
285:      7180 format(1X,10X,' mt mtinfo istmt iendmt mtpar neumt
286:      &      ' qval(ev) qval2(ev)'//)
287:      7200 format(1X,9X,7('-----'))
288:      7220 format(1X,10X,I3,5I8,1P2E12.5)
289: C
290:      write(NSYSO,7240)
291:      write(NSYSO,7260)
292:      write(NSYSO,7280)
293: C
294:      do 140 MT = 1, NMT
295: Cm if(mtinfo(mt).le.0) go to 60
296: Cb6 if(mtinfo(mt).le.0.and.nkf5(mt).le.0) go to 60
297:      ISAVE = IABS(NKF5(MT))
298:      if ( MTINFO(MT).le.0.and.ISAVE.eq.0 ) go to 140
299:      write(NSYSO,7300) MT, LSTF3(MT), LCTMT(MT), NEANG(MT),
300:      &      LSTF4(MT), NKF5(MT), NEF5(MT), LSTF5(MT)
301: Cm
302: Cm if(isave.gt.0) then
303: Cm jsave = isave + 1
304: Cm write(nsyso,113) (lff5s(j,mt),j=1,jsave)
305: Cm write(nsyso,114) (nef5s(j,mt),j=1,jsave)
306: Cm write(nsyso,115) (lstf5s(j,mt),j=1,jsave)
307: Cm endif
308:      140 continue
309:      write(NSYSO,7280)
310: C
311:      7240 format('1',20X,'*****'//
312:      &      1X,20X,'* #2 record information (part 2) **'//
313:      &      1X,20X,'*****'//)
314:      7260 format(1X,10X,' mt lstf3 lctmt neang lstf4 nkf5
315:      &      'nef5 lstf5')
316:      7280 format(1X,9X,7('-----'))
317:      7300 format(1X,10X,I3,8I8)
318:      7320 format(1X,10X,' lff5s : ',10I8)
319:      7340 format(1X,10X,' nef5s : ',10I8)
320:      7360 format(1X,10X,' lstf5s: ',10I8)
321: C
322:      if ( LF6.eq.1 ) then
323:      write(NSYSO,7380)
324:      write(NSYSO,7400)
325:      do 150 MT = 1, NMT

```

```

326:      if ( MTTOF6(MT).le.0 ) go to 150
327:      write(NSYSO,7420) MT, MTTOF6(MT), EF61L(MT), EF62U(MT)
328:      150 continue
329:      write(NSYSO,7400)
330:      end if
331: C
332:      7380 format('1',20X,'*****'//
333:      &      1X,20X,'* #2 record information (ddx data) **'//
334:      &      1X,20X,'*****'//)
335:      &      1X,10X,' mt mttof6 ef61l(ev) ef62u(ev)'//)
336:      7400 format(1X,9X,4('-----'))
337:      7420 format(1X,10X,I3,I8,3X,1P,2E12.5)
338: C
339: C *** print tfree control data
340: C
341:      if ( ITFREE.gt.0 ) then
342:      write(NSYSO,7440) LSTDOP, LSTTHS, LSTU3R, LENTHS, LENU3R
343: C
344:      if ( LSTU3R.gt.0 ) then
345:      write(NSYSO,7460) NTU3R, NURT, NBNU3R, IFISUN
346:      LSTU3 = LSTU3R + 3
347:      ISW0 = LSTU3 + NTU3R
348:      write(NSYSO,7480) (ADATA(LSTU3+I),I=1,NTU3R)
349:      write(NSYSO,7500) (ADATA(ISW0+I),I=1,NTU3R)
350:      end if
351:      if ( LSTTHS.gt.0 ) then
352:      write(NSYSO,7520) NTHSC, NPTHE, LCTTHE, NRETHER, NETHE,
353:      &      NBNTHE, NPETHIN, NIGTH, NFGTH, MXNGTH, NATOMT,
354:      &      NPTCUT, EMAXTH, EHITHE, LOCTHE
355:      LSTTSC = LSTTHS + 13
356:      ISW1 = LSTTSC + NTHSC
357:      ISW2 = LSTTSC + NTHSC*2
358:      write(NSYSO,7540) (ADATA(LSTTSC+I),I=1,NTHSC)
359:      write(NSYSO,7560) (ADATA(ISW1+I),I=1,NTHSC)
360:      if ( NPTHE.gt.0 ) write(NSYSO,7580)
361:      &      (IDATA(ISW2+I),I=1,NTHSC)
362:      end if
363:      end if
364: C
365: C
366:      7440 format('1',20X,'*****'//
367:      &      1X,20X,'* tfree control data information **'//
368:      &      1X,20X,'*****'//)
369:      &      1X,15X,'lstdop: start address of temp. dep. data ...',I10/
370:      &      1X,15X,'lstths: start address of thermal scattering...',I10/
371:      &      1X,15X,'lstu3r: start address of u.r.p. table .....',I10/
372:      &      1X,15X,'lenths: record length of thermal scattering...',I10/
373:      &      1X,15X,'lenu3r: record length of u.r.p. table .....',I10/
374:      &      )
375:      7460 format('0',15X,'***** unresolved probability data control ****'//
376:      &      1X,15X,'ntu3r : no of temperature in u.r.p. data ...',I10/
377:      &      1X,15X,'nurt : no of energy in file-2 u.r. eval....',I10/
378:      &      1X,15X,'nbnu3r: no of broability bin .....',I10/
379:      &      1X,15X,'ifisun: fission reaction flag(0/1:no/yes)...',I10)
380:      7480 format(' temperatures ... ',11F10.3)
381:      7500 format(' log(temp.) .... ',11F10.6)
382: C
383:      7520 format(/
384:      &      1X,15X,'***** thermal scattering data control ****'//
385:      &      1X,15X,'nthsc: no of temperature in th.sc. data ...',I10/
386:      &      1X,15X,'npthe : no of thermal elastic energy mesh...',I10/
387:      &      1X,15X,'lctthe: flame of elastic scattering angle...',I10/
388:      &      1X,15X,'nrethe: interpolation scheme for inci. eng...',I10/
389:      &      1X,15X,'nethe : no of incident energy in els. scat...',I10/
390:      &      1X,15X,'nbnthe: no of angle bins f9r elastic scat...',I10/

```

src/artcore/libout.f

```
391:      &      1X,15X,'npthin: no of thermal inel. energy mesh 1... ',I10/
392:      &      1X,15X,'nigth : no of thermal inel. energy mesh 2... ',I10/
393:      &      1X,15X,'nfgth : no of thermal inel. energy mesh 3... ',I10/
394:      &      1X,15X,'mxngth: =(nfg)*(nfg+1)/2 ..... ',I10/
395:      &      1X,15X,'natomt: no of atoms in molecule ..... ',I10/
396:      &      1X,15X,'nptcut: no of cut data in origianl mt=2 .... ',I10/
397:      &      1X,15X,'emaxth: = esav ..... ',F10.5/
398:      &      1X,15X,'ehithe: highest energy of isotropic els. sc',F10.5/
399:      &      1X,15X,'locthe: location of original thermal els. sc',I10/)
400: C
401: 7540 format(' temperatures .... ',11F10.3)
402: 7560 format(' effective temp... ',11F10.3)
403: 7580 format(' interpol. method. ',11I10)
404: C
405:      end if
406: C
407: C
408: C
409: C *** end of process
410: C
411:      return
412:      end
```


src/artcore/lish6.f

```

1:      SUBROUTINE LISTH6( NSYSO ,NTTHSC ,NPTHE ,NTHELS ,NEBRG ,
2:      *                  NPDBY ,NPTHIN ,NIGTH ,NFGTH ,MXNGTH ,
3:      *                  NPTCUT ,ENERGY ,NPTS ,IOPT ,LASYM ,
4:      *                  TMPTHS ,TMPELS ,EMESH3 ,INTELS ,EBRAG ,
5:      *                  SBG ,EMESH7 ,CRSTH7 ,EMESH5 ,ETRAN7 ,
6:      *                  ANGTH7 ,SIN ,CRSMT2 ,DBY ,SBMT2 )
7: C
8: C *** PRINT TEMP. DEPENDENT THERMAL SCATTERING DATA
9: C
10:      real      ENERGY(NPTS)
11: C
12:      real      TMPTHS(NTTHSC)
13:      real      EMESH3(NPTHE)
14:      real      TMPELS(NTHELS)
15:      integer    INTELS(NTHELS)
16:      real      EBRAG(NEBRG) ,SBG(NTHELS,NEBRG) ,DBY(NPDBY)
17:      real      EMESH7(NPTHIN)
18:      real      CRSTH7(NTTHSC,NPTHIN)
19:      real      EMESH5(NFGTH)
20:      real      ETRAN7(NTTHSC,NFGTH,NIGTH)
21:      real      ANGTH7(NTTHSC,4,MXNGTH)
22:      real      SIN (MXNGTH)
23:      real      CRSMT2(NPTCUT)
24: C
25: C *** START OF PROCESS
26: C
27: C
28: C *** PRINT THERMAL ELASTIC CROSS SECTION
29: C
30:      IF(NPTHE.GT.0) THEN
31: C
32:      WRITE(NSYSO, 21)
33:      WRITE(NSYSO, 22) (EMESH3(I),I=1,NPTHE)
34: C
35:      IF(NEBRG.GT.0) THEN
36:      WRITE(NSYSO,101)
37:      WRITE(NSYSO,112)          (TMPELS(K),K=1,NTHELS)
38:      WRITE(NSYSO,103)
39:      DO 110 I = 1 , NEBRG
40:      WRITE(NSYSO,104) I,EBRAG(I) ,(SBG(K,I),K=1,NTHELS)
41: 110 CONTINUE
42:      WRITE(NSYSO,103)
43:      WRITE(NSYSO,105)          (INTELS(K),K=1,NTHELS)
44:      WRITE(NSYSO,103)
45:      ENDIF
46: C
47:      IF(NPDBY.GT.0) THEN
48:      WRITE(NSYSO,111) SBMT2
49:      WRITE(NSYSO,112) (TMPELS(K),K=1,NTHELS)
50:      WRITE(NSYSO,114) (DBY(I),I=1,NPDBY)
51:      ENDIF
52:      ENDIF
53: C
54: 21 FORMAT(/10X,'*****' ,
55: *        /10X,'*      THERMAL ELASTIC UNION ENEGY MESH * ' ,
56: *        /10X,'*****' )
57: 22 FORMAT(' ENEGY(EV) : ' ,1P10E12.5)
58: C
59: 101 FORMAT(/10X,'*****' ,
60: *        /10X,'*      THERMAL COHERENT STRUCTURE FACTOR * ' ,
61: *        /10X,'*****' )
62: 102 FORMAT(5X,'NO ENERGY(EV)' ,12(F7.1,2H K,:))
63: 103 FORMAT(1X,13('-----'))
64: 104 FORMAT(3X,I4,F9.6,2X,1P12E12.5)
65: 105 FORMAT(1H ,5X,' INTERP.          ',12(I12,:))
66: C
67: 111 FORMAT(/10X,'*****' ,
68: *        /10X,'*      DEBYE-WALLER INTEGRAL (INCOHERENT) * ' ,
69: *        /10X,'*      BOUND XS IS ' ,F10.4,' BARN * ' ,
70: *        /10X,'*****' )
71: 112 FORMAT(5X,'NO ENERGY(EV)' ,12(F10.1,2H K,:))
72: 114 FORMAT(5X,'INTEGRAL VALUE' ,1P12E12.5)
73: C
74: C *** PRINT THERMAL INELASTIC X-SECTION
75: C
76:      IF(NPTHIN.GT.0) THEN
77: C
78:      WRITE(NSYSO,301)
79:      WRITE(NSYSO,302)          (TMPTHS(K),K=1,NTTHSC)
80:      WRITE(NSYSO,303)
81:      DO 310 I = 1 , NPTHIN
82:      WRITE(NSYSO,304) I,EMESH7(I) ,(CRSTH7(K,I),K=1,NTTHSC)
83: 310 CONTINUE
84:      WRITE(NSYSO,303)
85: C
86:      ENDIF
87: C
88: 301 FORMAT(/30X,'*****' ,
89: *        /30X,'*      THERMAL INELASTIC CROSS SECTION * ' ,
90: *        /30X,'*****' )
91: 302 FORMAT(5X,'NO ENERGY(EV)' ,12(F7.1,2H K,:))
92: 303 FORMAT(1X,13('-----'))
93: 304 FORMAT(3X,I4,F9.6,2X,12F9.4)
94: C
95: C *** PRINT THERMAL INELASTIC MATRIX DATA
96: C
97:      IF(NFGTH .GT.0) THEN
98: C
99:      WRITE(NSYSO,401)
100:      DO 430 J = 1 , NIGTH
101: CDEL IF(J.GT.10.AND.J.LT.(NIGTH-10)) GO TO 430
102:      IF(IOPT.LE.1) THEN
103:      IF(J.GT.10.AND.J.LT.(NIGTH-10)) GO TO 430
104:      ENDIF
105: C
106:      WRITE(NSYSO,402) J,EMESH5(J) ,(TMPTHS(K),K=1,NTTHSC)
107:      WRITE(NSYSO,403)
108:      DO 420 I = 1 , NFGTH
109:      SUM = 0.0
110:      DO 410 K = 1 , NTTHSC
111:      SUM = SUM + ETRAN7(K,I,J)
112: 410 CONTINUE
113:      IF(SUM.GT.0.0) THEN
114:      WRITE(NSYSO,404) I,EMESH5(I) ,(ETRAN7(K,I,J),K=1,NTTHSC)
115:      ENDIF
116: 420 CONTINUE
117:      WRITE(NSYSO,403)
118: 430 CONTINUE
119:      ENDIF
120: C
121: 401 FORMAT(/10X,'*****' ,
122: *        /10X,'*      THERMAL INELASTIC ENERGY TRANSFER * ' ,
123: *        /10X,'*****' )
124: 402 FORMAT(/10X,'SOURCE ENERGY NO = ' ,I3,5X,
125: *        'INCIDENT ENERGY= ' ,F10.6,' EV' ,
126: *        //5X,'NO SINK E.(EV)' ,12(F7.1,2H K,:))
127: 403 FORMAT(1X,13('-----'))
128: 404 FORMAT(4X,I3,F9.6,2X,12F9.6)
129: C
130: C *** PRINT THERMAL INELASTIC ANGULAR DATA

```

src/artcore/listh6.f

```

131: C
132:     IF(MXNGTH.GT.0) THEN
133: C
134:     WRITE(NSYSO,501) MXNGTH
135:     J      = 0
136:     DO 530 JJ= 1 , NFGTH
137: C
138:     IF(IOPT.LE.1) THEN
139:     IF(JJ.GT.10.AND.JJ.LE.(NFGTH-9)) THEN
140:         J = J + JJ
141:         GO TO 530
142:     ENDIF
143:     ENDIF
144: C
145:     WRITE(NSYSO,502) JJ,EMESH5(JJ),(TMPHS(K),K=1,NTTHSC)
146:     WRITE(NSYSO,503)
147: C
148:     LAST = JJ
149:     IF(LASYM.EQ.1) LAST = NFGTH
150: C
151:     DO 520 II= 1 , JJ
152:     J      = J + 1
153:     IF(SIN(J).GT.0.0) THEN
154:     DO 515 I = 1 , 4
155:     DO 510 K = 1 , NTTHSC
156:     SUM      = SUM + ANGTH7(K,I,J)
157: C
158:     WRITE(NSYSO,504) J,I,SIN(J),(ANGTH7(K,I,J),K=1,NTTHSC)
159:     515 CONTINUE
160:     ENDIF
161:     520 CONTINUE
162:     WRITE(NSYSO,503)
163:     530 CONTINUE
164: C
165: CDEL  WRITE(NSYSO,*) ' ** J MXNGTH : ',J,MXNGTH
166:     ENDIF
167: C
168:     501 FORMAT(/10X,'*****',
169:     *      /10X,'* THERMAL INELASTIC ANGULAR DATA *',
170:     *      /10X,'* < NO OF DATA = ',I6,' > *',
171:     *      /10X,'*****')
172:     502 FORMAT(/10X,'SOURCE ENERGY NO = ',I3,5X,
173:     *      'SOURCE ENERGY= ',F10.6,' EV',
174:     *      //2X,' J K SIN(BARN) :',12(F8.1,2H K,:))
175:     503 FORMAT(1X,13('-----'))
176:     504 FORMAT(1X,I4,I3,1PE12.5,0P,12F10.7)
177: C
178: C *** PRINT THERMAL ELASTIC CROSS SECTION FOR FREE-GAS
179: C
180:     IF(NPTCUT.GT.0) THEN
181: C
182:     NN      = NPTCUT
183:     LOOP = (NN+1)/2
184:     WRITE(NSYSO,601)
185: C
186:     DO 600 LOP = 1 , LOOP
187:     LOP1 = (LOP-1)*2 + 1
188:     LOP2 = LOP *2
189:     KOP1 = NPTS + 1 - LOP1
190:     KOP2 = NPTS + 1 - LOP2
191: C
192:     IF(LOP2.LE.NN) THEN
193:     WRITE(NSYSO,602) LOP1,ENERGY(KOP1),CRSMT2(LOP1),
194:     *      LOP2,ENERGY(KOP2),CRSMT2(LOP2)
195:     ELSE
196:     WRITE(NSYSO,602) LOP1,ENERGY(KOP1),CRSMT2(LOP1)
197:     ENDIF
198:     600 CONTINUE
199:     ENDIF
200: C
201:     601 FORMAT(/10X,' *****',
202:     *      /10X,' * THERMAL ELASTIC X-SECTION FOR FREE-GAS * ',
203:     *      /10X,' *****',
204:     *      / 2X,' NO ENERGY(EV)',2X,'X-SECTION',
205:     *      ,4X,' NO ENERGY(EV)',2X,'X-SECTION')
206:     602 FORMAT(I6,2X,1P2E12.5,I6,2X,1P2E12.5)
207: C
208: C *** END OF PROCESS
209: C
210:     RETURN
211:     END

```

src/artcore/lisths.f

```

1:      subroutine LISTHS( NSYSO, NTHSC,NPTHE, NRETHE,NETHE, NBNTHE,
2:      &                  NPTHIN,NIGTH, NFGTH, MXNGTH,NPTCUT,ENERGY,NPTS,
3:      &                  TMPTHS,EMESH3,CRSTH3,NBTTH4,INTTH4,ANGTH4,
4:      &                  EMESH7,CRSTH7,EMESH5,ETRAN7,ANGTH7,CRSMT2 )
5: c
6: c *** print temp. dependent thermal scattering data
7: c
8:      real ENERGY(NPTS)
9: c
10:     real TMPTHS(NTHSC)
11:     real EMESH3(NPTHE)
12:     real CRSTH3(NTHSC,NPTHE)
13:     integer NBTTH4(NRETHE)
14:     integer INTTH4(NRETHE)
15:     real ANGTH4(NTHSC,NBNTHE,NETHE)
16:     real EMESH7(NPTHIN)
17:     real CRSTH7(NTHSC,NPTHIN)
18:     real EMESH5(NFGTH)
19:     real ETRAN7(NTHSC,NFGTH,NIGTH)
20:     real ANGTH7(NTHSC,5,MXNGTH)
21:     real CRSMT2(NPTCUT)
22: c
23: c *** start of process
24: c
25: c
26: c *** print thermal elastic cross section
27: c
28:     if ( NPTHE.gt.0 ) then
29: c
30:         write(NSYSO,7000)
31:         write(NSYSO,7020) (TMPTHS(K),K=1,NTHSC)
32:         write(NSYSO,7040)
33:         do 100 I = 1, NPTHE
34:             write(NSYSO,7060) I, EMESH3(I), (CRSTH3(K,I),K=1,NTHSC)
35:         100 continue
36:         write(NSYSO,7040)
37: c
38: 7000 format('1',10X,'*****'/'
39: &          1X,10X,'*   thermal elastic cross section   *'/'
40: &          1X,10X,'*****'/'
41: 7020 format(1X,5X,'no energy(ev) ',12(F7.1,' k',:))
42: 7040 format(1X,1X,13('-----'))
43: 7060 format(1X,3X,I4,F9.6,2X,12F9.4)
44: c
45:     end if
46: c
47: c *** print thermal elastic angular data
48: c
49:     if ( NETHE.gt.0 ) then
50: c
51:         write(NSYSO,7080)
52:         write(NSYSO,7160) (NBTTH4(I),I=1,NRETHE)
53:         write(NSYSO,7180) (INTTH4(I),I=1,NRETHE)
54: c
55:         do 120 J = 1, NETHE
56:             if ( J.gt.10.and.J.lt.(NETHE-10) ) go to 120
57: c
58:             write(NSYSO,7100) J, ANGTH4(1,1,J), (TMPTHS(K),K=1,NTHSC)
59:             write(NSYSO,7120)
60:             PROB = 0.0
61:             DEL = 1.00/FLOAT(NBNTHE)
62:             do 110 I = 2, NBNTHE
63:                 PROB = PROB + DEL
64:                 write(NSYSO,7140) I, PROB, (ANGTH4(K,I,J),K=1,NTHSC)
65: 110 continue

```

```

66:             write(NSYSO,7120)
67: 120 continue
68: c
69:     end if
70: c
71: 7080 format('1',10X,'*****'/'
72: &          1X,10X,'*   thermal elastic angular data listing *'/'
73: &          1X,10X,'*****'/'
74: 7100 format(// ' ',10X,'energy no = ',I3,5X,'incident energy= ',F10.6,
75: &          ' ev',// ' ',5X,'no cum. prob ' ',12(F7.1,' k',:))
76: 7120 format(1X,1X,13('-----'))
77: 7140 format(1X,3X,I4,F9.6,2X,12F9.6)
78: 7160 format(1X,5X,'nbt : ',20I6)
79: 7180 format(1X,5X,'int : ',20I6)
80: c
81: c *** print thermal inelastic x-section
82: c
83:     if ( NPTHIN.gt.0 ) then
84: c
85:         write(NSYSO,7200)
86:         write(NSYSO,7220) (TMPTHS(K),K=1,NTHSC)
87:         write(NSYSO,7240)
88:         do 130 I = 1, NPTHIN
89:             write(NSYSO,7260) I, EMESH7(I), (CRSTH7(K,I),K=1,NTHSC)
90: 130 continue
91:         write(NSYSO,7240)
92: c
93:     end if
94: c
95: 7200 format('1',30X,'*****'/'
96: &          1X,30X,'*   thermal inelastic cross section   *'/'
97: &          1X,30X,'*****'/'
98: 7220 format(1X,5X,'no energy(ev) ',12(F7.1,' k',:))
99: 7240 format(1X,1X,13('-----'))
100: 7260 format(1X,3X,I4,F9.6,2X,12F9.4)
101: c
102: c *** print thermal inelastic matrix data
103: c
104:     if ( NFGTH.gt.0 ) then
105: c
106:         write(NSYSO,7280)
107:         do 160 J = 1, NIGTH
108:             if ( J.gt.10.and.J.lt.(NIGTH-10) ) go to 160
109: c
110:             write(NSYSO,7300) J, EMESH5(J), (TMPTHS(K),K=1,NTHSC)
111:             write(NSYSO,7320)
112:             do 150 I = 1, NFGTH
113:                 SUM = 0.0
114:                 do 140 K = 1, NTHSC
115:                     SUM = SUM + ETRAN7(K,I,J)
116: 140 continue
117:                 if ( SUM.gt.0.0 ) then
118:                     write(NSYSO,7340) I, EMESH5(I),
119:                     &          (ETRAN7(K,I,J),K=1,NTHSC)
120:                 end if
121: 150 continue
122:             write(NSYSO,7320)
123: 160 continue
124:         end if
125: c
126: 7280 format('1',10X,'*****'/'
127: &          1X,10X,'*   thermal inelastic energy transfer   *'/'
128: &          1X,10X,'*****'/'
129: 7300 format(//1X,10X,'source energy no = ',I3,5X,'incident energy= ',
130: &          F10.6,' ev'//1X,5X,'no sink e.(ev)',12(F7.1,' k',:))

```

src/artcore/lisths.f

```

131: 7320 format(1X,1X,13('-----'))
132: 7340 format(1X,4X,I3,F9.6,2X,12F9.6)
133: c
134: c *** print thermal inelastic angular data
135: c
136:       if ( MXNGTH.gt.0 ) then
137: c
138:         write(NSYSO,7360) MXNGTH
139:         J      = 0
140:         do 190 JJ = 1, NFGTH
141: c
142:           if ( JJ.gt.10.and.JJ.le.(NFGTH-5) ) then
143:             J      = J + JJ
144:             go to 190
145:           end if
146: c
147:           write(NSYSO,7380) JJ, EMESH5(JJ), (TMPTHS(K),K=1,NTTHSC)
148:           write(NSYSO,7400)
149: c
150:           do 180 II = 1, JJ
151:             J      = J + 1
152:             do 180 I = 1, 5
153:               SUM      = 0.0
154:               do 170 K = 1, NTTHSC
155:                 SUM      = SUM + ANGTH7(K,I,J)
156:             170       continue
157:             if ( SUM.gt.0.0 ) then
158:               write(NSYSO,7420) J, I, (ANGTH7(K,I,J),K=1,NTTHSC)
159:             end if
160:           180       continue
161:           write(NSYSO,7400)
162:         190       continue
163: c
164:         write(NSYSO,*) ' ** j mxngth : ', J, MXNGTH
165:       end if
166: c
167: 7360 format('1',10X,'*****'/
168: &          1X,10X,' * thermal inelastic angular data      '/
169: &          1X,10X,' *      < no of data = ',I6,' >      '/
170: &          1X,10X,'*****'/)
171: 7380 format(/
172: &          /1X,10X,'source energy no = ',I3,5X,
173: &          'source energy= ',F10.6,' ev'/
174: &          /1X,2X,' j k',12(F9.1,' k',:))
175: 7400 format(1X,1X,13('-----'))
176: 7420 format(1X,1X,I4,I2,1P,12E11.4)
177: c
178: c *** print thermal elastic cross section for free-gas
179: c
180:       if ( NPTCUT.gt.0 ) then
181: c
182:         NN      = NPTCUT
183:         LOOP     = (NN+1) /2
184:         write(NSYSO,7440)
185: c
186:         do 200 LOP = 1, LOOP
187:           LOP1    = (LOP-1)*2 + 1
188:           LOP2    = LOP*2
189:           KOP1    = NPTS + 1 - LOP1
190:           KOP2    = NPTS + 1 - LOP2
191: c
192:           if ( LOP2.le.NN ) then
193:             write(NSYSO,7460) LOP1, ENERGY(KOP1), CRSMT2(LOP1), LOP2,
194:             &          ENERGY(KOP2), CRSMT2(LOP2)
195:           else

```

```

196:             write(NSYSO,7460) LOP1, ENERGY(KOP1), CRSMT2(LOP1)
197:           end if
198:         200       continue
199:       end if
200: c
201: 7440 format('1',10X,'*****'/
202: &          1X,10X,' * thermal elastic x-section for free-gas */
203: &          1X,10X,' *****'/
204: &          1X,2X,' no energy(ev)',2X,'x-section',4X,
205: &          ' no energy(ev)',2X,'x-section')
206: 7460 format(1X,I6,2X,1P2E12.5,I6,2X,1P2E12.5)
207: c
208: c *** end of process
209: c
210:       return
211:     end

```

src/artcore/lisu3r.f

```

1:      subroutine LISU3R( NSYSO, NURT,  NBNU3R,NTU3R, IFISUN,TMPU3R,
2:      &                  EUNRTB,SIGEAV,SIGFAV,SIGTAV,PROBUN,SIGETB,
3:      &                  SIGFTB,SIGTTB )
4: c
5: c *** print temp. dependent unresolved probability data
6: c
7:      real TMPU3R(NTU3R)
8:      real EUNRTB(NURT)
9:      real SIGEAV(NURT)
10:     real SIGFAV(NURT)
11:     real SIGTAV(NURT)
12:     real PROBUN(NBNU3R,NURT)
13:     real SIGETB(NTU3R,NBNU3R,NURT)
14:     real SIGFTB(NTU3R,NBNU3R,NURT)
15:     real SIGTTB(NTU3R,NBNU3R,NURT)
16: c
17:     character*12 IDREAC(3)
18: c
19:     data IDREAC /' total      ', ' elastic    ', ' fission    '/
20: c
21: c *** start of printing to nsyso device
22: c
23:     write(NSYSO,7000)
24:     write(NSYSO,7020)
25:     write(NSYSO,7040)
26:     do 100 J = 1, NURT
27:       write(NSYSO,7060) J, EUNRTB(J), SIGTAV(J), SIGEAV(J), SIGFAV(J)
28: 100 continue
29:     write(NSYSO,7040)
30: c
31: 7000 format('1',12X,'*****'/
32:      &      1X,12X,'* unresolved resonance average x-section */'
33:      &      1X,12X,'*****'/
34: 7020 format(/1X,5X,'no energy(ev) sig-t(barn) sig-e(barn) ',
35:      &      'sig-f(barn)')
36: 7040 format(1X,3X,5('-----'))
37: 7060 format(1X,4X,I3,1P4E12.5)
38: c
39: c *** print temperature dependent x-section table
40: c
41:     do 130 LOP = 1, 3
42:       if ( LOP.eq.3.and.IFISUN.eq.0 ) go to 130
43: c
44:       write(NSYSO,7080) IDREAC(LOP)
45:       do 120 J = 1, NURT
46:         write(NSYSO,7100) J, EUNRTB(J), (TMPU3R(K),K=1,NTU3R)
47:         write(NSYSO,7120)
48:         do 110 I = 1, NBNU3R
49:           if ( LOP.eq.1 ) write(NSYSO,7140) I, PROBUN(I,J),
50:           &      (SIGTTB(K,I,J),K=1,NTU3R)
51:           if ( LOP.eq.2 ) write(NSYSO,7140) I, PROBUN(I,J),
52:           &      (SIGETB(K,I,J),K=1,NTU3R)
53:           if ( LOP.eq.3 ) write(NSYSO,7140) I, PROBUN(I,J),
54:           &      (SIGFTB(K,I,J),K=1,NTU3R)
55: 110 continue
56:         write(NSYSO,7120)
57: 120 continue
58: 130 continue
59: c
60: 7080 format('1',20X,'*****'/
61:      &      1X,20X,'* unresolved resonance probability table */'
62:      &      1X,20X,'*      (reaction:',A12,')      */'
63:      &      1X,20X,'*****'/
64: 7100 format(/1X,10X,'energy no = ',I3,5X,'incident energy= ',1PE12.5,
65:      &      ' ev',0P/
66:      &      /1X,5X,'no probability',12(F7.1,' k',:))
67: 7120 format(1X,1X,I3('-----'))
68: 7140 format(1X,4X,I3,1PE11.4,0P,12F9.4)
69: c
70: c *** end   of printing
71: c
72:     return
73:     end

```

src/artcore/mend.f

```
1:      subroutine MEND
2: c
3:      integer OTAPE, OUTP, SCR
4: c
5:      common /HEADR/  C1H,    C2H,    L1H,    L2H,    N1H,    N2H,
6: &      MATH,    MFH,    MTH,    NOSEQ
7:      common /UNITS/  INP,    OUTP,    ITAPE,    OTAPE,    SCR
8: c
9:      MATH      = 0
10:     MTH       = 0
11:     MFH       = 0
12:     write(OTAPE,7000) MATH, MFH, MTH, NOSEQ
13:     NOSEQ     = NXTSEQ(NOSEQ)
14: c
15: 7000 format(66X,I4,I2,I3,I5)
16: c
17:     return
18: end
```

src/artcore/normx.f

```
1:      subroutine NORMX( X,      XNORM, KSIGN, KXPLUS )
2: C=====
3: C      convert floating point number to normal form for output.
4: C      output will be in the form x.xxxxx+/-nn, which gives
5: C      an additional digit of accuracy when compared to e11.4
6: C      fortran output. in addition the output will be computer
7: C      independent (i.e. same on ibm, cdc, univac etc.)
8: C
9: C=====
10: C***** character *****
11:      character*4 KSIGN
12: C
13: C-----define two possible signs for exponent.
14:      data ZERO /0.0E+00/
15:      data ONE  /1.0E+00/
16:      data TEN  /1.0E+01/
17:      data XLOW /1.0E+00/
18:      data XHIGH /9.999995E+00/
19: C
20: C-----compute signed exponent from absolute value.
21:      XABS  = ABS(X)
22: C
23:      if ( XABS.gt.0.0 ) then
24: C-----define exponent to normalize mantissa.
25:          KXREAL = LOG10(XABS)
26:          if ( XABS.lt.ONE ) KXREAL = KXREAL - 1
27:          SHIFT  = TEN**KXREAL
28:          XNORM  = XABS/SHIFT
29: C-----adjust mantissa for rounding to exact powers of 10 during output.
30: C980828 if ( XNORM.gt.XLOW ) then
31:         if ( XNORM.lt.XLOW ) then
32:             XNORM = ONE
33:         else if ( XNORM.ge.XHIGH ) then
34:             KXREAL = KXREAL + 1
35:             XNORM = ONE
36:         end if
37: C-----compute signed mantissa.
38:         if ( X.lt.ZERO ) XNORM = -XNORM
39: C-----define sign of exponent and insure exponent is positive.
40:         if ( KXREAL.lt.0.0 ) then
41:             KSIGN  = '-'
42:             KXPLUS = -KXREAL
43:         else
44:             KSIGN  = '+'
45:             KXPLUS = KXREAL
46:         end if
47:     else
48: C-----x is 0.0. define as 0.00000+ 0
49:         XNORM = ZERO
50:         KSIGN = '+'
51:         KXREAL = 0
52:     end if
53: C
54:      return
55:      end
```

src/artcore/nxtseq.f

```
1:      function NXTSEQ(NOSEQ)
2: c
3: c      define next sequence number for endf/b output. allow for
4: c      more than 100000 cards per evaluation by resetting number
5: c      to 1 every time 100000 is reached.
6: c
7:      NN      = NOSEQ + 1
8:      if ( NN.eq.100000 ) NN  = 1
9:      NXTSEQ  = NN
10:     return
11:     end
```

SAFE

src/artcore/oute.f

```

1:      subroutine OUTE( E,      FIELD )
2:      c
3:      c      format energy into hollerith form for output.
4:      c      energy must be between 1 milli-ev and 100 mev.
5:      c
6:      real*8 E
7:      character*4 FIELD(11)
8:      C
9:      character*4 ZERO, DOT
10:     real*8 HALF
11:     real*8 TEN(0:10)
12:     character*4 DIGIT(10)
13:
14:     data ZERO /'0'/
15:     data DOT /'.'/
16:     data DIGIT /'0','1','2','3','4','5','6','7','8','9'/
17:     data HALF /0.5D+00/
18:     data TEN /1.0D+00, 1.0D+01, 1.0D+02, 1.0D+03, 1.0D+04, 1.0D+05,
19:     &      1.0D+06, 1.0D+07, 1.0D+08, 1.0D+09, 1.0D+10/
20:
21: c-----first column will always be blank.
22:
23:     FIELD(1) = ' '
24:
25: c-----select full 9 digit output (e at least 1) or partial digit output.
26:
27:     if ( E.eq.TEN(0) ) then
28:         IEXP = 3
29:
30: c----- (e less than 1).
31: c-----partial digit output. initialize characters and define fixed point
32: c-----output field.
33:
34:     else if ( E.lt.TEN(0) ) then
35:         FIELD(2) = ZERO
36:         FIELD(3) = DOT
37:         FIELD(4) = ZERO
38:         FIELD(5) = ZERO
39:         IN = TEN(8)*E + HALF
40:         IEXP = 0
41:     else
42:
43: c-----full 9 digit output. perform binary search to define column
44: c-----position of decimal point.
45:         if ( E.lt.TEN(4) ) then
46:             if ( E.lt.TEN(2) ) then
47:                 if ( E.lt.TEN(1) ) then
48:                     IEXP = 3
49:                 else
50:                     IEXP = 4
51:                 end if
52:             else if ( E.eq.TEN(2) ) then
53:                 IEXP = 5
54:             else if ( E.lt.TEN(3) ) then
55:                 IEXP = 5
56:             else
57:                 IEXP = 6
58:             end if
59:         else if ( E.eq.TEN(4) ) then
60:             IEXP = 7
61:         else if ( E.lt.TEN(6) ) then
62:             if ( E.lt.TEN(5) ) then
63:                 IEXP = 7
64:             else
65:                 IEXP = 8

```

```

66:         end if
67:     else if ( E.eq.TEN(6) ) then
68:         IEXP = 9
69:     else if ( E.lt.TEN(7) ) then
70:         IEXP = 9
71:     else
72:         IEXP = 10
73:     end if
74:
75: c-----define fixed point output field and initialize output columns.
76:
77:         INDX = 11 - IEXP
78:         IN = (E*TEN(INDX)) + HALF
79:         FIELD(IEXP) = DOT
80:     end if
81:
82: c-----convert fixed point output field to hollerith.
83:
84:     II = 11
85:     do 100 I = 1, 11
86:         if ( II.ne.IEXP ) then
87:             IN2 = IN/10
88:             IN3 = (IN-10*IN2) + 1
89:             FIELD(II) = DIGIT(IN3)
90:             if ( IN2.le.0 ) go to 110
91:             IN = IN2
92:         end if
93:         II = II - 1
94:     100 continue
95:     110 return
96:     end

```

src/artcore/pointv.f

```

1:      subroutine POINTV( X,      Y,      IXY )
2:      c
3:      c      write ixy data points. format of energies will vary to allow
4:      c      maximum precision of between 6 and 9 digits accuracy.
5:      c
6:      c      cross sections will always be output ell.4 format.
7:      c
8:      c      energies will be output in either standard ell.4 format or a
9:      c      variable f format (variable from f11.8 to f11.0) to give the
10:     c      maximum number of digits of accuracy. as output by this routine
11:     c      standard form ell.4 format gives 6 digits of accuracy. the
12:     c      variable form f format will give 6 to 9 digits accuracy. as
13:     c      long as the exponent of an energy in ell.4 format is between -3
14:     c      and +8, more digits will be included if the number is output in
15:     c      variable f format. in particular a full 9 digits will be output
16:     c      for all energies between 1 ev and 100 mev. between 1 milli-ev
17:     c      and 1 ev the number of digits will vary from 6 to 8.
18:     c
19:     real*8 X(IXY)
20:     real Y(IXY)
21:     c
22:     character*4 KSIGN, FIELD
23:     c***** character *****
24:     c***** integer *****
25:     c      integer field
26:     c***** double *****
27:     real*8 ELOW, EHIGH
28:     c
29:     integer OTAPE
30:     common /UNITS/      INP,      OUTP,      ITAPE,      OTAPE,      SCF
31:     common /NORMF/      XNORM(6),      KEXP(6)
32:     common /NORMC/      KSIGN(6)
33:     common /HEADR/      DUMMY(6),      MATH,      MFH,      MTH,      NOSEQ
34:     common /FLAGS/      MINUS3
35:     common /FIELDLC/      FIELD(11,3)
36:     c
37:     c-----define lower and upper energy limits for variable f output.
38:     c
39:     data ELOW /1.0D-03/
40:     data EHIGH /1.0D+08/
41:     c
42:     c-----set up loop over cards.
43:     do 120 I = 1, IXY, 3
44:         IP2      = I + 2
45:         if ( IP2.gt.IXY ) IP2      = IXY
46:         IOUT      = IP2 - I + 1
47:     c-----select variable f or e output format.
48:         if ( X(I).le.ELOW .or. X(IP2).ge.EHIGH ) then
49:             c
50:             c      output one card in standard e format.
51:             c
52:             c-----convert data to normal form.
53:             I3      = 0
54:             do 100 II = I, IP2
55:                 I3      = I3 + 1
56:                 ES      = X(II)
57:                 call NORMX( ES, XNORM(I3), KSIGN(I3), KEXP(I3) )
58:                 I3      = I3 + 1
59:             c-----set flag if cross section is negative.
60:             if ( Y(II).lt.0.0 ) MINUS3 = 1
61:             call NORMX( Y(II), XNORM(I3), KSIGN(I3), KEXP(I3) )
62:             100      continue
63:             c-----output record.
64:             if ( IOUT.eq.2 ) then
65:                 write(OTAPE,7020) (XNORM(II),KSIGN(II),KEXP(II),II=1,4),

```

```

66:         &      MATH, MFH, MTH, NOSEQ
67:         else if ( IOUT.eq.3 ) then
68:             write(OTAPE,7040) (XNORM(II),KSIGN(II),KEXP(II),II=1,6),
69:             &      MATH, MFH, MTH, NOSEQ
70:         else
71:             write(OTAPE,7000) (XNORM(II),KSIGN(II),KEXP(II),II=1,2),
72:             &      MATH, MFH, MTH, NOSEQ
73:         end if
74:     else
75:     c
76:     c      output one card with energy in f format and cross section in e
77:     c      format.
78:     c
79:     c-----convert data to normal form.
80:     I3      = 0
81:     do 110 II = I, IP2
82:         I3      = I3 + 1
83:         call OUTE( X(II), FIELD(1,I3) )
84:     c-----set flag if cross section is negative.
85:     if ( Y(II).lt.0.0 ) MINUS3 = 1
86:     call NORMX( Y(II), XNORM(I3), KSIGN(I3), KEXP(I3) )
87:     110      continue
88:     c-----output record.
89:     if ( IOUT.eq.2 ) then
90:         write(OTAPE,7080)
91:         &      ((FIELD(J,M),J=1,11),XNORM(M),KSIGN(M),KEXP(M),M=
92:         &      1,2), MATH, MFH, MTH, NOSEQ
93:     else if ( IOUT.eq.3 ) then
94:         write(OTAPE,7100)
95:         &      ((FIELD(J,M),J=1,11),XNORM(M),KSIGN(M),KEXP(M),M=
96:         &      1,3), MATH, MFH, MTH, NOSEQ
97:     else
98:         write(OTAPE,7060) (FIELD(J,1),J=1,11), XNORM(1),
99:         &      KSIGN(1), KEXP(1), MATH, MFH, MTH, NOSEQ
100:     end if
101:     end if
102:     NOSEQ      = NXTSEQ( NOSEQ )
103:     120      continue
104:     return
105: 7000 format(2(F8.5,A1,I2),44X,I4,I2,I3,I5)
106: 7020 format(4(F8.5,A1,I2),22X,I4,I2,I3,I5)
107: 7040 format(6(F8.5,A1,I2),I4,I2,I3,I5)
108: 7060 format(11A1,F8.5,A1,I2,44X,I4,I2,I3,I5)
109: 7080 format(2(11A1,F8.5,A1,I2),22X,I4,I2,I3,I5)
110: 7100 format(3(11A1,F8.5,A1,I2),I4,I2,I3,I5)
111:     end

```

src/artcore/prbtab.f

```

1:      subroutine PRBTAB( NURT, NP, E, SIGEAV,SIGFAV,SIGTAV,PROB,
2:      & SIGE, SIGF, SIGT, Q, IUPOS, WORKP,
3:      & WORKQ, NWORD, JNTUNR,LSSF, IDBG, LENG,
4:      & NPRT, ADATA, IDATA )
5: C=====
6: C *****
7: C * unresolved probability data output *
8: C *****
9: C=====
10:     real*8 WORKP(NP), WORKQ(NP)
11:     real E(NURT)
12:     real SIGEAV(NURT), SIGFAV(NURT), SIGTAV(NURT)
13:     real PROB(NP,NURT)
14:     real SIGT(NP,NURT), SIGE(NP,NURT), SIGF(NP,NURT)
15:     real Q(NP,NURT)
16: C
17:     integer IUPOS(2,NP,NURT)
18: C
19:     real ADATA(LENG)
20:     integer IDATA(LENG)
21: C
22: C
23: C
24:     LENG1 = NP*NURT
25:     call CLEA( Q, LENG1, 0.0 )
26:     call ICLEA( IUPOS, LENG1*2, 0 )
27: C
28: C
29: C-----calculates bmc sampling data
30: C
31:     do 130 J = 1, NURT
32:         call DCLEA( WORKP, NP, 0.0d0 )
33:         call DCLEA( WORKQ, NP, 0.0d0 )
34:         do 100 I = 1, NP
35:             WORKP(I) = DBLE(PROB(I,J))
36:         continue
37: Cm call bmcml( workp, q(1,j) , iupos(1,1,j) , np )
38:         call BMCMD( WORKP, WORKQ, IUPOS(1,1,J), NP )
39:         do 110 I = 1, NP
40:             Q(I,J) = WORKQ(I)
41:         continue
42: C-----check write
43:         if ( IDBG.gt.1 ) then
44:             if ( IDBG.gt.2.or.J.le.1.or.J.ge.NURT ) then
45: C
46:                 write(NPRT,7000) J, E(J), JNTUNR
47:                 write(NPRT,7020)
48:                 do 120 I = 1, NP
49:                     write(NPRT,7040) I, SIGT(I,J), PROB(I,J), Q(I,J),
50:                     & (IUPOS(K,I,J),K=1,2)
51:                 continue
52:                 write(NPRT,7020)
53:             end if
54:         end if
55:     continue
56: C
57:     KK = NURT + 1
58: C
59: C *** old-type output
60: C
61:     if ( LSSF.eq.0 ) then
62: C
63: C write(nwrk) ( e(i) ,i=1,nurt ) , ( jntunr , i=1,nurt ) ,
64: C + ( ( q(j,i) ,j=1,np ) ,
65: C + ( iupos(1,j,i) ,iupos(2,j,i) ,j=1,np ) ,

```

```

66: C + ( sigt(j,i),sige(j,i),sigf(j,i),j=1,np ) , i=1,nurt )
67: C
68:     ISW = 0
69:     do 140 I = 1, NURT
70:         ISW = ISW + 1
71:         ADATA(ISW) = E(KK-I)
72:     continue
73:     do 150 I = 1, NURT
74:         ISW = ISW + 1
75:         IDATA(ISW) = JNTUNR
76:     continue
77:     do 190 I = 1, NURT
78:         do 160 J = 1, NP
79:             ISW = ISW + 1
80:             ADATA(ISW) = Q(J,KK-I)
81:         continue
82:         do 170 J = 1, NP
83:             ISW = ISW + 1
84:             IDATA(ISW) = IUPOS(1,J,KK-I)
85:             ISW = ISW + 1
86:             IDATA(ISW) = IUPOS(2,J,KK-I)
87:         continue
88:         do 180 J = 1, NP
89:             ISW = ISW + 1
90:             ADATA(ISW) = SIGT(J,KK-I)
91:             ISW = ISW + 1
92:             ADATA(ISW) = SIGE(J,KK-I)
93:             ISW = ISW + 1
94:             ADATA(ISW) = SIGF(J,KK-I)
95:         continue
96:     continue
97: C
98:     NWORD = NURT*(2+NP*6)
99: end if
100: C
101: C *** new-type output
102: C
103:     if ( LSSF.eq.1 ) then
104: C
105: Cm write(nwrk) ( e(i) ,i=1,nurt ) , ( jntunr , i=1,nurt ) ,
106: Cm + ( ( q(j,i) ,j=1,np ) ,
107: Cm + ( iupos(1,j,i) ,iupos(2,j,i) ,j=1,np ) ,
108: Cm + ( sigt(j,i),sige(j,i),sigf(j,i),j=1,np ) ,
109: Cm + sigtav(i),sigeav(i),sigfav(i) , i=1,nurt )
110: C
111:     ISW = 0
112:     do 200 I = 1, NURT
113:         ISW = ISW + 1
114:         ADATA(ISW) = E(KK-I)
115:     continue
116:     do 210 I = 1, NURT
117:         ISW = ISW + 1
118:         IDATA(ISW) = JNTUNR
119:     continue
120:     do 250 I = 1, NURT
121:         do 220 J = 1, NP
122:             ISW = ISW + 1
123:             ADATA(ISW) = Q(J,KK-I)
124:         continue
125:         do 230 J = 1, NP
126:             ISW = ISW + 1
127:             IDATA(ISW) = IUPOS(1,J,KK-I)
128:             ISW = ISW + 1
129:             IDATA(ISW) = IUPOS(2,J,KK-I)
130:         continue

```

src/artcore/prbtab.f

```
131:          do 240 J = 1, NP
132:             ISW      = ISW + 1
133:             ADATA(ISW) = SIGT(J,KK-I)
134:             ISW      = ISW + 1
135:             ADATA(ISW) = SIGE(J,KK-I)
136:             ISW      = ISW + 1
137:             ADATA(ISW) = SIGF(J,KK-I)
138: 240      continue
139:             ISW      = ISW + 1
140:             ADATA(ISW) = SIGTAV(KK-I)
141:             ISW      = ISW + 1
142:             ADATA(ISW) = SIGEAV(KK-I)
143:             ISW      = ISW + 1
144:             ADATA(ISW) = SIGFAV(KK-I)
145: 250      continue
146: C
147:          NWORD = NURT*(5+NP*6)
148:          end if
149: C
150: C *** end of process
151: C
152:          return
153: C
154: 7000 format(/
155:      & /1X,10X,'*****' //
156:      & 1X,10X,'* unresolved resonance probability table *' //
157:      & 1X,10X,'* ( bmc sampling ) *' //
158:      & 1X,10X,'*****' //
159:      & /1X,5X,' lop = ',I3,5X,'incident energy= ',1PE12.5,'ev',5X,
160:      & ' intunr = ',I3/
161:      & /1X,5X,'no sig-t(barn) probability q-answer  ians1 ians2')
162: 7020 format(1X,4X,5('-----'),'--')
163: 7040 format(1X,4X,I3,1P,3E12.5,2I6)
164: C
165:          end
```

src/artcore/readin.f

```
1:      subroutine READIN
2: c
3: c      read and check all input parameters.
4: c
5: c***** character *****
6:      character*4 KSIGN
7: c***** character *****
8:      integer OUTP
9:      common /UNITS/  INP,      OUTP,      ITAPE,  OTAPE,  SCR
10:     common /MATZA/  MODGET,  NMATZA,  MATMIN(101),  MATMAX(101)
11:     common /OKERRT/  ERXC3T,  KERR3T,  MAXERT,  ENER3T(21),  ER3T(21)
12:     common /OKERRC/  ERXC3C,  ERXC30,  KERR3C,  MAXERC,  ENER3C(21),
13:     &      ER3C(21)
14:     common /NORMF/  XNORM(6),  KEXP(6)
15:     common /NORMC/  KSIGN(6)
16:     common /HOTS/   ALPHA,  HOTSF,  TEMPK,  TEMPEF,  N2TAPI,  N2TAPO
17:     common /SLIM/   ISTART,  NOTHIN,  ITHIN1,  ITHIN2,  ITHIN3,  MTEND
18:     common /IWATCH/ MONITR
19: c
20:      include 'INC/_MAINIO'
21: c
22: c      read selection mode (mat or za), 6/9 digit output selector (no
23: c      longer used...all energies output to 9 digit accuracy), and
24: c      kelvin temperature.
25: c
26: c
27:      100 NMATZA = NMATZA - 1
28:      ENER3T(1) = 0.0
29:      ER3T(1) = 0.0010
30: c
31:      110 ENER3C(1) = ENER3T(1)
32:      ER3C(1) = ER3T(1)
33:      if ( ER3C(1).le.0.0 ) ER3C(1) = 0.001
34:      PERCNT = 100.0*ER3T(1)
35:      PERCNC = 100.0*ER3C(1)
36:      ERRMAX = ER3T(1)
37:      call NORMX( ENER3T(1), XNORM(1), KSIGN(1), KEXP(1) )
38:      call NORMX( ER3T(1), XNORM(2), KSIGN(2), KEXP(2) )
39:      call NORMX( ENER3C(1), XNORM(3), KSIGN(3), KEXP(3) )
40:      call NORMX( ER3C(1), XNORM(4), KSIGN(4), KEXP(4) )
41: c
42: cdel write(outp,6070) (xnorm(i),ksign(i),kexp(i),i=3,4),percnc,
43: cdel 1 (xnorm(i),ksign(i),kexp(i),i=1,2),percnt
44: c
45: c
46:      MAXERT = 1
47:      MAXERC = MAXERT
48:      KERR3T = 0
49:      if ( MAXERT.gt.1 ) KERR3T = 1
50:      KERR3C = KERR3T
51:      ERXC3T = ER3T(1)
52:      ERXC3C = ER3C(1)
53:      ERXC30 = 0.1*ERXC3C
54: c----define whether or not broadened data will be thinned.
55:      NOTHIN = 0
56:      if ( ERRMAX.le.0.0 ) NOTHIN = 1
57: c
58:      if( NSYSO2.gt.0 ) then
59:          write(NSYSO2,*) ' ** nothin is ', NOTHIN
60:      end if
61: c
62:      return
63: c
64: c6070 format(1x,88(1h-)/19h accuracy criteria/1x,88(1h-)/14x,
65: c      1 11hcalculation,28x,8hthinning/6x,6henergy,3x,8haccuracy,3x,
```

```
66: c      2 8hper-cent,10x,6henergy,3x,8haccuracy,3x,8hper-cent/1x,88(1h-)/
67: c      3 f9.5,a1,i2,f8.5,a1,i2,f11.3,5x,f8.5,a1,i2,f8.5,a1,i2,f11.3)
68: c
69:      end
```

src/artcore/sctolc.f

```
1: C*****
2: C*   Change small character to large character
3: C*
4: C*   ICH : character string (arbitrary length)
5: C*   M   : change case up to M characters
6: C*****
7:   subroutine SCTOLC( ICH, M )
8: C
9:   character*(*) ICH
10: C
11:   character SMALL*26, LARGE*26
12: C
13:   SMALL = 'abcdefghijklmnopqrstuvwxyz'
14:   LARGE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
15: C
16:   do 100 I = 1, MIN(M, LEN(ICH))
17:     K = INDEX(SMALL, ICH(I:I))
18:     if ( K.gt.0 ) ICH(I:I) = LARGE(K:K)
19: 100 continue
20:   return
21:   end
```



src/artcore/sendo.f

```
1:      subroutine SENDO
2: c
3: c      output send, fend, mend or tend card in standard format.
4: c
5:      integer OTAPE
6:      common /UNITS/  INP,      OUTP,      ITAPE,  OTAPE,  SCR
7:      common /HEADR/  C1H,      C2H,      L1H,    L2H,    N1H,    N2H,
8:      &      MATH,      MFH,      MTH,      NOSEQ
9:      if ( OTAPE.le.0 ) return
10:     write(OTAPE,7000) MATH, MFH, NOSEQ
11:     NOSEQ = NXTSEQ(NOSEQ)
12:     return
13: 7000 format(66X,I4,I2,'  0',I5)
14:     end
```

src/artcore/setcon.f

```

1:      subroutine SETCON
2:      c
3:      c
4:      c
5:      c***** computer dependent coding *****
6:      real*8 DEMAXI
7:      c***** double *****
8:      real*8 DEMAX
9:      c***** double *****
10:     c***** character *****
11:     character*4 KSIGN, CARD
12:     c***** character *****
13:     c***** integer *****
14:     c      integer card
15:     c***** integer *****
16:     integer OTAPE, OUTP, SCR, COLD1, COLD2, COLD1P, COLD2P, HOT1,
17:     &      HOT2, HOT3, HOT3M1, UNRES1, UNRES2, UREVAL, UREACT
18:     c
19:     include 'INC/_MAINIO'
20:     c
21:     common /HEADR/ C1H, C2H, L1H, L2H, N1H, N2H,
22:     &      MATH, MFH, MTH, NOSEQ
23:     common /COPI/ MFIELD(3)
24:     common /COPC/ CARD(17)
25:     common /UNITS/ INP, OUTP, ITAPE, OTAPE, SCR
26:     common /HOTS/ ALPHA, HOTS, TEMPK, TEMPEF, N2TAPI, N2TAPO
27:     common /MATTOT/ MATIN, MATOUT
28:     common /NORMF/ XNORM(6), KEXP(6)
29:     common /NORMC/ KSIGN(6)
30:     common /EXTEND/ MESS, DTMAX
31:     common /THRESH/ ETHRES, EMIN
32:     common /INDEX/ COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2,
33:     &      HOT3, HOT3M1, N2IN, N2TOT, N2SCR
34:     c
35:     common /RESOLV/ UREVAL, UREACT, UNRES1, UNRES2, EULOW, EUHIGH,
36:     &      IL, IH
37:     common /MAXIE/ DEMAX
38:     c
39:     cccc real*8 OVPI, OVPI2
40:     cccc common /CONTAC/ OVPI, OVPI2, ATOP
41:     c
42:     c
43:     c-----define pi to double precision accuracy.
44:     c
45:     ccccc data PI /3.14159265358979D+00/
46:     c
47:     c-----define maximum allowable energy spacing (this number is selected
48:     c-----to approximate the 1/v tail to within 0.5 per-cent without
49:     c-----further iteration).
50:     c
51:     data DEMAXI /1.25961D+00/
52:     c
53:     c***** civic *****
54:     c      call dropfile(0)
55:     c***** civic *****
56:     c
57:     c      define all i/o units and optionally define file names and/or
58:     c      rewind units.
59:     c
60:     c-----define all i/o unit numbers.
61:     c      INP      = 5
62:     c      OUTP     = 6
63:     c      INP      = NSYSI
64:     c      OUTP     = NSYSO
65:     c      ITAPE    = 10

```

```

66:     c      OTAPE    = 11
67:     c      SCR      = 12
68:     c      MATNOW   = 0
69:     c
70:     c-----define where to start high energy approximate treatment (note,
71:     c-----hotsy corresponds to sqrt(a*e/kt). therefore setting hotsy equal
72:     c-----to 1,000 corresponds to starting the high energy approximation at
73:     c-----a*e/kt = 1,000,000, e.g., for u-238 doppler broadened to room
74:     c-----temperature, a=238, kt=0.0253, a/kt is approximately 10,000 and
75:     c-----the approximation starts at about 100 ev...saving a great deal of
76:     c-----time with essentially no loss of accuracy in the resolved
77:     c-----resonance region above 100 ev).
78:     c
79:     c      HOTS      = 1000.0
80:     c
81:     c-----define starting location for cold data points (cold data will
82:     c-----always start at first location in tables).
83:     c
84:     c      COLD1     = 1
85:     c
86:     c-----define page size (by changing dimension, common statements
87:     c-----define minimum energy of interest (in ev).
88:     c
89:     c      EMIN      = 1.0E-05
90:     c
91:     c-----define minimum energy of threshold (in ev).
92:     c
93:     c      ETHRES    = 100.0
94:     c
95:     c-----define required constants.
96:     c
97:     c ... changed to parameter constants in (INC/_CONST1)
98:     c      ATOP      = 4.0
99:     c      OVPI      = 1.0/SQRT(PI)
100:    c      OVPI2     = 2.0*OVPI
101:    c
102:    c-----initialize maximum allowable temperature step (to avoid
103:    c-----cross section extension).
104:    c
105:    c      DTMAX      = 0.0
106:    c
107:    c-----initialize total number of file3 points read and written.
108:    c
109:    c      N2TAPI     = 0
110:    c      N2TAPO     = 0
111:    c
112:    c-----define maximum allowable energy spacing.
113:    c
114:    c      DEMAX      = DEMAXI
115:    c      MATIN      = 0
116:    c      MATOUT     = 0
117:    c      UREVAL     = 0
118:    c
119:    c      NOSEQ      = 0
120:    c
121:    c      return
122:    c      end

```


src/artcore/sroadh.f

```

1:      subroutine SROADH( Y,      SIGY, XCY,  HEATM1,HEATM2,IWRKMX,
2:      &                  NPMAX, XCCOLD,DCOLD, YCOLD )
3: c
4: c      this routine is scalar version of subroutine broadh
5: c
6: c      high energy doppler broadening routine. this routine will
7: c      be used to doppler broaden all cross sections at energies
8: c      where ae/kt is greater than 16. any point with a lower
9: c      energy will already have been doppler broadened by
10: c      routine broadl.
11: c
12: c      for ae/kt less than or equal to 16 both exponentials in the
13: c      doppler broadening kernel must be considered. for higher energies
14: c      the second exponential may be ignored, which simplifies the
15: c      doppler broadening.
16: c
17: c      the routine has been designed with no subroutine calls
18: c      in order to minimize running time. the arithmetic
19: c      statement functions ration(a) and erfc(r,experf) will
20: c      be compiled as in line coding by virtually any fortran
21: c      compiler, and as such do not represent function calls.
22: c
23: caddb6th
24:      save IMAX
25: cend
26:      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HEATME, LOOPH,
27:      &      HIHEAT, HEATM1, HEATM2, HOT3, HOT3M1
28: c**** double *****
29:      real*8 A1, A2, A3, A4, A5, Y, A, B, R, BMY, BPY, YMA,
30:      &      YPA, YY, AALP5, BB1P5, YYA, YYB, YYC, YYD, DUMMY1, DUMMY2,
31:      &      F2B, F4B, F2A, F4A, EXPERF, RATION, ERFC,
32:      &      EXPBM, EXPAM, ERFCBM, ERFCAM, ZD, YMAA, YMBB,
33:      &      ZERO, HALF, QUART3, ONE, ONEP5, TWO, TWOP5, CONA
34: c**** double *****
35:      common /INDEX/  COLD1,  COLD2,  COLD1P, COLD2P, HOT1,  HOT2,
36:      &      HOT3,  HOT3M1, N2IN,  N2TOT, N2SCR
37:      common /INDICE/  HEATME, LOOPH, LOHEAT, HIHEAT
38: c
39: c
40: c      real*8 OVPI, OVPI2
41: c      common /CONTAC/  OVPI,  OVPI2,  ATOP
42: c
43:      include 'INC/_CONST1'
44: c
45: c/#IF INLINEEXP
46:      include 'INC/_EXPPRM'
47:      real*8 XX, YY
48:      real*8 EXPINL
49: c/#ENDIF
50: c
51: c
52: c      common/comm2/xccold(6012)
53: c      common/comm3/dcold(6012)
54: c      common/comm4/ycold(6012)
55: c
56:      real XCCOLD(NPMAX)
57:      real DCOLD(NPMAX)
58:      real*8 YCOLD(NPMAX)
59: c
60: c      complementary error function definition.
61: c
62: c-----define constants for complementary error function
63: c**** double *****
64: c      data A1 /0.254829592D+00/
65: c      data A2 /-0.284496736D+00/

```

```

66: c      data A3 /1.421413741D+00/
67: c      data A4 /-1.453152027D+00/
68: c      data A5 /1.061405429D+00/
69: c      data CONA /3.275911D-01/
70: c      data ZERO /0.0D+00/
71: c      data HALF /0.5D+00/
72: c      data QUART3 /0.75D+00/
73: c      data ONE /1.0D+00/
74: c      data ONEP5 /1.5D+00/
75: c      data TWO /2.0D+00/
76: c      data TWOP5 /2.5D+00/
77: c
78:      parameter ( A1 =0.254829592D+00)
79:      parameter ( A2 =-0.284496736D+00)
80:      parameter ( A3 =1.421413741D+00)
81:      parameter ( A4 =-1.453152027D+00)
82:      parameter ( A5 =1.061405429D+00)
83:      parameter ( CONA =3.275911D-01)
84:      parameter ( ZERO =0.0D+00)
85:      parameter ( HALF =0.5D+00)
86:      parameter ( QUART3 =0.75D+00)
87:      parameter ( ONE =1.0D+00)
88:      parameter ( ONEP5 =1.5D+00)
89:      parameter ( TWO =2.0D+00)
90:      parameter ( TWOP5 =2.5D+00)
91: c**** double *****
92: c**** single *****
93: c      data a1/0.254829592e+00/
94: c      data a2/-0.284496736e+00/
95: c      data a3/1.421413741e+00/
96: c      data a4/-1.453152027e+00/
97: c      data a5/1.061405429e+00/
98: c      data cona/3.275911e-01/
99: c      data zero/0.0/
100: c      data half/0.5/
101: c      data quart3/0.75/
102: c      data one/1.0/
103: c      data onep5/1.5/
104: c      data two/2.0/
105: c      data twop5/2.5/
106: c      zexp(zd)=exp(zd)
107: c**** single *****
108: c-----define arithmetic statement function for rational argument
109: c-----of complementary error function.
110:      RATION(A) = ONE/(ONE+CONA*A)
111: c-----define arithmetic statment function for complementary
112: c-----statment function.
113:      ERFC(R,EXPERF) = (((A5*R+A4)*R+A3)*R+A2)*R+A1)*R*EXPERF
114: c
115: c/#IF INLINEEXP
116:      EXPINL(XX) = 1.0d0
117:      &      +XX*( 1.0d0
118:      &      +XX*( FEXPC2
119:      &      +XX*( FEXPC3
120:      &      +XX*( FEXPC4
121:      &      ))))
122: c      &      +XX*( FEXPC5
123: c      &      ))))
124: c/#ENDIF
125: c
126: c
127: c      set up loop to doppler broaden cross sections.
128: c
129: c-----define all required constants for point.
130:      YY = Y*Y

```

src/artcore/sroadh.f

```

131:      YYA      = YY + HALF
132:      YYB      = TWOP5*YY + QUART3
133:      YYC      = OVPI2*Y
134:      YYD      = YYC*(YY+TWO)
135: c-----initialize integrals.
136:      DUMMY1   = ZERO
137:      DUMMY2   = ZERO
138: c*
139: c*   integrate over all energy intervals above the current energy
140: c*   point.
141: c*
142: c-----no intervals above the last point.
143: cm   if(heatm2.ge.cold2p) go to 30
144:      if ( HEATM2.ge.COLD2P ) then
145: c-----y is at upper end of tabulated range. continue cross section as
146: c-----1/v to ycold=infinity.
147:      DUMMY1   = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*(Y+OVPI)
148:      go to 110
149:      end if
150: c
151: c-----initialize integrals.
152: c
153:      A        = Y
154:      SIGA     = SIGY
155:      F2A      = YYA + YYC
156:      F4A      = YYB + YYD
157: c-----set up loop over intervals above current one.
158:      IMAX     = 0
159:      do 100 LOOPH = HEATM2, HIHEAT
160:          B      = YCOLD(LOOPH+1)
161:          BMY    = B - Y
162: c-----only extend range of integration up to atop units above y.
163: cm   if(bmy.le.atop) go to 10
164:      if ( BMY.gt.ATOP ) then
165:          BMY    = ATOP
166:          B      = Y + ATOP
167:      end if
168: c-----define constants for this point.
169:      BB1P5    = B*B + ONEP5
170:      BPY      = B + Y
171:      R        = RATION(BMY)
172: C/#IF INLINEEXP
173:      XX = BMY*BMY
174:      if ( XX.lt. dble(NEXPMX) ) then
175:          M = int(xx*nexpdv)
176:          YYY = XX - dble(M)* (1.0D0/NEXPDV)
177:          EXPBM = EXPINL(YYY)*EXPNN(M)
178:          EXPBM = 1.0D0/EXPBM
179:          ERFCBM = ERFC(R,EXPBM)
180:      else
181:          EXPBM = EXP(-XX)
182:          ERFCBM = ERFC(R,EXPBM)
183:      end if
184: C/#ELSE
185: *      EXPBM = EXP(-BMY*BMY)
186: *      ERFCBM = ERFC(R,EXPBM)
187: C/#ENDIF
188:      F2B      = YYA*ERFCBM + OVPI*BPY*EXPBM
189:      YYMAA    = (Y-A)*(Y+A)
190:      F4B      = (YYA*YYMAA+YYB)*ERFCBM + OVPI*(BPY*(BB1P5
191:      &          +YYMAA)+Y)*EXPBM
192: c-----add contribution from current interval.
193:      DUMMY1   = DUMMY1 + SIGA*(F2A-F2B)
194:      DUMMY2   = DUMMY2 + DCOLD(LOOPH)*(F4A-F4B)
195: c-----test for end of range of integration.

```

```

196:      if ( BMY.ge.ATOP ) go to 110
197: c-----save values from last intergal.
198:      A        = B
199:      SIGA     = XCCOLD(LOOPH+1)
200:      F2A      = F2B
201:      YYMAA    = (Y-A)*(Y+A)
202:      F4A      = (YYA*YYMAA+YYB)*ERFCBM + OVPI*(BPY*(BB1P5
203:      &          +YYMAA)+Y)*EXPBM
204:      IMAX     = IMAX + 1
205:      100 continue
206: c-----continue cross section as 1/v to ycold=infinity.
207:      DUMMY1   = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*
208:      &          (Y*ERFCBM+OVPI*EXPBM)
209: c
210: cdel go to 40
211: c*
212: c*   integrate over all energy intervals below the current energy
213: c*   point.
214: c*
215: c-----no intervals below current first point.
216:      110 continue
217:      if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
218: c
219:      if ( HEATM1.le.COLD1P ) then
220: c-----y is at lower end of tabulated range. continue cross section as
221: c-----1/v to ycold=0.0
222:      DUMMY1   = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*(Y-OVPI)
223:      XCY      = HALF*(DUMMY1+DUMMY2) /YY
224:      return
225:      end if
226: c
227: c-----re-initialize integrals to zero distance values
228: c
229:      B        = Y
230:      SIGB     = SIGY
231:      F2B      = YYA - YYC
232:      F4B      = YYB - YYD
233: c-----set up loop over intervals below current point.
234:      LOOPH    = HEATM1
235:      IMAX     = 0
236:      do 120 LL = LOHEAT, HEATM1
237:          LOOPH = LOOPH - 1
238:          A      = YCOLD(LOOPH)
239:          YMA    = Y - A
240: c-----only extend range of integration down to atop units below y.
241: cm   if(yma.le.atop) go to 50
242:      if ( YMA.gt.ATOP ) then
243:          YMA    = ATOP
244:          A      = Y - ATOP
245:      end if
246: c-----define constants for this point.
247:      AA1P5    = A*A + ONEP5
248:      YPA      = Y + A
249:      R        = RATION(YMA)
250: C/#IF INLINEEXP
251:      XX = YMA*YMA
252:      if ( XX.lt. dble(NEXPMX) ) then
253:          M = INT(XX*NEXPDV)
254:          YYY = XX - dble(M)* (1.0D0/NEXPDV)
255:          EXPAM = EXPINL(YYY)*EXPNN(M)
256:          EXPAM = 1.0D0/EXPAM
257:          ERFCAM = ERFC(R,EXPAM)
258:      else
259:          EXPAM = EXP(-XX)
260:          ERFCAM = ERFC(R,EXPAM)

```

src/artcore/sroadh.f

```
261:         end if
262: C/#ELSE
263: *      EXPAM   = EXP(-YMA*YMA)
264: *      ERFCAM  = ERFC(R,EXPAM)
265: C/#ENDIF
266:
267:         F2A    = YYA*ERFCAM - OVPI*YPA*EXPAM
268:         YYMBB   = (Y-B)*(Y+B)
269:         F4A     = (YYA*YYMBB+YYB)*ERFCAM - OVPI*(YPA*(AA1P5
270: &             +YYMBB)+Y)*EXPAM
271: c-----add contribution from current interval.
272:         DUMMY1  = DUMMY1 + SIGB*(F2B-F2A)
273:         DUMMY2  = DUMMY2 + DCOLD(LOOPH)*(F4B-F4A)
274: c-----test for end of range of integration.
275:         if ( YMA.ge.ATOP ) go to 130
276: c-----save values from last integral.
277:         B       = A
278:         SIGB    = XCCOLD(LOOPH)
279:         F2B     = F2A
280:         YYMBB   = (Y-B)*(Y+B)
281:         F4B     = (YYA*YYMBB+YYB)*ERFCAM - OVPI*(YPA*(AA1P5
282: &             +YYMBB)+Y)*EXPAM
283:         IMAX    = IMAX + 1
284: 120 continue
285: c
286: c-----continue cross section as 1/v to vold=0.0
287: c
288:         DUMMY1  = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*
289: &             (Y*ERFCAM-OVPI*EXPAM)
290: ccel go to 80
291: c
292: c-----define broadened cross section.
293: c
294: 130 XCY       = HALF*(DUMMY1+DUMMY2) /YY
295:         if ( IMAX.gt.IWRKMX ) IWRKMX  = IMAX
296: c
297:         return
298: end
```

src/artcore/sroads.f

```

1: cm      subroutine broads(y,sigy,xcy,heatm1,heatm2,iwrkmx)
2:      subroutine SROADS( Y,      SIGY,  XCY,   HEATM1,HEATM2,IWRKM2,
3:      &      NPMAX, XCCOLD,DCOLD, YCOLD )
4: c
5: c      this routine is scalar version of subroutine broads
6: c
7: c      high energy doppler broadening routine. this routine will
8: c      be used to doppler broaden all cross sections at energies
9: c      where ae/kt is greater than one million. any point with a lower
10: c      energy will already have been doppler broadened by routine
11: c      broadl or broadh.
12: c
13: c      for ae/kt less than or equal to 16 both exponentials in the
14: c      doppler broadening kernel must be considered. for higher energies
15: c      the second exponential may be ignored, which simplifies the
16: c      doppler broadening.
17: c
18: c      for ae/kt greater than 1,000,000 it is possible to assume the
19: c      term (x/y)**2 in the doppler integral is just unity (since x only
20: c      varies from y-4 to y+4, for y greater than 1000 this term is
21: c      essentially constant since it varies from (996/1000)**2 to
22: c      (1004/1000)**2.= 1.0 +/- 0.008.
23: c
24: c      the routine has been designed with no subroutine calls
25: c      in order to minimize running time. the arithmetic
26: c      statement functions ration(a) and erfc(r,experf) will
27: c      be compiled as in line coding by virtually any fortran
28: c      compiler, and as such do not represent function calls.
29: c
30: caddb6th
31:      SAVE IMAX
32: cend
33:      integer COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HEATME, LOOPS,
34:      &      HIHEAT, HEATM1, HEATM2, HOT3, HOT3M1
35: c***** double *****
36:      real*8 A1, A2, A3, A4, A5, Y, A, B, R, BMY, YMA, YIA,
37:      &      DUMMY1, DUMMY2, F2A, F2B, F4A, F4B, EXPAM, EXPBM, ERFCAM,
38:      &      ERFCBM, ZD, EXPERF, RATION, ERFC, ZERO,
39:      &      HALF, ONE, TWOP5, CONA
40: c***** double *****
41:      common /INDEX/  COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2,
42:      &      HOT3, HOT3M1, N2IN, N2TOT, N2SCR
43:      common /INDICE/ HEATME, LOOPS, LOHEAT, HIHEAT
44: c
45: c      common/comm2/xccold(6012)
46: c      common/comm3/dcold(6012)
47: c      common/comm4/ycold(6012)
48: c
49: c
50: c      real*8 OVPI, OVPI2
51: c      common /CONTAC/ OVPI, OVPI2, ATOP
52: c
53: c      include 'INC/_CONST1'
54: c
55: c/#IF INLINEEXP
56: c      include 'INC/_EXPPRM'
57: c      real*8 XX, YYY
58: c      real*8 EXPINL
59: c/#ENDIF
60: c
61: c      real XCCOLD(NPMAX)
62: c      real DCOLD(NPMAX)
63: c      real*8 YCOLD(NPMAX)
64: c
65: c      complementary error function definition.
66: c
67: c-----define constants for complementary error function
68: c***** double *****
69: c      data A1 /0.254829592D+00/
70: c      data A2 /-0.284496736D+00/
71: c      data A3 /1.421413741D+00/
72: c      data A4 /-1.453152027D+00/
73: c      data A5 /1.061405429D+00/
74: c      data CONA /3.275911D-01/
75: c      data ZERO /0.0D+00/
76: c      data HALF /0.5D+00/
77: c      data ONE /1.0D+00/
78: c      data TWOP5 /2.5D+00/
79: c
80: c      parameter( A1 =0.254829592D+00)
81: c      parameter( A2 =-0.284496736D+00)
82: c      parameter( A3 =1.421413741D+00)
83: c      parameter( A4 =-1.453152027D+00)
84: c      parameter( A5 =1.061405429D+00)
85: c      parameter( CONA =3.275911D-01)
86: c      parameter( ZERO =0.0D+00)
87: c      parameter( HALF =0.5D+00)
88: c      parameter( ONE =1.0D+00)
89: c      parameter( TWOP5 =2.5D+00)
90: c***** double *****
91: c***** single *****
92: c      data a1/0.254829592e+00/
93: c      data a2/-0.284496736e+00/
94: c      data a3/1.421413741e+00/
95: c      data a4/-1.453152027e+00/
96: c      data a5/1.061405429e+00/
97: c      data cona/3.275911e-01/
98: c      data zero/0.0/
99: c      data half/0.5/
100: c      data one/1.0/
101: c      data twop5/2.5/
102: c      zexp(zd)=exp(zd)
103: c***** single *****
104: c-----define arithmetic statement function for rational argument
105: c-----of complementary error function.
106: c      RATION(A) = ONE/(ONE+CONA*A)
107: c-----define arithmetic statment function for complementary
108: c-----statment function.
109: c      ERFC(R,EXPERF) = (((A5*R+A4)*R+A3)*R+A2)*R+A1)*R*EXPERF
110: c
111: c/#IF INLINEEXP
112: c      EXPINL(XX) = 1.0d0
113: c      &      +XX*( 1.0d0
114: c      &      +XX*( FEXPC2
115: c      &      +XX*( FEXPC3
116: c      &      +XX*( FEXPC4
117: c      &      )))
118: c      &      +XX*( FEXPC5
119: c      &      ))))
120: c/#ENDIF
121: c
122: c      set up loop to doppler broaden cross sections.
123: c
124: c-----define all required constants for point.
125: c      YYA = OVPI2*Y
126: c-----initialize integrals.
127: c      DUMMY1 = ZERO
128: c      DUMMY2 = ZERO
129: c*
130: c*      integrate over all energy intervals above the current energy

```

src/artcore/sroads.f

```

131: c*      point.
132: c*
133: c-----no intervals above the last point.
134:       if ( HEATM2.ge.COLD2P ) then
135: c-----y is at upper end of tabulated range. continue cross section as
136: c-----1/v to ycold=infinity.
137:       DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*(Y+OVPI) /(Y*Y)
138:       go to 110
139:       end if
140: c
141: c-----initialize integrals.
142: c
143:       A      = Y
144:       SIGA    = SIGY
145:       ERFCAM  = ONE
146:       F4A     = TWOP5 + YYA
147: c
148: c-----set up loop over intervals above current one.
149: c
150:       IMAX    = 0
151: c
152:       do 100 LOOPS = HEATM2, HIHEAT
153:         B      = YCOLD(LOOPS+1)
154:         BMY    = B - Y
155: c-----only extend range of integration up to atop units above y.
156:         if ( BMY.gt.ATOP ) then
157:           BMY  = ATOP
158:           B    = Y + ATOP
159:         end if
160: c-----define constants for this point.
161:         R      = RATION(BMY)
162:
163: C/#IF INLINEEXP
164:       XX = BMY*BMY
165:       if ( XX.lt. dble(NEXPMX) ) then
166:         M = int(xx*nexpdv)
167:         YYY = XX - dble(M)* (1.0D0/NEXPDV)
168:         EXPBM = EXPINL(YYY)*EXPNN(M)
169:         EXPBM = 1.0D0/EXPBM
170:         ERFCBM = ERFC(R,EXPBM)
171:       else
172:         EXPBM = EXP(-XX)
173:         ERFCBM = ERFC(R,EXPBM)
174:       end if
175: C/#ELSE
176: *       EXPBM = EXP(-BMY*BMY)
177: *       ERFCBM = ERFC(R,EXPBM)
178: C/#ENDIF
179:
180:       F2B     = OVPI*(B+Y)*EXPBM
181:       F4B     = ((Y-A)*(Y+A)+TWOP5)*ERFCBM + F2B
182: c-----add contribution from current interval.
183:       DUMMY1 = DUMMY1 + SIGA*(ERFCAM-ERFCBM)
184:       DUMMY2 = DUMMY2 + DCOLD(LOOPS)*(F4A-F4B)
185: c-----test for end of range of integration.
186:       if ( BMY.ge.ATOP ) go to 110
187: c-----save values from last intergal.
188:       A      = B
189:       SIGA    = XCCOLD(LOOPS+1)
190:       ERFCAM  = ERFCBM
191:       F4A     = ((Y-A)*(Y+A)+TWOP5)*ERFCBM + F2B
192:       IMAX    = IMAX + 1
193:       100 continue
194: c-----continue cross section as 1/v to ycold=infinity.
195:       DUMMY1 = DUMMY1 + XCCOLD(COLD2P)*YCOLD(COLD2P)*

```

```

196:       &      (Y*ERFCBM+OVPI*EXPBM) /(Y*Y)
197: c
198: cdel go to 40
199: c*
200: c*      integrate over all energy intervals below the current energy
201: c*      point.
202: c*
203: c-----no intervals below current first point.
204:       110 continue
205: c
206:       if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
207: c
208:       if ( HEATM1.le.COLD1P ) then
209: c-----y is at lower end of tabulated range. continue cross section as
210: c-----1/v to ycold=0.0
211:       DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*(Y-OVPI) /(Y*Y)
212: c-----define broadened cross section.
213:       XCY    = HALF*(DUMMY1+DUMMY2)
214:       return
215:       end if
216: c
217: c-----re-initialize integrals to zero distance values
218: c
219:       B      = Y
220:       SIGB    = SIGY
221:       ERFCBM  = ONE
222:       F4B     = TWOP5 - YYA
223: c-----set up loop over intervals below current point.
224:       LOOPS  = HEATM1
225:       IMAX   = 0
226: c
227:       do 130 LL = LOHEAT, HEATM1
228:         LOOPS = LOOPS - 1
229:         A      = YCOLD(LOOPS)
230:         YMA    = Y - A
231: c-----only extend range of integration down to atop units below y.
232:         if ( YMA.gt.ATOP ) then
233:           YMA  = ATOP
234:           A    = Y - ATOP
235:         end if
236: c-----define constants for this point.
237:         120 R = RATION(YMA)
238: C/#IF INLINEEXP
239:       XX = YMA*YMA
240:       if ( XX.lt. dble(NEXPMX) ) then
241:         M = INT(XX*NEXPDV)
242:         YYY = XX - dble(M)* (1.0D0/NEXPDV)
243:         EXPAM = EXPINL(YYY)*EXPNN(M)
244:         EXPAM = 1.0D0/EXPAM
245:         ERFCAM = ERFC(R,EXPAM)
246:       else
247:         EXPAM = EXP(-XX)
248:         ERFCAM = ERFC(R,EXPAM)
249:       end if
250: C/#ELSE
251: *       EXPAM = EXP(-YMA*YMA)
252: *       ERFCAM = ERFC(R,EXPAM)
253: C/#ENDIF
254:       F2A     = OVPI*(Y+A)*EXPAM
255:       F4A     = ((Y-B)*(Y+B)+TWOP5)*ERFCAM - F2A
256: c-----add contribution from current interval.
257:       DUMMY1 = DUMMY1 + SIGB*(ERFCBM-ERFCAM)
258:       DUMMY2 = DUMMY2 + DCOLD(LOOPS)*(F4B-F4A)
259: c-----test for end of range of integration.
260:       if ( YMA.ge.ATOP ) go to 140

```

src/artcore/sroads.f

```
261: c-----save values from last integral.
262:      B      = A
263:      SIGB   = XCCOLD(LOOPS)
264:      ERFCBM = ERFCAM
265:      F4B    = ((Y-B)*(Y+B)+TWOP5)*ERFCAM - F2A
266:      IMAX   = IMAX + 1
267:      130 continue
268: c-----continue cross section as 1/v to ycold=0.0
269:      DUMMY1 = DUMMY1 + XCCOLD(COLD1P)*YCOLD(COLD1P)*
270:      &      (Y*ERFCAM-OVPI*EXPAM) /(Y*Y)
271: c
272: c-----define broadened cross section.
273: c
274:      140 XCY = HALF*(DUMMY1+DUMMY2)
275:      if ( IMAX.gt.IWRKMX ) IWRKMX = IMAX
276: c
277:      return
278:      end
```

src/artcore/tend.f

```
1:      subroutine TEND
2: c
3:      integer OTAPE, OUTP, SCR
4: c
5:      common /HEADR/  C1H,    C2H,    L1H,    L2H,    N1H,    N2H,
6: &      MATH,    MFH,    MTH,    NOSEQ
7:      common /UNITS/  INP,    OUTP,    ITAPE,    OTAPE,    SCR
8: c
9:      NOSEQ    = 1
10:     MATH     = -1
11:     MTH      = 0
12:     MFH      = 0
13:     write(OTAPE,7000) MATH, MFH, MTH, NOSEQ
14: c
15: 7000 format(66X,I4,I2,I3,I5)
16: c
17:     return
18: end
```

src/artcore/terpl.f

```

1:      subroutine TERPl( X1,    Y1,    X2,    Y2,    X,    Y,    II,
2:      &                NSYSO )
3: c=====interpolate one point=====8=====
4: c      (x1,y1) and (x2,y2) are the end points of the line
5: c      (x,y) is the interpolated point
6: c      i is the interpolation code
7: c      note- if a negative or zero argument of a log is
8: c              detected, the interpolation code is automatically
9: c              changed from log to linear
10: c-----error stops
11: c      133 interpolation code out of range
12: c----- 134 zero or negative value cannot be interpolated by logs
13:
14:      100 XA    = X1
15:          YA    = Y1
16:          XB    = X2
17:          YB    = Y2
18:          XP    = X
19:
20:      if ( II.le.0 .or. II.gt.5 ) then
21:          write(NSYSO,7000) II
22:      7000 format(1X,'xxx(terpl) interpolation code ',I3,
23:      &          ' is out of range.')
24:          stop 888
25:      end if
26: c-----
27:      if ( II.eq.1 ) then
28:          YP    = YA
29:      else if ( II.eq.2 ) then
30:          YP    = YA + (XP-XA)*(YB-YA) / (XB-XA)
31:      else if ( II.eq.3 ) then
32:
33: c 60 if(xa)50,50,70
34: c 70 if(xb)50,50,80
35: c 80 if(xp)90,90,100
36: c 90 call error(134)
37: c 100 yp=ya+alog(xp/xa)*(yb-ya)/log(xb/xa)
38: c      go to 200
39:
40:      if ( XA.le.0.0 .or. XB.le.0.0 ) then
41:          YP    = YA + (XP-XA)*(YB-YA) / (XB-XA)
42:      else
43:          if ( XP.le.0.0 ) go to 110
44:          YP    = YA + LOG(XP/XA)*(YB-YA) / LOG(XB/XA)
45:      end if
46:
47:      else if ( II.eq.4 ) then
48:
49: c 110 if(ya)50,50,120
50: c 120 if(yb)50,50,130
51: c 130 yp=ya*exp((xp-xa)*log(yb/ya)/(xb-xa))
52:
53:      if ( YA.gt.0.0.and.YB.gt.0.0 ) then
54:          YP    = YA*EXP((XP-XA)*LOG(YB/YA)/(XB-XA))
55:      else
56:          YP    = YA + (XP-XA)*(YB-YA) / (XB-XA)
57:      end if
58:
59:      else if ( II.eq.5 ) then
60:
61: c 140 if(ya)60,60,150
62: c 150 if(yb)60,60,160
63:
64: c 160 if(xa)130,130,170
65: c 170 if(xb)130,130,180

```

```

66: c 180 if(xp)90,90,190
67: c 190 yp=ya*exp(alog(xp/xa)*alog(yb/ya)/alog(xb/xa))
68:
69:      if ( YA.le.0.0 .or. YB.le.0.0 ) then
70:          if ( XA.le.0.0 .or. XB.le.0.0 ) then
71:              YP    = YA + (XP-XA)*(YB-YA) / (XB-XA)
72:          else
73:              if ( XP.le.0.0 ) go to 110
74:              YP    = YA + LOG(XP/XA)*(YB-YA) / LOG(XB/XA)
75:          end if
76:      else
77:          if ( XA.le.0.0 .or. XB.le.0.0 ) then
78:              YP    = YA*EXP((XP-XA)*LOG(YB/YA)/(XB-XA))
79:          else
80:              if ( XP.le.0.0 ) go to 110
81:              YP    = YA*EXP(LOG(XP/XA)*LOG(YB/YA)/LOG(XB/XA))
82:          end if
83:      end if
84:  end if
85: c
86:      Y        = YP
87:      return
88: c -----
89:      110 write(NSYSO,7020) XP
90:          stop 888
91:      7020 format(1X,'xxx(TERPl) ',
92:      &          'zero or negative value cannot be interpolated by logs ',
93:      &          E12.5)
94:      end

```


src/artcore/terpo.f

```
1:      subroutine TERPO( NBT,  INT,  N1 )
2: c
3: c      write tabl interpolation law.
4: c
5:      integer NBT(N1), INT(N1)
6: c
7:      integer OTAPE
8:      common /UNITS/  INP,  OUTP,  ITAPE,  OTAPE,  SCR
9:      common /HEADR/  DUMMY(6),      MATH,  MFH,  MTH,  NOSEQ
10: C
11:      if ( OTAPE.le.0 ) return
12:      do 100 I = 1, N1, 3
13:          IP2      = I + 2
14:          if ( IP2.gt.N1 ) IP2      = N1
15:          IOUT      = IP2 - I + 1
16:          if ( IOUT.eq.2 ) then
17:              write(OTAPE,7020) (NBT(II),INT(II),II=I,IP2), MATH, MFH,
18:              &      MTH, NOSEQ
19:          else if ( IOUT.eq.3 ) then
20:              write(OTAPE,7000) (NBT(II),INT(II),II=I,IP2), MATH, MFH,
21:              &      MTH, NOSEQ
22:          else
23:              write(OTAPE,7040) NBT(I), INT(I), MATH, MFH, MTH, NOSEQ
24:          end if
25:          NOSEQ      = NXTSEQ(SEQ)
26:      100 continue
27:      return
28: 7000 format(6I11,I4,I2,I3,I5)
29: 7020 format(4I11,22X,I4,I2,I3,I5)
30: 7040 format(2I11,44X,I4,I2,I3,I5)
31:      end
```

src/artcore/therm6.f

```

1:      subroutine THERM6( NPRT, NTEMP, NTELAS, IBCMAX,
2:      &                  NPTHE, NPDBY, NEBRG, SBMT2,
3:      &                  NPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT, ADATA,
4:      &                  IDATA, ENERGY, CROSS,
5:      &                  TMPTHS, TMPEFF, INTTMP, TMPELS, INTELS,
6:      &                  EMESH3, DBY, ENGBRG, SBG,
7:      &                  EMESH7, CRSTH7, EMESH5, ETRAN7, ANGTH7, SIN,
8:      &                  CRSMT2, ANS1, ANS2, ANS3, ANS4, ANS5,
9:      &                  NBINBG, ANSSB, PIJE, ANGNEW, IANS, WORK1,
10:     &                  WORK2, WORKP, WORKQ, LENWRK, RTEMP0, NATOM)
11: C
12: C=<ARTCORE>=====
13: C Purpose :
14: C Called in : ARTPRC
15: C Calls : INTERPL, INTDBY, INTCOH, INTTH7, INTTH6, FILE5T, FIL5TE
16: C          CLEAR, TERP1
17: C=====
18: C
19: C
20:      include 'INC/_PARAM'
21: C
22:      include 'INC/_MAINIO'
23: C
24:      common /CONTRL/ MATD, ZA, ELUNR, EHUNR, CRSMIN, LEMORY,
25:     &          IUNRES, INTUNR, IDUMP, IDBG
26: C
27:      common /HEAD1/ NPTS, NTDATA, NUNR, NUNR2, NLEX, NBINA,
28:     &          NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU,
29:     &          ITERM, ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1,
30:     &          LNUD, NNUD, NNF, LSNUD, LASYM, NOTDOP,
31:     &          KDUMMY(68),
32:     &          ATW, TLAB, ETHERM, ESAB, ELOW,
33:     &          EHI, TSTAR, RDUMMY(43)
34: C
35:      common /HEAD2/ MTINFO(MTMAX), MTPAR(MTMAX), ISMT(MTMAX),
36:     &          IENDMT(MTMAX), NEUMT(MTMAX), LCTMT(MTMAX),
37:     &          NEANG(MTMAX), NKF5(MTMAX), NEF5(MTMAX),
38:     &          MTTHR(MTMAX), LSTF3(MTMAX), LSTF4(MTMAX),
39:     &          LSTF5(MTMAX), NEF5S(NKMAX, MTMAX),
40:     &          LFF5S(NKMAX, MTMAX), LSTF5S(NKMAX, MTMAX),
41:     &          QVAL(MTMAX), QVAL2(MTMAX)
42: C
43: C *** interpolate temp. dependent thermal scattering data
44: C
45:      real ENERGY(NPTS), CROSS(NPTS), ADATA(NTDATA)
46: C
47:      integer IDATA(NTDATA)
48: C
49:      real TMPTHS(NTEMP), TMPEFF(NTEMP)
50:      integer INTTMP(NTEMP)
51: C
52:      real TMPELS(NTELAS)
53:      integer INTELS(NTELAS), NBINBG(NEBRG)
54: C
55:      real EMESH3(NPTHE)
56:      real DBY(NPDBY)
57:      real ENGBRG(NEBRG)
58:      real SBG(NTELAS, NEBRG)
59: C
60:      real EMESH7(NPTHIN)
61:      real CRSTH7(NTEMP, NPTHIN)
62:      real EMESH5(NFGTH)
63:      real ETRAN7(NTEMP, NFGTH, NIGTH)
64:      real ANGTH7(NTEMP, 4, MXNGTH)
65:      real CRSMT2(NPTCUT)

```

```

66: C
67:      real ANS1(NPTHE)
68:      real ANS2(NBINA1, NPTHE)
69:      real ANS3(NPTHIN)
70:      real ANS4(NFGTH, NFGTH)
71:      real ANS5(5, MXNGTH)
72:      real PIJE(NFGTH, NFGTH)
73:      real ANGNEW(4, MXNGTH)
74:      real ANSSB(NEBRG)
75: C
76:      integer IANS(2, IBCMAX, IBCMAX)
77: C
78:      real WORK1(NFGTH)
79:      real WORK2(LENWRK)
80:      real*8 WORKP(IBCMAX), WORKQ(IBCMAX)
81: C
82: C ... local variables ...
83:      real*8 FACTXS, ENOW, XX1, XX2, YY1, YY2, ANS
84: C
85: C *** interpolates esab
86: C
87:      RTEMP = RTEMP0
88: C
89:      if ( RTEMP.lt.TMPTHS(1) ) then
90:         write(NSYSO,*) ' ** warning at sub(thermj) !!'
91:         write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
92:         &                ' is less than the lowest energy in',
93:         &                ' thermal scattering data !!'
94:         &
95:         write(NSYSO,*) ' ** so use ', TMPTHS(1), ' K data !!'
96:         RTEMP = TMPTHS(1)
97:      end if
98: C
99:      if ( RTEMP.gt.TMPTHS(NTEMP) ) then
100:         write(NSYSO,*) ' ** warning at sub(thermj) !!'
101:         write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
102:         &                ' is greater than the highest energy in',
103:         &                ' thermal scattering data !!'
104:         &
105:         write(NSYSO,*) ' ** so use ', TMPTHS(NTEMP), ' K data !!'
106:         RTEMP = TMPTHS(NTEMP)
107:      end if
108: C
109:      FACTXS = 1.0d0
110:      if ( NATOM.gt.0 ) FACTXS = 1.0d0/dble(NATOM)
111: C
112:      TSTAR = 0.0
113:      NT1 = 1
114:      do 10 NT = 2, NTEMP
115:         if ( RTEMP.gt.TMPTHS(NT-1).and.RTEMP.le.TMPTHS(NT) )
116:            & NT1 = NT - 1
117:      10 continue
118:      NT2 = NT1 + 1
119:      X1 = TMPTHS(NT1)
120:      X2 = TMPTHS(NT2)
121:      Y1 = TMPEFF(NT1)
122:      Y2 = TMPEFF(NT2)
123:      INTNOW = INTTMP(NT2)
124: C
125:      CALL TERP1(X1, Y1, X2, Y2, RTEMP, TSTAR, INTNOW, NSYSO)
126: C
127:      if( NSYSO2.gt.0 ) then
128:         write(NSYSO2,*) ' ** matd rtemp tstar : ', MATD, RTEMP, TSTAR
129:      end if
130: C

```

src/artcore/therm6.f

```

131: C *** interpolates thermal elastic cross section
132: C
133:       if ( NPTHE.gt.0.and.ITHERM.eq.-1 ) then
134:           MTNEW = -2
135: C
136:       if ( NPDBY.gt.0 ) then
137:           call INTDBY( MTNEW, NPTHE, NTELAS, INTELS, EMESH3, DBY,
138: &           WORK2, TMPELS, RTEMP, NSYSO, NSYSO2, ANS1, ANS2 ,
139: &           NPDBY, SBMT2 , LENWRK,NBINA1,IDBG )
140:       end if
141: C
142:       if ( NEBRG.gt.0 ) then
143:           call INTCOH( MTNEW, NPTHE, NTELAS, INTELS, EMESH3, ENGBRG,
144: &           SBG
145: &           WORK2, TMPELS, RTEMP, NSYSO, NSYSO2, ANS1,ANSSB
146: &           NEBRG, LENWRK, IDBG )
147:       end if
148: C
149:       MT = 93
150:       call CLEA( CROSS, NPTS, 0.0 )
151: C
152:       JST = 1
153:       IST = ISTMT(MT)
154:       IEND = IENDMT(MT) - 1
155: C
156:       do 120 I = IEND, IST, -1
157:           ENOW = ENERGY(I)
158: C
159:           if ( I.lt.IEND.and.ENOW.eq.ENERGY(I+1) ) then
160:               CROSS(I) = CROSS(I+1)
161:               go to 120
162:           end if
163: C
164:           do 100 J = JST, NPTHE - 1
165:               JSW = J
166:               if ( ENOW.ge.EMESH3(J).and.ENOW.lt.EMESH3(J+1) ) go to
167: &               110
168:           100 continue
169:               CROSS(I) = ANS1(NPTHE)*FACTXS
170:               go to 120
171: C
172:           110 JST = JSW
173:               XX1 = EMESH3(JSW)
174:               XX2 = EMESH3(JSW+1)
175:               YY1 = ANS1(JSW)*FACTXS
176:               YY2 = ANS1(JSW+1)*FACTXS
177:               DELX = XX2 - XX1
178:               if ( ABS(DE LX).gt.1.000E-30 ) then
179:                   ANS = (YY2*(ENOW-XX1)+YY1*(XX2-ENOW)) / (XX2-XX1)
180:               else
181:                   if ( NSYSO2.gt.0 ) then
182:                       write(NSYSO2,*) ' ** mt i xx1 xx2 yy1 yy2 : ',MT,I,
183: &                       XX1,XX2, YY1, YY2
184:                   end if
185:                   ANS = (YY1+YY2)*0.5d0
186:               end if
187: C
188:               if ( ANS.lt.1.00E-30 ) ANS = 0.0d0
189:               CROSS(I) = ANS
190:           120 continue
191: C
192:       MST = LSTF3(MT) - IST
193:       MST001 = LSTF3(1) - 1 + 1 - ISTMT(1)
194:       MST002 = LSTF3(2) - 1 + 1 - ISTMT(2)
195: C

```

```

196: C ****modify total x-section *****
197: C ****modify elastic x-section *****
198: C ****modify thermal elastic x-section *****
199: C
200:       do 130 I = IST, IEND
201:           ADATA(MST001+I) = ADATA(MST001+I) + CROSS(I)
202:           ADATA(MST002+I) = CROSS(I)
203:           ADATA(MST+I) = CROSS(I)
204:       130 continue
205: C
206:       end if
207: C
208: C *** interpolates thermal elastic angular data
209: C
210:       if ( NPTHE.gt.0.and.ITHERM.eq.-1 ) then
211: C
212:           MT = 93
213: C
214:           if ( NPDBY.gt.0 ) then
215: C
216: C ... modify thermal incoherent elastic angular data
217: C
218:               LST0 = LSTF4(MT)
219:               ISW = LST0 + 2*NPTHE - 1
220:               do 150 LOP = 1, NPTHE
221:                   do 140 J = 1, NBINA1
222:                       ISW = ISW + 1
223:                       ADATA(ISW) = ANS2(J,LOP)
224:                   140 continue
225:               150 continue
226: C
227:               if ( NSYSO2.gt.0 ) then
228:                   write(NSYSO2,*) ' ** isw lstf4(94) : ', ISW, LSTF4(94)
229:               end if
230:           end if
231: C
232:           if( NEBRG.gt.0 ) then
233: C
234: C ... modify thermal coherent elastic energy distribution data
235: C
236:               LST0 = LSTF5(MT) + 13
237:               IMAX = NEBRG
238:               IANSWB = LST0 + 2 + IMAX
239:               ISW = LST0 + 2 + 3*IMAX
240: C
241:               if ( NSYSO2.gt.0 ) then
242:                   write(NSYSO2,*) ' ** LST0 LF NEENG : ',
243: &                   LST0, IDATA(LST0), IDATA(LST0+1)
244:               end if
245: C
246:               LENGTH = IMAX + 3 + 3*IMAX*(IMAX+1)/2
247: C
248:               call FILSTE( ENGBRG , ANSSB , WORK2 , IANS, NBINBG , IMAX,
249: &               ADATA(ISW), IDATA(ISW), NSYSO, LENGTH,
250: &               ADATA(IANSWB), WORKKP, WORKQ, IDBG )
251: C
252:               ISW = ISW + LENGTH
253: C
254:               if ( NSYSO2.gt.0 ) then
255:                   write(NSYSO2,*) ' ** isw lstf5(94) : ', ISW, LSTF5(94)
256:               end if
257:           end if
258:       end if
259: C
260: C *** interpolates thermal inelastic data

```

src/artcore/therm6.f

```

261: C
262:   if ( NPTHIN.gt.0 ) then
263:     MTNEW = -4
264:     call INTTH7( MTNEW, NPTHIN, NTEMP, EMESH7, CRSTH7, TMPTHS,
265: &               ANS3, WORK2, RTEMP, LENWRK, NSYSO, NSYSO2, IDBG )
266: C
267:     MT = 92
268:     call CLEA( CROSS, NPTS, 0.0 )
269: C
270:     JST = 1
271:     IST = ISTMT(MT)
272:     IEND = IENDMT(MT)
273: C
274:     do 180 I = IEND, IST, -1
275:       ENOW = ENERGY(I)
276: C
277:       if ( I.lt.IEND.and.ENOW.eq.ENERGY(I+1) ) then
278:         CROSS(I) = CROSS(I+1)
279:         go to 180
280:       end if
281: C
282:       do 160 J = JST, NPTHIN - 1
283:         JSW = J
284:         if ( ENOW.ge.EMESH7(JSW).and.ENOW.lt.EMESH7(JSW+1) ) go to
285: &           170
286:       160 continue
287:         CROSS(I) = ANS3(NPTHIN)*FACTXS
288:         go to 180
289: C
290:       170 JST = JSW
291:         XX1 = EMESH7(JSW)
292:         XX2 = EMESH7(JSW+1)
293:         YY1 = ANS3(JSW)*FACTXS
294:         YY2 = ANS3(JSW+1)*FACTXS
295:         DELX = XX2 - XX1
296:         if ( ABS(DE LX).gt.1.000E-30 ) then
297:           ANS = (YY2*(ENOW-XX1)+YY1*(XX2-ENOW))/(XX2-XX1)
298:         else
299:           if ( NSYSO2.gt.0 ) then
300:             write(NSYSO2,*) ' ** mt i xx1 xx2 yy1 yy2 : ', MT, I,
301: &               XX1, XX2, YY1, YY2
302:           end if
303:           ANS = (YY1+YY2)*0.5d0
304:         end if
305: C
306:         if ( ANS.lt.1.00E-30 ) ANS = 0.0d0
307:         CROSS(I) = ANS
308:       180 continue
309: C
310:       MST = LSTF3(MT) - IST
311:       MST001 = LSTF3(1) - 1 + 1 - ISTMT(1)
312:       MST004 = LSTF3(4) - 1 + 1 - ISTMT(4)
313: C
314: C ****modify total x-section *****
315: C ****modify total inelastic x-section *****
316: C ****modify thermal inelastic x-section *****
317: C
318:       do 190 I = IST, IEND
319:         SAVEIN = ADATA(MST+I)
320:         if ( I.gt.IST ) then
321:           ADATA(MST001+I) = ADATA(MST001+I) - SAVEIN + CROSS(I)
322:         end if
323:         ADATA(MST004+I) = ADATA(MST004+I) - SAVEIN + CROSS(I)
324:         ADATA(MST+I) = CROSS(I)
325:       190 continue

```

```

326: C
327: C *** interpolates thermal inelastic angular & energy distribution data
328: C
329:       call INTTH6( MTNEW, NTEMP, NFGTH, NIGTH, MXNGTH, EMESH5,
330: &               ETRAN7, ANGTH7,SIN, WORK2, TMPTHS, RTEMP,
331: &               ANS4 , ANS5, PIJE, ANGNEW,
332: &               NSYSO, NSYSO2, WORK1, NFGTH, IDBG ,LASYM )
333: C
334: C ****modify thermal inelastic angular data *****
335: C
336:       LST0 = LSTF4(MT)
337:       ISW = LST0 + NFGTH*2 - 1
338:       do 210 LOP = 1, MXNGTH
339:         do 200 J = 1, 5
340:           ISW = ISW + 1
341:           ADATA(ISW) = ANS5(J,LOP)
342:         200 continue
343:       210 continue
344:       if ( NSYSO2.gt.0 ) then
345:         write(NSYSO2,*) ' ** isw lstf4(93) : ', ISW, LSTF4(93)
346:       end if
347: C
348: C ****modify thermal inelastic energy distribution data*****
349: C
350:       LST0 = LSTF5(MT) + 3
351:       IMAX = NFGTH
352:       ISW = LST0 + 2 + 2*IMAX
353: C
354:       LENGTH = 3*IMAX*IMAX
355: C
356:       call FILE5T( ANS4, WORK2, IANS, WORK1, IMAX, NIGTH, ADATA(ISW),
357: &               IDATA(ISW), NSYSO, EMESH5, LENGTH, WORKP, WORKQ, IDBG )
358: C
359:       ISW = ISW + 3*IMAX*IMAX
360: C
361:       if ( NSYSO2.gt.0 ) then
362:         write(NSYSO2,*) ' ** isw lstf5(93) : ', ISW, LSTF5(93)
363:       end if
364:     end if
365: C
366: C *** end of process
367: C
368:     return
369:   end

```

src/artcore/thermj.f

```

1:      subroutine THERMJ( NPRT,
2:      &                  NTEMP, NPTHE, NRETHE, NETHE, NBNTHE,
3:      &                  NPPTHIN, NIGTH, NFGTH, MXNGTH, NPTCUT, ADATA,
4:      &                  IDATA, ENERGY, CROSS, TMPTHS, EMESH3, CRSTH3,
5:      &                  NBTTH4, JNTTH4, ANGTH4, EMESH7, CRSTH7, EMESH5,
6:      &                  ETRAN7, ANGTH7, CRSMT2, TMPEFF, INTTHE, ANS1, ANS2,
7:      &                  ANS3, ANS4, ANS5, IANS, WORK1, WORK2,
8:      &                  WORKP, WORKQ, LENWRK, RTEMP0, NATOM )
9:      C
10:     C=<ARTCORE>=====
11:     C Purpose :
12:     C Called in : ARTPRC
13:     C Calls : INTRPL, INTTH3, CLEA, INTTH4, INTTH7, INTTHS, FILE5T
14:     C=====
15:     C
16:     include 'INC/_PARAM'
17:     C
18:     include 'INC/_MAINIO'
19:     C
20:     common /CONTRL/ MATD, ZA, ELUNR, EHUNR, CRSMIN, LEMORY,
21:     &          IUNRES, INTUNR, IDUMP, IDBG
22:     C
23:     common /HEAD1/ NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA,
24:     &          NBINE, NNU, NMT, IGFLAG, LFI, LNU,
25:     &          IOTHERM, ISTU, IENDU, LSUNR, LENU, NBINA1, NBINE1,
26:     &          LNUD, NNUD, NNF, LSNUD, LASYM, NOTOP,
27:     &          KDUMMY(68),
28:     &          ATW, TLAB, ETHERM, ESAB, ELOW,
29:     &          EHI, TSTAR, RDUMMY(43)
30:     C
31:     common /HEAD2/ MTINFO(MTMAX), MTPAR(MTMAX), ISTMT(MTMAX),
32:     &          IENDMT(MTMAX), NEUMT(MTMAX), LCTMT(MTMAX),
33:     &          NEANG(MTMAX), NKF5(MTMAX), NEF5(MTMAX),
34:     &          MTTHR(MTMAX), LSTF3(MTMAX), LSTF4(MTMAX),
35:     &          LSTF5(MTMAX), NEF5S(NKMAX, MTMAX),
36:     &          LFF5S(NKMAX, MTMAX), LSTF5S(NKMAX, MTMAX),
37:     &          QVAL(MTMAX), QVAL2(MTMAX)
38:     C
39:     C *** interpolate temp. dependent thermal scattering data
40:     C
41:     real ENERGY(NPTS), CROSS(NPTS), ADATA(NTDATA)
42:     integer IDATA(NTDATA)
43:     C
44:     real TMPTHS(NTEMP), TMPEFF(NTEMP)
45:     real EMESH3(NPTHE)
46:     real CRSTH3(NTEMP, NPTHE)
47:     integer NBTTH4(NRETHE), INTTHE(NTEMP)
48:     integer JNTTH4(NRETHE)
49:     real ANGTH4(NTEMP, NBNTHE, NETHE)
50:     real EMESH7(NPTHIN)
51:     real CRSTH7(NTEMP, NPTHIN)
52:     real EMESH5(NFGTH)
53:     real ETRAN7(NTEMP, NFGTH, NIGTH)
54:     real ANGTH7(NTEMP, 5, MXNGTH)
55:     real CRSMT2(NPTCUT)
56:     C
57:     real ANS1(NPTHE)
58:     real ANS2(NBNTHE, NETHE)
59:     real ANS3(NPTHIN)
60:     real ANS4(NFGTH, NFGTH)
61:     real ANS5(5, MXNGTH)
62:     C
63:     integer IANS(2, NFGTH, NFGTH)
64:     real WORK1(NFGTH)
65:     real WORK2(LENWRK)

```

```

66:     real*8 WORKP(NFGTH), WORKQ(NFGTH)
67:
68:     C ... local variables ...
69:     real*8 FACTXS, ENOW, XX1, XX2, YY1, YY2, ANS
70:     C
71:     C *** interpolates esab
72:     C
73:     RTEMP = RTEMP0
74:     if ( RTEMP.lt.TMPTHS(1) ) then
75:       write(NSYSO,*) ' ** warning at sub(thermj) !!'
76:       write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
77:       &                  ' is less than the lowest energy in',
78:       &                  ' thermal scattering data !!'
79:     &
80:     write(NSYSO,*) ' ** so use ', TMPTHS(1), ' K data !!'
81:     RTEMP = TMPTHS(1)
82:     end if
83:
84:     if ( RTEMP.gt.TMPTHS(NTEMP) ) then
85:       write(NSYSO,*) ' ** warning at sub(thermj) !!'
86:       write(NSYSO,*) ' ** requested temperature( ', RTEMP, ' K)',
87:       &                  ' is greater than the highest energy in',
88:       &                  ' thermal scattering data !!'
89:     &
90:     write(NSYSO,*) ' ** so use ', TMPTHS(NTEMP), ' K data !!'
91:     RTEMP = TMPTHS(NTEMP)
92:     end if
93:     C
94:     FACTXS = 1.0d0
95:     if ( NATOM.gt.0 ) FACTXS = 1.0d0/dble(NATOM)
96:     C
97:     TSTAR = 0.0
98:     call INTRPL( NSYSO, NTEMP, TMPTHS, TMPEFF, 1, RTEMP, TSTAR )
99:     C
100:    if( NSYSO2.gt.0 ) then
101:      write(NSYSO2,*) ' ** matd rtemp tstar : ', MATD, RTEMP, TSTAR
102:    end if
103:    C
104:    C *** interpolates thermal elastic cross section
105:    C
106:    if ( NPTHE.gt.0.and.IOTHERM.eq.-1 ) then
107:      MTNEW = -2
108:      call INTTH3( MTNEW, NPTHE, NTEMP, INTTHE, EMESH3, CRSTH3,
109:      &          WORK2, TMPTHS, RTEMP, NSYSO, NSYSO2, ANS1, LENWRK )
110:      &          WORK2, TMPTHS, RTEMP, NSYSO, NSYSO2, ANS1, LENWRK, IDBG )
111:    C
112:    MT = 93
113:    call CLEA( CROSS, NPTS, 0.0 )
114:    C
115:    JST = 1
116:    IST = ISTMT(MT)
117:    IEND = IENDMT(MT) - 1
118:    C
119:    do 120 I = IEND, IST, -1
120:      ENOW = ENERGY(I)
121:    C
122:      if ( I.lt.IEND.and.ENOW.eq.ENERGY(I+1) ) then
123:        CROSS(I) = CROSS(I+1)
124:        go to 120
125:      end if
126:    C
127:    do 100 J = JST, NPTHE - 1
128:      JSW = J
129:      if ( ENOW.ge.EMESH3(J).and.ENOW.lt.EMESH3(J+1) ) go to
130:      &          110

```

src/artcore/thermj.f

```

131: 100      continue
132:      CROSS(I) = ANS1(NPTHE)*FACTXS
133:      go to 120
134: C
135: 110      JST      = JSW
136:      XX1      = EMESH3(JSW)
137:      XX2      = EMESH3(JSW+1)
138:      YY1      = ANS1(JSW)*FACTXS
139:      YY2      = ANS1(JSW+1)*FACTXS
140:      DELX     = XX2 - XX1
141:      if ( ABS(DELX).gt.1.000E-30 ) then
142:          ANS    = (YY2*(ENOW-XX1)+YY1*(XX2-ENOW)) / (XX2-XX1)
143:      else
144:          if ( NSYSO2.gt.0 ) then
145:              write(NSYSO2,*) ' ** mt i xx1 xx2 yy1 yy2 : ',MT,I,
146:              &              XX1,XX2, YY1, YY2
147:          &          end if
148:          ANS    = (YY1+YY2)*0.5d0
149:      end if
150: C
151:      if ( ANS.lt.1.00E-30 ) ANS = 0.0d0
152:      CROSS(I) = ANS
153: 120      continue
154: C
155:      MST      = LSTF3(MT) - IST
156:      MST001   = LSTF3(1) - 1 + 1 - ISTMT(1)
157:      MST002   = LSTF3(2) - 1 + 1 - ISTMT(2)
158: C
159: C ****modify total x-section *****
160: C ****modify elastic x-section *****
161: C ****modify thermal elastic x-section *****
162: C
163:      do 130 I = IST, IEND
164:          ADATA(MST001+I) = ADATA(MST001+I) + CROSS(I)
165:          ADATA(MST002+I) = CROSS(I)
166:          ADATA(MST+I)    = CROSS(I)
167: 130      continue
168: C
169:      end if
170: C
171: C *** interpolates thermal elastic angular data
172: C
173:      if ( NPTHE.gt.0.and.ITHERM.eq.-1 ) then
174:          MTNEW = -2
175:          call INTTH4( MTNEW, NRETHER, NETHE, NBNTHE, NTEMP, NBTTH4,
176:          &          JNTH4, TMPHS, ANGTH4, ANS2, WORK2, RTEMP,
177:          &          NSYSO, NSYSO2,
178:          &          LENWRK, IDBG )
179: C
180: C *****modify thermal elastic angular data
181:      MT      = 93
182:      LST0    = LSTF4(MT)
183:      ISW     = LST0 + 2*NETHE - 1
184:      do 150 LOP = 1, NETHE
185:          ISW = ISW + 1
186:          ADATA(ISW) = -1.000
187:          do 140 J = 2, NBNTHE
188:              ISW = ISW + 1
189:              ADATA(ISW) = ANS2(J,LOP)
190: 140      continue
191:          ISW = ISW + 1
192:          ADATA(ISW) = +1.000
193: 150      continue
194:      if ( NSYSO2.gt.0 ) then
195:          write(NSYSO2,*) ' ** isw lstf4(94) :', ISW, LSTF4(94)

```

```

196:      end if
197:      end if
198: C
199: C *** interpolates thermal inelastic data
200: C
201:      if ( NPTHIN.gt.0 ) then
202:          MTNEW = -4
203:          call INTTH7( MTNEW, NPTHIN, NTEMP, EMESH7, CRSTH7, TMPHS,
204:          &          ANS3, WORK2, RTEMP, LENWRK, NSYSO, NSYSO2, IDBG )
205: C
206:          MT      = 92
207:          call CLEA( CROSS, NPTS, 0.0 )
208: C
209:          JST      = 1
210:          IST      = ISTMT(MT)
211:          IEND     = IENDMT(MT)
212: C
213:          do 180 I = IEND, IST, -1
214:              ENOW = ENERGY(I)
215: C
216:              if ( I.lt.IEND.and.ENOW.eq.ENERGY(I+1) ) then
217:                  CROSS(I) = CROSS(I+1)
218:                  go to 180
219:              end if
220: C
221:              do 160 J = JST, NPTHIN - 1
222:                  JSW = J
223:                  if ( ENOW.ge.EMESH7(J).and.ENOW.lt.EMESH7(J+1) ) go to
224:                  &          170
225:                  continue
226:                  CROSS(I) = ANS3(NPTHIN)*FACTXS
227:                  go to 180
228: C
229: 170      JST      = JSW
230:          XX1      = EMESH7(JSW)
231:          XX2      = EMESH7(JSW+1)
232:          YY1      = ANS3(JSW)*FACTXS
233:          YY2      = ANS3(JSW+1)*FACTXS
234:          DELX     = XX2 - XX1
235:          if ( ABS(DELX).gt.1.000E-30 ) then
236:              ANS    = (YY2*(ENOW-XX1)+YY1*(XX2-ENOW)) / (XX2-XX1)
237:          else
238:              if ( NSYSO2.gt.0 ) then
239:                  write(NSYSO2,*) ' ** mt i xx1 xx2 yy1 yy2 : ', MT, I,
240:                  &          XX1, XX2, YY1, YY2
241:              end if
242:              ANS    = (YY1+YY2)*0.5d0
243:          end if
244: C
245:          if ( ANS.lt.1.00E-30 ) ANS = 0.0d0
246:          CROSS(I) = ANS
247: 180      continue
248: C
249:      MST      = LSTF3(MT) - IST
250:      MST001   = LSTF3(1) - 1 + 1 - ISTMT(1)
251:      MST004   = LSTF3(4) - 1 + 1 - ISTMT(4)
252: C
253: C ****modify total x-section *****
254: C ****modify total inelastic x-section *****
255: C ****modify thermal inelastic x-section *****
256: C
257:      do 190 I = IST, IEND
258:          SAVEIN = ADATA(MST+I)
259:          if ( I.gt.IST ) then
260:              ADATA(MST001+I) = ADATA(MST001+I) - SAVEIN + CROSS(I)

```

src/artcore/thermj.f

```
261:          end if
262:          ADATA(MST004+I) = ADATA(MST004+I) - SAVEIN + CROSS(I)
263:          ADATA(MST+I)    = CROSS(I)
264: 190      continue
265: C
266: C *** intepolates thermal inelastic angular & energy distribution data
267: C
268:          call INTTHS( MTNEW, NTEMP, NFGTH, NIGTH, MXNGTH, EMESH5,
269:          &          ETRAN7, ANGTH7, IANS, WORK2, TMPTHS, RTEMP, ANS4, ANS5,
270:          &          NSYSO, NSYSO2, WORK1, NFGTH, IDBG )
271: C
272: C ***modify thermal inelastic angular data*****
273: C
274:          LST0    = LSTF4(MT)
275:          ISW     = LST0 + NFGTH*2 - 1
276:          do 210 LOP = 1, MXNGTH
277:              do 200 J = 1, 5
278:                  ISW = ISW + 1
279:                  ADATA(ISW) = ANS5(J,LOP)
280:              200      continue
281:          210      continue
282:          if ( NSYSO2.gt.0 ) then
283:              write(NSYSO2,*) ' ** isw lstf4(93) : ', ISW, LSTF4(93)
284:          end if
285: C
286: C ***modify thermal inelastic energy distribution data*****
287: C
288:          LST0    = LSTF5(MT) + 3
289:          IMAX    = NFGTH
290:          ISW     = LST0 + 2 + 2*IMAX
291: C
292:          LENGTH  = 3*IMAX*IMAX
293: C
294:          call FILE5T( ANS4, WORK2, IANS, WORK1, IMAX, NIGTH, ADATA(ISW),
295:          &          IDATA(ISW), NSYSO, EMESH5, LENGTH, WORKP, WORKQ, IDBG )
296: C
297:          ISW     = ISW + 3*IMAX*IMAX
298: C
299:          if ( NSYSO2.gt.0 ) then
300:              write(NSYSO2,*) ' ** isw lstf5(93) : ', ISW, LSTF5(93)
301:          end if
302:      end if
303: C
304: C *** end of process
305: C
306:          return
307:      end
```

src/artcore/thinit.f

```

1:      subroutine THINIT( NPMAX, EHOT, XCHOT )
2:      C
3:      C=<ARTCORE>=====
4:      C Purpose :
5:      C
6:      C      given a function represented by a table of energies ehot and
7:      C      cross sections xchot and linear-linear interpolation between
8:      C      tabulated points, this routine will remove all extraneous
9:      C      points that lie within a given accuracy of the function based
10:     C      upon interpolation from the surrounding points.
11:     C
12:     C      during one pass this routine will thin any unconverged points
13:     C      from the last broadened page plus the current broadened page.
14:     C      at the end of the current page the last converged point and all
15:     C      unconverged points will be left in the core to be thinned with
16:     C      the next page (unless the end of the table is reached in which
17:     C      case the last point is converged). if there is more than a full
18:     C      page of thinned points it will be copied to scratch and all
19:     C      remaining points will be shifted forward one page in the doppler
20:     C      broadened data pages.
21:     C
22:     C Called in : BROADN, FILE3
23:     C Calls : ERROKT
24:     C=====
25:     integer OUP, SCR, COLD1, COLD2, COLD1P, COLD2P, HOT1, HOT2, HOT3,
26:     &      HOT3M1
27:     C**** double *****
28:     Cm      double precision ehot,de,slope,dslope,slpmax,slpmin,slp1,slp2
29:     real*8 DE, SLOPE, DSLOPE, SLPMAX, SLPMIN, SLP1, SLP2
30:     real*8 DXC
31:     C**** double *****
32:     common /UNITS/  INP,      OUP,      ITAPE, OTAPE,  SCR
33:     common /INDEX/  COLD1,   COLD2,   COLD1P, COLD2P, HOT1,  HOT2,
34:     &      HOT3,      HOT3M1, N2IN,    N2TOT,  N2SCR
35:     common /PAGER/  NPAGE,   NPT2,    NPT3,    NP1P1, NP2P1
36:     common /OKERRT/ ERXC3T,  KERR3T,  MAXERT,  ENER3T(21), ER3T(21)
37:     common /SLIM/   ISTART,  NOTHIN,  ITHIN1,  ITHIN2, ITHIN3, MTEND
38:     common /WATCH/  MONITR
39:     C
40:     Cdel common/comm5/ehot(6012)
41:     Cdel common/comm6/xchot(6012)
42:     C
43:     real XCHOT(NPMAX)
44:     real*8 EHOT(NPMAX)
45:     C
46:     C ... dsign must be on static memory ...
47:     C
48:     data DSIGN /1.0/
49:     C
50:     C-----
51:     C
52:     C      if maximum allowable error is not positive no thinning can be
53:     C      performed on the table.
54:     C
55:     C      if ( NOTHIN.le.0.0 ) then
56:     C
57:     C      thinning will be performed.
58:     C
59:     C-----if no points for thinning skip to end of thinning section.
60:     C      if ( ITHIN2.le.HOT3 ) then
61:     C
62:     C      initialize sign of derivative at beginning of section.
63:     C
64:     C      if ( ISTART.gt.0 ) then
65:     C          ISTART = 0

```

```

66:     DSIGN = 1.0
67:     if ( XCHOT(1).gt.XCHOT(2) ) DSIGN = -1.0
68:     end if
69:     C
70:     C      set up loop over broadened points.
71:     C
72:     do 160 M = ITHIN2, HOT3
73:     MM1 = M - 1
74:     DXC = dble(XCHOT(M)) - dble(XCHOT(MM1))
75:     C
76:     C      keep all maxima and minima.
77:     C
78:     C-----preceding point was maximum or minimum if sign of change in
79:     C-----cross section has reversed.
80:     C      if ( DXC*DSIGN.lt.0 ) then
81:     C          DSIGN = -DSIGN
82:     C-----if energies of two points are the same treat as discontinuity.
83:     C      if ( EHOT(M).le.EHOT(MM1) ) go to 100
84:     C-----energy not the same. save maximum or minimum if it has not
85:     C-----already been saved.
86:     C      if ( MM1.gt.ITHIN3 ) go to 120
87:     C      end if
88:     C-----maximum or minimum already saved.
89:     C
90:     C      keep preceding and present points if sign of cross section has
91:     C      changed.
92:     C
93:     C modified 1999/05/19
94:     C This fortran expression is not wrong. But this product for very small
95:     C values may be 0 on some platforms.
96:     C
97:     CCCCCC      if ( XCHOT(M)*XCHOT(MM1).lt.0 ) then
98:     C          if ( (XCHOT(M).lt.0. .and. XCHOT(MM1).gt.0.) .or.
99:     C              &      (XCHOT(M).gt.0. .and. XCHOT(MM1).lt.0.) ) then
100:     C-----if preceding point has not been kept keep preceding and current
101:     C-----points, otherwise keep only current point.
102:     C          if ( M.gt.ITHIN3 ) go to 130
103:     C          go to 140
104:     C
105:     C      keep discontinuity.
106:     C
107:     C-----is energy of two points the same.
108:     C      else if ( EHOT(M).gt.EHOT(MM1) ) then
109:     C          go to 110
110:     C      end if
111:     C-----yes. check for same cross section.
112:     C      100      DXC = ABS(XCHOT(M)-XCHOT(MM1))
113:     C-----yes. check for beginning of thinning interval.
114:     C      if ( M.ne.ITHIN3 ) then
115:     C-----not beginning of interval (m-1 not saved). if cross sections are,
116:     C-----
117:     C-----
118:     C-----
119:     C-----
120:     C      else
121:     C-----beginning of interval (m-1 already saved). if cross sections are,
122:     C-----
123:     C-----
124:     C-----
125:     C-----
126:     C      end if
127:     C-----define energy interval between current point and last converged
128:     C-----point.
129:     C      110      DE = EHOT(M) - EHOT(ITHIN1)
130:     C

```


src/artcore/thinit.f

```

131: C      define slope of straight line that will pass within the allowable
132: C      error of each point. keep eliminating points until one or more
133: C      points cannot be approximated to within the allowable error. at
134: C      that point keep the last preceding point (i.e., keep the last
135: C      point that pasased the test).
136: C
137: C      SLOPE = (dble(XCHOT(M))-dble(XCHOT(ITHIN1)))/DE
138: C-----initialize maximum and minimum allowable slope at first point of
139: C-----interval.
140: C      if ( M.eq.ITHIN3 ) then
141: C      if ( KERR3T.ne.0 ) call ERROKT( EHOT(M) )
142: C      DSLOPE = dble(ERXC3T)*dble(XCHOT(M))/DE
143: C      SLPMAX = SLOPE + DSLOPE
144: C      SLPMIN = SLOPE - DSLOPE
145: C      go to 160
146: C-----after first point of interval see if slope to current point passes
147: C-----within the allowable error of all preceding points in current
148: C-----interval.
149: C      else if ( SLOPE.le.SLPMAX.and.SLOPE.ge.SLPMIN ) then
150: C-----can eliminate current point. update slope limits.
151: C      if ( KERR3T.ne.0 ) call ERROKT( EHOT(M) )
152: C      DSLOPE = dble(ERXC3T)*dble(XCHOT(M))/DE
153: C      SLP1 = SLOPE + DSLOPE
154: C      if ( SLP1.lt.SLPMAX ) SLPMAX = SLP1
155: C      SLP2 = SLOPE - DSLOPE
156: C      if ( SLP2.gt.SLPMIN ) SLPMIN = SLP2
157: C      go to 160
158: C      end if
159: C-----need to keep last preceding point (last one to pass test).
160: C      120      ITHIN1 = ITHIN1 + 1
161: C      EHOT(ITHIN1) = EHOT(MM1)
162: C      XCHOT(ITHIN1) = XCHOT(MM1)
163: C-----re-define index to beginning of next interval.
164: C      ITHIN3 = M
165: C      go to 110
166: C-----need to keep last preceding and current points.
167: C      130      ITHIN1 = ITHIN1 + 1
168: C      EHOT(ITHIN1) = EHOT(MM1)
169: C      XCHOT(ITHIN1) = XCHOT(MM1)
170: C-----need to keep current point.
171: C      140      ITHIN1 = ITHIN1 + 1
172: C      EHOT(ITHIN1) = EHOT(M)
173: C      XCHOT(ITHIN1) = XCHOT(M)
174: C-----re-define index to beginning of next interval.
175: C      150      ITHIN3 = M + 1
176: C-----end of thinning loop.
177: C      160      continue
178: C-----if last point of array was not saved save it now.
179: C      if ( ITHIN3.le.HOT3 ) then
180: C      ITHIN1 = ITHIN1 + 1
181: C      EHOT(ITHIN1) = EHOT(HOT3)
182: C      XCHOT(ITHIN1) = XCHOT(HOT3)
183: C      end if
184: C      end if
185: C-----re-define number of broadened points in core.
186: C      HOT3 = ITHIN1
187: C      HOT3M1 = HOT3 - 1
188: C
189: C-----define thinning indices for next time that this routine will be
190: C-----called.
191: C      ITHIN2 = HOT3 + 1
192: C      ITHIN3 = HOT3 + 1
193: C      else
194: C-----no thinning. set thinned point index to last broadened point.
195: C      ITHIN1 = HOT3
196: C      end if
197: C
198: C
199: C
200: C      end of reaction. if all data is core resident leave it in core.
201: C      otherwise copy all to scratch and indicate none remaining in
202: C      core (at this point there may be up to three pages of core
203: C      resident data). if scratch file is used position it to be read.
204: C
205: C-----define final number of points to output.
206: C      N2TOT = ITHIN1 + N2SCR
207: C      return
208: C
209: C      7000 format(' monitor output...points/elow/ehigh=',I8,2E12.4,' ev')
210: C      end

```

src/artcore/udate.f

```
1:      subroutine UDATE( ADATE )
2:      C
3:      C      DATE IS FACOM BUILTIN SERVICE ROUTINE TO RETURN CURRENT DATE
4:      C      ADATE(OUTPUT) : CURRENT DATE IN 8-BYTE
5:      C      EX. '95-02-09' IF WRITE ADATE IN A8 FORMAT
6:      C
7:      C=====
8:      C=====
9:      character*8 ADATE
10:     character*12 XDATE
11:     C
12:     character*3 MONTH(12)
13:     data MONTH /'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
14:     &          'Aug', 'Sep', 'Oct', 'Nov', 'Dec'/
15:     C
16:     C
17:     Cccc CALL DATE(ADATE)
18:     XDATE = ' '
19:     call HIZUKE( XDATE )
20:     C
21:     C convert DD-MMM-YY or DD-MMM-YYYY
22:     C      to YY-MM-DD      .... aha ... ;- )
23:     C
24:     I1      = INDEX(XDATE, '-')
25:     I2      = I1 + INDEX(XDATE(I1+1:), '-')
26:     do 100 LL = LEN(XDATE), 1, -1
27:     if ( XDATE(LL:LL).ne.' ' ) go to 110
28: 100 continue
29: 110 continue
30:     C
31:     C ... Year (only two digit)
32:     C
33:     ADATE(1:3) = XDATE(LL-1:LL) //'-'
34:     ADATE(4:5) = ' '
35:     C
36:     C ... Month
37:     C
38:     do 120 IM = 1, 12
39:     if ( XDATE(I1+1:I2-1).eq.MONTH(IM) ) then
40:     if ( IM.ge.10 ) then
41:     write(ADATE(4:5), '(i2)') IM
42:     else
43:     write(ADATE(4:5), '(i0', i1)') IM
44:     end if
45:     go to 130
46:     end if
47: 120 continue
48: 130 continue
49:     C
50:     C ... Day
51:     C
52:     if ( XDATE(1:1).eq.' ' ) XDATE(1:1) = '0'
53:     ADATE(6:8) = '-'//XDATE(1:2)
54:     C
55:     return
56:     end
```

src/artcore/unresr.f

```

1:      subroutine UNRESR( NPRT,  NURT,  NBNU3R,NTU3R, IFISUN,TMPU3R,
2:      &                  TMPLOG,EUNRTB,SIGEAV,SIGFAV,SIGTAV,PROBUN,
3:      &                  SIGETB,SIGFTB,SIGTTB,ANSE,  ANSF,  ANST,  Q,
4:      &                  IUPOS, WORKP, WORKQ, ADATA, IDATA, LENU3R,
5:      &                  RTEMP, LSSF,  IDBG,  MATD )
6: C=<MVPART>=====
7: C Purpose : Interpolate temperature-dependent unresolved probability
8: C          data.
9: C=====
10: C
11:      include 'INC/_MAINIO'
12: C
13:      real TMPU3R(NTU3R)
14:      real TMPLOG(NTU3R)
15:      real EUNRTB(NURT)
16:      real SIGEAV(NURT)
17:      real SIGFAV(NURT)
18:      real SIGTAV(NURT)
19:      real PROBUN(NBNU3R,NURT)
20:      real SIGETB(NTU3R,NBNU3R,NURT)
21:      real SIGFTB(NTU3R,NBNU3R,NURT)
22:      real SIGTTB(NTU3R,NBNU3R,NURT)
23: C
24:      real ANSE(NBNU3R,NURT)
25:      real ANSF(NBNU3R,NURT)
26:      real ANST(NBNU3R,NURT)
27: C
28:      real Q(NBNU3R,NURT)
29:      integer IUPOS(2,NBNU3R,NURT)
30: C
31:      real*8 WORKP(NBNU3R)
32:      real*8 WORKQ(NBNU3R)
33:      real ADATA(LENU3R)
34:      integer IDATA(LENU3R)
35: C
36: C
37: C *** interpolate unresolved probability table
38: C
39:      NPRT = NSYSO
40: C
41:      call UNRINT( NURT, NBNU3R, NTU3R, TMPU3R, TMPLOG, EUNRTB, SIGEAV,
42:      &             SIGFAV, SIGTAV, PROBUN, SIGETB, SIGFTB, SIGTTB, ANSE,
43:      &             ANSF, ANST, MATD, RTEMP, IFISUN, IDBG )
44: C      &             ANSF, ANST, MATD, RTEMP, NPRT, IFISUN )
45: C
46:      NWORD   = 0
47: C      nwrk    = 1
48: C      idbg    = 1
49:      JNTUNR   = IDATA(NURT+1)
50:      if ( IDBG.gt.1 ) then
51:          write(NPRT,*) ' ** lenu3r & jntunr : ', LENU3R, JNTUNR
52:      end if
53: C
54: C      rewind nwrk
55: C
56: C *** convert to bmc sampling data
57: C
58:      call PRBTAB( NURT, NBNU3R, EUNRTB, SIGEAV, SIGFAV, SIGTAV, PROBUN,
59:      &             ANSE, ANSF, ANST, Q, IUPOS, WORKP, WORKQ, NWORD, JNTUNR,
60:      &             LSSF, IDBG, LENU3R, NPRT, ADATA, IDATA )
61: C
62: C *** replace u.r.p.t
63: C
64:      if ( IDBG.gt.1 ) then
65:          write(NPRT,*) ' ** lenu3r nword : ', LENU3R, NWORD
66:      end if
67: C
68: C      rewind nwrk
69: C      read(nwrk) (adata(i),i=1,nword)
70: C      rewind nwrk
71: C
72: C
73: C *** end of process
74: C
75:      return
76:      end

```

src/artcore/unrint.f

```

1:      subroutine UNRINT( NURT, NP, NTEMP, TEMP, TMPLOG,EUNRTB,
2:      & SIGEAV,SIGFAV,SIGTAV,PROB, SIGE, SIGF, SIGC,
3:      & ANSE, ANSF, ANST, MATD, RTEMP,
4:      & IFISUN, IDBG )
5: c
6: c=<ARTCORE>=====
7: c=====
8: c
9:      include 'INC/_MAINIO'
10: c
11:      real TEMP(NTEMP), TMPLOG(NTEMP)
12:      real EUNRTB(NURT)
13:      real SIGEAV(NURT), SIGFAV(NURT), SIGTAV(NURT)
14: c
15:      real PROB(NP,NURT)
16:      real SIGE(NTEMP,NP,NURT)
17:      real SIGF(NTEMP,NP,NURT)
18:      real SIGC(NTEMP,NP,NURT)
19: c
20:      real ANSE(NP,NURT)
21:      real ANSF(NP,NURT)
22:      real ANST(NP,NURT)
23: c
24: c
25: c
26:      NPRT = NSYSO
27: c
28:      XREQ = RTEMP
29: cm if(xreq.lt.temp(1)) xreq = temp(1)
30:      if ( XREQ.lt.TEMP(1) ) then
31:          write(NSYSO,*) ' ** warning at sub(unrint) !!'
32:          write(NSYSO,*) ' ** requested temperature(' , XREQ,' K)',
33:      & ' is less than the lowest energy in u.r. probability data !!'
34:          write(NSYSO,*) ' ** so use ' , TEMP(1), ' K data !!'
35:          XREQ = TEMP(1)
36:      end if
37: cm if(xreq.gt.temp(ntemp)) xreq = temp(ntemp)
38:      if ( XREQ.gt.TEMP(NTEMP) ) then
39:          write(NSYSO,*) ' ** warning at sub(unrint) !!'
40:          write(NSYSO,*) ' ** requested temperature(' , XREQ,' K)',
41:      & ' is greater than the highest energy',
42:      & ' in u.r. probability data !!'
43:      &
44:          write(NSYSO,*) ' ** so use ' , TEMP(NTEMP), ' K data !!'
45:          XREQ = TEMP(NTEMP)
46:      end if
47: c
48: c+0klib
49: c XREQ = LOG(XREQ)
50:      ADDTMP = 0.0
51:      IF(TEMP(1).LT.1.0) ADDTMP = 1.0
52:      XREQ = LOG(XREQ+ADDTMP)
53: c-0klib
54: c
55: c --- start of interpolation
56: c
57:      call CLEA( ANSE, NP*NURT, 0.0 )
58:      call CLEA( ANSF, NP*NURT, 0.0 )
59:      call CLEA( ANST, NP*NURT, 0.0 )
60: c
61:      do 140 LOP = 1, NURT
62:          do 100 JOP = 1, NP
63:              ANS1 = 0.0
64:              ANS2 = 0.0
65:              ANS3 = 0.0

```

```

66:          call INTRPL( NPRT, NTEMP, TMPLOG, SIGE(1,JOP,LOP), 1, XREQ,
67:      & ANS1 )
68:          if ( IFISUN.eq.1 )
69:      & call INTRPL( NPRT, NTEMP, TMPLOG, SIGF(1,JOP,LOP), 1,
70:      & XREQ, ANS2 )
71: cmod call intrpl ( NPRT,ntemp,tmplog,sigt(1,jop,lop),1,xreq,ans3 )
72:          call INTRPL( NPRT, NTEMP, TMPLOG, SIGC(1,JOP,LOP), 1, XREQ,
73:      & ANS3 )
74:          ANSE(JOP,LOP) = ANS1
75:          ANSF(JOP,LOP) = ANS2
76:          ANST(JOP,LOP) = ANS3
77:      100 continue
78: c
79:      if ( IDBG.gt.1 ) then
80:          write(NPRT,7000) MATD, LOP, EUNRTB(LOP), RTEMP
81:          write(NPRT,7020)
82:      end if
83:      AVRGL = 0.0
84:      AVRGL2 = 0.0
85:      AVRGL3 = 0.0
86:      SUM = 0.0
87:      do 110 I = 1, NP
88:          SAVE = PROB(I,LOP)
89:          if ( SAVE.le.0.0 ) go to 110
90: c
91:          SUM = SUM + SAVE
92:          AVRGL = AVRGL + ANSE(I,LOP)*SAVE
93:          AVRGL2 = AVRGL2 + ANSF(I,LOP)*SAVE
94:          AVRGL3 = AVRGL3 + ANST(I,LOP)*SAVE
95: cmod write(nprr,751) i,save,sum,anse(i,lop),ansf(i,lop),anst(i,lop)
96:          if ( IDBG.gt.1 ) then
97:              write(NPRT,7040) I, SAVE, SUM, ANSE(I,LOP), ANSF(I,LOP),
98:      & ANSE(I,LOP) + ANSF(I,LOP) + ANST(I,LOP)
99:          end if
100:      110 continue
101:          AVRGL4 = AVRGL + AVRGL2 + AVRGL3
102: cmod write(nprr,752) avrg1,avrg2,avrg3
103:          if ( IDBG.gt.1 ) then
104:              write(NPRT,7060) AVRGL, AVRGL2, AVRGL4
105:              write(NPRT,7080) SIGEAV(LOP), SIGFAV(LOP), SIGTAV(LOP)
106:          end if
107: c----- renormalization
108:          RATIO1 = 1.000
109:          RATIO2 = 1.000
110:          RATIO3 = 1.000
111:          if ( AVRGL.gt.0.0 ) RATIO1 = SIGEAV(LOP) /AVRGL
112:          if ( AVRGL2.gt.0.0 ) RATIO2 = SIGFAV(LOP) /AVRGL2
113: cmod if(avrg3.gt.0.0) ratio3 = sigtav(lop)/avrg3
114:          if ( AVRGL3.gt.0.0 ) then
115:              SAVEC = SIGTAV(LOP) - SIGEAV(LOP) - SIGFAV(LOP)
116:              RATIO3 = SAVEC/AVRGL3
117:          end if
118:          do 120 JOP = 1, NP
119:              ANSE(JOP,LOP) = RATIO1*ANSE(JOP,LOP)
120:              ANSF(JOP,LOP) = RATIO2*ANSF(JOP,LOP)
121: cmod anst(jop,lop) = ratio3 * anst(jop,lop)
122:              SAVEC = RATIO3*ANST(JOP,LOP)
123:              ANST(JOP,LOP) = ANSE(JOP,LOP) + ANSF(JOP,LOP) + SAVEC
124:      120 continue
125: c
126:          if ( IDBG.gt.1 ) then
127:              write(NPRT,7100)
128:          end if
129:          AVRGL1 = 0.0
130:          AVRGL2 = 0.0

```

src/artcore/unrint.f

```
131:      AVR3   = 0.0
132:      SUM     = 0.0
133:      do 130 I = 1, NP
134:        SAVE   = PROB(I,LOP)
135:        if ( SAVE.le.0.0 ) go to 130
136:      c
137:        SUM     = SUM + SAVE
138:        AVR1    = AVR1 + ANSE(I,LOP)*SAVE
139:        AVR2    = AVR2 + ANSF(I,LOP)*SAVE
140:        AVR3    = AVR3 + ANST(I,LOP)*SAVE
141:      cadd
142:        SAVEC   = ANST(I,LOP) - ANSE(I,LOP) - ANSF(I,LOP)
143:      cend
144:      if ( IDBG.gt.1 ) then
145:        write(NPRT,7040) I, SAVE, SUM, ANSE(I,LOP), ANSF(I,LOP),
146:      &      ANST(I,LOP), SAVEC
147:      end if
148:      130 continue
149:      if ( IDBG.gt.1 ) then
150:        write(NPRT,7060) AVR1, AVR2, AVR3
151:        write(NPRT,7080) SIGEAV(LOP), SIGFAV(LOP), SIGTAV(LOP)
152:      end if
153:      140 continue
154:      c
155:      c-----end of process
156:      c
157:      return
158:      c
159:      c
160:      7000 format('1',//1X,20X,
161:      &      ' *****//1X,20X,
162:      &      ' *      results of u.r.p.t. after collasping  **//1X,20X,
163:      &      ' *****//1X,20X,
164:      &      ' *      material number = ',I6,'          **//1X,20X,
165:      &      ' *      r. parameter no = ',I6,'          **//1X,20X,
166:      &      ' *      neutron energy = ',1PE11.4,' ev    **//1X,20X,
167:      &      ' *      temperature   = ',E11.4,' kelvin  **//1X,20X,
168:      &      ' *****//')
169:      7020 format(/1X,10X,
170:      &      ' k      p      pcum      scat      fis      tot')
171:      7040 format(10X,I5,F10.6,F9.5,F10.3,F11.4,2F10.3,4F9.5)
172:      7060 format(22X,' averaged ',F10.3,F11.4,2F10.3)
173:      7080 format(22X,' infinite ',F10.3,F11.4,2F10.3)
174:      7100 format(/1X,10X,' ***** after renormalization ***** '//1X,10X,
175:      &      ' k      p      pcum      scat      fis      tot',7X,
176:      &      'capt')
177:      c
178:      end
```

src/artcore/wot6.f

```

1:      subroutine WOT6( X,      NCOL, LTBL,  LG,      TOP1, TOP2, TOP3,
2:      &                IPRINT )
3: c*****
4: c      Print 1,2, or 3-d arrays on I/O unit IPRINT
5: c*****
6: c      x(ltbl,ncol,lg) : 1,2 or 3-d array
7: c      top1           : comment for first  suffix of x array (a4)
8: c      top2           : comment for second suffix of x array (a4)
9: c      top3           : comment for third  suffix of x array (a4)
10: c*****
11: c
12:      real X(LTBL,NCOL,1)
13:      character*4 TOP1, TOP2, TOP3, BLK
14: c
15:      BLK      = ' '
16: c
17:      do 170 L = 1, LG
18:          I02      = 0
19:          I03      = (NCOL+9)/10
20:          if ( LG.gt.1 ) write(IPRINT,7000) TOP3, L
21:          do 160 I = 1, I03
22:              I01      = I02 + 1
23:              I02      = MIN0(I01+9,NCOL)
24:              write(IPRINT,7040) TOP1, (TOP2,J,J=I01,I02)
25:              do 150 K = 1, LTBL
26:                  if ( K.eq.1 ) go to 140
27:                  do 100 J = I01, I02
28:                      if ( X(K,J,L).ne.X(K-1,J,L) ) go to 130
29:                  100      continue
30:                      if ( KE.eq.KS ) go to 110
31:                      KE      = K
32:                      if ( K.eq.LTBL ) go to 130
33:                      go to 150
34:                  110      if ( K.eq.LTBL ) go to 140
35:                      do 120 J = I01, I02
36:                          if ( X(K,J,L).ne.X(K+1,J,L) ) go to 140
37:                  120      continue
38:                      KE      = KE + 1
39:                      go to 150
40:                  130      if ( KE.eq.KS ) go to 140
41:                      write(IPRINT,7020) TOP1, KS, TOP1, KE
42:                      if ( K.eq.LTBL.and.KE.eq.K ) go to 150
43: c 140      write(iprint,10300) K,(x(k,j,l),j=i01,i02)
44:      140      continue
45:                      if ( LG.eq.1.and.K.eq.LTBL ) then
46:                          if ( TOP3.eq.BLK ) then
47:                              write(IPRINT,7060) K, (X(K,J,L),J=I01,I02)
48:                          else
49:                              write(IPRINT,7080) TOP3, (X(K,J,L),J=I01,I02)
50:                          end if
51:                      else
52:                          write(IPRINT,7060) K, (X(K,J,L),J=I01,I02)
53:                      end if
54: c
55:                      KS      = K + 1
56:                      KE      = KS
57:                  150      continue
58:                  160      continue
59:                  170 continue
60:              return
61: c
62:      7000 format(/1X,2X,A4,I5)
63:      7020 format(8X,A4,I5,' THRU ',A4,I5,' SAME AS ABOVE')
64:      7040 format(/2X,A4,2X,A4,I3,9(5X,A4,I3))
65:      7060 format(I6,1P10E12.5)
66:      7080 format(' ',A4,1P10E12.5)
67: c
68:      end

```

src/artcore/INC/_CONST1

```
1: C
2: C  Numeric constants
3: C
4: C  * Coverted from common variables to parameter constants.
5: C
6: C  OVPI = 1/sqrt(pai) = 1/sqrt(3.141592653589793238d0)
7: C      = 5.641895835477563D-1
8: C
9: C      real  ATOP
10: C      real*8 OVPI, OVPI2
11: C      parameter ( ATOP = 4.0 )
12: C      parameter ( OVPI = 5.641895835477563D-1 )
13: C      parameter ( OVPI2 = 2*OVPI )
14: C
```

src/artcore/INC/_EXPPRM

```
1: C
2: C   === Data for inlined exponential function ===
3: C
4: C   June 1998
5: C
6: C       integer NEXPMX
7: C       real*8 DEXPMX
8: C
9: C   ... NEXPMX and DEXPMX must have the same value )
10: C
11: C       parameter ( NEXPMX = 16 )
12: C       parameter ( DEXPMX = 16.D0 )
13: C       parameter ( NEXPDV = 128 )
14: C       parameter ( NEXPMX = 32 )
15: C       parameter ( DEXPMX = 32.D0 )
16: C       parameter ( NEXPDV = 64 )
17: C
18: CCC   common /EXPINL/ EXPNN, IEXPI
19: common /EXPINL/ EXPNN
20: real*8 EXPNN(0:NEXPDV*(NEXPMX+1))
21: C
22: C   ... Tailor expansion coefficients for exp(x) ...
23: C
24: C   1/2! n= 2  0.5000000000000000D+00
25: C   1/3! n= 3  0.1666666666666666D+00
26: C   1/4! n= 4  0.4166666666666666D-01
27: C   1/5! n= 5  0.8333333333333333D-02
28: C   1/6! n= 6  0.1388888888888889D-02
29: C   1/7! n= 7  0.1984126984126984D-03
30: C   1/8! n= 8  0.2480158730158730D-04
31: C   1/9! n= 9  0.2755731922398589D-05
32: C   1/10! n= 10  0.2755731922398588D-06
33: C   1/11! n= 11  0.2505210838544172D-07
34: C   1/12! n= 12  0.2087675698786810D-08
35: C
36: C       real*8 FEXPC2,FEXPC3,FEXPC4,FEXPC5,FEXPC6,FEXPC7,FEXPC8,
37: C       & FEXPC9,FEXPC10,FEXPC11,FEXPC12
38: C
39: C       parameter( FEXPC2 = 0.5000000000000000D+00 )
40: C       parameter( FEXPC3 = 0.1666666666666666D+00 )
41: C       parameter( FEXPC4 = 0.4166666666666666D-01 )
42: C       parameter( FEXPC5 = 0.8333333333333333D-02 )
43: C       parameter( FEXPC6 = 0.1388888888888889D-02 )
44: C       parameter( FEXPC7 = 0.1984126984126984D-03 )
45: C       parameter( FEXPC8 = 0.2480158730158730D-04 )
46: C       parameter( FEXPC9 = 0.2755731922398589D-05 )
47: C       parameter( FEXPC10 = 0.2755731922398588D-06 )
48: C       parameter( FEXPC11 = 0.2505210838544172D-07 )
49: C       parameter( FEXPC12 = 0.2087675698786810D-08 )
50: C
```


src/artcore/INC/_MAINIO

1: common /MAINIO/ NSYSI,NSYSO,NSYSO2



src/artcore/INC/_PARAM

```
1:      parameter( MTMAX      = 120, NKMAX      = 90 )
2:      parameter( MXREC      = 1000 )
```



src/mvppburn/autobn.f

```

1:      subroutine AUTOBN( MODE, IBURNI,T0,      CPULIM,ARRAY, MEMORY,
2:      &                  TRGNAM,IDENT, IDWORK,MAXTRG,MAXISO,PARAMM,
3:      &                  PARATP,PARAV, NMVPPR,MAXSPR,MVPEX, MVPIDX,
4:      &                  NOUT1, NOUT2, DIRIN, DIROUT,CASEID ,MVPDBG,
5:      &                  HMINV0 )
6: C=====
7: C Input & Control routine of $BURNUP or $BRANCH mode
8: C=====
9: C
10: C --- array sizes -----
11: C
12:      include 'INC/_BURNP'
13: C
14:      character*(*) MODE
15:      real ARRAY(MEMORY)
16:      character*12 TRGNAM(MAXTRG)
17: C2003
18:      character*12 IDENT(MAXISO,MAXTRG)
19: C2003
20:      character*12 IDWORK(MAXISO,MAXTRG)
21: C2005 ... added
22:      character*12 TRGPOW(MAXTRG)
23: C
24:      integer      MCDYLD(10)
25: C2005 ... end
26: C
27: C
28: C      PARAMM(*) : symbolic parameter name to be replaced
29: C      PARATP(*) : data type of parameter
30: C      ( 'F': float, 'I': integer )
31: C      NMVPPR    : number of symbolic parameters
32: C      PARAV(1:nstep,*) : data value of symbolic parameter
33: C
34:      character*16 PARAMM(MAXSPR)
35:      character*1 PARATP(MAXSPR)
36:      real*8 PARAV(MXSTEP,MAXSPR)
37:      integer NMVPPR
38: C
39:      character*(*) DIRIN, DIROUT
40:      real CPULIM, CPUMIN
41:      character*(*) MVPEX
42:      character*(*) MVPIDX
43: C
44:      common /REAMIO/ NOUT6, NDTLS
45: C
46: C -- Common data -----
47: C
48:      include 'INC/_CBURN1'
49:      include 'INC/_CBURN2'
50:      include 'INC/_CBURN3'
51: C
52:      common /BNREST/ RSDUMY(3),      NOWSTP, IBEND, AKEFF(MXSTEP),
53:      &      ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
54:      &      EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
55:      &      INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
56:      &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
57:      &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
58:      &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
59:      &      NBATCH(MXSTEP)
60: C2005
61:      &      ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
62:      &      RMONIT(MXSTEP), EMONIT(MXSTEP)
63: Cend
64:      common /PDSPDS/ IPDSCK
65:      integer IOTSTP(MXSTEP)

```

```

66:      character*8 MEMBER
67:      character*8 DUMMY
68: C
69:      character*2 STEPNM
70:      external STEPNM
71: C
72: C ... local variables .....
73: C
74:      character*4 CASEID
75:      character*4 CASREF
76:      character*8 IDATE
77:      character*8 STIME
78:      character*6 CSTEP
79: C
80: C -----
81: C
82:      character*72 VNAME
83: C
84: C ... variables to create inputs to modules
85:      integer JPCFLG(MXSTEP)
86:      integer JSVMVP(MXSTEP)
87:      integer IBSTEP(MXSTEP)
88:      integer ISTEPS(MXSTEP)
89:      real HMINV0(MAXTRG)
90: C
91: C2003
92:      character*128 CWRK
93:      character*2 SID0, SID1, SID2
94:      character*72 TITL2
95:      character*62 ALPNM
96: C2003
97:      character*1 CH
98: C
99: C -----
100: C
101:      NIN      = NDTLS
102:      NOUT6     = NOUT1
103:      ALPNM     =
104:      &'ABCDEFGHGIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
105: C
106: C ***** Clear common area BURN1, BURN2, BURN3 & BNREST
107: C
108:      call CLRCOM
109: C
110: C *****
111: C
112:      TITLE(1)  = ' '
113:      TITLE(2)  = ' '
114:      CASEID    = ' '
115:      ISTFLG    = 0
116:      NSTFLG    = 0
117:      IHMIN0    = 0
118:      BZERO     = 1.0E-20
119: C
120:      do 100 I = 1, 20
121:          IBC(I) = 0
122:      100 continue
123:      do 110 I = 1, MXSTEP
124:          JPCFLG(I) = 0
125:          JSVMVP(I) = 1
126:          IBSTEP(I) = 0
127:          ISTEPS(I) = 0
128:      110 continue
129:      CPUMIN    = 0.0
130:      NEP       = 0

```

src/mvpburn/autobn.f

```

131:      ISTART = 1
132: C2003 .... for Branch mode
133:      ISTRT1 = 1
134:      ISTOP = 0
135: C2003 ... JHALF is set when ISTOP < 0 in input and it stops PC step
136: C2003      at ISTOP+1/2 step.
137:      JHALF = 0
138: C2003 ... END
139: C2003 STDNUC = ' '
140:      STDNUC = 0
141:      NEIS = 0
142:      NFER = 0
143:      do 120 I = 1, MXFISS
144:          IFISDN(I) = 0
145:          FISRCT(I) = 0.0
146:          IFRTDN(I) = 0
147:          FRTFCT(I) = 0.0
148:      120 continue
149: C
150:      JPERIOD = 0
151:      JPOWERL = 0
152:      NERR = 0
153:      IPRT = 0
154:      IPDSCK = 1
155: C2005 ... added
156:      MATPOW = 0
157:      do 125 I = 1, MAXTRG
158:          TRGPOW(I) = ' '
159:      125 continue
160: C
161:      JFPYLD = 0
162:      do 126 I = 1, 10
163:          MCDYLD(I) = 0
164:      126 continue
165: C
166:      NASTEP = 0
167: C
168:      IBNTYP = 0
169:      JBNTYP = 0
170:      JEXTSC = 0
171:      JDETCT = 0
172:      IDMONI = 0
173:      JMONIT = 0
174:      JRGPOW = 0
175:      JRGNAM = 0
176: C2005 ... end
177: C
178: C -----
179: C   Read data in $BURNUP or $BRANCH block
180: C -----
181: C
182:      130 continue
183: C
184:      call FREADS( VNAME, NLEN, IEND )
185:      if ( IEND.ne.0 ) then
186: C-----
187:          write(NOUT1,6888)
188:          write(NOUT1,*) ' unexpected end of input in ',
189:          &      MODE(:ICLEN2(MODE)), ' block'
190:          write(NOUT1,*) ' probably no "$END BURNUP" or ',
191:          &      "$END BRANCH" line.'
192:          stop 888
193:      end if
194: C
195: C ..... TITLES .....

```

```

196: C
197:      if ( VNAME(1:NLEN).eq.'TITLE1' ) then
198:          call CHREAD( VNAME(1:NLEN), TITLE(1), ' ', INLEN, ITERM, IEND )
199:          if ( ITERM.eq.0 ) then
200:              write(NOUT1,6888)
201:              write(NOUT1,7240) VNAME(1:NLEN)
202:              stop 888
203:          end if
204:          if ( IEND.ne.0 ) go to 330
205: C
206:      else if ( VNAME(1:NLEN).eq.'TITLE2' ) then
207:          call CHREAD( VNAME(1:NLEN), TITLE(2), ' ', INLEN, ITERM, IEND )
208:          if ( ITERM.eq.0 ) then
209:              write(NOUT1,6888)
210:              write(NOUT1,7240) VNAME(1:NLEN)
211:              stop 888
212:          end if
213:          if ( IEND.ne.0 ) go to 330
214: C2005 ... added
215: C
216: C ..... BNPTYP .....
217: C
218:      else if ( VNAME(1:NLEN).eq.'BNPTYP' ) then
219:          call I4READ( VNAME(1:NLEN), IBNTYP, NA, 1, IEND )
220:          if ( IEND.ne.0 ) go to 330
221: C
222:          JBNTYP = 1
223:          if( IBNTYP.lt.0 ) IBC(10) = 1
224:          IBNTYP = IABS(IBNTYP)
225: C
226:          if ( MODE.eq.'$BRANCH' ) then
227:              write(NOUT1,6666)
228:              write(NOUT1,7020) 'BNPTYP'
229:              call CNTERR('WARNING')
230:              JBNTYP = 0
231:              IBNTYP = 0
232:              IBC(10) = 0
233:          end if
234: C
235:          IBC(9) = IBNTYP
236: C
237: C ..... EXTSRC .....
238: C
239:      else if ( VNAME(1:NLEN).eq.'EXTSRC' ) then
240:          call R4READ( VNAME(1:NLEN), EXTSRC, NA, -MXSTEP, IEND )
241:          if ( IEND.ne.0 ) go to 330
242: C
243:          if ( NA.ne.0 ) then
244:              if ( NEP.ne.0.and.NA.ne.NEP ) then
245:                  write(NOUT1,7260) VNAME(1:NLEN)
246:                  call CNTERR('FATAL')
247:              else if ( NEP.eq.0 ) then
248:                  NEP = NA
249:              end if
250:          end if
251:          JEXTSC = 1
252: C
253:          if ( MODE.eq.'$BRANCH' ) then
254:              write(NOUT1,6666)
255:              write(NOUT1,7020) 'EXTSRC'
256:              call CNTERR('WARNING')
257:              JEXTSC = 0
258:              call CLEA ( EXTSRC , MXSTEP , 0.0 )
259:          end if
260: C

```

src/mvpburn/autobn.f

```

261: C ..... DETCTR .....
262: C
263:     else if ( VNAME(1:NLEN).eq.'DETCTR' ) then
264:         call R4READ( VNAME(1:NLEN), IDMONI, NA, 1, IEND )
265:         if ( IEND.ne.0 ) go to 330
266: C
267:         JDETCT = 1
268: C
269:         if ( MODE.eq.'$BRANCH' ) then
270:             write(NOUT1,6666)
271:             write(NOUT1,7020) 'DETCTR'
272:             call CNTERR('WARNING')
273:             JDETCT = 0
274:             IDMONI = 0
275:         end if
276: C
277:         if ( IDMONI.le.0.and.JDETCT.eq.1 ) then
278:             write(NOUT1,6888)
279:             write(NOUT1,*) ' invalid DETCTR number !!! '
280:             write(NOUT1,*) ' input DETCTR number is ',IDMONI,' !!!'
281:             call CNTERR('FATAL')
282:         endif
283: C
284: C ..... FMONIT .....
285: C
286:     else if ( VNAME(1:NLEN).eq.'FMONIT' ) then
287:         call R4READ( VNAME(1:NLEN), VMONIT, NA, -MXSTEP, IEND )
288:         if ( IEND.ne.0 ) go to 330
289: C
290:         if ( NA.ne.0 ) then
291:             if ( NEP.ne.0.and.NA.ne.NEP ) then
292:                 write(NOUT1,7260) VNAME(1:NLEN)
293:                 call CNTERR('FATAL')
294:             else if ( NEP.eq.0 ) then
295:                 NEP = NA
296:             end if
297:         end if
298:         JMONIT = 1
299: C
300:         if ( MODE.eq.'$BRANCH' ) then
301:             write(NOUT1,6666)
302:             write(NOUT1,7020) 'FMONIT'
303:             call CNTERR('WARNING')
304:             JMONIT = 0
305:             call CLEA ( VMONIT , MXSTEP , 0.0 )
306:         end if
307: C
308: C ..... REGPOW .....
309: C
310:     else if ( VNAME(1:NLEN).eq.'REGPOW' ) then
311:         call R4READ( VNAME(1:NLEN), REGPOW, NA, -MXSTEP, IEND )
312:         if ( IEND.ne.0 ) go to 330
313: C
314:         if ( NA.ne.0 ) then
315:             if ( NEP.ne.0.and.NA.ne.NEP ) then
316:                 write(NOUT1,7260) VNAME(1:NLEN)
317:                 call CNTERR('FATAL')
318:             else if ( NEP.eq.0 ) then
319:                 NEP = NA
320:             end if
321:         end if
322:         JRGPOW = 1
323: C
324:         if ( MODE.eq.'$BRANCH' ) then
325:             write(NOUT1,6666)

```

```

326:             write(NOUT1,7020) 'REGPOW'
327:             call CNTERR('WARNING')
328:             JRGPOW = 0
329:             call CLEA ( REGPOW , MXSTEP , 0.0 )
330:         end if
331: C
332: C ..... TRGPOW .....
333: C
334:     else if ( VNAME(1:NLEN).eq.'TRGPOW' ) then
335:         NN = 0
336:         135 call CHREAD( VNAME(1:NLEN), TRGPOW(MIN(MAXTRG,NN+1)), ' ',
337:             & INLEN, ITERM, IEND )
338: C
339:         if ( IEND.ne.0 ) go to 330
340: C
341:         NN = NN + 1
342:         if ( ITERM.eq.0 ) go to 135
343: Ctemp
344: Cdel write(NOUT1,*) ' Input TRGPOW(' ,NN,') is ',TRGPOW(NN)
345: Cend
346:         MATPOW = NN
347:         JRGNAM = 1
348:         if ( MODE.eq.'$BRANCH' ) then
349:             write(NOUT1,6666)
350:             write(NOUT1,7020) 'TRGPOW'
351:             MATPOW = 0
352:             JRGNAM = 0
353:         end if
354: C
355: C2005 ... end
356: C
357: C ..... CASEID .....
358: C
359:     else if ( VNAME(1:NLEN).eq.'CASEID' ) then
360:         call CHREAD( VNAME(1:NLEN), CASEID, ' ', INLEN, ITERM, IEND )
361:         if ( ITERM.eq.0 ) then
362:             write(NOUT1,6888)
363:             write(NOUT1,7240) VNAME(1:NLEN)
364:             stop 888
365:         end if
366:         if ( IEND.ne.0 ) go to 330
367: C
368: C ..... CASEREF .....
369: C
370:     else if ( VNAME(1:NLEN).eq.'CASEREF' ) then
371:         call CHREAD( VNAME(1:NLEN), CASREF, ' ', INLEN, ITERM, IEND )
372:         if ( ITERM.eq.0 ) then
373:             write(NOUT1,6888)
374:             write(NOUT1,7240) VNAME(1:NLEN)
375:             stop 888
376:         end if
377:         if ( IEND.ne.0 ) go to 330
378:         if ( MODE.ne.'$BRANCH' ) then
379:             write(NOUT1,6666)
380:             write(NOUT1,7000) VNAME(1:NLEN)
381:             call CNTERR('WARNING')
382:         end if
383: C00/01/21 Branching Case Get Reference case burn-up data
384:         if ( MODE.eq.'$BRANCH' ) then
385: C
386:             if ( CASREF.eq.' ' ) then
387:                 write(NOUT1,6888)
388:                 write(NOUT1,*)
389:                 & ' no specification of CASEREF ',
390:                 & 'in $BRANCH block.'

```

src/mvpburn/autobn.f

```

391:          call CNTERR('FATAL')
392:          go to 130
393:        end if
394: C
395:        if (DIRIN.eq.' ') then
396:          write(NOUT1,6888)
397:          write(NOUT1,*)
398:        >          ' PDS-directory is Null for Branching Case',
399:        >          ' PDS-directory must be inputted before CASEREF Data'
400:          call CNTERR('FATAL')
401:          go to 130
402:        end if
403: C ..... set NEP & PERIOD & POWERL from 'INC/_CBURNCl'
404:          call RWCOM1( 'READ', DIRIN, CASREF)
405: C2005 ... added
406:          IBS(20) = 0
407: C2004 ... end
408:        end if
409: C
410: C ..... PDS or PDSOUT .....
411: C
412:        else if ( VNAME(1:NLEN).eq.'PDS' ) then
413:          call CHREAD( VNAME(1:NLEN), DIROUT, ' '), INLEN, ITERM, IEND )
414:          DIRIN = DIROUT
415:          if ( ITERM.eq.0 ) then
416:            write(NOUT1,6888)
417:            write(NOUT1,7240) VNAME(1:NLEN)
418:            stop 888
419:          end if
420:          if ( IEND.ne.0 ) go to 330
421: C
422: C ..... POWERL .....
423: C
424:        else if ( VNAME(1:NLEN).eq.'POWERL' ) then
425:          call R4READ( VNAME(1:NLEN), POWERL, NA, -MXSTEP, IEND )
426:          if ( IEND.ne.0 ) go to 330
427:          if ( NA.ne.0 ) then
428:            if ( NEP.ne.0.and.NA.ne.NEP ) then
429:              write(NOUT1,7260) VNAME(1:NLEN)
430:              call CNTERR('FATAL')
431:            else if ( NEP.eq.0 ) then
432:              NEP = NA
433:            end if
434:          end if
435:          if ( MODE.eq.'$BRANCH' ) then
436:            write(NOUT1,6666)
437:            write(NOUT1,7020) 'POWERL'
438:            call CNTERR('WARNING')
439:          end if
440:          JPOWERL = 1
441: C
442: C ..... Period : MWD/MD/DAY/DAYINT/U235BURN .....
443: C
444:        else if ( VNAME(1:NLEN).eq.'MWD' .or. VNAME(1:NLEN).eq.'MWD'
445:          & .or. VNAME(1:NLEN).eq.'DAY' .or. VNAME(1:NLEN).eq.'DAYINT'
446:          & .or. VNAME(1:NLEN).eq.'U235BURN' ) then
447: C
448:          if ( JPERIOD.ne.0 ) then
449:            write(NOUT1,6888)
450:            write(NOUT1,*) ' "period" data appears more than once. <'
451:          &          , VNAME(1:NLEN), '>'
452:            call CNTERR('FATAL')
453:          end if
454:          if ( VNAME(1:NLEN).eq.'MWD' ) JPERIOD = 1
455:          if ( VNAME(1:NLEN).eq.'MWD' ) JPERIOD = 2

```

```

456:          if ( VNAME(1:NLEN).eq.'DAY' ) JPERIOD = 4
457:          if ( VNAME(1:NLEN).eq.'DAYINT' ) JPERIOD = 3
458:          if ( VNAME(1:NLEN).eq.'U235BURN' ) JPERIOD = 5
459: C
460:          call R4READ( VNAME(1:NLEN), PERIOD, NA, -MXSTEP, IEND )
461:          if ( IEND.ne.0 ) go to 330
462: C
463:          if ( NA.ne.0 ) then
464:            if ( NEP.ne.0.and.NA.ne.NEP ) then
465:              write(NOUT1,7260) VNAME(1:NLEN)
466:              call CNTERR('FATAL')
467:            else if ( NEP.eq.0 ) then
468:              NEP = NA
469:            end if
470:          end if
471:          if ( MODE.eq.'$BRANCH' ) then
472:            write(NOUT1,6666)
473:            write(NOUT1,7020) VNAME(1:NLEN)
474:            call CNTERR('WARNING')
475:          end if
476: C
477: C ..... NSTEP .....
478: C
479:          else if ( VNAME(1:NLEN).eq.'NSTEP' ) then
480:            call I4READ( VNAME(1:NLEN), NEP, NA, 1, IEND )
481:            if ( IEND.ne.0 ) go to 330
482:            if ( MODE.eq.'$BRANCH' ) then
483:              write(NOUT1,6666)
484:              write(NOUT1,7020) VNAME(1:NLEN)
485:              call CNTERR('WARNING')
486:            end if
487: C
488: C ..... START .....
489: C
490:          START = 1 : start calculation from step 1 (default)
491:          START > 1 : start calculation from non-initial step.
492:          Calculated PDS data must exist up to this step.
493:          START <= 0 : start calculation from the last existing calculated
494:          step (automatic restart)
495: C
496:          else if ( VNAME(1:NLEN).eq.'START' ) then
497:            call I4READ( VNAME(1:NLEN), ISTART, NA, 1, IEND )
498:            if ( IEND.ne.0 ) go to 330
499: C
500: C ..... STOP .....
501: C
502:          else if ( VNAME(1:NLEN).eq.'STOP' ) then
503:            call I4READ( VNAME(1:NLEN), ISTOP, NA, 1, IEND )
504:            if ( IEND.ne.0 ) go to 330
505: C2003
506:            if ( ISTOP.lt.0 ) then
507:              ISTOP = -ISTOP
508:              JHALF = 1
509:            end if
510: C2003 ... end
511: C
512: C ..... CPUTIME .....
513: C
514:          else if ( VNAME(1:NLEN).eq.'CPUTIME' ) then
515:            call R4READ( VNAME(1:NLEN), CPUMIN, NA, 1, IEND )
516:            CPULIM = CPUMIN*60.0
517:            if ( IEND.ne.0 ) go to 330
518: C
519: C ..... PRINT .....
520: C

```

src/mvpburn/autobn.f

```

521:      else if ( VNAME(1:NLEN).eq.'PRINT' ) then
522:        call I4READ( VNAME(1:NLEN), IPRT, NA, 1, IEND )
523:        if ( IEND.ne.0 ) go to 330
524: C
525: C ..... LCHA0 .....
526: C
527:      else if ( VNAME(1:NLEN).eq.'LCHA0' ) then
528:        call I4READ( VNAME(1:NLEN), IBC(7), NA, 1, IEND )
529:        if ( IEND.ne.0 ) go to 330
530: C
531: C ..... IBCMOD .....
532: C
533:      else if ( VNAME(1:NLEN).eq.'IBMOD' ) then
534:        call I4READ( VNAME(1:NLEN), IBC(8), NA, 1, IEND )
535:        if ( IEND.ne.0 ) go to 330
536:        if ( MODE.eq.'$BRANCH' ) then
537:          write(NOUT1,6666)
538:          write(NOUT1,7020) VNAME(1:NLEN)
539:          call CNTERR('WARNING')
540:          IBC(8) = 0
541:        end if
542: C
543: C ..... DEBUG .....
544: C
545:      else if ( VNAME(1:NLEN).eq.'DEBUG' ) then
546:        IBC(20) = 1
547: C2003 ... enable "DEBUG"and "DEBUG(h)"
548:        IBC = 0
549:        call FPROBE( IBC, ' ', CH )
550:        if ( IBC.ne.0.and.CH.ne.'(' ) IBC = 0
551:        if ( IBC.ne.0 ) then
552:          call I4READ( VNAME(1:NLEN), IBC(20), NA, 1, IEND )
553:          if ( IEND.ne.0 ) go to 330
554:        end if
555:        if ( IBC(20).lt.0.or.IBC(20).gt.2 ) then
556:          write(NOUT1,7123) IBC(20)
557: 7123      format(/lx,'XXX Value of "DEBUG" must be 0,1 or 2"/
558:      &          lx,4x,'DEBUG = ',I6)
559:          call CNTERR('FATAL')
560:        end if
561: C2003 ... end
562: C
563: C ..... DENINIT.....
564: C
565:      else if ( VNAME(1:NLEN).eq.'DENINIT' ) then
566:        call R4READ( VNAME(1:NLEN), BZERO , NA, 1, IEND )
567:        if ( IEND.ne.0 ) go to 330
568: C
569: C ..... PDSCHK.....
570: C
571:      else if ( VNAME(1:NLEN).eq.'PDSCHK' ) then
572:        call I4READ( VNAME(1:NLEN), IPDSCK, NA, 1, IEND )
573:        if ( IPDSCK.le.0 ) IPDSCK = 0
574:        if ( IEND.ne.0 ) go to 330
575: C
576: C ..... PC .....
577: C
578:      else if ( VNAME(1:NLEN).eq.'PC' ) then
579:        call I4READ( VNAME(1:NLEN), JPCFLG, NA, -MXSTEP, IEND )
580:        if ( IEND.ne.0 ) go to 330
581: C
582:        if ( MODE.eq.'$BRANCH' ) then
583:          write(NOUT1,6666)
584:          write(NOUT1,7020) VNAME(1:NLEN)
585:          call CNTERR('WARNING')

```

```

586: C2005 ... added
587:      call ICLEA( JPCFLG , MXSTEP , 0 )
588: C2005 ... end
589:      else
590:        if ( NA.ne.0 ) then
591:          if ( NEP.ne.0.and.NA.ne.NEP ) then
592:            write(NOUT1,7260) VNAME(1:NLEN)
593:            call CNTERR('FATAL')
594:          else if ( NEP.eq.0 ) then
595:            NEP = NA
596:          end if
597:        end if
598:      end if
599: C
600: C ..... SAVE-MVP-OUTPUT .....
601: C
602:      else if ( VNAME(1:NLEN).eq.'SAVE-MVP-OUTPUT' ) then
603:        call I4READ( VNAME(1:NLEN), JSVMVP, NA, -MXSTEP, IEND )
604:        if ( IEND.ne.0 ) go to 330
605:        if ( NA.ne.0 ) then
606:          if ( NEP.ne.0.and.NA.ne.NEP ) then
607:            write(NOUT1,7260) VNAME(1:NLEN)
608:            call CNTERR('FATAL')
609:          else if ( NEP.eq.0 ) then
610:            NEP = NA
611:          end if
612:        end if
613: C
614: C ..... SUBSTEPS .....
615: C
616:      else if ( VNAME(1:NLEN).eq.'SUBSTEPS' ) then
617:        call I4READ( VNAME(1:NLEN), NSTP, NA, -MXSTEP, IEND )
618:        if ( IEND.ne.0 ) go to 330
619: C
620:        NSTFLG = 1
621:        if ( NA.ne.0 ) then
622:          if ( NEP.ne.0.and.NA.ne.NEP ) then
623:            write(NOUT1,7260) VNAME(1:NLEN)
624:            call CNTERR('FATAL')
625:          else if ( NEP.eq.0 ) then
626:            NEP = NA
627:          end if
628:        end if
629: C
630:        if ( MODE.eq.'$BRANCH' ) then
631:          write(NOUT1,6666)
632:          write(NOUT1,7020) VNAME(1:NLEN)
633:          call CNTERR('WARNING')
634: C2005... added
635:          NSTFLG = 0
636:          call ICLEA ( NSTP , MXSTEP , 0 )
637: C2005 ... end
638:        end if
639: C
640: C ..... STEPS ..... (step#'s for $BRANCH mode)
641: C
642:      else if ( VNAME(1:NLEN).eq.'STEPS' ) then
643: C
644:        if ( MODE.ne.'$BRANCH' ) then
645:          write(NOUT1,7000) VNAME(1:NLEN)
646:          call CNTERR('WARNING')
647:        end if
648: C
649:        ISTFLG = 1
650:        call I4READ( VNAME(1:NLEN), ISTEPS, NA, -MXSTEP, IEND )

```

src/mvpburn/autobn.f

```

651:         if ( IEND.ne.0 ) go to 330
652: C2005 ... added
653:         NASTEP = NA
654: C2005 ... end
655:         do 140 I = 1, NA
656:             if ( ISTEPS(I).gt.0.and.ISTEPS(I).le.MXSTEP ) then
657:                 II = ISTEPS(I)
658:                 IBSTEP(II) = 1
659:             else if ( ISTEPS(I).le.0 .or. ISTEPS(I).gt.MXSTEP ) then
660:                 write(NOUT1,6888)
661:                 write(NOUT1,*) ' a "STEPS" data (= ',
662:                     & ISTEPS(I), ') is zero or negative or ',
663:                     & ' exceeds program limit (MXSTEP=', MXSTEP, ') '
664: C         NERR = NERR + 1
665:                 call CNTERR('FATAL')
666:             end if
667: 140 continue
668: C2005 ... added
669:         if ( MODE.ne.'$BRANCH' ) then
670:             ISTEFLG = 0
671:             call ICLEA ( IBSTEP , MXSTEP , 0 )
672:         endif
673: C2005 ... end
674: C
675: C ..... STDNUC .....
676: C
677:         else if ( VNAME(1:NLEN).eq.'STDNUC' ) then
678: C2003 call CHREAD( VNAME(1:NLEN), STDNUC, ' ', INLEN, ITERM, IEND )
679:         call I4READ( VNAME(1:NLEN), STDNUC, NA, -1, IEND )
680: C
681:         if ( MODE.eq.'$BRANCH' ) then
682:             write(NOUT1,6666)
683:             write(NOUT1,7020) VNAME(:NLEN)
684:             call CNTERR('WARNING')
685: C2005 ... added
686:             STDNUC = 0
687: C2005 ... end
688:         end if
689: C2003 if ( ITERM.eq.0 ) then
690: C         write(NOUT1,6888)
691: C         write(NOUT1,7240) VNAME(1:NLEN)
692: C         stop 888
693: C         end if
694:         if ( IEND.ne.0 ) go to 330
695: C
696: C ..... DEFINITION-CONVERSIONRATIO .....
697: C
698:         else if ( (NLEN.eq.8 .and. VNAME(1:NLEN).eq.'DEF-CONV') .or.
699: > (NLEN.eq.26 .and. VNAME(1:NLEN).eq.
700: > 'DEFINITION-CONVERSIONRATIO') ) then
701:             if ( MODE.eq.'$BRANCH' ) then
702:                 write(NOUT1,6666)
703:                 write(NOUT1,7020) VNAME(:NLEN)
704:                 call CNTERR('WARNING')
705:             end if
706:             call DEFCNV(NAMFIS,IFISDN ,FISFCT,NAMFER,IFRTDN ,FRTFCT,
707: > NFIS ,NFER ,MXFISS,NOUT1 )
708: C
709: C ..... HMINV0 .....
710: C
711:         else if ( VNAME(1:NLEN).eq.'HMINV0' ) then
712: C2005 ... deleted
713: Cdel if ( MODE.eq.'$BRANCH' ) then
714: Cdel write(NOUT1,6666)
715: Cdel write(NOUT1,7020) VNAME(:NLEN)

```

```

716: Cdel call CNTERR('WARNING')
717: Cdel end if
718: C2005 ... ended
719:         call R4READ( VNAME(1:NLEN), HMINV0, NA, -MAXTRG, IEND )
720:         if ( IEND.ne.0 ) go to 330
721: C
722:         IHMIN0 = NA
723:         if ( IHMIN0.le.0 ) then
724:             write(NOUT1,6888)
725:             write(NOUT1,*) ' number of inputed items',
726: & '(HMINV0) less or equal 0'
727:             call CNTERR('FATAL')
728:         end if
729: C
730:         if ( MODE.eq.'$BRANCH' ) then
731:             write(NOUT1,6666)
732:             write(NOUT1,7020) VNAME(1:NLEN)
733:             call CNTERR('WARNING')
734:             IHMIN0 = 0
735:         end if
736: C2003
737: C
738: C ..... ACCEPT-ZERO-RATE .....
739: C
740:         else if ( VNAME(1:NLEN).eq.'ACCEPT-ZERO-RATE' ) then
741:             IBC(19) = 1
742:             if ( MODE.eq.'$BRANCH' ) then
743:                 write(NOUT1,6666)
744:                 write(NOUT1,7020) VNAME(1:NLEN)
745:                 call CNTERR('WARNING')
746:             end if
747: C2003 ... end
748: C
749: C2005 ... added
750: C ..... YLDEDT .....
751: C
752:         else if ( VNAME(1:NLEN).eq.'YLDEDT' ) then
753:             call I4READ( VNAME(1:NLEN), MCDYLD, NA, 4, IEND )
754:             if ( IEND.ne.0 ) go to 330
755:             if ( MODE.eq.'$BRANCH' ) then
756:                 write(NOUT1,6666)
757:                 write(NOUT1,7020) VNAME(1:NLEN)
758:                 call CNTERR('WARNING')
759:                 IBC(11) = 0
760:             else
761:                 JFPYLD = 1
762:             end if
763: C2005 ... end
764: C
765: C ..... MVP symbolic parameter modifier .....
766: C
767: C @@name( v_1 v_2 ... v_nstep ) : floating point data
768: C @@#name( v_1 v_2 ... v_nstep ) : rounded to integer data
769: C
770:         else if ( VNAME(1:2).eq.'@@' ) then
771:             II = 0
772:             JJ = 3
773:             if ( VNAME(3:3).eq.'#' ) then
774:                 II = 1
775:                 JJ = 4
776:             end if
777:             KK = 0
778:             do 170 I = 1, NMVPPR
779:                 if ( PARAMN(I).eq.VNAME(JJ:NLEN) ) then
780:                     write(NOUT1,6666)

```


src/mvppburn/autobn.f

```

781:      &          ' MVP symbolic parameter modifier',
782:      &          ' <', VNAME(JJ:NLEN),
783:      &          '> specified more than once.'
784:      write(NOUT1,*)
785:      &          ' Values and data type are overwritten.'
786:      KK      = I
787:      call CNTERR('WARNING')
788:      go to 180
789:      end if
790: 170 continue
791:      KK      = NMVPPR + 1
792:      if ( KK.gt.MAXSPR ) then
793:      write(NOUT1,6888)
794:      write(NOUT1,*)
795:      &          ' number of MVP symbolic parameter',
796:      &          ' modifiers exceeds program limit (MAXSPR=', MAXSPR, ')
797:      call CNTERR('FATAL')
798:      KK      = 0
799:      else
800:      NMVPPR  = NMVPPR + 1
801:      end if
802: C
803: 180 continue
804:      if ( KK.eq.0 ) then
805:      call DMREAD( VNAME(:NLEN), NA, NB, IEND )
806:      else
807:      PARANM(KK) = VNAME(JJ:NLEN)
808:      if ( II.eq.0 ) then
809:      PARATP(KK) = 'F'
810:      else
811:      PARATP(KK) = 'I'
812:      end if
813:      call R8READ( VNAME(1:NLEN), PARAV(1,KK), NA, -MISTEP, IEND )
814:      if ( IEND.ne.0 ) go to 330
815:      if ( NA.ne.0 ) then
816:      if ( NEP.ne.0.and.NA.ne.NEP ) then
817:      write(NOUT1,7260) VNAME(1:NLEN)
818:      call CNTERR('FATAL')
819:      else if ( NEP.eq.0 ) then
820:      NEP      = NA
821:      end if
822:      end if
823:      end if
824: C
825: C ..... $END BURNUP .....
826: C
827:      else if ( VNAME(1:NLEN).eq.'$END' ) then
828:      call RESET
829:      go to 190
830: C
831: C ..... unknown data .....
832: C
833:      else
834:      write(NOUT1,6888)
835:      write(NOUT1,7200) VNAME(1:NLEN)
836:      call CNTERR('FATAL')
837:      call DMREAD( VNAME(:NLEN), NA, NB, IEND )
838:      end if
839: C
840:      go to 130
841: C
842: 190 continue
843: C-----
844: C ..... CHECK INPUT OR PROCESSING ERROR .....
845: C-----

```

```

846:      call CHKERR( 'FATAL', KK )
847: C
848:      if ( KK.gt.0 ) then
849:      write(NOUT1,6888)
850:      write(NOUT1,7145)
851:      stop 888
852:      end if
853: C
854:      if ( CASEID.eq.' ' ) then
855:      write(NOUT1,6888)
856:      write(NOUT1,*)
857:      &          ' no specification of CASEID ',
858:      &          ' in ', MODE(:ICLEN2(MODE)), ' block.'
859:      call CNTERR('FATAL')
860: C
861:      else
862:      do 200 I = 1, 4
863:      if ( INDEX(ALPNM,CASEID(I:I)).eq.0 ) then
864:      write(NOUT1,6888)
865:      write(NOUT1,*) I,
866:      &          '-th character of CASEID <', CASEID(I:I),
867:      &          '> is not',
868:      &          ' an alphabet or a numeric character.'
869:      call CNTERR('FATAL')
870:      end if
871: 200 continue
872:      end if
873: C
874:      if ( DIROUT.eq.' ' ) then
875:      write(NOUT1,6888)
876:      write(NOUT1,*)
877:      &          ' no specification of PDS ', ' in ',
878:      &          MODE(:ICLEN2(MODE)), ' block.'
879:      call CNTERR('FATAL')
880:      end if
881: C
882:      if ( DIRIN.eq.' ' ) then
883:      DIRIN  = DIROUT
884:      end if
885: C
886:      if ( MODE.eq.'$BURNUP'.and.JPERIOD.eq.0 ) then
887:      write(NOUT1,6888)
888:      write(NOUT1,*) ' no period data input.'
889:      call CNTERR('FATAL')
890:      end if
891: C2005 ... added
892: C
893: C ... Set ISTOP for no STOP input case
894: C
895:      if ( ISTOP.le.0 ) then
896:      if ( ISTFLG.eq.0 ) then
897:      ISTOP = NEP
898:      else
899:      do 206 I = 1,NEP
900:      if(ISTEP(I).eq.1) ISTOP = I
901: 206 continue
902:      endif
903:      endif
904: C ... Check ISTART & ISTOP values
905:      KSTART = ISTART
906:      if( ISTART.le.0 ) KSTART = 1
907: C
908:      if ( ISTOP.lt.KSTART ) then
909:      write(NOUT1,6888)
910:      write(NOUT1,*) ' STOP < START !!! '

```

src/mvppburn/autobn.f

```

911:      write(NOUT1,*) ' START = ',ISTART,' & STOP = ',ISTOP
912:      CALL CNTERR('FATAL')
913:      end if
914: C2005 ... end
915: C
916:      if ( MODE.eq.'$BURNUP') then
917:      if (IHMIN0.gt.0.and.JPERIOD.eq.5) then
918:      write(NOUT1,6888)
919:      write(NOUT1,'(A,A)') ' burn-up unit is ',
920:      >      ' U-235%. HMINV0 cannot used in this program.'
921:      call CNTERR('FATAL')
922:      end if
923: C
924:      if (ISTART.eq.1 .and. IBC(8).ne.0) then
925:      write(NOUT1,6666)
926:      write(NOUT1,'(A,A)') ' START(1) & IBCMOD',
927:      >      '(1)', reset IBCMOD(0) in program.
928:      IBC(8) = 0
929:      call CNTERR('WARNING')
930:      end if
931: C
932:      if (NSTFLG.ne.0) then
933:      do 205 I = 1,NEP
934:      if (POWERL(I).gt.0.0 .and. NSTP(I) .le. 1) then
935:      write(NOUT1,6888)
936:      write(NOUT1,*) ' POWERL is greater than 0.0 but,
937:      &      ' sub-step is less or equal 1 for ',I,'th step'
938:      CALL CNTERR('FATAL')
939:      end if
940:      205 continue
941:      end if
942: C
943: C2005 .. added
944: C
945: C ... Check IBNTYP
946: C
947:      if ( IBNTYP.gt.4) then
948:      write(NOUT1,6888)
949:      write(NOUT1,*) ' invalid BNTYP (>4) !!! '
950:      write(NOUT1,*) ' input BNTYP is ',IBNTYP,' !!! '
951:      CALL CNTERR('FATAL')
952:      end if
953: C
954: C ... Normal burnup mode check
955: C
956:      if ( IBNTYP.eq.0 ) then
957:      if ( JEXTSC.gt.0 ) then
958:      write(NOUT1,6888)
959:      write(NOUT1,*) ' BNPTYP is 0 !!! '
960:      write(NOUT1,*) ' But EXTRSC input data was found !!!'
961:      CALL CNTERR('FATAL')
962:      call CLEA ( EXTSRC , MXSTEP , 0.0 )
963:      end if
964: C
965:      if ( JDETC.TGT.0 ) then
966:      write(NOUT1,6888)
967:      write(NOUT1,*) ' BNPTYP is 0 !!! '
968:      write(NOUT1,*) ' But DETCTR input data was found !!!'
969:      CALL CNTERR('FATAL')
970:      IDMONI = 0
971:      end if
972: C
973:      if ( JMONIT.gt.0 ) then
974:      write(NOUT1,6888)
975:      write(NOUT1,*) ' BNPTYP is 0 !!! '

```

```

976:      write(NOUT1,*) ' But FMONIT input data was found !!!'
977:      CALL CNTERR('FATAL')
978:      call CLEA ( VMONIT , MXSTEP , 0.0 )
979:      end if
980: C
981:      if ( JRGPOW.gt.0 ) then
982:      write(NOUT1,6888)
983:      write(NOUT1,*) ' BNPTYP is 0 !!! '
984:      write(NOUT1,*) ' But REGPOW input data was found !!!'
985:      CALL CNTERR('FATAL')
986:      call CLEA ( REGPOW , MXSTEP , 0.0 )
987:      end if
988: C
989:      if ( JRGNAM.gt.0 ) then
990:      write(NOUT1,6888)
991:      write(NOUT1,*) ' BNPTYP is 0 !!! '
992:      write(NOUT1,*) ' But TRGPOW input data was found !!!'
993:      CALL CNTERR('FATAL')
994:      MATPOW = 0
995:      end if
996:      end if
997: C
998: C ... ADS burnup mode check ( IBNTYP=1 )
999: C
1000:      if ( IBNTYP.eq.1 ) then
1001:      if( JPERIOD.ne.3.and.JPERIOD.ne.4 ) then
1002:      write(NOUT1,6888)
1003:      write(NOUT1,*) ' Period data type was invalid (BNPTYP=1) !!! '
1004:      write(NOUT1,*) ' Period data type must be DAY or DAYINT !!! '
1005:      CALL CNTERR('FATAL')
1006:      else if( JPOWERL.gt.0 ) then
1007:      write(NOUT1,6888)
1008:      write(NOUT1,*) ' POWERL data input was found (BNPTYP=1) !!! '
1009:      write(NOUT1,*) ' POWERL data input should not be inputted !! '
1010:      CALL CNTERR('FATAL')
1011:      else if( JEXTSC.eq.0 ) then
1012:      write(NOUT1,6888)
1013:      write(NOUT1,*) ' EXTRSC input was not found !!!'
1014:      CALL CNTERR('FATAL')
1015:      else if( IBC(10).eq.1 ) then
1016:      write(NOUT1,6666)
1017:      write(NOUT1,*) ' BNPTYP input must be 1 !!! '
1018:      write(NOUT1,*) ' But your input is -1 !!! '
1019:      write(NOUT1,*) ' So reset BNPTYP( 1 ) !!! '
1020:      CALL CNTERR('WARNING')
1021:      IBC(10) = 0
1022:      else if ( JDETC.TGT.0 ) then
1023:      write(NOUT1,6666)
1024:      write(NOUT1,*) ' DETCTR input data was ignored !!!'
1025:      CALL CNTERR('WARNING')
1026:      IDMONI = 0
1027:      else if ( JMONIT.gt.0 ) then
1028:      write(NOUT1,6666)
1029:      write(NOUT1,*) ' FMONIT input data was ignored !!!'
1030:      CALL CNTERR('WARNING')
1031:      call CLEA ( VMONIT , MXSTEP , 0.0 )
1032:      else if ( JRGPOW.gt.0 ) then
1033:      write(NOUT1,6666)
1034:      write(NOUT1,*) ' REGPOW input data was ignored !!!'
1035:      CALL CNTERR('WARNING')
1036:      call CLEA ( REGPOW , MXSTEP , 0.0 )
1037:      else if ( JRGNAM.gt.0 ) then
1038:      write(NOUT1,6666)
1039:      write(NOUT1,*) ' TRGPOW input data was ignored !!!'
1040:      CALL CNTERR('WARNING')

```

src/mvpburn/autobn.f

```

1041:          MATPOW = 0
1042:        end if
1043:      end if
1044: C
1045: C ... Monitoring Reaction Rate constant Burnup mode check ( IBNTYP=2/3 )
1046: C
1047:   if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
1048:     if( JPERIOD.ne.3.and.JPERIOD.ne.4 ) then
1049:       write(NOUT1,6888)
1050:     write(NOUT1,*) ' Period data type was invalid (BNPTYP=2/3) !!! '
1051:     write(NOUT1,*) ' Period data type must be DAY or DAYINT !!! '
1052:     CALL CNTERR('FATAL')
1053:   else if( JPOWERL.gt.0 ) then
1054:     write(NOUT1,6888)
1055:     write(NOUT1,*) ' POWERL data input was found (BNPTYP=2/3) !!! '
1056:     write(NOUT1,*) ' POWERL data input should not be inputted !! '
1057:     CALL CNTERR('FATAL')
1058:   else if( JEXTSC.gt.0 ) then
1059:     write(NOUT1,6666)
1060:     write(NOUT1,*) ' EXTRSC input data was ignored !!!'
1061:     CALL CNTERR('WARNING')
1062:     call CLEA ( EXTRSC , MXSTEP , 0.0 )
1063:   else if ( JDETCT.eq.0 ) then
1064:     write(NOUT1,6888)
1065:     write(NOUT1,*) ' DETCTR input was not found !!!'
1066:     CALL CNTERR('FATAL')
1067:   else if ( JMONIT.eq.0 ) then
1068:     write(NOUT1,6666)
1069:     write(NOUT1,*) ' FMONIT input was not found !!!'
1070:     CALL CNTERR('FATAL')
1071:   else if ( JRGPOW.gt.0 ) then
1072:     write(NOUT1,6666)
1073:     write(NOUT1,*) ' REGPOW input data was ignored !!!'
1074:     CALL CNTERR('WARNING')
1075:     call CLEA ( REGPOW , MXSTEP , 0.0 )
1076:   else if ( JRGNAM.gt.0 ) then
1077:     write(NOUT1,6666)
1078:     write(NOUT1,*) ' TRGPOW input data was ignored !!!'
1079:     CALL CNTERR('WARNING')
1080:     MATPOW = 0
1081:   end if
1082: end if
1083: C
1084: C ... Specified Region Power constant Burnup mode check ( IBNTYP=4 )
1085: C
1086:   if ( IBNTYP.eq.4 ) then
1087:     if( JPERIOD.ne.3.and.JPERIOD.ne.4 ) then
1088:       write(NOUT1,6888)
1089:     write(NOUT1,*) ' Period data type was invalid (BNPTYP=4) !!! '
1090:     write(NOUT1,*) ' Period data type must be DAY or DAYINT !!! '
1091:     CALL CNTERR('FATAL')
1092:   else if( JPOWERL.gt.0 ) then
1093:     write(NOUT1,6888)
1094:     write(NOUT1,*) ' POWERL data input was found (BNPTYP=4) !!! '
1095:     write(NOUT1,*) ' POWERL data input should not be inputted !! '
1096:     CALL CNTERR('FATAL')
1097:   else if( JEXTSC.gt.0 ) then
1098:     write(NOUT1,6666)
1099:     write(NOUT1,*) ' EXTRSC input data was ignored !!!'
1100:     CALL CNTERR('WARNING')
1101:     call CLEA ( EXTRSC , MXSTEP , 0.0 )
1102:   else if( IBC(10).eq.1 ) then
1103:     write(NOUT1,6666)
1104:     write(NOUT1,*) ' BNPTYP input must be 4 !!! '
1105:     write(NOUT1,*) ' But your input is -4 !!! '

```

```

1106:       write(NOUT1,*) ' So reset BNPTYP( 4 ) !!! '
1107:       CALL CNTERR('WARNING')
1108:       IBC(10) = 0
1109:     else if ( JDETCT.gt.0 ) then
1110:       write(NOUT1,6666)
1111:       write(NOUT1,*) ' DETCTR input data was ignored !!!'
1112:       CALL CNTERR('WARNING')
1113:       IDMONI = 0
1114:     else if ( JMONIT.gt.0 ) then
1115:       write(NOUT1,6666)
1116:       write(NOUT1,*) ' FMONIT input data was ignored !!!'
1117:       CALL CNTERR('WARNING')
1118:       call CLEA ( VMONIT , MXSTEP , 0.0 )
1119:     else if ( JRGPOW.eq.0 ) then
1120:       write(NOUT1,6888)
1121:       write(NOUT1,*) ' REGPOW data was not found !!!'
1122:       CALL CNTERR('FATAL')
1123:     else if ( JRGNAM.eq.0 ) then
1124:       write(NOUT1,6888)
1125:       write(NOUT1,*) ' TRGPOW input was not found !!!'
1126:       CALL CNTERR('FATAL')
1127:     else if ( MATPOW.le.0 ) then
1128:       write(NOUT1,6888)
1129:       write(NOUT1,*) ' TRGPOW input was invalid !!!'
1130:       write(NOUT1,*) ' No of TRGPOW data was zero !!! '
1131:       CALL CNTERR('FATAL')
1132:     end if
1133:   end if
1134: C
1135: C *** Set POWERL data for IBNTYP>0 case
1136: C
1137:   if ( IBNTYP.eq.1 ) then
1138:     do 701 I = 1 , NEP
1139:       POWERL(I) = 0.0
1140:       if( EXTRSC(I).gt.0.0 ) POWERL(I) = 1.0
1141:       if( EXTRSC(I).lt.0.0 ) POWERL(I) = -1.0
1142:     continue
1143:   701 else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
1144:     do 702 I = 1 , NEP
1145:       POWERL(I) = 0.0
1146:       if( VMONIT(I).gt.0.0 ) POWERL(I) = 1.0
1147:       if( VMONIT(I).lt.0.0 ) POWERL(I) = -1.0
1148:     continue
1149:   702 else if ( IBNTYP.eq.4 ) then
1150:     do 703 I = 1 , NEP
1151:       POWERL(I) = 0.0
1152:       if( REGPOW(I).gt.0.0 ) POWERL(I) = 1.0
1153:       if( REGPOW(I).lt.0.0 ) POWERL(I) = -1.0
1154:     continue
1155:   703 end if
1156: C
1157: C2005 ... end
1158: C
1159: C *** CHECK PERIOD & POWERL DATA
1160: C
1161:   do 207 I = 1,NEP
1162: C2005 ... added
1163:   if ( IBNTYP.eq.0 ) then
1164: C2005 ... end
1165: C
1166:   if ( PERIOD(I).eq.0.0.and.POWERL(I).ge.0.0 ) then
1167:     write(NOUT1,6888)
1168:     write(NOUT1, '(A/A,I5)')
1169:       & ' PERIOD is 0.0. but POWERL >= 0.0',
1170:       & ' BURN-UP step =',I

```

src/mvpburn/autobn.f

```

1171:          CALL CNTERR('FATAL')
1172:        end if
1173: C
1174:        if (POWERL(I).eq.0.0) then
1175:          if (JPERIOD.ne.3.and.JPERIOD.ne.4) then
1176: C
1177:            if (PERIOD(I).ge.0.0) then
1178:              write(NOUT1,6888)
1179:              write(NOUT1,'(A/A,I5)')
1180:            &          ' POWERL is 0.0. but PERIOD >= 0.0',
1181:            &          ' BURN-UP step =',I
1182:            CALL CNTERR('FATAL')
1183:          end if
1184: C
1185:        else
1186:          if (PERIOD(I).le.0.0) then
1187:            write(NOUT1,6888)
1188:            write(NOUT1,'(A/A,I5)')
1189:          &          ' POWERL is 0.0. but PERIOD <= 0.0',
1190:          &          ' BURN-UP step =',I
1191:          CALL CNTERR('FATAL')
1192:        end if
1193:      end if
1194:    end if
1195: C
1196:    if (POWERL(I).gt.0.0.and.PERIOD(I).le.0.0) then
1197:      write(NOUT1,6888)
1198:      write(NOUT1,'(A/A,I5)')
1199:    &      ' POWERL is > 0.0. but PERIOD <= 0.0',
1200:    &      ' in burn-up step :',I
1201:    CALL CNTERR('FATAL')
1202:  end if
1203: C
1204: C2005 ... added for IBNTYP=1,2,3,4
1205: C
1206: C ..... IBNTYP=1
1207: C
1208:   else if ( IBNTYP.eq.1 ) then
1209:     if (PERIOD(I).eq.0.0.and.EXTSRC(I).ge.0.0) then
1210:       write(NOUT1,6888)
1211:       write(NOUT1,'(A/A,I5)')
1212:     &       ' PERIOD is 0.0. but EXTSRC >= 0.0',
1213:     &       ' BURN-UP step =',I
1214:     CALL CNTERR('FATAL')
1215:   end if
1216: C
1217:   if (EXTSRC(I).eq.0.0) then
1218:     if (JPERIOD.ne.3.and.JPERIOD.ne.4) then
1219: C
1220:       if (PERIOD(I).ge.0.0) then
1221:         write(NOUT1,6888)
1222:         write(NOUT1,'(A/A,I5)')
1223:       &         ' EXTSRC is 0.0. but PERIOD >= 0.0',
1224:       &         ' BURN-UP step =',I
1225:       CALL CNTERR('FATAL')
1226:     end if
1227: C
1228:   else
1229:     if (PERIOD(I).le.0.0) then
1230:       write(NOUT1,6888)
1231:       write(NOUT1,'(A/A,I5)')
1232:     &       ' EXTSRC is 0.0. but PERIOD <= 0.0',
1233:     &       ' BURN-UP step =',I
1234:     CALL CNTERR('FATAL')
1235:   end if

1236:         end if
1237:       end if
1238: C
1239:       if (EXTSRC(I).gt.0.0.and.PERIOD(I).le.0.0) then
1240:         write(NOUT1,6888)
1241:         write(NOUT1,'(A/A,I5)')
1242:       &         ' EXTSRC is > 0.0. but PERIOD <= 0.0',
1243:       &         ' in burn-up step :',I
1244:       CALL CNTERR('FATAL')
1245:     end if
1246: C
1247: C ..... IBNTYP=2 or IBNTYP=3
1248: C
1249:   else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
1250:     if (PERIOD(I).eq.0.0.and.VMONIT(I).ge.0.0) then
1251:       write(NOUT1,6888)
1252:       write(NOUT1,'(A/A,I5)')
1253:     &       ' PERIOD is 0.0. but FMONIT >= 0.0',
1254:     &       ' BURN-UP step =',I
1255:     CALL CNTERR('FATAL')
1256:   end if
1257: C
1258:   if (VMONIT(I).eq.0.0) then
1259:     if (JPERIOD.ne.3.and.JPERIOD.ne.4) then
1260: C
1261:       if (PERIOD(I).ge.0.0) then
1262:         write(NOUT1,6888)
1263:         write(NOUT1,'(A/A,I5)')
1264:       &         ' FMONIT is 0.0. but PERIOD >= 0.0',
1265:       &         ' BURN-UP step =',I
1266:       CALL CNTERR('FATAL')
1267:     end if
1268: C
1269:   else
1270:     if (PERIOD(I).le.0.0) then
1271:       write(NOUT1,6888)
1272:       write(NOUT1,'(A/A,I5)')
1273:     &       ' FMONIT is 0.0. but PERIOD <= 0.0',
1274:     &       ' BURN-UP step =',I
1275:     CALL CNTERR('FATAL')
1276:   end if
1277:   end if
1278: C
1279:   if (VMONIT(I).gt.0.0.and.PERIOD(I).le.0.0) then
1280:     write(NOUT1,6888)
1281:     write(NOUT1,'(A/A,I5)')
1282:   &     ' FMONIT is > 0.0. but PERIOD <= 0.0',
1283:   &     ' in burn-up step :',I
1284:   CALL CNTERR('FATAL')
1285:   end if
1286: C
1287: C ..... IBNTYP=4
1288: C
1289:   else if ( IBNTYP.eq.4 ) then
1290:     if (PERIOD(I).eq.0.0.and.REGPOW(I).ge.0.0) then
1291:       write(NOUT1,6888)
1292:       write(NOUT1,'(A/A,I5)')
1293:     &       ' PERIOD is 0.0. but REGPOW >= 0.0',
1294:     &       ' BURN-UP step =',I
1295:     CALL CNTERR('FATAL')
1296:   end if
1297: C
1298:   if (REGPOW(I).eq.0.0) then
1299:     if (JPERIOD.ne.3.and.JPERIOD.ne.4) then
1300:

```

src/mvpburn/autobn.f

```

1301: C
1302:         if (PERIOD(I).ge.0.0) then
1303:             write(NOUT1,6888)
1304:             write(NOUT1,'(A/A,I5)')
1305:             &         ' REGPOW is 0.0. but PERIOD >= 0.0',
1306:             &         ' BURN-UP step =',I
1307:             CALL CNTERR('FATAL')
1308:         end if
1309: C
1310:         else
1311:             if (PERIOD(I).le.0.0) then
1312:                 write(NOUT1,6888)
1313:                 write(NOUT1,'(A/A,I5)')
1314:                 &         ' REGPOW is 0.0. but PERIOD <= 0.0',
1315:                 &         ' BURN-UP step =',I
1316:                 CALL CNTERR('FATAL')
1317:             end if
1318:         end if
1319:     end if
1320: C
1321:         if (REGPOW(I).gt.0.0.and.PERIOD(I).le.0.0) then
1322:             write(NOUT1,6888)
1323:             write(NOUT1,'(A/A,I5)')
1324:             &         ' REGPOW is > 0.0. but PERIOD <= 0.0',
1325:             &         ' in burn-up step :',I
1326:             CALL CNTERR('FATAL')
1327:         end if
1328: C
1329:         end if
1330: C2005 ... end
1331: C
1332: 207 continue
1333: C
1334:         ACMBNP = 0.0
1335:         do 208 I = 1,NEP
1336: C
1337:             if (JPERIOD.eq.1.or.JPERIOD.eq.2.or.JPERIOD.eq.5) then
1338:                 if (POWERL(I).gt.0.0) then
1339:                     ACMBUN = PERIOD(I)
1340:                     if (ACMBUN.le.ACMBNP) then
1341:                         write(NOUT1,6888)
1342:                         write(NOUT1,7010) I
1343:                         CALL CNTERR('FATAL')
1344:                     end if
1345:                     ACMBNP = ACMBUN
1346:                 end if
1347:             end if
1348: C
1349:             if (JPERIOD.eq.3) then
1350:                 if (POWERL(I).ge.0.0) then
1351:                     ACMBUN = PERIOD(I)
1352:                     if (ACMBUN.le.ACMBNP) then
1353:                         write(NOUT1,6888)
1354:                         write(NOUT1,7010) I
1355:                         CALL CNTERR('FATAL')
1356:                     end if
1357:                     ACMBNP = ACMBUN
1358:                 end if
1359:             end if
1360: C
1361:             if (JPERIOD.eq.4) then
1362:                 if (POWERL(I).ge.0.0) then
1363:                     ACMBUN = ACMBNP+PERIOD(I)
1364:                     if (ACMBUN.le.ACMBNP) then
1365:                         write(NOUT1,6888)

```

```

1366:                 write(NOUT1,7010) I
1367:                 CALL CNTERR('FATAL')
1368:             end if
1369:             ACMBNP = ACMBUN
1370:         end if
1371:     end if
1372: C
1373: 208 continue
1374: end if
1375: C
1376: C ... get number of steps (NEP,NEP1) of the reference case CASREF
1377: C
1378: if ( MODE.eq.'$BRANCH' ) then
1379: C
1380:     if ( CASEID.eq.CASREF ) then
1381:         write(NOUT1,6888)
1382:         write(NOUT1,*) ' CASEID <', CASEID,
1383:             &         '> is identical to that of reference case <',
1384:             &         CASREF, '> in BRANCH mode.'
1385:         call CNTERR('FATAL')
1386:     end if
1387: C
1388:     if ( ISTFLG .eq.0 ) then
1389: C2005
1390: C         if ( ISTART.gt.0.and.ISTOP.gt.0 ) then
1391: C             do 210 I = 1, MXSTEP
1392: C                 if ( I.ge.ISTART.and.I.le.ISTOP ) then
1393: C                     IBSTEP(I) = 1
1394: C                 end if
1395: C                 continue
1396: C             end if
1397: C
1398:             call ICLEA ( IBSTEP , MXSTEP , 0 )
1399: C
1400:             do 210 I = KSTART , ISTOP
1401:                 IBSTEP(I) = 1
1402:             continue
1403: C
1404:         else
1405: Cdel         if ( ISTART.gt.0.and.ISTOP.gt.0 ) then
1406: C             do 215 I = 1, MXSTEP
1407: C                 if ( I.lt.ISTART.and.I.gt.ISTOP ) then
1408: C                     if ( I.lt.KSTART.or .I.gt.ISTOP ) then
1409: C                         IBSTEP(I) = 0
1410: C                     end if
1411: C                 continue
1412: C             end if
1413: C
1414:         end if
1415: C2005 ... end
1416: C
1417:         NBSTEP = 0
1418:         do 220 I = 1, MXSTEP
1419:             if ( IBSTEP(I).ne.0 ) then
1420:                 NBSTEP = NBSTEP + 1
1421:                 write(NOUT1,*)
1422:                 &         '>>> BRANCH-OFF CALCULATION REQUESTED FOR STEP ', I
1423:             end if
1424:             220 continue
1425: C
1426:         if ( NBSTEP.eq.0 ) then
1427:             write(NOUT1,6888)
1428:             write(NOUT1,*) ' no step is specified ',
1429:                 &         ' for $BRANCH mode.'
1430: C2005 ... added

```

src/mvpburn/autobn.f

```

1431:      write(NOUT1,*) ' START is ',ISTART,' & STOP is ',ISTOP
1432:      if( NASTEP.gt.0 ) then
1433:      write(NOUT1,*) ' input STEPS : '
1434:      write(NOUT1,'(20i3)') (ISTEPS(i),i=1,NASTEP)
1435:      end if
1436: C2005 ... end
1437:      call CNTERR('FATAL')
1438: C
1439:      else
1440: C2003      call CKBNUP( NOUT1, NOUT2, DIRIN, DIROUT, CASREF, ARRAY,
1441: C      &      ARRAY, MEMORY, IBSTEP, ISTART, ISTOP, IERR )
1442:      call CKBNUP( NOUT1, NOUT2, DIRIN, DIROUT, CASREF, ARRAY,
1443:      &      ARRAY, MEMORY, IBSTEP, ISTART, ISTOP, IERR )
1444:      if ( ISTART.ne.0 ) ISTART = ISTART1
1445: C2003 ... end
1446: C
1447:      if ( IERR.ne.0 ) then
1448:      write(NOUT1,6888)
1449:      write(NOUT1,*)
1450:      &      ' failed to extract necessary information ',
1451:      &      ' from reference case of branch mode.'
1452:      call CNTERR('FATAL')
1453:      end if
1454:      end if
1455: C
1456:      end if
1457: C
1458:      IBC(1) = NEP
1459:      IBC(2) = JPERIOD
1460:      IBC(3) = IPRT
1461:      IBEDIT = IPRT
1462: C2003
1463:      NHALF = 0
1464: C
1465: C2005 ... modified
1466: Cmod if ( MODE.eq.'$BURNUP'.and.JPOWERL.eq.0 ) then
1467: if ( MODE.eq.'$BURNUP'.and.JPOWERL.eq.0.and.IBNTYP.eq.0 ) then
1468: write(NOUT1,6888)
1469: write(NOUT1,*) ' no POWERL data input.'
1470: call CNTERR('FATAL')
1471: end if
1472: C
1473: if ( MODE.eq.'$BURNUP'.and.ISTART.gt.NEP ) then
1474: write(NOUT1,6888)
1475: write(NOUT1,*)
1476: &      ' can not start from step ', ISTART,
1477: &      ' which is greater than NSTEP.'
1478: call CNTERR('FATAL')
1479: C2005 ... added
1480: end if
1481: C2005 ... end
1482: C
1483: C ... starts from non initial step or restart ....
1484: C
1485: C2005 modified
1486: Cm else if ( MODE.eq.'$BURNUP'.and.(ISTART.gt.1.or.ISTART.le.0) )
1487: Cm & then
1488: C
1489: JSTART = ISTART
1490: C
1491: C .... '$BURNUP' case
1492: C
1493: if( MODE.eq.'$BURNUP' ) then
1494: if( JSTART.eq.1 ) then
1495: ISTART = 1

```

```

1496: C
1497: else
1498: C2005 ... end
1499: C
1500: C
1501: C ... check CASE+'HT'+step and CASE+'REST' ...
1502: C2003 ... add NHALF
1503: call CKSTEP( NOUT1, NOUT2, DIRIN, DIROUT, CASEID, ARRAY, ARRAY,
1504: &      MEMORY, NNSTEP, NHALF, IERR )
1505: C
1506: C ... calculated steps exist already ...
1507: C
1508: C2005 ISTAR0 = ISTART
1509: C
1510: if ( NNSTEP.gt.0 ) then
1511: C
1512: if ( ISTART.le.0 ) then
1513: ISTART = NNSTEP
1514: C
1515: else if ( ISTART.gt.NNSTEP ) then
1516: write(NOUT1,6888)
1517: write(NOUT1,*)
1518: &      ' calculation from non-initial',
1519: &      ' step( ', ISTART, ' ) is refused because',
1520: &      ' effective ',CASEID,'HT## members exist',
1521: &      ' upto ',NNSTEP,'-th step'
1522: write(NOUT1,*) ' start step should be ',NNSTEP
1523: call CNTERR('FATAL')
1524: end if
1525: C
1526: C ... no calculated steps exist ...
1527: C
1528: C
1529: C else
1530: if ( ISTART.gt.1 ) then
1531: write(NOUT1,6888)
1532: write(NOUT1,*)
1533: &      ' calculation from non-initial',
1534: &      ' step ', ISTART, ' is refused because no',
1535: &      ' effective ',CASEID,'HT## members exist.'
1536: call CNTERR('FATAL')
1537: C
1538: C
1539: C ISTART = 1
1540: C
1541: C
1542: C end if
1543: C
1544: C if ( ISTART.gt.1 ) then
1545: C write(NOUT1,*)
1546: C &      '>>> STARTING CALCULATION FROM NON INITIAL STEP ',
1547: C &      ISTART
1548: C
1549: C
1550: C ... reset "current" step # in CASE+'REST'
1551: C
1552: C call CHKERR( 'FATAL', NERR )
1553: C
1554: C if ( NERR.eq.0.and.NNSTEP.gt.0 ) then
1555: C call RWREST( 'READ', DIRIN, CASEID )
1556: C NOWSTP = ISTART
1557: C IBEND = 0
1558: C call RWREST( 'WRITE', DIRIN, CASEID )
1559: C write(NOUT1,*)
1560: C &      '>>> CURRENT STEP NUMBER IN MEMBER ',

```

src/mvpburn/autobn.f

```

1561:      &      CASEID,'REST IS RESET TO ', ISTART
1562:      end if
1563: C2005 ... modified
1564:      end if
1565:      end if
1566: C
1567: C2005 deleted after this line
1568: C
1569: C2003 ... restart calculation in BRANCH mode.
1570: C
1571: Cdel else if ( MODE.eq.'$BRANCH'.and.ISTART.le.0 ) then
1572: Cdel call CKSTP2( NOUT1, NOUT2, DIRIN, DIROUT, CASEID, ARRAY, ARRAY,
1573: Cdel &      MEMORY, NNSTEP, IERR )
1574: Cdel if ( NNSTEP.gt.0 ) then
1575: Cdel ISTART = NNSTEP
1576: Cdel ISTART = MAX(ISTART,ISTRT1)
1577: Cdel else
1578: Cdel write(NOUT1,*) '>>> No previously calculated steps for',
1579: Cdel &      ' restart mode in BRANCH calculation.'
1580: Cdel ISTART = ISTRT1
1581: Cdel end if
1582: C2003 ... end
1583: C
1584: Cdel else
1585: Cdel ISTART = 1
1586: Cdel end if
1587: C2005 ... end of delection
1588: C
1589: C2005 ... added
1590: C
1591: C .... CASE of $BRANCH
1592: C
1593:      if ( MODE.eq.'$BRANCH') then
1594:      if ( JSTART.gt.0) then
1595:      if( ISTART.lt.ISTRT1) ISTART = ISTRT1
1596:      NOWSTP = ISTART
1597: C
1598:      else
1599:      KSTART = ISTOP
1600:      call CKSTP2( NOUT1, NOUT2 , DIRIN , DIROUT, CASEID, ARRAY,
1601:      &      ARRAY, MEMORY, KSTART, IERR , IBSTEP, ISTOP )
1602:      ISTART = KSTART
1603:      NOWSTP = KSTART
1604:      end if
1605: C
1606:      endif
1607: C
1608: C
1609:      if ( MODE.eq.'$BURNUP'.and.ISTOP.gt.NEP ) then
1610:      write(NOUT1,6888)
1611:      write(NOUT1,*)
1612:      &      ' cannot calculate until step ',
1613:      &      ISTOP, ' which is greater than NSTEP.'
1614:      call CNTERR('FATAL')
1615: C2005 ... moved
1616:      end if
1617: C2005 ... end
1618: C2005 ... deleted
1619: Cdel else if ( ISTOP.eq.0 ) then
1620: Cdel ISTOP = NEP
1621: Cdel end if
1622: Cmove end if
1623: C2005 ... end
1624: C
1625: C      if ( IBCS(5).ne.0 .and. STDNUC.eq.' ' ) then

```

```

1626: C      write(NOUT1,*)
1627: C      &      'xxx ERROR : no input of STDNUC for non-zero IBC(5).'
1628: C      NERR = NERR + 1
1629: C      end if
1630: C
1631: C
1632: C2003 if ( STDNUC.ne.' ' ) then
1633: C      if ( STDNUC.ne.0 ) then
1634: C      call ZZMMM(1,1,STDNUC,CWRK(1:6))
1635: C      write(NOUT1,*) '>>> NUCLIDE FOR EXPOSURE INDICATOR (STDNUC)',
1636: C      &      ' IS <', STDNUC,'(',CWRK(1:6),')>'
1637: C      IBC(5) = 1
1638: C      end if
1639: C2003 ... end
1640: C
1641: C2005
1642: C      if( JFPYLD.eq.1) then
1643: C      IBC(11) = 1
1644: C      if( MCDYLD(1).gt.0 ) then
1645: C      NCXE35 = MCDYLD(1)
1646: C      else
1647: C      NCXE35 = 541350
1648: C      endif
1649: C      if( MCDYLD(2).gt.0 ) then
1650: C      NCII35 = MCDYLD(2)
1651: C      else
1652: C      NCII35 = 531350
1653: C      endif
1654: C      if( MCDYLD(3).gt.0 ) then
1655: C      NCSM49 = MCDYLD(3)
1656: C      else
1657: C      NCSM49 = 621490
1658: C      endif
1659: C      if( MCDYLD(4).gt.0 ) then
1660: C      NCPM49 = MCDYLD(4)
1661: C      else
1662: C      NCPM49 = 611490
1663: C      endif
1664: C      endif
1665: C2005 ... end
1666: C
1667: C      if ( IBCS(6).ne.0 .and. NFIS.eq.0 ) then
1668: C      write(NOUT1,*)
1669: C      &      'xxx ERROR : no input of NAMFIS etc. for non-zero IBC(6).'
1670: C      NERR = NERR + 1
1671: C      end if
1672: C      if ( IBCS(6).ne.0 .and. NFER.eq.0 ) then
1673: C      write(NOUT1,*)
1674: C      &      'xxx ERROR : no input of NAMFRT etc. for non-zero IBC(6).'
1675: C      NERR = NERR + 1
1676: C      end if
1677: C
1678: C      if ( NFIS.gt.0 .or. NFER.gt.0 ) then
1679: C      IBC(6) = 1
1680: C      end if
1681: C
1682: C      ... branch flag.
1683: C
1684: C      if ( MODE.eq.'$BRANCH' ) then
1685: C      IBC(4) = 1
1686: C      else
1687: C      IBC(4) = 0
1688: C      end if
1689: C
1690: C      ... check PC method flags

```

src/mvppburn/autobn.f

```

1691: C
1692:   if ( MODE.eq.'$BURNUP' ) then
1693:     do 230 I = 1, NEP
1694:       if ( JPCFLG(I).ne.0 ) then
1695: C2005 ... added
1696:         METHPC = 1
1697: C2005 ... end
1698:         write(NOUT1,7280)
1699:       &       '>>> PC METHOD WILL BE APPLIED FOR AT LEAST ONE STEP'
1700:       go to 240
1701:     end if
1702:   230 continue
1703:   240 continue
1704: end if
1705: C
1706: C .. check CPU limit
1707: C
1708:   if ( CPULIM.le.0.0 ) then
1709:     write(NOUT1,7280) '>>> NO CPU TIME LIMIT'
1710:   end if
1711: C
1712: C ... MVP symbolic parameter modifiers
1713: C
1714:   if ( NMVPPR.gt.0 ) then
1715:     write(NOUT1,7280) '>>> SYMBOLIC PARAMETER MODIFIER WILL BE USED'
1716:     do 250 K = 1, NMVPPR
1717:       if ( PARATP(K).eq.'F' ) then
1718:         write(NOUT1,7040) PARANM(K), 'Floating point'
1719:         write(NOUT1,7060) (I,PARAV(I,K),I=1,NEP)
1720:       else
1721:         write(NOUT1,7040) PARANM(K), 'Integer'
1722:         write(NOUT1,7080) (I,NINT(PARAV(I,K)),PARAV(I,K),I=1,NEP)
1723:       end if
1724:     250 continue
1725:   end if
1726: C
1727:   7040 format(/1X,' *parameter name: <',A,'> data type: ',A)
1728:   7060 format(/1X,' Step # Value'/(1X,2X,I4,5X,1P,D13.5))
1729:   7080 format(/1X,' Step # Integer Float'/(1X,2X,I4,5X,1P,D13.5))
1730:   &      (1X,2X,I4,5X,I8,3X,1P,D13.5))
1731: C
1732:   call CHKERR( 'FATAL', KK )
1733: C
1734:   if ( KK.gt.0 ) then
1735:     write(NOUT1,6888)
1736:     write(NOUT1,7145)
1737:     stop 888
1738:   end if
1739:   CASBRN = CASREF
1740: C
1741: C=====
1742: C-----Initialize PDS or other data.
1743: C=====
1744: C
1745: C -----
1746: C .... create input for BURNIN module ...
1747: C -----
1748: C
1749: C Character case of lines including floating point data should be
1750: C upper case because some version of FACOM Fortran produces
1751: C floating output with lower case "e" or "d" before exponent
1752: C for E/D-format but cannot read lower case "e" or "d" float numbers
1753: C created by itself !
1754: C
1755: C call RWIND( NDTLS )

```

```

1756: C
1757: C ... block 3
1758: C
1759: C write(NDTLS,'(5(2x,i5:))') (IBCS(I),I=1,MAXIBC)
1760: C
1761: C ... block 4
1762: C
1763: C do 260 I1 = 1, NEP, 5
1764: C   I2 = MIN(NEP,I1+5-1)
1765: C   CWRK(1:72) = ' '
1766: C   write(CWRK(1:72),'(5(1x,e12.5:))') (POWERL(I),I=I1,I2)
1767: C   call SCTOLC( CWRK, 72 )
1768: C   write(NDTLS,'(a)') CWRK(1:72)
1769: C 260 continue
1770: C
1771: C ... block 5
1772: C
1773: C do 270 I1 = 1, NEP, 5
1774: C   I2 = MIN(NEP,I1+5-1)
1775: C   CWRK(1:72) = ' '
1776: C   write(CWRK(1:72),'(5(1x,e12.5:))') (PERIOD(I),I=I1,I2)
1777: C   call SCTOLC( CWRK, 72 )
1778: C   write(NDTLS,'(a)') CWRK(1:72)
1779: C 270 continue
1780: C
1781: C ... block 6 (reference case of branch mode)
1782: C
1783: C if ( IBCS(4).eq.1 ) then
1784: C   write(NDTLS,'(a)') CASREF
1785: C end if
1786: C
1787: C ... block 7
1788: C
1789: C if( ibcs(4).eq.2 ) then
1790: C   write(ndtls,'(5(2x,i5:))') (jpcflg(i),i=1,nep)
1791: C end if
1792: C
1793: C ... block 8
1794: C
1795: C if ( IBCS(5).ne.0 ) then
1796: C   write(NDTLS,'(a)') STDNUC
1797: C end if
1798: C
1799: C ... block 9-1,9-2,9-3
1800: C
1801: C if ( IBCS(6).ne.0 ) then
1802: C   write(NDTLS,'(i5,2x,i5)') NFIS, NFER
1803: C   do 280 I = 1, NFIS
1804: C     CWRK(1:72) = ' '
1805: C     write(CWRK(1:72),'(a4,2x,i5,2x,e13.6)') NAMFIS(I),
1806: C   &     IFISFLG(I), FISFCT(I)
1807: C     call SCTOLC( CWRK, 72 )
1808: C     write(NDTLS,'(a)') CWRK(1:72)
1809: C 280 continue
1810: C   do 290 I = 1, NFER
1811: C     CWRK(1:72) = ' '
1812: C     write(CWRK(1:72),'(a4,2x,i5,2x,e13.6)') NAMFER(I),
1813: C   &     IFRTFLG(I), FRTFCT(I)
1814: C     call SCTOLC( CWRK, 72 )
1815: C     write(NDTLS,'(a)') CWRK(1:72)
1816: C 290 continue
1817: C   end if
1818: C
1819: C call RWIND( NDTLS )
1820: C

```


src/mvpburn/autobn.f

```

1821:      call CPUTM2( T00 )
1822: C
1823: C=====
1824: C----- Initialize PDS for the first step
1825: C=====
1826: C
1827:      if ( ISTART.eq.1 .or. MODE.eq.'$BRANCH'.or.IBC(8).ne.0 ) then
1828:      call BURNIN( MODE, NOUT1, NOUT2, T00, DIRIN, DIROUT, ARRAY,
1829:      &          MEMORY, IBSTEP, MXSTEP, TRGNAM, IDENT, MAXTRG, MAXISO,
1830:      &          CASEID, IBC, NMAT, KNMAX, NCARD, TITLE, NIN ,MVPDBG,
1831: C2005&          HMINV0, IHMIN0 )
1832:      &          HMINV0, IHMIN0 , TRGPOW , MATPOW, JPCFLG)
1833:      end if
1834: C2005 ... added
1835:      if ( IBEDIT.ge.1 ) then
1836:      write(NOUT1,*) ' '
1837:      write(NOUT1,*) ' *** MODE is ',MODE
1838:      write(NOUT1,*) ' *** ISTART is ',ISTART
1839:      write(NOUT1,*) ' *** ISTOP is ',ISTOP
1840:      end if
1841: C2005 ... end
1842: C
1843: C=====
1844: C----- ordinary Burnup loop
1845: C=====
1846: C
1847:      if ( MODE.ne.'$BRANCH' ) then
1848: C
1849:      do 310 ISTEP = ISTART, ISTOP
1850: C
1851: C----- 2 byte step ID of current step (SID1) and next step (SID2)
1852: C
1853: C      SID2 is ID of an intermediate step for PC method
1854: C
1855:      SID0 = STEPNM(ISTEP,0)
1856:      SID1 = SID0
1857: C
1858: C----- perform MVP & burnup twice for PC method
1859: C
1860:      JPC = JPCFLG(ISTEP)
1861: C
1862: C      .... PC method is impossible for zero power step ...
1863: C
1864:      if ( POWERL(ISTEP).le.0.0 ) JPC = 0
1865: C
1866:      if ( JPC.ne.0 ) then
1867:      NSPC = 2
1868: C
1869: C      .... SID2 is step ID of "istep+1/2"th step here
1870: C
1871:      SID2 = STEPNM(ISTEP,1)
1872: C2003
1873:      if ( ISTEP.eq.ISTOP .and. JHALF.ne.0 ) then
1874:      NSPC = 1
1875:      end if
1876: C2003 ... end
1877:      else
1878:      NSPC = 1
1879:      SID2 = STEPNM(ISTEP+1,0)
1880:      end if
1881: C2003
1882:      NSPC1 = 1
1883:      if ( JPC.ne.0.and.ISTEP.eq.ISTART.and.NHALF.gt.0 ) then
1884:      NSPC1 = 2
1885:      write(NOUT1,*) ' ... calculation start from ',

```

```

1886:      &          ' ISTART+1/2 step.'
1887:      end if
1888: C
1889: C2003      do 300 ISPC = 1, NSPC
1890:      do 300 ISPC = NSPC1, NSPC
1891: C
1892:      CSTEP = STEPNM(ISTEP,0)/// ' '
1893:      if ( ISPC.eq.2 ) then
1894:      SID1 = STEPNM(ISTEP,1)
1895:      SID2 = STEPNM(ISTEP+1,0)
1896:      CSTEP(3:6)= '+1/2'
1897:      end if
1898:      write(NOUT1,6000) CSTEP
1899:      if ( IBEDIT .gt. 1) then
1900:      write(NOUT1,*) ' *** step tag(SID1)=<', SID1, '>'
1901:      write(NOUT1,*) ' *** step tag(SID2)=<', SID2, '>'
1902:      end if
1903: C
1904: C ---- check CPU time ----
1905: C
1906:      call CPUTM2( T00 )
1907:      if ( CPULIM.gt.0.0.and.(T00-T0).gt.CPULIM ) then
1908:      write(NOUT1,6666)
1909:      write(NOUT1,7100) (T00 - T0)/60.0, CPULIM/60.0, ISTEP
1910:      stop 777
1911:      end if
1912: C
1913: C ===== MVP calculation (for non-zero power steps) =====
1914: C
1915:      JMVP = 0
1916: C
1917:      if ( POWERL(ISTEP).gt.0.0 ) then
1918: C2003      if ( POWERL(ISTEP).ne.0.0 ) then
1919:      JMVP = 1
1920: C
1921: C      Create MVP input
1922: C
1923:      call CPUTM2( T00 )
1924:      call MVPIN0( ISTEP, MEMORY, NOUT1, NOUT2, T00, DIRIN,
1925:      &          ARRAY, TRGNAM, IDENT, MAXTRG, MAXISO, CASEID,
1926:      &          SID1, ISPC, JSIN, TITL2, PARANM, PARATP,
1927:      &          PARAV, NMVPPR, MAXSPR,JSVMVP )
1928: C
1929: C----- MVP execution
1930: C
1931: C2003 ... add IBC as argument.
1932:      call MVPRUN( ISTEP, ISPC, DIRIN, DIROUT, CASEID, SID1,
1933:      &          MVPEX, MVPIDX, JSIN, NOUT1, IBC )
1934: C
1935: C      end if
1936: C ===== Depletion cal.=====
1937: C
1938:      call CPUTM2( T00 )
1939:      call BURN0( MEMORY, NOUT1, NOUT2, T00, DIRIN, DIROUT,
1940:      &          ARRAY, TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
1941:      &          CASEID, SID0, SID1, SID2, JPC, ISPC, TITL2 ,
1942: C2005&          CASREF )
1943:      &          CASREF, IBSTEP )
1944: C
1945:      call UDATE( IDATE )
1946:      call JIKAN( STIME )
1947:      write(NOUT1,7160) 'DEPLET', IDATE, STIME
1948:      call CPUTM2( TCPU )
1949:      write(NOUT1,7120) TCPU/60.0
1950: C

```

src/mvpburn/autobn.f

```

1951: C-----delete MVP outputs for internal step for PC method (if JPCFLG<0)
1952: C
1953:         if (ISPC.eq.2) then
1954: CMod         if (MVPDBG.eq.0) then
1955:         if ( JPCFLG(ISTEP).gt.0 ) then
1956:             ITEMP = 0
1957:             MEMBER = CASEID// 'HT'//SID1
1958:             call PDSIO('DELETE', DIROUT, MEMBER, DUMMY, 1,
1959: &                 1, NOUT1 )
1960:         else
1961:             ITEMP = JSVMVP(ISTEP)
1962:         end if
1963:         call MVPDEL( DIROUT, CASEID, SID1, ITEMP,NOUT1)
1964:         end if
1965: C
1966: C-----delete MVP outputs ---
1967: C
1968:         if ( ISPC.eq.1 .and. JMVP.ne.0 ) then
1969:             call MVPDEL( DIROUT, CASEID, SID1, JSVMVP(ISTEP),
1970: &                 NOUT1 )
1971:         end if
1972: 300         continue
1973: C
1974: 310         continue
1975: C
1976: C **** output sumary by print option ****
1977: C
1978:         call CPUTM2( T00 )
1979:         if (IBEDIT .gt. 0) then
1980:             NOTSTP = -1
1981:             do 400 I = 1,MXSTEP
1982:                 IOTSTP(I) = I
1983: 400             continue
1984:             call SUMRY0(MEMORY, NOUT1, NOUT2, T00, DIROUT, ARRAY
1985: &                 TRGNAM, IDENT, MAXTRG, MAXISO, CASEID, NOTSTP, IOTSTP )
1986:         endif
1987: C
1988: C=====
1989: C----- branch-off Burnup loop
1990: C=====
1991: C
1992:         else
1993: C
1994:             do 320 ISTEP = ISTART, ISTOP
1995: C
1996: C----- 2 byte step ID of current step (SID1) and next step (SID2)
1997: C
1998: C ( In branch mode, PC is invalid )
1999: C
2000: C2005         if ( IBSTEP(ISTEP).ne.0 ) then
2001:             if ( IBSTEP(ISTEP).eq.1 ) then
2002: C
2003:                 write(NOUT1,*)
2004: &                 ' == START BRANCH-OFF CALCULATION == STEP ',ISTEP
2005: C
2006:                 NOWSTP = ISTEP
2007: C
2008:                 SID0 = STEPNM(ISTEP,0)
2009:                 SID1 = SID0
2010:                 SID2 = STEPNM(ISTEP+1,0)
2011:                 CSTEP = SID0// ' '
2012: C
2013:                 JPC = 0
2014:                 ISPC = 1
2015:                 write(NOUT1,6000) CSTEP

```

```

2016: C
2017: C ---- check CPU time ----
2018: C
2019:         call CPUTM2( T00 )
2020:         if ( CPULIM.gt.0.0.and.(T00-T0).gt.CPULIM ) then
2021:             write(NOUT1,6666)
2022:             write(NOUT1,7100) (T00 - T0)/60.0, CPULIM/60.0, ISTEP
2023:             stop 777
2024:         end if
2025: C
2026: C----- Create PDS member case+'HT'+step
2027: C
2028:         LOC1 = 1
2029: C
2030:         ... HMINV ...
2031:         LOC2 = LOC1 + NMAT
2032:         ... EXPSZN ...
2033:         LOC3 = LOC2 + NMAT
2034:         ... DMWZON ...
2035:         LOC4 = LOC3 + NMAT
2036:         ... DNBURN ...
2037:         LOC5 = LOC4 + NTNUC*NMAT
2038:
2039:         MEM0 = MEMORY - LOC5 + 1
2040:
2041:         call COPYHT( ISTEP, CASREF, CASEID, NOUT1, NOUT2, DIRIN,
2042: &                 DIROUT, ARRAY(LOC1), ARRAY(LOC2), ARRAY(LOC3),
2043: &                 ARRAY(LOC4), MEM0, ARRAY(LOC5), ARRAY(LOC5) )
2044: C----- Create MVP input
2045: C
2046:         call CPUTM2( T00 )
2047:         call MVPIN0( ISTEP, MEMORY, NOUT1, NOUT2, T00, DIRIN,
2048: &                 ARRAY, TRGNAM, IDENT, MAXTRG, MAXISO, CASEID,
2049: &                 SID1, ISPC, JSIN, TITL2, PARANM, PARATP, PARAV,
2050: &                 NMVPPR, MAXSPR, JSVMVP )
2051: C
2052: C----- MVP execution
2053: C
2054: C2003 ... add IBC as argument.
2055:         call MVPRUN( ISTEP, ISPC, DIRIN, DIROUT, CASEID, SID1,
2056: &                 MVPEX, MVPIDX, JSIN, NOUT1, IBC )
2057: C
2058: C----- BURNUP
2059: C
2060:         call CPUTM2( T00 )
2061:         call BURN0( MEMORY, NOUT1, NOUT2, T00, DIRIN, DIROUT,
2062: &                 ARRAY, TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
2063: &                 CASEID, SID0, SID1, SID2, JPC, ISPC, TITL2 ,
2064: C2005&                 CASREF )
2065:         &                 CASREF, IBSTEP )
2066: C
2067:         call UDATE( IDATE )
2068:         call JIKAN( STIME )
2069:         write(NOUT1,7160) 'DEPLET', IDATE, STIME
2070:         call CPUTM2( TCPU )
2071:         write(NOUT1,7120) TCPU/60.0
2072: C
2073: C-----delete MVP outputs ---
2074: C
2075:         call MVPDEL( DIROUT, CASEID, SID1, JSVMVP(ISTEP), NOUT1 )
2076:         end if
2077: C
2078: 320         continue
2079: C
2080: C **** output sumary by print option ****

```

src/mvpburn/autobn.f

```
2081: C
2082:      call CPUTM2( T00 )
2083: C
2084:      if (IBEDIT.gt. 0) then
2085:          NOTSTP = 0
2086:          do 410 ISTEP = ISTART,ISTOP
2087:              if ( IBSTEP(ISTEP).ne.0 ) then
2088:                  NOTSTP = NOTSTP + 1
2089:                  IOTSTP(NOTSTP) = ISTEP
2090:              end if
2091:          410 continue
2092: C
2093:      call SUMRY0(MEMORY, NOUT1, NOUT2, T00, DIROUT, ARRAY,
2094: &      TRGNAM, IDENT, MAXTRG, MAXISO, CASEID, NOTSTP, IOTSTP )
2095:      endif
2096: C
2097:      end if
2098: C
2099:      return
2100: C
2101: C ----- Error -----
2102: C
2103:      330 write(NOUT1,6888)
2104:      write(NOUT1,7300) VNAME(:NLEN)
2105:      stop 888
2106: C
2107: C=====
2108:      6000 format(/1X,'xxxxxxxxxxxxxxxxxxxxxxxx',
2109: &      /1X,'x CURRENT-STEP: ',A6,' x'
2110: &      /1X,'xxxxxxxxxxxxxxxxxxxxxxxx' / )
2111:      6666 format(/' !!!(AUTOBN) WARNING : ' )
2112:      6888 format(/' XXX(AUTOBN) ERROR-STOP : ' )
2113:      7000      format(1X,'data "',A,'" have meanings only in'
2114: &      '$BRANCH mode.'/)
2115:      7010 format(1X,'integrated burnup is not ascending ',
2116: &      'order.'/ BURNUP STEP=',I5)
2117:      7020 format(1X,'data "',A,'" have no meaning in',
2118: &      '$BRANCH mode.'/)
2119:      7100 format( ' used CPU time is ',1P,E12.5,' min. and',
2120: &      ' it exceeds input limit (',1P,E12.5,')'/1X,2X,
2121: &      'stop calculation before step ',I3)
2122:      7120 format(/2X,'CUMULATIVE CPU TIME ',1P,E12.5,' MIN.'/)
2123:      7145 format( ' fatal ERRORS have been detected',
2124: &      ' in input-phase'/
2125: &      ' message for ERRORS is printed ',
2126: &      'with "XXX" symbol on the head of printed line'/
2127: &      ' check your input data carefully')
2128:      7160 format(/2X,'==== ',A,' MODE ENDED DATE : ',A,' AT TIME=',A)
2129: C7180 format(1X,'number of data <',A,
2130: C      &      '> does not match number of steps determined by other input.')
2131:      7200 format(1X,'unknown data name <',A,
2132: &      '>. Skipping ...')
2133:      7240 format(1X,'string data <',A,'> has no closing ")" ?'/1X,
2134: &      'or quotation mark is missing.')
2135: C
2136:      7260 format(1X,'XXX(AUTOBN) ERROR: number of data <',A,
2137: &      '> does not match number of steps determined by other input.')
2138: C
2139:      7280 format(1X,A)
2140:      7300 format( ' end of input data in reading data <',A,
2141: &      '> in $BURNUP block.')
2142: C
2143:      end
```

src/mvpburn/averg0.f

```

1:      subroutine AVERG0( MEMORY,NOUT1, NOUT2, T00,  DIROUT,CASEID,
2:      &                  DIRINS, CASEIN, NCASE, IDBG, MODEAV, JSAVE,
3:      &                  A,      IA,
4:      &                  TRGNAM, IDENT, MAXTRG, MAXISO,
5:      &                  NIN )
6: C=====
7: C Averaging if cases CASEIN(*) on PDS DIRINS(*).
8: C
9: C JSAVE > 0: create PDS members for the averaged case.
10: C=====
11:      common /MEMOCK/  MEUSED
12:      character*(*) DIROUT
13:      character*4 CASEID
14:      character*72 DIRINS(NCASE)
15:      character*4 CASEIN(NCASE)
16:      character*72 TITLE0(2)
17: C
18:      real A(MEMORY)
19:      integer IA(MEMORY)
20:      character*12 TRGNAM(MAXTRG)
21: C2003
22:      character*12 IDENT(MAXISO,MAXTRG)
23: C
24:      include 'INC/_BURNP'
25: C
26:      real INSCR, INTCR
27: C
28:      include 'INC/_CBURN1'
29:      include 'INC/_CBURN2'
30:      include 'INC/_CBURN3'
31: C
32:      common /BNREST/  RSDUMY(3),      NOWSTP, IBEND, AKEFF(MXSTEP),
33:      &      ERKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
34:      &      EXPST(MXSTEP),  CUMMWD(MXSTEP), INSCR(MXSTEP),
35:      &      INTCR(MXSTEP),  EINSR(MXSTEP), EINTCR(MXSTEP),
36:      &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
37:      &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
38:      &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
39:      &      NBATCH(MXSTEP)
40: C2005
41:      &      ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
42:      &      RMONIT(MXSTEP), EMONIT(MXSTEP)
43: Cend
44: C
45: C ----- Local data -----
46: C
47: C2005 character*72 DDNAME
48: C
49:      character*8 MEMBER
50: C
51: C2005 integer IDUM(10)
52: C
53: C *****
54: C
55: C ... save title input as title of "averaged" case.
56: C
57:      TITLE0(1) = TITLE(1)
58:      TITLE0(2) = TITLE(2)
59: C
60: C ... added May 1998
61: C
62: C2003
63: C Averaged weight is used in all case (from 14 Feb 2003)
64: C MODEAV=0 : average weighted by K-eff error.
65: C MODEAV=1 : use errors from MVP for calculation of

```

```

66: C errors of k-eff and micro rates.
67: C
68:      write(NOUT1,7000) CASEID, TITLE0
69:      write(NOUT1,7010) NCASE, IDBG, MODEAV
70: C
71: 7000 format(/15X,'xxxxxxxxxxxxxxxxxxxxxx'
72:      &      /15X,'x AVERAGE-MODE x'
73:      &      /15X,'xxxxxxxxxxxxxxxxxxxxxx'
74:      &      /1X,'CASEID:',A4
75:      &      /1X,'TITLE :',A72
76:      &      /1X,'      :',A72/ )
77: 7010 format(
78:      &      /6X,'NCASE : NUMBER OF CASES TO BE AVERAGED ..',
79:      &      '.....',I2
80:      &      /6X,'IDBG : DEBUG OPTION (0/1=NO/YES) .....',
81:      &      '.....',I2
82:      &      /6X,'MODEAV : AVERAGING MODE .....',
83:      &      '.....',I2//)
84: C
85:      if ( NCASE.le.1 ) then
86:      write(NOUT1,6888)
87:      write(NOUT1,*) ' input error at averaging mode'
88:      write(NOUT1,*) ' number of cases to averaged is ',NCASE
89:      write(NOUT1,*) ' this run has no meaning'
90:      stop 888
91:      end if
92: C2005 ... added
93: C
94: C ..... Check IBC(4) & IBC(9) option
95: C
96:      call RWCOM1('READ',DIRINS(1),CASEIN(1))
97:      KVOID = IBC(4)
98:      KBNTYP = IBC(9)
99:      IERR = 0
100: C
101:      do 77 LOP = 2, NCASE
102:      call RWCOM1('READ',DIRINS(LOP),CASEIN(LOP))
103:      LVOID = IBC(4)
104:      LBNTYP = IBC(9)
105:      if ( KVOID .ne.LVOID ) IERR = IERR + 1
106:      if ( KBNTYP.ne.LBNTYP ) IERR = IERR + 1
107: 77 continue
108: C
109:      if ( IERR.gt.0 ) then
110:      write(NOUT1,6888)
111:      write(NOUT1,*) ' input error at averaging mode'
112:      write(NOUT1,*) ' disagreement was found ',
113:      &      'in calculation mode or BNPTYP !!! '
114:      write(NOUT1,*) ' please check the calculation ',
115:      &      'condition of each CASE !!! '
116:      write(NOUT1,*) ' this run will be stopped. '
117:      stop 888
118:      end if
119: C2005 ... added
120: C
121: C
122: C*****
123: C Averaging stage 1: (subroutine AVERG1)
124: C * read data of each case and store them on working files
125: C (I/O unit 40 &41)
126: C
127: C * Output sumary of each case.
128: C*****
129: C
130:      NEPMIN = MXSTEP + 1

```

src/mvpburn/averg0.f

```

131:      NEPMAX = 0
132:      IBEDIT = IDBG
133: C
134:      call RWIND( 40 )
135: C2003 if( MODEAV.eq.2 ) call RWIND(41)
136: C2003 if( MODEAV.eq.0 ) call RWIND(41)
137:      call RWIND(41)
138: C
139:      do 100 LOP = 1, NCASE
140: C
141: C---- READ PDS
142: C
143:      call RWCOM1('READ',DIRINS(LOP),CASEIN(LOP))
144:      call RWCOM2('READ',DIRINS(LOP),CASEIN(LOP))
145:      call RWCOM3('READ',DIRINS(LOP),CASEIN(LOP))
146: C      call RWREST('READ',DIRINS(LOP),CASEIN(LOP))
147: C
148: C      NEP1 = NOWSTP
149: C      NEP1 = NEP+1
150: C      NEP = NEP1 - 1
151: CCCCCC KNMAX = NTNUC
152: C
153: C      if ( NEPMAX.lt.NEP1 ) NEPMAX = NEP1
154: C      if ( NEPMIN.gt.NEP1 ) NEPMIN = NEP1
155: C
156: C *** SET VAIRABLE DIMENSION
157: C
158:      LOC01 = 1
159: C
160: C --- MTPY
161:      LOC02 = LOC01 + NMAT
162: C --- NISO
163:      LOC03 = LOC02 + NMAT
164: C --- TEMP
165:      LOC04 = LOC03 + NMAT
166: C --- VOLM
167:      LOC05 = LOC04 + NMAT
168: C --- TRGNAM
169: CCCC LOC06 = LOC05 + NMAT*3
170: C2005 LOC06 = LOC05
171: C --- MATREG
172:      LOC06 = LOC05 + NMAT
173: Cend
174: C --- LENTRG
175:      LOC07 = LOC06 + NMAT
176: C --- MTBURN
177:      LOC08 = LOC07 + NMAT*3
178: C --- MTCARD
179:      LOC09 = LOC08 + NMAT*2
180: C --- IDENT
181: CCCC LOC10 = LOC09 + KNMAX*NMAT*2
182:      LOC10 = LOC09
183: C --- DNINIT
184:      LOC11 = LOC10 + KNMAX*NMAT
185: C --- MATMVP
186:      LOC12 = LOC11 + NMAT
187: C --- IPBURN
188:      LOC13 = LOC12 + KNMAX*NMAT
189: C --- MVPDAT
190: CCCCC LOC14 = LOC13 + NCARD*18
191:      LOC14 = LOC13
192: C --- NUCLP
193:      LOC15 = LOC14 + NPAR*NTNUC
194: C --- GAM
195:      LOC16 = LOC15 + NTFISS*NTNUC

```

```

196: C --- NBIC
197:      LOC17 = LOC16 + NPAR*NTNUC
198: C --- PBIC
199: C ***** LOCATION FOR CASEBNU *****
200:      LOC18 = LOC17 + NPAR*NTNUC
201: C --- POWRZN
202:      LOC19 = LOC18 + NMAT*NEP1
203: C --- EXPSZN
204:      LOC20 = LOC19 + NMAT*NEP1
205: C --- HMINV
206:      LOC21 = LOC20 + NMAT*NEP1
207: C --- DMWZON
208:      LOC22 = LOC21 + NMAT*NEP1
209: C --- GAMAVG
210:      LOC23 = LOC22 + NMAT*NEP1
211: C --- YLDXE5
212:      LOC24 = LOC23 + NMAT*NEP1
213: C --- YLDI35
214:      LOC25 = LOC24 + NMAT*NEP1
215: C --- YLDSM9
216:      LOC26 = LOC25 + NMAT*NEP1
217: C --- YLDPM9
218:      LOC27 = LOC26 + NMAT*NEP1
219: C --- DENTAB
220:      LOC28 = LOC27 + NTNUC*NMAT*NEP1
221: C --- RATMIC
222: C2003 LAST = LOC28 + 2*4*NTNUC*NMAT*NEP1
223:      LOC29 = LOC28 + 2*4*NTNUC*NMAT*NEP1
224: C2003 added
225: C --- NHIST0
226:      LAST = LOC29 + NEP1
227: C -----
228:      if (LAST.gt.MEUSED) MEUSED = LAST
229:      LEMORY = MEMORY - LAST + 1
230: C
231:      if ( LEMORY.lt.0 ) then
232:        write(NOUT1,6888)
233:        write(NOUT1,*) ' memory over'
234:        write(NOUT1,*) ' requested memory size = ', LAST,
235:          & ' words.'
236:        write(NOUT1,*) ' reserved memory size = ', MEMORY,
237:          & ' words.'
238:        write(NOUT1,*) ' change parameter value in include file'
239:        stop 999
240:      end if
241: C
242:      if ( NMAT.gt.MAXTRG ) then
243:        write(NOUT1,6888)
244:        write(NOUT1,*) ' too many tally regions'
245:        write(NOUT1,*) ' requested number of materials = ',NMAT
246:        write(NOUT1,*) ' limit for number of tally regions',
247:          & ' (MAXTRG)=',MAXTRG
248:        write(NOUT1,*) ' change parameter value in include file'
249:        stop 999
250:      end if
251: C
252:      if ( KNMAX.gt.MAXISO ) then
253:        write(NOUT1,6888)
254:        write(NOUT1,*) ' too many nuclides per tally region'
255:        write(NOUT1,*) ' requested number of nuclides = ',KNMAX
256:        write(NOUT1,*) ' limit for number of nuclides per ',
257:          & 'tally region(MAXISO)=',MAXISO
258:        write(NOUT1,*) ' change parameter value in include file'
259:        stop 999
260:      end if

```

src/mvpburn/averg0.f

```

261: C
262: C2003    call CLEA( A, LAST, 0.0 )
263: C    ... do not clear NHIST0
264:         call CLEA( A, LOC29-1, 0.0 )
265: C2003 ... end
266:
267: C
268: C2003 NHIST0() (LOC29) are added
269: C2005 MATREG() (LOC05) are added
270:         if ( LOP.eq.1 ) JHIST0 = 0
271:         call AVERG1( NOUT1, NOUT2, MODEAV, LEMORY,
272: &         DIRINS(LOP), CASEIN(LOP), LOP, JHIST0,
273: &         A(LOC01), A(LOC02), A(LOC03), A(LOC04), TRGNAM ,
274: &         A(LOC06), A(LOC07), A(LOC08), IDENT , A(LOC10),
275: &         A(LOC11), A(LOC12), , A(LOC14), A(LOC15),
276: &         A(LOC16), A(LOC17), A(LOC18), A(LOC19), A(LOC20),
277: &         A(LOC21), A(LOC22), A(LOC23), A(LOC24), A(LOC25),
278: &         A(LOC26), A(LOC27), A(LOC28), A(LOC29),
279: C2005&         A(LAST), A(LAST) )
280: &         A(LAST), A(LAST) , A(LOC05) )
281: C2003 ... end
282: C
283:         if ( NEPMAX.lt.NOWSTP ) NEPMAX = NOWSTP
284:         if ( NEPMIN.gt.NOWSTP ) NEPMIN = NOWSTP
285: 100 continue
286: C
287:         call RWIND( 40 )
288:         call RWIND( 41 )
289: C
290: C*****
291: C Averaging stage 2: (subroutine AVERG2)
292: C * Avaraging using contents of working files (I/O unit 40 &41)
293: C * Output sumary of averaged case.
294: C*****
295: C
296: C---- OPEN PDS
297: C
298: C
299: MEMBER = 'OPENING'
300: call PDSIO( 'OPEN', DIROUT, MEMBER, MEMBER, A, 2, NOUT1 )
301: C
302: C
303: NEP1 = NEPMAX
304: NEP = NEP1 - 1
305: NOWSTP = NEPMIN
306: C
307: C *** SET VAIRABLE DIMENSION
308: C
309: LOC01 = 1
310: C
311: C --- MTyp
312: LOC02 = LOC01 + NMAT
313: C --- NISO
314: LOC03 = LOC02 + NMAT
315: C --- TEMP
316: LOC04 = LOC03 + NMAT
317: C --- VOLM
318: LOC05 = LOC04 + NMAT
319: C --- TRGNAM
320: CCCC LOC06 = LOC05 + NMAT*3
321: C2005 LOC06 = LOC05
322: C --- MATREG
323: LOC06 = LOC05 + NMAT
324: Cend
325: C --- LENTRG

```

```

326: LOC07 = LOC06 + NMAT
327: C --- MTBURN
328: LOC08 = LOC07 + NMAT*3
329: C --- MTCARD
330: LOC09 = LOC08 + NMAT*2
331: C --- IDENT
332: CCCC LOC10 = LOC09 + KNMAX*NMAT*2
333: LOC10 = LOC09
334: C --- DNINIT
335: LOC11 = LOC10 + KNMAX*NMAT
336: C --- MATMVP
337: LOC12 = LOC11 + NMAT
338: C --- IPBURN
339: LOC13 = LOC12 + KNMAX*NMAT
340: C --- MVPDAT
341: CCCC LOC14 = LOC13 + NCARD*18
342: LOC14 = LOC13
343: C --- NUCLP
344: LOC15 = LOC14 + NPAR*NTNUC
345: C --- GAM
346: LOC16 = LOC15 + NTFISS*NTNUC
347: C --- NBIC
348: LOC17 = LOC16 + NPAR*NTNUC
349: C --- PBIC
350: C ***** LOCATION FOR CASEBNUP *****
351: LOC18 = LOC17 + NPAR*NTNUC
352: C --- POWRZN
353: LOC19 = LOC18 + NMAT*NEP1
354: C --- EXPSZN
355: LOC20 = LOC19 + NMAT*NEP1
356: C --- HMINV
357: LOC21 = LOC20 + NMAT*NEP1
358: C --- DMWZON
359: LOC22 = LOC21 + NMAT*NEP1
360: C --- GAMAVG
361: LOC23 = LOC22 + NMAT*NEP1
362: C --- YLDXE5
363: LOC24 = LOC23 + NMAT*NEP1
364: C --- YLDI35
365: LOC25 = LOC24 + NMAT*NEP1
366: C --- YLDSM9
367: LOC26 = LOC25 + NMAT*NEP1
368: C --- YLDPM9
369: LOC27 = LOC26 + NMAT*NEP1
370: C --- DENTAB
371: LOC28 = LOC27 + NTNUC*NMAT*NEP1
372: C --- RATMIC
373: LOC29 = LOC28 + 2*4*NTNUC*NMAT*NEP1
374: C --- DENWRK
375: LOC30 = LOC29 + NTNUC*NMAT*NEP1
376: C --- RATWRK
377: LOC31 = LOC30 + 2*4*NTNUC*NMAT*NEP1
378: C --- RWORK
379: LOC32 = LOC31 + NEP1
380: C --- IWORK
381: LOC33 = LOC32 + NEP1
382: C --- D2WORK
383: C2003 LAVF1 = LOC33 + NEP1*NMAT
384: LOC34 = LOC33 + NEP1*NMAT
385: C2003 .. added
386: C --- DENERR
387: C2005 ..... added
388: Cmod LAVF1 = LOC34 + NTNUC*NMAT*NEP1
389: C --- RWORK2
390: LOC35 = LOC34 + NTNUC*NMAT*NEP1

```

src/mvpburn/averg0.f

```
391: C --- RWORK3
392:      LOC36  = LOC35 + NEP1
393: C --- RWORK4
394:      LOC37  = LOC36 + NEP1
395: C ---LAVF1
396:      LAVF1   = LOC37 + NEP1
397: C2005 ..... end
398: C --- averaging factor 1 & 2 (1/sigma**2 or NHIST)
399: C2003 if( MODEAV.eq.2 ) then
400: C2003 if( MODEAV.eq.0 ) then
401:      LAVF2   = LAVF1 + NEP1*NCASE
402:      LAST    = LAVF2 + NEP1*NCASE
403: C2003 else
404: C2003      LAVF2 = LAVF1
405: C2003      LAST = LAVF2
406: C2003 end if
407: C -----
408:      if (LAST.gt.MEUSED) MEUSED = LAST
409:      LEMORY = MEMORY - LAST + 1
410: C
411:      if ( LEMORY.lt.0 ) then
412:      write(NOUT1,6888)
413:      write(NOUT1,*) ' memory over'
414:      write(NOUT1,*) ' requested memory size = ', LAST, ' words.'
415:      write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
416:      write(NOUT1,*) ' change parameter value in include file'
417:      stop 999
418:      end if
419: C
420:      call CLEA( A, LAST, 0.0 )
421: C
422: C2003 added DENERR (LOC34)
423: C2005 MATREG(NMAT) (LOC05) are added
424: C2005 RWORK2(NEP1) (LOC35) are added
425: C2005 RWORK3(NEP1) (LOC36) are added
426: C2005 RWORK4(NEP1) (LOC37) are added
427:      call AVERG2( NOUT1, NOUT2, MODEAV, JSAVE, LEMORY,
428:      &          DIROUT, CASEID, TITLE0, NCASE,
429:      &          A(LOC01), A(LOC02), A(LOC03), A(LOC04), TRGNAM ,
430:      &          A(LOC06), A(LOC07), A(LOC08), IDENT , A(LOC10),
431:      &          A(LOC11), A(LOC12), , A(LOC14), A(LOC15),
432:      &          A(LOC16), A(LOC17), A(LOC18), A(LOC19), A(LOC20),
433:      &          A(LOC21), A(LOC22), A(LOC23), A(LOC24), A(LOC25),
434:      &          A(LOC26), A(LOC27), A(LOC28), A(LOC29), A(LOC30),
435:      &          A(LOC31), A(LOC32), A(LOC33), A(LOC34),
436: C2005&          A(LAVF1), A(LAVF2), A(LAST) , A(LAST) )
437:      &          A(LAVF1), A(LAVF2), A(LAST) , A(LAST) ,
438:      &          A(LOC05), A(LOC35), A(LOC36), A(LOC37) )
439: C2003 ... end
440: C
441:      call CPUTM2( T1 )
442:      if (IBEDIT .gt. 0) then
443:      write(NOUT1,7040) T1 - T00
444:      end if
445: C
446: 6888 format(// ' XXX(AVERG0) ERROR-STOP : ' )
447: 7040 format(/1X,12X,'CPU ',1P,E12.5,' SEC.')
448: C
449: C *** END OF PROCESS
450: C
451:      return
452:      end
```

src/mvpburn/averg1.f

```

1: C2003 LOP0, JHIST0 and NHIST0 are added
2:   subroutine AVERG1( NOUT1, NOUT2, MODEAV, MEMORY, DIRINS, CASEID,
3:   &                  LOP0, JHIST0,
4:   &                  MTYP,
5:   &                  NISO, TEMP, VOLM, TRGNAM, LENTRG, MTBURN,
6:   &                  MTCARD, IDENT, DNINIT, MATMVP, IPBURN, NUCLP, GAM,
7:   &                  NBIC, PBIC, POWRZN, EXPSZN, HMINV, DMWZON,
8:   &                  GAMAVG, YLDXE5, YLDI35, YLDSM9, YLDPM9, DENTAB,
9: C2005&                  RATMIC, NHIST0, IARRAY, RARRAY )
10:  &                  RATMIC, NHIST0, IARRAY, RARRAY , MATREG )
11: C=====
12: C  Prepare data used in averaging over cases: extract data from a
13: C  results of a case in PDS DIRINS.
14: C=====
15:   character*4 CASEID
16:   character*72 DIRINS
17:   integer IARRAY(MEMORY)
18:   real RARRAY(MEMORY)
19: C2003
20:   character*12 IDENT(KNMAX, NMAT)
21:   character*12 TRGNAM(NMAT)
22: C
23:   integer MTYP(NMAT), NISO(NMAT), LENTRG(NMAT)
24:   integer IPBURN(KNMAX, NMAT), MATMVP(NMAT)
25:   integer MTBURN(3, NMAT), MTCARD(2, NMAT)
26: C
27:   real TEMP(NMAT), VOLM(NMAT)
28:   real DNINIT(KNMAX, NMAT)
29: C
30:   integer NUCLP(NPAR, NTNUC)
31:   real GAM(NTFISS, NTNUC)
32:   integer NBIC(NPAR, NTNUC)
33:   real PBIC(NPAR, NTNUC)
34: C
35:   real POWRZN(NMAT, NEP1)
36:   real EXPSZN(NMAT, NEP1)
37:   real HMINV(NMAT, NEP1)
38:   real DMWZON(NMAT, NEP1)
39:   real YLDXE5(NMAT, NEP1)
40:   real YLDI35(NMAT, NEP1)
41:   real YLDSM9(NMAT, NEP1)
42:   real YLDPM9(NMAT, NEP1)
43:   real GAMAVG(NMAT, NEP1)
44: C
45:   real DENTAB(NTNUC, NMAT, NEP1)
46:   real RATMIC(2, 4, NTNUC, NMAT, NEP1)
47: C2003
48:   integer NHIST0(NEP1)
49: C2005
50:   integer MATREG(NMAT)
51: Cend
52: C
53:   include 'INC/_BURNP'
54: C
55:   include 'INC/_CBURN1'
56:   include 'INC/_CBURN2'
57:   include 'INC/_CBURN3'
58: C
59:   real INSCR, INTCR
60:   common /MEMOCK/ MEUSED, MEMMAX
61:   common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
62:   &   ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
63:   &   EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
64:   &   INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
65:   &   FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),

```

```

66:   &   FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
67:   &   FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
68:   &   NBATCH(MXSTEP)
69: C2005
70:   &   ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
71:   &   RMONIT(MXSTEP), EMONIT(MXSTEP)
72: Cend
73: C
74:   character*2 STEPNM
75:   external STEPNM
76: C
77: C .... local variables .....
78: C
79:   character*8 MEMBER
80:   character*72 MLINE
81: C
82: C -----
83: C
84:   call RWMATD( 'READ', DIRINS, CASEID, NMAT, KNMAX, NISO(1),
85:   &   TEMP(1), VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1),
86:   &   MTCARD(1,1), IDENT(1,1), DNINIT(1,1), MATMVP(1),
87: C2005&   IPBURN(1,1) )
88:   &   IPBURN(1,1), MATREG(1) )
89: Cend
90:   do 100 M = 1, NMAT
91:     MTYP(M) = MTBURN(3,M)
92:   100 continue
93: C
94:   call RWCHAN( 'READ', DIRINS, CASEID, NPAR, NTNUC, NTFISS,
95:   &   NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
96: C
97:   MEMBER = 'OPENING'
98:   call PDSIO( 'OPEN', DIRINS, MEMBER, MEMBER, A, 1, NOUT1 )
99: C
100:   IWRT = 0
101:   do 120 LOP = 1, NOWSTP
102:     do 120 LOP = 1, NEP1
103:   103: C
104:     MEMBER = CASEID// 'HT' // STEPNM(LOP,0)
105:   105: C
106:     call PDSIO( 'EXIST', DIRINS, MEMBER, MEMBER, dummy, iflag, NOUT1 )
107:     if ( iflag.lt.0 ) then
108:       if ( IWRT.le.0 ) write(NOUT1,6000)
109:       write(NOUT1,6100) LOP, MEMBER
110:       IWRT = IWRT + 1
111:       go to 125
112:     end if
113:     NOWSTP = LOP
114: C2003 ... check member size
115:     call PDSIO( 'SEARCH', DIRINS, MEMBER, MEMBER, RARRAY(1), LENGHT,
116:   116:   &   NOUT1 )
117:     if (LENGHT.gt.MEMORY) then
118:       write(NOUT1,*) 'XXX(AVERG1) Size of PDS member ', MEMBER,
119:   119:   &   ' ( =', LENGHT, ' ) exceeds buffer size ( =', MEMORY, ' )'
120:     stop 999
121:   else
122:     MTMP = MEMMAX - MEMORY + LENGHT
123:     if (MTMP.gt.MEUSED) MEUSED = MTMP
124:   end if
125: C2003 ... end
126:   call PDSIO( 'READ', DIRINS, MEMBER, MEMBER, RARRAY(1), LENGHT,
127:   &   NOUT1 )
128: c
129: CKSK not case lenght < 0 in read mode
130: CKSK   if ( LENGHT.lt.0 ) then

```


src/mvpburn/averg1.f

```

131: CKSK      If ( IWRT.le.0) write(NOUT1,6000)
132: CKSK      write(NOUT1,*) '!!!(AVERG1) Step ', LOP, ' of case ',
133: CKSK &      CASEID, ' has no member CASE+"HT"+step.'
134: CKSK      IWRT = IWRT + 1
135: CKSK      go to 120
136: CKSK      end if
137: C
138:      call PCTRW( IARRAY, MEMBER, IRET )
139: C
140:      call UNPKND( IARRAY, 'STEP#', 'I4', ISWSTP, 1, NDATA2, IRET )
141: C
142:      call PCTLB( IARRAY, MEMBER, 'HMINV', IRET )
143:      call UNPKND( IARRAY, 'HMINV', 'R4', HMINV(1,LOP), NMAT, NDATA2,
144: &      IRET )
145: C
146:      call PCTLB( IARRAY, MEMBER, 'EXPSZN', IRET )
147:      call UNPKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1,LOP), NMAT,
148: &      NDATA2, IRET )
149: C
150:      call PCTLB( IARRAY, MEMBER, 'DMWZON', IRET )
151:      call UNPKND( IARRAY, 'DMWZON', 'R4', DMWZON(1,LOP), NMAT,
152: &      NDATA2, IRET )
153: C
154:      call PCTLB( IARRAY, MEMBER, 'DNBURN', IRET )
155:      call UNPKND( IARRAY, 'DNBURN', 'R4', DENTAB(1,1,LOP), NTNVC*
156: &      NMAT, NDATA2, IRET )
157: C
158:      call PCTLB2( IARRAY, MEMBER, 'INTEGER', IRET )
159:      if ( IRET.eq.0 ) then
160:      call UNPKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
161:      NHIST(LOP) = IARRAY(LP)
162:      NBATCH(LOP) = IARRAY(LP+1)
163:      end if
164: C
165:      call PCTLB2( IARRAY, MEMBER, 'REAL', IRET )
166:      if ( IRET.eq.0 ) then
167:      call UNPKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
168:      AKEFF(LOP) = RARRAY(LP)
169:      ERRKEF(LOP) = RARRAY(LP+1)
170:      DAYS(LOP) = RARRAY(LP+2)
171:      U235F(LOP) = RARRAY(LP+3)
172:      EXPST(LOP) = RARRAY(LP+4)
173:      CUMMWD(LOP) = RARRAY(LP+5)
174:      INSCR(LOP) = RARRAY(LP+6)
175:      INTCR(LOP) = RARRAY(LP+7)
176:      EINSR(LOP) = RARRAY(LP+8)
177:      EINTCR(LOP) = RARRAY(LP+9)
178:      FLXNRM(LOP) = RARRAY(LP+10)
179:      FACNRM(LOP) = RARRAY(LP+11)
180:      FISABS(LOP) = RARRAY(LP+12)
181:      FRTPAP(LOP) = RARRAY(LP+13)
182:      EFISAB(LOP) = RARRAY(LP+14)
183:      EFRTPA(LOP) = RARRAY(LP+15)
184:      FDECAY(LOP) = RARRAY(LP+16)
185:      CDECAY(LOP) = RARRAY(LP+17)
186: C2005 ... added
187:      ERRNRM(LOP) = RARRAY(LP+18)
188:      POWERW(LOP) = RARRAY(LP+19)
189:      POWERE(LOP) = RARRAY(LP+20)
190:      RMONIT(LOP) = RARRAY(LP+21)
191:      EMONIT(LOP) = RARRAY(LP+22)
192: C2005 ... end
193:      end if
194: C
195:      call PCTLB2( IARRAY, MEMBER, 'POWRZN', IRET )

```

```

196:      if ( IRET.eq.0 ) then
197:      call UNPKND( IARRAY, 'POWRZN', 'R4', POWRZN(1,LOP), NMAT,
198: &      NDATA2, IRET )
199:      else
200:      If ( IWRT.le.0) write(NOUT1,6000)
201:      write(NOUT1,6200) LOP, MEMBER, 'POWRZN'
202:      IWRT = IWRT + 1
203:      end if
204: C
205:      call PCTLB2( IARRAY, MEMBER, 'GAMAVG', IRET )
206:      if ( IRET.eq.0 ) then
207:      call UNPKND( IARRAY, 'GAMAVG', 'R4', GAMAVG(1,LOP), NMAT,
208: &      NDATA2, IRET )
209:      else
210:      If ( IWRT.le.0) write(NOUT1,6000)
211:      write(NOUT1,6200) LOP, MEMBER, 'GAMAVG'
212:      IWRT = IWRT + 1
213:      end if
214: C
215:      call PCTLB2( IARRAY, MEMBER, 'YLDXE5', IRET )
216:      if ( IRET.eq.0 ) then
217:      call UNPKND( IARRAY, 'YLDXE5', 'R4', YLDXE5(1,LOP), NMAT,
218: &      NDATA2, IRET )
219:      else
220:      If ( IWRT.le.0) write(NOUT1,6000)
221:      write(NOUT1,6200) LOP, MEMBER, 'YLDXE5'
222:      IWRT = IWRT + 1
223:      end if
224: C
225:      call PCTLB2( IARRAY, MEMBER, 'YLDI35', IRET )
226:      if ( IRET.eq.0 ) then
227:      call UNPKND( IARRAY, 'YLDI35', 'R4', YLDI35(1,LOP), NMAT,
228: &      NDATA2, IRET )
229:      else
230:      If ( IWRT.le.0) write(NOUT1,6000)
231:      write(NOUT1,6200) LOP, MEMBER, 'YLDI35'
232:      IWRT = IWRT + 1
233:      end if
234: C
235:      call PCTLB2( IARRAY, MEMBER, 'YLDISM9', IRET )
236:      if ( IRET.eq.0 ) then
237:      call UNPKND( IARRAY, 'YLDISM9', 'R4', YLDISM9(1,LOP), NMAT,
238: &      NDATA2, IRET )
239:      else
240:      If ( IWRT.le.0) write(NOUT1,6000)
241:      write(NOUT1,6200) LOP, MEMBER, 'YLDISM9'
242:      IWRT = IWRT + 1
243:      end if
244: C
245:      call PCTLB2( IARRAY, MEMBER, 'YLDPM9', IRET )
246:      if ( IRET.eq.0 ) then
247:      call UNPKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOP), NMAT,
248: &      NDATA2, IRET )
249:      else
250:      If ( IWRT.le.0) write(NOUT1,6000)
251:      write(NOUT1,6200) LOP, MEMBER, 'YLDPM9'
252:      IWRT = IWRT + 1
253:      end if
254: C
255:      call PCTLB2( IARRAY, MEMBER, 'RATMIC', IRET )
256:      if ( IRET.eq.0 ) then
257:      do 110 M = 1, NMAT
258:      call UNPKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOP),
259: &      2*4*NTNUC, NDATA2, IRET )
260: 110      continue

```

src/mvpburn/averg1.f

```

261:         else
262:           If ( IWRT.le.0) write(NOUT1,6000)
263:           write(NOUT1,6200) LOP, MEMBER, 'RATMIC'
264:           IWRT = IWRT + 1
265:         end if
266: C
267: 120 continue
268: 125 continue
269:       IDBG   = IBEDIT
270: C
271:       if ( IDBG.gt.0 ) then
272: C
273:         do 130 M = 1, NMAT
274:           MTBURN(1,M) = 1
275:           MTBURN(2,M) = NTNUC
276: 130 continue
277: C
278:       IMVPIN = -1
279:       NOTSTP = 0
280: C2003 .... add DENERR and flag JDENER are added (not used)
281: C       JDENER = 0 : not print density errors
282: C       DENERR : dummy array here
283: C       JDENER = 0
284:       call BURNPR( NOUT1, NOUT2, CASEID, MTYP, IMVPIN, TEMP, VOLM,
285: & TRGNAM, MTBURN, POWRZN, EXPSZN, HMINV, DMWZON, GAMAVG,
286: & YLDXE5, YLDI35, YLDISM9, YLDPM9, DENTAB, DENERR, JDENER,
287: & RATMIC, NOTSTP, IDUMMY )
288: C2003 .... end
289: C
290:       end if
291: C
292: C *** OUTPUT FOR AVERAGING
293: C
294:       write(40) NOWSTP, NMAT, NTNUC, NTFISS, LASTFP, NPAR
295:       write(40) (DAYS(I),I=1,NOWSTP)
296:       write(40) (EXPST(I),I=1,NOWSTP)
297:       write(40) (CUMMWD(I),I=1,NOWSTP)
298:       write(40) (AKEFF(I),I=1,NOWSTP)
299:       write(40) (ERRKEF(I),I=1,NOWSTP)
300:       write(40) (NHIST(I),I=1,NOWSTP)
301:       write(40) (NBATCH(I),I=1,NOWSTP)
302:       write(40) (POWERL(I),I=1,NOWSTP)
303:       write(40) (FLXNRM(I),I=1,NOWSTP)
304:       write(40) (FACNRM(I),I=1,NOWSTP)
305: C2005 ... added
306:       write(40) (ERRNRM(I),I=1,NOWSTP)
307:       write(40) (POWERW(I),I=1,NOWSTP)
308:       write(40) (POWRE(I),I=1,NOWSTP)
309:       write(40) (RMONIT(I),I=1,NOWSTP)
310:       write(40) (EMONIT(I),I=1,NOWSTP)
311: C ... convert RATMIC to original MVP results by FLXNRM
312: do 138 I = 1, NOWSTP - 1
313:   FACT = FLXNRM(I)
314:   if ( FLXNRM(I).gt.0.0 ) then
315:     FACT = 1.000 / FLXNRM(I)
316:     do 136 M = 1, NMAT
317:       do 134 N = 1, NTNUC
318:         do 132 MT = 1, 4
319:           RATMIC(1,MT,N,M,I) = RATMIC(1,MT,N,M,I) * FACT
320:           RATMIC(2,MT,N,M,I) = RATMIC(2,MT,N,M,I) * FACT
321: 132 continue
322: 134 continue
323: 136 continue
324:       end if
325: 138 continue

```

```

326: C2005 ... end
327:       write(40) ((EXPSZN(M,I),M=1,NMAT),I=1,NOWSTP)
328:       write(40) ((DMWZON(M,I),M=1,NMAT),I=1,NOWSTP)
329:       write(40) (
330: & (((RATMIC(K,MT,N,M,I),K=1,2),MT=1,4),N=1,NTNUC),M=1,NMAT
331: & ),I=1,NOWSTP)
332:       write(40) (((DENTAB(N,M,I),N=1,NTNUC),M=1,NMAT),I=1,NOWSTP)
333:       write(40) TRGNAM, MTYP, TEMP, VOLM, GAM
334: C
335: C ... data for weighted average mode
336: C
337: C2003 if ( MODEAV.eq.2 ) then
338: C2003 if ( MODEAV.eq.0 ) then
339: C       ... allways calculate weighted average (from 14 Feb 2003)
340: C2005 ... added
341:       write(41) NOWSTP, (MATREG(M),M=1,NMAT)
342: C2005 ... end
343:       write(41) (AKEFF(I)*ERRKEF(I),I=1,NOWSTP)
344:       write(41) (REAL(NHIST(I)),I=1,NOWSTP)
345: C2003 end if
346: C
347: C2003 ... check NHIST ....
348: C
349:       if ( LOP0.gt.1 ) then
350:         do 140 I=1,NOWSTP
351:           if( NHIST(I).ne.NHIST0(I).and.JHIST0.eq.0 ) then
352:             JHIST0 = 1
353:             write(NOUT1,*) '!!!(AVERG1) History number is not',
354: & ' the same over cases.'
355:           end if
356: 140 continue
357:         end if
358:         do 150 I=1,NOWSTP
359:           NHIST0(I) = NHIST(I)
360: 150 continue
361: C2003 ... end
362: C
363: C *** END OF PROCESS
364: C
365: 6000 format(/1X,9('='),' INFORMATION ON MEMBERS IN PDS ',9('='))
366: 6100 format(1X,'STEP',I3,' : MEMBER <',A,'> DOSE NOT EXIST IN PDS')
367: 6200 format(1X,'STEP',I3,' : MEMBER <',A,'> LACKS DATA <',A,'>')
368: C
369:       return
370:       end

```

src/mvpburn/averg2.f

```

1: C2003 ... added DENERR
2:   subroutine AVERG2( NOUT1, NOUT2, MODEAV, JSAVE, MEMORY, DIROUT,
3:     & CASEID, TITLE0, NCASE, MTYP, NISO, TEMP, VOLM,
4:     & TRGNAM, LENTRG, MTBURN, MTCARD, IDENT, DNINIT,
5:     & MATMVP, IPBURN, NUCLP, GAM, NBIC, PBIC,
6:     & POWRZN, EXPSZN, HMINV, DMWZON, GAMAVG, YLDXE5,
7:     & YLDI35, YLDSM9, YLDPM9, DENTAB, RATMIC, DENWRK,
8:     & RATWRK, RWORK, IWORK, D2WORK, DENERR, AVF1, AVF2,
9:   C2005& IARRAY, RARRAY )
10:    & IARRAY, RARRAY, MATREG, RWORK2, RWORK3, RWORK4 )
11: C
12: C=====
13: C Final stage of averaging
14: C=====
15: C
16:   include 'INC/_BURNP'
17: C
18:   include 'INC/_CBURNC1'
19:   include 'INC/_CBURNC2'
20:   include 'INC/_CBURNC3'
21: C
22:   character*4 CASEID
23:   character*72 TITLE0(2)
24:   character*(*) DIROUT
25: C
26: C2003
27:   character*12 IDENT(KNMAX, NMAT)
28:   character*12 TRGNAM(NMAT)
29: C
30:   integer MTYP(NMAT), NISO(NMAT), LENTRG(NMAT)
31:   integer IPBURN(KNMAX, NMAT), MATMVP(NMAT)
32:   integer MTBURN(3, NMAT), MTCARD(2, NMAT)
33: C
34:   real TEMP(NMAT), VOLM(NMAT)
35:   real DNINIT(KNMAX, NMAT)
36: C
37:   integer NUCLP(NPAR, NTNUC)
38:   real GAM(NTFISS, NTNUC)
39:   integer NBIC(NPAR, NTNUC)
40:   real PBIC(NPAR, NTNUC)
41: C
42: C
43:   real POWRZN(NMAT, NEP1)
44:   real EXPSZN(NMAT, NEP1)
45:   real HMINV(NMAT, NEP1)
46:   real DMWZON(NMAT, NEP1)
47:   real YLDXE5(NMAT, NEP1)
48:   real YLDI35(NMAT, NEP1)
49:   real YLDSM9(NMAT, NEP1)
50:   real YLDPM9(NMAT, NEP1)
51:   real GAMAVG(NMAT, NEP1)
52: C
53:   real DENTAB(NTNUC, NMAT, NEP1)
54: C2003
55:   real DENERR(NTNUC, NMAT, NEP1)
56:   real RATMIC(2, 4, NTNUC, NMAT, NEP1)
57: C
58:   real DENWRK(NTNUC, NMAT, NEP1)
59:   real RATWRK(2, 4, NTNUC, NMAT, NEP1)
60: C
61:   real RWORK(NEP1), D2WORK(NMAT, NEP1)
62:   integer IWORK(NEP1)
63: C
64:   integer IARRAY(MEMORY)
65:   real RARRAY(MEMORY)
66:   real AVF1(NEP1, NCASE)
67:   real AVF2(NEP1, NCASE)
68: C2005 ... added
69:   integer MATREG(NMAT)
70:   real RWORK2(NEP1), RWORK3(NEP1), RWORK4(NEP1)
71: C2005 ... moved to top statement
72: Cmove include 'INC/_BURNP'
73: C
74: Cmove include 'INC/_CBURNC1'
75: Cmove include 'INC/_CBURNC2'
76: Cmove include 'INC/_CBURNC3'
77: C2005 ... end
78:   real INSCR, INTCR
79: C
80:   common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
81:     & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
82:     & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
83:     & INTCR(MXSTEP), EINSCR(MXSTEP), EINTCR(MXSTEP),
84:     & FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
85:     & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRICA(MXSTEP),
86:     & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
87:     & NBATCH(MXSTEP)
88: C2005
89:     & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
90:     & RMONIT(MXSTEP), EMONIT(MXSTEP)
91: Cend
92: C
93:   character*2 STEPNM
94:   external STEPNM
95: C
96: C .... local variables .....
97: C
98:   character*72 DDNAME
99:   character*8 MEMBER
100: C
101:   real*8 AASUM
102: C
103: C -----
104: C
105:   call CLEA( POWERL, MXSTEP, 0.0 )
106: C2005 call CLEA( AKEFF, 20*MXSTEP, 0.0 )
107:   call CLEA( AKEFF, 25*MXSTEP, 0.0 )
108: C
109:   CASBRN = ' '
110:   CASPCM = ' '
111: C
112: C2005 NBOPT = 0
113: C2005 ** added
114:   call clea ( AVF1 , NEP1*NCASE , 0.0 )
115:   call clea ( AVF2 , NEP1*NCASE , 0.0 )
116:   call clea ( DENERR , NTNUC*NMAT*NEP1 , 0.0 )
117: C2005 ** ended
118: C
119:   call RWIND( 40 )
120: C
121: C
122: C ***** MODEAV = 0 : weighted average *****
123: C
124: C2003 if ( MODEAV.eq.2 ) then
125: C2003 if ( MODEAV.eq.0 ) then
126: C
127: C ... calculate factor of weighted average ...
128: C
129:   call RWIND( 41 )
130:   do 100 J = 1, NCASE

```

src/mvpburn/averg2.f

```

131: C2005 ... added
132:       if ( J.lt.NCASE ) read(41) NSTEP
133:       if ( J.eq.NCASE ) read(41) NSTEP,(MATREG(M),M=1,NMAT)
134: C2005 ... end
135: Cmod   read(41) (AVF1(I,J),I=1,NEP)
136: Cmod   read(41) (AVF2(I,J),I=1,NEP)
137:       read(41) (AVF1(I,J),I=1,NSTEP)
138:       read(41) (AVF2(I,J),I=1,NSTEP)
139: C2005 ... end
140: 100 continue
141: check %%%
142: c      write(*,*) '%% avf1 ',avf1
143: c      write(*,*) '%% avf2 ',avf2
144: c %%%
145: c
146:       do 150 I = 1, NEP
147: c
148: c      ... weight by 1/sigma**2
149: c
150:       AASUM = 0.0D0
151:       do 110 J = 1, NCASE
152: ccccc   if( AVF1(i,j).eq.0.0 ) then
153: ccccc   endif
154: C2005 .. modified
155: Cmod    AVF1(I,J) = 1.0/(AVF1(I,J)**2)
156:       if ( AVF1(i,j).gt.0.0 ) then
157: c        AVF1(I,J) = 1.0/(AVF1(I,J)**2)
158: c      else
159: c        AVF1(I,J) = 1.0/REAL(NCASE)
160: c      end if
161: C2005 ... end
162:       AASUM = AASUM + AVF1(I,J)
163: 110 continue
164:       AASUM = 1.0D0/AASUM
165:       do 120 J = 1, NCASE
166: c        AVF1(I,J) = AVF1(I,J)*AASUM
167: 120 continue
168: c
169: c      ... weight by NHIST
170: c
171:       AASUM = 0.0D0
172:       do 130 J = 1, NCASE
173: c        AASUM = AASUM + AVF2(I,J)
174: 130 continue
175:       AASUM = 1.0D0/AASUM
176:       do 140 J = 1, NCASE
177: c        AVF2(I,J) = AVF2(I,J)*AASUM
178: 140 continue
179:
180: 150 continue
181: C
182: c      ... simple average for step NEP1
183: C
184: C2005 ... modified
185: Cmod do 160 J = 1, NCASE
186: Cmod AVF1(NEP1,J) = 1.0/REAL(NCASE)
187: Cmod AVF2(NEP1,J) = 1.0/REAL(NCASE)
188: Cml60 continue
189: C
190:       do 161 I = NOWSTP,NEP1
191:       do 160 J = 1, NCASE
192: c        AVF1(I,J) = 1.0/REAL(NCASE)
193: c        AVF2(I,J) = 1.0/REAL(NCASE)
194: 160 continue
195: 161 continue

```

```

196: C2005 ... end
197: C
198:       do 340 LOP = 1, NCASE
199: C
200:       read(40) NSTEP, NMAT0, NTNUC0, NTFIS0, LASTF0, NPAR0
201:       call CHECKP( 'NMAT ', NMAT, NMAT0, NOUT1, NOUT2 )
202:       call CHECKP( 'NTNUC ', NTNUC, NTNUC0, NOUT1, NOUT2 )
203:       call CHECKP( 'NTFISS', NTFISS, NTFIS0, NOUT1, NOUT2 )
204:       call CHECKP( 'LASTFP', LASTFP, LASTF0, NOUT1, NOUT2 )
205:       call CHECKP( 'NPAR ', NPAR, NPAR0, NOUT1, NOUT2 )
206: C
207:       read(40) (RWORK(I),I=1,NSTEP)
208:       call AVERG4( RWORK, RARRAY, DAYS, NOWSTP, AVF2(1,LOP), 1 )
209:       read(40) (RWORK(I),I=1,NSTEP)
210:       call AVERG4( RWORK, RARRAY, EXPST, NOWSTP, AVF2(1,LOP), 1 )
211:       read(40) (RWORK(I),I=1,NSTEP)
212:       call AVERG4( RWORK, RARRAY, CUMMWD, NOWSTP, AVF2(1,LOP), 1 )
213:       read(40) (RWORK(I),I=1,NSTEP)
214: C2003   call AVERG4( RWORK, RARRAY, AKEFF, NOWSTP-1, AVF1(1,LOP), 1 )
215:       call AVERG4( RWORK, RARRAY, AKEFF, NOWSTP-1, AVF2(1,LOP), 1 )
216:       read(40) (RARRAY(I),I=1,NSTEP)
217: C2003   call AVERG4( RWORK, RARRAY, ERRKEF, NOWSTP-1, AVF1(1,LOP), 3 )
218:       if ( MODEAV.eq.1 ) then
219: c        call AVERG4( RWORK, RARRAY, ERRKEF, NOWSTP-1, AVF2(1,LOP), 3 )
220: c      &
221: c      end if
222: C2003 ... end
223:       read(40) (IWORK(I),I=1,NSTEP)
224:       do 170 I = 1, NOWSTP - 1
225: c        NHIST(I) = NHIST(I) + IWORK(I)
226: 170 continue
227: C
228:       read(40) (IWORK(I),I=1,NSTEP)
229:       do 180 I = 1, NOWSTP - 1
230: c        NBATCH(I) = NBATCH(I) + IWORK(I)
231: 180 continue
232: C
233:       read(40) (RWORK(I),I=1,NSTEP)
234:       call AVERG4( RWORK, RARRAY, POWERL, NOWSTP-1, AVF2(1,LOP), 1 )
235:       read(40) (RWORK(I),I=1,NSTEP)
236:       call AVERG4( RWORK, RARRAY, FLXNRM, NOWSTP-1, AVF2(1,LOP), 1 )
237: C
238: C2005 ... modified for ERRNRM
239: Cmod   read(40) (RWORK(I),I=1,NSTEP)
240: Cmod   call AVERG4( RWORK, RARRAY, FACNRM, NOWSTP-1, AVF2(1,LOP), 1 )
241:       read(40) (RWORK2(I),I=1,NSTEP)
242:       call AVERG4( RWORK2, RARRAY, FACNRM, NOWSTP-1, AVF2(1,LOP), 1 )
243: C2005 ... end
244: C
245: C2005 ... added
246: C
247: C      ... ERRNRM
248:       read(40) (RARRAY(I),I=1,NSTEP)
249:       if ( MODEAV.eq.1 ) then
250: c        call AVERG4( RWORK, RARRAY, ERRNRM, NOWSTP-1, AVF2(1,LOP), 3 )
251: c      &
252: c      end if
253: C      ... POWERW
254:       read(40) (RWORK(I),I=1,NSTEP)
255:       call AVERG4( RWORK, RARRAY, POWERW, NOWSTP-1, AVF2(1,LOP), 1 )
256: C      ... POWERE
257:       read(40) (RARRAY(I),I=1,NSTEP)
258:       if ( MODEAV.eq.1 ) then
259: c        call AVERG4( RWORK, RARRAY, POWERE, NOWSTP-1, AVF2(1,LOP), 3 )
260: c      &

```

src/mvpburn/averg2.f

```

261:      end if
262: C    ... RMONIT
263:      read(40) (RWORK(I),I=1,NSTEP)
264:      call AVERG4( RWORK, RARRAY, RMONIT, NOWSTP-1, AVF2(1,LOP), 1 )
265: C    ... EMONIT
266:      read(40) (RARRAY(I),I=1,NSTEP)
267:      if ( MODEAV.eq.1 ) then
268:          call AVERG4( RWORK, RARRAY, EMONIT, NOWSTP-1, AVF2(1,LOP), 3
269:              &
270:          )
271:      end if
272: C2005 ... end
273:      read(40) ((D2WORK(M,I),M=1,NMAT),I=1,NSTEP)
274:      LENG = NMAT*NOWSTP
275:      call AVERG4( D2WORK, RARRAY, EXPSZN, LENG, AVF2(1,LOP), 1 )
276:      do 200 I = 1, NOWSTP
277:          do 190 M = 1, NMAT
278:              EXPSZN(M,I) = EXPSZN(M,I) + AVF2(I,LOP)*D2WORK(M,I)
279:          continue
280:      continue
281:      read(40) ((D2WORK(M,I),M=1,NMAT),I=1,NSTEP)
282:      LENG = NMAT*NOWSTP
283:      call AVERG3( D2WORK, RARRAY, DMWZON, LENG, AVF2(1,LOP), 1 )
284:      do 220 I = 1, NOWSTP
285:          do 210 M = 1, NMAT
286:              DMWZON(M,I) = DMWZON(M,I) + AVF2(I,LOP)*D2WORK(M,I)
287:          continue
288:      continue
289: C    read(40) (
290:      &      (((RATWRK(K,MT,N,M,I),K=1,2),MT=1,4),N=1,NTNUC),M=1,
291:      &      NMAT),I=1,NSTEP)
292: C    do 260 I = 1, NOWSTP - 1
293:      do 250 M = 1, NMAT
294:          do 240 N = 1, NTNUC
295:              do 230 MT = 1, 4
296:                  RATMIC(1,MT,N,M,I) = RATMIC(1,MT,N,M,I)
297:                  + AVF2(I,LOP)*RATWRK(1,MT,N,M,I)
298:                  + AVF1(I,LOP)*RATWRK(2,MT,N,M,I)
299: C2003&
300:              continue
301:          continue
302:      continue
303:      continue
304: C2003 ... use RATMIC(2,MT,N,M,I) only when MODEAV=1
305:      if ( MODEAV.eq.1 ) then
306:          do 300 I = 1, NOWSTP - 1
307:              do 290 M = 1, NMAT
308:                  do 280 N = 1, NTNUC
309:                      do 270 MT = 1, 4
310:                          RATMIC(2,MT,N,M,I) = RATMIC(2,MT,N,M,I)
311:                          + (RATWRK(2,MT,N,M,I)*AVF2(I,LOP))**2
312:                          + (RATWRK(2,MT,N,M,I)*AVF1(I,LOP))**2
313: C2003&
314:                      continue
315:                  continue
316:              continue
317:          continue
318:      end if
319: C    read(40) ((DENWRK(N,M,I),N=1,NTNUC),M=1,NMAT),I=1,NSTEP)
320:      LENG = NTNUC*NMAT*NOWSTP
321:      do 330 I = 1, NOWSTP
322:          do 320 M = 1, NMAT
323:              do 310 N = 1, NTNUC
324:                  DENTAB(N,M,I) = DENTAB(N,M,I) + AVF2(I,LOP)*
325:

```

```

326:      &      DENWRK(N,M,I)
327:      310      continue
328:      320      continue
329:      330      continue
330: C
331:      if ( LOP.lt.NCASE ) read(40)
332:      if ( LOP.eq.NCASE ) read(40) TRGNAM, MTYP, TEMP, VOLM, GAM
333: C
334:      340 continue
335:
336: C2003 ...
337:      if ( MODEAV.eq.1 ) then
338: C *** MODIFY ERRKEFF FROM ABSOLUTE VALUE TO %
339:
340:          do 350 I = 1, NOWSTP - 1
341:              if ( AKEFF(I).gt.0.0 ) then
342:                  SAVE = ERRKEF(I)
343:                  ERRKEF(I) = SQRT(SAVE) /AKEFF(I)
344:              else
345:                  ERRKEF(I) = 0.0
346:              end if
347: C2005 ... added
348:              if ( FLXNRM(I).gt.0.0 ) then
349:                  SAVE = ERRNRM(I)
350:                  ERRNRM(I) = SQRT(SAVE) /FLXNRM(I)
351:              else
352:                  ERRNRM(I) = 0.0
353:              end if
354: C
355:              if ( POWERW(I).gt.0.0 ) then
356:                  SAVE = POWERE(I)
357:                  POWERE(I) = SQRT(SAVE) /POWERW(I)
358:              else
359:                  POWERE(I) = 0.0
360:              end if
361: C
362:              if ( RMONIT(I).gt.0.0 ) then
363:                  SAVE = EMONIT(I)
364:                  EMONIT(I) = SQRT(SAVE) /RMONIT(I)
365:              else
366:                  EMONIT(I) = 0.0
367:              end if
368: C2005 ... end
369:      350      continue
370:
371: C *** RESET MICROSCOPIC REACTION RATE ERROR
372:
373:          do 390 I = 1, NOWSTP - 1
374:              do 380 M = 1, NMAT
375:                  do 370 N = 1, NTNUC
376:                      do 360 MT = 1, 4
377:                          RATMIC(2,MT,N,M,I) = SQRT(RATMIC(2,MT,N,M,I))
378:                      continue
379:                  continue
380:              continue
381:          continue
382:      end if
383: C2003 ... end
384: C
385: C2003 ... calculate error of keff, micro rate (MODEAV=0) and density
386:      if ( NCASE.gt.1 ) then
387:          call RWIND( 40 )
388:          do 480 LOP = 1, NCASE
389:              read(40)
390:              read(40)

```

src/mvpburn/averg2.f

```

391:      read(40)
392:      read(40)
393: C      .... K-EFF
394:      read(40) (RWORK(I),I=1,NSTEP)
395:      read(40)
396:      read(40)
397:      read(40)
398:      read(40)
399: C2005      read(40)
400: C      .... FLXNRM
401:      read(40) (RWORK2(I),I=1,NSTEP)
402:      read(40)
403: C2005 ... added
404:      read(40)
405: C      .... POWERW
406:      read(40) (RWORK3(I),I=1,NSTEP)
407:      read(40)
408: C      .... RMONIT
409:      read(40) (RWORK4(I),I=1,NSTEP)
410:      read(40)
411: C2005 ... end
412:      read(40)
413:      read(40)
414: C      .... micro-reaction rate
415:      read(40) (
416: &      (((RATWRK(K,MT,N,M,I),K=1,2),MT=1,4),N=1,NTNUC),M=
417: &      1,NMAT),I=1,NSTEP)
418:      read(40) (((DENWRK(N,M,I),N=1,NTNUC),M=1,NMAT),I=1,NOWSTP)
419:      read(40)
420: C
421:      if ( MODEAV.eq.0 ) then
422:        do 400 I = 1, NOWSTP - 1
423:          ERRKEF(I) = ERRKEF(I) + AVF2(I,LOP) /
424: &          (1.0d0-AVF2(I,LOP))*(RWORK(I)-AKEFF(I))**2
425: C2005 ... added
426:          ERRNRM(I) = ERRNRM(I) + AVF2(I,LOP) /
427: &          (1.0d0-AVF2(I,LOP))*(RWORK2(I)-FLXNRM(I))**2
428:          POWERE(I) = POWERE(I) + AVF2(I,LOP) /
429: &          (1.0d0-AVF2(I,LOP))*(RWORK3(I)-POWERW(I))**2
430:          EMONIT(I) = EMONIT(I) + AVF2(I,LOP) /
431: &          (1.0d0-AVF2(I,LOP))*(RWORK4(I)-RMONIT(I))**2
432: C2004 ... end
433:      400      continue
434: C
435:      do 440 I = 1, NOWSTP - 1
436:        do 430 M = 1, NMAT
437:          do 420 N = 1, NTNUC
438:            do 410 MT = 1, 4
439:              RATMIC(2,MT,N,M,I) = RATMIC(2,MT,N,M,I)
440: &              + AVF2(I,LOP) / ((1.0d0-AVF2(I,LOP))*
441: &              (RATWRK(1,MT,N,M,I)
442: &              -RATMIC(1,MT,N,M,I))**2
443:      410      continue
444:      420      continue
445:      430      continue
446:      440      continue
447: C
448:      end if
449: C
450:      do 470 I = 1, NOWSTP
451:        do 460 M = 1, NMAT
452:          do 450 N = 1, NTNUC
453:            DENERR(N,M,I) = DENERR(N,M,I) + AVF2(I,LOP) /
454: &            (1.0d0-AVF2(I,LOP))*
455: &            (DENWRK(N,M,I)-DENTAB(N,M,I))**2

```

```

456:      450      continue
457:      460      continue
458:      470      continue
459: C
460:      480      continue
461: C
462:      if ( MODEAV.eq.0 ) then
463:        do 490 I = 1, NOWSTP - 1
464:          if ( AKEFF(I).ne.0.0 ) then
465:            ERRKEF(I) = 100.0*SQRT(ERRKEF(I)/DBLE(NCASE)) /
466: &            AKEFF(I)
467:          else
468:            ERRKEF(I) = 0.0
469:          end if
470: C2005 ... added
471:          if ( FLXNRM(I).ne.0.0 ) then
472:            ERRNRM(I) = 100.0*SQRT(ERRNRM(I)/DBLE(NCASE)) /
473: &            FLXNRM(I)
474:          else
475:            ERRNRM(I) = 0.0
476:          end if
477: C
478:          if ( POWERW(I).ne.0.0 ) then
479:            POWERE(I) = 100.0*SQRT(POWERE(I)/DBLE(NCASE)) /
480: &            POWERW(I)
481:          else
482:            POWERE(I) = 0.0
483:          end if
484: C
485:          if ( RMONIT(I).ne.0.0 ) then
486:            EMONIT(I) = 100.0*SQRT(EMONIT(I)/DBLE(NCASE)) /
487: &            RMONIT(I)
488:          else
489:            EMONIT(I) = 0.0
490:          end if
491: C2005 ... end
492:      490      continue
493: C
494:      do 530 I = 1, NOWSTP - 1
495:        do 520 M = 1, NMAT
496:          do 510 N = 1, NTNUC
497:            do 500 MT = 1, 4
498:              if ( RATMIC(1,MT,N,M,I).ne.0.0 ) then
499:                RATMIC(2,MT,N,M,I) =
500: &                SQRT(RATMIC(2,MT,N,M,I)/DBLE(NCASE))
501:              else
502:                RATMIC(2,MT,N,M,I) = 0.0
503:              end if
504:            500      continue
505:            510      continue
506:            520      continue
507:            530      continue
508:          end if
509: C
510:          do 560 I = 1, NOWSTP
511:            do 550 M = 1, NMAT
512:              do 540 N = 1, NTNUC
513:                ... change to % error
514:                if ( DENTAB(N,M,I).ne.0.0 ) then
515:                  DENERR(N,M,I) = 100.0*
516: &                  SQRT(DENERR(N,M,I)/DBLE(NCASE)) /
517: &                  DENTAB(N,M,I)
518:                else
519:                  DENERR(N,M,I) = 0.0
520:                end if

```

src/mvpburn/averg2.f

```

521:      540      continue
522:      550      continue
523:      560      continue
524:
525:      end if
526: C2003 ... end
527: C
528: C
529: C ***** MODEAV = 1 : simple average *****
530: C
531: C
532: C      else
533: C
534: C
535: C      FACT = 1.000/FLOAT(NCASE)
536: C
537: C      do 500 LOP = 1, NCASE
538: *      read(40) NSTEP, NMAT0, NTNUC0, NTFIS0, LASTF0, NPAR0
539: *      call CHECKP( 'NMAT', NMAT, NMAT0, NOUT1, NOUT2 )
540: *      call CHECKP( 'NTNUC', NTNUC, NTNUC0, NOUT1, NOUT2 )
541: *      call CHECKP( 'NTFISS', NTFISS, NTFIS0, NOUT1, NOUT2 )
542: *      call CHECKP( 'LASTFP', LASTFP, LASTF0, NOUT1, NOUT2 )
543: *      call CHECKP( 'NPAR', NPAR, NPAR0, NOUT1, NOUT2 )
544: C
545: C
546: CM      READ(40) (DAYS (I), I=1, NSTEP)
547: *      read(40) (RWORK(I), I=1, NSTEP)
548: *      call AVERG3( RWORK, RARRAY, DAYS, NOWSTP, FACT, 1 )
549: CM      READ(40) (EXPST (I), I=1, NSTEP)
550: *      read(40) (RWORK(I), I=1, NSTEP)
551: check %%%%%%%%%
552: c      write(*,*) '%% nstep nowstp ', NSTEP, nowstp
553: c      write(*,*) '%% expst ', (EXPST(I), I=1, nowstp)
554: c      write(*,*) '%% rwork ', (RWORK(I), I=1, nowstp)
555: c %%%%%%%%%%
556: *      call AVERG3( RWORK, RARRAY, EXPST, NOWSTP, FACT, 1 )
557: *      read(40) (RWORK(I), I=1, NSTEP)
558: *      call AVERG3( RWORK, RARRAY, CUMMWD, NOWSTP, FACT, 1 )
559: *      read(40) (RWORK(I), I=1, NSTEP)
560: *      call AVERG3( RWORK, RARRAY, AKEFF, NOWSTP-1, FACT, 1 )
561: *      read(40) (RARRAY(I), I=1, NSTEP)
562: *      call AVERG3( RWORK, RARRAY, ERRKEF, NOWSTP-1, 1.00E-2, 3 )
563: *      read(40) (IWORK(I), I=1, NSTEP)
564: *      do 400 I = 1, NOWSTP - 1
565: *          NHIST(I) = NHIST(I) + IWORK(I)
566: *      400      continue
567: *      read(40) (IWORK(I), I=1, NSTEP)
568: *      do 410 I = 1, NOWSTP - 1
569: *          NBATCH(I) = NBATCH(I) + IWORK(I)
570: *      410      continue
571: *      read(40) (RWORK(I), I=1, NSTEP)
572: *      call AVERG3( RWORK, RARRAY, POWERL, NOWSTP-1, FACT, 1 )
573: *      read(40) (RWORK(I), I=1, NSTEP)
574: *      call AVERG3( RWORK, RARRAY, FLXNRM, NOWSTP-1, FACT, 1 )
575: *      read(40) (RWORK(I), I=1, NSTEP)
576: *      call AVERG3( RWORK, RARRAY, FACNRM, NOWSTP-1, FACT, 1 )
577: *      read(40) ((D2WORK(M,I), M=1, NMAT), I=1, NSTEP)
578: *      LENG = NMAT*NOWSTP
579: *      call AVERG3( D2WORK, RARRAY, EXPSZN, LENG, FACT, 1 )
580: *      read(40) ((D2WORK(M,I), M=1, NMAT), I=1, NSTEP)
581: *      LENG = NMAT*NOWSTP
582: *      call AVERG3( D2WORK, RARRAY, DMWZON, LENG, FACT, 1 )
583: C
584: *      read(40) (
585: *          &

```

```

586: *          &
587: check %%%%%%%%%
588: c      write(*,*) '%% ratwrk '
589: c      do j=1, nstep
590: c          write(*, '(lx,a,i3/(lx,8e12.5))') '%% step ', J,
591: c          & (ratwrk(i,1,1,1,j), i=1, 2*4*NTNUC*NMAT)
592: c      end do
593: c %%%%%%%%%
594: C
595: *      do 450 I = 1, NOWSTP - 1
596: *          do 440 M = 1, NMAT
597: *              do 430 N = 1, NTNUC
598: *                  do 420 MT = 1, 4
599: *                      RATMIC(1, MT, N, M, I) = RATMIC(1, MT, N, M, I)
600: *                      & + FACT*RATWRK(1, MT, N, M, I)
601: *      420      continue
602: *      430      continue
603: *      440      continue
604: *      450      continue
605: *      do 490 I = 1, NOWSTP - 1
606: *          do 480 M = 1, NMAT
607: *              do 470 N = 1, NTNUC
608: *                  do 460 MT = 1, 4
609: *                      RATMIC(2, MT, N, M, I) = RATMIC(2, MT, N, M, I)
610: *                      & + RATWRK(2, MT, N, M, I)*RATWRK(2, MT, N, M, I)
611: *      460      continue
612: *      470      continue
613: *      480      continue
614: *      490      continue
615: C
616: *      read(40) (((DENWRK(N,M,I), N=1, NTNUC), M=1, NMAT), I=1, NSTEP)
617: *      LENG = NTNUC*NMAT*NOWSTP
618: *      call AVERG3( DENWRK, RARRAY, DENTAB, LENG, FACT, 1 )
619: C
620: *      if ( LOP.lt.NCASE ) read(40)
621: CCCC if ( LOP.eq.NCASE ) read(40) TRGNAM, MTYP, TEMP, VOLM, MVPDAT,
622: CCCC & GAM
623: *      if ( LOP.eq.NCASE ) read(40) TRGNAM, MTYP, TEMP, VOLM, GAM
624: C
625: *      500      continue
626: C
627: C *** MODIFY ERRKEFF FROM ABSOLUTE VALUE TO %
628: C
629: *      do 510 I = 1, NOWSTP - 1
630: *          if ( AKEFF(I).gt.0.0 ) then
631: *              SAVE = ERRKEF(I)
632: *              ERRKEF(I) = 100.0*SQRT(SAVE)*FACT/AKEFF(I)
633: *          else
634: *              ERRKEF(I) = 0.0
635: *          end if
636: *      510      continue
637: C
638: C *** RESET MICROSCOPIC REACTION RATE ERROR
639: C
640: *      do 550 I = 1, NOWSTP - 1
641: *          do 540 M = 1, NMAT
642: *              do 530 N = 1, NTNUC
643: *                  do 520 MT = 1, 4
644: *                      SAVE = RATMIC(2, MT, N, M, I)
645: *                      RATMIC(2, MT, N, M, I) = SQRT(SAVE)*FACT
646: *      520      continue
647: *      530      continue
648: *      540      continue
649: *      550      continue
650: C

```

src/mvpburn/averg2.f

```

651: C2003 ... calculate error of density
652: *      if ( NCASE.gt.1 ) then
653: *          call RWIND(40)
654: *          do 558 LOP=1,NCASE
655: *              read(40)
656: *              read(40)
657: *              read(40)
658: *              read(40)
659: *              read(40)
660: *              read(40)
661: *              read(40)
662: *              read(40)
663: *              read(40)
664: *              read(40)
665: *              read(40)
666: *              read(40)
667: *              read(40)
668: *              read(40)
669: *              read(40)((DENWRK(N,M,I),N=1,NTNUC),M=1,NMAT),I=1,NOWSTP)
670: *              read(40)
671:
672: *          do 556 I = 1, NOWSTP
673: *              do 554 M = 1, NMAT
674: *                  do 552 N = 1, NTNUC
675: *                      DENERR(N,M,I) = DENERR(N,M,I) +
676: *                          &      FACT/(1.0d0-FACT)*
677: *                          &      (DENWRK(N,M,I)-DENTAB(N,M,I))**2
678: *                  552      continue
679: *              554      continue
680: *          556      continue
681: *      558      continue
682: C
683: *      do 1556 I = 1, NOWSTP
684: *          do 1554 M = 1, NMAT
685: *              do 1552 N = 1, NTNUC
686: *                  ... change to % error
687: *                  if( DENTAB(N,M,I).ne.0.0 ) then
688: *                      DENERR(N,M,I) = 100.0*
689: *                          &      Sqrt(DENERR(N,M,I)/DBLE(NCASE))
690: *                      &      /DENTAB(N,M,I)
691: *                  else
692: *                      DENERR(N,M,I) = 0.0
693: *                  end if
694: *              1552      continue
695: *          1554      continue
696: *      1556      continue
697:
698: *      end if
699: C2003 ... end
700:
701: *      end if
702: C
703:      ST235 = 0.0
704:      TWTHVY = 0.0
705:      do 580 M = 1, NMAT
706:          WTSAVE = 0.0
707:          if ( IDU235.gt.0 ) then
708:              ST235 = ST235 + VOLM(M)*DENTAB(IDU235,M,1)
709:          end if
710:          do 570 I = 1, NTFISS
711:              WTSAVE = WTSAVE + AMASS(I)*DENTAB(I,M,1)
712:          570      continue
713:          SAVE = WTSAVE/ABOGA
714:          HMINV(M,1) = SAVE*1.000E-6
715:          TWTHVY = TWTHVY + VOLM(M)*SAVE*1.000E-6

```

```

716: C
717:      if ( IBEDIT.gt.2 ) write(NOUT1,*) ' M WT(GRAM) : ', M, SAVE
718:      580 continue
719:
720:      ST235 = ST235*1.000E+24
721:      if ( IBEDIT.gt.2 ) write(NOUT1,*) ' ST235 TWTHVY : ', ST235,
722:      &      TWTHVY
723: C
724: C *** LOOP OF BURNUP STEP
725: C
726: C2005 ... added
727:      IVOID = IBC(4)
728:      IBNTYP = IBC(9)
729: C2005 ... end
730: C
731: C >>> loop of burnup step
732: C
733:      do 790 J79 = 1, NOWSTP - 1
734:          K79 = J79 + 1
735:          U5FNOW = 0.0
736:          POWNOW = ABS( POWERL(J79) )
737: C2005 ... added
738:          if ( IBNTYP.eq.1 ) POWNOW = ABS( EXTSRC(J79) )
739:          if ( IBNTYP.eq.2 ) POWNOW = ABS( VMONIT(J79) )
740:          if ( IBNTYP.eq.3 ) POWNOW = ABS( VMONIT(J79) )
741:          if ( IBNTYP.eq.4 ) POWNOW = ABS( REGPOW(J79) )
742: C
743:          NBOPT = 0
744:          if ( POWNOW.eq.0.0 ) NBOPT = 1
745: Check
746:          if ( IBEDIT.ge.2 ) then
747:              write(NOUT1,*) ' ** IVOID IBNTYP (averg2) : ', IVOID, IBNTYP
748:              write(NOUT1,*) ' ** NBOPT POWNOW (averg2) : ', NBOPT, POWNOW
749:          end if
750: C2005 ... end
751: C
752:          do 600 M = 1, NMAT
753:              WTSAVE = 0.0
754:              if ( IDU235.gt.0 ) then
755:                  U5FNOW = U5FNOW + VOLM(M)*DENTAB(IDU235,M,K79)
756:              end if
757:              do 590 I = 1, NTFISS
758:                  WTSAVE = WTSAVE + AMASS(I)*DENTAB(I,M,K79)
759:              590      continue
760:              SAVE = WTSAVE/ABOGA
761:              HMINV(M,K79) = SAVE*1.000E-6
762:              if ( IBEDIT.gt.2 ) write(NOUT1,*) ' M WT(GRAM) : ', M, SAVE
763:              600      continue
764:              if ( IBEDIT.gt.2 ) write(NOUT1,*) ' U5FNOW : ', U5FNOW
765: C
766:              U5FNOW = U5FNOW*1.000E+24
767:              if ( ST235.gt.0.0 ) U5FNOW = (ST235-U5FNOW)*100.0/ST235
768:              if ( IBEDIT.gt.2 ) write(NOUT1,*) ' U5FNOW ST235 : ',
769:              &      U5FNOW, ST235
770:              U235F(K79) = U5FNOW
771: C
772: C2005 .... modified
773: C
774: Cmod      POWNRM = 0.0
775: Cmod      POWNOW = POWERL(J79)
776: C
777: Cmod      call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
778: Cmod &      POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
779: Cmod &      POWNOW, POWNRM, EVTOJ, REACEV, J79, NOUT1 )
780: C

```


src/mvpburn/averg2.f

```

781: Cmod      if ( POWNRM.gt.0.0 ) FLXNRM(J79)      = FLXNRM(J79)*POWNRM
782: C
783: C ... Branchinf-off calculation
784: C
785:      if ( IVOID.eq.1 ) then
786:          POWNRM = FLXNRM(J79)
787:          POWNOW = 0.0
788: C
789:          call BURNRV( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
790: &          POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
791: &          POWNOW,POWNRM,EVTOJ, REACEV, J79, NOUT1 )
792: C
793:          POWERW(J79) = POWNOW
794:          POWERE(J79) = ERRNRM(J79)
795:          endif
796: C
797: C ... Normal Burnup calculation
798: C
799:      if ( IVOID.eq.0 ) then
800:          if ( IBNTYP.eq.0 ) then
801:              POWNOW = ABS( POWERL(J79) )
802:              ERRNOW = 0.0
803: C
804:              call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
805: &              POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
806: &              POWNOW, POWNRM, EVTOJ, REACEV, J79, NOUT1 , ERRNOW )
807: C
808:              POWERW(J79) = POWNOW
809:              POWERE(J79) = ERRNOW
810:              FLXNRM(J79) = POWNRM
811:              ERRNRM(J79) = ERRNOW
812: C
813:          else if ( IBNTYP.eq.1 ) then
814:              POWNOW = 0.0
815:              POWNRM = ABS( EXTSRC(J79) )
816: C
817:              if ( IBEDIT.ge.2 ) then
818:                  write(NOUT1,*) ' ** J79 EXTSRC(J79) : ',J79,POWNRM
819:                  end if
820: C
821:              call BURNRX( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
822: &              POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
823: &              POWNOW,POWNRM,EVTOJ, REACEV, J79, NOUT1 )
824: C
825:              POWERW(J79) = POWNOW
826:              POWERE(J79) = 0.0
827:              FLXNRM(J79) = POWNRM
828:              ERRNRM(J79) = 0.0
829: C
830:          else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
831:              POWNOW = 0.0
832:              RATMON = ABS( VMONIT(J79) )
833:              RATNOW = RMONIT(J79)
834:              ERRNOW = EMONIT(J79)
835: C
836:              if(RATNOW.gt.0.0.and.RATMON.gt.0.0) then
837:                  POWNRM = RATMON/RATNOW
838:              else
839:                  POWNRM = 0.0
840:                  ERRNOW = 0.0
841:              endif
842: C
843:              if ( IBEDIT.ge.2 ) then
844:                  write(NOUT1,*) ' ** J79 ISPC      : ',J79,ISPC
845:                  write(NOUT1,*) ' ** VMONIT POWNRM : ',VMONIT(J79),POWNRM

```

```

846:                  write(NOUT1,*) ' ** RATNOW ERRNOW : ',RATNOW,ERRNOW
847:                  end if
848: C
849:          call BURNRX( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
850: &          POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
851: &          POWNOW,POWNRM,EVTOJ, REACEV, J79, NOUT1 )
852: C
853:          POWERW(J79) = POWNOW
854:          POWERE(J79) = ERRNOW
855:          FLXNRM(J79) = POWNRM
856:          ERRNRM(J79) = ERRNOW
857: C
858:          else if ( IBNTYP.eq.4 ) then
859:              POWNOW = 0.0
860:              POWNRM = 0.0
861:              ERRNOW = 0.0
862:              POWRGN = ABS( REGPOW(J79) )
863: C
864:              if ( IBEDIT.ge.2 ) then
865:                  write(NOUT1,*) ' ** J79 REGPOW(J79) : ',J79,REGPOW(J79)
866:                  write(NOUT1,*) ' ** NMAT is ',NMAT,' & MATREG is : '
867:                  write(NOUT1,'(10I4)') (MATREG(M),M=1,NMAT)
868:                  end if
869: C
870:              call BURNR4( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, RARRAY,
871: &              POWRZN(1,J79), DENTAB(1,1,J79), RATMIC(1,1,1,J79),
872: &              POWNOW,POWNRM,
873: &              EVTOJ, REACEV, J79, NOUT1, MATREG, POWRGN ,ERRNOW )
874: C
875:              POWERW(J79) = POWNOW
876:              POWERE(J79) = ERRNOW
877:              FLXNRM(J79) = POWNRM
878:              ERRNRM(J79) = ERRNOW
879: C
880:              end if
881:          end if
882: C
883: C2005 ... end
884: C
885: C
886: C **** SET FISSILE ABSORPTION RATE
887: C
888:          FISABS(J79) = 0.0
889:          FDECAY(J79) = 0.0
890:          ESAVE = 0.0
891: C2005 ... added
892:          DNTEMP = 0.0
893: C2005 ... end
894:          do 640 M = 1, NMAT
895:              RSAVE = 0.0
896:              do 630 ISO = 1, NFIS
897:                  SAVE = 0.0
898:                  do 610 J = 1, NTNUC
899:                      if ( NAMFIS(ISO)(1:3).eq.SRACID(J)(2:4) ) then
900:                          if ( NAMFIS(ISO)/10.eq.NCODE(J) ) then
901:                              JPOS = J
902:                              go to 620
903:                          end if
904:                      continue
905:                      go to 630
906:                  continue
907:                  DNTEMP = DENTAB(JPOS,M,J79)
908:                  MT = 0
909:                  if ( NAMFIS(ISO)(4:4).eq.'F' ) MT = 1
910:                  if ( NAMFIS(ISO)(4:4).eq.'C' ) MT = 2

```

src/mvpburn/averg2.f

```

911: C2003      if ( NAMFIS(ISO)(4:4).eq.'A' ) MT = 4
912: C2003      if ( NAMFIS(ISO)(4:4).eq.'P' ) MT = 0
913: C2003      if ( NAMFIS(ISO)(4:4).eq.'N' ) MT = 3
914: C2003      if ( NAMFIS(ISO)(4:4).eq.'D' ) MT = 5
915:           MT = mod(NAMFIS(ISO),10)
916: C2003 ... end
917:           if ( MT.eq.0 ) go to 630
918: C
919:           if ( MT.eq.5 ) then
920:             FDECAY(J79) = FDECAY(J79) + 1.00E+24*RAMDA(JPOS)*
921:             & DNTMP*VOLM(M)
922:             go to 630
923:           end if
924: C
925: C2005 ... added
926:           if ( NBOPT.eq.1 ) go to 630
927: C2004 ... end
928:           if ( IFISDN(ISO).eq.0 ) then
929:             RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FISFCT(ISO)
930:             SAVE = RATMIC(2,MT,JPOS,M,J79)*FISFCT(ISO)
931: C
932:           else
933:             RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FISFCT(ISO)*
934:             & DNTMP
935:             SAVE = RATMIC(2,MT,JPOS,M,J79)*FISFCT(ISO)*DNTMP
936:           end if
937: C
938:             ESAVE = ESAVE + SAVE*SAVE*VOLM(M)*VOLM(M)
939: C
940:           630 continue
941:             FISABS(J79) = FISABS(J79) + RSAVE*VOLM(M)
942: C
943:             if ( IBEDIT.gt.2 ) then
944:               write(NOUT1,*) '** M FISABS(J79) RSAVE ESAVE VOLM(M) : ',
945:               & M, FISABS(J79), RSAVE, ESAVE, VOLM(M)
946:             end if
947: C
948:           640 continue
949:             EFISAB(J79) = 0.0
950:             if ( ESAVE.gt.0.0 ) EFISAB(J79) = SQRT(ESAVE)
951: C
952: C **** SET FERTILE CAPTURE RATE
953: C
954:             CDECAY(J79) = 0.0
955:             FRTCAP(J79) = 0.0
956:             ESAVE = 0.0
957:             do 680 M = 1, NMAT
958:               RSAVE = 0.0
959:               do 670 ISO = 1, NFER
960:                 SAVE = 0.0
961:                 do 650 J = 1, NTNUC
962:                   C2003      if ( NAMFER(ISO)(1:3).eq.SRACID(J)(2:4) ) then
963:                     if ( NAMFER(ISO)/10.eq.NCODE(J) ) then
964:                       JPOS = J
965:                       go to 660
966:                     end if
967:                   650 continue
968:                     go to 670
969:                   660 continue
970:                     DNTMP = DENTAB(JPOS,M,J79)
971:                     MT = 0
972:                   C2003      if ( NAMFER(ISO)(4:4).eq.'F' ) MT = 1
973:                   C2003      if ( NAMFER(ISO)(4:4).eq.'C' ) MT = 2
974:                   C2003      if ( NAMFER(ISO)(4:4).eq.'A' ) MT = 4
975:                   C2003      if ( NAMFER(ISO)(4:4).eq.'P' ) MT = 0

```

```

976: C2003      if ( NAMFER(ISO)(4:4).eq.'N' ) MT = 3
977: C2003      if ( NAMFER(ISO)(4:4).eq.'D' ) MT = 5
978:           MT = mod(NAMFER(ISO),10)
979: C2003 ... end
980:           if ( MT.eq.0 ) go to 670
981: C
982:           if ( MT.eq.5 ) then
983:             CDECAY(J79) = CDECAY(J79) + 1.00E+24*RAMDA(JPOS)*
984:             & DNTMP*VOLM(M)
985:             go to 670
986:           end if
987: C
988: C2005 ... added
989:           if ( NBOPT.eq.1 ) go to 670
990: C2004 ... end
991:           if ( IFRTDN(ISO).eq.0 ) then
992:             RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FRTFCT(ISO)
993:             SAVE = RATMIC(2,MT,JPOS,M,J79)*FRTFCT(ISO)
994: C
995:           else
996:             RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FRTFCT(ISO)*
997:             & DNTMP
998:             SAVE = RATMIC(2,MT,JPOS,M,J79)*FRTFCT(ISO)*DNTMP
999:           end if
1000: C
1001:           670 continue
1002: C
1003:             FRTCAP(J79) = FRTCAP(J79) + RSAVE*VOLM(M)
1004:             ESAVE = ESAVE + SAVE*SAVE*VOLM(M)*VOLM(M)
1005: C
1006:             if ( IBEDIT.gt.2 ) then
1007:               write(NOUT1,*) '** M FRTCAP(J79) RSAVE ESAVE VOLM(M) : ',
1008:               & M, FRTCAP(J79), RSAVE, ESAVE, VOLM(M)
1009:             end if
1010: C
1011:           680 continue
1012:             EFRTCA(J79) = 0.0
1013:             if ( ESAVE.gt.0.0 ) EFRTCA(J79) = SQRT(ESAVE)
1014: C
1015: C **** SET CONVERSION RATIO
1016: C
1017:             ASAVE = FISABS(J79) + FDECAY(J79)
1018:             CSAVE = FRTCAP(J79) + CDECAY(J79)
1019:             ERRABS = 0.0
1020:             ERRCAP = 0.0
1021:             if ( ASAVE.gt.0.0 ) ERRABS = 100.0*EFISAB(J79) /ASAVE
1022:             if ( CSAVE.gt.0.0 ) ERRCAP = 100.0*EFRTCA(J79) /CSAVE
1023:             INSCR(J79) = 0.0
1024:             EINSR(J79) = 0.0
1025:             if ( ASAVE.gt.0.0 ) then
1026:               INSCR(J79) = CSAVE/ASAVE
1027:               EINSR(J79) = SQRT(ERRABS*ERRABS+ERRCAP*ERRCAP)
1028:             end if
1029: C
1030:             if ( POWNOW.le.0.0 ) then
1031:               if ( FDECAY(J79).gt.0.0.and.CDECAY(J79).gt.0.0 ) then
1032:                 INSCR(J79) = CDECAY(J79) /FDECAY(J79)
1033:                 EINSR(J79) = 0.0
1034:               end if
1035:             end if
1036: C
1037: C **** CALCULATE AVERAGE GAMMA DATA
1038: C
1039:             do 700 M = 1, NMAT
1040:               SUM1 = 0.0

```

src/mvpburn/averg2.f

```

1041:      SUM2      = 0.0
1042:      do 690 ISO = 1, NTFISS
1043:         DNTEMP  = DENTAB(ISO,M,J79)
1044:         GAMISO  = REACEV(1,ISO)
1045:         SUM1    = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1046:         SUM2    = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*GAMISO
1047: 690      continue
1048:         GAMAVG(M,J79) = 0.0
1049:         if ( SUM1.gt.0.0 ) then
1050:            SAVE   = SUM2/SUM1
1051:            GAMAVG(M,J79) = SAVE*EVTOJ*1.0000E+6
1052:         end if
1053: 700      continue
1054: C
1055: C *** CALCULATE AVERAGE XE-135 YIELD DATA
1056: C
1057:      if ( IDXE35.gt.0 ) then
1058:         do 720 M = 1, NMAT
1059:            YLDXE5(M,J79) = GAM(IDU235,IDXE35)
1060:            if ( MTYP(M).ne.1 ) go to 720
1061:            SUM1 = 0.0
1062:            SUM2 = 0.0
1063:            do 710 ISO = 1, NTFISS
1064:               DNTEMP = DENTAB(ISO,M,J79)
1065:               YLDTMP = GAM(ISO,IDXE35)
1066:               SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1067:               SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1068: 710          continue
1069:               YLDXE5(M,J79) = 0.0
1070:               if ( SUM1.gt.0.0 ) YLDXE5(M,J79) = SUM2/SUM1
1071: 720          continue
1072:            end if
1073: C
1074: C *** CALCULATE AVERAGE I-135 YIELD DATA
1075: C
1076:      if ( IDI135.gt.0 ) then
1077:         do 740 M = 1, NMAT
1078:            YLDI35(M,J79) = GAM(IDU235,IDI135)
1079:            if ( MTYP(M).ne.1 ) go to 740
1080:            SUM1 = 0.0
1081:            SUM2 = 0.0
1082:            do 730 ISO = 1, NTFISS
1083:               DNTEMP = DENTAB(ISO,M,J79)
1084:               YLDTMP = GAM(ISO,IDI135)
1085:               SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1086:               SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1087: 730          continue
1088:               YLDI35(M,J79) = 0.0
1089:               if ( SUM1.gt.0.0 ) YLDI35(M,J79) = SUM2/SUM1
1090: 740          continue
1091:            end if
1092: C
1093: C *** CALCULATE AVERAGE SM-149 YIELD DATA
1094: C
1095:      if ( IDSM49.gt.0 ) then
1096:         do 760 M = 1, NMAT
1097:            YLDSM9(M,J79) = GAM(IDU235,IDSM49)
1098:            if ( MTYP(M).ne.1 ) go to 760
1099:            SUM1 = 0.0
1100:            SUM2 = 0.0
1101:            do 750 ISO = 1, NTFISS
1102:               DNTEMP = DENTAB(ISO,M,J79)
1103:               YLDTMP = GAM(ISO,IDSM49)
1104:               SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1105:               SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP

```

```

1106: 750      continue
1107:         YLDSM9(M,J79) = 0.0
1108:         if ( SUM1.gt.0.0 ) YLDSM9(M,J79) = SUM2/SUM1
1109: 760      continue
1110:      end if
1111: C
1112: C *** CALCULATE AVERAGE PM-149 YIELD DATA
1113: C
1114:      if ( IDPM49.gt.0 ) then
1115:         do 780 M = 1, NMAT
1116:            YLDPM9(M,J79) = GAM(IDU235,IDPM49)
1117:            if ( MTYP(M).ne.1 ) go to 780
1118:            SUM1 = 0.0
1119:            SUM2 = 0.0
1120:            do 770 ISO = 1, NTFISS
1121:               DNTEMP = DENTAB(ISO,M,J79)
1122:               YLDTMP = GAM(ISO,IDPM49)
1123:               SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1124:               SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1125: 770          continue
1126:               YLDPM9(M,J79) = 0.0
1127:               if ( SUM1.gt.0.0 ) YLDPM9(M,J79) = SUM2/SUM1
1128: 780          continue
1129:            end if
1130: C
1131: C 790 continue
1132: C
1133: C *** CALL BURNISM
1134: C
1135:      LENGSM = 0
1136: C
1137:      do 800 M = 1, NMAT
1138:         MTBURN(1,M) = 1
1139:         MTBURN(2,M) = NTNUC
1140: C2003
1141:         MTBURN(3,M) = MTYP(M)
1142: 800      continue
1143: C
1144:      IMVPIN = -1
1145:      NOTSTP = 0
1146: C
1147:      TITLE(1) = TITLE0(1)
1148:      TITLE(2) = TITLE0(2)
1149: C
1150: C2003 ... add DENERR and JDENER
1151: C      JDENER = 1 : print DENERR
1152:      JDENER = 1
1153:      call BURNISM( LENGSM, MEMORY, NOUT1, NOUT2, CASEID, MTYP, IMVPIN,
1154: &      TEMP, VOLM, TRGNAM, MTBURN, POWRZN, EXPSZN, HMINV, DMWZON,
1155: &      GAMAVG, YLDXE5, YLDI35, YLDSM9, YLDPM9, DENTAB, DENERR,
1156: &      JDENER, RATMIC, GAM, IARRAY, RARRAY, NCASE, NOTSTP )
1157: C2003 ... end
1158: C
1159: C *** Output data of new case created by averaging as PDS members
1160: C
1161:      if ( JSAVE.ne.0 ) then
1162:         call RWCOM1( 'WRITE', DIROUT, CASEID )
1163:         call RWCOM2( 'WRITE', DIROUT, CASEID )
1164:         call RWCOM3( 'WRITE', DIROUT, CASEID )
1165:         call RWREST( 'WRITE', DIROUT, CASEID )
1166: C
1167:      call RWMATD( 'WRITE', DIROUT, CASEID, NMAT, KNMAX, NISO(1),
1168: &      TEMP(1), VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1),
1169: &      MTCARD(1,1), IDENT(1,1), DNINIT(1,1), MATMVP(1),
1170: C2005&      IPBURN(1,1) )

```

src/mvpburn/averg2.f

```

1171:      &          IPBURN(1,1), MATREG(1) )
1172: Cend
1173:      call RWCHAN( 'WRITE', DIROUT, CASEID, NPAR, NTNUC, NTFISS,
1174:      &          NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
1175: C
1176:      DDNAME = DIROUT
1177:      MEMBER = 'OPENING'
1178:      call PDSIO( 'OPEN', DIROUT, MEMBER, MEMBER, A, 3, NOUT1 )
1179: C
1180: C2005      do 820 LOP = 1, NEP1
1181:      do 820 LOP = 1, NOWSTP
1182: C
1183:      MEMBER = CASEID// 'HT' // STEPNM(LOP,0)
1184: C
1185: C .. Contents of CASE+'HT'+step are packed by "PACKET" series of
1186: C routines.
1187: C
1188:      call PACK00( IARRAY, MEMBER, MEMORY, IRET )
1189: C
1190: C .. pack STEP#, NMAT & NTNUC
1191: C
1192:      call PACKND( IARRAY, 'STEP#', 'I4', LOP, 1, IRET )
1193:      call PACKND( IARRAY, 'NMAT', 'I4', NMAT, 1, IRET )
1194:      call PACKND( IARRAY, 'NTNUC', 'I4', NTNUC, 1, IRET )
1195: C
1196:      call PACKLB( IARRAY, 'HMINV', 'HMINV', IRET )
1197:      call PACKND( IARRAY, 'HMINV', 'R4', HMINV(1,LOP), NMAT,
1198:      &          IRET1 )
1199:      if ( IRET1.ne.0 ) go to 830
1200: C
1201:      call PACKLB( IARRAY, 'EXPSZN', 'EXPSZN', IRET )
1202:      call PACKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1,LOP), NMAT,
1203:      &          IRET1 )
1204:      if ( IRET1.ne.0 ) go to 830
1205: C
1206:      call PACKLB( IARRAY, 'DMWZON', 'DMWZON', IRET )
1207:      call PACKND( IARRAY, 'DMWZON', 'R4', DMWZON(1,LOP), NMAT,
1208:      &          IRET1 )
1209:      if ( IRET1.ne.0 ) go to 830
1210: C
1211:      call PACKLB( IARRAY, 'DNBURN', 'DNBURN', IRET )
1212:      call PACKND( IARRAY, 'DNBURN', 'R4', DENTAB(1,1,LOP), NTNUC*
1213:      &          NMAT, IRET1 )
1214:      if ( IRET1.ne.0 ) go to 830
1215: C
1216:      call PACKLB( IARRAY, MEMBER, 'INTEGER', IRET )
1217:      NPK = 10
1218:      call PACKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
1219:      if ( IRET1.ne.0 ) go to 830
1220:      call ICLEA( IARRAY(LP), NPK, 0 )
1221:      IARRAY(LP) = NHIST(LOP)
1222:      IARRAY(LP+1) = NBATCH(LOP)
1223: C
1224:      call PACKLB( IARRAY, MEMBER, 'REAL', IRET )
1225: C2005 ... modified
1226: Cmod      NPK = 20
1227:      NPK = 30
1228: C2005 ... end
1229:      call PACKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
1230:      if ( IRET1.ne.0 ) go to 830
1231:      call CLEA( RARRAY(LP), NPK, 0.0 )
1232:      RARRAY(LP) = AKEFF(LOP)
1233:      RARRAY(LP+1) = ERRKEF(LOP)
1234:      RARRAY(LP+2) = DAYS(LOP)
1235:      RARRAY(LP+3) = U235F(LOP)

```

```

1236:      RARRAY(LP+4) = EXPST(LOP)
1237:      RARRAY(LP+5) = CUMMWD(LOP)
1238:      RARRAY(LP+6) = INSCR(LOP)
1239:      RARRAY(LP+7) = INTCR(LOP)
1240:      RARRAY(LP+8) = EINSR(LOP)
1241:      RARRAY(LP+9) = EINTCR(LOP)
1242:      RARRAY(LP+10) = FLXNRM(LOP)
1243:      RARRAY(LP+11) = FACNRM(LOP)
1244:      RARRAY(LP+12) = FISABS(LOP)
1245:      RARRAY(LP+13) = FRTCAP(LOP)
1246:      RARRAY(LP+14) = EFISAB(LOP)
1247:      RARRAY(LP+15) = EFRTCA(LOP)
1248:      RARRAY(LP+16) = FDECAY(LOP)
1249:      RARRAY(LP+17) = CDECAY(LOP)
1250: C2005 ... added
1251:      RARRAY(LP+18) = ERRNRM(LOP)
1252:      RARRAY(LP+19) = POWERW(LOP)
1253:      RARRAY(LP+20) = POWERE(LOP)
1254:      RARRAY(LP+21) = RMONIT(LOP)
1255:      RARRAY(LP+22) = EMONIT(LOP)
1256: C2005 ... end
1257: C
1258:      call PACKLB( IARRAY, 'power/zone', 'POWRZN', IRET )
1259:      call PACKND( IARRAY, 'POWRZN', 'R4', POWRZN(1,LOP), NMAT,
1260:      &          IRET1 )
1261:      if ( IRET1.ne.0 ) go to 830
1262: C
1263:      call PACKLB( IARRAY, 'GMAVG', 'GMAVG', IRET )
1264:      call PACKND( IARRAY, 'GMAVG', 'R4', GAMAVG(1,LOP), NMAT,
1265:      &          IRET1 )
1266:      if ( IRET1.ne.0 ) go to 830
1267: C
1268:      call PACKLB( IARRAY, 'YLDXE5', 'YLDXE5', IRET )
1269:      call PACKND( IARRAY, 'YLDXE5', 'R4', YLDXE5(1,LOP), NMAT,
1270:      &          IRET1 )
1271:      if ( IRET1.ne.0 ) go to 830
1272: C
1273:      call PACKLB( IARRAY, 'YLDI35', 'YLDI35', IRET )
1274:      call PACKND( IARRAY, 'YLDI35', 'R4', YLDI35(1,LOP), NMAT,
1275:      &          IRET1 )
1276:      if ( IRET1.ne.0 ) go to 830
1277: C
1278:      call PACKLB( IARRAY, 'YLDISM9', 'YLDISM9', IRET )
1279:      call PACKND( IARRAY, 'YLDISM9', 'R4', YLDISM9(1,LOP), NMAT,
1280:      &          IRET1 )
1281:      if ( IRET1.ne.0 ) go to 830
1282: C
1283:      call PACKLB( IARRAY, 'YLDPM9', 'YLDPM9', IRET )
1284:      call PACKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOP), NMAT,
1285:      &          IRET1 )
1286:      if ( IRET1.ne.0 ) go to 830
1287: C
1288:      call PACKLB( IARRAY, 'RATMIC', 'RATMIC', IRET )
1289:      do 810 M = 1, NMAT
1290:      call PACKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,M,LOP),
1291:      &          2*4*NTNUC, IRET1 )
1292:      810 continue
1293: C
1294:      call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
1295: C
1296: C .... Write PDS member CASE+'HT'+step-ID
1297: C
1298:      call PDSIO( 'WRITE', DIROUT, MEMBER, MEMBER, RARRAY, LENG,
1299:      &          NOUT1 )
1300: C

```

src/mvpburn/averg2.f

```
1301:      820      continue
1302:          end if
1303: C
1304: C *** END OF PROCESS
1305: C
1306:          return
1307: C
1308:      830 write(NOUT1,7000)
1309:          write(NOUT1,*) ' error in making data packet as ',
1310:          &          ' contents of member <', MEMBER, '> code =', IRET
1311:          stop 999
1312: C
1313: 7000 format(///' XXX(AVERG2) ERROR-STOP :')
1314: C
1315:          end
```

src/mvpburn/averg3.f

```
1:      subroutine AVERG3( DATA1, DATA2, ODATA, LENG,  FACT,  ITYPE )
2:      C
3:      C
4:      C
5:      real DATA1(LENG), DATA2(LENG), ODATA(LENG)
6:      C
7:      C *** START OF PROCESS
8:      C
9:      if ( ITYPE.eq.1 ) then
10:         do 100 I = 1, LENG
11:            ODATA(I) = ODATA(I) + FACT*DATA1(I)
12:         100 continue
13:         end if
14:      C
15:      if ( ITYPE.eq.2 ) then
16:         do 110 I = 1, LENG
17:            SAVE = DATA1(I)*FACT
18:            ODATA(I) = ODATA(I) + SAVE*SAVE
19:         110 continue
20:         end if
21:      C
22:      if ( ITYPE.eq.3 ) then
23:         do 120 I = 1, LENG
24:            SAVE = DATA1(I)*DATA2(I)*FACT
25:            ODATA(I) = ODATA(I) + SAVE*SAVE
26:         120 continue
27:         end if
28:      C
29:      return
30:      end
```

src/mvpburn/averg4.f

```
1:      subroutine AVERG4( DATA1, DATA2, ODATA, LENG,  FACT,  ITYPE )
2: C
3:      real DATA1(LENG), DATA2(LENG), ODATA(LENG)
4:      real FACT(LENG)
5: C
6: C *** START OF PROCESS
7: C
8:      if ( ITYPE.eq.1 ) then
9:          do 100 I = 1, LENG
10:             ODATA(I)  = ODATA(I) + FACT(i)*DATA1(I)
11:          100 continue
12:      end if
13: C
14:      if ( ITYPE.eq.2 ) then
15:          do 110 I = 1, LENG
16:             SAVE      = DATA1(I)*FACT(i)
17:             ODATA(I)  = ODATA(I) + SAVE*SAVE
18:          110 continue
19:      end if
20: C
21:      if ( ITYPE.eq.3 ) then
22:          do 120 I = 1, LENG
23:             SAVE      = DATA1(I)*DATA2(I)*FACT(I)
24:             ODATA(I)  = ODATA(I) + SAVE*SAVE
25:          120 continue
26:      end if
27: C
28:      return
29:      end
```

src/mvpburn/avgin0.f

```

1:      subroutine AVGIN0( MODE, T0,      ARRAY, MEMORY,TRGNAM,IDENT,
2: C2005&      MAXTRG,MAXISO,NOUT1, NOUT2, DIROUT,CASEID )
3:      &      MAXTRG,MAXISO,NOUT1, NOUT2, DIROUT,CASEID,PDSIN )
4: C=====
5: C  Input & Control routine of $AVERAGE mode
6: C=====
7: C
8:      include 'INC/_AVERG'
9: CKSK  PARAMETER( MAXCAS  = 16 )
10:      character*8 MODE
11:      real ARRAY(MEMORY)
12:      character*(*) DIROUT
13: C2005  ... added
14:      character*(*) PDSIN
15: C2005  .. end
16:      character*4 CASEID
17:      character*12 TRGNAM(MAXTRG)
18: C2003
19:      character*12 IDENT(MAXISO,MAXTRG)
20:      common /NAMEIO/  NOUT6,  NDTLS
21: C
22: C  --- array sizes -----
23: C
24:      include 'INC/_BURNP'
25: C
26: C  -- Common data -----
27: C
28:      include 'INC/_CBURNC1'
29:      include 'INC/_CBURNC2'
30:      include 'INC/_CBURNC3'
31:      common /PDSPDS/  IPDSCK
32: C
33:      character*2 STEPNM
34:      external STEPNM
35: C
36: C .... local variables .....
37: C
38:      character*72 DIRINS(MAXCAS)
39: C
40:      character*4 CASEIN(MAXCAS)
41: C2005 character*8 IDATE
42: C2005 character*8 STIME
43: C
44:      character*72 VNAME
45: C
46: C2005 character*128 CWRK
47: C2005 character*2 SID0, SID1, SID2
48: C
49: C -----
50: C
51:      NIN      = NDTLS
52:      NOUT6    = NOUT1
53: C
54:      NCASE    = 0
55:      CASEID   = ' '
56: C2003 MODEAV  = 1
57:      MODEAV   = 0
58:      TITLE(1) = ' '
59:      TITLE(2) = ' '
60:      IDEBG    = 0
61: CCC  dirout = ' '
62:      do 100 I = 1, MAXCAS
63: C2005  DIRINS(I) = ' '
64:      DIRINS(I) = PDSIN
65:      CASEIN(I) = ' '

```

```

66:      100 continue
67: C
68:      JSAVE    = 1
69: C
70:      IPDSCK   = 1
71:      NERR     = 0
72: C
73: C -----
74: C  Read data in $AVERAGE block
75: C -----
76: C
77:      110 continue
78: C
79:      call FREADS( VNAME, NLEN, IEND )
80:      if ( IEND.ne.0 ) then
81:          write(NOUT1,6888)
82:          write(NOUT1,*)
83:          &      ' unexpected end of input in $AVERAGE block'
84:          write(NOUT1,*) ' >>>> Probably no "$END AVERAGE" line.'
85:          stop 888
86:      end if
87: C
88: C ..... TITLES .....
89: C
90:      if ( VNAME(1:NLEN).eq.'TITLE1' ) then
91:          call CHREAD( VNAME(1:NLEN), TITLE(1), ' ', INLEN, ITERM, IEND )
92:          if ( ITERM.eq.0 ) then
93:              write(NOUT1,6888)
94:              write(NOUT1,7100) VNAME(1:NLEN)
95:              stop 888
96:          end if
97:          if ( IEND.ne.0 ) go to 150
98: C
99:      else if ( VNAME(1:NLEN).eq.'TITLE2' ) then
100:          call CHREAD( VNAME(1:NLEN), TITLE(2), ' ', INLEN, ITERM, IEND )
101:          if ( ITERM.eq.0 ) then
102:              write(NOUT1,6888)
103:              write(NOUT1,7100) VNAME(1:NLEN)
104:              stop 888
105:          end if
106:          if ( IEND.ne.0 ) go to 150
107: C
108: C ..... CASEID .....
109: C
110:      else if ( VNAME(1:NLEN).eq.'CASEID' ) then
111:          call CHREAD( VNAME(1:NLEN), CASEID, ' ', INLEN, ITERM, IEND )
112:          if ( ITERM.eq.0 ) then
113:              write(NOUT1,6888)
114:              write(NOUT1,7100) VNAME(1:NLEN)
115:              stop 888
116:          end if
117:          if ( IEND.ne.0 ) go to 150
118: C
119: C ..... PDS or PDSOUT .....
120: C
121:      else if ( VNAME(1:NLEN).eq.'PDS' .or. VNAME(1:NLEN).eq.'PDSOUT' )
122:          &      then
123:          call CHREAD( VNAME(1:NLEN), DIROUT, ' ', INLEN, ITERM, IEND )
124:
125:          if ( ITERM.eq.0 ) then
126:              write(NOUT1,6888)
127:              write(NOUT1,7100) VNAME(1:NLEN)
128:              stop 888
129:          end if
130:          if ( IEND.ne.0 ) go to 150

```


src/mvpburn/avgin0.f

```

131: C
132: C ..... PDSCHK .....
133: C
134:     else if ( VNAME(1:NLEN).eq.'PDSCHK' ) then
135:         call I4READ( VNAME(1:NLEN), IPDSCH, NA, 1, IEND )
136:         if ( IPDSCK.le.0 ) IPDSCK = 0
137:         if ( IEND.ne.0 ) go to 150
138: C
139: C ..... CASEIN .....
140: C
141:     else if ( VNAME(1:NLEN).eq.'CASEIN' ) then
142:         NN = 0
143: 120 call CHREAD( VNAME(1:NLEN), CASEIN(MIN(MAXCAS,NN+1)), ' ',
144: & INLEN, ITERM, IEND )
145:         if ( IEND.ne.0 ) go to 150
146:         NN = NN + 1
147:         if ( ITERM.eq.0 ) go to 120
148:
149:         if ( NCASE.eq.0 ) then
150:             NCASE = NN
151:         else if ( NN.ne.NCASE ) then
152:             write(NOUT1,6888)
153:             write(NOUT1,*) ' number of data in <',
154: & VNAME(1:NLEN),
155: & ' does not match number of cases determined by other input.'
156:             call CNTERR('FATAL')
157:         end if
158: C
159: C ..... PDSIN .....
160: C
161:     else if ( VNAME(1:NLEN).eq.'PDSIN' ) then
162:         NN = 0
163: 130 call CHREAD( VNAME(1:NLEN), DIRINS(MIN(MAXCAS,NN+1)), ' ',
164: & INLEN, ITERM, IEND )
165:         if ( IEND.ne.0 ) go to 150
166:         NN = NN + 1
167:         if ( ITERM.eq.0 ) go to 130
168:
169:         if ( NCASE.eq.0 ) then
170:             NCASE = NN
171:         else if ( NN.ne.NCASE ) then
172:             write(NOUT1,6888)
173:             write(NOUT1,*) ' number of data in <',
174: & VNAME(1:NLEN),
175: & ' does not match number of cases determined by other input.'
176:             call CNTERR('FATAL')
177:         end if
178:         if ( IEND.ne.0 ) go to 150
179: C
180: C ..... MODEAVG .....
181: C
182:     else if ( VNAME(1:NLEN).eq.'MODEAVG' ) then
183:         call I4READ( VNAME(1:NLEN), MODEAV, NA, 1, IEND )
184:         if ( IEND.ne.0 ) go to 150
185: C
186: C ..... SAVEPDS ..... (save averaged data as PDS members)
187: C
188:     else if ( VNAME(1:NLEN).eq.'SAVEPDS' ) then
189:         call I4READ( VNAME(1:NLEN), JSAVE, NA, 1, IEND )
190:         if ( IEND.ne.0 ) go to 150
191: C
192: C ..... PRINT OR IDEBG .....
193: C
194:     else if ( VNAME(1:NLEN).eq.'IDEBG'
195: > .or. VNAME(1:NLEN).eq.'PRINT' ) then

```

```

196:         call I4READ( VNAME(1:NLEN), IDEBG, NA, 1, IEND )
197:         if ( IEND.ne.0 ) go to 150
198: C
199: C ..... $END AVERAGE .....
200: C
201:     else if ( VNAME(1:NLEN).eq.'$END' ) then
202:         call RESET
203:         go to 140
204: C
205: C ..... unknown data .....
206: C
207:     else
208:         write(NOUT1,6888)
209:         write(NOUT1,7080) VNAME(1:NLEN)
210:         call CNTERR('FATAL')
211:         call DMREAD( VNAME(:NLEN), NA, NB, IEND )
212:     end if
213: C
214: go to 110
215: C
216: 140 continue
217: call CHKERR( 'WARNING', KK )
218: if ( KK.gt.0 ) then
219:     write(NOUT1,7125)
220: 7125 format(' *** some WARNING messages have been issued. ****'/
221: & ' I recommend you checking your input data',
222: & ' once more.'/ ' WARNING message is printed ',
223: & 'with "!!!" symbol on the head of printed line.'/)
224: end if
225: call CHKERR( 'FATAL', KK )
226: C
227: if ( KK.gt.0 ) then
228:     write(NOUT1,7145)
229: 7145 format(' *** fatal ERRORS have been detected',
230: & ' in input-phase. stop execution. ***'/
231: & ' message for fatal ERROR is printed ',
232: & 'with "XXX" symbol on the head of printed line.'/
233: & ' you must check your input data', ' carefully !!')
234: stop 888
235: end if
236: if ( CASEID.eq.' ' ) then
237:     write(NOUT1,6888)
238:     write(NOUT1,*)
239: & ' no specification of CASEID ',
240: & 'in ', MODE(:ICLEN2(MODE)), ' block.'
241:     call CNTERR('FATAL')
242: end if
243: do 160 I = 1,NCASE
244:     if ( DIRINS(I).eq.' ' ) then
245:         write(NOUT1,6888)
246:         write(NOUT1,*)
247: > ' no specification ',I,'-th input pds directory'
248:         call CNTERR('FATAL')
249:     end if
250: 160 continue
251: do 170 I = 1,NCASE
252:     if ( CASEIN(I).eq.' ' ) then
253:         write(NOUT1,6888)
254:         write(NOUT1,*)
255: > ' no specification ',I,'-th input case name'
256:         call CNTERR('FATAL')
257:     end if
258: 170 continue
259: C
260: if ( JSAVE.eq.1 .and. DIROUT.eq.' ' ) then

```

src/mvpburn/avgin0.f

```

261:      write(NOUT1,6888)
262:      write(NOUT1,*)
263:      &      ' no specification of output PDS ',
264:      &      ' in ', MODE(:ICLEN2(MODE)), ' block.'
265:      call CNTERR('FATAL')
266:      end if
267: C
268:      if ( NCASE.eq.0 ) then
269:      write(NOUT1,6888)
270:      write(NOUT1,*)
271:      &      ' no averaging case is specified.'
272:      call CNTERR('FATAL')
273:      end if
274: C2003
275:      if ( NCASE.eq.1 ) then
276:      write(NOUT1,6888)
277:      write(NOUT1,*)
278:      &      ' Only 1 case is specified for averaging mode.'
279:      call CNTERR('FATAL')
280:      end if
281: C2003 ... end
282: C
283:      if ( NCASE.gt.MAXCAS ) then
284:      write(NOUT1,6888)
285:      write(NOUT1,*)
286:      &      ' number of averaging cases ',
287:      &      ' exceeds program limit (MAXCAS= ', MAXCAS, ') '
288:      write(NOUT1,*) ' change parameter value in include file'
289:      call CNTERR('FATAL')
290:      end if
291: C
292: C2003 if ( MODEAV.le.0 .or. MODEAV.gt.2 ) then
293:      if ( MODEAV.lt.0 .or. MODEAV.gt.1 ) then
294:      write(NOUT1,6888)
295:      write(NOUT1,*)
296:      &      ' invalid averaging mode ', MODEAV,
297:      &      ' is specified.'
298:      call CNTERR('FATAL')
299:      end if
300: C
301:      call CHKERR( 'FATAL', KK )
302:      if ( KK.gt.0 ) then
303:      write(NOUT1,7145)
304:      stop 888
305:      end if
306: C
307: C
308:      call CPUTM( T00 )
309:      call AVERG0( MEMORY, NOUT1, NOUT2, T00, DIROUT, CASEID, DIRINS,
310:      &      CASEIN, NCASE, IDEBG, MODEAV, JSAVE, ARRAY, ARRAY, TRGNAM,
311:      &      IDENT, MAXTRG, MAXISO, NIN )
312: C
313:      return
314: C
315: C ----- Error -----
316: C
317:      150 write(NOUT1,6888)
318:      write(NOUT1,7120) VNAME(:NLEN)
319:      stop 888
320: C
321:      6888 format( //' XXX(AVGIN0) ERROR-STOP : ' )
322:      7080 format(1X,' unknown data name <',A,
323:      &      '>. Skipping ...')
324:      7100 format(1X,'string data <',A,'> has no closing ") " ?'/1X,
325:      &      'or quotation mark is missing.')
326:      7120 format(1X,'end of input data in reading data <',A,
327:      &      '> in $AVERAGE block.')
328: C
329:      end

```

src/mvpburn/burn0.f

```

1:      subroutine BURN0( MEMORY,NOUT1, NOUT2, T00,   DIRIN, DIROUT,A,
2:      &                TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
3:      &                CASEID,SID0,SID1,SID2,JPC,ISPC,TITL2,
4: C2005&                CASREF )
5:      &                CASREF,IBSTEP )
6: C=====
7: C Perform burnup calculation from step SID1 to SID2.
8: C ISPC is substep # (1,2) for PC method ( JPC .ne. 0 )
9: C=====
10:     character*(*) DIRIN, DIROUT
11:     character*72 TITL2
12:     character*4 CASEID,CASREF
13:     character*2 SID0, SID1, SID2
14: C
15:     real A(MEMORY)
16:     character*12 TRGNAM(MAXTRG)
17: C2003
18:     character*12 IDENT(MAXISO,MAXTRG)
19: C2003
20:     character*12 IDWORK(MAXISO,MAXTRG)
21: C
22:     include 'INC/_BURNP'
23: C
24:     include 'INC/_CBURNC1'
25:     include 'INC/_CBURNC2'
26: C
27:     real INSCR, INTCR
28:     common /MEMOCK/ MEUSED
29:     common /BNREST/ RSDUMY(3),   NOWSTP, IBEND,  AKEFF(MXSTEP),
30: &      ERKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
31: &      EXPST(MXSTEP),  CUMMWD(MXSTEP), INSCR(MXSTEP),
32: &      INTCR(MXSTEP),  EINSCR(MXSTEP), EINTCR(MXSTEP),
33: &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
34: &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
35: &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
36: &      NBATCH(MXSTEP)
37: C2005
38: &      ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
39: &      RMONIT(MXSTEP), EMONIT(MXSTEP)
40: Cend
41: C
42:     character*72 DDNAME
43:     character*8 MEMBER
44: C2005 ... added
45:     integer IBSTEP(MXSTEP)
46: C2005 ... end
47: C
48: C---- READ PDS
49: C
50:     call RWCOM1('READ', DIRIN, CASEID )
51:     call RWCOM2('READ', DIRIN, CASEID )
52:     call RWCOM3('READ', DIRIN, CASEID )
53: Ctemp
54: Cm write(NOUT1,*) ' >> IBC IBEDIT in sub(BURN0) : '
55: Cm write(NOUT1,'(10I6)') IBC,IBEDIT
56: Cend
57: C00/01/24
58: C Skip read restart file in BRANCH mode
59: if (IVOID.eq.0) then
60:     call RWREST('READ', DIRIN, CASEID )
61: end if
62: C
63: write(NOUT1,7000) NOWSTP, SID1
64: if (IVOID.eq.1) then
65:     write(NOUT1,7010) CASREF
66: end if
67: C
68: C *** SET VAIRABLE DIMENSION
69: C
70:     LOC01 = 1
71: C --- MTYP
72:     LOC02 = LOC01 + NMAT
73: C --- NISO
74:     LOC03 = LOC02 + NMAT
75: C --- TEMP
76:     LOC04 = LOC03 + NMAT
77: C --- VOLM
78:     LOC05 = LOC04 + NMAT
79: C --- TRGNAM
80: Ckuni 2003/12/21 New use of LOC05 for KPBURN(NTNUC,NMAT)
81: Cmod LOC06 = LOC05
82:     LOC06 = LOC05 + NTNUC*NMAT
83: C --- LENTRG
84:     LOC07 = LOC06 + NMAT
85: C --- MTBURN
86:     LOC08 = LOC07 + NMAT*3
87: C --- MTCARD
88:     LOC09 = LOC08 + NMAT*2
89: C --- IDENT
90: Ckuni 2003/12/21 New use of LOC09 for KZEROP(NCHA,NTNUC)
91: Cmod LOC10 = LOC09
92:     LOC10 = LOC09 + NCHA*NTNUC
93: C --- DNINIT
94:     LOC11 = LOC10 + KNMAX*NMAT
95: C --- MATMVP
96:     LOC12 = LOC11 + NMAT
97: C --- IPBURN
98:     LOC13 = LOC12 + KNMAX*NMAT
99: C --- MVPDAT
100: C2005 LOC14 = LOC13
101: C --- MATREG
102:     LOC14 = LOC13 + NMAT
103: Cend
104: C --- NUCLP
105:     LOC15 = LOC14 + NPAR*NTNUC
106: C --- GAM
107:     LOC16 = LOC15 + NTFISS*NTNUC
108: C --- NBIC
109:     LOC17 = LOC16 + NPAR*NTNUC
110: C --- PBIC
111:     LOC18 = LOC17 + NPAR*NTNUC
112: C --- GAMI
113:     LOC19 = LOC18 + NTFISS*NTNUC
114: C --- KSTP
115:     LOC20 = LOC19 + NCHA*NTNUC
116: C --- LONG
117:     LOC21 = LOC20 + NCHA*NTNUC
118: C --- LL
119:     LOC22 = LOC21 + NCHA*NTNUC
120: C --- IP
121:     LOC23 = LOC22 + LCHA*NCHA*NTNUC
122: C --- KP
123:     LOC24 = LOC23 + LCHA*NCHA*NTNUC
124: C --- NPN
125:     LOC25 = LOC24 + NPAR*NTNUC
126: C --- PHAI
127:     LOC26 = LOC25 + NPAR*NTNUC
128: C --- DNOLD
129:     LOC27 = LOC26 + NTNUC*NMAT
130: C --- DNNEW

```

src/mvpburn/burn0.f

```

131:      LOC28  = LOC27 + NTNUC*NMAT
132: C --- DNWRK1
133:      LOC29  = LOC28 + NTNUC*NMAT
134: C --- DNWRK2
135:      LOC30  = LOC29 + NTNUC*NMAT
136: C --- IDWORK
137:      LOC31  = LOC30
138: C --- PHI
139:      LOC32  = LOC31 + NMAT
140: C --- WRKMAT
141:      LOC33  = LOC32 + NMAT
142: C --- POWZM1
143:      LOC34  = LOC33 + NMAT
144: C --- POWZM2
145:      LOC35  = LOC34 + NMAT
146: C --- SSF
147:      LOC36  = LOC35 + NTNUC*NMAT
148: C --- SSC
149:      LOC37  = LOC36 + NTNUC*NMAT
150: C --- SSN2N
151: C ***** LOCATION FOR CASEBNUP *****
152:      LOC38  = LOC37 + NTNUC*NMAT
153: C --- POWRZN
154:      LOC39  = LOC38 + NMAT*NEP1
155: C --- EXPSZN
156:      LOC40  = LOC39 + NMAT*NEP1
157: C --- HMINV
158:      LOC41  = LOC40 + NMAT*NEP1
159: C --- DMWZON
160:      LOC42  = LOC41 + NMAT*NEP1
161: C --- GAMAVG
162:      LOC43  = LOC42 + NMAT*NEP1
163: C --- YLDXE5
164:      LOC44  = LOC43 + NMAT*NEP1
165: C --- YLDI35
166:      LOC45  = LOC44 + NMAT*NEP1
167: C --- YLDSM9
168:      LOC46  = LOC45 + NMAT*NEP1
169: C --- YLDPM9
170:      LOC47  = LOC46 + NMAT*NEP1
171: C --- DENTAB
172:      LOC48  = LOC47 + NTNUC*NMAT*NEP1
173: C --- RATMIC
174:      LOC49  = LOC48 + 2*4*NTNUC*NMAT*NEP1
175: C --- RPCMIC
176: C2005 LOC50  = LOC49 + 2*4*NTNUC*NMAT*NEP1
177:      LOC50  = LOC49 + 2*4*NTNUC*NMAT
178: C --- CUMWD1
179:      LOC51  = LOC50 + NMAT
180: C --- CUMWD2
181:      LAST   = LOC51 + NMAT
182:      ISW    = LAST - (LAST/2)*2
183:      if ( ISW.eq.0 ) LAST   = LAST + 1
184: C -----
185:      if ( LAST.gt.MEUSED) MEUSED = LAST
186:      LEMORY  = MEMORY - LAST + 1
187: C
188:      if ( LEMORY.lt.0 ) then
189:        write(NOUT1,6888)
190:        write(NOUT1,*) ' memory over'
191:        write(NOUT1,*) ' requested memory size = ', LAST, ' words.'
192:        write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
193:        write(NOUT1,*) ' change parameter value in include file'
194:        stop 999
195:      end if

```

```

196: C
197:      if ( NMAT.gt.MAXTRG ) then
198:        write(NOUT1,6888)
199:        write(NOUT1,*) ' too many materials'
200:        write(NOUT1,*) ' requested number of materials = ',NMAT
201:        write(NOUT1,*) ' limit for number of tally regions(MAXTRG)',
202:          & ' =',MAXTRG
203:        write(NOUT1,*) ' change parameter value in include file'
204:        stop 999
205:      end if
206: C
207:      call CLEA( A, LAST, 0.0 )
208: C
209:      call BURNUP( NOUT1, NOUT2, LEMORY, DIRIN, DIROUT,
210:        & CASEID, SID0, SID1, SID2, JPC, ISPC,
211:        & TITL2, A(LOC01), A(LOC02), A(LOC03), A(LOC04),
212:        & TRGNAM , A(LOC06), A(LOC07), A(LOC08), IDENT ,
213:        & A(LOC10), A(LOC11), A(LOC12), A(LOC14),
214:        & A(LOC15), A(LOC16), A(LOC17), A(LOC18), A(LOC19),
215:        & A(LOC20), A(LOC21), A(LOC22), A(LOC23), A(LOC24),
216:        & A(LOC25), A(LOC26), A(LOC27), A(LOC28), A(LOC29),
217:        & IDWORK , A(LOC31), A(LOC32), A(LOC33), A(LOC34),
218:        & A(LOC35), A(LOC36), A(LOC37), A(LOC38), A(LOC39),
219:        & A(LOC40), A(LOC41), A(LOC42), A(LOC43), A(LOC44),
220:        & A(LOC45), A(LOC46), A(LOC47), A(LOC48), A(LOC49),
221:        & Ckuni 2003/12/21 LOC05(KPBURN) & LOC09(KZEROP) Arrays are added
222:        & Cmod & A(LOC50), A(LOC51), A(LAST), A(LAST), CASREF )
223:        & A(LOC50), A(LOC51), A(LAST), A(LAST), CASREF ,
224:        & C2005& A(LOC05), A(LOC09) )
225:        & A(LOC05), A(LOC09), A(LOC13), IBSTEP )
226: Cend
227: C
228:      call CPUTM2( T1 )
229:      if (IBEDIT .gt. 0) then
230:        write(NOUT1,7020) (T1 - T00)/60.0
231:      end if
232: C
233:      6888 format(// ' XXX(BURN0) ERROR-STOP : ' )
234:      7000 format(//15X,'xxxxxxxxxxxxxxxxxxxxx'
235:        & /15X,'x DEPLET-MODE x PRESENT STEP NO. : ',I3,
236:        & ' (',A2,')'
237:        & /15X,'xxxxxxxxxxxxxxxxxxxxx'/ )
238:      7010 format(1X,'>>> NO DEPLETION IN BRANCH-OFF MODE => ',
239:        & 'FUEL COMPOSITION READ FROM CASE ',A4 )
240:      7020 format(/1X,12X,'CPU ',E12.5,' MIN.')
241: C
242: C *** END OF PROCESS
243: C
244:      return
245:      end

```

src/mvpburn/burnc4.f

```

1:      subroutine BURNc4( PHIP, VOL, DELT, NMAT, NBOPT, NOUT1, NDN,
2:      &                  LNMAX, NMAX, NFIS, NPAR, EVTOJ, EFC, DN,
3:      &                  DNOLD, SSF, SSC, SSN2N, CUMWD, IDBG,
4:      &                  NOUT2, LCHA, NCHA, MTBURN, NUCLP, GAM, GAMI,
5:      &                  NBIC, PBIC, KSTP, LONG, LL, IP, KP,
6:      &                  NPN, PHAI, IBC2, ST235, TIMEUS, IDU235,
7:      &                  FACPOW, POWZN1, POWZN2, NUCL, NCH, RAMDA,
8:      &                  FACT2N, IBC, KPBURN, KZEROP, BZERO, MATREG)
9: C=====
10: C      BURNUP DENSITY CALCULATION USING D-CHAIN METHOD
11: C      THIS ROUTINE IS SPECIFIED REGION POWER CONSTANTS MODE
12: C      MATREG IS SPECIFIED REGION FLAG
13: C      MATREG(M)=0 : M-th material is not contained in SPECIFIED REGION
14: C      MATREG(M)=1 : M-th material is contained in SPECIFIED REGION
15: C
16: C      NBOPT = 0 : normal burnup
17: C      NBOPT = 1 : cooling (zero power)
18: C=====
19:      real PHIP(NMAT)
20:      real VOL(NMAT)
21:      real POWZN1(NMAT)
22:      real POWZN2(NMAT)
23: C
24:      real EFC(2, LNMAX)
25:      real DN(NMAX, NMAT)
26:      real DNOLD(NMAX, NMAT)
27:      real SSF(NMAX, NMAT)
28:      real SSC(NMAX, NMAT)
29:      real SSN2N(NMAX, NMAT)
30:      real CUMWD(NMAT)
31:      integer MTBURN(3, NMAT)
32: C
33:      integer NUCLP(NPAR, NMAX)
34:      real GAM(NFIS, NMAX)
35:      real GAMI(NFIS, NMAX)
36: C
37:      integer NBIC(NPAR, NMAX)
38:      real PBIC(NPAR, NMAX)
39:      integer KSTP(NCHA, NMAX)
40:      integer LONG(NCHA, NMAX)
41:      integer LL(NCHA, NMAX)
42:      integer IP(LCHA, NCHA, NMAX)
43:      integer KP(LCHA, NCHA, NMAX)
44:      integer NPN(NPAR, NMAX)
45:      real PHAI(NPAR, NMAX)
46: C
47:      integer KZEROP(NCHA, NMAX), KPBURN(NMAX, NMAT)
48: C
49: C      PARAMETER ( MXSTEP = 100, MXNUC = 200, MXFISS = 50 )
50:      include 'INC/_BURNP'
51: C
52:      real RAMDA(MXNUC)
53:      integer NCH(MXNUC)
54:      integer NUCL(MXNUC)
55:      real FACT2N(MXNUC)
56: C
57:      integer IBC(20)
58:      integer MATREG(NMAT)
59: C
60: C      .... local arrays ....
61: C
62:      integer NOL(MXNUC)
63:      real GAMMA(MXNUC)
64:      real GAMX(MXNUC)
65:      real PHIC(MXNUC)

```

```

66:      real PHIL(MXNUC)
67:      real R(MXNUC)
68:      real G(MXNUC)
69:      real P(MXNUC)
70:      real RAM(MXNUC)
71:      integer MAP(MXNUC)
72:      real*8 DEX(MXNUC)
73:      real*8 DEXX(MXNUC)
74:      real*8 DCOEF(MXNUC)
75:      real*8 TSTEP
76:      real*8 TTSTEP
77: C
78:      real SGF(MXNUC)
79:      real SGA(MXNUC)
80:      real SGC(MXNUC)
81:      real SGN2N(MXNUC)
82:      real FNN(MXNUC)
83:      real FNF(MXNUC)
84: C
85:      real FU235(MXNUC)
86: C
87:      real*8 FLUX
88: C-----
89: C
90: C *** START PROCESS
91: C
92:      call CLEA( CUMWD, NMAT, 0.0 )
93: C
94:      if ( DELT.le.0.0 ) return
95: Ctemp
96:      if ( IDBG.ge.0 ) then
97:          write(NOUT1,*) ' ** NMAT (BURNc4) : ', NMAT
98:          write(NOUT1,*) ' ** MATREG is : '
99:          write(NOUT1, '(10I4)') (MATREG(M), M=1, NMAT)
100:      end if
101: Cend
102: C
103:      if ( NMAX.gt.MXNUC ) then
104:          write(NOUT1,6888)
105:          write(NOUT1,*) ' number of depleting nuclides ( ', NMAX,
106:      &                  ' ) is over reserved size( MXNUC= ', MXNUC, ' )'
107:          write(NOUT1,*) ' Change parameter value in include file.'
108:          stop 999
109:      end if
110: C
111:      ITYP = 1
112:      if ( NBOPT.eq.1 ) ITYP = 2
113: C
114:      call LNCHAI( NUCL, RAMDA, NCH, NUCLP, NBIC, PBIC, NOL, KSTP, LONG,
115:      &              LL, IP, KP, NPN, NMAX, NPAR, NCHA, LCHA, ITYP )
116: C
117: C-----+
118: C
119:      call CLEA( GAMI, NFIS*NMAX, 0.0 )
120:      call MEMOVE( GAM, GAMI, NFIS*NMAX )
121: C
122:      TSTEP = DELT*24.0*3600.0
123: C
124: C *** Cooling Case
125: C
126:      if ( NBOPT.ne.0 ) then
127:          KOOPT = 1
128:          do 140 KZ = 1, NMAT
129:              call ICLEA( MAP, MXNUC, 0 )
130:              MAPLEN = 0

```

src/mvpburn/burnc4.f

```

131:      do 100 I = MTBURN(1,KZ), MTBURN(2,KZ)
132:      if ( KPBURN(I,KZ).gt.0 ) then
133:          MAPLEN = MAPLEN + 1
134:          MAP(MAPLEN) = I
135:      endif
136: 100      continue
137: C
138:      do 110 J = 1, NMAX
139:          FNN(J) = DN(J,KZ)
140:          FNF(J) = 0.0
141:          SGF(J) = SSF(J,KZ)
142:          SGC(J) = SSC(J,KZ)
143:          SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
144:          SGA(J) = SGF(J) + SGC(J)
145: 110      continue
146: C
147: C **** COOLING *****
148: C
149:      FLUX = 0.0
150:      XM = 0.0
151: C
152:      do 120 II = 1, NMAX
153:          GAMX(II) = 0.0
154:          GAMMA(II) = 0.0
155: 120      continue
156: C
157:      IMAT = KZ
158:      call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
159:      & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
160:      & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
161:      & DCOEF, MAP, FLUX, TSTEP, XM, NMAX, NPAR, NCHA, LCHA,
162:      & MAPLEN, IMAT, IBC, KOOPT, KZEROP, IDBG, EZERO, NBOPT)
163: C=====
164: C
165:      do 130 J = 1, NMAX
166:          DN(J,KZ) = FNF(J)
167: 130      continue
168: 140      continue
169: C
170:      goto 9999
171:      end if
172: C
173: C *** BURN-UP
174: C
175:      call CLEA( FU235, MXNUC, 0.0 )
176:      FACPOW = 0.0
177: C
178:      if ( IDBG.gt.1 ) then
179:          do 170 L = 1, NMAX
180:              write(NOUT1,7000) L, NUCL(L), NCH(L), RAMDA(L)
181:              do 160 LN = 1, NCH(L)
182:                  write(NOUT1,7020) LN, NUCLP(LN,L), NBIC(LN,L), PBIC(LN,L)
183: 160          continue
184: 170          continue
185:          end if
186: C
187:      if ( IDBG.gt.2 ) then
188:          do 180 L = 1, NFIS
189:              write(NOUT1,7040) L, NUCL(L)
190:              write(NOUT1,7060) (GAM(L,LN),LN=1,NMAX)
191: 180          continue
192:          end if
193: C
194: 7000 format(' ## L NUCL NCH RAMDA ## ',3I8,1P,E12.5)
195: 7020 format(' ', ' << LN NUCLP NBIC PBIC >> ',3I8,F10.5)

```

```

196: 7040 format(' ', ' ## L NUCL (GAM)      ## ',2I8)
197: 7060 format(' ', ' << GAM >> ',1P,10E11.4)
198: C
199:      TEMP0 = 0.0
200:      ACOEF = EVTOJ*1.0000E+6
201: C
202:      SUMU5 = 0.0
203:      do 200 KZ = 1, NMAT
204: C
205:          if ( IDBG.gt.0 ) then
206:              write(NOUT1,*) ' ** KZ VOL PHIP ACOEF **', KZ, VOL(KZ),
207:      & PHIP(KZ), ACOEF
208:          end if
209: C
210:          SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
211: C
212:          if( MATREG(KZ).gt.0 ) then
213:              do 190 LN = 1, NMAX
214:                  TEMP0 = TEMP0 +
215:      & (SSF(LN,KZ)*EFC(1,LN)+SSC(LN,KZ)*EFC(2,LN))*VOL(KZ)*
216:      & PHIP(KZ)*DN(LN,KZ)*ACOEF
217: C
218:                  if ( IDBG.gt.1 ) then
219:                      write(NOUT1,*) ' ** KZ LN SSF EF SSC EC : ', KZ, LN,
220:      & SSF(LN,KZ), EFC(1,LN), SSC(LN,KZ), EFC(2,LN)
221:                  end if
222: C
223: 190          continue
224:          endif
225: 200      continue
226:      FU235(1) = 100.0*(ST235-SUMU5*1.000E+24) /ST235
227: C
228:      NLOOP = NDN
229:      SAVED = DELT/DBLE(NLOOP)
230:      if ( SAVED.lt.1.00 ) then
231:          NLOOP = DELT + 0.01
232:      end if
233:      if ( SAVED.gt.10.0 ) then
234:          NLOOP = DELT/10.000 + 0.01
235:      end if
236: C
237:      if ( NLOOP.lt.5 ) NLOOP = 5
238:      if ( NLOOP.gt.50 ) NLOOP = 50
239:      NLOOP1 = NLOOP - 1
240:      TTSTEP = TSTEP/DBLE(NLOOP)
241:      DELDAY = DELT/DBLE(NLOOP)
242: C
243:      if ( IDBG.gt.0 ) then
244:          write(NOUT1,*) ' ** TEMP0 NLOOP DELDAY (BURN4) : ', TEMP0,
245:      & NLOOP, DELDAY
246:      end if
247: C
248: C-----LOOP OF SUB-TIME STEP
249: C
250:      DAYSUM = 0.0
251:      FACFLX = 1.00000
252: C
253:      do 370 LOOPT = 1, NLOOP
254:          if ( TTSTEP.le.0.0 ) go to 380
255:          DAYSUM = DAYSUM + DELDAY
256: C-----FIRST LOOP OF MATERIAL ZONE
257: C-----LOOP OF MATERIAL ZONE
258:          TEMP1 = 0.0
259:          TEMP2 = 0.0
260:          KOOPT = LOOPT

```

src/mvpburn/burnc4.f

```

261: C
262:       do 270 KZ = 1, NMAT
263:       if( MATREG(KZ).gt.0) then
264:         VOLDEP = VOL(KZ)
265:         call ICLEA( MAP, MXNUC, 0 )
266:         MAPLEN = 0
267:         do 210 I = MTBURN(1,KZ), MTBURN(2,KZ)
268:           if ( KPBURN(I,KZ).gt.0) then
269:             MAPLEN = MAPLEN + 1
270:             MAP(MAPLEN) = I
271:           endif
272:         210 continue
273: C
274:         do 220 J = 1, NMAX
275:           FNN(J) = DN(J,KZ)
276:           FNF(J) = 0.0
277:           SGF(J) = SSF(J,KZ)
278:           SGC(J) = SSC(J,KZ)
279:           SGA(J) = SGF(J) + SGC(J)
280:           SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
281:         220 continue
282: C
283:         FLUX = PHIP(KZ)*1.0E-24*FACFLX
284:         XM = 0.0
285: C
286:         do 230 J = 1, NFIS
287:           XM = XM + SGF(J)*FNN(J)
288:         230 continue
289:         RXM = 0.0
290:         if ( XM.gt.0.0 ) RXM = 1.00000/XM
291: C
292:         do 250 II = 1, NMAX
293:           SUMGX = 0.0
294:           SUMGI = 0.0
295:           do 240 J = 1, NFIS
296:             SUMGX = SUMGX + GAM(J,II)*FNN(J)*SGF(J)
297:             SUMGI = SUMGI + GAMI(J,II)*FNN(J)*SGF(J)
298:           240 continue
299:           GAMX(II) = SUMGX*RXM
300:           GAMMA(II) = SUMGI*RXM
301:         250 continue
302: C
303:         XM = XM*FLUX
304: C
305: C2003 ... pass IMAT, IBC as an argument
306:         IMAT = KZ
307:         call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
308:           & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
309:           & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
310:           & DCOEF, MAP, FLUX, TTSTEP, XM, NMAX, NPAR, NCHA,
311:           & LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO,NBOPT)
312: C=====
313: C
314:         do 260 LN = 1, NMAX
315:           SAVE = (SGF(LN)*EFC(1, LN)+SGC(LN)*EFC(2, LN))*FACFLX
316:           TEMP1 = TEMP1 + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*ACOE
317:           TEMP2 = TEMP2 + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*ACOE
318:         260 continue
319:       endif
320:       270 continue
321: C
322: C-----SECOND LOOP OF MATERIAL ZONE
323: C
324:       FACFLX0 = FACFLX
325:       FACFLX = FACFLX*2.000*TEMP0/(TEMP1+TEMP2)

```

```

326: C
327: check ###
328: c       write(NOUT1,*) ' ## FACFLX ',FACFLX,' LOOPT ',LOOPT
329:       if ( IDBG.gt.0 ) then
330:         write(NOUT1,*)
331:         & ' **LOOPT TEMP1 TEMP2 FACFLX DAYSUM (BURN4) ** ',
332:         & LOOPT, TEMP1, TEMP2, FACFLX, DAYSUM
333:       end if
334:       call CLEA( POWZN1, NMAT, 0.0 )
335:       call CLEA( POWZN2, NMAT, 0.0 )
336: C-----LOOP OF MATERIAL ZONE
337:       TEMP3 = 0.0
338:       TEMP4 = 0.0
339:       KOOPT = LOOPT + 1
340: C
341:       do 340 KZ = 1, NMAT
342:         VOLDEP = VOL(KZ)
343:         call ICLEA( MAP, MXNUC, 0 )
344:         MAPLEN = 0
345:         do 280 I = MTBURN(1,KZ), MTBURN(2,KZ)
346:           if ( KPBURN(I,KZ).gt.0) then
347:             MAPLEN = MAPLEN + 1
348:             MAP(MAPLEN) = I
349:           endif
350:         280 continue
351: C
352:         do 290 J = 1, NMAX
353:           FNN(J) = DN(J,KZ)
354:           FNF(J) = 0.0
355:           SGF(J) = SSF(J,KZ)
356:           SGC(J) = SSC(J,KZ)
357:           SGA(J) = SGF(J) + SGC(J)
358:           SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
359:         290 continue
360: C
361:         FLUX = PHIP(KZ)*1.0E-24*FACFLX
362:         XM = 0.0
363: C
364:         do 300 J = 1, NFIS
365:           XM = XM + SGF(J)*FNN(J)
366:         300 continue
367: C
368:         RXM = 0.0
369:         if ( XM.gt.0.0 ) RXM = 1.00000/XM
370: C
371:         do 320 II = 1, NMAX
372:           SUMGX = 0.0
373:           SUMGI = 0.0
374:           do 310 J = 1, NFIS
375:             SUMGX = SUMGX + GAM(J,II)*FNN(J)*SGF(J)
376:             SUMGI = SUMGI + GAMI(J,II)*FNN(J)*SGF(J)
377:           310 continue
378:           GAMX(II) = SUMGX*RXM
379:           GAMMA(II) = SUMGI*RXM
380:         320 continue
381: C
382:         XM = XM*FLUX
383: C
384:         IMAT = KZ
385:         call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
386:           & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
387:           & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
388:           & DCOEF, MAP, FLUX, TTSTEP, XM, NMAX, NPAR, NCHA,
389:           & LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO,NBOPT)
390: CCHECK

```

src/mvpburn/burnc4.f

```

391:         if ( IDBG.gt.1 ) then
392:             if ( KZ.eq.1 ) then
393:                 write(NOUT1,7180) NMAX, TEMP0, FACFLX
394:                 write(NOUT1,7200) (EFC(1,LN),LN=1,NMAX)
395:                 write(NOUT1,7220) (EFC(2,LN),LN=1,NMAX)
396:             end if
397:             write(NOUT1,7080) LOOPT, KZ, PHIP(KZ), FLUX
398:             write(NOUT1,7120) SGF
399:             write(NOUT1,7140) SGC
400:             write(NOUT1,7100) SGA
401:             write(NOUT1,7160) SGN2N
402:             write(NOUT1,7240) (FNN(J),J=1,NMAX)
403:             write(NOUT1,7260) (FNF(J),J=1,NMAX)
404:             write(NOUT1,7280) (DNOLD(J,KZ),J=1,NMAX)
405:         end if
406: C
407: 7080      format(/' ## LOOPT KZ PHIP FLUX (BURN4)## ',2I6,
408: &          1P,5E12.5)
409: 7100      format(' ', ' ## SGA ## ',1P,10E11.4)
410: 7120      format(' ', ' ## SGF ## ',1P,10E11.4)
411: 7140      format(' ', ' ## SGC ## ',1P,10E11.4)
412: 7160      format(' ', ' ## SGN2N## ',1P,10E11.4)
413: 7180      format(/' ## NMAX TEMP1 TEMP2 (BURN4)## ',I6,1P,5E12.5)
414: 7200      format(' ', ' ## EFIS## ',1P,10E11.4)
415: 7220      format(' ', ' ## CFIS## ',1P,10E11.4)
416: 7240      format(' ', ' ## FNN ## ',1P,10E11.4)
417: 7260      format(' ', ' ## FNF ## ',1P,10E11.4)
418: 7280      format(' ', ' ## DNOLD## ',1P,10E11.4)
419: C
420:         do 330 LN = 1, NMAX
421:             SAVE = (SGF(LN)*EFC(1,LN)+SGC(LN)*EFC(2,LN))*FACFLX
422: C
423:             if( MATREG(KZ).gt.0) then
424:                 TEMP3 = TEMP3 + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*ACOE
425:                 TEMP4 = TEMP4 + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*ACOE
426:             endif
427: C
428:             POWZN1(KZ) = POWZN1(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*
429: &             ACOEF
430:             POWZN2(KZ) = POWZN2(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*
431: &             ACOEF
432:             DN(LN,KZ) = FNF(LN)
433: 330      continue
434: 340      continue
435: C
436:         SUMU5 = 0.0
437:         do 350 KZ = 1, NMAT
438:             SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
439: 350      continue
440:         SAVEU5 = 100.0*(ST235-SUMU5*1.000E+24) /ST235
441: C
442:         FU235(LOOPT+1) = SAVEU5
443: C2003     FACPOW = FACPOW + FACFLX*DELDAY
444:         FACPOW = FACPOW + (FACFLX0+FACFLX)*0.5*DELDAY
445: C
446:         if ( IDBG.gt.0 ) then
447:             write(NOUT1,*) ' ** LOOPT TEMP3 TEMP4 SAVEU5(BURN4) ** ',
448: &             LOOPT, TEMP3, TEMP4, SAVEU5
449:             write(NOUT1,*) ' ** LOOPT TEMP0 (TEMP3+TEMP4)/2 FACFLX** '
450:             write(NOUT1,*) LOOPT, TEMP0, (TEMP3+TEMP4) /2.000, FACFLX
451:             write(NOUT1,*) ' ** LOOPT FACPOW DAYSUM FACPOW/DAYSUM ** '
452:             write(NOUT1,*) LOOPT, FACPOW, DAYSUM, FACPOW/DAYSUM
453:         end if
454: C
455:         do 360 KZ = 1, NMAT

```

```

456:             CUMWD(KZ) = CUMWD(KZ) + DELDAY*(POWZN1(KZ)+POWZN2(KZ))*
457: &             0.5000000
458: 360      continue
459: C
460:             if ( IBC2.eq.5.and.LOOPT.ge.2.and.LOOPT.le.NLOOP1 ) then
461:                 if ( SAVEU5.ge.TIMEU5 ) go to 380
462:                 DELU5 = FU235(LOOPT+1) - FU235(LOOPT)
463:                 U5NEXT = SAVEU5 + DELU5
464:                 if ( IDBG.gt.0 ) write(NOUT1,*)
465: &                 ' ** LOOPT DELU5 U5NEXT TIMEU5 : ', LOOPT, DELU5, U5NEXT,
466: &                 TIMEU5, FU235(LOOPT+1), FU235(LOOPT)
467: C
468:                 if ( LOOPT.lt.NLOOP1 ) then
469:                     if ( U5NEXT.gt.TIMEU5 ) then
470:                         TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
471:                         DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
472:                     end if
473:                 else
474:                     TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
475:                     DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
476:                 end if
477:             end if
478: 370      continue
479: C
480: C *** END OF PROCESS
481: C
482: 380      DELT = DAYSUM
483:             FACPOW = FACPOW/DAYSUM
484:             6888 format(/' XXX(BURN4) ERROR-STOP :' )
485: C
486: C=====
487: 9999      continue
488:             return
489:         end

```


src/mvpburn/burncl.f

```

1:      subroutine BURNCL( PHIP, VOL, DELT, NMAT, NBOPT, NOUT1, NDN,
2:      &                  LNMAX, NMAX, NFIS, NPAR, EVTOJ, EFC, DN,
3:      &                  DNOLD, SSF, SSC, SSN2N, CUMWD, IDBG,
4:      &                  NOUT2, LCHA, NCHA, MTBURN, NUCLP, GAM, GAMI,
5:      &                  NBIC, PBIC, KSTP, LONG, LL, IP, KP,
6:      &                  NPN, PHAI, IBC2, ST235, TIMEU5, IDU235,
7:      &                  FACPOW, POWZN1, POWZN2, NUCL, NCH, RAMDA,
8:      Ckuni&          FACT2N, IBC )
9:      &                  FACT2N, IBC, KPBURN, KZEROP, BZERO )
10: C-----
11: C      BURNUP DENSITY CALCULATION USING D-CHAIN METHOD
12: C      THIS ROUTINE IS POWER CONSTANTS MODE
13: C
14: C      NBOPT = 0 : normal burnup
15: C      NBOPT = 1 : cooling (zero power)
16: C-----
17:      real PHIP(NMAT)
18:      real VOL(NMAT)
19:      real POWZN1(NMAT)
20:      real POWZN2(NMAT)
21: C
22:      real EFC(2, LNMAX)
23:      real DN(NMAX, NMAT)
24:      real DNOLD(NMAX, NMAT)
25:      real SSF(NMAX, NMAT)
26:      real SSC(NMAX, NMAT)
27:      real SSN2N(NMAX, NMAT)
28:      real CUMWD(NMAT)
29:      integer MTBURN(3, NMAT)
30: C
31:      integer NUCLP(NPAR, NMAX)
32:      real GAM(NFIS, NMAX)
33:      real GAMI(NFIS, NMAX)
34: C
35:      integer NBIC(NPAR, NMAX)
36:      real PBIC(NPAR, NMAX)
37:      integer KSTP(NCHA, NMAX)
38:      integer LONG(NCHA, NMAX)
39:      integer LL(NCHA, NMAX)
40:      integer IP(LCHA, NCHA, NMAX)
41:      integer KP(LCHA, NCHA, NMAX)
42:      integer NPN(NPAR, NMAX)
43:      real PHAI(NPAR, NMAX)
44: Ckuni 2003/12/21
45:      integer KZEROP(NCHA, NMAX), KPBURN(NMAX, NMAT)
46: Cend
47: C
48: C      PARAMETER ( MXSTEP = 100 , MXNUC = 200 , MXFISS = 50 )
49:      include 'INC/_BURNP'
50: C
51:      real RAMDA(MXNUC)
52:      integer NCH(MXNUC)
53:      integer NUCL(MXNUC)
54:      real FACT2N(MXNUC)
55: C2003 ... added as an argument
56:      integer IBC(20)
57: C
58: C      .... local arrays ....
59: C
60:      integer NOL(MXNUC)
61:      real GAMMA(MXNUC)
62:      real GAMX(MXNUC)
63:      real PHIC(MXNUC)
64:      real PHIL(MXNUC)
65:      real R(MXNUC)

```

```

66:      real G(MXNUC)
67:      real P(MXNUC)
68:      real RAM(MXNUC)
69:      integer MAP(MXNUC)
70:      real*8 DEX(MXNUC)
71:      real*8 DEXX(MXNUC)
72:      real*8 DCOEF(MXNUC)
73:      real*8 TSTEP
74:      real*8 TTSTEP
75: C
76:      real SGF(MXNUC)
77:      real SGA(MXNUC)
78:      real SGC(MXNUC)
79:      real SGN2N(MXNUC)
80:      real FNN(MXNUC)
81:      real FNF(MXNUC)
82: C
83:      real FU235(MXNUC)
84: C
85:      real*8 FLUX
86: C-----
87: C
88: C *** START PROCESS
89: C
90:      call CLEA( CUMWD, NMAT, 0.0 )
91: C
92:      if ( DELT.le.0.0 ) return
93: C
94:      if ( NMAX.gt.MXNUC ) then
95:          write(NOUT1,6888)
96:          write(NOUT1,*) ' number of depleting nuclides (', NMAX,
97:      &                  ' ) is over reserved size( MXNUC=', MXNUC, ' )'
98:          write(NOUT1,*) ' Change parameter value in include file.'
99:          stop 999
100:      end if
101: C
102:      ITYP = 1
103:      if ( NBOPT.eq.1 ) ITYP = 2
104: C
105:      call LNCHAI( NUCL, RAMDA, NCH, NUCLP, NBIC, PBIC, NOL, KSTP, LONG,
106:      &              LL, IP, KP, NPN, NMAX, NPAR, NCHA, LCHA, ITYP )
107: C
108: C-----+
109: C
110:      call CLEA( GAMI, NFIS*NMAX, 0.0 )
111:      call MEMOVE( GAM, GAMI, NFIS*NMAX )
112: C
113:      TSTEP = DELT*24.0*3600.0
114: C
115:      if ( NBOPT.ne.0 ) then
116: Ckuni
117:          KOOPT = 1
118: Cend
119:      do 140 KZ = 1, NMAT
120:          call ICLEA( MAP, MXNUC, 0 )
121:          MAPLEN = 0
122:          do 100 I = MTBURN(1,KZ), MTBURN(2,KZ)
123:              MAPLEN = MAPLEN + 1
124:              MAP(MAPLEN) = I
125:              if ( KPBURN(I,KZ).gt.0 ) then
126:                  MAPLEN = MAPLEN + 1
127:                  MAP(MAPLEN) = I
128:              endif
129: Cend
130:          100      continue

```

src/mvpburn/burncl.f

```

131: C
132:       do 110 J = 1, NMAX
133:         FNN(J) = DN(J,KZ)
134:         FNF(J) = 0.0
135:         SGF(J) = SSF(J,KZ)
136:         SGC(J) = SSC(J,KZ)
137:         SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
138:         SGA(J) = SGF(J) + SGC(J)
139:       110 continue
140: C
141: C **** COOLING ****
142: C
143:       FLUX = 0.0
144:       XM = 0.0
145: C
146:       do 120 II = 1, NMAX
147:         GAMX(II) = 0.0
148:         GAMMA(II) = 0.0
149:       120 continue
150: C
151: C2003 ... pass IMAT, IBC as an argument
152:       IMAT = KZ
153:       call FNCALC( RAMDA, NCH, NBIC, PBIC, NOB, KSTP, LONG, LL,
154:         & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
155:         & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
156:         & DCOEF, MAP, FLUX, TSTEP, XM, NMAX, NPAR, NCHA, LCHA,
157:         & CKuni& MAPLEN, IMAT, IBC )
158:       CKSK & MAPLEN, IMAT, IBC, KOOPT, KZEROP, IDBG, BZERO)
159:       & MAPLEN, IMAT, IBC, KOOPT, KZEROP, IDBG, BZERO, NBOP)
160: C=====
161: C
162:       do 130 J = 1, NMAX
163:         DN(J,KZ) = FNF(J)
164:       130 continue
165:       140 continue
166: C2003+ return
167:       goto 9999
168:     end if
169: C
170: C *** BURN-UP
171: C
172:       call CLEA( FU235, MXNUC, 0.0 )
173:       FACPOW = 0.0
174: C
175:       if ( IDBG.gt.1 ) then
176:         do 170 L = 1, NMAX
177:           write(NOUT1,7000) L, NUCL(L), NCH(L), RAMDA(L)
178:           do 160 LN = 1, NCH(L)
179:             write(NOUT1,7020) LN, NUCLP(LN,L), NBIC(LN,L), PBIC(LN,L)
180:           160 continue
181:         170 continue
182:       end if
183: C
184:       if ( IDBG.gt.2 ) then
185:         do 180 L = 1, NFIS
186:           write(NOUT1,7040) L, NUCL(L)
187:           write(NOUT1,7060) (GAM(L,LN), LN=1,NMAX)
188:         180 continue
189:       end if
190: C
191: 7000 format(' ## L NUCL NCH RAMDA ## ',3I8,1P,E12.5)
192: 7020 format(' ', ' << LN NUCLP NBIC PBIC >> ',3I8,F10.5)
193: 7040 format(' ', ' ## L NUCL (GAM) ## ',2I8)
194: 7060 format(' ', ' << GAM >> ',1P,10E11.4)
195: C

```

```

196:       TEMP0 = 0.0
197:       ACOEF = EVTOJ*1.0000E+6
198: C
199:       SUMU5 = 0.0
200:       do 200 KZ = 1, NMAT
201: C
202:         if ( IDBG.gt.0 ) then
203:           write(NOUT1,*) ' ** KZ VOL PHIP ACOEF **', KZ, VOL(KZ),
204:         & PHIP(KZ), ACOEF
205:         end if
206: C
207:         SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
208:         do 190 LN = 1, NMAX
209:           TEMP0 = TEMP0 +
210:             & (SSF(LN,KZ)*EFC(1,LN)+SSC(LN,KZ)*EFC(2,LN))*VOL(KZ)*
211:             & PHIP(KZ)*DN(LN,KZ)*ACOEF
212: C
213:           if ( IDBG.gt.1 ) then
214:             write(NOUT1,*) ' ** KZ LN SSF EF SSC EC : ', KZ, LN,
215:           & SSF(LN,KZ), EFC(1,LN), SSC(LN,KZ), EFC(2,LN)
216:           end if
217: C
218:           190 continue
219:           200 continue
220:           FU235(1) = 100.0*(ST235-SUMU5*1.000E+24) /ST235
221: C
222:           NLOOP = NDN
223:           SAVED = DELT/DBLE(NLOOP)
224:           if ( SAVED.lt.1.00 ) then
225:             NLOOP = DELT + 0.01
226:           end if
227:           if ( SAVED.gt.10.0 ) then
228:             NLOOP = DELT/10.000 + 0.01
229:           end if
230: C
231:           if ( NLOOP.lt.5 ) NLOOP = 5
232:           if ( NLOOP.gt.50 ) NLOOP = 50
233:           NLOOP1 = NLOOP - 1
234:           TTSTEP = TSTEP/DBLE(NLOOP)
235:           DELDAY = DELT/DBLE(NLOOP)
236: C
237:           if ( IDBG.gt.0 ) then
238:             write(NOUT1,*) ' ** TEMP0 NLOOP DELDAY (BURNCL) : ', TEMP0,
239:           & NLOOP, DELDAY
240:           end if
241: C
242: C-----LOOP OF SUB-TIME STEP
243: C
244:           DAYSUM = 0.0
245:           FACFLX = 1.00000
246:           do 370 LOOPT = 1, NLOOP
247:             if ( TTSTEP.le.0.0 ) go to 380
248:             DAYSUM = DAYSUM + DELDAY
249: C-----FIRST LOOP OF MATERIAL ZONE
250: C-----LOOP OF MATERIAL ZONE
251:             TEMP1 = 0.0
252:             TEMP2 = 0.0
253: Ckuni
254:             KOOPT = LOOPT
255: Cend
256:           do 270 KZ = 1, NMAT
257:             VOLDEP = VOL(KZ)
258:             call ICLEA( MAP, MXNUC, 0 )
259:             MAPLEN = 0
260:             do 210 I = MTBURN(1,KZ), MTBURN(2,KZ)

```

src/mvpburn/burncl.f

```

261: Ckuni      MAPLEN = MAPLEN + 1
262: Ckuni      MAP(MAPLEN) = I
263:            if ( KPBURN(I,KZ).gt.0 ) then
264:                MAPLEN = MAPLEN + 1
265:                MAP(MAPLEN) = I
266:            endif
267: Cend
268: 210      continue
269: C
270:      do 220 J = 1, NMAX
271:          FNN(J) = DN(J,KZ)
272:          FNF(J) = 0.0
273:          SGF(J) = SSF(J,KZ)
274:          SGC(J) = SSC(J,KZ)
275:          SGA(J) = SGF(J) + SGC(J)
276:          SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
277: 220      continue
278: C
279:      FLUX = PHIP(KZ)*1.0E-24*FACFLX
280:      XM = 0.0
281: C
282:      do 230 J = 1, NFIS
283:          XM = XM + SGF(J)*FNN(J)
284: 230      continue
285:      RXM = 0.0
286:      if ( XM.gt.0.0 ) RXM = 1.00000/XM
287: C
288:      do 250 II = 1, NMAX
289:          SUMGX = 0.0
290:          SUMGI = 0.0
291:          do 240 J = 1, NFIS
292:              SUMGX = SUMGX + GAM(J,II)*FNN(J)*SGF(J)
293:              SUMGI = SUMGI + GAMI(J,II)*FNN(J)*SGF(J)
294: 240          continue
295:          GAMX(II) = SUMGX*RXM
296:          GAMMA(II) = SUMGI*RXM
297: 250      continue
298: C
299:      XM = XM*FLUX
300: C
301: C2003 ... pass IMAT, IBC as an argument
302:      IMAT = KZ
303:      call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
304:          &      IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
305:          &      PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
306:          &      DCOEF, MAP, FLUX, TTSTEP, XM, NMAX, NPAR, NCHA,
307:          Ckuni&      LCHA, MAPLEN, IMAT, IBC )
308:      CKSK&      LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO)
309:      &      LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO,NBOP)
310: C=====
311: C
312:      do 260 LN = 1, NMAX
313:          SAVE = (SGF(LN)*EFC(1,LN)+SGC(LN)*EFC(2,LN))*FACFLX
314:          TEMP1 = TEMP1 + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*ACOE
315:          TEMP2 = TEMP2 + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*ACOE
316: 260      continue
317: 270      continue
318: C
319: C-----SECOND LOOP OF MATERIAL ZONE
320: C2003
321:      FACFLX0 = FACFLX
322:      FACFLX = FACFLX*2.000*TEMP0/(TEMP1+TEMP2)
323: C
324: check ###
325: c      write(NOUT1,*) ' ## FACFLX ',FACFLX,' LOOPT ',LOOPT

```

```

326:      if ( IDBG.gt.0 ) then
327:          write(NOUT1,*)
328:          &      ' **LOOPT TEMP1 TEMP2 FACFLX DAYSUM (BURNCL) ** ',
329:          &      LOOPT, TEMP1, TEMP2, FACFLX, DAYSUM
330:      end if
331:      call CLEA( POWZN1, NMAT, 0.0 )
332:      call CLEA( POWZN2, NMAT, 0.0 )
333: C-----LOOP OF MATERIAL ZONE
334:      TEMP3 = 0.0
335:      TEMP4 = 0.0
336: Ckuni
337:      KOOPT = LOOPT + 1
338: Cend
339:      do 340 KZ = 1, NMAT
340:          VOLDEP = VOL(KZ)
341:          call ICLEA( MAP, MXNUC, 0 )
342:          MAPLEN = 0
343:          do 280 I = MTBURN(1,KZ), MTBURN(2,KZ)
344:              MAPLEN = MAPLEN + 1
345:              MAP(MAPLEN) = I
346:              if ( KPBURN(I,KZ).gt.0 ) then
347:                  MAPLEN = MAPLEN + 1
348:                  MAP(MAPLEN) = I
349:              endif
350: Cend
351: 280      continue
352: C
353:      do 290 J = 1, NMAX
354:          FNN(J) = DN(J,KZ)
355:          FNF(J) = 0.0
356:          SGF(J) = SSF(J,KZ)
357:          SGC(J) = SSC(J,KZ)
358:          SGA(J) = SGF(J) + SGC(J)
359:          SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
360: 290      continue
361: C
362:      FLUX = PHIP(KZ)*1.0E-24*FACFLX
363:      XM = 0.0
364: C
365:      do 300 J = 1, NFIS
366:          XM = XM + SGF(J)*FNN(J)
367: 300      continue
368: C
369:      RXM = 0.0
370:      if ( XM.gt.0.0 ) RXM = 1.00000/XM
371: C
372:      do 320 II = 1, NMAX
373:          SUMGX = 0.0
374:          SUMGI = 0.0
375:          do 310 J = 1, NFIS
376:              SUMGX = SUMGX + GAM(J,II)*FNN(J)*SGF(J)
377:              SUMGI = SUMGI + GAMI(J,II)*FNN(J)*SGF(J)
378: 310          continue
379:          GAMX(II) = SUMGX*RXM
380:          GAMMA(II) = SUMGI*RXM
381: 320      continue
382: C
383:      XM = XM*FLUX
384: C
385: C2003 ... pass IMAT, IBC as an argument
386:      IMAT = KZ
387:      call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
388:          &      IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
389:          &      PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
390:          &      DCOEF, MAP, FLUX, TTSTEP, XM, NMAX, NPAR, NCHA,

```

src/mvpburn/burncl.f

```

391: Ckuni&          LCHA, MAPLEN, IMAT, IBC )
392: CKSK &          LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO)
393:      &          LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO,NBOPT)
394: CCHECK
395:      if ( IDBG.gt.1 ) then
396:      if ( KZ.eq.1 ) then
397:      write(NOUT1,7180) NMAX, TEMP0, FACFLX
398:      write(NOUT1,7200) (EFC(1,LN),LN=1,NMAX)
399:      write(NOUT1,7220) (EFC(2,LN),LN=1,NMAX)
400:      end if
401:      write(NOUT1,7080) LOOPT, KZ, PHIP(KZ), FLUX
402:      write(NOUT1,7120) SGF
403:      write(NOUT1,7140) SGC
404:      write(NOUT1,7160) SGA
405:      write(NOUT1,7180) SGN2N
406:      write(NOUT1,7240) (FNN(J),J=1,NMAX)
407:      write(NOUT1,7260) (FNF(J),J=1,NMAX)
408:      write(NOUT1,7280) (DNOLD(J,KZ),J=1,NMAX)
409:      end if
410: C
411: 7080      format('  ## LOOPT KZ PHIP FLUX (BURNCL)## ',216,
412:      &          1P,5E12.5)
413: 7100      format(' ', ' ## SGA ## ',1P,10E11.4)
414: 7120      format(' ', ' ## SGF ## ',1P,10E11.4)
415: 7140      format(' ', ' ## SGC ## ',1P,10E11.4)
416: 7160      format(' ', ' ## SGN2N ## ',1P,10E11.4)
417: 7180      format('  ## NMAX TEMP1 TEMP2 (BURNCL)## ',16,1P,5E12.5)
418: 7200      format(' ', ' ## EFC## ',1P,10E11.4)
419: 7220      format(' ', ' ## CFIS## ',1P,10E11.4)
420: 7240      format(' ', ' ## FNN ## ',1P,10E11.4)
421: 7260      format(' ', ' ## FNF ## ',1P,10E11.4)
422: 7280      format(' ', ' ## DNOLD# ',1P,10E11.4)
423: C
424:      do 330 LN = 1, NMAX
425:      SAVE = (SGF(LN)*EFC(1,LN)+SGC(LN)*EFC(2,LN))*FACFLX
426:      TEMP3 = TEMP3 + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*ACOE
427:      TEMP4 = TEMP4 + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*ACOE
428:      POWZN1(KZ) = POWZN1(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*
429:      &          ACOE
430:      POWZN2(KZ) = POWZN2(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*
431:      &          ACOE
432:      DN(LN,KZ) = FNF(LN)
433: 330      continue
434: 340      continue
435: C
436:      SUMU5 = 0.0
437:      do 350 KZ = 1, NMAT
438:      SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
439: 350      continue
440:      SAVEU5 = 100.0*(ST235-SUMU5*1.000E+24) /ST235
441: C
442:      FU235(LOOPT+1) = SAVEU5
443: C2003      FACPOW = FACPOW + FACFLX*DELDAY
444:      FACPOW = FACPOW + (FACFLX0+FACFLX)*0.5*DELDAY
445: C
446:      if ( IDBG.gt.0 ) then
447:      write(NOUT1,*) ' ** LOOPT TEMP3 TEMP4 SAVEU5(BURNCL) ** ',
448:      &          LOOPT, TEMP3, TEMP4, SAVEU5
449:      write(NOUT1,*) ' ** LOOPT TEMP0 (TEMP3+TEMP4)/2 FACFLX** '
450:      write(NOUT1,*) LOOPT, TEMP0, (TEMP3+TEMP4) /2.000, FACFLX
451:      write(NOUT1,*) ' ** LOOPT FACPOW DAYSUM FACPOW/DAYSUM ** '
452:      write(NOUT1,*) LOOPT, FACPOW, DAYSUM, FACPOW/DAYSUM
453:      end if
454: C
455:      do 360 KZ = 1, NMAT

```

```

456:      CUMWD(KZ) = CUMWD(KZ) + DELDAY*(POWZN1(KZ)+POWZN2(KZ))*
457:      &          0.5000000
458: 360      continue
459: C
460:      if ( IBC2.eq.5.and.LOOPT.ge.2.and.LOOPT.le.NLOOP1 ) then
461:      if ( SAVEU5.ge.TIMEU5 ) go to 380
462:      DELU5 = FU235(LOOPT+1) - FU235(LOOPT)
463:      U5NEXT = SAVEU5 + DELU5
464:      if ( IDBG.gt.0 ) write(NOUT1,*)
465:      &          ' ** LOOPT DELU5 U5NEXT TIMEU5 : ', LOOPT, DELU5, U5NEXT,
466:      &          TIMEU5, FU235(LOOPT+1), FU235(LOOPT)
467: C
468:      if ( LOOPT.lt.NLOOP1 ) then
469:      if ( U5NEXT.gt.TIMEU5 ) then
470:      TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
471:      DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
472:      end if
473:      else
474:      TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
475:      DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
476:      end if
477:      end if
478: 370      continue
479: C
480: C *** END OF PROCESS
481: C
482: 380      DELT = DAYSUM
483:      FACPOW = FACPOW/DAYSUM
484: 6888      format('  XXX(BURNCL) ERROR-STOP : ' )
485: C
486: C=====
487: 9999      continue
488:      return
489:      end

```

src/mvpburn/burncx.f

```

1:      subroutine BURNCX( PHIP, VOL, DELT, NMAT, NBOPT, NOUT1, NDN,
2:      &                  LNMAX, NMAX, NFIS, NPAR, EVTOJ, EFC, DN,
3:      &                  DNOLD, SSF, SSC, SSN2N, CUMWD, IDBG,
4:      &                  NOUT2, LCHA, NCHA, MTBURN, NUCLP, GAM, GAMI,
5:      &                  NBIC, PBIC, KSTP, LONG, LL, IP, KP,
6:      &                  NPN, PHAI, IBC2, ST235, TIMEUS, IDU235,
7:      &                  FACPOW, POWZN1, POWZN2, NUCL, NCH, RAMDA,
8:      &                  FACT2N, IBC, KPBURN, KZEROP, BZERO )
9: C=====
10: C      BURNUP DENSITY CALCULATION USING D-CHAIN METHOD
11: C      THIS ROUTINE IS MICROSCOPIC REACTION RATE CONSTANTS MODE
12: C
13: C      NBOPT = 0 : normal burnup
14: C      NBOPT = 1 : cooling (zero power)
15: C=====
16:      real PHIP(NMAT)
17:      real VOL(NMAT)
18:      real POWZN1(NMAT)
19:      real POWZN2(NMAT)
20: C
21:      real EFC(2, LNMAX)
22:      real DN(NMAX, NMAT)
23:      real DNOLD(NMAX, NMAT)
24:      real SSF(NMAX, NMAT)
25:      real SSC(NMAX, NMAT)
26:      real SSN2N(NMAX, NMAT)
27:      real CUMWD(NMAT)
28:      integer MTBURN(3, NMAT)
29: C
30:      integer NUCLP(NPAR, NMAX)
31:      real GAM(NFIS, NMAX)
32:      real GAMI(NFIS, NMAX)
33: C
34:      integer NBIC(NPAR, NMAX)
35:      real PBIC(NPAR, NMAX)
36:      integer KSTP(NCHA, NMAX)
37:      integer LONG(NCHA, NMAX)
38:      integer LL(NCHA, NMAX)
39:      integer IP(LCHA, NCHA, NMAX)
40:      integer KP(LCHA, NCHA, NMAX)
41:      integer NPN(NPAR, NMAX)
42:      real PHAI(NPAR, NMAX)
43: C
44:      integer KZEROP(NCHA, NMAX), KPBURN(NMAX, NMAT)
45: C
46: C      PARAMETER ( MXSTEP = 100 , MXNUC = 200 , MXFISS = 50 )
47: C      include 'INC/_BURNP'
48: C
49:      real RAMDA(MXNUC)
50:      integer NCH(MXNUC)
51:      integer NUCL(MXNUC)
52:      real FACT2N(MXNUC)
53: C
54:      integer IBC(20)
55: C
56: C      .... local arrays ....
57: C
58:      integer NOL(MXNUC)
59:      real GAMMA(MXNUC)
60:      real GAMX(MXNUC)
61:      real PHIC(MXNUC)
62:      real PHIL(MXNUC)
63:      real R(MXNUC)
64:      real G(MXNUC)
65:      real P(MXNUC)

```

```

66:      real RAM(MXNUC)
67:      integer MAP(MXNUC)
68:      real*8 DEX(MXNUC)
69:      real*8 DEXX(MXNUC)
70:      real*8 DCOEF(MXNUC)
71:      real*8 TSTEP
72:      real*8 TTSTEP
73: C
74:      real SGF(MXNUC)
75:      real SGA(MXNUC)
76:      real SGC(MXNUC)
77:      real SGN2N(MXNUC)
78:      real FNN(MXNUC)
79:      real FNF(MXNUC)
80: C
81:      real FU235(MXNUC)
82: C
83:      real*8 FLUX
84: C-----
85: C
86: C *** START PROCESS
87: C
88:      call CLEA( CUMWD, NMAT, 0.0 )
89: C
90:      if ( DELT.le.0.0 ) return
91: C
92:      if ( NMAX.gt.MXNUC ) then
93:          write(NOUT1,6888)
94:          write(NOUT1,*) ' number of depleting nuclides (', NMAX,
95:          &                ' ) is over reserved size( MXNUC=', MXNUC, ' )'
96:          write(NOUT1,*) ' Change parameter value in include file.'
97:          stop 999
98:      end if
99: C
100:      ITYP = 1
101:      if ( NBOPT.eq.1 ) ITYP = 2
102: C
103:      call LNCHAI( NUCL, RAMDA, NCH, NUCLP, NBIC, PBIC, NOL, KSTP, LONG,
104:      &              LL, IP, KP, NPN, NMAX, NPAR, NCHA, LCHA, ITYP )
105: C
106: C-----+
107: C
108:      call CLEA( GAMI, NFIS*NMAX, 0.0 )
109:      call MEMOVE( GAM, GAMI, NFIS*NMAX )
110: C
111:      TSTEP = DELT*24.0*3600.0
112: C
113:      if ( NBOPT.ne.0 ) then
114:          KOOPT = 1
115: C
116:          do 140 KZ = 1, NMAT
117:              call ICLEA( MAP, MXNUC, 0 )
118:              MAPLEN = 0
119:              do 100 I = MTBURN(1,KZ), MTBURN(2,KZ)
120:                  if ( KPBURN(I,KZ).gt.0 ) then
121:                      MAPLEN = MAPLEN + 1
122:                      MAP(MAPLEN) = I
123:                  endif
124:              100 continue
125: C
126:              do 110 J = 1, NMAX
127:                  FNN(J) = DN(J,KZ)
128:                  FNF(J) = 0.0
129:                  SGF(J) = SSF(J,KZ)
130:                  SGC(J) = SSC(J,KZ)

```

src/mvpburn/burncx.f

```

131:          SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
132:          SGA(J) = SGF(J) + SGC(J)
133: 110      continue
134: C
135: C **** COOLING ****
136: C
137:          FLUX = 0.0
138:          XM = 0.0
139: C
140:          do 120 II = 1, NMAX
141:              GAMX(II) = 0.0
142:              GAMMA(II) = 0.0
143: 120      continue
144: C
145:          IMAT = KZ
146:          call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
147: & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
148: & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
149: & DCOEF, MAP, FLUX, TSTEP, XM, NMAX, NPAR, NCHA, LCHA,
150: & MAPLEN, IMAT, IBC, KOOPT, KZEROP, IDBG, BZERO, NBOPT)
151: C-----
152: C
153:          do 130 J = 1, NMAX
154:              DN(J,KZ) = FNF(J)
155: 130      continue
156: 140      continue
157: C
158:          goto 9999
159:      end if
160: C
161: C *** BURN-UP
162: C
163:          call CLEA( FU235, MXNUC, 0.0 )
164:          FACPOW = 0.0
165: C
166:          if ( IDBG.gt.1 ) then
167:              do 170 L = 1, NMAX
168:                  write(NOUT1,7000) L, NUCL(L), NCH(L), RAMDA(L)
169:                  do 160 LN = 1, NCH(L)
170:                      write(NOUT1,7020) LN, NUCLP(LN,L), NBIC(LN,L), PBIC(LN,L)
171: 160          continue
172: 170      continue
173:          end if
174: C
175:          if ( IDBG.gt.2 ) then
176:              do 180 L = 1, NFIS
177:                  write(NOUT1,7040) L, NUCL(L)
178:                  write(NOUT1,7060) (GAM(L,LN), LN=1,NMAX)
179: 180          continue
180:          end if
181: C
182:          7000 format(' ## L NUCL NCH RAMDA ## ',3I8,1P,E12.5)
183:          7020 format(' ', ' ' << LN NUCLP NBIC PBIC >> ',3I8,F10.5)
184:          7040 format(' ', ' ' ## L NUCL (GAM) ## ',2I8)
185:          7060 format(' ', ' ' << GAM >> ',1P,10E11.4)
186: C
187:          TEMP0 = 0.0
188:          ACOEF = EVTOJ*1.0000E+6
189: C
190:          SUMU5 = 0.0
191:          do 200 KZ = 1, NMAX
192: C
193:              if ( IDBG.gt.0 ) then
194:                  write(NOUT1,*) ' ** KZ VOL PHIP ACOEF **', KZ, VOL(KZ),
195: &

```

```

196:          end if
197: C
198:          SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
199:          do 190 LN = 1, NMAX
200:              TEMP0 = TEMP0 +
201: & (SSF(LN,KZ)*EFC(1,LN)+SSC(LN,KZ)*EFC(2,LN))*VOL(KZ)*
202: & PHIP(KZ)*DN(LN,KZ)*ACOEF
203: C
204:              if ( IDBG.gt.1 ) then
205:                  write(NOUT1,*) ' ** KZ LN SSF EF SSC EC : ', KZ, LN,
206: & SSF(LN,KZ), EFC(1,LN), SSC(LN,KZ), EFC(2,LN)
207:              end if
208: C
209: 190      continue
210: 200      continue
211:          FU235(1) = 100.0*(ST235-SUMU5*1.000E+24) /ST235
212: C
213:          NLOOP = NDN
214:          SAVED = DELT/DBLE(NLOOP)
215:          if ( SAVED.lt.1.00 ) then
216:              NLOOP = DELT + 0.01
217:          end if
218:          if ( SAVED.gt.10.0 ) then
219:              NLOOP = DELT/10.000 + 0.01
220:          end if
221: C
222:          if ( NLOOP.lt.5 ) NLOOP = 5
223:          if ( NLOOP.gt.50 ) NLOOP = 50
224:          NLOOP1 = NLOOP - 1
225:          TTSTEP = TSTEP/DBLE(NLOOP)
226:          DELDAY = DELT/DBLE(NLOOP)
227: C
228:          if ( IDBG.gt.0 ) then
229:              write(NOUT1,*) ' ** TEMP0 NLOOP DELDAY (BURNCX) : ', TEMP0,
230: & NLOOP, DELDAY
231:          end if
232: C
233: C-----LOOP OF SUB-TIME STEP
234: C
235:          DAYSUM = 0.0
236:          FACFLX = 1.00
237: C
238:          do 370 LOOPT = 1, NLOOP
239:              if ( TTSTEP.le.0.0 ) go to 380
240: C
241:              KOOPT = LOOPT
242:              DAYSUM = DAYSUM + DELDAY
243: C
244:              call CLEA( POWZN1, NMAT, 0.0 )
245:              call CLEA( POWZN2, NMAT, 0.0 )
246: C
247: C-----LOOP OF MATERIAL ZONE
248: C
249:              do 340 KZ = 1, NMAT
250:                  VOLDEP = VOL(KZ)
251:                  call ICLEA( MAP, MXNUC, 0 )
252:                  MAPLEN = 0
253:                  do 280 I = MTBURN(1,KZ), MTBURN(2,KZ)
254:                      if ( KPBURN(I,KZ).gt.0 ) then
255:                          MAPLEN = MAPLEN + 1
256:                          MAP(MAPLEN) = I
257:                      endif
258: 280          continue
259: C
260:              do 290 J = 1, NMAX

```

src/mvpburn/burncx.f

```

261:      FNN(J) = DN(J,KZ)
262:      FNF(J) = 0.0
263:      SGF(J) = SSF(J,KZ)
264:      SGC(J) = SSC(J,KZ)
265:      SGA(J) = SGF(J) + SGC(J)
266:      SGN2N(J) = SSN2N(J,KZ)*FACT2N(J)
267: 290      continue
268: C
269:      FLUX = PHIP(KZ)*1.0E-24*FACFLX
270:      XM = 0.0
271: C
272:      do 300 J = 1, NFIS
273:          XM = XM + SGF(J)*FNN(J)
274: 300      continue
275: C
276:      RXM = 0.0
277:      if ( XM.gt.0.0 ) RXM = 1.00000/XM
278: C
279:      do 320 II = 1, NMAX
280:          SUMGX = 0.0
281:          SUMGI = 0.0
282:          do 310 J = 1, NFIS
283:              SUMGX = SUMGX + GAM(J,II)*FNN(J)*SGF(J)
284:              SUMGI = SUMGI + GAM(J,II)*FNN(J)*SGF(J)
285: 310          continue
286:          GAMX(II) = SUMGX/RXM
287:          GAMMA(II) = SUMGI/RXM
288: 320      continue
289: C
290:      XM = XM*FLUX
291: C
292:      IMAT = KZ
293:      call FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG, LL,
294:          & IP, KP, NPN, GAMMA, GAMX, SGA, SGC, SGN2N, PHIC,
295:          & PHIL, PHAI, P, R, G, RAM, FNN, FNF, DEX, DEXX,
296:          & DCOEF, MAP, FLUX, TTSTEP, XM, NMAX, NPAR, NCHA,
297:          & LCHA, MAPLEN, IMAT, IBC ,KOOPT,KZEROP,IDBG,BZERO,NBOPT)
298: CCHECK
299:      if ( IDBG.gt.1 ) then
300:          if ( KZ.eq.1 ) then
301:              write(NOUT1,7180) NMAX, TEMP0, FACFLX
302:              write(NOUT1,7200) (EFC(1,LN),LN=1,NMAX)
303:              write(NOUT1,7220) (EFC(2,LN),LN=1,NMAX)
304:          end if
305:          write(NOUT1,7080) LOOPT, KZ, PHIP(KZ), FLUX
306:          write(NOUT1,7120) SGF
307:          write(NOUT1,7140) SGC
308:          write(NOUT1,7100) SGA
309:          write(NOUT1,7160) SGN2N
310:          write(NOUT1,7240) (FNN(J),J=1,NMAX)
311:          write(NOUT1,7260) (FNF(J),J=1,NMAX)
312:          write(NOUT1,7280) (DNOLD(J,KZ),J=1,NMAX)
313:      end if
314: C
315: 7080      format(/' ## LOOPT KZ PHIP FLUX (BURNCX)## ',2I6,
316:          & 1P,5E12.5)
317: 7100      format(' ', ' ## SGA ## ',1P,10E11.4)
318: 7120      format(' ', ' ## SGF ## ',1P,10E11.4)
319: 7140      format(' ', ' ## SGC ## ',1P,10E11.4)
320: 7160      format(' ', ' ## SGN2N## ',1P,10E11.4)
321: 7180      format(/' ## NMAX TEMP1 TEMP2 (BURNCX)## ',I6,1P,5E12.5)
322: 7200      format(' ', ' ## EFIS## ',1P,10E11.4)
323: 7220      format(' ', ' ## CFIS## ',1P,10E11.4)
324: 7240      format(' ', ' ## FNN ## ',1P,10E11.4)
325: 7260      format(' ', ' ## FNF ## ',1P,10E11.4)

```

```

326: 7280      format(' ', ' ## DNOLD# ',1P,10E11.4)
327: C
328:      do 330 LN = 1, NMAX
329:          SAVE = (SGF(LN)*EFC(1,LN)+SGC(LN)*EFC(2,LN))*FACFLX
330:          POWZN1(KZ) = POWZN1(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNN(LN)*
331:          & ACOEF
332:          POWZN2(KZ) = POWZN2(KZ) + SAVE*VOLDEP*PHIP(KZ)*FNF(LN)*
333:          & ACOEF
334:          DN(LN,KZ) = FNF(LN)
335: 330      continue
336: 340      continue
337: C
338:      SUMU5 = 0.0
339:      do 350 KZ = 1, NMAX
340:          SUMU5 = SUMU5 + VOL(KZ)*DN(IDU235,KZ)
341: 350      continue
342:          SAVEU5 = 100.0*(ST235-SUMU5*1.000E+24) /ST235
343: C
344:          FU235(LOOPT+1) = SAVEU5
345: C
346:          if ( IDBG.gt.0 ) then
347:              write(NOUT1,*) ' ** LOOPT SAVEU5 DAYSUM (BURNCX) ** ',
348:              & LOOPT, SAVEU5, DAYSUM
349:          end if
350: C
351:          do 360 KZ = 1, NMAX
352:              CUMWD(KZ) = CUMWD(KZ) + DELDAY*(POWZN1(KZ)+POWZN2(KZ))*
353:              & 0.5000000
354: 360          continue
355: C
356:          if ( IBC2.eq.5.and.LOOPT.ge.2.and.LOOPT.le.NLOOP1 ) then
357:              if ( SAVEU5.ge.TIMEU5 ) go to 380
358:              DELU5 = FU235(LOOPT+1) - FU235(LOOPT)
359:              U5NEXT = SAVEU5 + DELU5
360: C
361:              if ( IDBG.gt.0 ) write(NOUT1,*)
362:              & ' ** LOOPT DELU5 U5NEXT TIMEU5 : ', LOOPT, DELU5, U5NEXT,
363:              & TIMEU5, FU235(LOOPT+1), FU235(LOOPT)
364: C
365:              if ( LOOPT.lt.NLOOP1 ) then
366:                  if ( U5NEXT.ge.TIMEU5 ) then
367:                      TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
368:                      DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
369:                  end if
370:              else
371:                  TTSTEP = TTSTEP*(TIMEU5-SAVEU5) /DELU5
372:                  DELDAY = DELDAY*(TIMEU5-SAVEU5) /DELU5
373:              end if
374:          end if
375: 370      continue
376: C
377: C *** END OF PROCESS
378: C
379: 380      DELT = DAYSUM
380:          FACPOW = 1.0
381: 6888      format(/' XXX(BURNCX) ERROR-STOP :' )
382: C
383: C=====
384: 9999      continue
385:          return
386:          end

```

src/mvpburn/burnin.f

```

1:      subroutine BURNIN( MODE, NOUT1, NOUT2, T00,  DIRIN, DIROUT,
2:      &                  A, MEMORY, IBSTEP, MXSTEP,
3:      &                  TRGNAM, IDENT, MAXTRG, MAXISO,
4:      &                  CASEID, IBC, NMAT,  KNMAX, NCARD, TITLE, NIN,
5:      &                  MVPDBG, HMINV0, IHMIN0, TRGPOW, MATPOW, JPCFLG)
6: C
7: C=====
8: C      THIS ROUTINE TREATS MVP-BURN INITIAL INPUT DATA PROCESSING
9: C=====
10: C
11: C      common /MEMOCK/  MEUSED, MEMMAX
12: C
13: C      character*4 CASEID
14: C      character*72 TITLE(2)
15: C      character*(*) MODE
16: C      character*(*) DIRIN
17: C      character*(*) DIROUT
18: C
19: C      integer IBSTEP(MXSTEP)
20: C      integer IBC(20)
21: C
22: C      real A(MEMORY)
23: C      character*12 TRGNAM(MAXTRG)
24: C      character*12 IDENT(MAXISO,MAXTRG)
25: C      real HMINV0(MAXTRG)
26: C      character*12 TRGPOW(MAXTRG)
27: C
28: C      character*80 LINE
29: C
30: C -----
31: C
32: C *** COPY CHAIN DATA FROM FT50F001 TO FT93F001
33: C
34: C      call RWIND( 50 )
35: C      call RWIND( 93 )
36: C
37: C      100 read(50,'(A80)',end =110) LINE
38: C      LINE(73:80) = ' '
39: C      if ( LINE(1:1).ne.'*' ) write(93,'(A80)') LINE
40: C      go to 100
41: C
42: C      110 call RWIND( 50 )
43: C      call RWIND( 93 )
44: C
45: C      READ CHAIN LIBRARY PARAMETER
46: C
47: C      read(93,*) LNMAX0, NTNUC0, NTFIS0, NPAR0, NYLDT0
48: C
49: C      call RWIND( 93 )
50: C
51: C      READ #BLOCK-4 INPUT DATA
52: C
53: C      call REAM( 'DUMM', IBC, IBC, 0, 20, 0 )
54: C
55: C      LCHA0  = IBC(7)
56: C      if ( LCHA0.le.0 ) LCHA0 = 6
57: C      if ( LCHA0.gt.NTFIS0 ) LCHA0  = NTFIS0
58: C
59: C *** READ ORIGINAL MVP INPUT DATA to check burnup material ***
60: C
61: C      LIN      = 92
62: C      NMAT      = 0
63: C      KNMAX      = 0
64: C      NCARD      = 0
65: C      NISO      = 0

```

```

66:      IFLG      = 0
67: C
68: C      call RWIND( LIN )
69: C
70: C      120 read(LIN,'(A80)',end =130) LINE
71: C      call SCTOLC( LINE, 72 )
72: C      if ( LINE(1:8).eq.'*MVPBURN' ) then
73: C          NMAT = NMAT + 1
74: C          NISO = 0
75: C          IFLG = 1
76: C      else if( IFLG.eq.1 ) then
77: C          if ( LINE(1:1).ne.'*' .and. LINE(1:1).ne.'%' .and.
78: C              &      LINE(1:1).ne.'$' .and. LINE(1:1).ne.'&' .and.
79: C              &      LINE(1:72).ne.' ' .and. LINE(1:1).ne.'@' ) then
80: C              NISO = NISO + 1
81: C          else if(LINE(1:7).eq.'& IDMAT' .or. LINE(1:9).eq.'$END XSEC'
82: C              &      .or. LINE(1:10).eq.'$END CROSS') then
83: C              if( NISO.gt.KNMAX ) KNMAX = NISO
84: C              IFLG = 0
85: C          end if
86: C      end if
87: C      NCARD = NCARD + 1
88: C      go to 120
89: C
90: C      130 continue
91: C
92: C      call RWIND( LIN )
93: C
94: C
95: C *** SET VAIRABLE DIMENSION
96: C
97: C      LOC01  = 1
98: C ---  NUCLP
99: C      LOC02  = LOC01 + NPAR0*NTNUC0
100: C ---  GAM
101: C      LOC03  = LOC02 + NTFIS0*NTNUC0
102: C ---  NBIC
103: C      LOC04  = LOC03 + NPAR0*NTNUC0
104: C ---  PBIC
105: C      LOC05  = LOC04 + NPAR0*NTNUC0
106: C ---  DNBURN
107: C      LOC06  = LOC05 + NMAT*NTNUC0
108: C ---  NISO
109: C      LOC07  = LOC06 + NMAT
110: C ---  TEMP
111: C      LOC08  = LOC07 + NMAT
112: C ---  VOLM
113: C      LOC09  = LOC08 + NMAT
114: C ---  MATREG
115: C      LOC10  = LOC09 + NMAT
116: C ---  LENTRG
117: C      LOC11  = LOC10 + NMAT
118: C ---  MTBURN
119: C      LOC12  = LOC11 + NMAT*3
120: C ---  MTCARD
121: C      LOC13  = LOC12 + NMAT*2
122: C ---  IDENT
123: C      LOC14  = LOC13
124: C ---  DNINIT
125: C      LOC15  = LOC14 + KNMAX*NMAT
126: C ---  MATMVP
127: C      LOC16  = LOC15 + NMAT
128: C ---  IPBURN
129: C      LOC17  = LOC16 + KNMAX*NMAT
130: C ---  WTHVY0

```


src/mvpburn/burnin.f

```

131:      LOC18   = LOC17 + NMAT
132: C
133: C --- MVPDAT : Base MVP input is stored on memory as 72-column
134: C      text lines.
135:      LOC19 = LOC18
136: C
137: CC/#IF WORD(64)
138: C*      LOC19   = LOC18 + NCARD*9
139: CC/#ELSE
140: C      LOC19   = LOC18 + NCARD*18
141: CC/#ENDIF
142:      LAST    = LOC19
143: C -----
144: if (LAST.gt.MEUSED) MEUSED = LAST
145: MEMORY = MEMORY - LAST + 1
146: if ( MEMORY.lt.0 ) then
147:   write(NOUT1,6888)
148:   write(NOUT1,*) ' memory over '
149:   write(NOUT1,*) ' requested memory size = ', LAST-1, ' words.'
150:   write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
151:   write(NOUT1,*) ' change parameter value in include file'
152:   stop 999
153: else
154:   call CLEA( A, LOC19, 0.0 )
155: end if
156: C
157: if ( KNMAX.gt.MAXISO ) then
158:   write(NOUT1,6888)
159:   write(NOUT1,*) ' too many nuclides per tally region'
160:   write(NOUT1,*) ' requested number of nuclides = ',KNMAX
161:   write(NOUT1,*) ' limit for number of nuclides per ',
162: &   'tally region(MAXISO)=',MAXISO
163:   write(NOUT1,*) ' change parameter value in include file'
164:   stop 999
165: end if
166: C
167: if ( NMAT.gt.MAXTRG ) then
168:   write(NOUT1,6888)
169:   write(NOUT1,*) ' too many tally regions'
170:   write(NOUT1,*) ' requested number of materials = ',NMAT
171:   write(NOUT1,*) ' limit for number of tally regions',
172: &   '(MAXTRG)=',MAXTRG
173:   write(NOUT1,*) ' change parameter value in include file'
174:   stop 999
175: end if
176: C
177: if (IHMIN0.gt.0) then
178:   if (IHMIN0.ne.NMAT) then
179:     write(NOUT1,6888)
180:     write(NOUT1,*)
181: &     ' number of inputted values of HMINV0 is not equal to',
182: &     ' number of burnup materials'
183:     write(NOUT1,*)
184: &     ' inputted values =',IHMIN0,' number of burnup materials=',
185: &     NMAT
186:     stop 888
187:   end if
188:
189:   IERR = 0
190:   do I = 1, IHMIN0
191:     if (HMINV0(I).le.0.0) then
192:       write(NOUT1,*) ' XXX(BURNIN) ERROR : ',
193: &       'HMINV0 is less or equal 0.0. for material=',
194: &       I, ' TRGNAM=',TRGNAM(I)
195:       IERR = IERR + 1
196:     end if
197:   end do
198:
199:   if (IERR.gt.0) then
200:     write(NOUT1,6888)
201:     write(NOUT1,*) ' some errors founded in input data'
202:     stop 888
203:   end if
204: end if
205: C
206: C
207: call BURNIP( NOUT1, NOUT2, DIRIN, DIROUT, CASEID,
208: &   LEMORY, NTNUC0, NTFIS0, NPAR0, LCHA0, IBSTEP,
209: &   A(LOC01), A(LOC02), A(LOC03), A(LOC04), A(LOC05),
210: &   A(LOC06), A(LOC07), A(LOC08), TRGNAM , A(LOC10),
211: &   A(LOC11), A(LOC12), IDENT , A(LOC14), A(LOC15),
212: &   A(LOC16), A(LOC17), A(LOC19),
213: &   A(LOC19), NIN , MVPDBG, HMINV0, IHMIN0,
214: &   A(LOC09), TRGPOW , MATPOW, JPCFLG)
215: C
216: call CPUTM2( TM )
217: C
218: if (IBC(3) .gt. 0) then
219:   write(NOUT1,7040) TM/60.0
220: end if
221: C
222: 6888 format(//' XXX(BURNIN) ERROR-STOP :' )
223: 7040 format(/1X,12X,'CPU ',1P,E12.5,' MIN.')
224: C
225: C *** END OF PROCESS
226: C
227: return
228: end

```

src/mvpburn/burnip.f

```

1:      subroutine BURNIP( NOUT1, NOUT2, DIRIN, DIROUT,CASEID,MEMORY,
2:      &                  NTNUC0,NTFIS0,NPAR0, LCHA0, IBSTEP,NUCLP, GAM,
3:      &                  NBIC, PBIC, DNBURN,NISO, TEMP, VOLM,
4:      &                  TRGNAM,LENTRG,MTBURN,MTCARD,IDENT, DNINIT,
5:      &                  MATMVP,IPBURN,WITHVY0,IARRAY,RARRAY,NIN ,MVPDBG,
6:      &                  HMINV0,IHMIN0,MATREG,TRGPOW,MATPOW,JPCFLG)
7: C=====
8: C Read chain data and check base MVP input data.
9: C=====
10: C
11: C ---- Arguments -----
12: C
13:      character*(*) DIRIN, DIROUT
14:      character*4 CASEID
15:      integer NUCLP(NPAR0,NTNUC0)
16:      real GAM(NTFIS0,NTNUC0)
17:      integer NBIC(NPAR0,NTNUC0)
18:      real PBIC(NPAR0,NTNUC0)
19:
20:      character*12 IDENT(KNMAX,NMAT)
21:      character*12 TRGNAM(NMAT)
22:      integer NISO(NMAT), LENTRG(NMAT)
23:      integer IPBURN(KNMAX,NMAT), MATMVP(NMAT)
24:      integer MTBURN(3,NMAT), MTCARD(2,NMAT)
25:      real TEMP(NMAT), VOLM(NMAT), DNINIT(KNMAX,NMAT)
26:      real DNBURN(NTNUC0,NMAT), WITHVY0(NMAT)
27:      integer IARRAY(MEMORY)
28:      real RARRAY(MEMORY)
29:      real HMINV0(NMAT)
30:      character*12 TRGNMM
31: C
32: C ---- COMMON area -----
33: C
34:      include 'INC/_BURNP'
35: C
36:      integer IBSTEP(MXSTEP)
37: C
38:      include 'INC/_CBURN1'
39:      include 'INC/_CBURN2'
40:      include 'INC/_CBURN3'
41: C
42:      real INSCR, INTCR
43:      common /BNREST/ RSDUMY(3),      NOWSTP, IBEND, AKEFF(MXSTEP),
44:      &      ERKKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
45:      &      EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
46:      &      INTCR(MXSTEP), EINTCR(MXSTEP), EINTCR(MXSTEP),
47:      &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
48:      &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
49:      &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
50:      &      NBATCH(MXSTEP),
51:      &      ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
52:      &      RMONIT(MXSTEP), EMONIT(MXSTEP)
53: C
54:      character*12 TRGPOW(MATPOW)
55:      character*8 CWRK1,CWRK3,CWRK4
56: C
57:      integer      MATREG(NMAT)
58:      integer      JPCFLG(MXSTEP)
59:      real          EXTSIN(MXSTEP),RVMOIN(MXSTEP),REGPIN(MXSTEP)
60: C
61:      real          PERIIN(MXSTEP),POWEIN(MXSTEP)
62:      integer      NSTPIN(MXSTEP)
63: C
64: C FUNCTION DENTAK INTERFACE DATA
65: C

```

```

66: C ..... INPUT BUFFER ....
67: C      MAXCOL : EFFECTIVE CHARACTERS IN AN INPUT-LINE
68: C      parameter( MAXCOL = 72 )
69: C      character MLINE*(MAXCOL)
70: C
71: C ..... CHARACTER LIST .... ( ALPHABETS & NUMBERS ) .....
72: C
73: C      common /FRDATC/ MLINE,      ALPNUM
74: C      common /FRDATA/ NAA,      IS,      IUIN,      IUPR,      NCHAR,      MXCERR,
75: C      &      NCERR
76: C
77: C      real*8 DENTAK,D
78: C
79: C      integer KEEPIC(20)
80: C
81: C      character*72 DDNAME
82: C      character*8 IHOLF(MXNUC), IHOLW(MXNUC)
83: C      character*8 KIDENT, TUNIT, IDREAC, MEMBER
84: C      character*12 NUCID
85: C      character*8 CWRK
86: C      character*8 CWRK2
87: C      character*80 LINE
88: C
89: C      integer NOL(MXNUC), IRESSB(MXNUC)
90: C      real YIELDW(MXNUC)
91: C      integer KPBURN(MXNUC)
92: C
93: C ---- external function -----
94: C
95: C      character*2 STEPNM
96: C      external STEPNM
97: C
98: C *** START OF PROCESS *****
99: C
100: C ***** ZERO CLEAR *****
101: C
102: C
103: C      IF (IBC(8).eq.0.and.IBC(4).eq.0) then
104: C          NOWSTP = 1
105: C          IBEND = 0
106: C      end if
107: C
108: C      CASPCM = ' '
109: C      DDNAME = ' '
110: C
111: C      NEP = IBC(1)
112: C
113: C      if ( NEP.ge.MXSTEP ) then
114: C          write(NOUT1,6888)
115: C          write(NOUT1,*)
116: C      &      ' over limit for burnup steps (MXSTEP=',MXSTEP,')'
117: C          stop 999
118: C      end if
119: C
120: C      NEP1 = IBC(1) + 1
121: C      IBUNIT = IBC(2)
122: C      IBEDIT = IBC(3)
123: C      IBRANC = 0
124: C      if ( IBC(4).eq.1 ) IBRANC = 1
125: C      IBSTND = IBC(5)
126: C      IBCONV = IBC(6)
127: C      IBC(7) = LCHA0
128: C      IBMOD = IBC(8)
129: C      IBNTYP = IBC(9)
130: C      IBFPYL = IBC(11)

```

src/mvpburn/burnip.f

```

131: C
132: C *** OPENING DIROUT FILE IF IBMOD=1 : only 'BURNUP' case !!!
133: C
134:   if ( IBMOD.eq.1 ) then
135:     IBSAVE = NEP
136: C
137:     do 105 I = 1, MXSTEP
138:       PERIIN(I) = PERIOD(I)
139:       POWEIN(I) = POWERL(I)
140:       NSTPIN(I) = NSTP(I)
141:       EXTSIN(I) = EXTSRC(I)
142:       RVMOIN(I) = VMONIT(I)
143:       REGPIN(I) = REGPOW(I)
144: 105 continue
145:
146:     do 106 I = 1, 20
147:       KEEPIC(I) = IBC(I)
148: 106 continue
149:
150:     KEEPIC(8) = 0
151:     call RWCOM1( 'READ', DIROUT, CASEID )
152:
153:     do 107 I = 1, 20
154:       IBC(I) = KEEPIC(I)
155: 107 continue
156:
157:     NEPOLD = NEP
158:     IBC(1) = IBSAVE
159:     NEP = IBSAVE
160:     NEPl = IBSAVE + 1
161: C
162:     if ( IBEDIT.ge.2 ) write(NOUT1,*) ' ** NEP NEPOLD NOWSTP : ',
163: &     NEP,NEPOLD,NOWSTP
164:     if ( IBEDIT.ge.2 ) write(NOUT1,*) ' ** IBNTYP is ',IBNTYP
165: C
166:     if(NEP.lt.NOWSTP) then
167:       write(NOUT1,6888)
168:       write(NOUT1,'(A)')
169: &     ' burnup step is less than calculated step'
170:       write(NOUT1,'(A,I5,A,I5)')
171: &     ' calculated step=',NOWSTP-1,' inputed=',NEP
172:       stop 888
173:     end if
174: C
175: C ... check PERIOD,POWERL,EXTSRC,VMONIT,REGPOW between Old & New values
176: C
177:     do 115 I = 1, NOWSTP-1
178: C
179: C ... check PERIOD & PERIIN
180:       delp = PERIOD(I) - PERIIN(I)
181:       if(PERIIN(I).ne.0.0) delp = delp/PERIIN(I)
182:       if(abs(delp).gt.1.0E-4) then
183:         write(NOUT1,'(A,I5)')
184: &         ' !!! WARNING PERIOD IS NOT EQUAL OLD PERIOD STEP=',I
185:         write(NOUT1,'(A,1P,E12.5,A,1P,E12.5)')
186: &         ' OLD=',PERIOD(I),' NEW=',PERIIN(I)
187:       end if
188: C
189: C ... check POWERL & POWEIN
190: C
191:       if ( IBNTYP.eq.0 ) then
192:         delp = POWERL(I) - POWEIN(I)
193:         if(POWEIN(I).ne.0.0) delp = delp/POWEIN(I)
194:         if ( abs(delp).gt.1.0E-4) then
195:           write(NOUT1,'(A,I5)')

```

```

196: &         ' !!! WARNING POWER IS NOT EQUAL OLD PERIOD STEP=',I
197:         write(NOUT1,'(A,1P,E12.5,A,1P,E12.5)')
198: &         ' OLD=',POWERL(I),' NEW=',POWEIN(I)
199:       end if
200: C
201: C ... check EXTSRC & EXTSIN
202:       else if ( IBNTYP.eq.1 ) then
203:         delp = EXTSRC(I) - EXTSIN(I)
204:         if(EXTSIN(I).ne.0.0) delp = delp/EXTSIN(I)
205:         if ( abs(delp).gt.1.0E-4) then
206:           write(NOUT1,'(A,I5)')
207: &           ' !!! WARNING EXTSRC IS NOT EQUAL OLD PERIOD STEP=',I
208:           write(NOUT1,'(A,1P,E12.5,A,1P,E12.5)')
209: &           ' OLD=',EXTSRC(I),' NEW=',EXTSIN(I)
210:         end if
211: C ... check VMONIT & RVMOIN
212:       else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
213:         delp = VMONIT(I) - RVMOIN(I)
214:         if(RVMOIN(I).ne.0.0) delp = delp/RVMOIN(I)
215:         if ( abs(delp).gt.1.0E-4) then
216:           write(NOUT1,'(A,I5)')
217: &           ' !!! WARNING VMONIT IS NOT EQUAL OLD PERIOD STEP=',I
218:           write(NOUT1,'(A,1P,E12.5,A,1P,E12.5)')
219: &           ' OLD=',VMONIT(I),' NEW=',RVMOIN(I)
220:         end if
221: C ... check REGPOW & REGPIN
222:       else if ( IBNTYP.eq.4 ) then
223:         delp = REGPOW(I) - REGPIN(I)
224:         if(REGPIN(I).ne.0.0) delp = delp/REGPIN(I)
225:         if ( abs(delp).gt.1.0E-4) then
226:           write(NOUT1,'(A,I5)')
227: &           ' !!! WARNING REGPOW IS NOT EQUAL OLD PERIOD STEP=',I
228:           write(NOUT1,'(A,1P,E12.5,A,1P,E12.5)')
229: &           ' OLD=',REGPOW(I),' NEW=',REGPIN(I)
230:         end if
231: C
232:       end if
233: 115 continue
234: C
235:     do 116 I = NOWSTP, NEP
236:       PERIOD(I) = PERIIN(I)
237:       POWERL(I) = POWEIN(I)
238:       NSTP(I) = NSTPIN(I)
239:       if(NSTP(I).eq.0) then
240:         NSTP(I) = 20
241:       end if
242:       EXTSRC(I) = EXTSIN(I)
243:       VMONIT(I) = RVMOIN(I)
244:       REGPOW(I) = REGPIN(I)
245: 116 continue
246: C
247: C *** OUTPUT 'CASECOM1' MEMBER TO DIROUT FILE IF IBMOD=1
248: C
249:       call RWCOM1( 'WRITE', DIROUT, CASEID )
250: C
251:       write(NOUT1,*) ' ** After RWCOM1 : IBMOD case '
252:       write(NOUT1,'(A,20I3)') ' IBC : ',IBC
253: C
254: C
255: C ***** PRINT OUT OF MODIFIED NEP & POWERL & PERIDO DATA
256: C
257:       write(NOUT1,7000) CASEID, TITLE
258:       write(NOUT1,7010) NEP, NEPOLD
259:       if(IBNTYP.eq.0) write(NOUT1,7020) (POWERL(I),I=1,NEP)
260:       if(IBNTYP.eq.1) write(NOUT1,7642) (EXTSRC(I),I=1,NEP)

```

src/mvpburn/burnip.f

```

261:      if(IBNTYP.eq.2) write(NOUT1,7643) (VMONIT(I),I=1,NEP)
262:      if(IBNTYP.eq.3) write(NOUT1,7643) (VMONIT(I),I=1,NEP)
263:      if(IBNTYP.eq.4) write(NOUT1,7644) (REGPOW(I),I=1,NEP)
264:      if(METHPC.eq.1) write(NOUT1,7641) (JPCFLG(I),I=1,NEP)
265:      write(NOUT1,7040) (PERIOD(I),I=1,NEP)
266:      write(NOUT1,7030) (NSTP (I),I=1,NEP)
267: C
268:      return
269: C
270: 7000 format(/1X,15X,' ## BURNUP INPUT MODIFICATION LISTING ## '//7X,
271: & 'CASEID: ',A4/7X,'TITLE : ',A72/7X,'      : ',A72)
272: 7010 format(/1X,5X,
273: & ' NEW NO OF BURNUP STEPS ..... ',I5/1X,5X,
274: & ' OLD NO OF BURNUP STEPS ..... ',I5)
275: 7020 format(/1X,5X,
276: & ' POWER FOR EACH BURNUP STEP (MWT/CM) ..... ',
277: & 'IP,5E12.5:/(57X,1P,5E12.5:))
278: 7040 format(/1X,5X,
279: & ' PERIOD FOR EACH BURNUP STEP ACCORDING IBC2 ..... ',
280: & 'IP,5E12.5:/(57X,1P,5E12.5:))
281: 7030 format(/1X,5X,
282: & ' BURNUP SUB-STEPS NUMBER FOR EACH BURNUP STEP .... ',
283: & '5I12:/(57X,5I12:))
284:      end if
285: C
286: C =====
287: C Read input data
288: C =====
289: C
290: C *** READ BLOCK #6 INPUT DATA
291: C
292: C call REAM( 'DUMM', DUM, POWERL, 0, 0, NEP )
293: C
294:      if(IBRANC.eq.0) then
295:      do 120 I = 1, NEP
296:      if(NSTP(I).eq.0) then
297:      NSTP(I) = 20
298:      end if
299:      if(POWERL(I).eq.0.0) NSTP(I) = 1
300: 120 continue
301:      end if
302: C
303: C *** READ BLOCK #7 INPUT DATA
304: C
305: C call REAM( 'DUMM', PERIOD, PERIOD, 0, 0, NEP )
306: C
307: C *** CHECK PERIOD DATA
308: C
309: Cdel if ( IBRANC.eq.0 ) then
310: Cdel if ( IBUNIT.ne.4 ) then
311: Cdel SAVE = PERIOD(1)
312: Cdel IERR = 0
313: Cdel if ( SAVE.lt.0 ) SAVE = 0.0
314: Cdel do 130 I = 2, NEP
315: Cdel if ( PERIOD(I).gt.0.0 ) then
316: Cdel if ( PERIOD(I).ge.SAVE ) then
317: Cdel SAVE = PERIOD(I)
318: Cdel go to 130
319: Cdel else
320: Cdel IERR = 1
321: Cdel go to 140
322: Cdel end if
323: Cdel else
324: Cdel if ( IBUNIT.le.2 .or. IBUNIT.eq.5 ) then
325: Cdel if ( POWERL(I).eq.0.0 ) go to 130

```

```

326: Cdel end if
327: Cdel IERR = 1
328: Cdel go to 140
329: Cdel end if
330: Cdel 130 continue
331: C
332: Cdel 140 continue
333: Cdel if ( IERR.eq.1 ) then
334: Cdel write(NOUT1,*) ' ** ERROR-STOP : INVALID PERIOD DATA !!!'
335: Cdel write(NOUT1,*)
336: Cdel & ' ** PLEASE CHECK YOUR INPUT DATA & RERUN !!'
337: Cdel stop 888
338: Cdel end if
339: Cdel end if
340: C
341: Cdel if ( IBUNIT.eq.4 ) then
342: Cdel do 150 I = 1, NEP
343: Cdel if ( PERIOD(I).lt.0.0 ) then
344: Cdel write(NOUT1,*)
345: Cdel & ' ** ERROR-STOP : NEGATIVE PERIOD DATA !!!'
346: Cdel write(NOUT1,*)
347: Cdel & ' ** PLEASE CHECK YOUR INPUT DATA & RERUN !!'
348: Cdel stop 888
349: Cdel end if
350: Cdel 150 continue
351: Cdel end if
352: Cdel end if
353: Cdel
354: Cdel *** CHECK IBUNIT HERE !!
355: Cdel
356: Cdel if ( IBRANC.eq.0.and.(IBUNIT.le.0.or.IBUNIT.gt.5) ) then
357: Cdel write(NOUT1,*) ' ** ERROR-STOP AT SUBR(BURNIP) !!'
358: Cdel write(NOUT1,*) ' ** INVALID IBUNIT !!!'
359: Cdel write(NOUT1,*) ' ** INPUT IBUNIT IS ', IBUNIT,
360: Cdel & ' , BUT MUST BE 0<IBUNIT<6 !!!'
361: Cdel stop 888
362: Cdel end if
363: C
364: C *** READ BLOCK #8 INPUT DATA
365: C
366: Cdel if ( IBRANC.eq.1 ) then
367: Cdel call REAM( CASBRN, IDUM, IDUM, 1, 0, 0 )
368: Cdel IVOID = 1
369: Cdel else
370: Cdel CASBRN = ' '
371: Cdel IVOID = 0
372: Cdel end if
373: C
374: C *** READ BLOCK #9 INPUT DATA (useless after ver2)
375: C
376: Cdel if ( METHPC.eq.1 ) then
377: Cdel call REAM( CASPCM, IDUM, DUM, 1, 0, 0 )
378: Cdel else
379: Cdel CASPCM = ' '
380: Cdel end if
381: C
382: C *** READ BLOCK #10 INPUT DATA
383: C
384: Cdel if ( IBSTND.eq.1 ) then
385: Cdel call REAM( STDNUC, IDUM, IDUM, 1, 0, 0 )
386: Cdel else
387: Cdel if ( IBSTND.eq.0 ) then
388: Cdel STDNUC = 922350
389: Cdel end if
390: C

```

src/mvpburn/burnip.f

```

391: C *** READ BLOCK #11-13 INPUT DATA
392: C
393: C   if ( IBC(6).eq.1 ) then
394: C       call REAM( 'DUMM', IDUM, DUM, 0, 2, 0 )
395: C       NFIS = IDUM(1)
396: C       NFER = IDUM(2)
397: C       do 160 M = 1, NFIS
398: C           call REAM( NAMFIS(M), IFISDN(M), FISFCT(M), 1, 1, 1 )
399: C       continue
400: C       do 170 M = 1, NFER
401: C           call REAM( NAMFER(M), IFRTDN(M), FRTFCT(M), 1, 1, 1 )
402: C       continue
403: C   end if
404: C
405: C   if ( NCARD.le.0 ) then
406: C       write(NOUT1,6888)
407: C       write(NOUT1,*) ' no MVP input data, check your input data'
408: C       stop 888
409: C   end if
410: C
411: C
412: C   call RWIND( LIN )
413: C
414: C
415: C =====
416: C   READ CHAIN LIBRARY
417: C =====
418: C
419: C   call RWIND( 93 )
420: C
421: C
422: C   read(93,*) LNMAX, NTNUC, NTFISS, NPAR, NYLDTY
423: C   read(93,*) AWUNIT, ABOGA, EVTOJ
424: C
425: C   ABOGA = ABOGA*1.00000E-24
426: C   LCHA = LCHA0
427: C
428: C   if ( LNMAX.gt.MXNUC ) then
429: C       write(NOUT1,6888)
430: C       write(NOUT1,*) ' over limit for number of depleting nuclides',
431: C   &       ': requested is ', LNMAX, ' but reserved is ', MXNUC
432: C       write(NOUT1,*) ' Change parameter(MXNUC) value in include file'
433: C       stop 999
434: C   end if
435: C
436: C   if ( NTFISS.gt.MXFISS ) then
437: C       write(NOUT1,6888)
438: C       write(NOUT1,*) ' over limit for number of fissionable nuclides',
439: C   &       ': requested is ', NTFISS, ' but reserved is ', MXFISS
440: C       write(NOUT1,*) ' Change parameter(MXNUC) value in include file'
441: C       stop 999
442: C   end if
443: C
444: C   .... READ NUCLIDE INFORMATION
445: C
446: C   do 200 I = 1, LNMAX
447: C       read(93, '(A80)') LINE
448: C       IHOL(I) = LINE(1:8)
449: C       IS = 19
450: C       call NOBLNK( LINE, IS, IE, LEN(LINE) )
451: C       call READI4( LINE, IS, IE, NCODE(I) )
452: C       IS = IE + 1
453: C       call NOBLNK( LINE, IS, IE, LEN(LINE) )
454: C       call READR4( LINE, IS, IE, AMASS(I) )
455: C       IS = IE + 1

```

```

456:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
457:       call READI4( LINE, IS, IE, IFISS(I) )
458:       IS = IE + 1
459:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
460:       call READI4( LINE, IS, IE, IRESSB(I) )
461:       IS = IE + 1
462:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
463:       call READR4( LINE, IS, IE, REACEV(1,I) )
464:       IS = IE + 1
465:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
466:       call READR4( LINE, IS, IE, REACEV(2,I) )
467:       IS = IE + 1
468:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
469:       call READR4( LINE, IS, IE, FACT2N(I) )
470:
471:       AMASS(I) = AWUNIT*AMASS(I)
472: C *** CHECK REACEV : REACEV(2,I) MUST BE ZERO IN PRESENT VERSION
473: C   if ( REACEV(1,I).gt.0.0 ) then
474: C       REACEV(1,I) = REACEV(1,I) + REACEV(2,I)
475: C       REACEV(2,I) = 0.0
476: C   else
477: C       REACEV(1,I) = 0.0
478: C       REACEV(2,I) = 0.0
479: C   end if
480: C   if ( FACT2N(I).le.0.0 ) FACT2N(I) = 1.175000
481: C   200 continue
482: C
483: C   if ( IBEDIT.ge.3 ) then
484: C       do 210 I = 1, LNMAX
485: C           write(NOUT1,7060) I, IHOL(I), NCODE(I), IFISS(I), IRESSB(I),
486: C   &           AMASS(I), (REACEV(J,I),J=1,2), FACT2N(I)
487: C       210 continue
488: C   end if
489: C
490: C   7060 format(1X, ' << #2 >> ', I3, 2X, A8, 2X, 3I6, 2X,      4F10.4)
491: C   7080 format(A8, I2, E10.3, A8)
492: C   7100 format(A8, 2X, A8, 2X, E12.5)
493: C   7120 format(1X, ' error at reading burnup chain library '/1X,
494: C   &       ' reaction('A8,') is not found for parent('A8,
495: C   &       ') => daughter('A8,') chain.')
496: C   7140 format(1X, ' error at reading burnup chain library '/1X,
497: C   &       ' time unit('A8,') is not allowed for 'A8 )
498: C
499: C   READ #3 & #4 FOR CHAIN DATA
500: C
501: C   COEFS = 0.69314718
502: C   COEFM = COEFS/60.0
503: C   COEFH = COEFM/60.0
504: C   COEFD = COEFH/24.0
505: C   COEFY = COEFD/365.0
506: C
507: C   do 240 LOP = 1, NTNUC
508: C       if ( IFISS(LOP).ge.0 ) LASTFP = LASTFP + 1
509: C
510: C       read(93,7080) KDENT, NCH0, HALFT, TUNIT
511: C       if ( IBEDIT.ge.3 ) write(NOUT1,*) KDENT, NCH0, HALFT, TUNIT
512: C
513: C       if ( NCH0.gt.NPAR ) then
514: C           write(NOUT1,6888)
515: C           write(NOUT1,*) ' too many daughter path : ',
516: C   &           ' requested is ', NCH0,
517: C   &           ' but NPAR of chain-library is ', NPAR
518: C       stop 999
519: C   end if
520: C

```

src/mvpburn/burnip.f

```

521:      IPOS      = 0
522:      call HOLPOS( IPOS, KIDENT, LNMAX, IHOL )
523:      NUCL(IPOS) = NCODE(IPOS)
524:      NCH(IPOS)  = NCH0
525:      RAMDA(IPOS) = 0.0
526:      if ( HALFT.gt.0.0 ) RAMDA(IPOS) = 1.0/HALFT
527:      FACT      = 0.0
528:      if ( TUNIT(1:1).eq.'S' ) FACT = COEFS
529:      if ( TUNIT(1:1).eq.'M' ) FACT = COEFM
530:      if ( TUNIT(1:1).eq.'H' ) FACT = COEFH
531:      if ( TUNIT(1:1).eq.'D' ) FACT = COEFD
532:      if ( TUNIT(1:1).eq.'Y' ) FACT = COEFY
533:      RAMDA(IPOS) = FACT*RAMDA(IPOS)
534:      if ( HALFT.gt.0.0.and.FACT.eq.0.0 ) then
535:        write(NOUT1,6888)
536:        write(NOUT1,7140) TUNIT, KIDENT
537:        stop 888
538:      end if
539: C
540:      do 220 J = 1, NPAR
541:        NUCLP(J,IPOS) = 0
542:        NBIC(J,IPOS)  = 0
543:        PBIC(J,IPOS)  = 0.0
544: 220 continue
545: C
546:      if ( NCH0.gt.0 ) then
547: C
548:        do 230 JOP = 1, NCH0
549:          read(93,7100) KIDENT, IDREAC, PROB
550:          JPOS      = 0
551:          call HOLPOS( JPOS, KIDENT, LNMAX, IHOL )
552: C
553:          if ( PROB.lt.0.0 .or. PROB.gt.1.0 ) then
554:            write(NOUT1,6888)
555:            write(NOUT1,*)
556:            &      ' branching ratio is over 1.0 or negative'
557:            write(NOUT1,*) ' Check burnup chain library'
558:            stop 888
559:          end if
560: C
561:          NUCLP(JOP,IPOS) = NCODE(JPOS)
562:          PBIC(JOP,IPOS) = PROB
563:          if ( IDREAC(1:5).eq.'BETA-' ) NBIC(JOP,IPOS) = 1
564:          if ( IDREAC(1:5).eq.'IT   ' ) NBIC(JOP,IPOS) = 2
565:          if ( IDREAC(1:5).eq.'CAPTU' ) NBIC(JOP,IPOS) = 3
566:          if ( IDREAC(1:5).eq.'BETA+' ) NBIC(JOP,IPOS) = 4
567:          if ( IDREAC(1:5).eq.'EC   ' ) NBIC(JOP,IPOS) = 5
568:          if ( IDREAC(1:5).eq.'ALPHA' ) NBIC(JOP,IPOS) = 6
569:          if ( IDREAC(1:5).eq.'DELAY' ) NBIC(JOP,IPOS) = 7
570:          if ( IDREAC(1:5).eq.'2N   ' ) NBIC(JOP,IPOS) = 8
571:          if ( NBIC(JOP,IPOS).eq.0 ) then
572:            write(NOUT1,6888)
573:            write(NOUT1,7120) IDREAC, KIDENT, IHOL(IPOS)
574:            stop 888
575:          end if
576: 230 continue
577:        end if
578: 240 continue
579: C
580:      if ( IBEDIT.ge.3 ) then
581:        do 260 I = 1, NTNUC
582:          write(NOUT1,7160) I, IHOL(I), NCODE(I), NCH(I), RAMDA(I)
583:          do 250 J = 1, NCH(I)
584:            write(NOUT1,7180) J, NUCLP(J,I), NBIC(J,I), PBIC(J,I)
585: 250 continue

```

```

586: 260 continue
587:      write(NOUT1,7220) LNMAX, NTNUC, NTFISS, LASTFP
588:      end if
589: C
590: 7160 format(1X,' << #3 >> ',I3,2X,A8,2X,2I6,1P,E12.5,' (/SEC)')
591: 7180 format(1X,' << #4 >> ',I3,2I6,F10.6)
592: 7200 format(5(A8,2X))
593: 7220 format(1X,' ## LBNAX,NTNUC,NTFISS,LASTFP ## ',6I6)
594: C
595: C      READ #5,#6,#7 FP YIELD DATA
596: C
597:      call CLEA( GAM, NTFIS0*NTNUC0, 0.0 )
598: C
599: C +++ for Non-depletion chain model for parametric survey
600: C
601:      if ( NYLDTY.gt.0 ) then
602:        do 360 LOP = 1, NYLDTY
603:          read(93,'(A80)') LINE
604:          KIDENT = LINE(1:8)
605:          IS      = 9
606:          call NOBLNK( LINE, IS, IE, LEN(LINE) )
607:          call READI4( LINE, IS, IE, NYNUCL )
608:          IS      = IE + 1
609:          call NOBLNK( LINE, IS, IE, LEN(LINE) )
610:          call READI4( LINE, IS, IE, NFP )
611:
612:          read(93,7200) (IHOLF(I),I=1,NYNUCL)
613:          do 270 I = 1, LNMAX
614:            YIELDW(I) = 0.0
615: 270 continue
616:
617:          SUM      = 0.0
618:          do 280 I = 1, NFP
619:            read(93,'(A80)') LINE
620:            IHOLF(I) = LINE(1:8)
621:            IS      = 9
622:            call NOBLNK( LINE, IS, IE, LEN(LINE) )
623:            call READR4( LINE, IS, IE, YIELDW(I) )
624:            SUM      = SUM + YIELDW(I)
625: 280 continue
626:
627:          if ( IBEDIT.ge.4 ) then
628:            write(NOUT1,7240) LOP, KIDENT, NYNUCL, NFP, SUM
629:            write(NOUT1,7260) (IHOLF(I),I=1,NYNUCL)
630:            do 290 I = 1, NFP
631:              write(NOUT1,7280) I, IHOLF(I), YIELDW(I)
632: 290 continue
633:          end if
634:
635:          do 350 J = 1, NYNUCL
636:            KIDENT = IHOLF(J)
637:            do 300 JJ = 1, NTFISS
638:              JPOS = JJ
639:              if ( KIDENT.eq.IHOL(JJ) ) go to 310
640: 300 continue
641:            write(NOUT1,7300) KIDENT
642:            go to 350
643:
644: 310 continue
645:          do 340 I = 1, NFP
646:            KIDENT = IHOLF(I)
647:            if ( NTFISS.lt.NTNUC ) then
648:              do 320 III = NTFISS + 1, NTNUC
649:                IPOS = III
650:                if ( KIDENT.eq.IHOL(III) ) go to 330

```

src/mvpburn/burnip.f

```

651:      320          continue
652:          write(NOUT1,7300) KDENT
653:          go to 340
654:      330          continue
655:          GAM(JPOS,IPOS) = YIELDW(I)
656:          end if
657:      340          continue
658:      350          continue
659:      360          continue
660: C
661: C --- for Non-depletion chain model for parametric survey
662: C
663: C end if
664: C
665: C if ( IBEDIT.ge.4 ) then
666: C   do 380 LOP = 1, NFISS
667: C     write(NOUT1,7320) IHOL(LOP)
668: C     SUM = 0.0
669: C     do 370 J = 1, NTFUC
670: C       write(NOUT1,7340) IHOL(J), GAM(LOP,J)
671: C       SUM = SUM + GAM(LOP,J)
672: C   370      continue
673: C     write(NOUT1,7360) SUM
674: C   380      continue
675: C   end if
676: C
677: C 7240 format(1X,' << #5 >> ',I3,2X,A8,2X,2I6,1P,E12.5)
678: C 7260 format(1X,' << #6 >> ',10(A8,2X)/1X,' << #6 >> ',10(A8,2X)/1X,
679: C   &      ' << #6 >> ',10(A8,2X))
680: C 7280 format(1X,' << #7 >> ',I3,2X,A8,2X,1P,E12.5)
681: C 7300 format(1X,2X,A8,' IS NOT FOUND IN IHOL TABLE !!')
682: C 7320 format('1',' ## YIELD DATA FOR ',A8,' ##')
683: C 7340 format(1X,' << KDENT YIELD >> ',A8,2X,1P,E12.5)
684: C 7360 format(1X,' << SUM OF YIELD >> ',1P,E12.5)
685: C
686: C      call RWIND( 93 )
687: C
688: C =====
689: C *** SET DEFAULT CONVERSION RATIO DEFINISION
690: C =====
691: C
692: C if ( NFIS.eq.0.and.NFER.eq.0 ) then
693: C   do 390 I = 1, NTFISS
694: C     if ( IFISS(I).eq.3 ) then
695: C       NFIS = NFIS + 1
696: C       NAMFIS(NFIS) = NCODE(I)*10 + 4
697: C       IFISDN(NFIS) = 1
698: C       FIFCT(NFIS) = 1.0
699: C     end if
700: C     if ( IFISS(I).eq.2 ) then
701: C       NFER = NFER + 1
702: C       NAMFER(NFER) = NCODE(I)*10 + 2
703: C       IFRTDN(NFER) = 1
704: C       FRTFCT(NFER) = 1.0
705: C     end if
706: C     if ( IFISS(I).eq.4 ) then
707: C       NFER = NFER + 1
708: C       NAMFER(NFER) = NCODE(I)*10 + 5
709: C       IFRTDN(NFER) = 1
710: C       FRTFCT(NFER) = 1.0
711: C     end if
712: C   390      continue
713: C   end if
714: C
715: C =====

```

```

716: C **** CHECK ORIGINAL MVP INPUT DATA & SET MATERIAL DATA
717: C =====
718: C
719: C      LIN = 92
720: C      ICNT = 0
721: C      call RWIND( LIN )
722: C
723: C DENTAK INITIALIZATION
724: C
725: C      call FRINIT( LIN, NOUT1, 10)
726: C      IS = 0
727: C      D = DENTAK('&&CLEAR', NOUT1, IERR)
728: C      IMICRO = 0
729: C
730: C 400 continue
731: C      ICNT = ICNT + 1
732: C      if ( ICNT.gt.NCARD ) then
733: C        if ( ICNT.ge.NCARD ) then
734: C          write(NOUT1,6888)
735: C          write(NOUT1,*) ' Perhaps tally-region data is not defined ',
736: C            &      ' in MVP input data '
737: C          write(NOUT1,*) ' NCARD is ', NCARD
738: C          stop 888
739: C        end if
740: C      read(LIN,'(A)') MLINE
741: C      call GETLN2(LIN, NOUT1, MLINE, NCHAR, IEND, ICNT)
742: C
743: C .... "EDIT-MICRO..." option is necessary for burnup ....
744: C
745: C      if ( MLINE(1:5).ne.'$XSEC'.and.MLINE(1:6).ne.'$CROSS' ) then
746: C
747: C        IANS = 0
748: C        call CHKWRD( MLINE, 'EDIT-MICRO', 10, IANS )
749: C        if ( IANS.eq.0 ) go to 400
750: C
751: C        JANS = 0
752: C        call CHKWRD( MLINE, 'NO-EDIT-MICRO', 13, JANS )
753: C        if ( JANS.eq.1 ) then
754: C          write(NOUT1,6888)
755: C          write(NOUT1,*) ' Perhaps microscopic-editting option is ',
756: C            &      ' not specified in MVP input data '
757: C          stop 888
758: C        end if
759: C
760: C      call GETMIC( MLINE, IMICRO )
761: C      go to 400
762: C    end if
763: C
764: C .... "$XSEC" or "$CROSS SECTION"
765: C
766: C ... check EDIT-MICRO option value ....
767: C
768: C      CWRK = ' '
769: C      write(CWRK,'(I8.8)') IMICRO
770: C      if ( IBEDIT.gt.2 ) write(NOUT1,*) ' ## IMICRO : ', CWRK
771: C
772: C
773: C      IERR = 0
774: C      if ( CWRK(3:3).eq.'0' ) IERR = 1
775: C      if ( CWRK(5:5).eq.'0' ) IERR = 1
776: C      if ( CWRK(7:7).eq.'0' ) IERR = 1
777: C      if ( IERR.ne.0 ) then
778: C        write(NOUT1,6888)
779: C        write(NOUT1,*) ' value of EDIT-MICRO option in MVP input ',
780: C          &      ' data is not appropriate'

```

src/mvpburn/burnip.f

```

781:      stop 888
782:      end if
783: C
784:      MPOS      = 0
785: C
786: C      ---- input burnup material composition ----
787: C
788: 410 continue
789: C      read(LIN,'(A)') MLINE
790: C      ICNT      = ICNT + 1
791: C      call GETLN2(LIN, NOUT1, MLINE, NCHAR, IEND, ICNT)
792: C
793: C      if ( MLINE(1:9).eq.'$END XSEC' .or. MLINE(1:10).eq.'$END CROSS' )
794: C      & go to 440
795: C      if ( MLINE(1:9).ne.'$END XSEC' .and. MLINE(1:10).ne.'$END CROSS' )
796: C      & then
797: C
798: C      if ( MLINE(1:8).eq.'*MVPBURN' ) then
799: C      MPOS      = MPOS + 1
800: C      call GETVOL( MLINE, VOLM(MPOS) )
801: C      call GETTRG( MLINE, TRGNAM(MPOS), LENTRG(MPOS) )
802: C      call GETTMP( MLINE, TEMP(MPOS) )
803: C      if (VOLM(MPOS).le.0.0) then
804: C      write(NOUT1,6888)
805: C      write(NOUT1,*) ' volume data is zero or negative for ',
806: C      & 'depleting material in MVP-BURN input'
807: C      write(NOUT1,*)
808: C      & ' Material Number=',MPOS,' Volume =',VOLM(MPOS)
809: C      call CNTERR('FATAL')
810: C      end if
811: C      if (TEMP(MPOS).le.0.0) then
812: C      write(NOUT1,6888)
813: C      write(NOUT1,*)
814: C      & ' temperature data is zero or negative for ',
815: C      & 'depleting material in MVP-BURN input'
816: C      write(NOUT1,*)
817: C      & ' Material Number=',MPOS,' Temperature =',TEMP(MPOS)
818: C      call CNTERR('FATAL')
819: C      end if
820: C      call GETNIS( MLINE, NISO(MPOS) )
821: C      MTCARD(1,MPOS) = ICNT + 1
822: C
823: C      ICNT2 = 0
824: 430 continue
825: C
826: 420 continue
827: C      call GETLN2(LIN, NOUT1, MLINE, NCHAR, IEND, ICNT)
828: C      if ( MLINE(1:1) .eq.'*' ) go to 420
829: C      if ( MLINE(1:72).eq.' ' ) go to 420
830: C      if ( MLINE(1:1) .eq.'@' ) go to 420
831: C      if ( MLINE(1:1) .eq.'&' ) go to 431
832: C      if ( MLINE(1:9) .eq.'$END XSEC' ) go to 431
833: C      if ( MLINE(1:10) .eq.'$END CROSS' ) go to 431
834: C      ICNT2 = ICNT2 + 1
835: C      SAVE      = 0.0
836: C      call GETDEN( MLINE, SAVE, NUCID )
837: C      DNINIT(ICNT2,MPOS) = SAVE
838: C
839: C      if ( SAVE.lt.0.0 ) then
840: C      write(NOUT1,6888)
841: C      write(NOUT1,*)
842: C      & ' negative nuclide density data was found ',
843: C      & 'in MVP input data '
844: C      stop 888
845: C      end if

```

```

846: C
847: C      call SETMID( NUCID, TEMP(MPOS) )
848: C      IDENT(ICNT2,MPOS) = NUCID
849: C      goto 430
850: 431 continue
851: C      ICNT = ICNT-1
852: C      backspace LIN
853: C      NISO(MPOS) = ICNT2
854: C      MTCARD(2,MPOS) = ICNT
855: C      go to 410
856: C      end if
857: C
858: C      if ( MLINE(1:1).eq.'*' ) go to 410
859: C      if ( MLINE(1:1).eq.'&' ) go to 410
860: C980820 ....
861: C      if ( MLINE(1:1).eq.'%' ) go to 410
862: C
863: C      if ( MLINE(1:10).eq.' ' ) go to 410
864: C
865: C      call GETMID( MLINE, NUCID )
866: C      IANS      = 0
867: C      call CHKPID( NUCID, IANS )
868: C      if ( IANS.eq.0 ) then
869: C      write(NOUT1,6888)
870: C      write(NOUT1,*) ' Perhaps nuclide ID name in MVP input data '
871: C      & ', is out of rule for MVP-BURN'
872: C      write(NOUT1,*) ' input nuclide ID name is ', NUCID
873: C      stop 888
874: C      end if
875: C
876: C      go to 410
877: C
878: C      end if
879: C
880: C
881: C
882: 440 continue
883: C      ICNT      = ICNT + 1
884: C      if ( ICNT.gt.NCARD ) then
885: C      write(NOUT1,6888)
886: C      write(NOUT1,*) ' Perhaps tally region is not defined in ',
887: C      & 'MVP input data'
888: C      stop 888
889: C      end if
890: C
891: C      read(LIN,'(A)') MLINE
892: C
893: C      if ( MLINE(1:8).ne.'#TALLY R' ) go to 440
894: C
895: 450 continue
896: C      ICNT      = ICNT + 1
897: C      if ( ICNT.gt.NCARD ) then
898: C      write(NOUT1,6888)
899: C      write(NOUT1,*) ' "DEFINE" data is missing in MVP input data'
900: C      stop 888
901: C      end if
902: C
903: C      read(LIN,'(A)') MLINE
904: C
905: C      IANS = 0
906: C      call CHKWRD(MLINE, 'DEFINE', 6, IANS)
907: C      if(IANS.eq.0) go to 450
908: C      NOTRG = 0
909: C      call TRGCHK(MLINE, TRGNMM, LENTRM, IANS, 1, NOUT1)
910: C      if(IANS.eq.1) then

```


src/mvpburn/burnip.f

```

911:      NOTRG = NOTRG + 1
912:      do 460 M = 1, NMAT
913:        if( LENTRG(M).eq.LENTRM ) then
914:          if( TRGNMM(1:LENTRM).eq.TRGNAM(M)(1:LENTRG(M)) )
915:            &      MATMVP(M) = NOTRG
916:          end if
917: 460      continue
918:        end if
919: C
920: 470 ICNT = ICNT + 1
921:      if ( ICNT.le.NCARD ) then
922:        read(LIN,'(A)') MLINE
923:        if ( MLINE(1:1).eq.'*' ) go to 470
924:        if ( MLINE(1:9).ne.'$END GEOM' ) then
925:          CALL TRGCHK( MLINE, TRGNMM, LENTRM, IANS, 0, NOUT1)
926:          if ( IANS.eq.1 ) then
927:            NOTRG = NOTRG + 1
928:            do 480 M = 1, NMAT
929:              if ( LENTRG(M).eq.LENTRM ) then
930:                if ( TRGNMM(1:LENTRM).eq.TRGNAM(M)(1:LENTRG(M)) )
931:                  &      MATMVP(M) = NOTRG
932:              end if
933: 480          continue
934:            end if
935:            go to 470
936:          end if
937:        end if
938:        call CHKERR( 'WARNING', KK )
939:        if ( KK.gt.0 ) then
940:          write(NOUT1,7125)
941: 7125      format('*** some WARNING messages have been issued. ****')
942:          &      ' I recommend you checking your input data',
943:          &      ' once more.'// ' WARNING message is printed ',
944:          &      'with "!!!" symbol on the head of printed line.//')
945:        end if
946:        call CHKERR( 'FATAL', KK )
947:        if ( KK.gt.0 ) then
948:          write(NOUT1,7145)
949:          write(NOUT1,6888)
950: 7145      format('1','*** fatal ERRORS have been detected',
951:          &      ' in input-phase. stop execution. ****')
952:          &      ' message for fatal ERROR is printed ',
953:          &      'with "XXX" symbol on the head of printed line.//')
954:          &      ' you must check your input data',' carefully !!!')
955:        stop 888
956:      end if
957: C
958: C ----- End of MVP input check
959: C
960:      call RWIND( LIN )
961: C
962: C
963:      do 500 M = 1, NMAT
964:        if ( MATMVP(M).le.0 ) then
965:          write(NOUT1,6888)
966:          write(NOUT1,*) ' tally region is not defined for ',
967:          &      TRGNAM(M), '.'
968:          write(NOUT1,*) ' Check your MVP input data !'
969:          stop 888
970:        end if
971: 500      continue
972: C
973: C
974: C *** SET MATREG
975: C

```

```

976:      call ICLEA( MATREG , NMAT , 1 )
977: C
978:      if( IBC(9).eq.4.and.IBRANC.eq.0 ) then
979:        call ICLEA( MATREG , NMAT , 0 )
980: C
981:        do 505 J = 1 , MATPOW
982:          LENG1 = ICLEN2(TRGPOW(J))
983:          MSW = 0
984:          do 503 M = 1 , NMAT
985:            LENG2 = ICLEN2(TRGNAM(M))
986:            if ( LENG1.eq.LENG2 ) then
987:              if ( TRGNAM(M)(1:LENG2).eq.TRGPOW(J)(1:LENG1)) then
988:                MATREG(M) = 1
989:                MSW = 1
990:              end if
991:            end if
992: 503          continue
993: C
994:            if( MSW.le.0 ) then
995:              write(NOUT1,6888)
996:              write(NOUT1,*) ' ** input TRGPOW(' ,TRGPOW(J)(1:LENG1),
997:              &      ' ) was not found in TRGNAM names !!! '
998:              write(NOUT1,*) ' ** input TRGNAM are : '
999:              write(NOUT1,'(5(A12,3X))') (TRGNAM(I),I=1,NMAT)
1000:              stop 888
1001:            end if
1002: C
1003: 505          continue
1004:        end if
1005: C
1006: C *** SET IPBURN,MTBURN
1007: C
1008:      NTNUCX = 0
1009: C
1010:      do 540 M = 1, NMAT
1011:        MMK = NISO(M)
1012:        do 520 I = 1, MMK
1013:          call ZZMM(0,1,ICODE2,IDENT(I,M)(1:6))
1014:          if ( ICODE2.gt.0 ) then
1015:            do 510 K = 1, NTNUC
1016:              if ( ICODE2.eq.NCODE(K) ) then
1017:                IPBURN(I,M) = K
1018:                DNBURN(K,M) = DNINIT(I,M)
1019:              end if
1020: 510            continue
1021:          end if
1022: 520          continue
1023: C
1024:          MTBURN(1,M) = NTNUC
1025:          MTBURN(2,M) = 0
1026:          MTBURN(3,M) = 0
1027:          do 530 I = 1, MMK
1028:            ISAVE = IPBURN(I,M)
1029:            if ( ISAVE.gt.0 ) then
1030:              if ( ISAVE.lt.MTBURN(1,M) ) MTBURN(1,M) = ISAVE
1031:              if ( ISAVE.gt.MTBURN(2,M) ) MTBURN(2,M) = ISAVE
1032:            end if
1033: 530          continue
1034: C
1035: C=====
1036: C Search nuclides to be treated in MVP input
1037: C MTBURN(1,m) : position of the first burnable nuclide in the chain
1038: C MTBURN(2,m) : position of the last burnable nuclide in the chain
1039: C MTBURN(3,m) : material type
1040: C NTNUCX      : total number of burnable nuclides in all materials(m)

```

src/mvpburn/burnip.f

```

1041: C          NTNUC is replaced by NTNUCX
1042: C*****
1043: C --- CASE FOR FUEL MATERAIL
1044: C*****
1045:       if ( MTBURN(1,M).le.NTFISS ) then
1046:           MTBURN(1,M) = 1
1047: C
1048: CKSK       if ( MTBURN(2,M).le.LASTFP ) MTBURN(2,M) = LASTFP
1049: CKSK : We must consider BP nuclides in fuel. BP nuclides may have
1050: CKSK decay-chain to lower position
1051: CKSK Therefore, all nuclide in BP-chain must be included,
1052: CKSK otherwise, MVP reaction rate can not be obtained.
1053: CKSK => zero division in fnbate routine
1054: CKSK MVP library must be prepared also for not-necessary nuclides.
1055: C
1056:       if ( MTBURN(2,M).le.LASTFP ) then
1057:           MTBURN(2,M) = LASTFP
1058:       else
1059:           MTBURN(2,M) = NTNUC
1060:       end if
1061:       MTBURN(3,M) = 1
1062:       go to 1000
1063:     end if
1064: C*****
1065: C --- CASE FOR ABSORBER IN FP-CHAIN ( Ex. Gd Absorbor in non-fuel)
1066: C*****
1067:       if ( MTBURN(1,M).gt.NTFISS.and.MTBURN(2,M).le.LASTFP ) then
1068: CKSK decay in FP-chain is considered
1069: CKSK but we assume no EC or beta+ decay pathes from the absorbers
1070:       MTBURN(2,M) = LASTFP
1071:       MTBURN(3,M) = 2
1072:       go to 1000
1073:     end if
1074: C*****
1075: C --- CASE FOR ABSORBER IN BP-CHAIN (Ex. B-10, Hf, Er in BP-chain)
1076: C*****
1077:       if ( MTBURN(1,M).gt.LASTFP ) then
1078: CKSK decay in BP-chain is considered
1079: CKSK but we assume no EC or beta+ decay pathes from the absorbers
1080:       MTBURN(2,M) = NTNUC
1081:       MTBURN(3,M) = 3
1082:       go to 1000
1083:     end if
1084: C*****
1085: C --- CASE FOR ABSORBER NUCLIDES IN FP-CHAIN AND BP-CHAIN
1086: C*****
1087: CKSK We assume no EC or beta+ decay pathes from the absorbers
1088: CKSK in FP-chain
1089:       if ( MTBURN(1,M).le.LASTFP.and.MTBURN(2,M).gt.LASTFP ) then
1090:           MTBURN(2,M) = NTNUC
1091:           MTBURN(3,M) = 4
1092:           go to 1000
1093:       end if
1094: C*****
1095: C --- CASE FOR OTHERS (error stop)
1096: C*****
1097:       if( MTBURN(3,M).eq.0 ) then
1098:           write(NOUT1,6888)
1099:           write(NOUT1,7350) M, MTBURN(1,M), MTBURN(2,M), MTBURN(3,M),
1100:           &          NTNUC
1101:           stop 999
1102:       end if
1103: C
1104:       if( MTBURN(1,M).gt.MTBURN(2,M)) then
1105:           write(NOUT1,6888)

```

```

1106:           write(NOUT1,7350) M, MTBURN(1,M), MTBURN(2,M), MTBURN(3,M),
1107:           &          NTNUC
1108:           stop 999
1109:       end if
1110: C
1111:       if( MTBURN(2,M).gt.NTNUC) then
1112:           write(NOUT1,6888)
1113:           write(NOUT1,7350) M, MTBURN(1,M), MTBURN(2,M), MTBURN(3,M),
1114:           &          NTNUC
1115:           stop 999
1116:       end if
1117: C
1118: 7350 format(1X,'Chain model may be out of consideration in program. '/
1119: & '=> Contact to developpers !'
1120: & /' Material number (M)=' , I4
1121: & /' First nuclide position in chain MTBURN(1,M) =', I4
1122: & /' Last nuclide position in chain MTBURN(2,M) =', I4
1123: & /' Type number of burnable material MTBURN(3,M) =', I4
1124: & /' Total number of nuclides in chain model      =', I4 )
1125: C=====
1126: C
1127: 1000 continue
1128:       if ( MTBURN(2,M).gt.LASTFP ) MTBURN(2,M) = NTNUC
1129: C
1130: C*****
1131: C Set real maximum number of nuclides to be treated in each material
1132: C*****
1133:       if ( NTNUCX.lt.MTBURN(2,M) ) NTNUCX = MTBURN(2,M)
1134: C
1135: 540 continue
1136: C
1137: C *** RESET NTNUC
1138:       NTNUC = NTNUCX
1139: C
1140: C **** KEEP ID NUMBER OF U-235 & XE-135 , I-135 , SM-149 , PM-149
1141: C
1142:       IDU235 = 0
1143:       IDXE35 = 0
1144:       IDI135 = 0
1145:       IDSM49 = 0
1146:       IDPM49 = 0
1147:       if(IBFPYL.eq.0) then
1148:           NCXE35 = 541350
1149:           NCI135 = 531350
1150:           NCSM49 = 621490
1151:           NCPM49 = 611490
1152:       end if
1153:       do 550 I = 1, NTNUC
1154:           if ( NCODE(I).eq.STDNUC ) IDU235 = I
1155:           if ( NCODE(I).eq.NCXE35 ) IDXE35 = I
1156:           if ( NCODE(I).eq.NCI135 ) IDI135 = I
1157:           if ( NCODE(I).eq.NCSM49 ) IDSM49 = I
1158:           if ( NCODE(I).eq.NCPM49 ) IDPM49 = I
1159:       550 continue
1160: C
1161:       if ( IDU235.eq.0 ) then
1162:           do 560 I = 1, NTFISS
1163:               if ( IFISS(I).eq.3 ) then
1164:                   IIWRK = NCODE(I)
1165:                   IDU235 = I
1166:                   go to 570
1167:               end if
1168:           560 continue
1169: C
1170:           write(NOUT1,6888)

```

src/mvpburn/burnip.f

```

1171:      write(NOUT1,*) ' standard nuclide (', STDNUC,') as ',
1172:      & 'exposure indicator was not found in burnup chain library.'
1173:      stop 888
1174: C
1175: 570 continue
1176: C
1177: C=====
1178:      write(NOUT1,*) ' STDNUC (nuclide as exposure indicator) will ',
1179:      & 'be replaced from ', STDNUC, ' to ', IIWRK
1180:      STDNUC = IIWRK
1181:      end if
1182: C
1183:      if ( IBEDIT.ge.2 ) then
1184:          call ZZMMM(1,1,STDNUC,CWRK(1:6))
1185:          call ZZMMM(1,1,NCXE35,CWRK1(1:6))
1186:          call ZZMMM(1,1,NCI135,CWRK2(1:6))
1187:          call ZZMMM(1,1,NCSM49,CWRK3(1:6))
1188:          call ZZMMM(1,1,NCPM49,CWRK4(1:6))
1189:          write(NOUT1,7380) CWRK(1:6), IDU235,CWRK1(1:6), IDXE35,
1190:          & CWRK2(1:6), IDI135,CWRK3(1:6), IDSM49,CWRK4(1:6), IDPM49
1191:          end if
1192: C
1193: 7380 format(/1X,'ID NUMBER OF ',A,' =',I4/1X,'ID NUMBER OF ',A,' =',
1194:      & I4,9X,'ID NUMBER OF ',A,' =',I4/1X,'ID NUMBER OF ',A,
1195:      & ' =',I4,9X,'ID NUMBER OF ',A,' =',I4)
1196: 7400 format(1X,'NMP = ',I3,' : NMAT = ',I3,' : NTDEPZ = ',I3)
1197: 7420 format(1X,' >> MATD : ',20I5)
1198: 7440 format(1X,' >> NDEPZ : ',20I5)
1199: 7460 format(1X,' >> MATDPL: ',20I5)
1200: C
1201: C **** SET ST235 & WTHVY0 & TWTHVY
1202: C
1203:      ST235 = 0.0
1204:      TWTHVY = 0.0
1205: C
1206:      do 590 M = 1, NMAT
1207:          ST235 = ST235 + VOLM(M)*DNBURN(IDU235,M)
1208: C
1209: C *** IHMIN0 > 0 : Initial Heavy Nuclide weight is input
1210: C
1211:      if ( IHMIN0.eq.0 ) then
1212:          WTSAVE = 0.0
1213:          do 580 I = 1, NTFISS
1214:              WTSAVE = WTSAVE + AMASS(I)*DNBURN(I,M)
1215: 580 continue
1216:          SAVE = WTSAVE/ABOGA
1217:          WTHVY0(M) = SAVE*1.000E-6
1218:      else
1219:          WTHVY0(M) = HMINV0(M)
1220:      end if
1221: C
1222:      if ( IBEDIT.gt.2 ) write(NOUT1,*) ' M WT(GRAM) : ', M, SAVE
1223: C
1224:          TWTHVY = TWTHVY + VOLM(M)*WTHVY0(M)
1225: 590 continue
1226: C
1227:      ST235 = ST235*1.00000E+24
1228: C
1229:      if ( IBEDIT.gt.2 ) write(NOUT1,*) ' ST235 TWTHVY : ', ST235,
1230:      & TWTHVY
1231: C
1232: C *** CHECK STANDARD NUCLIDE DENSITY IF IBC(2)=5
1233: C
1234:      if ( ST235.le.0.0 ) then
1235:          if ( IBC(2).eq.5 ) then

```

```

1236:          write(NOUT1,6888)
1237:          write(NOUT1,*) ' initial density of ', STDNUC,
1238:          & '(standard nuclide as exposure indicator) is zero or ',
1239:          & 'negative'
1240:          write(NOUT1,*) ' Use other nuclides for STDNUC.'
1241:          stop 888
1242:      else
1243:          ST235 = 1.00000
1244:      end if
1245:      end if
1246: C
1247:      if ( IBEDIT.ge.4 ) then
1248:          write(NOUT1,*) '** CHAIN-CHECK-WRITE AT BURNIP ** '
1249:          do 610 L = 1, NTNUC
1250:              write(NOUT1,7480) L, NUCL(L), NCH(L), RAMDA(L)
1251:              do 600 LN = 1, NCH(L)
1252:                  write(NOUT1,7500) LN, NUCLP(LN,L), NBIC(LN,L), PBIC(LN,L)
1253: 600 continue
1254: 610 continue
1255:          end if
1256: C
1257: 7480 format(/1X,' ## L NUCL NCH RAMDA ## ',3I8,1P,E12.5)
1258: 7500 format(1X,' << LN NUCLP NBIC PBIC >> ',3I8,F10.5)
1259: C
1260: C DEFINE NCHA PARAMETER
1261: C
1262:      NCHA0 = (MEMORY-NPAR*NTNUC) / (3*NTNUC+2*LCHA*NTNUC)
1263:      LOC1 = 1
1264:      LOC2 = LOC1 + NCHA0*NTNUC
1265:      LOC3 = LOC2 + NCHA0*NTNUC
1266:      LOC4 = LOC3 + NCHA0*NTNUC
1267:      LOC5 = LOC4 + NCHA0*NTNUC*LCHA
1268:      LOC6 = LOC5 + NCHA0*NTNUC*LCHA
1269:      LOC7 = LOC6 + NCHA0*NTNUC*LCHA
1270: C -----
1271: C
1272:      call ICLEA( NOL, MXNUC, 0 )
1273: C
1274:      KTYPE = 1
1275:      call LNCHAI( NUCL, RAMDA, NCH, NUCLP, NBIC, PBIC, NOL,
1276:      & IARRAY(LOC1), IARRAY(LOC2), IARRAY(LOC3), IARRAY(LOC4),
1277:      & IARRAY(LOC5), IARRAY(LOC6), NTNUC, NPAR, NCHA0, LCHA,
1278:      & KTYPE )
1279: C
1280: C *** check input density data for BP nuclide
1281: C
1282:      NUCERR = 0
1283:      do 1660 M = 1, NMAT
1284:          MMK = NISO(M)
1285:          IST = MTBURN(1,M)
1286:          IEND = MTBURN(2,M)
1287:          RTEMP = TEMP(M)
1288:          MTYP = MTBURN(3,M)
1289:          KERR = 0
1290: C
1291:      call ICLEA(KPBURN, MXNUC, 0)
1292: C
1293:      do 1620 I = IST, IEND
1294:          ISW = 0
1295:          do 1610 K = 1, MMK
1296:              if ( IPBURN(K,M).eq.I ) ISW = K
1297: 1610 continue
1298: C
1299:          if(MTYP.eq.1 .and. I.le.LASTFP) then
1300:              KPBURN(I) = 1

```

src/mvpburn/burnip.f

```

1301:         else
1302:             if (ISW.gt.0) then
1303:                 KPBURN(I) = 1
1304:             end if
1305:         end if
1306: 1620      continue
1307: C
1308:         if (MTYP.eq.1) IST = LASTFP + 1
1309: C
1310:         call CHKCHN(NUCL, RAMDA, NCH, NUCLP, NBIC, PBIC, NOL,
1311: & IARRAY(LOC1), IARRAY(LOC2), IARRAY(LOC3), IARRAY(LOC4),
1312: & IARRAY(LOC5), IARRAY(LOC6), NTNUC, NPAR, NCHA0, LCHA,
1313: & IST, IEND, KPBURN, KERR)
1314: C
1315:         NUCERR = NUCERR + KERR
1316: C
1317:         if (KERR.gt.0) then
1318:             write(NOUT1,*)
1319: & ' ** ',M,'-th material composition input error !!!'
1320:         do 1650 I = IST, IEND
1321:             if (KPBURN(I).lt.0) then
1322:                 ... make nuclide ID from NCODE
1323:                 call ZZMMM(1,1,NCODE(I),NUCID)
1324:                 if (NUCID(6:6).eq.' ') NUCID(6:6) = '0'
1325:                 if (NUCID(6:6).eq.'M') NUCID(6:6) = '1'
1326:                 NUCID(7:10) = '0000'
1327:             ... end
1328:             call SETMID(NUCID, RTEMP)
1329:             write(NOUT1,*) ' Initial density data must be inputted',
1330: & ' for ',NUCID,' even if zero density !!!'
1331:             write(NOUT1,*) ' And ',NUCID,' MVP library must be',
1332: & ' prepared and assigned in MVP index file !!!'
1333:         end if
1334: 1650      continue
1335:         end if
1336: 1660      continue
1337: C
1338:         if (NUCERR.gt.0) then
1339:             write(NOUT1,*) ' ** Fatal material composition input error',
1340: & ' was found in sub(BURNIP) !!!'
1341:             write(NOUT1,6888)
1342:             stop 999
1343:         end if
1344: C
1345:         NCHA = 0
1346:         do 620 I = 1, NTNUC
1347:             if (NOL(I).gt.NCHA) NCHA = NOL(I)
1348: 620      continue
1349: C
1350:         if (IBEDIT.ge.3) then
1351:             write(NOUT1,*) ' *** NCHA = ', NCHA
1352:             write(NOUT1,7520) (NOL(I),I=1,NTNUC)
1353:         end if
1354: C
1355:         NCHA = NCHA + 10
1356: C
1357: 7520      format(' ', ' ## NOL ## ',20I5)
1358: C
1359: C *** READ CASBRN+'BNUP' MEMBER FOR BRANCH-OFF CASE
1360: C
1361:         if (IVOID.eq.1) then
1362:             write(NOUT1,*) ' ---- CHECKING MEMBERS ',CASBRN,
1363: & 'HT## FOR BRANCH-OFF CALCULATION'
1364: C
1365:             call BURNVD(NEPVD, IBSTEP, DIRIN, RARRAY, IARRAY, MEMORY,

```

```

1366: & WTHVY0,NTNUC0, NOUT1 )
1367: C
1368:         IBNTYP = IBC(9)
1369:         end if
1370: C
1371:         call RWCOM1( 'WRITE', DIROUT, CASEID )
1372:         call RWCOM2( 'WRITE', DIROUT, CASEID )
1373:         call RWCOM3( 'WRITE', DIROUT, CASEID )
1374:         call RWREST( 'WRITE', DIROUT, CASEID )
1375: C
1376:         ... write caseMVPI as an virtual member
1377: C
1378:         MEMBER = 'OPENING'
1379:         call PDSIO( 'OPEN', DIROUT, MEMBER, MEMBER, RARRAY, 3, NOUT1 )
1380:         MEMBER = CASEID// 'MVPI'
1381:         IMVPI = 61
1382:         call PDSFIL( 'WRITE', DIROUT, MEMBER, 'TEXT', IMVPI, NOUT1, IIERR
1383: & )
1384: C
1385:         if ( IIERR.ne.0 ) then
1386:             write(NOUT1,6888)
1387:             write(NOUT1,*) ' failed to open member <',MEMBER,'>'
1388:             stop 999
1389:         end if
1390: C
1391:         LIN = 92
1392:         call RWIND( LIN )
1393:         do 650 I = 1, NCARD
1394:             read(LIN,'(a)') MLINE
1395:             write(IMVPI,'(A)') MLINE(:ICLEN2(MLINE))
1396: 650      continue
1397: C
1398:         call PDSFIL( 'CLOSE', DIROUT, MEMBER, 'TEXT', IMVPI, NOUT1, IIERR
1399: & )
1400: C
1401:         call RWCHAN( 'WRITE', DIROUT, CASEID, NPAR, NTNUC, NTFISS,
1402: & NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
1403: C
1404:         call RWMATD( 'WRITE', DIROUT, CASEID, NMAT, KNMAX, NISO(1),
1405: & TEMP(1), VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1),
1406: & MTCARD(1,1), IDENT(1,1), DNINIT(1,1), MATMVP(1),
1407: & IPBURN(1,1),MATREG(1) )
1408: C
1409: C
1410: C .... make "HIST" member ....
1411: C
1412:         if ( IVOID.eq.0 ) then
1413:             MEMBER = CASEID// 'HT'//STEPNM(1,0)
1414: C
1415: C ... Contents of CASE+'HT'+step are packed by "PACKET" series
1416: C routines.
1417: C
1418:             call PACK00( IARRAY, MEMBER, MEMORY, IRET )
1419: C
1420: C .. pack STEP#, NMAT & NTNUC
1421:             call PACKND( IARRAY, 'STEP#', 'I4', NOWSTP, 1, IRET )
1422:             call PACKND( IARRAY, 'NMAT', 'I4', NMAT, 1, IRET )
1423:             call PACKND( IARRAY, 'NTNUC', 'I4', NTNUC, 1, IRET )
1424: C
1425: C ... WTHVY0 is packed as HMINV here
1426:             call PACKLB( IARRAY, 'HMINV', 'HMINV', IRET )
1427:             call PACKND( IARRAY, 'HMINV', 'R4', WTHVY0, NMAT, IRET1 )
1428:             if ( IRET1.ne.0 ) go to 720
1429: C
1430: C

```

src/mvpburn/burnip.f

```

1431:      call PACKLB( IARRAY, 'EXPSZN', 'EXPSZN', IRET )
1432:      call PACKNN( IARRAY, 'EXPSZN', 'R4', 0.0E0, 1, NMAT, IRET1 )
1433:      if ( IRET1.ne.0 ) go to 720
1434:
1435:      call PACKLB( IARRAY, 'DMWZON', 'DMWZON', IRET )
1436:      call PACKNN( IARRAY, 'DMWZON', 'R4', 0.0E0, 1, NMAT, IRET1 )
1437:      if ( IRET1.ne.0 ) go to 720
1438:
1439: C
1440: C ... first dimension of DNBURN is NTNUC0 and it may not be the
1441: C same as NTNUC
1442: C
1443:      call PACKLB( IARRAY, 'DNBURN', 'DNBURN', IRET )
1444:      call PACKPT( IARRAY, 'DNBURN', 'R4', LP, NTNUC*NMAT, IRET1 )
1445:      if ( IRET1.ne.0 ) go to 720
1446: C
1447:      do 670 M = 1, NMAT
1448:      do 660 I = 1, NTNUC
1449:      RARRAY(LP) = DNBURN(I,M)
1450:      LP = LP + 1
1451:      660      continue
1452:      670      continue
1453: C
1454:      call PACKLB( IARRAY, MEMBER, 'INTEGER', IRET )
1455:      NPK = 10
1456:      call PACKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
1457:      if ( IRET1.ne.0 ) go to 720
1458: C
1459:      call ICLEA( IARRAY(LP), NPK, 0 )
1460: C
1461:      call PACKLB( IARRAY, MEMBER, 'REAL', IRET )
1462:      NPK = 30
1463: C
1464:      call PACKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
1465:      if ( IRET1.ne.0 ) go to 720
1466: C
1467:      call CLEA( RARRAY(LP), NPK, 0.0 )
1468: C
1469:      RARRAY(LP) = AKEFF(1)
1470:      RARRAY(LP+1) = ERRKEF(1)
1471:      RARRAY(LP+2) = DAYS(1)
1472:      RARRAY(LP+3) = U235F(1)
1473:      RARRAY(LP+4) = EXPST(1)
1474:      RARRAY(LP+5) = CUMMWD(1)
1475:      RARRAY(LP+6) = INSCR(1)
1476:      RARRAY(LP+7) = INTCR(1)
1477:      RARRAY(LP+8) = EINSR(1)
1478:      RARRAY(LP+9) = EINTCR(1)
1479:      RARRAY(LP+10) = FLXNR(1)
1480:      RARRAY(LP+11) = FACNR(1)
1481:      RARRAY(LP+12) = FISABS(1)
1482:      RARRAY(LP+13) = FRTCAP(1)
1483:      RARRAY(LP+14) = EFISAB(1)
1484:      RARRAY(LP+15) = EFRTCA(1)
1485:      RARRAY(LP+16) = FDECAY(1)
1486:      RARRAY(LP+17) = CDECAY(1)
1487:      RARRAY(LP+18) = ERRNR(1)
1488:      RARRAY(LP+19) = POWERW(1)
1489:      RARRAY(LP+20) = POWERE(1)
1490:      RARRAY(LP+21) = RMONIT(1)
1491:      RARRAY(LP+22) = EMONIT(1)
1492:      call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
1493: C
1494:      call PDSIO( 'WRITE', DIROUT, MEMBER, MEMBER, IARRAY, LENG,
1495:      &      NOUT1 )

```

```

1496:      end if
1497: C
1498: C -----
1499: C
1500: C *** PRINT OUT OF INPUT DATA FOR BURNUP
1501: C
1502: C -----
1503: C
1504:      write(NOUT1,7540) CASEID, TITLE
1505:      write(NOUT1,7560) (IBC(I),I=1,7)
1506:      write(NOUT1,7562) (IBC(I),I=8,8), BZERO
1507:      write(NOUT1,7565) (IBC(I),I=9,11)
1508:      call ZZMMM(1,1,STDNUC,CWRK(1:6))
1509:      write(NOUT1,7580) CWRK(1:6)
1510:      call ZZMMM(1,1,NCXE35,CWRK1(1:6))
1511:      call ZZMMM(1,1,NCI135,CWRK2(1:6))
1512:      call ZZMMM(1,1,NCSM49,CWRK3(1:6))
1513:      call ZZMMM(1,1,NCPM49,CWRK4(1:6))
1514:      write(NOUT1,7646) CWRK1(1:6),CWRK2(1:6),CWRK3(1:6),CWRK4(1:6)
1515: C
1516:      if(IBNTYP.eq.2.or.IBNTYP.eq.3) then
1517:      write(NOUT1,7647) IDMONI
1518:      end if
1519:      if ( IVOID.eq.1 ) then
1520:      write(NOUT1,7600) CASBRN
1521:      end if
1522: C
1523:      if( IBNTYP.eq.0 ) then
1524:      write(NOUT1,7640) (POWERL(I),I=1,NEP)
1525:      if(METHPC.eq.1) write(NOUT1,7641) (JPCFLG(I),I=1,NEP)
1526:      if(IVOID.eq.1) write(NOUT1,7648) (IBSTEP(I),I=1,NEP)
1527: C
1528:      else if ( IBNTYP.eq.1 ) then
1529:      write(NOUT1,7642) (EXTSRC(I),I=1,NEP)
1530:      if(METHPC.eq.1) write(NOUT1,7641) (JPCFLG(I),I=1,NEP)
1531:      if(IVOID.eq.1) write(NOUT1,7648) (IBSTEP(I),I=1,NEP)
1532: C
1533:      else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
1534:      write(NOUT1,7643) (VMONIT(I),I=1,NEP)
1535:      if(METHPC.eq.1) write(NOUT1,7641) (JPCFLG(I),I=1,NEP)
1536:      if(IVOID.eq.1) write(NOUT1,7648) (IBSTEP(I),I=1,NEP)
1537: C
1538:      else if ( IBNTYP.eq.4 ) then
1539:      write(NOUT1,7644) (REGPOW(I),I=1,NEP)
1540:      if(METHPC.eq.1) write(NOUT1,7641) (JPCFLG(I),I=1,NEP)
1541:      if(IVOID.eq.1) write(NOUT1,7648) (IBSTEP(I),I=1,NEP)
1542:      if(IVOID.eq.0) write(NOUT1,7645) (TRGPOW(I),I=1,MATPOW)
1543:      if(IVOID.eq.0) write(NOUT1,7649) (MATREG(I),I=1,NMAT)
1544:      end if
1545: C
1546:      write(NOUT1,7660) (PERIOD(I),I=1,NEP)
1547: C
1548:      write(NOUT1,7680) NFIS, NFER
1549:      ILOOP = NFIS
1550:      if ( NFER.gt.NFIS ) ILOOP = NFER
1551:      do 680 I = 1, ILOOP
1552:      if ( I.le.NFIS.and.I.le.NFER ) then
1553:      call NAMRST(NAMFIS(I),CWRK)
1554:      call NAMRST(NAMFER(I),CWRK2)
1555:      write(NOUT1,7700) CWRK, IFISDN(I), FISFCT(I),
1556:      &      CWRK2, IFRTDN(I), FRTFCT(I)
1557:      else
1558:      if ( NFIS.eq.ILOOP ) then
1559:      call NAMRST(NAMFIS(I),CWRK)
1560:      write(NOUT1,7700) CWRK, IFISDN(I), FISFCT(I)

```

src/mvpburn/burnip.f

```

1561:         else
1562:             call NAMRST(NAMFER(I),CWRK2)
1563:             write(NOUT1,7720) CWRK2, IFRDNI(I), FRTFCT(I)
1564:         end if
1565:     end if
1566:     680 continue
1567: C
1568:         write(NOUT1,7690)
1569: C
1570: 7540 format(/1X,15X,' ## BURNUP INPUT DATA LISTING ## '//1X,
1571: & 'CASEID:',A4/1X,'TITLE:',A72/1X,
1572: & ':',A72)
1573: 7560 format(/1X,5X,
1574: & ' NO OF BURNUP STEPS (NEP) ..... ',I3/
1575: & 1X,5X,
1576: & ' UNIT OF BURNUP ..... ',I3/
1577: & 1X,5X,' (1:2:3:4:5=MWD/T:MWD:DAYS:DELTA DAYS:% STDNUC)'/
1578: & 1X,5X,
1579: & ' PRINTING OPTION ..... ',I3/
1580: & 1X,5X,' (0:1:2 =BRIEF:DETATIL:DEBUG DETAIL) ' /
1581: & 1X,5X,
1582: & ' BURNUP CALCULATION MODE ..... ',I3/
1583: & 1X,5X,' (0:1=NORMAL/BANCH-OFF) ' /1X,5X,
1584: & ' DEFINITION OF STANDARD NUCLIDE ..... ',I3/
1585: & 1X,5X,' (0:1 =DEFAULT:DEFINED BY USER-INPUT) ' /
1586: & 1X,5X,
1587: & ' DEFINITION OF CONVERSION RATIO ..... ',I3/
1588: & 1X,5X,' (0:1 =DEFAULT:DEFINED BY USER-INPUT) ' /
1589: & 1X,5X,
1590: & ' MAXIMUM LENGTH OF LINEAR CHAIN (DEFAULT IS 6) ... ',I3)
1591: 7562 format( 1X,5X,
1592: & ' BURNUP POWER & PERIOD & NEP MODIFICATION OPTION... ',I3/
1593: & 1X,5X,' (0:1 =NO:MODIFY) ' /
1594: & 1X,5X,
1595: & ' NUMBER DESITY SETTING FOR INITIAL ZERO DENSITY... ',
1596: & 1P,E9.2 / 1X,5X,' FOR MVP-INPUT')
1597: 7565 format( 1X,5X,
1598: & ' BURNUP MODE IN EACH BURNUP STEP ..... ',I3/
1599: & 1X,5X,' =0:CORE POWER CONSTANT (DEFAULT) ' /
1600: & 1X,5X,' =1:EXTERNAL NEUTRON SOURCE CONSTANT (ADS MODE)'/1X,5X,
1601: & ' =2:MONITORING REACTION RATE CONSTANT (FIXED SOURCE MODE)'/
1602: & 1X,5X,' =3:MONITORING REACTION RATE CONSTANT (EIGEN-VALUE MODE)'/
1603: & 1X,5X,' =4:MONITORING REGION POWER CONSTANT ' /
1604: & 1X,5X,
1605: & ' RENORMALIZATION OPTION in PC METHOD BURNUP ..... ',I3/
1606: & 1X,5X,' =0:NOT REQUESTED ' /
1607: & 1X,5X,' =1:REQUESTED (ONLY ALLOWED for IBC9=2,3) ' /
1608: & 1X,5X,
1609: & ' DEFINITION OF FP YIEDLD EDIT ..... ',I3/
1610: & 1X,5X,' (0:1 =DEFAULT:DEFINED BY USER-INPUT) ' /)
1611: 7580 format(1X,5X,
1612: & ' STANDARD NUCLIDE NAME (STDNUC) ..... ',A)
1613: 7600 format(1X,5X,
1614: & ' REFERENCE CASE FOR BANCH-OFF..... ',A4)
1615: 7640 format(/1X,5X,
1616: & ' POWER FOR EACH BURNUP STEP (MWT) ..... ',
1617: & 1P,5E12.5:/(57X,1P,5E12.5:))
1618: 7641 format(/1X,5X,
1619: & ' PC METHOD APPLIED FLAG (0/1=NORMAL/PC) ..... ',
1620: & 5I12 :/(57X,5I12:))
1621: 7642 format(/1X,5X,
1622: & ' EXTERNAL NEUTRON SOURCE LEVEL (N/SEC) ..... ',
1623: & 1P,5E12.5:/(57X,1P,5E12.5:))
1624: 7643 format(/1X,5X,
1625: & ' MONITORING REACTION RATE ..... ',

```

```

1626: & 1P,5E12.5:/(57X,1P,5E12.5:))
1627: 7644 format(/1X,5X,
1628: & ' MONITORING REGION POWER FOR EACH BURNUP STEP(MWT) ',
1629: & 1P,5E12.5:/(57X,1P,5E12.5:))
1630: 7645 format(/1X,5X,
1631: & ' TALLY REGION NAME OF MONITORING POWER MATERIAL .. ',
1632: & 1P,5A12:/(57X,1P,5A12:))
1633: 7646 format(1X,5X,
1634: & ' NUCLIDE NAME of FP YIELD EDIT ..... ',
1635: & 4(A6,2X) )
1636: 7647 format(1X,5X,
1637: & ' SPECAIL TALLY ID NUMBER (BNPTYP=2,3) ..... ',
1638: & I12 )
1639: 7648 format(1X,5X,
1640: & ' BARANCHING-OFF CAL. FLAG (0/1/2:NO/YES/DONE) .... ',
1641: & 5I12:/(57X,5I12:))
1642: 7649 format(1X,5X,
1643: & ' MONITORING FLAG FOR EACH MATERIAL (0/1:NO/YES) .. ',
1644: & 5I12:/(57X,5I12:))
1645: C
1646: C
1647: 7660 format(/1X,5X,
1648: & ' PERIOD IN EACH BURNUP STEP (SPECIFIED UNIT) ..... ',
1649: & 1P,5E12.5:/(57X,1P,5E12.5:))
1650: C
1651: 7680 format(/19X,'<< DEFINITION OF CONVERSION ',
1652: & 'RATIO >> '//26X,
1653: & 'NUMBER OF FISSILE =',I3/26X,'NUMBER OF FERTILE =',
1654: & I3//4X,' FISSILE DENSITY MULTIPLICATION',
1655: & ' FERTILE DENSITY MULTIPLICATION'/4X,
1656: & ' NAME FLAG FACTOR ',
1657: & ' NAME FLAG FACTOR '/4X,
1658: & '-----',
1659: & '-----')
1660: 7690 format(4X,
1661: & '-----',
1662: & '-----')
1663: C
1664: 7700 format(1X,4X,A8,6X,I2,6X,1P,E12.5,1X,A8,6X,I2,6X,1P,E12.5)
1665: 7720 format(1X,38X,A8,6X,I2,6X,1P,E12.5)
1666: C
1667: C *** PRINT MATERIAL DATA
1668: C
1669: if (MVPDBG.ne.0) then
1670:     write(NOUT1,7740) NMAT, NCARD
1671: C
1672: do 700 M = 1, NMAT
1673: C
1674:     write(NOUT1,7760) M, TRGNAM(M), VOLM(M), TEMP(M), NISO(M),
1675: & MATMVP(M), MTBURN(3,M), MTBURN(1,M), MTBURN(2,M),
1676: & MTCARD(1,M), MTCARD(2,M)
1677: C
1678:     write(NOUT1,7780)
1679: C
1680:     do 690 I = 1, NISO(M)
1681:         write(NOUT1,7820) I, IDENT(I,M), DNINIT(I,M), IPBURN(I,M)
1682: 690 continue
1683: C
1684:     write(NOUT1,7800)
1685: C
1686: 700 continue
1687: end if
1688: C
1689: C----- skip printing of original MVP input data if IBC(3)<2
1690: C

```

src/mvpburn/burnip.f

```
1691:      if (IBC(3).LT.2) goto 9999
1692: C
1693:      write(NOUT1,7840)
1694: C
1695:      LIN      = 92
1696:      call RWIND( LIN )
1697:      do 710 I = 1, NCARD
1698:          read(LIN,'(A)') MLINE
1699:          write(NOUT1,7860) I, MLINE(:ICLEN2(MLINE))
1700:      710 continue
1701: C
1702:      write(NOUT1,7880)
1703: C
1704: 7740 format('//1X,15X,'<< MATERIAL INFORMATION LIST AT BURNIN STEP >>'//
1705:      &      '/1X,10X,' NMAT : NO OF BURNUP MATERIALS ..... ',I5/
1706:      &      '/1X,10X,' NCARD : NO OF ORIGINAL MVP INPUT DATA .. ',I5//)
1707: 7760 format(1X,10X,I2,'--TH MATERIAL INFORMATION LISTING : '//1X,5X,
1708:      &      'TALLY REGION NAME ..... ',A12/1X,5X,
1709:      &      'VOLUME (CC) ..... ',1P,E12.5/1X,5X,
1710:      &      'TEMPERATURE IN KELVIN..... ',0P,F12.4/1X,5X,
1711:      &      'NO OF INITIAL NUCLIDE ..... ',I12/1X,5X,
1712:      &      'TALLY RESION NO IN MVP-GEOM... ',I12/1X,5X,
1713:      &      'MATERIAL TYPE..... ',I12,1X,
1714:      &      '(1/2/3:FUEL/GD/ABSORBER)'/1X,5X,
1715:      &      'FIRST NUCLIDE POS. IN CHAIN ... ',I12/1X,5X,
1716:      &      'LAST NUCLIDE POS. IN CHAIN ... ',I12/1X,5X,
1717:      &      'FISRT CARD POS. IN MVP-DATA ... ',I12/1X,5X,
1718:      &      'LAST CARD POS. IN MVP-DATA ... ',I12//)
1719: 7780 format(1X,15X,'== NUCLIDE INFORMATION TABLE == '//1X,5X,
1720:      &      ' NO NUCLIDE DENSITY POSITION IN CHAIN '/1X,5X,
1721:      &      '-----')
1722: 7800 format(1X,5X,'-----')
1723: 7820 format(1X,5X,I4,2X,A8,1P,E12.5,1X,0P,I4)
1724: 7840 format(1X,15X,'== ORIGINAL MVP INPUT DATA LISTING == '//1X,5X,
1725:      &      ' NO : MVP INPUT DATA ..... '/1X,5X,
1726:      &      '-----',18('----'))
1727: 7860 format(1X,5X,I7,' : ',A)
1728: 7880 format('/1X,15X,'== END OF ORIGINAL MVP INPUT DATA LISTING == ')
1729: C
1730: 6888 format('// XXX(BURNIP) ERROR-STOP : ' )
1731: C
1732: C *** END OF PROCESS
1733: C
1734: 9999 continue
1735:      return
1736: C
1737: 720 write(NOUT1,6888)
1738:      write(NOUT1,*) ' error in making data packet as ',
1739:      &      ' contents of member <', MEMBER, '> code =', IRET
1740: C
1741:      stop 999
1742:      end
```

src/mvpburn/burnpr.f

```

1: C2003 added DENERR,JDENERR
2: subroutine BURNPR( NOUT1, NOUT2, CASEID, MTYP,
3: & IMVPIN,TEMP, VOLM, TRGNAM,MTBURN,POWRZN,
4: & EXPSZN,HMINV, DMWZON,GAMAVG,YLDXE5,YLDI35,
5: & YLDSM9,YLDPM9,DENTAB,DENERR,JDENERR,
6: & RATMIC,NOTSTP,IOTSTP)
7: C=====
8: C
9: include 'INC/_BURNP'
10: C
11: include 'INC/_CBURNC1'
12: include 'INC/_CBURNC2'
13: include 'INC/_CBURNC3'
14: C
15: real INSCR, INTCR
16: common /ENREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
17: & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
18: & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
19: & INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
20: & FKNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
21: & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
22: & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
23: & NBATCH(MXSTEP)
24: C2005
25: & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
26: & RMONIT(MXSTEP), EMONIT(MXSTEP)
27: Cend
28: C
29: character*4 CASEID
30: C
31: character*12 TRGNAM(NMAT)
32: C
33: integer MTYP(NMAT), MTBURN(3,NMAT)
34: C
35: real TEMP(NMAT), VOLM(NMAT)
36: C
37: real POWRZN(NMAT,NEP1)
38: real EXPSZN(NMAT,NEP1)
39: real HMINV(NMAT,NEP1)
40: real DMWZON(NMAT,NEP1)
41: real YLDXE5(NMAT,NEP1)
42: real YLDI35(NMAT,NEP1)
43: real YLDSM9(NMAT,NEP1)
44: real YLDPM9(NMAT,NEP1)
45: real GAMAVG(NMAT,NEP1)
46: C
47: real DENTAB(NTNUC,NMAT,NEP1)
48: C2003
49: real DENERR(NTNUC,NMAT,NEP1)
50: real RATMIC(2,4,NTNUC,NMAT,NEP1)
51: integer IOTSTP(NOTSTP)
52: C
53: C **** LOCAL ARRAY
54: C
55: real TABWTH(MXSTEP), ERRMIC(MXSTEP)
56: C
57: CKSK character*8 IDMTYP(3), IDREAC(3)
58: character*8 IDMTYP(4), IDREAC(3)
59: character*72 MLINE
60: C2003
61: character*6 CWRK
62: C
63: CKSK data IDMTYP /'FUEL', 'GD ABS.', 'ABSORBER'/
64: data IDMTYP /'FUEL', 'FP-ABS', 'BP-ABS', 'FP&BP' /
65: data IDREAC /'FISSION', 'CAPTURE', '(N,2N) /

```

```

66: C
67: C *** START OF PROCESS ****
68: C
69: call CLEA( TABWTH, MXSTEP, 0.0 )
70: C
71: do 110 I = 1, NOWSTP
72: do 100 M = 1, NMAT
73: TABWTH(I) = TABWTH(I) + VOLM(M)*HMINV(M,I)
74: 100 continue
75: 110 continue
76: C
77: C *** PRINT OUT PROCESS ****
78: C
79: C 120 continue
80: C2005 ... added
81: IBNTYP = IBC(9)
82: C2005 ... end
83: C
84: write(NOUT1,7000)
85: write(NOUT1,7010) CASEID, TITLE
86: write(NOUT1,7020)
87: write(NOUT1,7021)
88: write(NOUT1,7022)
89: write(NOUT1,7023)
90: write(NOUT1,7024)
91: write(NOUT1,7025)
92: write(NOUT1,*)
93: write(NOUT1,*)
94: if( NOTSTP.le.0 ) then
95: write(NOUT1,7030) (I,I=1,NOWSTP)
96: NOWST1 = NOWSTP
97: else
98: write(NOUT1,7030) (IOTSTP(I),I=1,NOWSTP)
99: NOWST1 = NOWSTP+1
100: end if
101: write(NOUT1,7040) (DAYS(I),I=1,NOWSTP)
102: write(NOUT1,7060) (EXPST(I),I=1,NOWSTP)
103: C2003
104: C write(NOUT1,7080) STDNUC(2:4), (U235F(I),I=1,NOWSTP)
105: call ZZMMM(1,1,STDNUC,CWRK(1:6))
106: do 90 II=6,1,-1
107: if (CWRK(II:II).ne.' ') go to 95
108: 90 continue
109: 95 continue
110: write(NOUT1,7080) CWRK(1:II), (U235F(I),I=1,NOWSTP)
111: C2003 ... end
112: C
113: write(NOUT1,7100) (AKEFF(I),I=1,NOWST1-1)
114: write(NOUT1,7120) (ERRKEF(I),I=1,NOWST1-1)
115: write(NOUT1,7140) (NHIST(I),I=1,NOWST1-1)
116: write(NOUT1,7160) (NBATCH(I),I=1,NOWST1-1)
117: C
118: write(NOUT1,7180) (INSCR(I),I=1,NOWST1-1)
119: write(NOUT1,7200) (EINSR(I),I=1,NOWST1-1)
120: write(NOUT1,7220) (INTCR(I),I=1,NOWST1-1)
121: write(NOUT1,7240) (CUMMWD(I),I=1,NOWSTP)
122: C2005 ... modified
123: Cmod write(NOUT1,7260) (POWERL(I),I=1,NOWST1-1)
124: write(NOUT1,7260) (POWERW(I),I=1,NOWST1-1)
125: C2005 ... end
126: write(NOUT1,7280) (TABWTH(I),I=1,NOWSTP)
127: C2005 ... added
128: if ( IBNTYP.eq.1 ) then
129: write(NOUT1,7510) (EXTSRC(I),I=1,NOWST1-1)
130: endif

```


src/mvpburn/burnpr.f

```

131: C
132:   if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
133:     write(NOUT1,7520) (VMONIT(I),I=1,NOWST1-1)
134:     write(NOUT1,7521) (RMONIT(I),I=1,NOWST1-1)
135:     write(NOUT1,7522) (EMONIT(I),I=1,NOWST1-1)
136:   endif
137: C
138:   if ( IBNTYP.eq.4 ) then
139:     write(NOUT1,7530) (REGPOW(I),I=1,NOWST1-1)
140:   endif
141: C2005 ... ended
142:   write(NOUT1,7300) (FLXNRM(I),I=1,NOWST1-1)
143: C2005 ... added
144:   write(NOUT1,7310) (ERRNRM(I),I=1,NOWST1-1)
145: C2005 ... end
146:   write(NOUT1,7320) (FISABS(I),I=1,NOWST1-1)
147:   write(NOUT1,7340) (FDECAY(I),I=1,NOWST1-1)
148:   write(NOUT1,7360) (FRTCAP(I),I=1,NOWST1-1)
149:   write(NOUT1,7380) (CDECAY(I),I=1,NOWST1-1)
150:   write(NOUT1,7400) (FACNRM(I),I=1,NOWST1-1)
151: C
152: C=====
153: 7000 format(//1X,47('=')/1X,'RESULT OF DEPLETION (OR BRANCH-OFF) ',
154:   & 'CALCULATION'/1X,47('='))
155: 7010 format(1X,'CASEID:',A4/1X,'TITLE :',A72/1X,
156:   & ',A72)
157: C----- FOR NOTE
158: 7020 format(//1X,37('-'),' NOTE ',37('-'))
159: 7021 format(' U0235-% : DEPLETED FRACTION ON NUMBER DENSITY OF '
160:   & 'U-235(OR DEFINED NUCLIDE) ')
161: 7022 format(' INST.-C.R. : INSTANTANEOUS CONVERSION RATIO(OR USER',
162:   & ' DEFINED REACT.RATE RATIO)'/
163:   & 14X,'= (FER.-CAPTR + PRE.-DECAY)/(FIS.-ABSPT + FIS.-DECAY)'/
164:   & ' INTE.-C.R. : TIME-INTEGRATED CONVERSION RATIO',
165:   & '(OR USER DEFINED REACT.RATE RATIO)'/
166:   & 14X,'= INTEG[(FER.-CAPTR + PRE.-DECAY)]/INTEG[(FIS.-ABSPT + ',
167:   & 'FIS.-DECAY)]' )
168: 7023 format(' FIS.-ABSPT : ABSORPTION RATE OF FISSILE NUCLIDES FOR ',
169:   & 'CONVERSION RATIO CAL.'/
170:   & ' FIS.-DECAY : DECAY RATE OF FISSILE NUCLIDES FOR ',
171:   & 'CONVERSION RATIO CAL.'/
172:   & ' FER.-CAPTR : CAPTURE RATE OF FERTILE NUCLIDES FOR ',
173:   & 'CONVERSION RATIO CAL.'/
174:   & ' PRE.-DECAY : DECAY RATE OF PRECURSOR NUCLIDES FOR ',
175:   & 'CONVERSION RATIO CAL.' )
176: C2005 *** modified
177: C7024 format(' TON-HM : INVENTORY(TON) OF HEAVY NUCLIDES'/
178:   & ' R-NORM-FAC : POWER NORMALIZATION FACTOR FOR REACTION ',
179:   & 'RATES FROM MVP'/
180:   & 14X,'= REACTION RATE(MVP-BURN) / REACTION RATE(MVP)'/
181:   & ' A-NORM-FAC : TIME AVERAGED R-NORM-FAC IN EACH TIME ',
182:   & 'STEP INTERVAL')
183: C
184: 7024 format(' TON-HM : INVENTORY(TON) OF HEAVY NUCLIDES'/
185:   & ' EXTSRC(n/s) : EXTERNAL NEUTRON SOURCE LEVEL (BNPTYP=1) '/
186:   & ' INPUT-MONIT: INPUT MONITORING REACTION RATE(BNPTYP=2/3)'/
187:   & ' MVP-MONITOR: MVP MONITORING REACTION RATE (BNPTYP=2/3)'/
188:   & ' ERR-MONI(%): MVP MONITORING R.R. ERROR (BNPTYP=2/3)'/
189:   & ' REGPOW(MW) : MONITORING REGION POWER (BNPTYP=4) '/
190:   & ' R-NORM-FAC : POWER NORMALIZATION FACTOR FOR REACTION ',
191:   & 'RATES FROM MVP'/
192:   & 14X,'= REACTION RATE(MVP-BURN) / REACTION RATE(MVP)'/
193:   & ' ERR-NRM(%): POWER NORMALIZATION FACTOR ERROR '/
194: C & ' A-NORM-FAC : TIME AVERAGED R-NORM-FAC IN EACH TIME ',
195: C & 'STEP INTERVAL')

```

```

196:   & ' A-NORM-FAC : TIME AVERAGED CHANGE OF R-NORM-FAC IN ',
197:   & 'EACH TIME STEP PERIOD')
198: C2005 ... end
199: 7025 format(1X,80('-'))
200: C=====
201: 7030 format(' * STEP ',I7,255I12:/(7X,255I12:))
202: 7040 format(' * DAYS ',1P,255E12.5:/(12X,1P,255E12.5:))
203: 7060 format(' * MWD/TON ',1P,255E12.5:/(12X,1P,255E12.5:))
204: C2003
205: C7080 format(' ',A3,'-% ',1P,255E12.5:/(12X,1P,255E12.5:))
206: 7080 format(' * ',A,'-% ',T15,1P,255E12.5:/(12X,1P,255E12.5:))
207: C2003 ... end
208: 7100 format('/ * K-EFF ',F9.6,254F12.6:/(9X,0P,255F12.6:))
209: 7120 format(' * ERROR(%) ',F9.6,254F12.6:/(9X,0P,255F12.6:))
210: 7140 format(' * HISTORY ',I9,254I12:/(9X,255I12:))
211: 7160 format(' * NUM-BATCH ',I9,254I12:/(9X,255I12:))
212: 7180 format('/ * INST.-C.R. ',F9.6,254F12.6:/(9X,0P,255F12.6:))
213: 7200 format(' * ERROR(%) ',F9.6,254F12.6:/(9X,0P,255F12.6:))
214: 7220 format(' * INTE.-C.R. ',F9.6,254F12.6:/(9X,0P,255F12.6:))
215: 7240 format(' * MWD ',1P,255E12.5:/(12X,1P,255E12.5:))
216: 7260 format(' * POWER(MW) ',1P,255E12.5:/(12X,1P,255E12.5:))
217: 7280 format(' * TON-HM ',1P,255E12.5:/(12X,1P,255E12.5:))
218: 7300 format(' * R-NORM-FAC ',1P,255E12.5:/(12X,1P,255E12.5:))
219: C2005 ... added
220: 7510 format(' * EXTSRC(n/s)',1P,255E12.5:/(12X,1P,255E12.5:))
221: 7520 format(' * INPUT-MONIT',1P,255E12.5:/(12X,1P,255E12.5:))
222: 7521 format(' * MVP-MONITOR',1P,255E12.5:/(12X,1P,255E12.5:))
223: 7522 format(' * ERR-MONI(%)',F9.6,254F12.6:/(9X,0P,255F12.6:))
224: 7530 format(' * REGPOW(MW) ',1P,255E12.5:/(12X,1P,255E12.5:))
225: C
226: 7310 format(' * ERR-NRM(%) ',F9.6,254F12.6:/(9X,0P,255F12.6:))
227: C2005 ... end
228: 7320 format(' * FIS.-ABSOR ',1P,255E12.5:/(12X,1P,255E12.5:))
229: 7340 format(' * FIS.-DECAY ',1P,255E12.5:/(12X,1P,255E12.5:))
230: 7360 format(' * FER.-CAPTR ',1P,255E12.5:/(12X,1P,255E12.5:))
231: 7380 format(' * PRE.-DECAY ',1P,255E12.5:/(12X,1P,255E12.5:))
232: 7400 format(' * A-NORM-FAC ',1P,255E12.5:/(12X,1P,255E12.5:))
233: C
234: CKSK Skip Printing of number densities when IBEDIT<0
235: C
236:   if (IBEDIT.LT. 0) goto 9999
237: C
238: C *** MATERIAL-WIZE PRINT OUT
239: C
240:   do 170 M = 1, NMAT
241:     VOL = VOLM(M)
242:     TMP = TEMP(M)
243:     write(NOUT1,7410) M, VOL, HMINV(M,1), IDMTYP(MTYP(M)),
244:     & TRGNAM(M), TMP
245:     write(NOUT1,*)
246:     if( NOTSTP.le.0 ) then
247:       write(NOUT1,7420) (I,I=1,NOWSTP)
248:     else
249:       write(NOUT1,7420) (IOTSTP(I),I=1,NOWSTP)
250:     endif
251:     write(NOUT1,7430) (DAYS(I),I=1,NOWSTP)
252:     write(NOUT1,7440) (EXPSZN(M,I),I=1,NOWSTP)
253:     write(NOUT1,7450) (POWRZN(M,I),I=1,NOWST1-1)
254:     write(NOUT1,7540) (GAMAVG(M,I),I=1,NOWST1-1)
255: C
256: C2005 ... modified
257: C
258: Cm write(NOUT1,7560) (YLDXE5(M,I),I=1,NOWST1-1)
259: Cm write(NOUT1,7580) (YLDI35(M,I),I=1,NOWST1-1)
260: Cm write(NOUT1,7600) (YLDMS9(M,I),I=1,NOWST1-1)

```

```

      write(NOUT1,7565) CWRK(1:6), (YLDI35(M,I), I=1, NOWST1-1)
    endif
  endif

  if( IDI135.gt.0.and.NCI135.gt.0 ) then
    call ZZMMM(1,1,NCI135,CWRK(1:6))
    write(NOUT1,7565) CWRK(1:6), (YLDI35(M,I), I=1, NOWST1-1)
    if(CWRK(6:6).eq.' ') then
      write(NOUT1,7565) CWRK(1:5), (YLDI35(M,I), I=1, NOWST1-1)
    else
      write(NOUT1,7566) CWRK(1:6), (YLDI35(M,I), I=1, NOWST1-1)
    endif
  endif

  endif

  if( IDSM49.gt.0.and.NCSM49.gt.0 ) then
    call ZZMMM(1,1,NCSM49,CWRK(1:6))
    write(NOUT1,7565) CWRK(1:6), (YLDISM9(M,I), I=1, NOWST1-1)
    if(CWRK(6:6).eq.' ') then
      write(NOUT1,7565) CWRK(1:5), (YLDISM9(M,I), I=1, NOWST1-1)
    else
      write(NOUT1,7566) CWRK(1:6), (YLDISM9(M,I), I=1, NOWST1-1)
    endif
  endif

  endif

  if( IDPM49.gt.0.and.NCPM49.gt.0 ) then
    call ZZMMM(1,1,NCPM49,CWRK(1:6))
    write(NOUT1,7565) CWRK(1:6), (YLDPM9(M,I), I=1, NOWST1-1)
    if(CWRK(6:6).eq.' ') then
      write(NOUT1,7565) CWRK(1:5), (YLDPM9(M,I), I=1, NOWST1-1)
    else
      write(NOUT1,7566) CWRK(1:6), (YLDPM9(M,I), I=1, NOWST1-1)
    endif
  endif

  endif

... end

ITEMP = MIN(NOWSTP,255)
write(NOUT1,7500) ( ' -----', I=1, ITEMP)
do 130 N = 1, NTNUC
... use 6 byte nuclide ID
  write(NOUT1,7460) N, SRACID(N), (DENTAB(N,M,I), I=1, NOWSTP)
  call ZZMMM(1,1,NCODE(N),CWRK(1:6))
  write(NOUT1,7460) N, CWRK(1:6), (DENTAB(N,M,I), I=1, NOWSTP)
... end
... print error of density (AVERAGE mode)

```

```

326:      write(NOUT1,7500) ( ' -----',I=1,ITEMP)
327:      do 160 MT = 1, 3
328:          ISTART = MTBURN(1,M)
329:          LAST    = MTBURN(2,M)
330:          if ( MT.eq.1.and.LAST.gt.NTFISS ) LAST = NTFISS
331:          if ( ISTART.gt.LAST ) go to 160
332: C
333:         write(NOUT1,7640) IDREAC(MT)
334: C         write(NOUT1,7660)
335:         ITEMP = MIN(NOWST1-1,255)
336: C2003     write(NOUT1,7660) ( ' -----',I=1,ITEMP)
337:         write(NOUT1,7500) ( ' -----',I=1,ITEMP)
338:         do 150 N = ISTART, LAST
339: C2003 ... use 6 byte nuclide ID
340: C             write(NOUT1,7680) N, SRACID(N),
341: C &                 (RATMIC(1,MT,N,M,I),I=1,NOWST1-1)
342:             call ZZMMM(1,1,NCODE(N),CWRK(1:6))
343:             write(NOUT1,7680) N, CWRK(1:6),
344: C &                 (RATMIC(1,MT,N,M,I),I=1,NOWST1-1)
345: C2003 ... end
346:             call CLEA( ERRMIC, MXSTEP, 0.0 )
347:             do 140 I = 1, NOWSTP
348:                 SAVE = RATMIC(1,MT,N,M,I)
349:                 ESAVE = RATMIC(2,MT,N,M,I)
350:                 if ( SAVE.gt.0.0 ) ERRMIC(I) = 100.0*ESAVE/SAVE
351:             140 continue
352:                 write(NOUT1,7700) (ERRMIC(I),I=1,NOWST1-1)
353:             150 continue
354:                 ITEMP = MIN(NOWST1-1,255)
355:                 write(NOUT1,7660) ( ' -----',I=1,ITEMP)
356:             160 continue
357:             165 continue
358:             170 continue
359: cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
360: C
361: C-- easy readable format with EXCEL
362: C
363:       7410 format(//1X,72('*')/2X,
364: &               'MATERIAL NO.= ',I3,2X,'VOLUME=',1P,E12.5,'(CC)',2X,
365: &               'WEIGHT=',1P,E12.5,'(TON/CC)'/2X,
366: &               'MATERIAL TYPE : ',A8/
367: &               2X,'MATERIAL NAME : ',A12,4X,'TEMPERATURE=',0P,F8.2,
368: &               '(KELVIN) '/1X,72('*')
369:       7420 format(' * STEP ',I7,255I12:/(7X,255I12:))
370:       7430 format(' * DAYS ',1P,255E12.5:/(12X,1P,255E12.5:))
371:       7440 format(' * MWD/TON ',1P,255E12.5:/(12X,1P,255E12.5:))
372:       7450 format(' * POW(MW/CC) ',1P,255E12.5:/(12X,1P,255E12.5:))
373: C2003 ... use 6 byte code for nuclide ID
374: C7460 format(1X,I4,1X,A4,2X,1P,255E12.5:/(21X,1P,255E12.5:))
375:       7460 format(1X,I3,1X,A6,3X,1P,255E12.5:/(21X,1P,255E12.5:))
376: C2003 ... use 6 byte code for nuclide ID
377: C7500 format(2(1X,4('-')),2X,255A12)
378:       7500 format(1X,3('-'),1X,8('-'),1X,255A12)
379: C
380: C7540 format(' * ENERGY/FIS.',1P,255E12.5:/(12X,1P,255E12.5:))
381:       7540 format(' * ENRGY/FIS.',1P,255E12.5:/(12X,1P,255E12.5:))
382:       7560 format(' * XE135-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
383: C2005 added
384:       7565 format(' * ',A5,'-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
385:       7566 format(' * ',A6,'-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
386: Cend
387:       7580 format(' * I0135-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
388:       7600 format(' * SML49-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
389:       7620 format(' * PML49-YLD.',1P,255E12.5:/(12X,1P,255E12.5:))
390: C

```

```

326:      write(NOUT1,7500) ( ' -----',I=1,ITEMP)
327:      do l60 MT = 1, 3
328:          ISTART = MTBURN(1,M)
329:          LAST    = MTBURN(2,M)
330:          if ( MT.eq.1.and.LAST.gt.NTFISS ) LAST = NTFISS
331:          if ( ISTART.gt.LAST ) go to l60
332: C
333:          write(NOUT1,7640) IDREAC(MT)
334: C          write(NOUT1,7660)
335:             ITEMP = MIN(NOWST1-1,255)
336: C2003       write(NOUT1,7660) ( ' -----',I=1,ITEMP)
337:             write(NOUT1,7500) ( ' -----',I=1,ITEMP)
338:             do l50 N = ISTART, LAST
339: C2003 ... use 6 byte nuclide ID
340: C             write(NOUT1,7680) N, SRACID(N),
341: C &                (RATMIC(1,MT,N,M,I),I=1,NOWST1-1)
342:             call ZZMM(1,1,NCODE(N),CWRK(1:6))
343:             write(NOUT1,7680) N, CWRK(1:6),
344: C &                (RATMIC(1,MT,N,M,I),I=1,NOWST1-1)
345: C2003 ... end
346:             call CLEA( ERRMIC, MXSTEP, 0.0 )
347:             do l40 I = 1, NOWSTP
348:                 SAVE = RATMIC(1,MT,N,M,I)
349:                 ESAVE = RATMIC(2,MT,N,M,I)
350:                 if ( SAVE.gt.0.0 ) ERRMIC(I) = 100.0*ESAVE/SAVE
351:             l40 continue
352:                 write(NOUT1,7700) (ERRMIC(I),I=1,NOWST1-1)
353:             l50 continue
354:                 ITEMP = MIN(NOWST1-1,255)
355:             write(NOUT1,7660) ( ' -----',I=1,ITEMP)
356:             l60 continue
357:             l65 continue
358:             l70 continue
359: cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
360: C
361: C-- easy readable format with EXCEL
362: C
363: 7410 format(///1X,72('*')/2X,
364:           &   'MATERIAL NO.=' ,I3,2X,'VOLUME=' ,1P,E12.5,'(CC)' ,2X,
365:           &   'WEIGHT=' ,1P,E12.5,'(TON/CC)' /2X,
366:           &   'MATERIAL TYPE : ',A8/
367:           &   2X,'MATERIAL NAME : ',A12,4X,'TEMPERATURE=' ,0P,F8.2,
368:           &   '(KELVIN )' /1X,72('*'))
369: 7420 format(' * STEP ',I7,255I12:/(7X,255I12:))
370: 7430 format(' * DAYS ',1P,255E12.5:/(12X,1P,255E12.5:))
371: 7440 format(' * MWD/TON ',1P,255E12.5:/(12X,1P,255E12.5:))
372: 7450 format(' * POW(MW/CC) ',1P,255E12.5:/(12X,1P,255E12.5:))
373: C2003 ... use 6 byte code for nuclide ID
374: C7460 format(1X,I4,1X,A4,2X,1P,255E12.5:/(21X,1P,255E12.5:))
375: 7460 format(1X,I3,1X,A6,3X,1P,255E12.5:/(21X,1P,255E12.5:))
376: C2003 ... use 6 byte code for nuclide ID
377: C7500 format(2(1X,4('-')),2X,255A12)
378: 7500 format(1X,3('-'),1X,8('-'),1X,255A12)
379: C
380: C7540 format(' * ENERGY/FIS.' ,1P,255E12.5:/(12X,1P,255E12.5:))
381: 7540 format(' * ENRGY/FIS.' ,1P,255E12.5:/(12X,1P,255E12.5:))
382: 7560 format(' * XE135-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
383: C2005 added
384: 7565 format(' * ',A5,'-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
385: 7566 format(' * ',A6,'-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
386: Cend
387: 7580 format(' * I0135-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
388: 7600 format(' * SMI49-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
389: 7620 format(' * PMI49-YLD.' ,1P,255E12.5:/(12X,1P,255E12.5:))
390: C

```

src/mvpburn/burnpr.f

```
391: 7640 format(/' MICROSCOPIC REACTION RATE : REACTION IS ',A8)
392: C7660 format(2(1X,4('-')),2X,255A12)
393: 7660 format(1X,3('-'),1X,8('-'),1X,255A12)
394: C2003 ... use 6 byte code for nuclide ID
395: C7680 format(1X,I4,1X,A4,2X,1P,255E12.5:/(12X,1P,255E12.5:))
396: 7680 format(1X,I3,1X,A6,3X,1P,255E12.5:/(12X,1P,255E12.5:))
397: C7700 format(12X,255(' ',F9.5,'%'):/ (12X,255(' ',F9.5,'%'):/))
398: C7700 format(12X,255(' ',F9.5,'%'):/ (12X,255(' ',F9.5,'%'):/))
399: 7700 format(5X,'ERR (%)',255(' ',F9.5,' ')/
400: & (12X,255(' ',F9.5,' ')))
401: C=====
402: C
403: C-----Print original MVP input data
404: CKSK if (IBEDIT .GE. 1) then
405:     if (IBEDIT .GE. 2) then
406:         write(NOUT1,7720)
407:         if ( IMVPIN.gt.0 ) then
408:             call RWIND( IMVPIN )
409:             do 180 I = 1, NCARD
410:                 read(IMVPIN,'(A)') MLINE
411:                 write(NOUT1,7740) I, MLINE(:ICLEN2(MLINE))
412:             180 continue
413:             write(NOUT1,7760)
414:         end if
415:     end if
416: C
417: C=====
418: 7720 format(/1X,15X,'== ORIGINAL MVP INPUT DATA LISTING == '//1X,5X,
419: & ' NO : MVP INPUT DATA ' /1X,5X,
420: & '-----',18('----'))
421: 7740 format(1X,5X,I7,' : ',A)
422: 7760 format(/1X,15X,'== END OF ORIGINAL MVP INPUT DATA LISTING == ')
423: C
424: C *** END OF PROCESS ****
425: C
426: 9999 return
427: end
```

src/mvpburn/burnr4.f

```

1:      subroutine BURNR4( NMAT, NTNUC, NOUT2, IBEDIT,VOLM,  PHIP,
2:      &                  POWMAT,DNOLD,  RATMIC,POWERL,POWNRM,EVTOJ,
3:      &                  REACEV,J79,   NOUT1 ,MATREG,POWRGN,ERRNOW )
4: C
5: C **   IBC(9)=4 :
6: C   power normalozation is done to keep POWRGN power
7: C **   SET MATERIAL-WISE POWER   ***
8: C **   SET NORMARIZATION OF MICROSCOPIC REACTION RATE ***
9: C
10:      real VOLM(NMAT)
11:      real PHIP(NMAT)
12:      real DNOLD(NTNUC,NMAT)
13:      real REACEV(2,NTNUC)
14:      real POWMAT(NMAT)
15:      real RATMIC(2,4,NTNUC,NMAT)
16:      integer MATREG(NMAT)
17: C
18: C ***   SET ONE-GROUP MICROSCOPIC X-SECTION FOR BURNCL ROUTINE
19: C   & SET HEAVY METAL WEIGHT
20: C
21: Ctemp
22:      if ( IBEDIT.ge.2 ) then
23:          write(NOUT1,*) ' ** NMAT POWRGN (BURNR4) : ',NMAT,POWRGN
24:          write(NOUT1,*) ' ** MATREG is : '
25:          write(NOUT1,'(10I4)') (MATREG(M),M=1,NMAT)
26:      end if
27: Cend
28:      POWERL = 0.0
29: C
30:      call CLEA( POWMAT, NMAT, 0.0 )
31:      call CLEA( PHIP, NMAT, 1.0 )
32: C
33:      if ( POWRGN.le.0.0 ) then
34:          POWNRM = 0.0
35:          ERRNOW = 0.0
36:          return
37:      end if
38: C
39: C ***   CALCULATE POWER AND SET FACTOR FOR GETTING ABSOLUTE REACTION RATE
40: C
41:      ACOEF = EVTOJ
42:      POWCAL = 0.0
43:      do 110 M = 1, NMAT
44:          if( MATREG(M).gt.0) then
45:              TEMP = 0.0
46:              do 100 I = 1, NTNUC
47:                  SAVE = RATMIC(1,1,I,M)*REACEV(1,I)
48:                  &    + RATMIC(1,2,I,M)*REACEV(2,I)
49:                  TEMP = TEMP + SAVE*DNOLD(I,M)
50:              100 continue
51:              POWMAT(M) = TEMP*ACOEF
52:              POWCAL = POWCAL + POWMAT(M)*VOLM(M)
53:          endif
54:      110 continue
55: C
56: C --- CHECK ZERO POWER CASE
57: C
58:      if ( POWCAL.le.0.0 ) then
59:          write(NOUT1,6888)
60:          write(NOUT1,*) ' zero power was calculated'
61:          write(NOUT1,*) ' Check your input data '
62:          stop 888
63:      end if
64: C
65:      POWNRM = POWRGN/POWCAL
66: C
67:      if ( IBEDIT.gt.2 ) then
68:          write(NOUT1,*) ' POWRGN POWCAL POWNRM : ',
69:          & POWRGN, POWCAL, POWNRM
70:      end if
71: C
72:      do 140 M = 1, NMAT
73:          do 130 I = 1, NTNUC
74:              do 120 K = 1, 4
75:                  RATMIC(1,K,I,M) = POWNRM*RATMIC(1,K,I,M)
76:                  RATMIC(2,K,I,M) = POWNRM*RATMIC(2,K,I,M)
77:              120 continue
78:          130 continue
79:      140 continue
80: C
81:      POWERL = 0.0
82:      ERRNOW = 0.0
83: C
84:      do 160 M = 1, NMAT
85:          POWMAT(M) = 0.0
86:          TEMP = 0.0
87:          ERR = 0.0
88:          do 150 I = 1, NTNUC
89:              SAVE = RATMIC(1,1,I,M)*REACEV(1,I)
90:              &    + RATMIC(1,2,I,M)*REACEV(2,I)
91:              ESAVE = RATMIC(2,1,I,M)*REACEV(1,I)
92:              &    + RATMIC(2,2,I,M)*REACEV(2,I)
93:              TEMP = TEMP + SAVE*DNOLD(I,M)
94:              ERR = ERR + ESAVE*DNOLD(I,M)*ESAVE*DNOLD(I,M)
95:          150 continue
96:          POWMAT(M) = TEMP*PHIP(M)*ACOEF
97:          POWERL = POWERL + POWMAT(M)*VOLM(M)
98:          if(MATREG(M).gt.0) then
99:              ERRNOW = ERRNOW +
100:              & ERR*PHIP(M)*ACOEF*VOLM(M)*PHIP(M)*ACOEF*VOLM(M)
101:          endif
102:      160 continue
103: C
104:      if ( ERRNOW.gt.0.0) then
105:          ERRNOW = 100.0*SQRT(ERRNOW)/POWRGN
106:      else
107:          ERRNOW = 0.0
108:      endif
109: C
110: Ctemp if ( IBEDIT.le.2 ) return
111:      if ( IBEDIT.le.1 ) return
112: C
113:      write(NOUT1,*) ' ** J79 POWCAL POWNRM ERRNOW ** ',
114:      & J79,POWERL,POWNRM,ERRNOW
115:      write(NOUT1,*) ' ** POWMAT : ', POWMAT
116: C
117:      6888 format(/// XXX(BURNR4) ERROR-STOP :')
118: C
119:      return
120: end

```

src/mvpburn/burnrm.f

```

1:      subroutine BURNRM( NMAT, NTNUC, NOUT2, IBEDIT,VOLM,  PHIP,
2:      &                  POWMAT,DNOLD, RATMIC,POWERL,POWNRM,EVTOJ,
3:      &                  REACEV,J79,  NOUT1 ,ERRNOW )
4: C
5: C ** SET MATERIAL-WISE POWER ***
6: C ** SET NORMARIZATION OF MICROSCOPIC REACTION RATE ***
7: C
8:      real VOLM(NMAT)
9:      real PHIP(NMAT)
10:     real DNOLD(NTNUC,NMAT)
11:     real REACEV(2,NTNUC)
12:     real POWMAT(NMAT)
13:     real RATMIC(2,4,NTNUC,NMAT)
14: C
15: C *** SET ONE-GROUP MICROSCOPIC X-SECTION FOR BURNCL ROUTINE
16: C & SET HEAVY METAL WEIGHT
17: C
18:     call CLEA( POWMAT, NMAT, 0.0 )
19:     call CLEA( PHIP, NMAT, 1.0 )
20: C
21:     if ( POWERL.le.0.0 ) then
22:         POWNRM = 0.000
23:         ERRNOW = 0.0
24:         return
25:     end if
26: C
27: C *** CALCULATE POWER AND SET FACTOR FOR GETTING ABSOLUTE REACTION RATE
28: C
29:     ACOEF = EVTOJ
30:     POWCAL = 0.0
31:     do 110 M = 1, NMAT
32:         TEMP = 0.0
33:         do 100 I = 1, NTNUC
34:             SAVE = RATMIC(1,1,I,M)*REACEV(1,I) + RATMIC(1,2,I,M)*
35:             &      REACEV(2,I)
36:             TEMP = TEMP + SAVE*DNOLD(I,M)
37:         100 continue
38:         POWMAT(M) = TEMP*ACOEF
39:         POWCAL = POWCAL + POWMAT(M)*VOLM(M)
40:     110 continue
41: C
42: C --- CHECK ZERO POWER CASE
43: C
44:     if ( POWCAL.le.0.0 ) then
45:         write(NOUT1,6888)
46:         write(NOUT1,*) ' zero power was calculated'
47:         write(NOUT1,*) ' Check your input data '
48:         stop 888
49:     end if
50: C
51:     POWNRM = POWERL/POWCAL
52:     if ( IBEDIT.gt.2 ) then
53:         write(NOUT1,*) ' POWERL POWCAL POWNRM : ',
54:         & POWERL, POWCAL, POWNRM
55:     end if
56: C
57:     do 140 M = 1, NMAT
58:         do 130 I = 1, NTNUC
59:             do 120 K = 1, 4
60:                 RATMIC(1,K,I,M) = POWNRM*RATMIC(1,K,I,M)
61:                 RATMIC(2,K,I,M) = POWNRM*RATMIC(2,K,I,M)
62:             120 continue
63:         130 continue
64:     140 continue
65: C

```

```

66:     POWER0 = POWERL
67:     POWERL = 0.0
68:     ERRNOW = 0.0
69: C
70:     do 160 M = 1, NMAT
71:         POWMAT(M) = 0.0
72:         TEMP = 0.0
73:         ERR = 0.0
74:         do 150 I = 1, NTNUC
75:             SAVE = RATMIC(1,1,I,M)*REACEV(1,I) + RATMIC(1,2,I,M)*
76:             &      REACEV(2,I)
77:             ESAVE = RATMIC(2,1,I,M)*REACEV(1,I) + RATMIC(2,2,I,M)*
78:             &      REACEV(2,I)
79:             TEMP = TEMP + SAVE*DNOLD(I,M)
80:             ERR = ERR + ESAVE*DNOLD(I,M)*ESAVE*DNOLD(I,M)
81:         150 continue
82:         POWMAT(M) = TEMP*PHIP(M)*ACOEF
83:         POWERL = POWERL + POWMAT(M)*VOLM(M)
84:         ERRNOW = ERRNOW +
85:         &      ERR*PHIP(M)*ACOEF*VOLM(M)*PHIP(M)*ACOEF*VOLM(M)
86:     160 continue
87: C
88:     if ( ERRNOW.gt.0.0 ) then
89:         ERRNOW = 100.0*SQRT(ERRNOW)/POWERL
90:     else
91:         ERRNOW = 0.0
92:     endif
93: C
94:     if ( IBEDIT.le.2 ) return
95: C
96:     write(NOUT1,*) ' ** J79 POWERL POWCAL POWNRM ERRNOW ** ',
97:     &      J79, POWER0, POWERL, POWNRM, ERRNOW
98:     write(NOUT1,*) ' ** POWMAT : ', POWMAT
99: C
100: 6888 format(/// ' XXX(BURNRM) ERROR-STOP : ' )
101:     return
102: end

```

src/mvpburn/burnrv.f

```
1:      subroutine BURNRV( NMAT, NTNUC, NOUT2, IBEDIT,VOLM,  PHIP,
2:      &
3:      &
4:      C
5:      C ** SET MATERIAL-WISE POWER for Branching-Off case ***
6:      C ** SET NORMARIZATION OF MICROSCOPIC REACTION RATE ***
7:      C
8:      real VOLM(NMAT)
9:      real PHIP(NMAT)
10:     real DNOLD(NTNUC,NMAT)
11:     real REACEV(2,NTNUC)
12:     real POWMAT(NMAT)
13:     real RATMIC(2,4,NTNUC,NMAT)
14:     C
15:     C *** SET ONE-GROUP MICROSCOPIC X-SECTION FOR BURNCL ROUTINE
16:     C *** CALCULATE POWER AND SET FACTOR FOR GETTING ABSOLUTE REACTION RATE
17:     C
18:     call CLEA( POWMAT, NMAT, 0.0 )
19:     call CLEA( PHIP, NMAT, 1.0 )
20:     C
21:     if ( POWNRM.le.0.0 ) then
22:         POWERL = 0.0
23:         return
24:     end if
25:     C
26:     ACOEF = EVTOJ
27:     C
28:     do 140 M = 1, NMAT
29:         do 130 I = 1, NTNUC
30:             do 120 K = 1, 4
31:                 RATMIC(1,K,I,M) = POWNRM*RATMIC(1,K,I,M)
32:                 RATMIC(2,K,I,M) = POWNRM*RATMIC(2,K,I,M)
33:             120 continue
34:             130 continue
35:             140 continue
36:         C
37:         POWERL = 0.0
38:         do 160 M = 1, NMAT
39:             POWMAT(M) = 0.0
40:             TEMP = 0.0
41:             do 150 I = 1, NTNUC
42:                 SAVE = RATMIC(1,1,I,M)*REACEV(1,I)
43:                 &
44:                 + RATMIC(1,2,I,M)*REACEV(2,I)
45:                 TEMP = TEMP + SAVE*DNOLD(I,M)
46:             150 continue
47:             POWMAT(M) = TEMP*PHIP(M)*ACOEF
48:             POWERL = POWERL + POWMAT(M)*VOLM(M)
49:             160 continue
50:         C
51:         Ctemp if ( IBEDIT.le.2 ) return
52:         if ( IBEDIT.le.1 ) return
53:         C
54:         write(NOUT1,*) ' ** J79 POWERL POWNRM ** ', J79,
55:         &
56:         write(NOUT1,*) ' ** POWMAT : ', POWMAT
57:         C
58:         6888 format(/// XXX(BURNRV) ERROR-STOP : ' )
59:         C
60:         return
61:     end
```

src/mvpburn/burnrx.f

```
1:      subroutine BURNRX( NMAT, NTNUC, NOUT2, IBEDIT,VOLM,  PHIP,
2:      &
3:      &
4:      C
5:      C ** BURNUP mode is ADS or Moniterig Reaction Rate. : added Jan 2005
6:      C ** SET MATERIAL-WISE POWER ***
7:      C ** SET NORMARIZATION OF MICROSCOPIC REACTION RATE ***
8:      C
9:      real VOLM(NMAT)
10:     real PHIP(NMAT)
11:     real DNOLD(NTNUC,NMAT)
12:     real REACEV(2,NTNUC)
13:     real POWMAT(NMAT)
14:     real RATMIC(2,4,NTNUC,NMAT)
15:     C
16:     C *** SET ONE GROUP MICROSCOPIC X-SECTION FOR BURNCL ROUTINE
17:     C
18:     C
19:     POWERL = 0.0
20:     C
21:     call CLEA( POWMAT, NMAT, 0.0 )
22:     call CLEA( PHIP, NMAT, 1.0 )
23:     C
24:     if ( POWNRM.le.0.0 ) then
25:         return
26:     and if
27:     C
28:     C *** CALCULATE POWER AND SET FACTOR FOR GETTING ABSOLUTE REACTION RATE
29:     C
30:     ACOEF = EVTOJ
31:     C
32:     do 140 M = 1, NMAT
33:         do 130 I = 1, NTNUC
34:             do 120 K = 1, 4
35:                 RATMIC(1,K,I,M) = POWNRM*RATMIC(1,K,I,M)
36:                 RATMIC(2,K,I,M) = POWNRM*RATMIC(2,K,I,M)
37:             120 continue
38:             130 continue
39:         140 continue
40:     C
41:     do 160 M = 1, NMAT
42:         TEMP = 0.0
43:         do 150 I = 1, NTNUC
44:             SAVE = RATMIC(1,1,I,M)*REACEV(1,I) + RATMIC(1,2,I,M)*
45:             &
46:             REACEV(2,I)
47:             TEMP = TEMP + SAVE*DNOLD(I,M)
48:             150 continue
49:             POWMAT(M) = TEMP*PHIP(M)*ACOEf
50:             POWERL = POWERL + POWMAT(M)*VOLM(M)
51:         160 continue
52:     C
53:     if ( IBEDIT.le.2 ) return
54:     write(NOUT1,*) ' ** J79 POWERL POWNRM ** ', J79,POWERL,POWNRM
55:     write(NOUT1,*) ' ** POWMAT : ', POWMAT
56:     C
57:     6888 format(/// XXX(BURNRX) ERROR-STOP : ' )
58:     C
59:     return
60:     end
```

src/mvpburn/burnsm.f

```

1: C2003 add DENPR and JDENER
2:   subroutine BURNM( LENGSM, MEMORY, NOUT1, NOUT2, CASEID, MTPY,
3:     &               IMVPIN, TEMP, VOLM, TRGNAM, MTBURN, POWRZN,
4:     &               EXPSZN, HMINV, DMWZON, GAMAVG, YLDXE5, YLDI35,
5:     &               YLDSM9, YLDPM9, DENTAB, DENERR, JDENER,
6:     &               RATMIC, GAM, IARRAY,
7:     &               RARRAY, IOTSTP, NOTSTP )
8: C=====
9: C Control of burnup summary output
10: C=====
11: C
12: C --- Arguments ---
13: C
14: C   character*4 CASEID
15: C
16: C   character*12 TRGNAM(NMAT)
17: C   integer MTPY(NMAT), MTBURN(3, NMAT)
18: C   real TEMP(NMAT), VOLM(NMAT)
19: C
20: C   real POWRZN(NMAT, NEP1)
21: C   real EXPSZN(NMAT, NEP1)
22: C   real HMINV(NMAT, NEP1)
23: C   real DMWZON(NMAT, NEP1)
24: C   real YLDXE5(NMAT, NEP1)
25: C   real YLDI35(NMAT, NEP1)
26: C   real YLDSM9(NMAT, NEP1)
27: C   real YLDPM9(NMAT, NEP1)
28: C   real GAMAVG(NMAT, NEP1)
29: C
30: C   real DENTAB(NTNUC, NMAT, NEP1)
31: C2003
32: C   real DENERR(NTNUC, NMAT, NEP1)
33: C   real RATMIC(2, 4, NTNUC, NMAT, NEP1)
34: C   real GAM(NTFISS, NTNUC)
35: C
36: C   integer IARRAY(MEMORY)
37: C   real RARRAY(MEMORY)
38: C   integer IOTSTP(NOTSTP)
39: CCCC character*4 CARRAY(MEMORY)
40: C
41: C
42: C --- COMMON area ---
43: C
44: C
45: C   include 'INC/_BURNP'
46: C
47: C   include 'INC/_CBURNC1'
48: C   include 'INC/_CBURNC2'
49: C   include 'INC/_CBURNC3'
50: C
51: C   real INSCR, INTCR
52: C   common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
53: C   & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
54: C   & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
55: C   & INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
56: C   & FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
57: C   & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
58: C   & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
59: C   & NBATCH(MXSTEP)
60: C2005
61: C   & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
62: C   & RMONIT(MXSTEP), EMONIT(MXSTEP)
63: Cend
64: C
65: C --- local data ---

```

```

66: C
67: C2005 character*8 IDATE
68: C2005 character*72 MLINE
69: C
70: C -----
71: C
72: C   call CLEA( INTCR, MXSTEP, 0.0 )
73: C   call CLEA( EINTCR, MXSTEP, 0.0 )
74: C
75: C *** CALCULATES INETEGATED CONVERSION RATIO
76: C
77: C   SUMABS = 0.0
78: C   SUMCAP = 0.0
79: C2003 days=0 is special case ....
80: C   SMA = FISABS(1)+FDECAY(1)
81: C   SMC = FRTCAP(1)+CDECAY(1)
82: C   if ( SMA.gt.0.0 ) INTCR(1) = SMC/SMA
83: C2003 ... end
84: C2003 do 100 IT = 1, NOWSTP - 1
85: C   do 100 IT = 2, NOWSTP - 1
86: C2003   IT1 = IT + 1
87: C2003   DELT = DAYS(IT1) - DAYS(IT)
88: C2003   SUMABS = SUMABS + DELT*(FACNRM(IT)*FISABS(IT)+FDECAY(IT))
89: C2003   SUMCAP = SUMCAP + DELT*(FACNRM(IT)*FRTCAP(IT)+CDECAY(IT))
90: C   DELT = DAYS(IT) - DAYS(IT-1)
91: C   if ( POWERL(IT-1).gt.0 ) then
92: C     SUMABS = SUMABS + DELT*(FISABS(IT)+FDECAY(IT)
93: C     & + FISABS(IT-1)+FDECAY(IT-1))*0.5
94: C     SUMCAP = SUMCAP + DELT*(FRTCAP(IT)+CDECAY(IT)
95: C     & + FRTCAP(IT-1)+CDECAY(IT-1))*0.5
96: C   else
97: C     SUMABS = SUMABS + DELT*(FDECAY(IT)+FDECAY(IT-1))*0.5
98: C     SUMCAP = SUMCAP + DELT*(CDECAY(IT)+CDECAY(IT-1))*0.5
99: C   end if
100: C2003 ... end
101: C   if ( SUMABS.gt.0.0 ) INTCR(IT) = SUMCAP/SUMABS
102: C
103: C   if ( IBEDIT.ge.2 ) then
104: C     write(NOUT1,*)
105: C     & ' **IT DELT FACNRM FISABS FDECAY FRTCAP CDECAY **'
106: C     write(NOUT1,*) IT, DELT, FACNRM(IT), FISABS(IT), FDECAY(IT),
107: C     & FRTCAP(IT), CDECAY(IT), SUMABS, SUMCAP
108: C   end if
109: C
110: C   100 continue
111: C
112: C2003 .... add DENERR and flag JDENER are added (not used)
113: C   JDENER = 1/0 : print/not print density errors
114: C   DENERR : a dummy array when JDENER=0
115: C   call BURNPR( NOUT1, NOUT2, CASEID, MTPY, IMVPIN,
116: C   & TEMP, VOLM, TRGNAM, MTBURN, POWRZN, EXPSZN, HMINV, DMWZON,
117: C   & GAMAVG, YLDXE5, YLDI35, YLDSM9, YLDPM9, DENTAB, DENERR, JDENER,
118: C   & RATMIC, NOTSTP, IOTSTP )
119: C2003 ... end
120: C
121: C *** END OF PROCESS
122: C
123: C   return
124: C   end

```


src/mvpburn/burntm.f

```

1:      subroutine BURNM( NOWSTP,IBC2,  IBEDIT,IDU235,NMAT,  NTNUC,
2:      &                  DNOLD, RATMIC,MXSTEP,POWERL,TWTHVY,ST235,
3:      &                  PERIOD,DAYS,  U235F, EXPST, CUMMWD,DELDAY,VOLM,
4:      &                  NOUT2, NOUT1 )
5: C
6:      real VOLM(NMAT)
7:      real DNOLD(NTNUC,NMAT)
8:      real PERIOD(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP)
9:      real EXPST(MXSTEP),  CUMMWD(MXSTEP)
10:     real RATMIC(2,4,NTNUC,NMAT)
11: C
12:     if ( IBEDIT.gt.1 ) then
13:       write(NOUT1,*) ' ** NOWSTP IDU235 ST235 : ', NOWSTP, IDU235,
14:       &              ST235
15:     end if
16: C
17:     if ( NOWSTP.le.1 ) then
18:       call CLEA( DAYS, MXSTEP, 0.0 )
19:       call CLEA( U235F, MXSTEP, 0.0 )
20:       call CLEA( EXPST, MXSTEP, 0.0 )
21:       call CLEA( CUMMWD, MXSTEP, 0.0 )
22: C
23:       ST235 = 0.000000
24:       do 100 M = 1, NMAT
25:         ST235 = ST235 + DNOLD(IDU235,M)*VOLM(M)*1.00E+24
26: 100    continue
27: C
28:       if ( IBEDIT.gt.1 ) then
29:         write(NOUT1,*) ' ** NOWSTP IDU235 ST235 : ', NOWSTP, IDU235,
30:         &              ST235
31:       end if
32: C
33:     end if
34: C
35: C *** CASE FOR IBC2=1 : PERIOD IS GIVEN  BY CUMULATIVE MWD/T UNIT
36: C
37:     if ( IBC2.eq.1.and.POWERL.gt.0.0 ) then
38:       if ( NOWSTP.eq.1 ) then
39:         DELEXP = PERIOD(1)
40:       else
41:         DELEXP = PERIOD(NOWSTP) - PERIOD(NOWSTP-1)
42:       end if
43:       DELDAY = DELEXP*TWTHVY/POWERL
44:       EXPST(NOWSTP+1) = EXPST(NOWSTP) + DELEXP
45:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP) + DELDAY*POWERL
46:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
47:     end if
48: C
49:     if ( IBC2.eq.1.and.POWERL.le.0.0 ) then
50:       DELDAY = -PERIOD(NOWSTP)
51:       EXPST(NOWSTP+1) = EXPST(NOWSTP)
52:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP)
53:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
54:       PERIOD(NOWSTP) = EXPST(NOWSTP)
55:     end if
56: C
57: C *** CASE FOR IBC2=2 : PERIOD IS GIVEN  BY CUMULATIVE MWD UNIT
58: C
59:     if ( IBC2.eq.2.and.POWERL.gt.0.0 ) then
60:       if ( NOWSTP.eq.1 ) then
61:         DELEXP = PERIOD(1)
62:       else
63:         DELEXP = PERIOD(NOWSTP) - PERIOD(NOWSTP-1)
64:       end if
65:       DELDAY = DELEXP/POWERL
66:
67:       EXPST(NOWSTP+1) = EXPST(NOWSTP) + DELEXP/TWTHVY
68:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP) + DELEXP
69:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
70:     end if
71: C
72:     if ( IBC2.eq.2.and.POWERL.le.0.0 ) then
73:       DELDAY = -PERIOD(NOWSTP)
74:       EXPST(NOWSTP+1) = EXPST(NOWSTP)
75:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP)
76:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
77:       PERIOD(NOWSTP) = CUMMWD(NOWSTP)
78:     end if
79: C
80: C *** CASE FOR IBC2=3 : PERIOD IS GIVEN  BY CUMULATIVE DAYS UNIT
81: C
82:     if ( IBC2.eq.3 ) then
83:       if ( NOWSTP.eq.1 ) then
84:         DELDAY = PERIOD(1)
85:       else
86:         DELDAY = PERIOD(NOWSTP) - PERIOD(NOWSTP-1)
87:       end if
88:       RFACT = 0.000
89:       if ( TWTHVY.gt.0.0 ) RFACT = 1.00000/TWTHVY
90:       EXPST(NOWSTP+1) = EXPST(NOWSTP) + POWERL*DELDAY*RFACT
91:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP) + POWERL*DELDAY
92:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
93:     end if
94: C
95: C *** CASE FOR IBC2=4 : PERIOD IS GIVEN  BY DELTA DAYS UNIT
96: C
97:     if ( IBC2.eq.4 ) then
98:       DELDAY = PERIOD(NOWSTP)
99:       RFACT = 0.000
100:      if ( TWTHVY.gt.0.0 ) RFACT = 1.00000/TWTHVY
101:      EXPST(NOWSTP+1) = EXPST(NOWSTP) + POWERL*DELDAY*RFACT
102:      CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP) + POWERL*DELDAY
103:      DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
104:    end if
105: C
106: C *** CASE FOR IBC2=5 : PERIOD IS GIVEN  BY U-235 FRACTION
107: C
108:     if ( IBC2.eq.5.and.POWERL.gt.0.0 ) then
109:       if ( NOWSTP.eq.1 ) then
110:         DELU5 = PERIOD(1)
111:       else
112:         DELU5 = PERIOD(NOWSTP) - PERIOD(NOWSTP-1)
113:       end if
114:       U5ABS = 0.0
115:       do 110 M = 1, NMAT
116:         U5ABS = U5ABS + VOLM(M)*DNOLD(IDU235,M)*
117:         &      RATMIC(1,4,IDU235,M)
118: 110    continue
119:       SEC = DELU5*1.000E-2*ST235/U5ABS
120:       DELDAY = SEC/3600.00/24.0000
121:
122:       EXPST(NOWSTP+1) = EXPST(NOWSTP) + POWERL*DELDAY/TWTHVY
123:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP) + POWERL*DELDAY
124:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY
125:     end if
126: C
127:     if ( IBC2.eq.5.and.POWERL.le.0.0 ) then
128:       DELDAY = -PERIOD(NOWSTP)
129:       EXPST(NOWSTP+1) = EXPST(NOWSTP)
130:       CUMMWD(NOWSTP+1) = CUMMWD(NOWSTP)
131:       DAYS(NOWSTP+1) = DAYS(NOWSTP) + DELDAY

```

src/mvpburn/burntm.f

```
131:          PERIOD(NOWSTP) = U235F(NOWSTP)
132:      end if
133:  C
134:  C--- DELDAY=0.0, POWERL<0 : for parametric survey calculations
135:  C
136:      if ( IBEDIT.gt.1 .or. DELDAY.le.0 ) then
137:      if ( DELDAY.eq.0.0 .and. POWERL.lt.0.0 ) then
138:          write(NOUT1,7080)
139:          return
140:      else
141:          write(NOUT1,*) ' ** IBC2 NOWSTP TWTHVY ST235 POWERL DELDAY : ',
142:      &          IBC2, NOWSTP, TWTHVY, ST235, POWERL, DELDAY
143:          write(NOUT1,7000) (PERIOD(I),I=1,NOWSTP)
144:          write(NOUT1,7020) (DAYS(I),I=1,NOWSTP+1)
145:          write(NOUT1,7040) (EXPST(I),I=1,NOWSTP+1)
146:          write(NOUT1,7060) (CUMMWD(I),I=1,NOWSTP+1)
147:      endif
148:      end if
149:  C
150:      if ( DELDAY.le.0.0 ) then
151:          write(NOUT1,6888)
152:          write(NOUT1,*) ' zero or negative burnup time was calculated '
153:          write(NOUT1,*) ' check input PERIOD or POWERL data.'
154:          stop 888
155:      end if
156:  C2005 ... added
157:      if ( DELDAY.gt.0.0 .and. POWERL.lt.0.0 ) then
158:          write(NOUT1,6888)
159:          write(NOUT1,*) ' Negative Power was specified !!! '
160:          write(NOUT1,*) ' But positive burnup time was calculated.'
161:          write(NOUT1,*) ' Check your input PERIOD !!! '
162:          write(NOUT1,*)
163:          &      ' Recommendation is to set PERIOD(' ,NOWSTP,')=0.0 !!!'
164:          stop 888
165:      end if
166:  C2005 ... end
167:  C
168:      6888 format(/// ' XXX(BURNM) ERROR-STOP : ' )
169:      7000 format(1X,' PERIOD : ',1P,10E11.4)
170:      7020 format(1X,' DAYS   : ',1P,10E11.4)
171:      7040 format(1X,' EXPST  : ',1P,10E11.4)
172:      7060 format(1X,' CUMMWD : ',1P,10E11.4)
173:      7080 format(1X,'>>> DEPLETION CALCULATION IS SKIPPED IN THIS STEP',
174:      &          ' (ZERO PERIOD & NEGATIVE POWER)')
175:  C
176:      return
177:  end
```

src/mvpburn/burnup.f

```

1:      subroutine BURNUP( NOUT1, NOUT2, MEMORY,DIRIN, DIROUT,CASEID,SID0,
2:      &                  SID1, SID2, JPC,  ISPC,  TITL2, MTYP, NISO,
3:      &                  TEMP, VOLM, TRGNAM, LENTRG,MTBURN,MTCARD,
4:      &                  IDENT, DNINIT,MATMVP,IPBURN,NUCLP, GAM,  NBIC,
5:      &                  PBIC, GAMI, KSTP, LONG, LL,  IP,  KP,
6:      &                  NPN,  PHAI, DNOLD, DNNEW, DNWRK1,DNWRK2,
7:      &                  IDWORK,PHIP, WRKMAT,POWZN1,POWZN2,SSF,  SSC,
8:      &                  SSN2N, POWRZN,EXPSZN,HMINV, DMWZON,GAMAVG,
9:      &                  YLDXE5,YLDI35,YLDSM9,YLDPM9,DENTAB,RATMIC,
10: Ckuni  2003/12/21
11: Cmod &                  RPCMIC,CUMWD1,CUMWD2,IARRAY,RARRAY ,CASREF )
12:      &                  RPCMIC,CUMWD1,CUMWD2,IARRAY,RARRAY ,CASREF ,
13: C2005 &                  KPBURN,KZEROP )
14:      &                  KPBURN,KZEROP,MATREG,IBSTEP )
15: Cend
16: C=<MVPBURN>=====
17: C Purpose : Calculate number density change from burnup step SID1 to
18: C           SID2. Here SID1 and SID2 are burnup step IDs.
19: C
20: C   SID1 : get flux, reaction rates etc. from MVP calculation
21: C   SID2 : calculate new number density of this step.
22: C   SID0 : "old" number density is get from this step.
23: C           Normally SID0=SID1, but in the second substep of PC method
24: C           SID0= step i and SID1 = step i+1/2
25: C
26: C   NOWSTP (common) : current step # (counted from 1)
27: C   JPC : if non zero current step is calculated by PC method.
28: C   ISPC : for PC method, ISPC=1 is step NOWSTP --> NOWSTP+1/2
29: C           ISPC=2 is step NOWSTP --> NOWSTP+1
30: C           for non-PC case, ISPC=1.
31: C=====
32: C
33:      character*4 CASEID,CASREF
34:      character*2 SID0, SID1, SID2
35:      character*(*) DIRIN, DIROUT
36:      character*72 TITL2
37: C
38:      integer IARRAY(MEMORY)
39:      real RARRAY(MEMORY)
40: C
41: C2003
42:      character*12 IDENT(KNMAX,NMAT)
43:      character*12 TRGNAM(NMAT)
44: C
45:      integer MTYP(NMAT), NISO(NMAT), LENTRG(NMAT)
46:      integer IPBURN(KNMAX,NMAT), MATMVP(NMAT)
47:      integer MTBURN(3,NMAT), MTCARD(2,NMAT)
48: C
49:      real TEMP(NMAT), VOLM(NMAT)
50:      real DNINIT(KNMAX,NMAT)
51: C
52:      integer NUCLP(NPAR,NTNUC)
53:      real GAM(NTFISS,NTNUC)
54:      real GAMI(NTFISS,NTNUC)
55: C
56:      integer NBIC(NPAR,NTNUC)
57:      real PBIC(NPAR,NTNUC)
58:      integer KSTP(NCHA,NTNUC)
59:      integer LONG(NCHA,NTNUC)
60:      integer LL(NCHA,NTNUC)
61:      integer IP(LCHA,NCHA,NTNUC)
62:      integer KP(LCHA,NCHA,NTNUC)
63:      integer NPN(NPAR,NTNUC)
64:      real PHAI(NPAR,NTNUC)
65: C

```

```

66:      real DNOLD(NTNUC,NMAT)
67:      real DNNEW(NTNUC,NMAT)
68:      real DNWRK1(NTNUC,NMAT)
69:      real DNWRK2(NTNUC,NMAT)
70: C
71: C2003
72:      character*12 IDWORK(NTNUC,NMAT)
73: C
74:      real PHIP(NMAT)
75:      real POWZN1(NMAT), POWZN2(NMAT)
76:      real WRKMAT(NMAT), CUMWD1(NMAT), CUMWD2(NMAT)
77:      real SSF(NTNUC,NMAT)
78:      real SSC(NTNUC,NMAT)
79:      real SSN2N(NTNUC,NMAT)
80: C
81:      real POWRZN(NMAT,NEP1)
82:      real EXPSZN(NMAT,NEP1)
83:      real HMINV(NMAT,NEP1)
84:      real DMWZON(NMAT,NEP1)
85:      real YLDXE5(NMAT,NEP1)
86:      real YLDI35(NMAT,NEP1)
87:      real YLDSM9(NMAT,NEP1)
88:      real YLDPM9(NMAT,NEP1)
89:      real GAMAVG(NMAT,NEP1)
90: C
91:      real DENTAB(NTNUC,NMAT,NEP1)
92:      real RATMIC(2,4,NTNUC,NMAT,NEP1)
93:      real RPCMIC(2,4,NTNUC,NMAT)
94: Ckuni  2003/12/21
95:      integer KZEROP(NCHA,NTNUC),KPBURN(NTNUC,NMAT)
96: Cend
97: C2005 ... added
98:      integer MATREG(NMAT)
99: C2005 ... end
100: C
101:      include 'INC/_BURNP'
102: C
103:      include 'INC/_CBURNC1'
104:      include 'INC/_CBURNC2'
105:      include 'INC/_CBURNC3'
106: C
107:      real INSCR, INTCR
108:      common /MEMOCK/ MEUSED, MEMMAX
109:      common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
110:      & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
111:      & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
112:      & INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
113:      & FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
114:      & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
115:      & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
116:      & NBATCH(MXSTEP)
117: C2005 ... added
118:      & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
119:      & RMONIT(MXSTEP), EMONIT(MXSTEP)
120: C
121:      integer IBSTEP(MXSTEP)
122: C2005 ... end
123: C
124:      character*2 STEPNM
125:      external STEPNM
126: C
127: C ... local variables .....
128: C
129: C2005 character*72 DDNAME
130:      character*8 MEMBER

```

src/mvpburn/burnup.f

```

131:      real*8      DKEFF, DERROR
132:      character*120 LINE
133: C2003
134:      character*12  NUCID
135: C2003
136:      character*8   CWRK
137: C2003+
138: CKSK  character*8   CWRK2(NTNUC)
139:      character*8   CWRK2(MXNUC)
140: C
141: C-----
142: C
143:      J79      = NOWSTP
144:      K79      = NOWSTP + 1
145:      if (IBEDIT.gt. 0) then
146:        write(NOUT1,7040) J79
147:      end if
148: C
149:      NBOPT    = 0
150:      POWNOW   = POWERL(J79)
151: C2005 ... added
152:      IBNTYP = 0
153:      IBC10  = 0
154:      if (IVOID.eq.0) then
155:        IBNTYP = IBC(9)
156:        IBC10  = IBC(10)
157:      end if
158: C2005 ... end
159: C
160: C POWERL < 0.0 & PERIOD(J79) < 0.0 BURNUP CALCULATION SKIP
161: C POWERL < 0.0 & PERIOD(J79) = 0.0 BURNUP CALCULATION SKIP
162: C
163: C2005 if (POWNOW.lt.0.0 .and. PERIOD(J79).lt.0.0) then
164:   if (POWNOW.lt.0.0 .and. PERIOD(J79).le.0.0) then
165:     NOBURN = 1
166:     POWNOW = ABS(POWNOW)
167:   else
168:     NOBURN = 0
169:   end if
170: C
171:   if ( POWNOW.eq.0.0 ) then
172:     NBOPT = 1
173:   end if
174: C
175:   MEMBER = ' '
176: C
177: C *****
178: C *** READ MATERIAL DATA
179: C *****
180: C
181:   call RWMATD( 'READ', DIRIN, CASEID, NMAT, KNMAX, NISO(1), TEMP(1),
182: &             VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1), MTCARD(1,1),
183: C2005&         IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) )
184: &         IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) ,
185: &         MATREG(1) )
186: C
187:   if ( IBEDIT.ge.2 ) then
188:     write(NOUT1,7060) (VOLM(I),I=1,NMAT)
189:   end if
190: C
191: C *****
192: C *** READ CHAIN DATA
193: C *****
194: C
195:   call RWCHAN( 'READ', DIRIN, CASEID, NPAR, NTNUC, NTFISS,

```

```

196: &             NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
197: C
198: C
199: C *****
200: C *** READ HISTORY DATA until step SID0
201: C *****
202: C
203:   do 150 LOP = 1, NOWSTP
204: C
205:     LENGHT = 0
206: C
207:     if ( LOP.eq.NOWSTP ) then
208:       if ( IVOID.eq.0 ) then
209:         MEMBER = CASEID//'HT'//SID0
210:       else
211:         MEMBER = CASREF//'HT'//SID0
212:       end if
213:     else
214:       if ( IVOID.eq.0 ) then
215:         MEMBER = CASEID//'HT'//STEPNM(LOP,0)
216:       else
217:         MEMBER = CASREF//'HT'//STEPNM(LOP,0)
218:       end if
219:     end if
220: C2003 ... check member size
221:   call PDSIO( 'SEARCH',DIROUT,MEMBER, MEMBER, RARRAY, LENGHT,
222: &             NOUT1 )
223:   if (LENGHT.gt.MEMORY) then
224:     write(NOUT1,*) 'XXX(BURNUP) Size of PDS member ',MEMBER,
225: &               ' (=' ,LENGHT,') exceeds buffer size (=' ,MEMORY,')'
226:     stop 999
227:   else
228:     MTMP = MEMMAX - MEMORY + LENGHT
229:     if (MTMP.gt.MEUSED) MEUSED = MTMP
230:   end if
231: C2003 ... end
232:   call PDSIO( 'READ', DIROUT, MEMBER, MEMBER, RARRAY, LENGHT,
233: &             NOUT1 )
234: C
235:   if ( IBEDIT.ge.1 ) write(NOUT1,*) ' ** LENGHT : ', LENGHT
236: C
237:   LOP1 = LOP - 1
238: C
239:   call PCTRW( IARRAY, MEMBER, IRET )
240: C
241:   call UNPKND( IARRAY, 'STEP#', 'I4', ISWSTP, 1, NDATA2, IRET )
242: C
243:   call PCTLB( IARRAY, MEMBER, 'HMINV', IRET )
244:   call UNPKND( IARRAY, 'HMINV', 'R4', HMINV(1,LOP), NMAT, NDATA2,
245: &             IRET )
246: C
247:   call PCTLB( IARRAY, MEMBER, 'EXPSZN', IRET )
248:   call UNPKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1,LOP), NMAT,
249: &             NDATA2, IRET )
250: C
251:   call PCTLB( IARRAY, MEMBER, 'DMWZON', IRET )
252:   call UNPKND( IARRAY, 'DMWZON', 'R4', DMWZON(1,LOP), NMAT,
253: &             NDATA2, IRET )
254: C
255:   call PCTLB( IARRAY, MEMBER, 'DNBURN', IRET )
256:   call UNPKND( IARRAY, 'DNBURN', 'R4', DENTAB(1,1,LOP), NTNUC*
257: &             NMAT, NDATA2, IRET )
258: C
259: C ... data appended to case+'HT'+step after burn up of the step ...
260: C

```

src/mvpburn/burnup.f

```

261:         if ( LOP.ne.NOWSTP ) then
262:
263:             call PCTLB( IARRAY, MEMBER, 'POWRZN', IRET )
264:             call UNPKND( IARRAY, 'POWRZN', 'R4', POWRZN(1,LOP), NMAT,
265: &                     NDATA2, IRET )
266:
267:             call PCTLB( IARRAY, MEMBER, 'GMAVGV', IRET )
268:             call UNPKND( IARRAY, 'GMAVGV', 'R4', GAMAVG(1,LOP), NMAT,
269: &                     NDATA2, IRET )
270:
271:             call PCTLB( IARRAY, MEMBER, 'YLDXE5', IRET )
272:             call UNPKND( IARRAY, 'YLDXE5', 'R4', YLDXE5(1,LOP), NMAT,
273: &                     NDATA2, IRET )
274:
275:             call PCTLB( IARRAY, MEMBER, 'YLDI35', IRET )
276:             call UNPKND( IARRAY, 'YLDI35', 'R4', YLDI35(1,LOP), NMAT,
277: &                     NDATA2, IRET )
278:
279:             call PCTLB( IARRAY, MEMBER, 'YLDISM9', IRET )
280:             call UNPKND( IARRAY, 'YLDISM9', 'R4', YLDISM9(1,LOP), NMAT,
281: &                     NDATA2, IRET )
282:
283:             call PCTLB( IARRAY, MEMBER, 'YLDPM9', IRET )
284:             call UNPKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOP), NMAT,
285: &                     NDATA2, IRET )
286:
287:             call PCTLB( IARRAY, MEMBER, 'RATMIC', IRET )
288:             do 100 M = 1, NMAT
289:                 call UNPKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOP),
290: &                     2*4*NTNUC, NDATA2, IRET )
291:             100 continue
292:             end if
293: C
294: C ... save reaction rate of step SID0 in RPCMIC for the second
295: C substep of PC method. and restore value before power
296: C normalization.
297: C
298:         if ( ISPC.eq.2.and.LOP.eq.NOWSTP ) then
299: C2005 ... deleted
300: Cdel         call PCTLB( IARRAY, MEMBER, 'REAL', IRET )
301: Cdel         call UNPKND( IARRAY, 'REAL', 'R4', LP, NDATA2, IRET )
302: Cdel         POWNRM = RARRAY(LP+10)
303: C2005 ... end
304:         call PCTLB( IARRAY, MEMBER, 'RATMIC', IRET )
305: C
306:         do 140 M = 1, NMAT
307: C
308:             call UNPKND( IARRAY, 'RATMIC', 'R4', RPCMIC(1,1,1,M), 2*4
309: &                     *NTNUC, NDATA2, IRET )
310:         140 continue
311:         end if
312:         150 continue
313: C
314: C C00/01/20 DEL
315: C         if ( POWNRM.ne.0.0 ) then
316: C             do 130 I = 1, NTNUC
317: C                 do 120 MT = 1, 4
318: C                     do 110 K = 1, 2
319: C                         RPCMIC(K,MT,I,M) = RPCMIC(K,MT,I,M) /
320: C                             POWNRM
321: C                 &
322: C             110 continue
323: C             120 continue
324: C             130 continue
325: C         end if

```

```

326: C 140         continue
327: C             end if
328: C 150 continue
329: C
330: C *** SET DNOLD,DNWRK1,IDWORK
331: C
332:         call CLEA( DNOLD, NTNUC*NMAT, 0.0 )
333:         call CLEA( DNWRK1, NTNUC*NMAT, 0.0 )
334: Ckuni 2003/12/21
335:         call ICLEA( KPBURN, NTNUC*NMAT, 0 )
336: C
337:         do 180 M = 1, NMAT
338:             MMK = NISO(M)
339:             IST = MTBURN(1,M)
340:             IEND = MTBURN(2,M)
341:             MTYP(M) = MTBURN(3,M)
342:             RTEMP = TEMP(M)
343:             do 170 I = IST, IEND
344:                 DSAVE = DENTAB(I,M,NOWSTP)
345:                 DNOLD(I,M) = DSAVE
346:                 if ( DSAVE.le.0.0.and.NOWSTP.eq.1 ) DSAVE = BZERO
347: CMOD 970221
348:                 if ( NOWSTP.gt.1.and.DSAVE.lt.BZERO ) then
349:                     if ( POWERL(K79).gt.0.0 ) then
350:                         DSAVE = BZERO
351:                     end if
352:                 end if
353: C
354:                 DNWRK1(I,M) = DSAVE
355:                 NUCID = ' '
356:                 ISW = 0
357:                 do 160 K = 1, MMK
358:                     if ( IPBURN(K,M).eq.I ) ISW = K
359:                 160 continue
360: C
361:                 if ( ISW.eq.0 ) then
362: C2003             NUCID(1:4) = SRACID(I) (2:4) //'0'
363: C2003 ... make nuclide ID from NCODE
364:                     call ZZMM(1,1,NCODE(I),NUCID)
365:                     if(NUCID(6:6).eq.' ') NUCID(6:6) = '0'
366:                     if(NUCID(6:6).eq.'M') NUCID(6:6) = '1'
367:                     NUCID(7:10) = '0000'
368: C2003 ... end
369:                     call SETMID( NUCID, RTEMP )
370:                 else
371:                     NUCID = IDENT(ISW,M)
372:                 end if
373: C
374:                 IDWORK(I,M) = NUCID
375: C
376:                 if ( IBEDIT.ge.2 ) then
377:                     write(NOUT1,7000) M, I, NUCID, DNOLD(I,M), DSAVE
378:                 end if
379: Ckuni 2003/12/22
380:                 if( MTYP(M).eq.1.and.I.le.LASTFP ) then
381:                     KPBURN(I,M) = 1
382:                 else
383:                     if( ISW.gt.0 ) then
384:                         KPBURN(I,M) = 1
385:                     end if
386:                 end if
387: Cend
388:         170 continue
389:         180 continue
390: C

```

src/mvpburn/burnup.f

```

391: C2003
392: C7000 format(1X,' M I NUCID DNOLD DNBURN : ',2I4,2X,A8,2X,1P,2E12.5)
393: 7000 format(1X,' M I NUCID DNOLD DNBURN : ',2I4,2X,A,1P,2E12.5)
394: C
395: C *****
396: C *** GET MICROSCOPIC REACTION RATE
397: C *****
398: C
399: C2005 ... deleted
400: Cdel if ( NBOPT.eq.0 ) then
401: C2005 ... end
402: METHOD = 0
403: C2003 if ( IBC(20).gt.0 ) METHOD = 1
404: if ( IBC(20).eq.2 ) METHOD = 1
405: C
406: if ( METHOD.eq.0 ) then
407: LIN = 30
408: C
409: C ... number density data DNWRK1 used in REDMVP should be that of
410: C step SIDI for second substep of PC method ...
411: C ( used to get reaction rate per atom )
412: C
413: if ( ISPC.eq.2 ) then
414: MEMBER = CASEID//'VR'//SIDI
415: C2003 ... check member size
416: call PDSIO( 'SEARCH', DIROUT, MEMBER, MEMBER, RARRAY,
417: & LENGHT, NOUT1 )
418: if ( LENGHT.gt.MEMORY ) then
419: write(NOUT1,*) 'XXX(BURNUP) Size of PDS member ',
420: & MEMBER,
421: & ' (=' ,LENGHT,') exceeds buffer size (=' ,MEMORY,') '
422: & stop 999
423: end if
424: C2003 ... end
425: call PDSIO( 'READ', DIROUT, MEMBER, MEMBER, RARRAY,
426: & LENGHT, NOUT1 )
427: call PCTRW( IARRAY, MEMBER, IRET )
428: C
429: call PCTLB( IARRAY, MEMBER, 'DNBURN', IRET )
430: call UNPKND( IARRAY, MEMBER, 'R4', DNWRK1(1:1), NTNUC
431: & NMAT, NDATA2, IRET )
432: end if
433: C
434: C ... open MVP binary output as virtual member of PDS ...
435: C
436: MEMBER = CASEID//'VR'//SIDI
437: call PDSFIL( 'READ', DIROUT, MEMBER, 'BINARY', LIN, NOUT1,
438: & IERR0 )
439: C
440: C
441: C **** Read k-eff and micro reaction rate from MVP output.
442: C RATMIC is normalized to per volume and per atom
443: C
444: LENG = MEMORY/2
445: C2003 ... pass IBC to REDMVP
446: Cm call REDMVP( RARRAY, LENG, LIN, NMAT, NTNUC, VOLM, MATMVP,
447: Cm & RATMIC(1,1,1,1,J79), DNWRK1, IDWORK, DKEFF, DERROR,
448: Cm & NHIST(J79), NBATCH(J79), MTBURN, TITL2, NOUT1,
449: Cm & NOUT2, IBC, KPBURN )
450: Ckuni& NOUT2, IBC)
451: C2005 ... modified
452: KHIST = 0
453: KBATCH = 0
454: RATNOW = 0.0
455: ERRNOW = 0.0

```

```

456: ISWRAT = 0
457: C
458: C ..... No specail tally data edit for Branching-off case
459: if( IVOID.eq.0.and.IBNTYP.eq.2 ) ISWRAT = 1
460: if( IVOID.eq.0.and.IBNTYP.eq.3 ) ISWRAT = 1
461: C
462: C ..... if you want to get specail tally data edit for Branching-off case
463: C if( IBNTYP.eq.2.or.BNTYP.eq.3 ) ISWRAT = 1
464: C
465: call REDMVP( RARRAY, LENG, LIN, NMAT, NTNUC, VOLM, MATMVP,
466: & RATMIC(1,1,1,1,J79), DNWRK1, IDWORK, DKEFF, DERROR,
467: & KHIST, KBATCH, MTBURN, TITL2, NOUT1,
468: & NOUT2, IBC, KPBURN, ISWRAT, IDMONI, RATNOW, ERRNOW )
469: C2005 ... end
470: C
471: call PDSFIL( 'CLOSE', DIROUT, MEMBER, 'BINARY', LIN, NOUT1,
472: & IERR0 )
473: C
474: if ( ISPC.eq.1 ) then
475: AKEFF(NOWSTP) = DKEFF
476: ERKEF(NOWSTP) = DERROR
477: C2005 ... added
478: NHIST(NOWSTP) = KHIST
479: NBATCH(NOWSTP) = KBATCH
480: RMONIT(NOWSTP) = RATNOW
481: EMONIT(NOWSTP) = ERRNOW
482: C2005 ... end
483: end if
484: C00/01/20 Move
485: C
486: C ... For PC method, mix RATMIC of case SIDI (i) and SIDI (i+1/2)
487: C
488: if ( ISPC.eq.2 ) then
489: C do 220 M = 1, NMAT
490: C do 210 I = 1, NTNUC
491: C do 200 MT = 1, 4
492: C do 190 K = 1, 2
493: C RATMIC(K,MT,I,M,J79) = 0.5*
494: C & (RATMIC(K,MT,I,M,J79)
495: C & +RPCMIC(K,MT,I,M))
496: C 190 continue
497: C 200 continue
498: C 210 continue
499: C 220 continue
500: C end if
501: C
502: C ... dummy reaction rate only for program development
503: C (not used now)
504: C else
505: C
506: call RWIND( 86 )
507: do 250 LOP = 1, J79
508: IERR = 0
509: read(86,'(A120)') LINE
510: if ( LINE(1:3).ne.'@@' ) IERR = 1
511: read(LINE(23:120),*) NOW, NTDEPZ, NTNUC0
512: if ( NOW.ne.LOP ) IERR = 1
513: if ( NTDEPZ.ne.NMAT ) IERR = 1
514: if ( NTNUC0.ne.NTNUC ) IERR = 1
515: C
516: if ( IERR.eq.1 ) then
517: write(NOUT1,6888)
518: write(NOUT1,*) ' dummy burn-up data file is invalid'
519: stop
520: end if

```

src/mvpburn/burnup.f

```

521: C
522:       do 240 MOP = 1, NMAT
523:         read(86,'(A120)') LINE
524:         read(LINE(61:62),'(I2)') M
525:         do 230 N = 1, NTNUC
526:           read(86,'(9X,A4,3(6X,E12.5),8X,E12.5)') NUCID,
527:             read(86,'(9X,A6,3(6X,E12.5),8X,E12.5)') NUCID(1:6),
528: Cend
529:       &
530:         if ( LOP.eq.J79 ) then
531:           write(NOUT1,7020)
532:           &
533:           &
534:           &
535:           &
536:           &
537:           &
538:           &
539:           &
540:           &
541:           &
542:           &
543:           &
544:           &
545:           &
546:           &
547:           &
548:           &
549:           &
550:           &
551:           &
552:           &
553:           &
554:           &
555:           &
556:           &
557:           &
558:           &
559:           &
560:           &
561:           &
562:           &
563:           &
564:           &
565:           &
566:           &
567:           &
568:           &
569:           &
570:           &
571:           &
572:           &
573:           &
574:           &
575:           &
576:           &
577:           &
578:           &
579:           &
580:           &
581:           &
582:           &
583:           &
584:           &
585:           &

```

```

586:       & 'CURRENT 1-G EFFECTIVE MICRO. REACTION RATE OF DEPLETING ZONE ',
587:       & I2,2X,5(''')//)
588: C2003
589: C7120 format(' ID=',I2,2X,A4,' SIGF=',E12.5,' SIGC=',E12.5,' SIGN2N=',
590: 7120 format(' ID=',I2,2X,A6,' SIGF=',E12.5,' SIGC=',E12.5,' SIGN2N=',
591: & E12.5)
592: 7140 format(///1X,7X,5('''),2X,
593: & 'CURRENT 1-G EFFEC. PC MICRO. REACTION RATE OF DEPLETING ZONE ',
594: & I2,2X,5(''')//)
595: 7160 format(///1X,7X,5('''),2X,
596: & 'READING 1-G EFFEC. PC MICRO. REACTION RATE OF DEPLETING ZONE ',
597: & I2,2X,5(''')//)
598: C
599: C *****
600: C 1. SET ONE-GROUP MICROSCOPIC RACTION RATE FOR BURNCL ROUTINE
601: C & SET HEAVY METAL WEIGHT
602: C 2. SET ABSOLUTE FLUX LEVEL AND MATERIAL-WISE POWER
603: C *****
604: C
605: C
606: C ... copy DNOLD to DNNEW ...
607: C call MEMOVE( DNOLD, DNNEW, NTNUC*NMAT )
608: C
609: C POWNRM = 0.0
610: C
611: C *** normalize reaction rate *****
612: C
613: C00/01/21 Number density data DNWRK1 used in BURNRM should be that of
614: C step SID1 for second substep of PC method
615: C
616: C2005 ... modified
617: C
618: C if ( IVOID.eq.1 ) then
619: C POWNRM = FLXNRM(J79)
620: C POWNOW = 0.0
621: C call BURNRV( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
622: & POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRM,
623: & EVTOJ, REACEV, J79, NOUT1 )
624: C POWERW(J79) = POWNOW
625: C POWERE(J79) = ERRNRM(J79)
626: C
627: C
628: C if ( IVOID.eq.0 ) then
629: C
630: C if ( IBNTYP.eq.0 ) then
631: C POWNOW = POWERL(J79)
632: C POWNOW = ABS(POWNOW)
633: C ERRNOW = 0.0
634: C if ( ISPC.eq.1 ) then
635: C call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
636: & POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRM,
637: & EVTOJ, REACEV, J79, NOUT1 , ERRNOW )
638: C POWERW(J79) = POWNOW
639: C POWERE(J79) = ERRNOW
640: C FLXNRM(J79) = POWNRM
641: C ERRNRM(J79) = ERRNOW
642: C else
643: C call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
644: & POWRZN(1,J79), DNWRK1, RATMIC(1,1,1,1,J79), POWNOW,POWNRM,
645: & EVTOJ, REACEV, J79, NOUT1 , ERRNOW )
646: C end if
647: C
648: C else if ( IBNTYP.eq.1 ) then
649: C POWNOW = 0.0
650: C POWNRM = EXTSRC(J79)

```

src/mvpburn/burnup.f

```

651:      POWNRM = ABS(POWNRN)
652: C
653:      if ( IBEDIT.ge.2 ) then
654:        write(NOUT1,*) ' ** J79 EXTSRC(J79) : ',J79,POWNRN
655:      end if
656: C
657:      if ( ISPC.eq.1 ) then
658:        call BURNRX(NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
659: &      POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
660: &      EVTOJ, REACEV, J79, NOUT1 )
661:        POWERW(J79) = POWNOW
662:        POWERE(J79) = 0.0
663:        FLXNRM(J79) = POWNRN
664:        ERRNRM(J79) = 0.0
665:      else
666:        call BURNRX(NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
667: &      POWRZN(1,J79), DNWRK1, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
668: &      EVTOJ, REACEV, J79, NOUT1 )
669:      end if
670: C
671:      else if ( IBNTYP.eq.2.or.IBNTYP.eq.3 ) then
672:        POWNOW = 0.0
673:        RATMON = VMONIT(J79)
674:        RATMON = ABS(RATMON)
675:        if ( ISPC.eq.1 ) then
676:          if(RATNOW.gt.0.0.and.RATMON.gt.0.0) then
677:            POWNRN = RATMON/RATNOW
678:          else
679:            POWNRN = 0.0
680:            ERRNOW = 0.0
681:          end if
682: C
683:          if ( IBEDIT.ge.2 ) then
684:            write(NOUT1,*) ' ** J79 ISPC      : ',J79,ISPC
685:            write(NOUT1,*) ' ** VMONIT POWNRN : ',VMONIT(J79),POWNRN
686:            write(NOUT1,*) ' ** RATNOW ERRNOW : ',RATNOW,ERRNOW
687:          end if
688: C
689:          call BURNRX(NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
690: &      POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
691: &      EVTOJ, REACEV, J79, NOUT1 )
692:          POWERW(J79) = POWNOW
693:          POWERE(J79) = ERRNOW
694:          FLXNRM(J79) = POWNRN
695:          ERRNRM(J79) = ERRNOW
696: C
697:        else
698:          POWNOW = 0.0
699:          POWNRN = FLXNRM(J79)
700:          OLDNRN = FLXNRM(J79)
701:          if(IBC10.eq.1.and.RATNOW.gt.0.0.and.RATMON.gt.0.0) then
702:            POWNRN = RATMON/RATNOW
703:          end if
704: C
705:          if ( IBEDIT.ge.2 ) then
706:            write(NOUT1,*) ' ** J79 ISPC IBC10 : ',J79,ISPC,IBC10
707:            write(NOUT1,*) ' ** VMONIT POWNRN OLDNRN : ',
708: &      VMONIT(J79),POWNRN,OLDNRN
709:          &
710:            write(NOUT1,*) ' ** RATNOW ERRNOW : ',RATNOW,ERRNOW
711:          end if
712: C
713:          call BURNRX( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
714: &      POWRZN(1,J79), DNWRK1, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
715: &      EVTOJ, REACEV, J79, NOUT1 )
716:        end if

```

```

716: C
717:      else if ( IBNTYP.eq.4 ) then
718:        POWRGN = REGPOW(J79)
719:        POWRGN = ABS(POWRGN)
720: C
721:      if ( IBEDIT.ge.2 ) then
722:        write(NOUT1,*) ' ** J79 REGPOW(J79) : ',J79,REGPOW(J79)
723:        write(NOUT1,*) ' ** NMAT is ',NMAT,' & MATREG is : '
724:        write(NOUT1,'(10I4)') (MATREG(M),M=1,NMAT)
725:      end if
726: C
727:      if ( ISPC.eq.1 ) then
728:        call BURNR4( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
729: &      POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
730: &      EVTOJ, REACEV, J79, NOUT1, MATREG, POWRGN ,ERRNOW )
731:        POWERW(J79) = POWNOW
732:        POWERE(J79) = ERRNOW
733:        FLXNRM(J79) = POWNRN
734:        ERRNRM(J79) = ERRNOW
735:      else
736:        call BURNR4( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
737: &      POWRZN(1,J79), DNWRK1, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
738: &      EVTOJ, REACEV, J79, NOUT1, MATREG, POWRGN ,ERRNOW )
739:      end if
740:    end if
741: C2005 ... end
742:    end if
743: C
744:    if ( IBEDIT.ge.2 ) then
745:      write(NOUT1,*) ' ***** POWER ***** '
746:      write(NOUT1,*) (POWRZN(M,J79),M=1,NMAT)
747:    end if
748: C2005
749: Cdel FLXNRM(J79) = 0
750: Cdel if ( POWNRN.gt.0.0 ) FLXNRM(J79) = POWNRN
751: Cend
752: C00/01/20 Move
753: C
754: C ... For PC method, mix RATMIC of case SID0 (i) and SID1 (i+1/2)
755: C
756:    if ( ISPC.eq.2 ) then
757:      do 220 M = 1, NMAT
758:        do 210 I = 1, NTNUC
759:          do 200 MT = 1, 4
760:            do 190 K = 1, 2
761:              RATMIC(K,MT,I,M,J79) = 0.5*
762: &      (RATMIC(K,MT,I,M,J79)
763: &      +RPCMIC(K,MT,I,M))
764:            190      continue
765:          200      continue
766:        210      continue
767:      220      continue
768: C
769: Cmod call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
770: Cmod &      POWRZN(1,J79), DNOLD, RATMIC(1,1,1,1,J79), POWNOW, POWNRN,
771: Cmod &      EVTOJ, REACEV, J79, NOUT1 )
772: C00/01/20 Move end
773: C
774:    if ( IBNTYP.eq.0 ) then
775:      POWNOW = POWERL(J79)
776:      POWNOW = ABS(POWNOW)
777:      call BURNRM( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
778: &      POWRZN(1,J79),DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWNRN,
779: &      EVTOJ , REACEV, J79, NOUT1 , ERRNOW )
780:    end if

```


src/mvpburn/burnup.f

```

781: C
782:       if ( IBNTYP.eq.4 ) then
783:         POWRGN = REGPOW(J79)
784:         POWRGN = ABS(POWRGN)
785:         call BURNR4( NMAT, NTNUC, NOUT2, IBEDIT, VOLM, PHIP,
786:           & POWRZN(1,J79),DNOLD, RATMIC(1,1,1,1,J79), POWNOW,POWRNM,
787:           & EVTOJ , REACEV, J79, NOUT1, MATREG, POWRGN ,ERRNOW )
788:       end if
789: C
790:     end if
791: C
792:     do 290 M = 1, NMAT
793:       do 280 I = 1, NTNUC
794:         SSF(I,M) = RATMIC(1,1,I,M,J79)
795:         SSC(I,M) = RATMIC(1,2,I,M,J79)
796:         SSN2N(I,M) = RATMIC(1,3,I,M,J79)
797:       280 continue
798:     290 continue
799: C
800:       if ( IBEDIT.ge.1 ) then
801:         do 310 M = 1, NMAT
802:           write(NOUT1,7100) M
803:           do 300 I = 1, NTNUC
804: C2003 ... use 6 byte nuclide ID
805: CCCC       write(NOUT1,7120) I, SRACID(I), SSF(I,M), SSC(I,M),
806:           call ZZMMM(1,I,NCODE(I),CWRK(1:6))
807:           write(NOUT1,7120) I, CWRK(1:6), SSF(I,M), SSC(I,M),
808:           & SSN2N(I,M)
809: C2003 ... end
810:       300 continue
811:     310 continue
812:       end if
813: C
814:       call CPUTM2( T1 )
815:       FACPOW = 1.000
816:       DELDAY = 0.0
817: C2005 ... added
818:       FACNRM(J79) = FACPOW
819:       if ( NBOPT.eq.1 ) FACNRM(J79) = 0.0
820: C2005 ... end
821: C
822: C *****
823: C **** CASE FOR NORMAL (NON-BRANCH) BURNUP CALCULATION
824: C *****
825: C
826: CMOD if ( IVOID.eq.0 ) then
827: C2005 if ( IVOID.eq.0.and.NOBURN.eq.0 ) then
828:       if ( IVOID.eq.0 ) then
829:         DELDAY = 0.0
830:         call CLEA( WRKMAT, NMAT, 0.0 )
831: C
832: C .... No Burnup case
833: C
834:       if( NOBURN.eq.1 ) then
835:         DELT = 0.0
836:         FACPOW = 1.0
837:         if(IBC(2).eq.1) PERIOD(J79) = EXPST(J79)
838:         if(IBC(2).eq.2) PERIOD(J79) = CUMMWD(J79)
839:         if(IBC(2).eq.3) PERIOD(J79) = DAYS(J79)
840:         if(IBC(2).eq.4) PERIOD(J79) = 0.0
841:         if(IBC(2).eq.5) PERIOD(J79) = U235F(J79)
842:         write(NOUT1,8080)
843: C
844: 8080 format(1X,'>>> DEPLETION CALCULATION IS SKIPPED IN THIS STEP',
845:       & ' (ZERO PERIOD & NEGATIVE POWER)')

```

```

846: C
847: C .... Normal Burnup case
848: C
849:       else
850: C
851:         call BURNM( J79, IBC(2), IBEDIT, IDU235, NMAT, NTNUC, DNOLD,
852:           & RATMIC(1,1,1,1,J79), MXSTEP, POWNOW, TWTHVY, ST235,
853:           & PERIOD, DAYS, U235F, EXPST, CUMMWD, DELDAY, VOLM,
854:           & NOUT2, NOUT1 )
855: C
856:         DELT = DELDAY
857:         ITD = NSTP(J79)
858:         IDBG = IBEDIT - 2
859: C
860:
861:         FACPOW = 0.0
862: Cmoved call CLEA( WRKMAT, NMAT, 0.0 )
863: C
864: C2003 ... pass IBC as an argument
865: C
866:         call BURNCL( PHIP, VOLM, DELT, NMAT, NBOPT, NOUT1, ITD, MXNUC,
867:           & NTNUC, NTFISS, NPAR, EVTOJ, REACEV, DNNEW, DNOLD, SSF,
868:           & SSC, SSN2N, WRKMAT, IDBG, NOUT2, LCHA, NCHA, MTBURN,
869:           & NUCLP, GAM, GAMI, NBIC, PBIC, KSTP, LONG, LL, IP, KP,
870:           & NPN, PHAI, IBC(2), ST235, PERIOD(J79), IDU235, FACPOW,
871:           & POWZN1, POWZN2, NUCL, NCH, RAMDA, FACT2N, IBC )
872:         Ckuni&
873:         C
874:         if( IBNTYP.eq.0 ) then
875:           call BURNCL( PHIP, VOLM, DELT, NMAT, NBOPT, NOUT1, ITD, MXNUC,
876:             & NTNUC, NTFISS, NPAR, EVTOJ, REACEV, DNNEW, DNOLD, SSF,
877:             & SSC, SSN2N, WRKMAT, IDBG, NOUT2, LCHA, NCHA, MTBURN,
878:             & NUCLP, GAM, GAMI, NBIC, PBIC, KSTP, LONG, LL, IP, KP,
879:             & NPN, PHAI, IBC(2), ST235, PERIOD(J79), IDU235, FACPOW,
880:             & POWZN1, POWZN2, NUCL, NCH, RAMDA, FACT2N, IBC ,
881:             & KPBURN, KZEROP, BZERO )
882: C
883:         else if ( IBNTYP.le.3 ) then
884:           call BURNCX( PHIP, VOLM, DELT, NMAT, NBOPT, NOUT1, ITD, MXNUC,
885:             & NTNUC, NTFISS, NPAR, EVTOJ, REACEV, DNNEW, DNOLD, SSF,
886:             & SSC, SSN2N, WRKMAT, IDBG, NOUT2, LCHA, NCHA, MTBURN,
887:             & NUCLP, GAM, GAMI, NBIC, PBIC, KSTP, LONG, LL, IP, KP,
888:             & NPN, PHAI, IBC(2), ST235, PERIOD(J79), IDU235, FACPOW,
889:             & POWZN1, POWZN2, NUCL, NCH, RAMDA, FACT2N, IBC ,
890:             & KPBURN, KZEROP, BZERO )
891: C
892:         else if ( IBNTYP.eq.4 ) then
893:           call BURN4( PHIP, VOLM, DELT, NMAT, NBOPT, NOUT1, ITD, MXNUC,
894:             & NTNUC, NTFISS, NPAR, EVTOJ, REACEV, DNNEW, DNOLD, SSF,
895:             & SSC, SSN2N, WRKMAT, IDBG, NOUT2, LCHA, NCHA, MTBURN,
896:             & NUCLP, GAM, GAMI, NBIC, PBIC, KSTP, LONG, LL, IP, KP,
897:             & NPN, PHAI, IBC(2), ST235, PERIOD(J79), IDU235, FACPOW,
898:             & POWZN1, POWZN2, NUCL, NCH, RAMDA, FACT2N, IBC ,
899:             & KPBURN, KZEROP, BZERO, MATREG )
900: C
901:         end if
902: C
903:       end if
904: C2005 ... end
905: C
906:       FACNRM(J79) = FACPOW
907: C
908:       if ( IBEDIT.gt.1 ) then
909:         write(NOUT1,*) ' **J79 DELDAY DELT FACPOW:', J79, DELDAY,
910:           & DELT, FACPOW

```

src/mvpburn/burnup.f

```

911:      end if
912: C
913:      DAYS(K79) = DAYS(J79) + DELT
914: C
915:      SUMMWD = 0.0
916:      do 320 M = 1, NMAT
917:          SUMMWD = SUMMWD + WRKMAT(M)
918:      320 continue
919: C
920:      CUMMWD(K79) = CUMMWD(J79) + SUMMWD*1.0000E-6
921:      if ( TWTHVY.gt.0.0 ) EXPST(K79) = CUMMWD(K79) /TWTHVY
922: C
923:      do 340 M = 1, NMAT
924:          DMWZON(M,K79) = DMWZON(M,J79) + WRKMAT(M)*1.00000E-6
925:          SAVEWT = HMINV(M,1)
926:          if ( SAVEWT.le.0.0 ) SAVEWT = 1.00000
927:          EXPSSZ(M,K79) = DMWZON(M,K79) /SAVEWT/VOLM(M)
928:          do 330 I = 1, NTNUC
929:              if ( DNNEW(I,M).le.BZERO ) DNNEW(I,M) = BZERO
930:              DENTAB(I,M,K79) = DNNEW(I,M)
931:          330 continue
932:      340 continue
933: C
934:      end if
935: C
936:      if ( IVOID.eq.1 ) then
937:          do 360 M = 1, NMAT
938:              do 350 I = 1, NTNUC
939:                  DNNEW(I,M) = DENTAB(I,M,K79)
940:              350 continue
941:          360 continue
942:      end if
943: C
944: C **** EDIT CONVERSION RATIO & YIELD DATA HERE !!!!!
945: C
946:      if ( JPC.eq.0 .or. ISPC.eq.2 ) then
947: C
948:          U5FNOW = 0.0
949: C
950:          do 380 M = 1, NMAT
951:              WTSAVE = 0.0
952:              if ( IDU235.gt.0 ) then
953:                  U5FNOW = U5FNOW + VOLM(M)*DNNEW(IDU235,M)
954:              end if
955:              do 370 I = 1, NTFISS
956:                  WTSAVE = WTSAVE + AMASS(I)*DNNEW(I,M)
957:              370 continue
958:              SAVE = WTSAVE/ABOGA
959:              HMINV(M,K79) = SAVE*1.000E-6
960:              if ( IBEDIT.gt.2 ) write(NOUT1,*) ' M WT(GRAM) : ', M, SAVE
961:          380 continue
962:          if ( IBEDIT.gt.2 ) write(NOUT1,*) ' U5FNOW      : ', U5FNOW
963: C
964:          U5FNOW = U5FNOW*1.000E+24
965:          if ( ST235.gt.0.0 ) U5FNOW = (ST235-U5FNOW)*100.0/ST235
966:          if ( IBEDIT.gt.2 ) write(NOUT1,*) ' U5FNOW ST235      : ',
967:              & U5FNOW, ST235
968: C2005 ... added
969:          if( ABS(U5FNOW).lt.1.0E-5 ) U5FNOW = 0.0
970: C2005 ... end
971:          U235F(K79) = U5FNOW
972: C00/01/21 Modify Calculate ISPC=1 for PC-Method
973:      end if
974:      if ( ISPC.eq.1 ) then
975: C

```

```

976: C **** SET FISSILE ABSORPTION RATE
977: C
978:      FISABS(J79) = 0.0
979:      FDECAY(J79) = 0.0
980:      ESAVE = 0.0
981:      do 420 M = 1, NMAT
982:          RSAVE = 0.0
983:          do 410 ISO = 1, NFIS
984:              SAVE = 0.0
985:              do 390 J = 1, NTNUC
986:                  C2003 if ( NAMFIS(ISO)(1:3).eq.SRACID(J)(2:4) ) then
987:                      if ( NAMFIS(ISO)/10.eq.NCODE(J) ) then
988:                          JPOS = J
989:                          go to 400
990:                      end if
991:                  390 continue
992:                  go to 410
993:                  400 continue
994:                  DNTEMP = DNOLD(JPOS,M)
995:                  MT = 0
996:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'F' ) MT = 1
997:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'C' ) MT = 2
998:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'A' ) MT = 4
999:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'P' ) MT = 0
1000:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'N' ) MT = 3
1001:                  C2003 if ( NAMFIS(ISO)(4:4).eq.'D' ) MT = 5
1002:                  MT = mod(NAMFIS(ISO),10)
1003:                  C2003 ... end
1004:                  if ( MT.eq.0 ) go to 410
1005: C
1006:                  if ( MT.eq.5 ) then
1007:                      FDECAY(J79) = FDECAY(J79) + 1.00E+24*RAMDA(JPOS)*
1008:                          & DNTEMP*VOLM(M)
1009:                      go to 410
1010:                  end if
1011: C
1012:                  if ( IFISDN(ISO).eq.0 ) then
1013:                      RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FISFCT(ISO)
1014:                      SAVE = RATMIC(2,MT,JPOS,M,J79)*FISFCT(ISO)
1015: C
1016:                  else
1017:                      RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FISFCT(ISO)*
1018:                          & DNTEMP
1019:                      SAVE = RATMIC(2,MT,JPOS,M,J79)*FISFCT(ISO)*DNTEMP
1020:                  end if
1021: C
1022:                  ESAVE = ESAVE + SAVE*SAVE*VOLM(M)*VOLM(M)
1023: C
1024:                  410 continue
1025:                  FISABS(J79) = FISABS(J79) + RSAVE*VOLM(M)
1026: C
1027:                  if ( IBEDIT.gt.2 ) then
1028:                      write(NOUT1,*) '** M FISABS(J79) RSAVE ESAVE VOLM(M) : ',
1029:                          & M, FISABS(J79), RSAVE, ESAVE, VOLM(M)
1030:                  end if
1031: C
1032:                  420 continue
1033:                  EFISAB(J79) = 0.0
1034:                  if ( ESAVE.gt.0.0 ) EFISAB(J79) = SQRT(ESAVE)
1035: C
1036: C **** SET FERTILE CAPTURE RATE
1037: C
1038:      CDECAY(J79) = 0.0
1039:      FRTCAP(J79) = 0.0
1040:      ESAVE = 0.0

```

src/mvpburn/burnup.f

```

1041:      do 460 M = 1, NMAT
1042:        RSAVE = 0.0
1043:        do 450 ISO = 1, NFER
1044:          SAVE = 0.0
1045:          do 430 J = 1, NTNUC
1046:            C2003 if ( NAMFER(ISO)(1:3).eq.SRACID(J)(2:4) ) then
1047:              if ( NAMFER(ISO)/10.eq.NCODE(J) ) then
1048:                JPOS = J
1049:                go to 440
1050:              end if
1051:            430 continue
1052:            go to 450
1053:          440 continue
1054:          DNTEMP = DNOLD(JPOS,M)
1055:          MT = 0
1056:          C2003 if ( NAMFER(ISO)(4:4).eq.'F' ) MT = 1
1057:          C2003 if ( NAMFER(ISO)(4:4).eq.'C' ) MT = 2
1058:          C2003 if ( NAMFER(ISO)(4:4).eq.'A' ) MT = 4
1059:          C2003 if ( NAMFER(ISO)(4:4).eq.'P' ) MT = 0
1060:          C2003 if ( NAMFER(ISO)(4:4).eq.'N' ) MT = 3
1061:          C2003 if ( NAMFER(ISO)(4:4).eq.'D' ) MT = 5
1062:          MT = mod(NAMFER(ISO),10)
1063:          C2003 ... end
1064:          if ( MT.eq.0 ) go to 450
1065:          C
1066:          if ( MT.eq.5 ) then
1067:            CDECAY(J79) = CDECAY(J79) + 1.00E+24*RAMBA(JPOS)*
1068:              & DNTEMP*VOLM(M)
1069:            go to 450
1070:          end if
1071:          C
1072:          if ( IFRTDN(ISO).eq.0 ) then
1073:            RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FRTFCT(ISO)
1074:            SAVE = RATMIC(2,MT,JPOS,M,J79)*FRTFCT(ISO)
1075:          C
1076:          else
1077:            RSAVE = RSAVE + RATMIC(1,MT,JPOS,M,J79)*FRTFCT(ISO)*
1078:              & DNTEMP
1079:            SAVE = RATMIC(2,MT,JPOS,M,J79)*FRTFCT(ISO)*DNTEMP
1080:          end if
1081:          C
1082:          450 continue
1083:          C
1084:          FRTCAP(J79) = FRTCAP(J79) + RSAVE*VOLM(M)
1085:          ESAVE = ESAVE + SAVE*SAVE*VOLM(M)*VOLM(M)
1086:          C
1087:          if ( IBEDIT.gt.2 ) then
1088:            write(NOUT1,*) '** M FRTCAP(J79) RSAVE ESAVE VOLM(M) : ',
1089:              & M, FRTCAP(J79), RSAVE, ESAVE, VOLM(M)
1090:          end if
1091:          C
1092:          460 continue
1093:          EFRTCA(J79) = 0.0
1094:          if ( ESAVE.gt.0.0 ) EFRTCA(J79) = SQRT(ESAVE)
1095:          C
1096:          C **** SET CONVERSION RATIO
1097:          C
1098:          ASAVE = FISABS(J79) + FDECAY(J79)
1099:          CSAVE = FRTCAP(J79) + CDECAY(J79)
1100:          ERRABS = 0.0
1101:          ERRCAP = 0.0
1102:          if ( ASAVE.gt.0.0 ) ERRABS = 100.0*EFISAB(J79) /ASAVE
1103:          if ( CSAVE.gt.0.0 ) ERRCAP = 100.0*EFRTCA(J79) /CSAVE
1104:          INSCR(J79) = 0.0
1105:          EINSR(J79) = 0.0

```

```

1106:      if ( ASAVE.gt.0.0 ) then
1107:        INSCR(J79) = CSAVE/ASAVE
1108:        EINSR(J79) = SQRT(ERRABS*ERRABS+ERRCAP*ERRCAP)
1109:      end if
1110:      C
1111:      if ( POWNOW.le.0.0 ) then
1112:        if ( FDECAY(J79).gt.0.0.and.CDECAY(J79).gt.0.0 ) then
1113:          INSCR(J79) = CDECAY(J79) /FDECAY(J79)
1114:          EINSR(J79) = 0.0
1115:        end if
1116:      end if
1117:      C
1118:      C **** CALCULATE AVERAGE GAMMA DATA
1119:      C
1120:      do 480 M = 1, NMAT
1121:        SUM1 = 0.0
1122:        SUM2 = 0.0
1123:        do 470 ISO = 1, NTFISS
1124:          DNTEMP = DNOLD(ISO,M)
1125:          GAMISO = REACEV(1,ISO)
1126:          C2005
1127:          if ( NBOPT.eq.1 ) then
1128:            SUM1 = SUM1 + DNTEMP*RPCMIC(1,1,ISO,M)
1129:            SUM2 = SUM2 + DNTEMP*RPCMIC(1,1,ISO,M)*GAMISO
1130:          C
1131:          else
1132:            SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1133:            SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*GAMISO
1134:          end if
1135:        Cend
1136:        470 continue
1137:        GAMAVG(M,J79) = 0.0
1138:        if ( SUM1.gt.0.0 ) then
1139:          SAVE = SUM2/SUM1
1140:          GAMAVG(M,J79) = SAVE*EVTOJ*1.0000E+6
1141:        end if
1142:        480 continue
1143:      C
1144:      C *** CALCULATE AVERAGE XE-135 YIELD DATA
1145:      C
1146:      if ( IDX35.gt.0 ) then
1147:        do 500 M = 1, NMAT
1148:          YLDXE5(M,J79) = GAM(IDU235,IDX35)
1149:          if ( MTYP(M).ne.1 ) go to 500
1150:          SUM1 = 0.0
1151:          SUM2 = 0.0
1152:          do 490 ISO = 1, NTFISS
1153:            DNTEMP = DNOLD(ISO,M)
1154:            YLDTMP = GAM(ISO,IDX35)
1155:          C2005
1156:          if ( NBOPT.eq.1 ) then
1157:            SUM1 = SUM1 + DNTEMP*RPCMIC(1,1,ISO,M)
1158:            SUM2 = SUM2 + DNTEMP*RPCMIC(1,1,ISO,M)*YLDTMP
1159:          else
1160:            SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1161:            SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1162:          end if
1163:        Cend
1164:        490 continue
1165:        YLDXE5(M,J79) = 0.0
1166:        if ( SUM1.gt.0.0 ) YLDXE5(M,J79) = SUM2/SUM1
1167:        500 continue
1168:      end if
1169:      C
1170:      C *** CALCULATE AVERAGE I-135 YIELD DATA

```

src/mvpburn/burnup.f

```

1171: C
1172:   if ( IDI135.gt.0 ) then
1173:     do 520 M = 1, NMAT
1174:       YLDI35(M,J79) = GAM(IDU235,IDI135)
1175:       if ( MTYP(M).ne.1 ) go to 520
1176:       SUM1 = 0.0
1177:       SUM2 = 0.0
1178:       do 510 ISO = 1, NTFISS
1179:         DNTEMP = DNOLD(ISO,M)
1180:         YLDTMP = GAM(ISO,IDI135)
1181: C2005
1182:       if ( NBOPT.eq.1 ) then
1183:         SUM1 = SUM1 + DNTEMP*RPCMIC(1,1,ISO,M)
1184:         SUM2 = SUM2 + DNTEMP*RPCMIC(1,1,ISO,M)*YLDTMP
1185:       else
1186:         SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1187:         SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1188:       end if
1189: Cend
1190:   510   continue
1191:       YLDI35(M,J79) = 0.0
1192:       if ( SUM1.gt.0.0 ) YLDI35(M,J79) = SUM2/SUM1
1193:   520   continue
1194:   end if
1195: C
1196: C *** CALCULATE AVERAGE SM-149 YIELD DATA
1197: C
1198:   if ( IDSM49.gt.0 ) then
1199:     do 540 M = 1, NMAT
1200:       YLDSM9(M,J79) = GAM(IDU235,IDSM49)
1201:       if ( MTYP(M).ne.1 ) go to 540
1202:       SUM1 = 0.0
1203:       SUM2 = 0.0
1204:       do 530 ISO = 1, NTFISS
1205:         DNTEMP = DNOLD(ISO,M)
1206:         YLDTMP = GAM(ISO,IDSM49)
1207: C2005
1208:       if ( NBOPT.eq.1 ) then
1209:         SUM1 = SUM1 + DNTEMP*RPCMIC(1,1,ISO,M)
1210:         SUM2 = SUM2 + DNTEMP*RPCMIC(1,1,ISO,M)*YLDTMP
1211: C
1212:       else
1213:         SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1214:         SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1215:       end if
1216: Cend
1217:   530   continue
1218:       YLDSM9(M,J79) = 0.0
1219:       if ( SUM1.gt.0.0 ) YLDSM9(M,J79) = SUM2/SUM1
1220:   540   continue
1221:   end if
1222: C
1223: C *** CALCULATE AVERAGE PM-149 YIELD DATA
1224: C
1225:   if ( IDPM49.gt.0 ) then
1226:     do 560 M = 1, NMAT
1227:       YLDPM9(M,J79) = GAM(IDU235,IDPM49)
1228:       if ( MTYP(M).ne.1 ) go to 560
1229:       SUM1 = 0.0
1230:       SUM2 = 0.0
1231:       do 550 ISO = 1, NTFISS
1232:         DNTEMP = DNOLD(ISO,M)
1233:         YLDTMP = GAM(ISO,IDPM49)
1234: C2005
1235:       if ( NBOPT.eq.1 ) then

```

```

1236:         SUM1 = SUM1 + DNTEMP*RPCMIC(1,1,ISO,M)
1237:         SUM2 = SUM2 + DNTEMP*RPCMIC(1,1,ISO,M)*YLDTMP
1238:       else
1239:         SUM1 = SUM1 + DNTEMP*RATMIC(1,1,ISO,M,J79)
1240:         SUM2 = SUM2 + DNTEMP*RATMIC(1,1,ISO,M,J79)*YLDTMP
1241:       end if
1242: Cend
1243:   550   continue
1244:       YLDPM9(M,J79) = 0.0
1245:       if ( SUM1.gt.0.0 ) YLDPM9(M,J79) = SUM2/SUM1
1246:   560   continue
1247:   end if
1248: C
1249:   end if
1250: C
1251:   call CPUTM2( T2 )
1252: C
1253:   TT = T2 - T1
1254:   if ( IBEDIT.gt.0 ) then
1255:     write(NOUT1,7080) J79, TT
1256:   end if
1257: C
1258: C ***** UPDATE NOWSTP FOR BURNUP STEP NUMBER
1259: C
1260:   NOWSTP0 = NOWSTP
1261:   if ( JPC.eq.0 .or. ISPC.eq.2 ) then
1262: C BURNXT EXPAND THIS ROUTINE
1263: C   call BURNXT( J79, NMAT, NTNNUC, IBEND, NEP, NOWSTP, NOUT2,
1264: C   &          DNNEW, TRGNAM, SRACID )
1265: C
1266: C
1267:   if ( IBEDIT.gt.0 ) then
1268:     do 115 M = 1, NMAT
1269:       write(NOUT1,7200) M, TRGNAM(M)
1270: C2003 ... use 6 byte nuclide ID
1271: CC   write(NOUT1,7220) (SRACID(N),DNNEW(N,M),N=1,NTNUC)
1272: C2003+ NCODE(N) is not defined. What shall we do?
1273: CKSK   call ZZMM(1,1,NCODE(N),CWRK(1:6))
1274: CKSK   write(NOUT1,7220) (CWRK(1:6),DNNEW(N,M),N=1,NTNUC)
1275:       do 114 N=1, NTNUC
1276:         call ZZMM(1,1,NCODE(N),CWRK2(N)(1:6))
1277:       continue
1278:       write(NOUT1,7220) (CWRK2(N)(1:6),DNNEW(N,M),N=1,NTNUC)
1279: C2003 ... end
1280:   115   continue
1281:   end if
1282: C
1283:   7200   format(/1X,4X,'MATERIAL NO.=' ,I3,' & MATERIAL NAME = ',A12)
1284: C2003
1285: C7220   format(/8(1X,A4,1P,E11.4))
1286:   7220   format(/8(1X,A6,1P,E11.4))
1287: C
1288: C ***** UPDATE I79 FOR BURNUP STEP NUMBER
1289: C
1290:   if ( J79.ge.NEP ) IBEND = 1
1291: C
1292:   NOWSTP = NOWSTP + 1
1293: C
1294:   end if
1295: C
1296: C ---write PDS
1297: C
1298:   call RWCOM1( 'WRITE', DIROUT, CASEID )
1299:   call RWCOM2( 'WRITE', DIROUT, CASEID )
1300:   call RWCOM3( 'WRITE', DIROUT, CASEID )

```

src/mvpburn/burnup.f

```

1301:      call RWREST( 'WRITE', DIROUT, CASEID )
1302:      call RWCHAN( 'WRITE', DIROUT, CASEID, NPAR, NTNUC, NTFISS,
1303:      &          NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
1304:
1305: C
1306: C   ... write PDS member CASE+'MATD'
1307: C
1308:      call RWMATD( 'WRITE', DIROUT, CASEID, NMAT, KNMAX, NISO(1),
1309:      &          TEMP(1), VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1),
1310:      &          MTCARD(1,1), IDENT(1,1), DNINIT(1,1), MATMVP(1),
1311: C2005&          IPBURN(1,1) )
1312:      &          IPBURN(1,1), MATREG(1) )
1313: C
1314: CKSK... append data to CASE+'HT'+SID1
1315: C   ... append data to CASE+'HT'+SID0
1316: C
1317: CKSK MEMBER = CASEID/'HT'/'SID1
1318: MEMBER = CASEID/'HT'/'SID0
1319: C2003 ... check member size
1320: call PDSIO( 'SEARCH', DIROUT, MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
1321: if (LENG.gt.MEMORY) then
1322:   write(NOUT1,*) 'XXX(BURNUP) Size of PDS member ', MEMBER,
1323:   &   ' (=,LENG,') exceeds buffer size (=,MEMORY,')
1324:   stop 999
1325: end if
1326: C2003 ... end
1327: call PDSIO( 'READ', DIROUT, MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
1328: call PACKRI( IARRAY, MEMBER, MEMORY, IRET )
1329: if ( IRET.ne.0 ) then
1330:   write(NOUT1,6888)
1331:   write(NOUT1,*) ' failed to re-initialize data of
1332:   &   '<', MEMBER, '> to append data.'
1333:   stop 999
1334: end if
1335: C
1336:      LOPl = NOWSTP0
1337:
1338: C00/01/20 Mod write ISPC=1 for PC-Method
1339: C   if (JPC.eq.0 .or. ISPC.eq.2) then
1340:   if (ISPC.eq.1) then
1341:
1342:      call PACKLB( IARRAY, 'power/zone', 'POWRZN', IRET )
1343:      call PACKND( IARRAY, 'POWRZN', 'R4', POWRZN(1,LOPl),NMAT,IRET1)
1344:      if ( IRET1.ne.0 ) go to 580
1345: C
1346:      call PACKLB( IARRAY, 'GAMAVG', 'GAMAVG', IRET )
1347:      call PACKND( IARRAY, 'GAMAVG', 'R4', GAMAVG(1,LOPl),NMAT,IRET1)
1348:      if ( IRET1.ne.0 ) go to 580
1349:      call PACKLB( IARRAY, 'YLDXE5', 'YLDXE5', IRET )
1350:      call PACKND( IARRAY, 'YLDXE5', 'R4', YLDXE5(1,LOPl),NMAT,IRET1)
1351:      if ( IRET1.ne.0 ) go to 580
1352:      call PACKLB( IARRAY, 'YLDI35', 'YLDI35', IRET )
1353:      call PACKND( IARRAY, 'YLDI35', 'R4', YLDI35(1,LOPl),NMAT,IRET1)
1354:      if ( IRET1.ne.0 ) go to 580
1355:      call PACKLB( IARRAY, 'YLDPM9', 'YLDPM9', IRET )
1356:      call PACKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOPl),NMAT,IRET1)
1357:      if ( IRET1.ne.0 ) go to 580
1358:      call PACKLB( IARRAY, 'YLDPM9', 'YLDPM9', IRET )
1359:      call PACKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOPl),NMAT,IRET1)
1360:      if ( IRET1.ne.0 ) go to 580
1361: C00/01/20 Del
1362: C   end if
1363: C
1364: C00/01/20 Del
1365: C   if (ISPC.eq.1) then
1366:      call PACKLB( IARRAY, 'RATMIC', 'RATMIC', IRET )
1367: C
1368:      do 570 M = 1, NMAT
1369: C2005 ... modified
1370: Cmod      call PACKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOPl),2*4*
1371: Cmod &      NTNUC, IRET1 )
1372: C
1373:      if ( NBOPT.eq.1 ) then
1374:      call PACKND( IARRAY, 'RATMIC', 'R4', RPCMIC(1,1,1,M),
1375:      &      2*4*NTNUC, IRET1 )
1376:      else
1377:      call PACKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOPl),
1378:      &      2*4*NTNUC, IRET1 )
1379:      end if
1380: C2005 ... end
1381: 570 continue
1382: C00/01/20 Del
1383: C   else
1384:
1385: C   call PCTLB ( IARRAY, 'RATMIC', 'RATMIC', IRET )
1386: C   call UNPKPT( IARRAY, 'RATMIC', 'R4', LP, NDATA, IRET )
1387: C   if (IRET.ne.0) go to 580
1388: C   if (NDATA.ne.2*4*NTNUC) go to 580
1389:
1390: C   do 590 M = 1, NMAT
1391: C   call PACKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOPl), 2*4*
1392: C   &      NTNUC, IRET1 )
1393: C   call MEMOVE( RATMIC(1,1,1,M,LOPl), RARRAY(LP), 2*4*NTNUC)
1394: C 590 continue
1395: C
1396:   end if
1397: C
1398:   call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
1399: C
1400: C
1401: C   ... modify packet
1402: C
1403:   call PCTLB( IARRAY, MEMBER, 'INTEGER', IRET )
1404:   call UNPKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
1405:   if ( IRET1.ne.0 ) go to 580
1406: C
1407:   IARRAY(LP) = NHIST(LOPl)
1408:   IARRAY(LP+1) = NBATCH(LOPl)
1409: C
1410:   call PCTLB( IARRAY, MEMBER, 'REAL', IRET )
1411: C2005 NPK = 20
1412: NPK = 30
1413: C
1414:   call UNPKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
1415:   if ( IRET1.ne.0 ) go to 580
1416: C
1417:   RARRAY(LP) = AKEFF(LOPl)
1418:   RARRAY(LP+1) = ERRKEF(LOPl)
1419:   RARRAY(LP+2) = DAYS(LOPl)
1420:   RARRAY(LP+3) = U235F(LOPl)
1421:   RARRAY(LP+4) = EXPST(LOPl)
1422:   RARRAY(LP+5) = CUMMWD(LOPl)
1423:   RARRAY(LP+6) = INSCR(LOPl)
1424:   RARRAY(LP+7) = INTCR(LOPl)
1425:   RARRAY(LP+8) = EINSR(LOPl)
1426:   RARRAY(LP+9) = EINTCR(LOPl)
1427:   RARRAY(LP+10) = FLXNRM(LOPl)
1428:   RARRAY(LP+11) = FACNRM(LOPl)
1429:   RARRAY(LP+12) = FISABS(LOPl)
1430:   RARRAY(LP+13) = FRTCAP(LOPl)

```

src/mvpburn/burnup.f

```

1431:      RARRAY(LP+14) = EFISAB(LOP1)
1432:      RARRAY(LP+15) = EFRTCA(LOP1)
1433:      RARRAY(LP+16) = FDECAY(LOP1)
1434:      RARRAY(LP+17) = CDECAY(LOP1)
1435: C2005 ... added
1436:      RARRAY(LP+18) = ERRNRM(LOP1)
1437:      RARRAY(LP+19) = POWERW(LOP1)
1438:      RARRAY(LP+20) = POWERE(LOP1)
1439:      RARRAY(LP+21) = RMONIT(LOP1)
1440:      RARRAY(LP+22) = EMONIT(LOP1)
1441: C2005 ... end
1442: C
1443: C ... overwrite PDS member CASE+'HT'+step-ID with data appended
1444: C
1445: C call PDSIO( 'WRITE', DIROUT, MEMBER, MEMBER, RARRAY, LENG, NOUT1 )
1446: C
1447: Cdel if ( IVOID.eq.0 ) then
1448: C
1449: C **** write new member CASE+'HT'+step ****
1450: C
1451: C MEMBER = CASEID//'HT'//SID2
1452: C
1453: C --- STORE PRESENT BURNUP STEP NUMBER
1454: C IARRAY(LENGHT+1) = NOWSTP
1455: C --- HMINV FOR NOWSTP-TH STEP
1456: C JSW = LENGHT + 2
1457: C
1458: C LOP = NOWSTP0 + 1
1459: C
1460: C2005 ... added
1461: C
1462: C if( IVOID.eq.1.and.IBSTEP(LOP).eq.2) then
1463: C write(NOUT1,*) ' ** ',MEMBER,' RECORD OUTPUT is SKIPPED !!! '
1464: C
1465: C else
1466: C
1467: C ... Contents of CASE+'HT'+step are packed by "PACKET" series of
1468: C routines.
1469: C
1470: C call PACK00( IARRAY, MEMBER, MEMORY, IRET )
1471: C
1472: C .. pack STEP#, NMAT & NTNUC
1473: C
1474: C call PACKND( IARRAY, 'STEP#', 'I4', NOWSTP, 1, IRET )
1475: C call PACKND( IARRAY, 'NMAT', 'I4', NMAT, 1, IRET )
1476: C call PACKND( IARRAY, 'NTNUC', 'I4', NTNUC, 1, IRET )
1477: C
1478: C call PACKLB( IARRAY, 'HMINV', 'HMINV', IRET )
1479: C call PACKND( IARRAY, 'HMINV', 'R4', HMINV(1,LOP), NMAT, IRET1 )
1480: C if ( IRET1.ne.0 ) go to 580
1481: C
1482: C call PACKLB( IARRAY, 'EXPSZN', 'EXPSZN', IRET )
1483: C call PACKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1,LOP), NMAT, IRET1 )
1484: C &
1485: C if ( IRET1.ne.0 ) go to 580
1486: C
1487: C call PACKLB( IARRAY, 'DMWZON', 'DMWZON', IRET )
1488: C call PACKND( IARRAY, 'DMWZON', 'R4', DMWZON(1,LOP), NMAT, IRET1 )
1489: C &
1490: C if ( IRET1.ne.0 ) go to 580
1491: C
1492: C call PACKLB( IARRAY, 'DNBURN', 'DNBURN', IRET )
1493: C call PACKND( IARRAY, 'DNBURN', 'R4', DNNEW(1,1), NTNUC*NMAT,
1494: C & IRET1 )
1495: C if ( IRET1.ne.0 ) go to 580
1496: C
1497: C call PACKLB( IARRAY, MEMBER, 'INTEGER', IRET )
1498: C NPK = 10
1499: C call PACKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
1500: C if ( IRET1.ne.0 ) go to 580
1501: C
1502: C call ICLEA( IARRAY(LP), NPK, 0 )
1503: C
1504: C2005 ... deleted
1505: Cdel IARRAY(LP) = NHIST(LOP)
1506: Cdel IARRAY(LP+1) = NBATCH(LOP)
1507: C2005 ... end
1508: C
1509: C call PACKLB( IARRAY, MEMBER, 'REAL', IRET )
1510: C NPK = 20
1511: C NPK = 30
1512: C
1513: C call PACKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
1514: C if ( IRET1.ne.0 ) go to 580
1515: C
1516: C call CLEA( RARRAY(LP), NPK, 0.0 )
1517: C
1518: C RARRAY(LP) = AKEFF(LOP)
1519: C RARRAY(LP+1) = ERRKEF(LOP)
1520: C RARRAY(LP+2) = DAYS(LOP)
1521: C RARRAY(LP+3) = U235F(LOP)
1522: C RARRAY(LP+4) = EXPST(LOP)
1523: C RARRAY(LP+5) = CUMMWD(LOP)
1524: C RARRAY(LP+6) = INSCR(LOP)
1525: C RARRAY(LP+7) = INTCR(LOP)
1526: C RARRAY(LP+8) = EINSR(LOP)
1527: C RARRAY(LP+9) = EINTCR(LOP)
1528: C RARRAY(LP+10) = FLXNRM(LOP)
1529: C RARRAY(LP+11) = FACNRM(LOP)
1530: C RARRAY(LP+12) = FISABS(LOP)
1531: C RARRAY(LP+13) = FRTCAP(LOP)
1532: C RARRAY(LP+14) = EFISAB(LOP)
1533: C RARRAY(LP+15) = EFRTCA(LOP)
1534: C RARRAY(LP+16) = FDECAY(LOP)
1535: C RARRAY(LP+17) = CDECAY(LOP)
1536: C2005 ... added
1537: C RARRAY(LP+18) = ERRNRM(LOP)
1538: C RARRAY(LP+19) = POWERW(LOP)
1539: C RARRAY(LP+20) = POWERE(LOP)
1540: C RARRAY(LP+21) = RMONIT(LOP)
1541: Cmod RARRAY(LP+21) = RMONIT(LOP)
1542: Cmod RARRAY(LP+22) = EMONIT(LOP)
1543: C RARRAY(LP+21) = 0.0
1544: C RARRAY(LP+22) = 0.0
1545: C2005 ... end
1546: C
1547: C call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
1548: C
1549: C .... Write PDS member CASE+'HT'+step-ID
1550: C
1551: C call PDSIO( 'WRITE', DIROUT, MEMBER, MEMBER, RARRAY, LENG,
1552: C & NOUT1 )
1553: C
1554: C2005 ... added
1555: C end if
1556: C2005 ... end
1557: C
1558: C --- CLOSE PDS
1559: C
1560: C MEMBER = 'CLOSING'

```

src/mvpburn/burnup.f

```
1561: C
1562:      call PDSIO( 'CLOSE', DIROUT, MEMBER, MEMBER, RARRAY, 0, NOUT1 )
1563: C
1564: C *** END OF PROCESS
1565: C
1566:      return
1567: C
1568:      580 write(NOUT1,6888)
1569:          write(NOUT1,*) ' error in making data packet as ',
1570:              &          ' contents of member <', MEMBER, '> code =', IRET
1571:          stop 999
1572: C
1573:      6888 format(///' XXX(BURNUP) ERROR-STOP :' )
1574:
1575:      end
```

src/mvpburn/burnvd.f

```

1:      subroutine BURNVD( NEPVD, IBSTEP, DIRIN, RARRAY,IARRAY,MEMORY,
2: C2005&          DNBURN,NTNUCX,NOUT1 )
3:      &          WTHVY0,NTNUCX,NOUT1 )
4: C=====
5:      character*(*) DIRIN
6: C
7:      integer IARRAY(MEMORY)
8: C
9:      real      RARRAY(MEMORY)
10: C2005 real      DNBURN(NTNUCX,NMAT)
11:      real      WTHVY0(NMAT)
12: C
13:      include 'INC/_BURNP'
14: C
15:      include 'INC/_CBURN1'
16:      include 'INC/_CBURN2'
17:      include 'INC/_CBURN3'
18: C
19:      real INSCR, INTCR
20:      common /MEMOCK/  MEMSED, MEMMAX
21:      common /BNREST/  RSDUMY(3),      NOWSTP, IBEND,  AKEFF(MXSTEP),
22:      &      ERRKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
23:      &      EXPST(MXSTEP),  CUMMWD(MXSTEP),  INSCR(MXSTEP),
24:      &      INTCR(MXSTEP),  EINSR(MXSTEP),  EINTCR(MXSTEP),
25:      &      FLXNRM(MXSTEP),  PACNRM(MXSTEP),  EFISABS(MXSTEP),
26:      &      FRTCAP(MXSTEP),  EFISAB(MXSTEP),  EFRTCA(MXSTEP),
27:      &      FDECAY(MXSTEP),  CDECAY(MXSTEP),  NHIST(MXSTEP),
28:      &      NBATCH(MXSTEP)
29: C2005 ... added
30:      &      ,ERRNRM(MXSTEP),  POWERW(MXSTEP),  POWERE(MXSTEP),
31:      &      RMONIT(MXSTEP),  EMONIT(MXSTEP)
32: C
33:      integer IBSTEP(MXSTEP)
34: C2005 ... end
35: C
36: C ---- external function -----
37: C
38:      character*2 STEPNM
39:      external STEPNM
40: C
41: C ---- local variables -----
42: C
43:      character*8 MEMBER
44:      character*32 TAG
45: C2005 ... added
46:      integer      KBC(20)
47: C2005 ... end
48: C
49: C-----
50: C
51:      MEMBER = 'OPENING'
52:      call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, RARRAY, 1, NOUT1 )
53: C
54: C
55:      ISTEP = 0
56: 100 ISTEP = ISTEP + 1
57:      MEMBER = CASBRN//'HT'//STEPNM(ISTEP,0)
58:      call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, RARRAY, LENG, NOUT1
59:      &          )
60:      if ( LENG.lt.0 ) then
61:          ISTEP = ISTEP - 1
62:          go to 110
63:      end if
64: C
65:      go to 100

```

```

66: C
67: 110 continue
68: C
69:      if ( ISTEP.eq.0 ) then
70:          write(NOUT1,6888)
71:          write(NOUT1,*) ' case <', CASBRN, '> has not burnup history',
72:          &          ' member ('',MEMBER,'') in PDS file'
73:          stop 999
74:      end if
75: C
76:      NEPVD = ISTEP
77: C
78:      if ( NEPVD.lt.NEP1 ) then
79:          write(NOUT1,6666)
80:          write(NOUT1,*)
81:          &          ' number of output burnup steps in reference case : ',NEPVD
82:          write(NOUT1,*)
83:          &          ' reference case has not finished until last step'
84:      end if
85: C
86: C ***** input POWERL from CASE+'COM1' *****
87: C
88:      MEMBER = CASBRN//'COM1'
89: C
90:      call PDSFIL( 'READ', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
91: C2005 ... added
92:      call SRTAG2(62,'IBC',IRC)
93:      read(62) TAG,KBC
94: C
95:      IBC( 1) = KBC( 1)
96:      IBC( 2) = KBC( 2)
97:      IBC( 5) = KBC( 5)
98:      IBC( 6) = KBC( 6)
99: C
100:      IBC( 9) = KBC( 9)
101:      IBC(10) = KBC(10)
102:      IBC(11) = KBC(11)
103:      IBC(19) = KBC(19)
104: C2005 ... end
105:      call SRTAG2(62,'PERIOD',IRC)
106:      read(62) TAG,(PERIOD(KK),KK=1,MXSTEP)
107: C
108:      call SRTAG2(62,'POWERL',IRC)
109:      read(62) TAG,(POWERL(KK),KK=1,MXSTEP)
110: C2005 ... added
111:      call SRTAG2(62,'EXTSRC',IRC)
112:      read(62) TAG,(EXTSRC(KK),KK=1,MXSTEP)
113: C
114:      call SRTAG2(62,'VMONIT',IRC)
115:      read(62) TAG,IDMONI,(VMONIT(KK),KK=1,MXSTEP)
116: C
117:      call SRTAG2(62,'REGPOW',IRC)
118:      read(62) TAG,(REGPOW(KK),KK=1,MXSTEP)
119: C2005 ... end
120:      call PDSFIL( 'CLOSE', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
121: C
122: C ***** input DAYS EXPST U235F CUMMWD from CASE+"HT"+ step *****
123: C
124:      write(NOUT1,*) ' ---- READING INVENTORY DATA FROM ',CASBRN,
125:      &          'HT## FROM STEP 1 TO ',NEPVD
126: C
127:      do 120 ISTEP = 1, NEPVD
128: C
129:          MEMBER = CASBRN//'HT'//STEPNM(ISTEP,0)
130: C

```


src/mvpburn/burnvd.f

```

131: C2003 ... check member size
132: C
133:     call PDSIO( 'SEARCH',DIRIN,MEMBER, MEMBER, RARRAY, LENG, NOUT1
134:     &
135: C
136:     if (LENG.gt.MEMORY) then
137:         write(NOUT1,*) 'XXX(BURNVD) Size of PDS member ',MEMBER,
138:     & ' (=,LENG,) exceeds buffer size (=,MEMORY,)'
139:         stop 999
140:     else
141:         MTMP = MEMMAX - MEMORY + LENG
142:         if (MTMP.gt.MEUSED) MEUSED = MTMP
143:     end if
144: C2003 ... end
145:     call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, RARRAY, LENG, NOUT1
146:     &
147: C
148:     call PCTRW( IARRAY, MEMBER, IRET )
149:     call UNPKND( IARRAY, 'STEP#', 'I4', ISWSTP, 1, NDATA2, IRET )
150:     call UNPKND( IARRAY, 'NMAT', 'I4', NMAT0, 1, NDATA3, IRET )
151:     call UNPKND( IARRAY, 'NTNUC', 'I4', NTNUC0, 1, NDATA2, IRET )
152: C
153:     if ( NTNUC0.ne.NTNUC ) then
154:         write(NOUT1,6888)
155:         write(NOUT1,*)
156:     & ' number of nuclides in reference case(=,NTNUC0,
157:     & ' ) is different from requested one(=,NTNUC,)'
158:         write(NOUT1,*) ' member <', MEMBER, '>'
159:         stop 888
160:     end if
161: C
162:     if ( NMAT0.ne.NMAT ) then
163:         write(NOUT1,6888)
164:         write(NOUT1,*)
165:     & ' number of materials in reference case(=,NMAT0,
166:     & ' ) is different from requested one(=,NMAT,)'
167:         write(NOUT1,*) ' member <', MEMBER, '>'
168:         stop 888
169:     end if
170: C2005 ... added to reset WTHVY0 data
171:     if( ISTEP.eq.1 ) then
172:         call PCTLB( IARRAY, MEMBER, 'HMINV', IRET )
173:         call UNPKND( IARRAY, 'HMINV', 'R4', WTHVY0(1),
174:     & NMAT, NDATA2, IRET )
175:     end if
176: C2005 ... end
177: C
178:     call PCTLB( IARRAY, MEMBER, 'REAL', IRET )
179:     call UNPKND( IARRAY, MEMBER, 'R4', LP, NPK, IRET )
180: C
181:     DAYS (ISTEP) = RARRAY(LP+2)
182:     EXPST (ISTEP) = RARRAY(LP+4)
183:     U235F (ISTEP) = RARRAY(LP+3)
184:     CUMMWD(ISTEP) = RARRAY(LP+5)
185: C2005 ... added
186:     FLXNRM(ISTEP) = RARRAY(LP+10)
187:     ERRNRM(ISTEP) = RARRAY(LP+18)
188:     RMONIT(ISTEP) = RARRAY(LP+21)
189:     EMONIT(ISTEP) = RARRAY(LP+22)
190:     120 continue
191: C
192: C ... read STDNUC
193: C
194: C2003 MEMBER = CASBRN//COM3'
195: C

```

```

196:     MEMBER = CASBRN//COM2'
197:     call PDSFIL( 'READ', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
198: C
199:     if ( IERR0.ne.0 ) then
200:         write(NOUT1,6888)
201:         write(NOUT1,*) ' reference case <', CASBRN, '> has',
202:     & ' no caseCOM3 member in PDS file'
203:         write(NOUT1,*) ' member <', MEMBER, '>'
204:         stop 999
205:     end if
206: C2005 added
207:     call SRTAG2(62,'SIZE',IRC)
208:     read(62) TAG,IDUM01,IDUM02,IDUM03,IDUM04,IDUM05,ST235, TWTHVY,
209:     & RDUM01, RDUM02,RDUM03,IDUM06,IDU235, IBUNIT, IDUM08,
210:     & IDUM09, IDUM10,IDUM11,IDUM12,IDUM13, IDXE35, IDI135,
211:     & IDSM49, IDPM49,NFIS, NFER
212: Check
213:     if( IBC(3).ge.2 ) then
214:         write(NOUT1,*) ' *** check write at sub(BURNVD) *** '
215:         write(NOUT1,*) ' > ST235 TWTHVY : ',ST235, TWTHVY
216:         write(NOUT1,*) ' > IDU235 IBUNIT : ',IDU235, IBUNIT
217:         write(NOUT1,*) ' > NFIS NFER : ',NFIS, NFER
218:         write(NOUT1,*) ' > IDXE35 IDI135 : ',IDXE35, IDI135
219:         write(NOUT1,*) ' > IDSM49 IDPM49 : ',IDSM49, IDPM49
220:     end if
221: Cend
222: C
223:     call SRTAG2(62,'IFISDN',IRC)
224:     read(62) TAG,(IFISDN(I),I=1,MXFISS)
225: C
226:     call SRTAG2(62,'FISFCT',IRC)
227:     read(62) TAG,(FISFCT(I),I=1,MXFISS)
228: C
229:     call SRTAG2(62,'IFRTDN',IRC)
230:     read(62) TAG,(IFRTDN(I),I=1,MXFISS)
231: C
232:     call SRTAG2(62,'FRTFCT',IRC)
233:     read(62) TAG,(FRTFCT(I),I=1,MXFISS)
234: C2005 ... end
235:     call SRTAG2(62,'STDNUC',IRC)
236:     read(62) TAG,STDNUC
237: C2005 ... added
238:     call SRTAG2(62,'NAMFIS',IRC)
239:     read(62) TAG,(NAMFIS(I),I=1,MXFISS)
240: C
241:     call SRTAG2(62,'NAMFER',IRC)
242:     read(62) TAG,(NAMFER(I),I=1,MXFISS)
243: C
244:     call SRTAG2(62,'YLDNUC',IRC)
245:     read(62) TAG,NCXE35, NCI135, NCSM49, NCPM49
246: Check
247:     if( IBC(3).ge.2 ) then
248:         write(NOUT1,'(A,5I12)')
249:     & ' > STDNUC NCXE35 NCI135 NCSM49 NCPM49 : ',
250:     & STDNUC,NCXE35,NCI135,NCSM49,NCPM49
251:     end if
252: C2005 end
253: C
254:     call PDSFIL( 'CLOSE', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
255: C
256: C *** END OF PROCESS
257: C
258:     return
259: C
260: 6666 format(/// '!!! (BURNVD) WARNING :' )

```

src/mvpburn/burnvd.f

```
261: 6888 format(// ' XXX(BURNVD) ERROR-STOP : ' )  
262: C  
263:      end
```

SAFE

src/mvpburn/catstr.f

```
1:      subroutine CATSTR( STR,  LSTR,  STR2,  IERR )
2: c==<MVP/GMVP>=====
3: c purpose: concatenate an character string STR2 to a character string
4: c          STR after STR(:LSTR) and return the length of the resulting
5: c          string in LSTR.
6: c=====
7:      character*(*) STR, STR2
8: c
9:      LLS      = LSTR + LEN(STR2)
10:     if ( LLS.gt.LEN(STR) ) then
11:         IERR   = 1
12:         return
13:     end if
14: c
15:     IERR   = 0
16:     STR(LSTR+1:LLS) = STR2
17:     LSTR   = LLS
18:     return
19: end
20:
21:      subroutine CATST2( STR,  LSTR,  STR1,  STR2,  IERR )
22: c==<MVP/GMVP>=====
23: c purpose: concatenate character strings STR1 & STR2 to a character
24: c          string STR after STR(:LSTR) and return the length of the
25: c          resulting string in LSTR.
26: c history: programmed by M.Sasaki  (25 Jun 1994)
27: c=====
28:      character*(*) STR, STR1, STR2
29: c
30:      LLS1     = LSTR + LEN(STR1)
31:      LLS2     = LLS1 + LEN(STR2)
32:      if ( LLS2.gt.LEN(STR) ) then
33:          IERR  = 1
34:          return
35:      end if
36: c
37:      IERR     = 0
38:      STR(LSTR+1:LLS1) = STR1
39:      STR(LLS1+1:LLS2) = STR2
40:      LSTR     = LLS2
41:      return
42: end
```

src/mvpburn/ccomp.f

```
1:      subroutine CCOMP( OUTCH, LN,      INCH, IFL )
2: C
3: C      GMVP/MVP UTILITY
4: C
5: C=====
6: C  PURPOSE:  'COMPRESS' CHARACTER STRING INCH BY REMOVING BLANK
7: C            INTO CHARACTER STRING 'OUTCH' UPTO LN CHARACTERS.
8: C            ('OUTCH' & 'INCH' CAN OVERLAP IN MEMORY BUT
9: C            THE STARTING ADDRESS OF 'OUTCH' MUST PRECEED OR BE SAME AS
10: C           THAT OF 'INCH'. )
11: C
12: C           IFL : FLAG
13: C               0 = NORMAL END
14: C               1 = NUMBER OF NON-BLANK CHARACTERS IN INCH EXCEEDS LN.
15: C
16: C  HISTORY:  PROGRAMMED BY M.SASAKI ( 18 MAR 1992 )
17: C=====
18: C           character*(*) OUTCH, INCH
19: C           character*1 CH1
20: C
21: C           IFL      = 0
22: C           LL       = LEN(INCH)
23: C           NN       = 0
24: C           do 100 I = 1, LL
25: C             if ( INCH(I:I).ne.' ' ) then
26: C               NN      = NN + 1
27: C               if ( NN.le.LN ) then
28: C                 CH1   = INCH(I:I)
29: C                 OUTCH(NN:NN) = CH1
30: C               end if
31: C             end if
32: C           100 continue
33: C           if ( NN.lt.LN ) OUTCH(NN+1:LN) = ' '
34: C           if ( NN.gt.LN ) IFL = 1
35: C           return
36: C           end
37: C
```

src/mvpburn/checkp.f

```
1:      SUBROUTINE CHECKP(ID, NDAT , NDAT0 , NOUT1 , NOUT2 )
2: C
3:      CHARACTER*6      ID
4: C
5:      IF(NDAT.EQ.NDAT0.AND.NDAT.GT.0) RETURN
6: C
7:      WRITE(NOUT1,6888)
8:      WRITE(NOUT1,*) ' unmached parameter length for ',ID,
9:      &              ' in averaging process'
10:     WRITE(NOUT1,*) ' * definision of programming : ',NDAT
11:     WRITE(NOUT1,*) ' * value from file      : ',NDAT0
12:     stop 999
13: C
14: 6888 format(/// ' XXX(CHECKP) ERROR-STOP :' )
15:     end
```

src/mvpburn/chkchn.f

```

1:      subroutine CHKCHN( NUCL,  RAMDA, NCH,   NUCLP, NBIC,  PBIC,  NOL,
2:      &                  KSTP,  LONG,  KK,    IP,    KP,    NPN,  NMAX,
3:      &                  NPAR,  NCHA,  LCHA,  IST,   IEND,  KPBURN,IERR)
4: C
5:      common /REAMIO/  NOUT6,  NDTLS
6: C
7:      integer NUCL(NMAX)
8:      real RAMDA(NMAX)
9:      integer NCH(NMAX), NUCLP(NPAR,NMAX)
10:     integer NBIC(NPAR,NMAX)
11:     real PBIC(NPAR,NMAX)
12:     integer NOL(NMAX), KSTP(NCHA,NMAX), LONG(NCHA,NMAX),
13:     &      KK(NCHA,NMAX), IP(LCHA,NCHA,NMAX), KP(LCHA,NCHA,NMAX),
14:     &      NPN(NPAR,NMAX)
15: C
16:     integer KPBURN(NMAX)
17: C
18: C *** LOOP OF NUCLIDE
19: C
20:     IERR      = 0
21:     if( IEND.lt.IST ) return
22: C
23:     DO 1000 LOP = IST , IEND
24:     IF(KPBURN(LOP).GT.0) then
25:         NOS      = NOL(LOP)
26:         DO 900 JOP = 1 , NOS
27:         LL      = LONG(JOP,LOP) -1
28:         if ( LL.gt.0 ) then
29:             JST      = LL + 1
30:             DO 700  L = 1 , LL
31:             IPNOW     = IP(L,JOP,LOP)
32: Cdel      IP1      = IP(L+1,JOP,LOP)
33: Cdel      KP1      = KP(L+1,JOP,LOP)
34: Cdel      KBIC     = NBIC(KP1,IP1)
35:             if(KPBURN(IPNOW).gt.0.and.L.lt.JST) then
36:                 JST = L
37:             endif
38:             700      CONTINUE
39: C
40: Cdel      write(NOUT6,*) ' ** LOP JOP JST LL : ',LOP,JOP,JST,LL
41: C
42:             if( LL.ge.JST ) then
43:                 DO 800  L = JST , LL
44:                 IPNOW     = IP(L,JOP,LOP)
45:                 if(KPBURN(IPNOW).le.0) then
46: Cdel      write(NOUT6,*) ' ** IPNOW KPBURN(IPNOW) : ',IPNOW,KPBURN(IPNOW)
47:                     KPBURN(IPNOW) = -1
48:                     IERR = IERR + 1
49:                 endif
50:             800      CONTINUE
51:             endif
52:         endif
53:     900      CONTINUE
54:     endif
55: 1000 CONTINUE
56: C
57:     RETURN
58:     END

```

src/mvpburn/chkenv.f

```

1:      subroutine CHKENV( PROG, MVPEX, MVPIDX, PDSIN, PDSOUT, MVPDBG)
2:
3:      C/#!/IF .NOT.NOGETENV
4:
5:      C=<MVP>=====
6:      C Purpose: Check enviromnet variables
7:      C=====
8:      C argument :
9:      C
10:     C i prog : code name ( "MVPBURN" )
11:     C o mvpeX : command name to execute MVP
12:     C o mvpidx : MVP neutron library index file
13:     C o JALL6 : if non zero output all printout on I/O unit 6
14:     C
15:     C=====
16:     C/#!/IF MS_VISUAL
17:     C
18:     C ... This interface block is for CUTIL & MS-Visual tools. ...
19:     C
20:     * interface
21:     * subroutine FUNBUF()
22:     *CDEC$ ATTRIBUTES C :: FUNBUF
23:     * end subroutine
24:     * end interface
25:     C/#!/ENDIF
26:     C
27:     C ... external data ...
28:     C
29:     character*(*) PROG
30:     character*(*) MVPEX
31:     character*(*) MVPIDX
32:     character*(*) PDSIN
33:     character*(*) PDSOUT
34:     C
35:     C ... local data ...
36:     C
37:     character*64 VAR
38:     character*128 STRING
39:     C
40:     C-----
41:     C
42:     NENV = 0
43:     C ==== check MVPUNBUF : set standard output unbufferd
44:     C ("line buffered" saying exactly )
45:     C Other than null or blank should be defined for the variable
46:     C to activate it.
47:     C
48:     STRING = ' '
49:     call ENVGET( 'MVPUNBUF', STRING )
50:     if( STRING.ne.' ' ) then
51:         call FUNBUF()
52:         if ( NENV.le.0 ) write(6,6000)
53:         write(6,*) '== MVPUNBUF : Standard output is line buffered.'
54:         NENV = NENV + 1
55:     end if
56:     C
57:     C-----
58:     C
59:     C ==== check MVPBURN_MVPEX : set name of MVP execution command
60:     C
61:     STRING = ' '
62:     call ENVGET( 'MVPBURN_MVPEX', STRING )
63:     if( STRING.ne.' ' ) then
64:         mvpeX = string
65:         if ( NENV.le.0 ) write(6,6000)

```

```

66:         write(6,*) '== MVPBURN_MVPEX: <',mvpeX(:iclen2(mvpeX)),>'
67:         NENV = NENV + 1
68:     end if
69:     C
70:     C-----
71:     C
72:     C ==== check MVPBURN_NIDX : MVP neutron library index
73:     C
74:     STRING = ' '
75:     call ENVGET( 'MVPBURN_NIDX', STRING )
76:     if( STRING.ne.' ' ) then
77:         mvpidx = string
78:         if ( NENV.le.0 ) write(6,6000)
79:         write(6,*) '== MVPBURN_NIDX : <',mvpidx(:iclen2(mvpidx)),>'
80:         NENV = NENV + 1
81:     end if
82:     C
83:     C-----
84:     C
85:     C ==== check MVPBURN_PDS : default input/output PDS directory
86:     C
87:     STRING = ' '
88:     call ENVGET( 'MVPBURN_PDS', STRING )
89:     if( STRING.ne.' ' ) then
90:         pdsin = string
91:         pdsout = string
92:         if ( NENV.le.0 ) write(6,6000)
93:         write(6,*) '== MVPBURN_PDS : <',pdsin(:iclen2(pdsin)),>'
94:         NENV = NENV + 1
95:     end if
96:     C
97:     C-----
98:     C
99:     C ==== check MVPBURN_DEBUG : Print out of Debug mode &
100:    C PDS File Save Middle Burn-up step
101:    C
102:    STRING = ' '
103:    call ENVGET( 'MVPBURN_DEBUG', STRING )
104:    if( STRING.ne.' ' ) then
105:        MVPDBG = 1
106:        if ( NENV.le.0 ) write(6,6000)
107:        write(6,*) '== MVPBURN_DEBUG: Debug mode is ON.'
108:        NENV = NENV + 1
109:    end if
110:    C
111:    if ( NENV.gt.0 ) write(6,*)
112:    C
113:    C/#!/ENDIF
114:    C
115:    6000 format(2X,'<< ENVIRONMENT VARIABLES >>')
116:    return
117:    end

```

src/mvpburn/chkmid.f

```
1:      subroutine CHKMID( ID,      IANS )
2: C
3: C2003 character*8 ID
4:      character*(*) ID
5: check
6: CCC  write(6,*) 'check CHKMID <',ID,'>'
7: C
8:      IANS      = 1
9: C2003 do 100 I = 5, 8
10:      do 80 K=len(ID),1,-1
11:          if(ID(K:K).ne.' ') goto 90
12:      80 continue
13:      90 continue
14:      do 100 I = K-3, K
15:          ISW      = INDEX('0123456789',ID(I:I))
16:          IANS      = IANS*ISW
17:      100 continue
18: C
19:      return
20:      end
```


src/mvpburn/chkwrđ.f

```
1:      subroutine CHKWRD( LINE,  WORD,  LENG,  IANS )
2:  C
3:      character*1 WORD(LENG)
4:      character*72 LINE
5:  C
6:  C
7:  C
8:      IANS      = 0
9:  C
10:     LAST      = 73 - LENG
11:     do 100 I = 1, LAST
12:         I1      = I
13:         I2      = I + LENG - 1
14:         ISW      = INDEX(LINE(I1:I2),WORD(1)(1:LENG))
15:         if ( ISW.eq.1 ) then
16:             IANS      = 1
17:             return
18:         end if
19:     100 continue
20:     return
21: end
22: subroutine trgchk( line, trgnam, lentrg, ians, first,nout1)
23: C
24:     character*(*) trgnam
25:     character*72 line
26:     integer      lentrg,ians,first
27: C
28:     trgnam = ' '
29:     ians   = 0
30:     lentrg = 0
31:     j1     = 1
32: C
33:     if (first.eq.1) then
34:         do 100 i = 1,67
35:             i2 = i+5
36:             if (line(i:i2).eq.'DEFINE') go to 110
37:     100 continue
38:     write(nout1,6888)
39:     write(nout1,*) ' something wrong : word "DEFINE" was not found '
40:     write(nout1,*) ' LINE:',line
41:     stop 888
42:     110 continue
43:     j1 = i2+1
44:     end if
45:     do 120 i = j1,71
46:         if (line(i:i).eq. '@') go to 130
47:     120 continue
48:     return
49:     130 continue
50:     j1 = i+1
51:     j2 = min(j1+11,72)
52:     trgnam(1:1) = '@'
53:     lentrg = 1
54:     do 140 i = j1,j2
55:         if (line(i:i).eq.' ' .or. line(i:i).eq.'(') go to 150
56:         lentrg = lentrg + 1
57:         if (lentrg.le.12) trgnam(lentrg:lentrg) = line(i:i)
58:     140 continue
59:     write(nout1,6666)
60:     write(nout1,*)
61:     &      ' tally region name is too long'
62:     write(nout1,*)
63:     &      ' first 12 characters are used as tally region name'
64:     write(nout1,*) ' LINE:',line
65:     ians = 1
```

```
66:         lentrg = 12
67:         return
68:     150 continue
69:     ians = 1
70:     return
71: 6666 format(/// ' !!!(CHKWRD) WARNING :' )
72: 6888 format(/// ' XXX(CHKWRD) ERROR-STOP :' )
73:     end
```

src/mvpburn/ckbnup.f

```

1:      subroutine CKBNUP( NOUT1, NOUT2, DIRIN, DIROUT,CASE,  ARRAY,
2:      &                  IARRAY,MEMORY,IBSTEP,ISTART,ISTOP, IERR )
3: C=====
4: C  check number of steps etc. for case CASE.
5: C
6: C  IERR: returns non zero if member does not exist or zero length.
7: C=====
8:      include 'INC/_BURNP'
9: C
10: C  ---- Arguments -----
11: C
12:      character*(*) DIRIN, DIROUT
13:      character*4 CASE
14: C
15:      real          ARRAY(MEMORY)
16:      integer       IARRAY(MEMORY)
17:      integer       IBSTEP(MXSTEP)
18: C
19: C  ---- COMMON area -----
20: C
21: C
22:      include 'INC/_CBURN1'
23:      include 'INC/_CBURN2'
24:      include 'INC/_CBURN3'
25: C
26:      real INSCR, INTCR
27:      common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKKEF(MXSTEP),
28:      & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
29:      & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
30:      & INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
31:      & FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
32:      & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
33:      & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
34:      & NBATCH(MXSTEP)
35: C2005
36:      & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
37:      & RMONIT(MXSTEP), EMONIT(MXSTEP)
38: Cend
39: C
40: C  ---- local data -----
41: C
42:      character*72 MLINE
43: C
44:      real      DUM(50)
45:      integer   IDUM(50)
46: C
47:      character*8 MEMBER
48:      character*8 CWRK
49:      character*32 TAG
50: C
51: C  ---- external function -----
52: C
53:      character*2 STEPNM
54:      external  STEPNM
55: C
56: C  *** Get current step # by checking member CASE+'HT'+step *****
57: C
58:      IERR      = 0
59: C
60:      MEMBER    = 'OPENING'
61:      call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, RARRAY, 1, NOUT1 )
62: C
63:      ISTART    = MXSTEP
64:      ISTOP     = 0
65: C

```

```

66:      do 100 ISTEP = 1, MXSTEP
67:          MEMBER = CASE// 'HT'//STEPNM(ISTEP,0)
68:          call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, RARRAY, LENG,
69:          &          NOUT1 )
70: C
71:          if (IBEDIT .gt. 0) then
72:              if ( LENG.gt.0 ) then
73:                  write(NOUT1,*) ' *** member <', MEMBER, '> exist. length ',
74:                  &          LENG
75:              end if
76:          end if
77: C
78:          if ( IBSTEP(ISTEP).ne.0.and.LENG.lt.0 ) then
79: C
80:              write(NOUT1,*) 'XXX(CKBNUP) WARNING : member <', MEMBER,
81:              &          '> for branch-off',
82:              &          ' calculation dose not exist', ' in PDS <',
83:              &          DIRIN(:ICLEN2(DIRIN)), '>'
84:              IERR      = IERR + 1
85:          end if
86: C
87:          if ( IBSTEP(ISTEP).ne.0 ) then
88:              ISTART    = MIN(ISTEP,ISTART)
89:              ISTOP     = MAX(ISTEP,ISTOP)
90:          end if
91:      100 continue
92: C
93:      if ( IERR.gt.0 ) return
94: C
95: C  ... check number of step of CASE
96: C
97:      MEMBER = CASE// 'COM1'
98:      call PDSFIL( 'READ', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
99: C
100:      if ( IERR0.ne.0 ) then
101:          write(NOUT1,*) 'XXX(CKBNUP) ERROR : failed to open member <',
102:          &          MEMBER, '>',
103:          &          ' reference case of branch-off calculation ',
104:          &          ' in PDS <', DIRIN(:ICLEN2(DIRIN)), '>'
105:          IERR      = IERR + 1
106:          return
107:      end if
108: C
109:      call SRTAG2( 62, 'SIZE', IRC)
110: C
111:      if( IRC.ne.0 ) then
112:          read(62) TAG, NEP, NEP1
113: C
114:      else
115:          write(NOUT1,*) 'XXX(CKBNUP) ERROR : member <', MEMBER, '> of',
116:          &          ' reference case for branch-off calculation ',
117:          &          'has no record tagged as "SIZE".'
118:          IERR      = IERR + 1
119:          return
120:      end if
121: C
122:      CCC read(62) IBC
123:      C read(62)
124:      CCC read(62) NEP, NEP1, LENG2, LENG3, LENG4
125:      C read(62) NEP, NEP1
126:      CCC read(62) (PERIOD(KK),KK=1,MXSTEP)
127:      CCC read(62) (POWERL(KK),KK=1,MXSTEP)
128:      CCC read(62) (NSTP(KK),KK=1,MXSTEP)
129:      C
130:      call PDSFIL( 'CLOSE', DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )

```

src/mvpburn/ckbnup.f

```
131: C  
132: C *** END OF PROCESS  
133: C  
134:      return  
135:      end
```

SAFE

src/mvpburn/ckstep.f

```

1:      subroutine CKSTEP( NOUT1, NOUT2, DIRIN, DIROUT,CASE,  ARRAY,
2:      &                  IARRAY,MEMORY,NNSTEP ,NHALF, IERR )
3: C=====
4: C  check already calculated steps exist or not and get number of steps
5: C  etc. for case CASE.
6: C
7: C  NNSTEP: max step # calculated as CASE (zero if none exists)
8: C2003
9: C  NHALF : .ne.0 when half way step's caseHT## is found probably
10: C      PC calculation is stopped n+1/2'the step.
11: C  IERR: returns non zero if any inconsistency is found.
12: C=====
13: C
14: C ---- Arguments -----
15: C
16: C      character*(*) DIRIN, DIROUT
17: C      character*4 CASE
18: C
19: C      real      ARRAY(MEMORY)
20: C      integer IARRAY(MEMORY)
21: C
22: C ---- COMMON area -----
23: C
24: C      include 'INC/_BURNP'
25: C
26: C      include 'INC/_CBURNCL'
27: C      include 'INC/_CBURNC2'
28: C      include 'INC/_CBURNC3'
29: C
30: C      real INSCR, INTCR
31: C      common /MEMOCK/  MEUSED, MEMMAX
32: C      common /BNREST/  RSDUMY(3),      NOWSTP, IBEND, AKEFF(MXSTEP),
33: C      &      ERKKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
34: C      &      EXPST(MXSTEP),    CUMMWD(MXSTEP), INSCR(MXSTEP),
35: C      &      INTCR(MXSTEP),    EINSR(MXSTEP), EINTCR(MXSTEP),
36: C      &      FLXNRM(MXSTEP),   FACNRM(MXSTEP), FISABS(MXSTEP),
37: C      &      FRTCAP(MXSTEP),   EFISAB(MXSTEP), EFRTCA(MXSTEP),
38: C      &      FDECAY(MXSTEP),   CDECAY(MXSTEP), NHIST(MXSTEP),
39: C      &      NBATCH(MXSTEP)
40: C2005
41: C      &      ,ERRNRM(MXSTEP),  POWERW(MXSTEP), POWERE(MXSTEP),
42: C      &      RMONIT(MXSTEP),  EMONIT(MXSTEP)
43: Cend
44: C
45: C ---- local data -----
46: C
47: C2005 character*72 MLINE
48: C
49: C2005 real      DUM(50)
50: C2005 integer   IDUM(50)
51: C
52: C      character*8 MEMBER
53: C2005 character*8 CWRK
54: C      character*32 TAG
55: C
56: C ---- external function -----
57: C
58: C      character*2 STEPNM
59: C      external STEPNM
60: C
61: C *** Get current step # by checking member CASE+'HT'+step *****
62: C
63: C      IERR = 0
64: C2003
65: C      NHALF = 0

```

```

66: C
67: C      MEMBER = 'OPENING'
68: C      call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, ARRAY, 1, NOUT1 )
69: C
70: C      NNSTEP = 0
71: C
72: C      do 100 ISTEP = 1, MXSTEP
73: C          MEMBER = CASE//'HT'//STEPNM(ISTEP,0)
74: C          call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY, LENG,
75: C      &              NOUT1 )
76: C          if ( LENG.lt.0 ) then
77: C              NNSTEP = ISTEP - 1
78: C              goto 105
79: C          end if
80: C      100 continue
81: C
82: C      NNSTEP = MXSTEP
83: C
84: C      105 continue
85: C
86: C2003 ... check NNSTEP+1/2' step of caseHT##
87: C
88: C      MEMBER = CASE//'HT'//STEPNM(NNSTEP,1)
89: C
90: C      call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY, LENG,
91: C      &              NOUT1 )
92: C
93: C      if ( LENG.gt.0 ) then
94: C          NHALF = 1
95: C      end if
96: C
97: C2003 ... end
98: C
99: C      if( NNSTEP.le.0 ) return
100: C
101: C      write(NOUT1,('( '))
102: C      write(NOUT1,*) ' *** MEMBERS ',CASE,'HT## EXIST UPTO ',NNSTEP,
103: C      &      '-TH STEP. ***'
104: C
105: C      ... check existence of CASE//'REST' and "NOWSTP"
106: C
107: C      MEMBER = CASE//'REST'
108: C
109: C      call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY, LENG,
110: C      &              NOUT1 )
111: C
112: C      .... No CASE//'REST' member case
113: C
114: C
115: C      if ( LENG.lt.0 ) then
116: C          write(NOUT1,*)
117: C      &      '!!!(CKSTEP) WARNING : members ',CASE,'HT## exist but no ',
118: C      &      CASE,'REST member.'
119: C2003
120: C      NHALF = 0
121: C
122: C      ... get DAYS,U235F,CUMMWD for NNSTEP from HT## file
123: C
124: C      MEMBER = CASE//'HT'//STEPNM(NNSTEP,0)
125: C
126: C      write(NOUT1,*)
127: C      &      ' Some infomation get from ',MEMBER,' file.'
128: C
129: C2003 ... check member size
130: C

```

src/mvpburn/ckstep.f

```
131:      call PDSIO( 'SEARCH',DIRIN,MEMBER, MEMBER, IARRAY, LENG,
132:      &          NOUT1 )
133: C
134:      if (LENG.gt.MEMORY) then
135:        write(NOUT1,*) 'XXX(CKSTEP) Size of PDS member ',MEMBER,
136:        &          ' (=',LENG,') exceeds buffer size (=',MEMORY,')'
137:        stop 999
138:      else
139:        MTMP = MEMMAX - MEMORY + LENG
140:        if (MTMP.gt.MEUSED) MEUSED = MTMP
141:      end if
142: C
143: C2003 ... end
144: C
145:      call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, IARRAY, LENG,
146:      &          NOUT1 )
147:      call PCTRW( IARRAY, MEMBER, IRET )
148:      call PCTLB2( IARRAY, MEMBER, 'REAL', IRET )
149: C
150:      if ( IRET.eq.0 ) then
151:        LOP = NNSTEP
152:        call UNPKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
153:        DAYS(LOP) = ARRAY(LP+2)
154:        U235F(LOP) = ARRAY(LP+3)
155:        EXPST(LOP) = ARRAY(LP+4)
156:        CUMMWD(LOP) = ARRAY(LP+5)
157:      end if
158: C2005 ... added to write CASE/'REST' mamber
159: C
160:        NOWSTP = NNSTEP
161:        if(NOWSTP.gt.NEP) IBEND = 1
162:        call RWREST( 'WRITE' , DIRIN , CASE )
163: C2005 ... end
164: C
165: C .... CASE/'REST' exists case
166: C
167:      else
168:        call PDSFIL('READ',DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
169: C
170:        if( IERR0 .eq. 0 ) then
171:          call SRTAG2( 62, 'STEP', IRC)
172:          if( IRC.ne.0 ) then
173:            read(62) TAG, NWSTP0,IBEND0
174: C
175:            if ( NNSTEP.ne.NWSTP0 ) then
176:              write(NOUT1,*)
177:              &          '!!!(CKSTEP) WARNING : a member ',CASE,
178:              &          'REST exists but ',
179:              &          'current step number ',NWSTP0,'(NOWSTP) in the member ',
180:              &          'does not match highest contiguous step number in ',
181:              &          ' members ',CASE,'HT##.'
182:              NNSTEP = min(NNSTEP,NWSTP0)
183:              write(NOUT1,*) '      Use smaller step number ',NNSTEP
184: C2003
185:              NHALF = 0
186:            end if
187: C
188:          end if
189:        end if
190: C
191:        call PDSFIL('CLOSE',DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
192:      end if
193: C
194:      return
195:      end
```

src/mvpburn/ckstp2.f

```

1:      subroutine CKSTP2( NOUT1, NOUT2, DIRIN, DIROUT,CASE,  ARRAY,
2:      C2005&          IARRAY,MEMORY,NNSTEP, IERR )
3:      &              IARRAY,MEMORY,KSTART, IERR ,IBSTEP , ISTOP )
4:      C=====
5:      C  check already calculated steps exist or not and get number of steps
6:      C  etc. for case CASE. For BRANCH mode.
7:      C2003 ... added
8:      C
9:      C  NNSTEP: max step # calculated as CASE (zero if none exists)
10:     C  IERR: returns non zero if any inconsistency is found.
11:     C2005 ... modified
12:     C
13:     C  NNSTEP is deleted.
14:     C  KSTART is minimum step no of which branching-off calculation is not done.
15:     C  IBSTEP array is newly passed from AUTOBN subroutine.
16:     C  IBSTEP(1) = 0 : not requested branching-off calculation
17:     C  IBSTEP(1) = 1 : requested branching-off calculation & not yet calculated
18:     C  IBSTEP(1) = 2 : requested branching-off calculation & already calculated
19:     C=====
20:     C
21:     C ---- Arguments -----
22:     C
23:     character*(*) DIRIN, DIROUT
24:     character*4 CASE
25:     C
26:     real          ARRAY(MEMORY)
27:     integer       IARRAY(MEMORY)
28:     C
29:     C ---- COMMON area -----
30:     C
31:     include 'INC/_BURNP'
32:     C
33:     include 'INC/_CBURN1'
34:     include 'INC/_CBURN2'
35:     include 'INC/_CBURN3'
36:     C
37:     real INSCR, INTCR
38:     common /BNREST/ RSDUMY(3),      NOWSTP, IBEND,  AKEFF(MXSTEP),
39:     &      ERKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
40:     &      EXPST(MXSTEP),  CUMMWD(MXSTEP),  INSCR(MXSTEP),
41:     &      INTCR(MXSTEP),  EINSCR(MXSTEP),  EINTCR(MXSTEP),
42:     &      FLXNRM(MXSTEP),  FACNRM(MXSTEP),  FISABS(MXSTEP),
43:     &      FRTCAP(MXSTEP),  EFISAB(MXSTEP),  EFRTCA(MXSTEP),
44:     &      FDECAY(MXSTEP),  CDECAY(MXSTEP),  NHIST(MXSTEP),
45:     &      NBATCH(MXSTEP)
46:     C2005 ... added
47:     &      ,ERRNRM(MXSTEP),  POWERW(MXSTEP),  POWERE(MXSTEP),
48:     &      RMONIT(MXSTEP),  EMONIT(MXSTEP)
49:     C
50:     integer IBSTEP(MXSTEP)
51:     C
52:     C ---- local data -----
53:     C
54:     character*8 MEMBER
55:     C2005 character*32 TAG
56:     C
57:     C ---- external function -----
58:     C
59:     character*2 STEPNM
60:     external STEPNM
61:     C
62:     C *** Get current step # by checking member CASE+'HT'+step *****
63:     C
64:     IERR      = 0
65:     C

```

```

66:     MEMBER = 'OPENING'
67:     call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, ARRAY, 1, NOUT1 )
68:     C
69:     C2005 ... modified
70:     C
71:     LENG      = 0
72:     IRET      = 0
73:     C
74:     do 100 LOP = 1 , ISTOP
75:     MEMBER = CASE// 'HT' //STEPNM(LOP,0)
76:     C
77:     call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY,LENG,
78:     &          NOUT1 )
79:     C
80:     if ( LENG.gt.0 ) then
81:     C
82:         if (LENG.gt.MEMORY) then
83:             write(NOUT1,*) 'XXX(CKSTP2) Size of PDS member ',MEMBER,
84:             &          ' ( =',LENG,') exceeds buffer size ( =',MEMORY,')'
85:             stop 999
86:             end if
87:     C
88:     call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, IARRAY, LENG,
89:     &          NOUT1 )
90:     C
91:     call PCTRW ( IARRAY, MEMBER, IRET )
92:     call PCTLB2( IARRAY, MEMBER, 'INTEGER', IRET )
93:     C
94:     if ( IRET.eq.0 ) then
95:         call UNPKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
96:         KHIST      = IARRAY(LP)
97:         if ( KHIST.gt.0 ) IBSTEP(LOP) = 2
98:     Check
99:     Cdel      write(NOUT1,*) ' ** LOP KHIST (CKSTP2) : ',LOP,KHIST
100:    Cend
101:    end if
102:    C
103:    endif
104:    100 continue
105:    C
106:    do 110 I = ISTOP , 1 , -1
107:    if ( IBSTEP(I).eq.1 ) KSTART = I
108:    110 continue
109:    C
110:    Check
111:    Cdel      write(NOUT1,*) ' ** ISTOP KSTART (CKSTP2) : ',ISTOP,KSTART
112:    Cend
113:    C2005 ... end
114:    C
115:    C2005 ... deleted from this line to 'return' line
116:    C
117:    Cdel NNSTEP = 0
118:    C
119:    C ... branch mode has non-continuous steps, so check from MXSTEP
120:    C
121:    Cdel do 100 ISTEP = MXSTEP,1,-1
122:    Cdel MEMBER = CASE// 'HT' //STEPNM(ISTEP,0)
123:    Cdel call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY, LENG,
124:    Cdel &          NOUT1 )
125:    C
126:    Cdel if ( LENG.gt.0 ) then
127:    Cdel NNSTEP = ISTEP
128:    Cdel goto 105
129:    Cdel end if
130:    C

```

src/mvpburn/ckstp2.f

```

131: Cdl00 continue
132: C
133: Cdel  NNSTEP = 0
134: C
135: Cdl05 continue
136: C
137: Cdel  if( NNSTEP.le.0 ) return
138: C
139: Cdel  write(NOUT1, '()')
140: Cdel  write(NOUT1,*) ' *** MEMBERS ',CASE,'HT## EXIST UPTO ',NNSTEP,
141: Cdel & '-TH STEP. ***'
142: C
143: C ... check existence of CASE/'REST' and "NOWSTP"
144: C
145: Cdel  MEMBER = CASE/'REST'
146: C
147: Cdel  call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, ARRAY, LENG
148: Cdel & NOUT1 )
149: C
150: Cdel  if ( LENG.lt.0 ) then
151: Cdel    write(NOUT1,*)
152: Cdel & '!!!(CKSTP2) WARNING : members ',CASE,'HT## exist but no '
153: Cdel & CASE,'REST member.'
154: C
155: C ... get DAYS,U235F,CUMMWD for NNSTEP from HT## file
156: C
157: Cdel  MEMBER = CASE/'HT'/STEPNM(NNSTEP,0)
158: C
159: Cdel  write(NOUT1,*)
160: Cdel & ' Some infomation get from ',MEMBER,' file.'
161: C
162: C ... check member size
163: C
164: Cdel  call PDSIO( 'SEARCH',DIRIN,MEMBER, MEMBER, IARRAY, LENG
165: Cdel & NOUT1 )
166: C
167: Cdel  if (LENG.gt.MEMORY) then
168: Cdel    write(NOUT1,*) 'XXX(CKSTP2) Size of PDS member ',MEMBER,
169: Cdel & ' (='LENG,') exceeds buffer size (='MEMORY,')'
170: Cdel    stop 999
171: Cdel  end if
172: C
173: Cdel  call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, IARRAY, LENG
174: Cdel & NOUT1 )
175: Cdel  call PCTRW( IARRAY, MEMBER, IRET )
176: Cdel  call PCTLB2( IARRAY, MEMBER, 'REAL', IRET )
177: C
178: Cdel  if ( IRET.eq.0 ) then
179: Cdel    LOP = NNSTEP
180: Cdel    call UNPKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
181: Cdel    DAYS(LOP) = ARRAY(LP+2)
182: Cdel    U235F(LOP) = ARRAY(LP+3)
183: Cdel    EXPST(LOP) = ARRAY(LP+4)
184: Cdel    CUMMWD(LOP) = ARRAY(LP+5)
185: Cdel  end if
186: C
187: Cdel  else
188: Cdel    call PDSFIL('READ',DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
189: Cdel    if( IERR0 .eq. 0 ) then
190: Cdel      call SRTAG2( 62, 'STEP', IRC)
191: Cdel      if( IRC.ne.0 ) then
192: Cdel        read(62) TAG, NWSTP0,IBEND0
193: C
194: Cdel        if ( NNSTEP.ne.NWSTP0 ) then
195: Cdel          write(NOUT1,*)

```

```

196: Cdel & '!!!(CKSTP2) WARNING : a member ',CASE,
197: Cdel & 'REST exists but ',
198: Cdel & 'current step number ',NWSTP0,'(NOWSTP) in the member ',
199: Cdel & 'does not match highest contiguous step number in ',
200: Cdel & ' members ',CASE,'HT##.'
201: Cdel    NNSTEP = min(NNSTEP,NWSTP0)
202: Cdel    write(NOUT1,*) ' Use smaller step number ',NNSTEP
203: Cdel  end if
204: C
205: Cdel  end if
206: Cdel  end if
207: C
208: Cdel  call PDSFIL('CLOSE',DIRIN, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
209: Cdel  end if
210: C
211: C2005 ... end of deletion
212: C
213: Cdel  return
214: Cdel  end

```

src/mvpburn/clea.f

```
1:      subroutine CLEA( A,      LENG, B )
2: C*****
3: C      CLEA  IS USED FOR VALUE SET OF REAL      1-DIMENSION ARRAY.
4: C      ICLEA IS USED FOR VALUE SET OF INTEGER 1-DIMENSION ARRAY.
5: C*****
6:      real A(1)
7: C
8:      do 100 I = 1, LENG
9:          A(I)      = B
10: 100 continue
11:      return
12:  end
13: C
14:      subroutine ICLEA( IA,      LENGI, IB )
15:      integer IA(1)
16: C
17:      do 110 I = 1, LENGI
18:          IA(I)      = IB
19: 110 continue
20:      return
21:  end
22:      subroutine DCLEA( A,      IMAX, DSET )
23: C
24:      real*8 A(IMAX), DSET
25: C
26:      do 100 I = 1, IMAX
27:          A(I)      = DSET
28: 100 continue
29: C
30:      return
31:  end
```


src/mvpburn/clrcom.f

```
1:      subroutine CLRCOM
2: C
3: C   Clear common areas (is it really necessary ?!)
4: C
5: C   --- array sizes -----
6: C
7:      include 'INC/_BURNP'
8: C
9: C -- Common data -----
10: C
11:      include 'INC/_CBURN1'
12:      include 'INC/_CBURN2'
13:      include 'INC/_CBURN3'
14: C
15:      common /BNREST/ RSDUMY(3),      NOWSTP, IBEND, AKEFF(MXSTEP),
16: &      ERKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
17: &      EXPST(MXSTEP),  CUMMWD(MXSTEP), INSCR(MXSTEP),
18: &      INTCR(MXSTEP),  EINSR(MXSTEP),  EINTCR(MXSTEP),
19: &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
20: &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRICA(MXSTEP),
21: &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
22: &      NBATCH(MXSTEP)
23: C2005
24: &      ,ERRNRM(MXSTEP), POWERN(MXSTEP), POWERB(MXSTEP),
25: &      RMONIT(MXSTEP),  EMONIT(MXSTEP)
26: Cend
27: C
28: C
29: C -----
30: C
31: C ***** some common areas are cleared by dirty technique (ouch!!!)
32: C
33: C2005 LENG1   = 25 + 3*MXSTEP
34: C2005 LENG2   = 30 + 4*MXFISS + 8*MXNUC
35: C2005 LENG3   = 3 + 2*MXFISS + MXNUC
36: C2005 LENG4   = 5 + 20*MXSTEP
37:      LENG1     = 26 + 6*MXSTEP
38:      LENG2     = 30 + 4*MXFISS + 9*MXNUC + 5 + 2*MXFISS
39:      LENG3     = 0
40:      LENG4     = 5 + 25*MXSTEP
41: C
42: C   ... common /BURN1/ ...
43: C
44:      call CLEA( PERIOD, MXSTEP, 0.0 )
45:      call CLEA( POWERL, MXSTEP, 0.0 )
46:      call ICLEA( NSTP,   MXSTEP, 0 )
47: C2005 added
48:      IDMONI = 0
49: C
50:      call CLEA( EXTSRC, MXSTEP, 0.0 )
51:      call CLEA( VMONIT, MXSTEP, 0.0 )
52:      call CLEA( REGPOW, MXSTEP, 0.0 )
53: Cend
54: C
55: C   ... common /BURN2/ ...
56: C
57:      call CLEA( BNDUMY, LENG2, 0.0 )
58: C
59: C   ... common /BNREST/ ...
60: C
61:      call CLEA( RSDUMY, LENG4, 0.0 )
62: C
63:      return
64: C
65:      end
```

src/mvpburn/cmpchi.f

```
1: C/#IF MS_VISUAL
2: *      subroutine CMPCHI( N,      IM,      CCC,  III )
3: C/#ELSE
4:      subroutine CMPCHI( N,      IM,      CHAR, INT )
5: C/#ENDIF
6: c==<MVP/GMVP>=====
7: c purpose: byte-wise comparison of character string and integer.
8: c called in: data packet handling routines
9: c=====
10: C/#IF MS_VISUAL
11: C
12: C ... The type of real & dummy arguments must match each other
13: C in MS-Visual Fortran. ...
14: C
15: *      character*1 CCC(*)
16: *      integer*1   III(*)
17: *      do 100 I = 1, N
18: *          if ( char(III(I)).ne.CCC(I) ) then
19: *              IM      = 0
20: *              return
21: *          end if
22: * 100 continue
23: *      IM      = 1
24: C/#ELSE
25: *      character*1 CHAR(*)
26: *      character*1 INT(*)
27: *      do 100 I = 1, N
28: *          if ( INT(I).ne.CHAR(I) ) then
29: *              IM      = 0
30: *              return
31: *          end if
32: * 100 continue
33: *      IM      = 1
34: C/#ENDIF
35:      return
36:      end
```

```
src/mvpburn/cnterr.f
```

[illegible][illegible]

urn/cnterr.f

[illegible]

```

196:      TYPES(1) = 'FATAL'
197:      TYPES(2) = 'WARNING'
198:      TYPES(3) = 'MESSAGE'
199: C
200:      return
201: end

```

src/mvpburn/copyht.f

```

1:      subroutine COPYHT( ISTEP, CASREF,CASEID,NOUT1, NOUT2, DIRIN,
2:      &
3:      &
4:      C=====
5:      C Create 'HT' member by copying that of reference case in branch-off
6:      C calculation mode.
7:      C=====
8:      C
9:      character*4 CASREF
10:     character*4 CASEID
11:     character*(*) DIRIN, DIROUT
12: C
13: C2005 character*72 TITL2
14: C
15:     integer IARRAY(MEMORY)
16:     real RARRAY(MEMORY)
17: C
18:     real HMINV(NMAT)
19:     real EXPSZN(NMAT)
20:     real DMWZON(NMAT)
21:     real DNBURN(NTNUC,NMAT)
22: C
23:     include 'INC/_BURNP'
24: C
25:     include 'INC/_CBURNC1'
26:     include 'INC/_CBURNC2'
27:     include 'INC/_CBURNC3'
28: C
29:     real INSCR, INTCR
30:     common /MEMOCK/ MEUSED, MEMMAX
31:     common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEP(MXSTEP),
32:     &
33:     &
34:     &
35:     &
36:     &
37:     &
38:     &
39: C2005
40:     &
41:     &
42: Cend
43: C
44:     character*2 STEPNM
45:     external STEPNM
46: C
47: C ... local variables .....
48: C
49:     character*8 MEMBER
50: C2005 character*8 NUCID
51: C
52: C **** read member CASREF+'HT'+step ****
53: C
54:     MEMBER = 'OPENING'
55:     call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, IARRAY, 3, NOUT1 )
56: C
57:     MEMBER = CASREF// 'HT' //STEPNM(ISTEP,0)
58: C2003 ... check member size
59:     call PDSIO( 'SEARCH',DIRIN,MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
60: C
61:     if (LENG.gt.MEMORY) then
62:         write(NOUT1,*) 'XXX(COPYHT) Size of PDS member ',MEMBER,
63:         &
64:         stop 999
65:     else

```

```

66:         MTMP = MEMMAX - MEMORY + LENG
67:         if (MTMP.gt.MEUSED) MEUSED = MTMP
68:     end if
69: C2003 ... end
70:     call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
71: C
72:     call PCTRW( IARRAY, MEMBER, IRET )
73: C
74: C .. unpack STEP#, NMAT & NTNUC
75: C
76:     call UNPKND( IARRAY, 'STEP#', 'I4', NOWSTP1, 1, ND2, IRET )
77:     call UNPKND( IARRAY, 'NMAT', 'I4', NMAT1, 1, ND2, IRET )
78:     call UNPKND( IARRAY, 'NTNUC', 'I4', NTNUC1, 1, ND2, IRET )
79: C
80:     call PCTLB( IARRAY, 'HMINV', 'HMINV', IRET )
81:     call UNPKND( IARRAY, 'HMINV', 'R4', HMINV(1), NMAT1, ND2, IRET1 )
82:     if ( IRET1.ne.0 ) go to 100
83: C
84:     call PCTLB( IARRAY, 'EXPSZN', 'EXPSZN', IRET )
85:     call UNPKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1), NMAT1, ND2,
86:     &
87:     if ( IRET1.ne.0 ) go to 100
88: C
89:     call PCTLB( IARRAY, 'DMWZON', 'DMWZON', IRET )
90:     call UNPKND( IARRAY, 'DMWZON', 'R4', DMWZON(1), NMAT1, ND2,
91:     &
92:     if ( IRET1.ne.0 ) go to 100
93: C
94:     call PCTLB( IARRAY, 'DNBURN', 'DNBURN', IRET )
95:     call UNPKND( IARRAY, 'DNBURN', 'R4', DNBURN(1,1), NTNUC1*NMAT1,
96:     &
97:     if ( IRET1.ne.0 ) go to 100
98: C
99: C call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
100: C
101:     call PDSIO( 'CLOSE', DIRIN, MEMBER, MEMBER, IARRAY, 3, NOUT1 )
102: C
103: C **** write member CASEID+'HT'+step ****
104: C
105:     call PACK00( IARRAY, MEMBER, MEMORY, IRET )
106: C
107: C .. pack STEP#, NMAT & NTNUC
108: C
109:     NOWSTP = ISTEP
110:     LOP = ISTEP
111: C
112:     call PACKND( IARRAY, 'STEP#', 'I4', NOWSTP, 1, IRET )
113:     call PACKND( IARRAY, 'NMAT', 'I4', NMAT1, 1, IRET )
114:     call PACKND( IARRAY, 'NTNUC', 'I4', NTNUC1, 1, IRET )
115: C
116:     call PACKLB( IARRAY, 'HMINV', 'HMINV', IRET )
117:     call PACKND( IARRAY, 'HMINV', 'R4', HMINV(1), NMAT, IRET1 )
118:     if ( IRET1.ne.0 ) go to 100
119: C
120:     call PACKLB( IARRAY, 'EXPSZN', 'EXPSZN', IRET )
121:     call PACKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1), NMAT, IRET1 )
122:     if ( IRET1.ne.0 ) go to 100
123: C
124:     call PACKLB( IARRAY, 'DMWZON', 'DMWZON', IRET )
125:     call PACKND( IARRAY, 'DMWZON', 'R4', DMWZON(1), NMAT, IRET1 )
126:     if ( IRET1.ne.0 ) go to 100
127: C
128:     call PACKLB( IARRAY, 'DNBURN', 'DNBURN', IRET )
129:     call PACKND( IARRAY, 'DNBURN', 'R4', DNBURN(1,1), NTNUC*NMAT,
130:     &

```

src/mvpburn/copyht.f

```
131:      if ( IRET1.ne.0 ) go to 100
132: C
133:      call PACKLB( IARRAY, MEMBER, 'INTEGER', IRET )
134:      NPK      = 10
135:      call PACKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
136:      if ( IRET1.ne.0 ) go to 100
137: C
138:      call ICLEA( IARRAY(LP), NPK, 0 )
139: C2005 ... deleted
140: Cdel IARRAY(LP)      = NHIST(LOP)
141: Cdel IARRAY(LP+1)    = NBATCH(LOP)
142: C2005 ... end
143: C
144:      call PACKLB( IARRAY, MEMBER, 'REAL', IRET )
145: C2005      NPK      = 20
146:           NPK      = 30
147: C
148:      call PACKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
149:      if ( IRET1.ne.0 ) go to 100
150: C
151:      call CLEA( RARRAY(LP), NPK, 0.0 )
152: C
153:      RARRAY(LP)      = AKEFF(LOP)
154:      RARRAY(LP+1)    = ERRKEF(LOP)
155:      RARRAY(LP+2)    = DAYS(LOP)
156:      RARRAY(LP+3)    = U235F(LOP)
157:      RARRAY(LP+4)    = EXPST(LOP)
158:      RARRAY(LP+5)    = CUMMWD(LOP)
159:      RARRAY(LP+6)    = INSCR(LOP)
160:      RARRAY(LP+7)    = INTCR(LOP)
161:      RARRAY(LP+8)    = EINSR(LOP)
162:      RARRAY(LP+9)    = EINTCR(LOP)
163:      RARRAY(LP+10)   = FLXNRM(LOP)
164:      RARRAY(LP+11)   = FACNRM(LOP)
165:      RARRAY(LP+12)   = FISABS(LOP)
166:      RARRAY(LP+13)   = FRTCAP(LOP)
167:      RARRAY(LP+14)   = EFISAB(LOP)
168:      RARRAY(LP+15)   = EFRTCA(LOP)
169:      RARRAY(LP+16)   = FDECAY(LOP)
170:      RARRAY(LP+17)   = CDECAY(LOP)
171: C2005 ... added
172:      RARRAY(LP+18)   = ERRNRM(LOP)
173:      RARRAY(LP+19)   = POWERW(LOP)
174:      RARRAY(LP+20)   = POWERE(LOP)
175:      RARRAY(LP+21)   = RMONIT(LOP)
176:      RARRAY(LP+22)   = EMONIT(LOP)
177: C2005 ... end
178: C
179:      call PCTCLS( IARRAY, MEMBER, MEMORY0, LENG, IRET )
180: C
181: C .... Write PDS member CASE+'HT'+step-ID
182: C
183:      MEMBER = CASEID//'HT'//STEPNM(ISTEP,0)
184:      call PDSIO( 'WRITE', DIROUT, MEMBER, MEMBER, RARRAY, LENG, NOUT1 )
185: C
186:      call PDSIO( 'CLOSE', DIROUT, MEMBER, MEMBER, IARRAY, 3, NOUT1 )
187: C
188: C *** END OF PROCESS
189: C
190:      return
191: C
192: 100 write(NOUT1,6888)
193:      write(NOUT1,*) ' error in making data packet as ',
194:      &      ' contents of member <', MEMBER, '> code =', IRET
195:      stop 999

196: 6888 format(///' XXX(COPYHT) ERROR-STOP :' )
197:
198:      end
```

src/mvpburn/defcnv.f

```

1:      subroutine DEFCNV(NAMFIS,IFISFLG,FISFCT,NAMFER,IFRTFLG,FRTFCT,
2:      >                  NFIS ,NFER ,MXFISS,NOUT1
3: C=====
4: C Input of Definition conversion ratio block
5: C=====
6: C2003 character*4 NAMFIS(MXFISS), NAMFER(MXFISS)
7:      integer NAMFIS(MXFISS), NAMFER(MXFISS)
8:      integer IFISFLG(MXFISS),IFRTFLG(MXFISS)
9:      real    FISFCT(MXFISS) ,FRTFCT(MXFISS)
10: C
11:      character*72 VNAME
12:      character*128 CWRK
13: C2003
14:      character*12 TMPCH
15: C
16:      INAMFS = 0
17:      IFISFL = 0
18:      IFISFC = 0
19:      INAMFT = 0
20:      IFRTFL = 0
21:      IFRTFC = 0
22: C
23:      100 continue
24: C
25:      call FREADS( VNAME, NLEN, IEND )
26:      if ( IEND.ne.0 ) then
27:        write(NOUT1,6888)
28:        write(NOUT1,*) ' unexpected end of input in ',
29:        & 'Definition Conversion Ratio block'
30:        write(NOUT1,*)
31:        & ' Probably no "END DEF-CONV" line'
32:        stop 888
33:      end if
34: C
35: C ..... NAMFIS .....
36: C
37:      if ( VNAME(1:NLEN).eq.'NAMFIS' ) then
38:        INAMFS = 1
39:        NA = 0
40:      110 call CHREAD( VNAME(1:NLEN), CWRK, ' ', INLEN, ITERM, IEND )
41:      if ( IEND.ne.0 ) go to 330
42:      if ( INLEN.gt.0 ) then
43:        NA = NA + 1
44:        if ( NA.gt.MXFISS ) then
45:          write(NOUT1,6888)
46:          write(NOUT1,7220) VNAME(1:NLEN), MXFISS
47:          stop 999
48:        end if
49: C2003 NAMFIS(NA) = CWRK(:INLEN)
50:        MT = -1
51:        if( CWRK(INLEN:INLEN).eq.'F' ) MT=1
52:        if( CWRK(INLEN:INLEN).eq.'C' ) MT=2
53:        if( CWRK(INLEN:INLEN).eq.'A' ) MT=4
54:        if( CWRK(INLEN:INLEN).eq.'P' ) MT=0
55:        if( CWRK(INLEN:INLEN).eq.'N' ) MT=3
56:        if( CWRK(INLEN:INLEN).eq.'D' ) MT=5
57:        if( MT.eq.-1 ) then
58:          write(NOUT1,*) 'XXX(DEFCONV) tail of NAMFIS must be',
59:          & ' "F","C","A","P","N" or "D" : ',CWRK(:INLEN)
60:          call CNTERR('FATAL')
61:        end if
62:        TMPCH = CWRK(:INLEN-1)
63:        read(TMPCH,'(BN,I12)') NM
64:        NAMFIS(NA) = NM*10 + MT
65: C2003 ... end

```

```

66:      end if
67:      if ( ITERM.eq.0 ) go to 110
68: C
69:      if ( NA.ne.0 ) then
70:        if ( NFIS.ne.0.and.NA.ne.NFIS ) then
71:          write(NOUT1,6888)
72:          write(NOUT1,7180) VNAME(1:NLEN)
73:          call CNTERR('FATAL')
74:        else if ( NFIS.eq.0 ) then
75:          NFIS = NA
76:        end if
77:      end if
78: C
79: C ..... IFISFLG .....
80: C
81:      else if ( VNAME(1:NLEN).eq.'IFISFLG' ) then
82:        IFISFL = 1
83:        call I4READ( VNAME(1:NLEN), IFISFLG, NA, -MXFISS, IEND )
84:        if ( IEND.ne.0 ) go to 330
85:        if ( NA.ne.0 ) then
86:          if ( NFIS.ne.0.and.NA.ne.NFIS ) then
87:            write(NOUT1,6888)
88:            write(NOUT1,7180) VNAME(1:NLEN)
89:            call CNTERR('FATAL')
90:          else if ( NFIS.eq.0 ) then
91:            NFIS = NA
92:          end if
93:        end if
94: C
95: C ..... FISFACT .....
96: C
97:      else if ( VNAME(1:NLEN).eq.'FISFACT' ) then
98:        IFISFC = 1
99:        call R4READ( VNAME(1:NLEN), FISFCT, NA, -MXFISS, IEND )
100:        if ( IEND.ne.0 ) go to 330
101:        if ( NA.ne.0 ) then
102:          if ( NFIS.ne.0.and.NA.ne.NFIS ) then
103:            write(NOUT1,6888)
104:            write(NOUT1,7180) VNAME(1:NLEN)
105:            call CNTERR('FATAL')
106:          else if ( NFIS.eq.0 ) then
107:            NFIS = NA
108:          end if
109:        end if
110: C
111: C ..... NAMFRT .....
112: C
113:      else if ( VNAME(1:NLEN).eq.'NAMFRT' ) then
114:        INAMFT = 1
115:        NA = 0
116:      120 call CHREAD( VNAME(1:NLEN), CWRK, ' ', INLEN, ITERM, IEND )
117:      if ( IEND.ne.0 ) go to 330
118:      if ( INLEN.gt.0 ) then
119:        NA = NA + 1
120:        if ( NA.gt.MXFISS ) then
121:          write(NOUT1,6888)
122:          write(NOUT1,7220) VNAME(1:NLEN), MXFISS
123:          stop 999
124:        end if
125: C2003 NAMFER(NA) = CWRK(:INLEN)
126:        MT = -1
127:        if( CWRK(INLEN:INLEN).eq.'F' ) MT=1
128:        if( CWRK(INLEN:INLEN).eq.'C' ) MT=2
129:        if( CWRK(INLEN:INLEN).eq.'A' ) MT=4
130:        if( CWRK(INLEN:INLEN).eq.'P' ) MT=0

```

src/mvpburn/defcnv.f

```

131:         if( CWRK(INLEN:INLEN).eq.'N') MT=3
132:         if( CWRK(INLEN:INLEN).eq.'D') MT=5
133:         if( MT.eq.-1 ) then
134:             write(NOUT1,*) 'XXX(DEFCONV) tail of NAMFER must be',
135:             & ' "F","C","A","P","N" or "D" : ',CWRK(:INLEN)
136:             call CNTERR('FATAL')
137:         end if
138:         TMPCH = CWRK(:INLEN-1)
139:         read(TMPCH,'(BN,I12)') NM
140:         NAMFER(NA) = NM*10 + MT
141: C2003 ... end
142:     end if
143:     if ( ITERM.eq.0 ) go to 120
144: C
145:     if ( NA.ne.0 ) then
146:         if ( NFER.ne.0.and.NA.ne.NFER ) then
147:             write(NOUT1,6888)
148:             write(NOUT1,7190) VNAME(1:NLEN)
149:             call CNTERR('FATAL')
150:         else if ( NFER.eq.0 ) then
151:             NFER = NA
152:         end if
153:     end if
154: C
155: C ..... IFRTFLG .....
156: C
157:     else if ( VNAME(1:NLEN).eq.'IFRTFLG' ) then
158:         IFRTFL = 1
159:         call I4READ( VNAME(1:NLEN), IFRTFLG, NA, -MXFISS, IEND )
160:         if ( IEND.ne.0 ) go to 330
161:         if ( NA.ne.0 ) then
162:             if ( NFER.ne.0.and.NA.ne.NFER ) then
163:                 write(NOUT1,6888)
164:                 write(NOUT1,7190) VNAME(1:NLEN)
165:                 call CNTERR('FATAL')
166:             else if ( NFER.eq.0 ) then
167:                 NFER = NA
168:             end if
169:         end if
170: C
171: C ..... FRTFACT .....
172: C
173:     else if ( VNAME(1:NLEN).eq.'FRTFACT' ) then
174:         IFRTFC = 1
175:         call R4READ( VNAME(1:NLEN), FRTFACT, NA, -MXFISS, IEND )
176:         if ( IEND.ne.0 ) go to 330
177:         if ( NA.ne.0 ) then
178:             if ( NFER.ne.0.and.NA.ne.NFER ) then
179:                 write(NOUT1,6888)
180:                 write(NOUT1,7190) VNAME(1:NLEN)
181:                 call CNTERR('FATAL')
182:             else if ( NFER.eq.0 ) then
183:                 NFER = NA
184:             end if
185:         end if
186: C
187: C ..... END CONVERSION RATIO .....
188: C
189:     else if ( VNAME(1:NLEN).eq.'END') then
190:         call RESET
191:         go to 130
192:     else
193:         write(NOUT1,6888)
194:         write(NOUT1,7200) VNAME(1:NLEN)
195:         call CNTERR('FATAL')
196:
197:         call DMREAD( VNAME(:NLEN), NA, NB, IEND )
198:         end if
199:         go to 100
200: C
201: 130 continue
202:         if ( INAMFS .eq.0 ) then
203:             write(NOUT1,6888)
204:             write(NOUT1,*) ' NAMFIS data is not inputed'
205:             call CNTERR('FATAL')
206:         end if
207:         if ( IFISFL .eq.0 ) then
208:             write(NOUT1,6888)
209:             write(NOUT1,*) ' IFISFLG data is not inputed'
210:             call CNTERR('FATAL')
211:         end if
212:         if ( IFISFC .eq.0 ) then
213:             write(NOUT1,6888)
214:             write(NOUT1,*) ' FISFACT data is not inputed'
215:             call CNTERR('FATAL')
216:         end if
217:         if ( INAMFT .eq.0 ) then
218:             write(NOUT1,6888)
219:             write(NOUT1,*) ' NAMFRT data is not inputed'
220:             call CNTERR('FATAL')
221:         end if
222:         if ( IFRTFL .eq.0 ) then
223:             write(NOUT1,6888)
224:             write(NOUT1,*) ' IFRTFLG data is not inputed'
225:             call CNTERR('FATAL')
226:         end if
227:         if ( IFRTFC .eq.0 ) then
228:             write(NOUT1,6888)
229:             write(NOUT1,*) ' IFRTFC data is not inputed'
230:             call CNTERR('FATAL')
231:         end if
232:         return
233: C
234: 330 continue
235:         write(NOUT1,6888)
236:         write(NOUT1,7300) VNAME(:NLEN)
237:         stop 888
238: C
239: 6888 format(/// ' XXX(DEFCONV) ERROR-STOP :' )
240: 7180 format(1X,'number of data <',A,
241:             & '> does not match number of NFIS determined by other input.')
242: 7190 format(1X,'number of data <',A,
243:             & '> does not match number of NFER determined by other input.')
244: 7200 format(1X,'unknown data name <',A,
245:             & '>. Skipping ...')
246: 7220 format(1X,'number of data <',A,> exceeds internal',
247:             & ' buffer size (' ,I5,')')
248: 7300 format(1X,'end of input data in reading data <',A,
249:             & '> in $BURNUP block.')
250: C
251:     END

```


src/mvpburn/dtlist.f

```

1:      subroutine DTLIST( NIN,  NOUT, LMVPI,  IPR,  IOPT ,MVPDBG )
2:      C
3:      C=<MVPBURN>=====
4:      C Purpose :
5:      C Print input list and save in NOUT-th device. After DTLIST,
6:      C all input data should be read from NOUT-th device.
7:      C Therefore, DTLIST should be called first of all routines.
8:      C If IOPT=1, then change all small chracters to large
9:      C characters.
10:     C
11:     C * For MVP-BURN
12:     C
13:     C Separate MVP input and store on I/O unit LMVPI.
14:     C
15:     C-----
16:     C
17:     C character*80 AA
18:     C character*26 SMALL, LARGE
19:     C
20:     C logical OPD, NAMED
21:     C character*256 file
22:     C
23:     C SMALL = 'abcdefghijklmnopqrstuvwxyz'
24:     C LARGE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
25:     C call RWIND( NOUT )
26:     C
27:     C ... check that unit LMVPI is opened or not.
28:     C If the unit is opened, print warning and ignore MVP input data
29:     C from standard input.
30:     C
31:     C JMVP = 1
32:     C
33:     C inquire( LMVPI, opened=opd, named=named )
34:     C if ( OPD ) then
35:     C   write(IPR,*) '!!! (DTLIST) I/O unit for MVP-Input (',LMVPI,
36:     C   & ' ) is already opened.',
37:     C   & ' MVP-input in MVP-BURN input will be ignored (is it OK?).'
38:     C   if( named ) then
39:     C     file = ' '
40:     C     inquire( LMVPI, name=file )
41:     C     write(ipr,*) '!!! Filename : ',file(:iclen2(file))
42:     C   else
43:     C     write(ipr,*) '!!! The opened unit seems scratch ???'
44:     C   end if
45:     C   JMVP = 0
46:     C else
47:     C   open(LMVPI, status='SCRATCH', form='FORMATTED' )
48:     C   end if
49:     C
50:     C-----
51:     C
52:     C Possible Format of MVP-BURN input combined with MVP input.
53:     C
54:     C Type 1:
55:     C
56:     C *#MVPBURN (from the first column of the first line)
57:     C
58:     C $BURNUP
59:     C ...
60:     C $END BURNUP
61:     C
62:     C * ... comment on any position
63:     C
64:     C $MVPINP
65:     C

```

```

66:     C (MVP input lines)
67:     C
68:     C $END MVPINP
69:     C
70:     C
71:     C Type 2: MVP-BURN control input is given in comment line of MVP input
72:     C
73:     C title1 (other than "#MVPBURN")
74:     C title2
75:     C .....
76:     C
77:     C *$MVPBURN
78:     C *
79:     C *$BURNUP
80:     C * ...
81:     C *$END BURNUP
82:     C * ...
83:     C ** ... this is a comment in MVP-BURN input
84:     C *
85:     C *$MVPBURN
86:     C ...
87:     C ...
88:     C
89:     C-----
90:     C
91:     C ICOUNT = 0
92:     C JEX = 0
93:     C JTYPE = 2
94:     C do 130 IPAGE = 1, 10000
95:     C   if (MVPDBG.ne.0) then
96:     C     write(IPR,7000) IPAGE
97:     C     write(IPR,7020)
98:     C     write(IPR,7040)
99:     C   end if
100:    C do 120 I = 1, 50
101:    C   ICOUNT = ICOUNT + 1
102:    C   read(NIN,'(A)',end =140) AA
103:    C
104:    C   if( ICOUNT.eq.1 .and. AA.eq.'#MVPBURN' ) then
105:    C     JTYPE = 1
106:    C   end if
107:    C
108:    C   if ( JTYPE.eq.1 ) then
109:    C     if( AA.eq.'$MVPINP' ) then
110:    C       JEX = 1
111:    C     else if( AA.eq.'$END MVPINP' ) then
112:    C       JEX = 0
113:    C     end if
114:    C   end if
115:    C
116:    C   if ( JTYPE.eq.2 ) then
117:    C     if( AA.eq.'*$MVPBURN' ) then
118:    C       JEX = 1
119:    C     else if( AA.eq.'*$END MVPBURN' ) then
120:    C       JEX = 0
121:    C     end if
122:    C   end if
123:    C
124:    C***** CHANGE small TO LARGE characters *****
125:    C
126:    C   if ( IOPT.eq.1 ) then
127:    C     do 110 J = 1, 80
128:    C       do 100 K = 1, 26
129:    C         if ( AA(J:J).eq.SMALL(K:K) ) AA(J:J) =
130:    C         & LARGE(K:K)

```

src/mvpburn/dtlist.f

```
131: 100          continue
132: 110          continue
133:          end if
134: C
135: C*****
136: C
137:          if (MVPDBG.ne.0) then
138:              write(IPR,7060) ICOUNT, AA, ICOUNT
139:          end if
140: C
141: C ... separate input for MVP-BURN & MVP .....
142: C
143: C
144:          if( JTYPE.eq.1 ) then
145:              if( JEX.eq.0 ) then
146:                  if( AA.ne.'$END MVPINP' ) write(NOUT,'(A)') AA
147:              else
148:                  if( JMVP.ne.0.and.JEX.ge.2 ) write(LMVPI,'(A)') AA
149:                  JEX = JEX + 1
150:              end if
151:          else
152:              if( JEX.eq.0 ) then
153:                  if( JMVP.ne.0.and. AA.ne.'*$END MVPBURN' ) then
154:                      write(LMVPI,'(A)') AA
155:                  end if
156:              else
157:                  if( JEX.ge.2 ) write(NOUT,'(A)') AA(2:)
158:                  JEX = JEX + 1
159:              end if
160:          end if
161: C
162: 120          continue
163:          if (MVPDBG.ne.0) then
164:              write(IPR,7040)
165:              write(IPR,7080)
166:          end if
167: 130 continue
168: C
169: 140 continue
170: C
171:          call RWIND( NOUT )
172:          if( JMVP.ne.0 ) call RWIND(LMVPI)
173: C
174:          if (MVPDBG.ne.0) then
175:              write(IPR,7100)
176:          end if
177:          return
178: C
179: 7000 format('1',92X,'PAGE-',I4.4)
180: 7020 format(18X,'*****')
181:      &      /18X,'*'
182:      &      /18X,'* INPUT DATA LIST *'
183:      &      /18X,'*'
184:      &      /18X,'*****')
185: 7040 format(12X,'...*...1...*...2...*...3...*...4',
186:      &      '...*...5...*...6...*...7...*...8')
187: 7060 format(18,4X,A,I7)
188: 7080 format(12X,'*** CONTINUE ***')
189: 7100 format(12X,'*** INPUT DATA END ***')
190: C
191:          end
```

src/mvpburn/fnbate.f

```

1:      function FNBATE(LL, LONG, RAMDA, GAMMA, ANO, R, G, P, GAMX, JX, KSTP, T, XM,
2:      C2004&      NMAX, DEX, DEXX, DCOEF, IMAT, ISKIP, SRACID, KOOPT)
3:      &      NMAX, DEX, DEXX, DCOEF, IMAT, ISKIP, IHOL, KOOPT)
4:      C=====
5:      C
6:      common /REAMIO/  NOUT6,  NDTLS
7:      C
8:      real*8 DEX(NMAX), DEXX(NMAX), DCOEF(LONG), T
9:      C
10:     real RAMDA(NMAX), GAMMA(NMAX), ANO(NMAX), R(LONG), G(LONG),
11:     &      P(LONG), GAMX(NMAX)
12:     integer JX(LONG)
13: C2003 ... add ISKIP, SRACID
14: integer ISKIP(LONG)
15: C2004 .. replace SRACID by IHOL
16: CCC character*4 SRACID(NMAX)
17: character*8 IHOL(NMAX)
18: Cend
19: C
20: real*8 DA, DGXM, DMS, QG, QF, RL, RM, RI, RLPL, FN, EPS0
21: data EPS0 /1.0000D-10/
22: C
23: C *** START OF PROCESS
24: C
25: FN = 0.0
26: if ( LL.gt.0.and.LONG.gt.0 ) then
27: C
28: C
29: LONGT = LONG
30: C
31: C
32: JXL = JX(LONGT)
33: if ( LL.ge.LONGT ) then
34: DA = ANO(JXL)
35: DGXM = GAMMA(JXL)*XM
36: if ( LONGT.eq.1.and.KSTP.eq.1 ) DGXM = GAMX(JXL)*XM
37: FN = FN + DA*DEX(JXL) + DGXM*DEXX(JXL)
38: end if
39: C
40: C
41: if ( LONGT.gt.1 ) then
42: C
43: C
44: LONGM = LONGT - 1
45: RL = RAMDA(JXL)
46: do 100 M = 1, LONGM
47: JXM = JX(M)
48: RM = RAMDA(JXM)
49: DCOEF(M) = 1.0D0
50: Ckuni 2003/12/22
51: Cmod if ( DABS((RL-RM)/RM).gt.EPS0 ) DCOEF(M)=DCOEF(M)/(RL-RM)
52: C
53: if ( ISKIP(M).gt.0.and.DABS((RL-RM)/RM).gt.EPS0 ) then
54: DCOEF(M) = DCOEF(M) / (RL-RM)
55: endif
56: Cend
57: 100 continue
58: DCOEF(LONGT) = 1.0D0
59: C
60: KSM = 0
61: KSL = 0
62: ICOUNT = 0
63: C
64: do 150 N = 2, LONGT
65: L = LONGT - N + 1

```

```

66: Ckuni 2003/12/22
67: if( ISKIP(L).eq.0 ) go to 150
68: Cend
69: LONGM = L + 1
70: JXL = JX(L)
71: RL = RAMDA(JXL)
72: RLPL = RAMDA(JXL)*P(L)
73: DA = ANO(JXL)
74: DGXM = GAMMA(JXL)*XM
75: if ( L.eq.1.and.KSTP.eq.1 ) DGXM = GAMX(JXL)*XM
76: Ckuni 2003/12/22
77: Cdeleted beacuse RLPL zero check is done in sub(FNCALC) !!!
78: cdel if ( RLPL.le.0.0 ) go to 170
79: Cend
80: C
81: do 110 M = LONGM, LONGT
82: Ckuni 2003/12/22
83: if( ISKIP(M).gt.0 ) then
84: Cend
85: JXM = JX(M)
86: RM = RAMDA(JXM)
87: DCOEF(M) = DCOEF(M)*RLPL
88: C
89: Ckuni 2003/12/22
90: Cmod if ( DABS((RL-RM)/RL).le.EPS0 ) then
91: if ( DABS((RL-RM)/RL).gt.EPS0 ) then
92: DCOEF(M) = DCOEF(M) / (RL-RM)
93: C
94: else
95: KSM = M
96: KSL = L
97: ICOUNT = ICOUNT + 1
98: Ckuni if ( ICOUNT.gt.1 ) go to 180
99: if ( ICOUNT.gt.1 ) then
100: FNBATE = FN
101: C
102: if(KOOPT.eq.1) then
103: write(NOUT6,6666)
104: C write(NOUT6,*) ' Warning at sub(FNBATE) !!! '
105: write(NOUT6,*) ' In ',IMAT,'-th material, ',
106: C20041 SRACID(JXL), ' & ', SRACID(JXM),
107: 1 IHOL(JXL), ' & ', IHOL(JXM),
108: 2 ' have the same disappearance(decay + absorption) rate !'
109: C
110: write(NOUT6,*) ' Following reactions are ignored in ',
111: C20041 SRACID(JX(LONG)), ' burnup density calculation !'
112: 1 IHOL(JX(LONG)), ' burnup density calculation !'
113: C2004 write(NOUT6,777) (SRACID(JX(KK)), KK=1, JXL)
114: write(NOUT6,777) (IHOL(JX(KK)), KK=1, JXL)
115: endif
116: C
117: RETURN
118: endif
119: C
120: end if
121: endif
122: Cend
123: Cmove else
124: Cmove DCOEF(M) = DCOEF(M) / (RL-RM)
125: Cmove end if
126: 110 continue
127: C
128: do 120 M = 1, L
129: Ckuni 2003/12/22
130: if(ISKIP(M).gt.0 ) then

```

src/mvpburn/fnbate.f

```

131: C
132:       JXM      = JX(M)
133:       RM      = RAMDA(JXM)
134:       DCOEF(M) = DCOEF(M)*RLPL
135:       if ( DABS((RL-RM)/RL).gt.EPS0 ) then
136:         DCOEF(M) = DCOEF(M) / (RL-RM)
137:       end if
138: Ckuni
139:       endif
140: Cend
141: 120      continue
142: C
143:       if ( LL.ge.L.and.(DA.ne.0.0.or.DGXM.ne.0.0) ) then
144: C
145:         if ( KSM.gt.0 ) then
146: C
147:           DMS      = 0.0
148:           QG       = 0.0
149:           QF       = 0.0
150:           do 130 M = L, LONGT
151: Ckuni
152:             if ( M.ne.KSL ) then
153:               if ( M.ne.KSL .and. ISKIP(M).gt.0 ) then
154: C
155:                 if ( M.ne.KSM ) then
156:                   JXM      = JX(M)
157:                   DMS      = DMS + DCOEF(M)
158:                   QG       = QG + DCOEF(M)*DEX(JXM)
159:                   QF       = QF + DCOEF(M)*DEXX(JXM)
160:                 end if
161:               end if
162:             continue
163:             JXKSM      = JX(KSM)
164:             RI         = RAMDA(JXKSM)
165:             QG         = QG + DCOEF(KSM)*DEX(JXKSM)*T
166:             &          - DMS*DEX(JXKSM)
167:             QF         = QF + DCOEF(KSM)*
168:             &          (DEXX(JXKSM)-T*DEX(JXKSM)) / RI
169:             &          - DMS*DEXX(JXKSM)
170:           else
171: C
172:             DMS      = 0.0
173:             QG       = 0.0
174:             QF       = 0.0
175: C
176:             do 140 M = LONGM, LONGT
177: Ckuni 2003/12/22
178:             if( ISKIP(M).gt.0 ) then
179: C
180:               JXM      = JX(M)
181:               DMS      = DMS + DCOEF(M)
182:               QG       = QG + DCOEF(M)*DEX(JXM)
183:               QF       = QF + DCOEF(M)*DEXX(JXM)
184:             end if
185:           Cend
186: C
187:           continue
188:           QG      = QG - DMS*DEX(JXL)
189:           QF      = QF - DMS*DEXX(JXL)
190:         end if
191: C
192:         FN      = FN + DA*QG + DGXM*QF
193:         if ( FN.le.0.0 ) FN = 0.0
194:       end if
195: 150      continue

```

```

196:       end if
197:     end if
198: C
199: C
200: C
201: 160 FNBATE = FN
202:       return
203: C
204: C
205: C
206: C 170 write(NOUT6,6888)
207: C       write(NOUT6,7000) IMAT
208: C       write(NOUT6,7020) JXL, L, RL, P(L), DA, XM, DGXM
209: C       stop 888
210: C
211: C
212: C=====
213: C
214: C 180 write(NOUT6,6666)
215: C       write(NOUT6,7040) ICOUNT, JX(LONG), LL
216: C7000 format(1X,'Probably flux or micro-reaction rate of the ', I4,
217: C      & '-th material region is zero'/1X,
218: C      &'It is caused by small history or small tally region')
219: C7020 format(1X,' ## JXL L RL P(L) ANO XM DGXM ## ',2I6,1P,5E12.5)
220: C7040 format(1X,'ICOUNT is greater than 1'/1X,
221: C      &      'ICOUNT = ',I6,' for ',I4,'-th nuclide. ** LL = ',I6)
222: C 6666 format(///' !!!(FNBATE) WARNING :' )
223: C6888 format(///' XXX(FNBATE) ERROR-STOP :' )
224: C
225: C       go to 160
226: C
227: C204
228: CC777 format(' chain : ',10(A4,'->'))
229: 777 format(' chain : ',10(A8,'->'))
230: Cend
231: C
232: end

```

```

1:      subroutine FNCALC( RAMDA, NCH, NBIC, PBIC, NOL, KSTP, LONG,
2:      & LL, IP, KP, NPN, GAMMA, GAMX, SIGA,
3:      & SIGC, SIGN2N, PHIC, PHIL, PHAI, P, R,
4:      & G, RAM, ANO, AN, DEX, DEXX,
5:      & DCOEF, MAP, FLX, T, XM, NMAX, NPAR,
6:      Ckuni& NCHA, LCHA, MAPLEN, IMAT, IBC )
7:      & NCHA, LCHA, MAPLEN, IMAT, IBC, KOOPT, KZEROP,
8:      CKSK & IDBG, BZERO )
9:      & IDBG, BZERO, NBOPT)
10: C
11:      common /REAMIO/ NOUT6, NDTLS
12: C
13:      real*8 DEX(NMAX), XX, T, DEXX(NMAX), DCOEF(LCHA)
14:      real*8 FLX
15: C
16:      real RAMDA(NMAX)
17:      integer NCH(NMAX)
18:      integer NBIC(NPAR,NMAX)
19:      real PBIC(NPAR,NMAX)
20:      integer NOL(NMAX)
21:      integer KSTP(NCHA,NMAX), LONG(NCHA,NMAX), LL(NCHA,NMAX),
22:      & IP(LCHA,NCHA,NMAX), KP(LCHA,NCHA,NMAX), NPN(NPAR,NMAX)
23:      real GAMMA(NMAX), GAMX(NMAX), SIGA(NMAX), SIGC(NMAX),
24:      & SIGN2N(NMAX), PHIC(NMAX), PHIL(NMAX), PHAI(NPAR,NMAX)
25:      & R(LCHA), G(LCHA), P(LCHA), RAM(NMAX), ANO(NMAX), AN(NMAX)
26:      integer MAP(MAPLEN)
27: C2003 ... added as argument
28:      integer IBC(20)
29: Ckuni 2003/12/21
30:      integer KZEROP(NCHA,NMAX)
31: C
32:      include 'INC/_BURNP'
33: C
34:      include 'INC/_CBURN3'
35: C
36:      integer MAP2(MXNUC), KBIC(MXNUC), ISKIP(MXNUC)
37:      character*12 IDREAC(MXNUC)
38: C
39:      real PMIN
40:      data PMIN / 1.0E-35 /
41: Cend
42: C
43:      real*8 ERR
44: C
45:      data ERR, EPS /1.0D-30, 1.0E-10/
46: C
47: C
48:      if ( MAPLEN.gt.0 ) then
49: Ckuni
50:          ICHECK = 0
51:          if( MAPLEN.lt.20.and.IDBG.ge.0 ) ICHECK = 1
52: Cm          if( MAPLEN.lt.20 ) ICHECK = 1
53: Cend
54: C
55:      do 100 NN = 1, NMAX
56:          PHIL(NN) = 0.0
57:          PHIC(NN) = 0.0
58:          DEX(NN) = 1.0
59:          DEXX(NN) = T
60:          RAM(NN) = RAMDA(NN) + FLX*SIGA(NN)
61:          if ( RAM(NN).ne.0.0 ) then
62:              PHIL(NN) = RAMDA(NN) /RAM(NN)
63:              XX = RAM(NN)*T
64:              DEX(NN) = 0.0
65:              if ( XX.le.174.673 ) DEX(NN) = DEXP(-XX)

```

[illegible]

src/mvpburn/fncalc.f

```

131: 110      continue
132:      end if
133: 120      continue
134: Ckuni 2003/12/22
135: C
136: C *** CHECK CHAIN DATA & REACTION RATE DATA
137: C
138:      IERR      = 0
139:      CALL ICLEA ( KZEROP , NCHA*NMAX , 0 )
140: C
141:      DO 320 MP = 1 , MAPLEN
142:      NN      = MAP(MP)
143:      NOS      = NOL(NN)
144: C
145:      DO 310 I=1,NOS
146:      IST = 0
147:      LL2 = LONG(I,NN) - 1
148:      if ( LL2.ge.1 ) then
149:      DO 260 K=1,LL2
150:      IP1 = IP(K+1,I,NN)
151:      KP1 = KP(K+1,I,NN)
152:      P(K) = PHAI(KP1,IP1)
153:      KBIC(K) = NBIC(KP1,IP1)
154:      IP0 = IP(K,I,NN)
155:      if(IST.eq.0.and.MAP2(IP0).gt.0) IST = K
156:      ISKIP(K) = 0
157: C
158:      if(KBIC(K).eq.1) IDREAC(K) = ' Beta- '
159:      if(KBIC(K).eq.2) IDREAC(K) = ' I.1 '
160:      if(KBIC(K).eq.3) IDREAC(K) = ' Gamma '
161:      if(KBIC(K).eq.4) IDREAC(K) = ' Beta+ '
162:      if(KBIC(K).eq.5) IDREAC(K) = ' E.C. '
163:      if(KBIC(K).eq.6) IDREAC(K) = ' Alpha '
164:      if(KBIC(K).eq.7) IDREAC(K) = ' Delayed '
165:      if(KBIC(K).eq.8) IDREAC(K) = ' n,2n '
166: 260      CONTINUE
167:      endif
168: C
169:      LLL      = LONG(I,NN)
170:      P(LL2)    = 1.0
171:      KBIC(LL2) = 0
172:      IDREAC(LL2) = ' Self '
173:      if(IST.eq.0) IST = LLL
174: C
175:      do 305 K = IST , LLL
176:      ISKIP(K) = 1
177: 305      continue
178: C
179: C **** check small reation rate
180: C
181:      if( IST.lt.LLL ) then
182:      do 307 K = LLL-1 , IST , -1
183:      RAMNOW = RAM(IP(K,I,NN))
184:      RLPL = P(K)*RAMNOW
185: C
186:      if(RLPL.lt.PMIN.and.KZEROP(I,NN).eq.0) then
187:      KZEROP(I,NN) = K
188:      IERR      = IERR + 1
189:      endif
190: C
191: 307      continue
192:      endif
193: C
194:      if(ICHECK.gt.0.and.KOOPT.eq.1) then
195: C2004      write(NOUT6,*) ' *** check write at FNCALC for ',SRACID(NN)

```

```

196:      write(NOUT6,*) ' *** check write at FNCALC for ',IHOL(NN)
197:      WRITE(NOUT6,600) NN,I,LL(I,NN),LONG(I,NN),KSTP(I,NN),NCHA
198:      +      ,KZEROP(I,NN)
199:      if(I.eq.1) then
200:      WRITE(NOUT6,604) RAM(NN),SIGA(NN),SIGC(NN),SIGN2N(NN),
201:      +      ANO(NN)
202:      WRITE(NOUT6,603) FLX,PHIL(NN),PHIC(NN),PHIC(NN)*SIGC(NN)
203:      WRITE(NOUT6,606) (NPN(J,NN),J=1,NCH(NN))
204:      endif
205:      WRITE(NOUT6,610) (IP(J,I,NN),J=1,LLL)
206: C2004      WRITE(NOUT6,605) (SRACID(IP(J,I,NN)),J=1,LLL)
207:      WRITE(NOUT6,605) (IHOL(IP(J,I,NN)),J=1,LLL)
208:      WRITE(NOUT6,620) (KP(J,I,NN),J=1,LLL)
209:      WRITE(NOUT6,625) (KBIC(J),J=1,LLL)
210:      WRITE(NOUT6,626) (IDREAC(J),J=1,LLL)
211:      WRITE(NOUT6,630) (P(J),J=1,LLL)
212:      WRITE(NOUT6,631) (P(J)*RAM(IP(J,I,NN)),J=1,LLL)
213:      WRITE(NOUT6,635) (MAP2(IP(J,I,NN)),J=1,LLL)
214:      WRITE(NOUT6,636) (ISKIP(J),J=1,LLL)
215:      endif
216: C
217: 310      CONTINUE
218: 320      CONTINUE
219: C
220: 600      FORMAT(1H,'NN I LL LONG KSTP NCHA KZEROP : ',10I6)
221: 603      FORMAT(1H,'FLX PHIL PHIC,PHIC*SIGC : ',1P5E12.4)
222: 604      FORMAT(1H,'RAM SIGA SIGC SIGN2N DN : ',1P5E12.4)
223: C2004
224: C 605      FORMAT(1H,'NUC : ',10(8X,A4))
225: 605      FORMAT(1H,'NUC : ',10(4X,A8))
226: Cend
227: 606      FORMAT(1H,'NPN : ',10I12)
228: 610      FORMAT(1H,'IP : ',10I12)
229: 620      FORMAT(1H,'KP : ',10I12)
230: 625      FORMAT(1H,'KBIC : ',10I12)
231: 626      FORMAT(1H,'REAC : ',10A12)
232: 630      FORMAT(1H,'P : ',1P10E12.4)
233: 631      FORMAT(1H,'P*RAM : ',1P10E12.4)
234: 635      FORMAT(1H,'MAP2 : ',10I12)
235: 636      FORMAT(1H,'ISKIP : ',10I12)
236: C
237:      if(IERR.gt.0.and.IBC(19).ne.1) then
238:      write(NOUT6,6888)
239:      write(NOUT6,*) ' Burnup calculation stopped because of too small'
240:      write(NOUT6,*) ' reaction rate in ',IMAT,'-th depleting material.'
241:      write(NOUT6,*) ' Restart using ACCEPT-ZERO-RATE option'
242:      write(NOUT6,*) ' if no input error & you want to continue.'
243: C-----1-----2-----3-----4-----5-----6-----7--
244:      stop 888
245:      endif
246: Cend 2003/12/22
247: C
248: C *** CALCULATE BURNUP DENSITY
249: C
250:      do 150 MP = 1 , MAPLEN
251:      NN      = MAP(MP)
252:      AN(NN) = 0.0
253:      NOS      = NOL(NN)
254: C
255:      do 140 I = 1 , NOS
256:      IST = 0
257:      LL2 = LONG(I,NN) - 1
258:      if ( LL2.gt.0 ) then
259:      do 130 K = 1 , LL2
260:      IP1 = IP(K+1,I,NN)

```

src/mvpburn/fncalc.f

```
261:          KP1      = KP(K+1,I,NN)
262:          P(K)      = PHAI(KP1,IP1)
263: Ckuni 2003/12/22
264:          IP0      = IP(K,I,NN)
265:          KBIC(K)    = NBIC(KP1,IP1)
266:          if(IST.eq.0.and.MAP2(IP0).gt.0) IST = K
267:          ISKIP(K) = 0
268:          130      continue
269: Cend
270:          end if
271: C
272: Ckuni 2003/12/22
273: Cmod      P(LL2+1)  = 1.0
274:          LLL      = LONG(I,NN)
275:          KKK      = LL(I,NN)
276:          IZERO     = KZEROP(I,NN)
277: C
278:          P(LL)     = 1.0
279:          if(IST.eq.0) IST = LLL
280: C
281:          if(IZERO.gt.0.and.IZERO.ge.KKK) then
282: C
283:          if(KOOPT.eq.1.and.IDBG.gt.0) then
284: C2004      write(NOUT6,*) ' ** ',I,'-th chain of ',SRACID(NN),
285:          write(NOUT6,*) ' ** ',I,'-th chain of ',IHOL(NN),
286:          &      ' (',IMAT,'-th material) was ignored',
287:          &      ' becuase of small reaction rate !'
288:          endif
289: C
290:          go to 140
291:          endif
292: C
293:          if(IZERO.gt.0.and.IST.le.IZERO) IST = IZERO + 1
294: C
295:          do 135 K = IST , LLL
296:          ISKIP(K) = 1
297:          135      continue
298: Cend
299: C2003 ... add argument IMAT, ISKIP,SRACID,KOOPT
300: C2004 ... add argument IMAT, ISKIP,IHOL ,KOOPT
301:          AN(NN)    = AN(NN) + FNBATE(LL(I,NN),LONG(I,NN),RAM,GAMMA,
302:          &          ANO,R,G,P,GAMX,IP(1,I,NN),KSTP(I,NN),T,XM,NMAX,
303: C2004&          DEX,DEXX,DCOEF,IMAT,ISKIP,SRACID,KOOPT)
304:          &          DEX,DEXX,DCOEF,IMAT,ISKIP,IHOL ,KOOPT)
305: C
306:          140      continue
307:          150      continue
308:          end if
309: C
310: C *** end of process
311: C
312:          return
313: C
314:          6666 format(/// '!!!(FNCALC) WARNING :' )
315:          6888 format(/// 'XXX(FNCALC) ERROR-STOP :' )
316: C
317:          end
```

src/mvpburn/getden.f

```
1:      subroutine GETDEN( LINE,  ANS,  NUCID )
2:  C
3:      character*72 LINE
4: C2004 character*1 IBRA, IKETO
5:      character*(*) NUCID
6:  C
7:      ANS      = -1.00E+30
8:      NUCID    = ' '
9:  C
10:     do 100 I = 1, 72
11:         IST      = I
12:         if ( LINE(I:I).ne.' ' ) go to 110
13:     100 continue
14:     return
15:  C
16:     110 continue
17: C2003 IEND      = IST + 7
18: C2003 NUCID     = LINE(IST:IEND)
19:     JST         = 0
20:     JEND        = 0
21:     do 120 I = 1, 72
22: C2003     if ( LINE(I:I).eq.'(' ) JST      = I + 1
23:         if ( LINE(I:I).eq.'(' ) then
24:             call NOBLNK( LINE,IST,IEND,I-1)
25:             NUCID      = LINE(IST:IEND)
26: C2003 ... end
27:             JST        = I + 1
28:         endif
29:         if ( LINE(I:I).eq.')' ) then
30:             JEND        = I - 1
31:             go to 130
32:         end if
33:     120 continue
34:     return
35:  C
36:     130 continue
37:     if ( JEND.ge.JST.and.JST.gt.0 ) then
38:         call READR4(LINE,JST,JEND,ANS)
39:     end if
40:  C
41:     return
42: end
```


src/mvpburn/getln2.f

```
1:      subroutine GETLN2( IUIN, IUPR, LINE, NCHAR, IEND, ICNT )
2: C
3: C      GMVP/MVP UTILITY
4: C      copy from ../utils/getlin.f & modified
5: C
6: C=====
7: C PURPOSE: INPUT A LINE FROM UNIT IUIN.
8: C      (SKIP COMMENT LINE & COMMENT CHARACTERS)
9: C      IUPR : message printout I/O unit
10: C      IEND = -1 / 0 : end of input / no
11: C      NCHAR : effective characters as input. ( 0 TO LEN(LINE))
12: C CALLED IN: FREAD & MANY PLACE
13: C HISTORY: PROGRAMMED BY M.SASAKI ( 13 APR 1992 )
14: C UPDATE:
15: C 11 FEB 1993 : Handle symbolic parameter line (%) in this routine.
16: C 19 APR 1993 : set input character from '/' to blanks.
17: C 4 Aug 1995: return IEND = -1 for end of file. Other routines
18: C      may return 1 as "end of input" flags.
19: C 14 Apr 1998: added argument IUPR and pass it to PARA.
20: C 09 Nov 1999: add line count for mvpburn
21: C=====
22:      character*(*) LINE
23: C
24: C      .... DATA FOR VIRTUAL FILE .
25: C      ( THE SAME COMMON DATA IN FREAD ROUTINE )
26:      parameter( MXVLIN   = 32, MAXCOL   = 72 )
27:      common /VFREAD/ IOSAVE, ISSAVE, NVLINE, IVLINE
28:      common /VFILE/  LNSAVE
29:      character*(MAXCOL) LNSAVE(0:MXVLIN)
30: C
31:      IEND   = 0
32:      NCHAR  = 0
33:      LINE   = ' '
34: C
35: 100 if ( IUIN.gt.0 ) then
36:      read(IUIN,'(A)',end =110) LINE
37: C
38:      ICNT = ICNT + 1
39:      else
40:      IVLINE = IVLINE + 1
41:      if ( IVLINE.gt.NVLINE ) go to 110
42:      LINE   = LNSAVE(IVLINE)
43:      end if
44: C
45: C      .... COMMENT .....
46: C
47: C      if ( LINE(1:1).eq.'*' ) go to 100
48: C      if ( LINE(1:1).eq.'*' ) return
49: C
50: C      .... SYMBOLIC PARAMETER DEFINITION ....
51: C
52:      if ( LINE(1:1).eq.'%' ) then
53:          call PARA( LINE, IUPR )
54:          go to 100
55:      end if
56: C
57:      K      = INDEX(LINE,'/*')
58:      if ( K.eq.0 ) then
59:          NCHAR = LEN(LINE)
60:      else
61:          NCHAR = K - 1
62:          LINE(K:) = ' '
63:      end if
64:      if ( NCHAR.eq.0 ) go to 100
65:      return
66: C
67: 110 IEND   = -1
68:      return
69:      end
```

src/mvpburn/getmic.f

```
1:      subroutine GETMIC( LINE,  IANS )
2: C
3:      character*72 LINE
4:      character*1 IBRA, IKETO
5: C
6:      data IBRA, IKETO /'(', ')'/
7: C
8:      do 100 I = 1, 63
9:          I1      = I
10:         I2      = I + 9
11:         ISW      = INDEX(LINE(I1:I2), 'EDIT-MICRO')
12:         if ( ISW.eq.1 ) go to 110
13: 100 continue
14:         IANS      = 0
15:         return
16: C
17: 110 continue
18:         IST      = 0
19:         IEND      = 0
20:         do 120 I = I1 + 4, 72
21:             if ( LINE(I:I).eq.IBRA ) IST      = I + 1
22:             if ( LINE(I:I).eq.IKETO ) then
23:                 IEND      = I - 1
24:                 go to 130
25:             end if
26: 120 continue
27:         IANS      = 0
28:         return
29: C
30: 130 call READI4(LINE,IST,IEND,IANS)
31: C
32:         return
33:     end
```

src/mvpburn/getmid.f

```
1:      subroutine GETMID( LINE,  IDENT )
2:      C
3:      character*72 LINE
4:      character*(*) IDENT
5:      C
6:      C
7:      C
8:      IDENT   = ' '
9:      C2003
10:     C
11:     *      do 100 I = 1, 72
12:     *          IST      = I
13:     *          if ( LINE(I:I).ne.' ' ) go to 110
14:     * 100 continue
15:     *      return
16:     C
17:     * 110 continue
18:     *          IEND      = IST + 7
19:     *          IDENT     = LINE(IST:IEND)
20:     C2003
21:     K = index(LINE,'(')
22:     IS = 1
23:     call NOBLNK(LINE,IS,IE,K-1)
24:     IDENT = LINE(IS:IE)
25: check
26: CCC write(6,*) 'check %% GETMID <' ,IDENT,'>'
27: return
28: end
```

src/mvpburn/gettmp.f

```
1:      subroutine GETTMP( LINE,  ANS )
2: C
3:      character*72 LINE
4:      character*1 IBRA, IKETO
5: C
6:      data IBRA, IKETO /'(', ')'/
7: C
8:      do 100 I = 1, 72
9:          I1      = I
10:         I2      = I + 3
11:         ISW     = INDEX(LINE(I1:I2), 'TEMP')
12:         if ( ISW.eq.1 ) go to 110
13: 100 continue
14:         ANS     = 0.0
15:         return
16: C
17: 110 continue
18:         IST     = 0
19:         IEND    = 0
20:         do 120 I = I1 + 4, 72
21:             if ( LINE(I:I).eq.IBRA ) IST  = I + 1
22:             if ( LINE(I:I).eq.IKETO ) then
23:                 IEND  = I - 1
24:                 go to 130
25:             end if
26: 120 continue
27:         ANS     = 0.0
28:         return
29: C
30: 130 call READR4(LINE,IST,IEND,ANS)
31:     return
32:     end
```

src/mvpburn/gettrg.f

```
1:      subroutine GETTRG( LINE,  CANS,  LENG )
2:  C
3:      character*72 LINE
4:      character*1 IBRA, IKETO
5:      character*12 CANS
6:  C
7:      data IBRA, IKETO /'@', '/' /
8:  C
9:      CANS      = ' '
10:     LENG      = 0
11:  C
12:     do 100 I = 1, 72
13:         I1      = I
14:         I2      = I + 5
15:         ISW      = INDEX(LINE(I1:I2), 'TRGNAM')
16:         if ( ISW.eq.1 ) go to 110
17:     100 continue
18:     return
19:  C
20:     110 continue
21:         IST      = 0
22:         IEND      = 0
23:         do 120 I = I1 + 4, 72
24:             if ( LINE(I:I).eq.IBRA ) IST      = I
25:             if ( IST.gt.0 ) then
26:                 if ( LINE(I:I).eq.IKETO .or. LINE(I:I).eq.'/' ) then
27:                     IEND      = I - 1
28:                     go to 130
29:                 end if
30:             end if
31:     120 continue
32:     return
33:  C
34:     130 LENG      = IEND - IST + 1
35:     CANS(1:LENG)  = LINE(IST:IEND)
36:  C
37:     return
38:     end
```

src/mvpburn/getvol.f

```
1:      subroutine GETVOL( LINE,  ANS )
2: C
3:      character*72 LINE
4:      character*1 IBRA, IKETO
5: C
6:      data IBRA, IKETO /'(', ')/
7: C
8:      do 100 I = 1, 72
9:          I1      = I
10:         I2      = I + 3
11:         ISW      = INDEX(LINE(I1:I2), 'VOLM')
12:         if ( ISW.eq.1 ) go to 110
13: 100 continue
14:         ANS      = 0.0
15:         return
16: C
17: 110 continue
18:         IST      = 0
19:         IEND      = 0
20:         do 120 I = I1 + 4, 72
21:             if ( LINE(I:I).eq.IBRA ) IST      = I + 1
22:             if ( LINE(I:I).eq.IKETO ) then
23:                 IEND      = I - 1
24:                 go to 130
25:             end if
26: 120 continue
27:         ANS      = 0.0
28:         return
29: C
30: 130 call readR4(LINE,IST,IEND,ANS)
31:     return
32:     end
```

src/mvpburn/gtvval.f

```
1:      subroutine GTVVAL( VNAME, VALUE, IERR )
2: C=<MVP>=====
3: C  PURPOSE: RETURN THE VALUE OF AN INTERNAL VARIABLE NAMED VNAME:
4: C  CALLED IN:  DENTAK, PARA & OTHERS
5: C  ARGUMENTS:
6: C      VNAME : VARIABLE NAME (NOT VARIABLE ITSELF !)
7: C      VALUE : VALUE (EXPANDED TO DOUBLE PRECISION IF NECESSARY)
8: C      IERR  : ERROR CODE
9: C              0 : SUCCESSFUL
10: C             1 : NON-EXISTENT VARIABLE NAME OR VALUE FETCHING
11: C              NOT ALLOWED.
12: C
13: C             2 : PROBABLY VALUE UNDEFINED YET
14: C
15: C CAUTION: THIS ROUTINE IS CALLED FROM UTILITY ROUTINE DENTAK & PARAM
16: C          BUT CODE-DEPENDENT (MVP & GMVP USE DIFFRENT SOURCE FOR
17: C          THIS ROUTINE.
18: C=====
19:      character*(*) VNAME
20:      real*8 VALUE
21:      integer IERR
22: C
23: C ----- Common data of MVP-BURN -----
24: C
25: C --- array sizes -----
26: C      include 'INC/_BURNP'
27: C
28: C -- Common data -----
29: C
30: C      include 'INC/_CBURN1'
31: C      include 'INC/_CBURN2'
32: C
33: C -----
34: C
35: C      real*8 V0
36: C
37: C      .... VJ : A JUNK VALUE TO CONFIRM VALUE ASSIGNMENT ....
38: C
39: C      real*8 VJ
40: C      parameter( VJ  = -9.9876543D38 )
41: C
42: C      IERR   = 0
43: C      V0     = VJ
44: C
45: C  AVAILABLE VRIABLES :   ( 16 MAR 1992 )
46: C
47: C      if ( VNAME.eq.'NSTEP' ) V0   = NEP
48: C      if ( VNAME.eq.'NFIS' ) V0   = NFIS
49: C      if ( VNAME.eq.'NFER' ) V0   = NFER
50: C
51: C .... VERIFICATION ...
52: C
53: C 100 continue
54: C      if ( V0.ne.VJ ) then
55: C          VALUE = V0
56: C          IERR   = 0
57: C      else
58: C          IERR   = 1
59: C      end if
60: C      return
61: C      end
```

src/mvpburn/holpos.f

```
1:      subroutine HOLPOS( IPOS,  IDENT, MAXNUB,IHOL )
2:  C
3:      common /REAMIO/  NOUT6,  NDTLS
4:  C
5:      character*8 IDENT
6:      character*8 IHOL(MAXNUB)
7:  C
8:      IPOS      = -1
9:      do 100 I = 1, MAXNUB
10:         II      = I
11:         if ( IDENT.eq.IHOL(I) ) go to 110
12:      100 continue
13:      write(NOUT6,7000) IDENT
14:      stop 888
15:  C
16:  6888 format(// ' XXX(HOLPOS) ERROR:STOP :' )
17:  7000 format( 1X,A8,' is undefined.')
18:  C
19:  110 IPOS      = II
20:      return
21:      end
```


src/mvpburn/lchain.f

```

1:      subroutine LNCHAI( NUCL,  RAMDA, NCH,   NUCLP, NBIC,  PBIC,  NOL,
2:      &                  KSTP,  LONG,  LL,    IP,    KP,    NPN,  NMAX,
3:      &                  NPAR,  NCHA,  LCHA,  ITYP )
4: C
5:      common /REAMIO/  NOUT6,  NDTLS
6: C
7:      integer NUCL(NMAX)
8:      real RAMDA(NMAX)
9:      integer NCH(NMAX), NUCLP(NPAR,NMAX)
10:     integer NBIC(NPAR,NMAX)
11:     real PBIC(NPAR,NMAX)
12:     integer NOL(NMAX), KSTP(NCHA,NMAX), LONG(NCHA,NMAX),
13:     &      LL(NCHA,NMAX), IP(LCHA,NCHA,NMAX), KP(LCHA,NCHA,NMAX),
14:     &      NPN(NPAR,NMAX)
15: C
16: C      LINEAR CHAIN CALCULATION
17: C
18: C
19: C      NUCL      CODE NO. OF NUCLIDE
20: C      RAMDA     DECAY CONSTANT
21: C      NCH       THE NUMBER OF PARENTS
22: C      NUCLP     CODE NO. OF PARENTS
23: C      NBIC      REACTION TYPE INDICATOR
24: C              1  DECAY
25: C              3  N-GAMMA
26: C      PBIC      BRANCHING RATIO
27: C      NOL       THE NUMBER OF LINEAR CHAINS
28: C      KSTP      INDICATOR TO SPECIFY THE TOP NUCLIDE OF LINEAR CHAIN
29: C      LONG      LENGTH OF LINEAR CHAIN
30: C      LL        INDICATOR TO POINT STARTING NUCLIDE
31: C      IP        NUCLIDE SEQUENCE NUMBER OF LINEAR CHAIN
32: C      KP        BRANCHING INDICATOR IN LINEAR CHAIN
33: C      NPN       NUCLIDE SEQUENCE NUMBER OF PARENT
34: C      NMAX      THE NUMBER OF NULIDES
35: C      NPAR      MAXIMUM NUMBER OF PARENTS
36: C      NCHA      MAXIMUM NUMBER OF LINEAR CHAINS
37: C      LCHA      MAXIMUM LENGTH OF LINEAR CHAIN
38: C      ITYP      CHAIN TYPE
39: C              1 : HOT (DECAYS AND NEUTRON REACTIONS)
40: C              2 : COLD (DECAYS ONLY)
41: C
42: C      PROLOGUE
43: C
44:      do 120 N = 1, NMAX
45:        NCHN = NCH(N)
46:        if ( NCHN.eq.0 ) then
47:          NPN(1,N) = 0
48:        else
49:          do 110 M = 1, NCHN
50:            if ( NBIC(M,N).eq.3 .or. NBIC(M,N).ge.7 ) NBIC(M,N) =
51:            &      -NBIC(M,N)
52:            do 100 NN = 1, NMAX
53:              if ( NUCLP(M,N).eq.NUCL(NN) ) then
54:                NPN(M,N) = NN
55:                if ( PBIC(M,N).eq.0.0 .or. RAMDA(NN).eq.0.0 )
56:                &      NPN(M,N) = -NPN(M,N)
57:                go to 110
58:              end if
59:            100      continue
60:            NPN(M,N) = 0
61:          110      continue
62:        end if
63:      120 continue
64: C
65:      do 150 NN = 1, NMAX

```

```

66:      do 140 J = 1, NCHA
67:        LL(J,NN) = 0
68:        KSTP(J,NN) = 0
69:        do 130 L = 1, LCHA
70:          IP(L,J,NN) = 0
71:          KP(L,J,NN) = 0
72:        130      continue
73:        140      continue
74:        150      continue
75: C
76: C      CHAIN CALCULATION
77: C
78: C2005 ... added
79:      NOS = 0
80: C2005 ... end
81:      do 230 NN = 1, NMAX
82:        I = 1
83:        K = 1
84:        NOSN = 1
85:        IP(1,1,NN) = NN
86: C
87:      do 210 K = 1, LCHA
88:        NOS = NOSN
89:        MSTP = 0
90:        do 200 I = 1, NOS
91:          if ( IABS(KSTP(I,NN)).ne.1 ) then
92:            MSTP = 1
93:            NCL = IP(K,I,NN)
94:            if ( K.ne.1 ) then
95:              KM1 = K - 1
96:              do 160 KX = 1, KM1
97:                if ( NCL.eq.IP(KX,I,NN) ) go to 190
98:              160      continue
99:            end if
100:            if ( NCH(NCL).gt.0.and.K.lt.LCHA ) then
101: C
102: C
103: C
104:              J1 = NOSN
105:              NOSN = NOSN + NCH(NCL) - 1
106:              if ( NOSN.gt.NCHA ) go to 290
107:              J2 = NOSN
108:              if ( J1.le.J2 ) then
109:                JZ = 0
110:                do 180 JJ = J1, J2
111:                  JZ = JZ + 1
112:                  J = JJ
113:                  if ( JZ.ne.1 ) then
114:                    J = JJ
115:                    LL(J,NN) = K
116:                    do 170 L = 1, K
117:                      IP(L,J,NN) = IP(L,I,NN)
118:                      if ( L.ne.K ) KP(L,J,NN) =
119:                      &      KP(L,I,NN)
120:                    170      continue
121:                  end if
122:                  NB = NBIC(JZ,NCL)
123: C
124:                  if ( NPN(JZ,NCL).ne.0
125:                  &      .and.(NB.le.0.or.NPN(JZ,NCL).ge.0) ) then
126:                    if ( NB.gt.0 .or. ITYP.eq.1 ) then
127:                      IP(K+1,J,NN) = IABS(NPN(JZ,NCL))
128:                      KP(K,J,NN) = JZ
129:                      go to 180
130:                    end if

```

src/mvpburn/lchai.f

```

131:                end if
132:                KSTP(J,NN) = 1
133:                if ( NB.lt.0 ) KSTP(J,NN) = -1
134:                LONG(J,NN) = K
135: C
136: 180                continue
137:                end if
138:                go to 200
139:                end if
140: 190                KSTP(I,NN) = 1
141:                LONG(I,NN) = K
142:                end if
143: 200                continue
144:                if ( MSTP.eq.0 ) go to 220
145: 210                continue
146:
147: C                if ( MSTP.eq.0 ) then
148: C                    write(NOUT6,6888)
149: C                    write(NOUT6,129) NUCL(NN), LCHA
150: C 129                format(1X,'NUCL:',
151: C &                    'I8/1X,' chain length is over LCHA. :LCHA= ',I5)
152: C                    stop 888
153: C                end if
154:
155: C
156: 220                continue
157:                NOL(NN) = NOS
158: C
159: 230                continue
160: C
161: C                EPILOGUE
162: C
163:                do 260 NN = 1, NMAX
164:                NOS = NOL(NN)
165:                do 250 I = 1, NOS
166:                LL(I,NN) = LONG(I,NN) - LL(I,NN)
167:                if ( LONG(I,NN).gt.0 ) then
168:                LLL2 = (LONG(I,NN)-1) /2 + 1
169:                LLL1 = LONG(I,NN) + 1
170:                do 240 L = 1, LLL2
171:                LLL1 = LLL1 - 1
172:                IWORK = IP(L,I,NN)
173:                IP(L,I,NN) = IP(LLl1,I,NN)
174:                IP(LLl1,I,NN) = IWORK
175:                IWORK = KP(L,I,NN)
176:                KP(L,I,NN) = KP(LLl1,I,NN)
177:                KP(LLl1,I,NN) = IWORK
178: 240                continue
179:                end if
180: 250                continue
181: 260                continue
182: C
183:                do 280 I = 1, NMAX
184:                do 270 J = 1, NPAR
185:                NBIC(J,I) = IABS(NBIC(J,I))
186:                NPN(J,I) = IABS(NPN(J,I))
187: 270                continue
188: 280                continue
189:                return
190: C
191: C                ERROR MESSAGE SECTION
192: C
193: 290                write(NOUT6,6888)
194:                write(NOUT6,7020) NUCL(NN), NOSN, NCHA
195: 6888                format(// ' XXX(LNCHAI) ERROR-STOP : ' )

196: 7020                format(1X,'NUCL(code No of nuclide):',I8/1X,
197:                &                ' the number of linear chains (',I5,',') over NCHA ',
198:                &                ' : NCHA = ',I5)
199:                stop 888
200:                end

```

src/mvpburn/logopr.f

```
1: C*****
2: C      PRINT DATE & TIME & CODE NAME & VERSION NO.
3: C*****
4:      SUBROUTINE LOGOPR(NOUT1)
5: C=====
6: C
7: C+++  PRINT LOGO (MVP-BURN)
8:      write(NOUT1,*)
9:      write(NOUT1,*)
10:     write(NOUT1,6001)
11:     write(NOUT1,6002)
12:     write(NOUT1,6003)
13:     write(NOUT1,6004)
14:     write(NOUT1,6005)
15:     write(NOUT1,6006)
16:     write(NOUT1,6007)
17:     write(NOUT1,6008)
18:     write(NOUT1,6009)
19:     write(NOUT1,6010)
20:     write(NOUT1,6011)
21: C*****
22: 6001 format(15X,'_/_/_/_/_/_/_/_/_/_/_/_/_/_')
23: 6002 format(14X,'_/_/_/_/_/_/_/_/_/_/_/_/_')
24: 6003 format(13X,'_/_/_/_/_/_/_/_/_/_/_/_')
25: 6004 format(12X,'_/_/_/_/_/_/_/_/_/_/_')
26: 6005 format(11X,'_/_/_/_/_/_/_/_/_/_')
27: 6006 format(10X,'_/_/_/_/_/_/_/_/_')
28: 6007 format(9X,'_/_/_/_/_/_/_/_/_ -BURN')
29: 6008 format(1X)
30: 6009 format(/9X,'VERSION : BURN v2.33      RELEASE DATE : ',
31: &          '14 JUN. 2006')
32: 6010 format(9X,'LAST MODIFIED DATE : 14 JUN. 2006')
33: 6011 format(/)
34: C
35:      RETURN
36:      END
```

src/mvpburn/main.f

```

1: *VOCL TOTAL,SCALAR
2: C
3: C   JAEA MONTE CARLO BURN-UP CALCULATION CODE. MAIN ROUTINE
4: C
5: C
6: C
7: C
8: C
9: C
10: C
11: C
12: C
13: C
14: C
15: C
16: C
17: C
18: C
19: C
20: C
21: C
22: C
23: C   Burn-up calculation code with the continuous energy Monte Carlo
24: C   code MVP of (JAEA)
25: C
26: C -----
27: C   Development staff
28: C -----
29: C       designed and programed by Okumura K.(JAEA)
30: C       programing supportted by Kaneko K.
31: C                               and Sasaki M.
32: C                               and Ido M.
33: C
34: C
35: C/#IF CMAIN
36: *   subroutine FTMAIN
37: C/#ELSE
38:   program MVPBURN
39: C/#ENDIF
40: C
41: C/#IF MS_VISUAL
42: C ... enable to use iargc, getarg for Visual Fortran.
43: *   use dflib
44: *   use dfport
45: C/#ENDIF
46: C
47:   character*256 MVPEX
48:   character*256 MVPIDX
49:   character*128 PDSIN
50:   character*128 PDSOUT
51: C
52:   character*128 CSTRNG
53: C
54: C -----
55: C
56:   MVPEX = 'mvp'
57:   MVPIDX = ' '
58:   PDSIN = ' '
59:   PDSOUT = ' '
60:   MVPDBG = 0
61: C
62: C ... check environment variables if possible ...
63: C
64: C (FUNBUF may be called by checking environment variable)
65: C

```

```

66: C
67: C/#IF .NOT.NOGETENV
68:   call CHKENV('MVPBURN', MVPEX, MVPIDX, PDSIN, PDSOUT, MVPDBG)
69: C/#ENDIF
70: C
71: C
72:   call WDBL
73: C
74: C   ... set default input/print I/O unit of FREAD routines.
75: C
76:   call FRINIT(5,6,10)
77: C
78: C   ... set error counter for CNTERR routines.
79: C
80:   call ERINIT
81: C
82:   IPREIN = 0
83: C
84: C -----
85: C   Program control information from command line arguments.
86: C -----
87: C
88: C ... ARGNONE : FAT flag indicating no means to get command argument.
89: C
90: C/#IF ARGV
91: C
92: C/# IF ARGV( GETARG2 )
93: *   KJ = IARGC() - 1
94: C/# ELSEIF ARGV( MSF )
95: *   KJ = NARGS() - 1
96: C/# ELSE
97:   KJ = IARGC()
98: C/# ENDIF
99: C
100:   do I = 1, KJ
101:
102:     IPREIN = IPREIN + 1
103: C/# IF SYSTEM( CRAY )
104: *   call PXFGETARG(I, CSTRNG, LLLLL, ierr )
105: C/# ELSEIF ARGV( IGETARG )
106: *   LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
107: C/# ELSE
108: C/# IF ARGV( GETARG2 )
109: *   call GETARG(I+1, CSTRNG )
110: C/# ELSEIF ARGV( HPF90 )
111: *   call GETARG(I+1, CSTRNG )
112: C/# ELSEIF ARGV( MSF )
113: C ... LL2 should be integer*2
114: *   call GETARG(I, CSTRNG, LL2 )
115: C/# ELSE
116:   call GETARG( I, CSTRNG )
117: C/# ENDIF
118:   LLLLL = MIN( INDEX(CSTRNG, ' ') - 1, LEN(CSTRNG) )
119:   if ( LLLLL.eq.0 ) LLLLL = LEN(CSTRNG)
120: C/# ENDIF
121: C
122: C ... quit command line check if '@@' is encountered ...
123: C
124:   if ( CSTRNG(1:LLLLL) .eq. '@@' ) goto 105
125: C
126:   if ( CSTRNG(1:1).eq.'/' ) then
127:     JJJJJ = JFOPEN(CSTRNG(:LLLLL))
128:     if (JJJJJ .ne. 0) then
129:       write(6,*)
130:       write(6,*)

```

src/mvpburn/main.f

```
131:         write(6,*) ' XXX(MAIN) ERROR-STOP :',
132:         &         'file open error at the first step'
133:         write(6,*) ' File : ',CSTRNG(:LLLLL)
134:         stop 999
135:     end if
136:     else if ( CSTRNG(1:4).eq.'mvp='.or.
137:     &         CSTRNG(1:4).eq.'MVP=' ) then
138:         write(*,*) ' >> MVP executable is <',CSTRNG(5:11111),'>'
139:         mvpex = CSTRNG(5:11111)
140:     else if ( CSTRNG(1:5).eq.'nidx='.or.
141:     &         CSTRNG(1:5).eq.'NIDX=' ) then
142:         write(*,*) ' >> MVP lib index is <',CSTRNG(6:11111),'>'
143:         mvpidx = CSTRNG(6:11111)
144:     else
145:         write(*,*) '!!! command argument <',CSTRNG(:LLLLL),'> is '
146:         &         'ignored.'
147:     end if
148: end do
149: C
150: 105 continue
151: C
152: if ( KJ.gt.0 ) write(6,(''1''))
153:
154: C ... END OF C/#IF ARGV ...
155: C/#ENDIF
156: C
157: C-----
158: C ... pre-setting of SIGINT event handler that flushes standard
159: C output.
160: C-----
161: C
162: C/#IF FLOATHANDLER( ONSTMT )
163: C
164: C === Floating point error handler specification by "ON" statement
165: C (Specific to HP fortran)
166: C ---- TO PREVENT UNDERFLOW ABORT in trapping mode (+T option)---
167: C
168: * ON REAL*4 UNDERFLOW CALL UNDERF4
169: * ON REAL*8 UNDERFLOW CALL UNDERF8
170: C/#ENDIF
171:
172: C/#IF UNIX
173: call FSIGSET()
174: C/#ENDIF
175: C
176: C-----
177: C Processing Starts (call MVPBRN)
178: C-----
179: C
180: call MVPBRN(MVPPEX, MVPIDX, PDSIN, PDSOUT, MVPDBG)
181: C
182: stop
183: end
184: C
185: C === Floating point error handler specified by "ON" statement
186: C (Specific to HP fortran)
187: C
188: C
189: C ... underflow handling ...
190: C
191: subroutine UNDERF4( R4 )
192: real R4
193: R4 = 0.0
194: return
195: end
196: C
197: C
198: subroutine UNDERF8( R8 )
199: real*8 R8
200: R8 = 0.0D0
201: return
202: end
```

src/mvpburn/memove.f

```
1:      subroutine MEMOVE( A,      B,      N )
2: C
3: C      ARRAY MOVE FROM B TO A
4: C
5:      real A(1), B(1)
6: C
7:      do 100 I = 1, N
8:          B(I)      = A(I)
9:      100 continue
10: C
11:      return
12:      end
```

SAFE

src/mvpburn/mvpbrn.f

```

1:      subroutine MVPBRN( MVPEX, MVPIDX, PDSIN, PDSOUT, MVPDBG)
2: C=====
3: C      Control routine for MVP-BURN code
4: C
5: C      DEVICE INFORMATION
6: C
7: C      UNIT NO      FORMAT      CONTENT
8: C      -----
9: C          5          FORMATTED  STANDARD INPUT
10: C          6          FORMATTED  PRINT OUTPUT FOR THIS CODE (RESULT)
11: C          30         BINARY     PLOT FILE FROM MVP-RUN
12: C          40         BINARY     WORK FILE FOR AVERAGING PROCESS
13: C          50         FORMATTED  SRAC-95 CHAIN LIBRARY
14: C          60         BINARY     UNIT NO OF PDS-FILES FOR UNIX-MACHINE
15: C          61         text       UNIT NO OF text virtual PDS-FILES
16: C          62         binary     UNIT NO OF binary virtual PDS-FILES
17: C          63         text       UNIT NO OF text virtual PDS-FILES
18: C          (86)        FORMATTED  UNIT FOR TEST CALCULATION WITHOUT MVP
19: C                               (not used now)
20: C          92         FORMATTED  ORIGINAL MVP INPUT USED ONLY BURNIN MODULE
21: C          93         FORMATTED  WORK FILE IN BURNIN MODULE and MVPIN
22: C                               OUTPUT DATA FOR MVP-INPUT IN MVPINP MODE
23: C          95         FORMATTED  WORK FILE FOR FREE FORMAT READ ROUTINE
24: C          99         FORMATTED  STANDARD OUTPUT (LOG)
25: C          DIRIN      BINARY     DDNAME FOR INPUT PDS-FILE (IFACOM-M780)
26: C          DIROUT     BINARY     DDNAME FOR OUTPUT PDS-FILE (IFACOM-M780)
27: C      -----
28: C
29: C=====
30: C      character*(*) MVPEX
31: C      character*(*) MVPIDX
32: C      character*(*) PDSIN, PDSOUT
33: C
34: C      common /MACHTY/  IFACOM
35: C
36: C      ===== Memory area allocation =====
37: C
38: C      .... Non character data .....
39: C
40: C      include 'INC/_MEMRY'
41: C      CKSK parameter( MAXMEM = 8000000 )
42: C      real ARRAY(MAXMEM)
43: C      common /MAINM/   ARRAY
44: C
45: C      ... system limit definition such as ;
46: C
47: C      include 'INC/_BURNP'
48: C
49: C      .... character data .....
50: C
51: C      include 'INC/_MVPBRN'
52: C
53: C      character*12 TRGNAM
54: C2003
55: C      character*12 IDENT
56: C2003
57: C      character*12 IDWORK
58: C      common /MAINC/ TRGNAM(MAXTRG), IDENT(MAXISO,MAXTRG),
59: C      & IDWORK(MAXISO,MAXTRG)
60: C
61: C      ... step dependent MVP symbolic parameter ...
62: C
63: C      PARANM(*) : symbolic parameter name to be replaced
64: C      PARATP(*) : data type of parameter
65: C      ( 'F': float, 'I': integer )

```

```

66: C      NMVPPR      : number of symbolic parameters
67: C      PARAV(1:nstep,*) : data value of symbolic parameter
68: C
69: C      character*16 PARANM
70: C      character*1 PARATP
71: C      common /STPPRN/ PARANM(MAXSPR),PARATP(MAXSPR)
72: C      real*8 PARAV
73: C      common /STPPRV/ PARAV(MXSTEP,MAXSPR), NMVPPR
74: C
75: C      HMINV0 : initial heavy nuclide weight (Ton) of user input
76: C
77: C      real HMINV0(MAXTRG)
78: C
79: C      =====
80: C
81: C      common /REAMIO/  NOUT6,  NDTLS
82: C      common /IOPRNT/  NOUT1,  NOUT2
83: C      common /MEMOCK/  MEUSED, MEMMAX
84: C
85: C      -----
86: C
87: C      Variables of Fixed size
88: C
89: C      character*128 DIRIN, DIROUT
90: C2005 character*72 TITLE(2)
91: C      character*8 MODE
92: C      character*4 CASEID
93: C      character*8 IDATE
94: C      character*8 STIME
95: C
96: C      logical OPD, NMD
97: C
98: C      -----
99: C
100: C      character*72 VNAME
101: C
102: C
103: C      T0      = 0.0
104: C
105: C      .... input I/O unit for "FREAD" routines
106: C
107: C      NIOFR    = 55
108: C
109: C      .... input I/O unit for "REAM" or ordinary READ statements
110: C      for module routines
111: C
112: C      NDTLS    = 95
113: C
114: C      DIRIN = PDSIN
115: C      DIROUT = PDSOUT
116: C
117: C      NOUT1    = 6
118: C      NOUT2    = 6
119: C      NOUT6    = 6
120: C
121: C      ... MVP input data is separated from stdin and store on this unit
122: C
123: C      LMVPI = 92
124: C
125: C *** IN M-780 SYSTEM : IFACOM = 1
126: C *** IN UNIX  SYSTEM : IFACOM = 0
127: C
128: C      IFACOM = 0
129: C
130: C      call CPUTM2( T0 )

```

src/mvpburn/mvpbrn.f

```

131:      call UDATE( IDATE )
132: C
133:      call JIKAN( STIME )
134: C
135: C --- list and copy input data to I/O unit NDTLS ---
136: C MVP input data is extracted in this routine
137: C
138:      call DTLIST( 5, NIOFR, LMVPI, NOUT1, 0 ,MVPDBG )
139: C
140:      call LOGOPR(NOUT1)
141:      write(NOUT1,8000)
142:      write(NOUT1,7020) IDATE, STIME
143:      write(NOUT1,8000)
144:      write(NOUT1,*)
145:      MEMORY = MAXMEM
146:      MEMMAX = MAXMEM
147:      MEUSED = 0
148: C
149: C --- SET NMAT & KNMAX
150:      NMAT = 0
151:      KNMAX = 0
152:      NCARD = 0
153: C
154:      NMVPPR = 0
155: C
156: C --- change input I/O unit of FREAD routines ---
157: C
158:      call RIUNIT( NIOFR )
159: C
160: C-----
161: C
162:      if (MXSTEP.gt.99) then
163:          write(NOUT1,6888)
164:          write(NOUT1,*) ' limit for number of burn-up step is 99'
165:          write(NOUT1,*) ' change parameter value(MXSTEP) in include file'
166:          stop 999
167:      endif
168: C
169: C-----
170: C Check the path of neutron library index file (may be opened
171: C on I/O unit 25
172: C-----
173: C
174:      if ( MVPIDX.eq.' ' ) then
175:          inquire(25,opened =OPD,named =NMD)
176:          if ( .not.OPD ) then
177:              write(NOUT1,6888)
178:              write(NOUT1,*) ' MVP neutron index file should',
179:              & ' be opened on I/O unit 25 ',
180:              & ' or given by environment variable MVPBURN_NIDX.'
181:              stop 999
182:          end if
183:          if ( .not.NMD ) then
184:              write(NOUT1,6888)
185:              write(NOUT1,*) ' MVP neutron index file ',
186:              & ' is opened on I/O unit 25 ',
187:              & ' but it is not a named file.'
188:              stop 999
189:          end if
190:          inquire(25,name =MVPIDX)
191:      end if
192: C
193: C
194: C --- initialization of some variables
195: C

```

```

196:      IBURNI = 0
197:      CPULIM = 0.0
198: C
199: C-----LOOP OF MODE
200: C
201:      100 continue
202: C
203:      call FREADS( VNAME, NLEN, IEND )
204:      if ( IEND.ne.0 ) go to 110
205: C
206: C-----
207: C ..... $BURNUP .....
208: C-----
209: C
210:      if ( VNAME.eq.'$BURNUP' ) then
211:          MODE = '$BURNUP'
212:          IBURNI = IBURNI + 1
213:          call AUTOBN( MODE,
214:          & IBURNI, T0, CPULIM, ARRAY, MEMORY,
215:          & TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
216:          & PARAMN,PARATP,PARAV,NMVPPR,MAXSPR,
217:          & MVPEX, MVPIDX,
218:          & NOUT1, NOUT2, DIRIN, DIROUT, CASEID , MVPDBG ,
219:          & HMINV0 )
220: C
221: C-----
222: C-----
223: C ..... $BRANCH .....
224: C-----
225: C
226:      else if ( VNAME.eq.'$BRANCH' ) then
227:          MODE = '$BRANCH'
228:          IBURNI = IBURNI + 1
229:          call AUTOBN( MODE,
230:          & IBURNI, T0, CPULIM, ARRAY, MEMORY,
231:          & TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
232:          & PARAMN,PARATP,PARAV,NMVPPR,MAXSPR,
233:          & MVPEX, MVPIDX,
234:          & NOUT1, NOUT2, DIRIN, DIROUT, CASEID , MVPDBG,
235:          & HMINV0 )
236: C
237: C-----
238: C ..... $AVERAGE.....
239: C-----
240: C
241:      else if ( VNAME.eq.'$AVERAGE' ) then
242:          MODE = '$AVERAGE'
243:          call AVGIN0( MODE,
244:          & T0, ARRAY, MEMORY,
245:          & TRGNAM, IDENT, MAXTRG, MAXISO,
246:          & NOUT1, NOUT2, DIROUT, CASEID )
247:          & NOUT1, NOUT2, DIROUT, CASEID , DIRIN )
248: C
249: C-----
250: C ..... $SUMMARY .....
251: C-----
252: C
253:      else if ( VNAME.eq.'$SUMMARY' ) then
254:          MODE = '$SUMMARY'
255:          call SMRYIN( MODE, ARRAY, MEMORY,
256:          & TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
257:          & NOUT1, NOUT2, DIRIN, DIROUT, CASEID )
258: C
259: C-----
260: C ..... What is it ? .....

```


src/mvpburn/mvpbrn.f

```
261: C-----
262: C
263:     else
264:         write(nout1,6888)
265:         write(nout1,7000) vname(:iclen2(vname))
266:         stop 888
267:     end if
268: C
269:     ITIME = 0
270:     call JIKAN( STIME )
271:     call CPUTM2( T1 )
272:     write(NOUT1,7080) MODE, IDATE, STIME
273:     write(NOUT1,7040) (T1-T0)/60.0, MEMUSED+1
274: C
275:     go to 100
276: C
277: 110 ITIME = 0
278:     call JIKAN( STIME )
279:     call CPUTM2(T1)
280:     write(NOUT1,*)
281:     write(NOUT1,8000)
282:     write(NOUT1,7100) IDATE, STIME
283:     write(NOUT1,8000)
284: C
285: 6888 format(// ' XXX(MVPBRN) ERROR-STOP : ' )
286: 7000 format( ' MVP-BURN function mode < ,A,'> is not supported ' )
287: 7020 format(6X, 'MVP-BURN CODE SYSTEM STARTED   DATE : ',A8,
288: & ' TIME=',A,':')
289: 7040 format(/2X, 'CUMULATIVE CPU TIME ',1P,E12.5, ' MIN ',
290: & /2X, 'USED MEMORY FOR BURN ',I12, ' WORD' /)
291: 7080 format(/2X, '=== ',A, ' MODE ENDED DATE : ',A, ' AT TIME=',A,
292: & ':')
293: 7100 format(6X, 'MVP-BURN CODE SYSTEM FINISHED   DATE : ',A8,
294: & ' TIME=',A,':')
295: 8000 format( ' ',79('*') )
296: C
297:     stop
298:     end
```

src/mvpburn/mvpdel.f

```
1:      subroutine MVPDEL( DIROUT,CASEID,SID1, JSVMVP,NOUT1 )
2: C=====
3: C Delete MVP outputs/inputs of case CASID, step SID1
4: C
5: C
6: C*obcolete
7: CC* JSVMVP = j2 * 2**2 + j1* 2**1 + j0
8: C
9: C JSVMVP = j3* 10**3 + j2 * 10**2 + j1* 10**1 + j0
10: C
11: C J0 = 1 : do not delete printout file (member caseVPst)
12: C J1 = 1 : do not delete result file (member caseVRst)
13: C J2 = 1 : do not delete source file (member caseVSst)
14: C J3 = 1 : do not delete input file (member caseVSst)
15: C
16: C-----
17:      character*(*) DIROUT
18:      character*4 CASEID
19:      character*2 SID1
20: C
21:      character*8 MEMBER
22: C
23: C-----
24: C
25:      JJ      = abs(JSVMVP)
26:      J0      = MOD(JJ,10)
27:      JJ      = (JJ-J0) /10
28:      J1      = MOD(JJ,10)
29:      JJ      = (JJ-J1) /10
30:      J2      = MOD(JJ,10)
31:      JJ      = (JJ-J2) /10
32:      J3      = MOD(JJ,10)
33:      JJ      = (JJ-J3) /10
34:      J4      = MOD(JJ,10)
35: C
36:      call PDSIO( 'OPEN', DIROUT, MEMBER, MEMBER, DUMMY, 1,
37: &      NOUT1 )
38: C
39:      if ( J0.eq.0 ) then
40:          MEMBER = CASEID//'VP'//SID1
41:          call PDSIO( 'DELETE', DIROUT, MEMBER, MEMBER, DUMMY, 1,
42: &      NOUT1 )
43:      end if
44: C
45:      if ( J1.eq.0 ) then
46:          MEMBER = CASEID//'VR'//SID1
47:          call PDSIO( 'DELETE', DIROUT, MEMBER, MEMBER, DUMMY, 1,
48: &      NOUT1 )
49:      end if
50: C
51:      if ( J2.eq.0 ) then
52:          MEMBER = CASEID//'VS'//SID1
53:          call PDSIO( 'DELETE', DIROUT, MEMBER, MEMBER, DUMMY, 1,
54: &      NOUT1 )
55:      end if
56: C
57:      if ( J3.eq.0 ) then
58:          MEMBER = CASEID//'VI'//SID1
59:          call PDSIO( 'DELETE', DIROUT, MEMBER, MEMBER, DUMMY, 1,
60: &      NOUT1 )
61:      end if
62: C
63:      MEMBER = 'CLOSING'
64:      call PDSIO( 'CLOSE', DIROUT, MEMBER, MEMBER, DUMMY, 1,
65: &      NOUT1 )
66:      return
67:      end
```

src/mvpburn/mvpin0.f

```

1:      subroutine MVPIN0( ISTEP, MEMORY,NOUT1, NOUT2, T00,  DIRIN, A,
2:      &                  TRGNAM, IDENT, MAXTRG, MAXISO,
3:      &                  CASEID, SID1, ISPC, JSIN, TITL2,
4: C2005&                  PARANM,PARATP,PARAV,NMVPPR,MAXSPR,JSVMVP )
5:      &                  PARANM,PARATP,PARAV,NMVPPR,MAXSPR,JSVMVP )
6: C=====
7: C  Create MVP input from number density of a step with ID "SID1".
8: C  (NOREQ'th step)
9: C
10: C  SID1 is step # in 2 digit
11: C  or ID of an intermediate step for PC method.
12: C
13: C  TITL2 is to memorize 2'nd line of MVP input data to be used to
14: C  check that the result file is output exactly from the data in
15: C  burnup phase.
16: C=====
17: C
18: C
19:      include 'INC/_BURNP'
20: C -----
21:      character*(*) DIRIN
22:      character*72 TITL2
23:      character*4 CASEID
24:      character*2 SID1
25:      real A(MEMORY)
26:      character*12 TRGNAM(MAXTRG)
27: C2003
28:      character*12 IDENT(MAXISO,MAXTRG)
29: C
30: C
31: C      PARANM(*) : symbolic parameter name to be replaced
32: C      PARATP(*) : data type of parameter
33: C                  ( 'F': float, 'I': integer )
34: C      NMVPPR    : number of symbolic parameters
35: C      PARAV(1:nstep,*) : data value of symbolic parameter
36: C
37:      character*16 PARANM(MAXSPR)
38:      character*1  PARATP(MAXSPR)
39:      real*8      PARAV(MXSTEP,MAXSPR)
40:      integer     NMVPPR
41:      integer     JSVMVP(MXSTEP)
42: C
43:      include 'INC/_CBURN1'
44:      include 'INC/_CBURN2'
45:      include 'INC/_CBURN3'
46: C
47: C
48:      real INSCR, INTCR
49:      common /MEMOCK/  MEUSED
50:      common /BNREST/  RSDUMY(3),      NOWSTP, IBEND,  AKEFF(MXSTEP),
51:      &      ERRKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
52:      &      EXPST(MXSTEP),  CUMMWD(MXSTEP),  INSCR(MXSTEP),
53:      &      INTCR(MXSTEP),  EINSR(MXSTEP),  EINTCR(MXSTEP),
54:      &      FLXNRM(MXSTEP),  FACNRM(MXSTEP),  FISABS(MXSTEP),
55:      &      FRTCAP(MXSTEP),  EFISAB(MXSTEP),  EFRTCA(MXSTEP),
56:      &      FDECAY(MXSTEP),  CDECAY(MXSTEP),  NHIST(MXSTEP),
57:      &      NBATCH(MXSTEP)
58: C2005
59:      &      ,ERRNRM(MXSTEP),  POWERW(MXSTEP),  POWERE(MXSTEP),
60:      &      RMONIT(MXSTEP),  EMONIT(MXSTEP)
61: Cend
62: C
63:      character*72 DDNAME
64:      character*8  MEMBER
65: C

```

```

66:      integer IDUM(10)
67: C
68: C
69:      if (IBEDIT.gt. 0) then
70:          write(NOUT1,7000) ISTEP, SID1, CASEID, TITLE
71:      end if
72:      NOREQ = ISTEP
73: C
74: C---- READ PDS
75: C
76:      call RWCOM1('READ', DIRIN, CASEID )
77:      call RWCOM2('READ', DIRIN, CASEID )
78:      call RWCOM3('READ', DIRIN, CASEID )
79: C  BRANCH MODE RESTART FILE NOT READ
80:      if (IBC(4) .eq. 0) then
81:          call RWREST('READ', DIRIN, CASEID )
82:      end if
83: C
84:      MEMBER = 'OPENING'
85:      call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, A, 3, NOUT1 )
86: C
87: C  ... check length of "HT" data of step SID1 ...
88: C
89:      MEMBER = CASEID//'HT'//SID1
90:      call PDSIO( 'SEARCH', DIRIN, MEMBER, MEMBER, RSDUMY, LHT,
91:      &          NOUT1 )
92: C
93:      if( LHT.le.0 ) then
94:          write(NOUT1,6888)
95:          write(NOUT1,*) ' requested burnup step (ID: ',SID1,' has',
96:      &          ' not burnup history member : ',MEMBER
97:      &          MEMBER = 'CLOSING'
98:      &          call PDSIO( 'CLOSE', DIRIN, MEMBER,MEMBER,A,0,NOUT1)
99:      &          stop 999
100:      end if
101: C
102:      if ( IBEND.eq.1 ) then
103:          write(NOUT1,6888)
104:          write(NOUT1,*) ' requested burnup step No is greater than',
105:      &          ' NEP (BURNIN data)'
106:          write(NOUT1,*) ' so production of MVP input data will be',
107:      &          ' skipped'
108: C
109: C ----- CLOSE PDS
110: C
111:      MEMBER = 'CLOSING'
112: C
113:      call PDSIO( 'CLOSE', DIRIN, MEMBER, MEMBER, A, 0, NOUT1
114:      &          )
115:      DDNAME = ' '
116: C
117:      call RWIND( 93 )
118: C
119:      write(93,'(A72,8x)') DDNAME
120:      write(93,'(A72,8x)') DDNAME
121: C
122:      call RWIND( 93 )
123:      stop 888
124:      end if
125: C
126:      IWRITE = 0
127: C
128:      if ( NOREQ.eq.0 ) then
129:          NOREQ = NOWSTP
130:      end if

```

src/mvpburn/mvpin0.f

```

131: C
132:   if ( NOREQ.lt.NOWSTP ) IWRITE  = 1
133: C
134:   if ( NOREQ.lt.0 ) then
135:     NOREQ  = -NOREQ
136:     IWRITE = 0
137:   end if
138: C
139:   if ( NOREQ.gt.NOWSTP ) then
140:     write(NOUT1,6888)
141:     write(NOUT1,*) ' requested burnup step No (',NOREQ,
142: & ' ) is greater than',
143: & ' finished step No (',NOWSTP,') until now'
144: C
145: C ----- CLOSE PDS
146: C
147:   MEMBER = 'CLOSING'
148: C
149:   call PDSIO( 'CLOSE', DIRIN, MEMBER, MEMBER, A, 0, NOUT1
150: & )
151:   DDNAME = ' '
152:   call RWIND( 93 )
153:   write(93, '(A72,8x)') DDNAME
154:   write(93, '(A72,8x)') DDNAME
155:   call RWIND( 93 )
156:   stop 888
157: end if
158: C
159: C *** SET VARIABLE DIMENSION
160: C
161:   LOC01 = 1
162: C --- DNBURN
163:   LOC02 = LOC01 + NMAT*NTNUC
164: C --- NISO
165:   LOC03 = LOC02 + NMAT
166: C --- TEMP
167:   LOC04 = LOC03 + NMAT
168: C --- VOLM
169:   LOC05 = LOC04 + NMAT
170: C --- TRGNAM
171: CCC LOC06 = LOC05 + NMAT*3
172: C2005 LOC06 = LOC05
173: C --- MATREG
174:   LOC06 = LOC05 + NMAT
175: Cend
176: C --- LENTRG
177:   LOC07 = LOC06 + NMAT
178: C --- MTBURN
179:   LOC08 = LOC07 + NMAT*3
180: C --- MTCARD
181:   LOC09 = LOC08 + NMAT*2
182: C --- IDENT
183: CCC LOC10 = LOC09 + KNMAX*NMAT*2
184:   LOC10 = LOC09
185: C --- DNINIT
186:   LOC11 = LOC10 + KNMAX*NMAT
187: C --- MATMVP
188:   LOC12 = LOC11 + NMAT
189: C --- IPBURN
190:   LOC13 = LOC12 + KNMAX*NMAT
191: C --- MVPDAT
192: CCC LOC14 = LOC13 + NCARD*18
193:   LOC14 = LOC13
194: C
195:   LAST = LOC14 + LHT

```

```

196: C -----
197:   if (LAST.gt.MEUSED) MEUSED = LAST
198:   LEMORY = MEMORY - LAST + 1
199:   if ( MEMORY.lt.LAST ) then
200:     write(NOUT1,6888)
201:     write(NOUT1,*) ' memory over '
202:     write(NOUT1,*) ' requested memory size = ', LAST-1,
203: & ' words.'
204:     write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
205:     write(NOUT1,*) ' change parameter value in include file'
206:     stop 999
207:   end if
208: C
209:   call CLEA( A, LAST-1, 0.0 )
210: C
211:   call MVPINP( ISTEP, NOUT1, NOUT2, NOREQ, LHT, DIRIN, CASEID,
212: &   SID1, ISPC, JSIN,
213: &   TITL2,
214: &   IWRITE, A(LOC01), A(LOC02), A(LOC03), A(LOC04),
215: &   TRGNAM, A(LOC06), A(LOC07), A(LOC08), IDENT, A(LOC10),
216: &   A(LOC11), A(LOC12), A(LOC14), A(LOC14),
217: C2005&   PARAMN, PARATP, PARAV, NMVPPR, MAXSPR, JSVMVP )
218: &   PARAMN, PARATP, PARAV, NMVPPR, MAXSPR, JSVMVP, A(LOC05) )
219: C
220:   call CPUTM2( T1 )
221:   if (IBEDIT .gt. 0) then
222:     write(NOUT1,7020) (T1 - T00)/60.0
223:   end if
224: C
225: C --- CLOSE PDS
226: C
227:   MEMBER = 'CLOSING'
228: C
229:   call PDSIO( 'CLOSE', DIRIN, MEMBER, MEMBER, A, 0, NOUT1 )
230: C
231: C *** END OF PROCESS
232: C
233: 6888 format(// ' XXX(MVPIN0) ERROR-STOP : ' )
234: 7000 format(//15X, 'xxxxxxxxxxxxxxxxxxxxx'
235: &   /15X, 'x MVPINPT-MODE x PRESENT STEP NO. : ', I3,
236: &   ' (', A2, ') '
237: &   /15X, 'xxxxxxxxxxxxxxxxxxxxx'
238: &   /1X, 'CASEID:', A4
239: &   /1X, 'TITLE : ', A72
240: &   /1X, ' : ', A72/ )
241: 7020 format(//1X, 12X, 'CPU ', E12.5, ' MIN. ' )
242: C
243:   return
244: end

```

src/mvpburn/mvpinp.f

```

1:      subroutine MVPINP( ISTEP,
2:      &                  NOUT1, NOUT2, NOREQ, MEMORY,DIRIN, CASEID,SID1,
3:      &                  ISPC, JSIN, TITL2,IWRITE,DNBURN,NISO,TEMP,VOLM,
4:      &                  TRGNAM,LENTRG,MTBURN,MTCARD,IDENT, DNINIT,
5:      &                  MATMVP,IPBURN,IARRAY,RARRAY,
6: C2005&                  PARAMN,PARATP,PARAV,NMVPPR,MAXSPR,JSVMVP )
7:      &                  PARAMN,PARATP,PARAV,NMVPPR,MAXSPR,JSVMVP ,
8:      &                  MATREG )
9: C=====
10: C Create MVP input from number density of step ISTEP with ID "SID1".
11: C
12: C SID1 : burnup step ID
13: C      SID1 is step # in 2 digit
14: C      or ID of an intermediate step for PC method.
15: C ISTEP : burnup step #
16: C ISPC : burn substep # in predictor-corrector mode (1 or 2)
17: C JSIN : return non zero value if input data requires fission source
18: C      file from previous step and input is setup to use it
19: C      (it must exist as a non-empty file.)
20: C=====
21:      character*72 TITL2
22: C
23:      include 'INC/_BURNP'
24: C
25:      integer IARRAY(MEMORY)
26:      real RARRAY(MEMORY)
27: C
28: C2003
29:      character*12 IDENT(KNMAX,NMAT)
30:      character*12 TRGNAM(NMAT)
31: C
32:      integer NISO(NMAT), LENTRG(NMAT)
33:      integer IPBURN(KNMAX,NMAT), MATMVP(NMAT)
34:      integer MTBURN(3,NMAT), MTCARD(2,NMAT)
35: C
36:      real TEMP(NMAT), VOLM(NMAT), DNINIT(KNMAX,NMAT)
37:      real DNBURN(NTNUC,NMAT)
38: Ckuni 2003/12/22
39:      integer KPBURN(MXNUC)
40: Cend
41: C2005
42:      integer MATREG(NMAT)
43: Cend
44: C
45: C PARAMN(*) : symbolic parameter name to be replaced
46: C PARATP(*) : data type of parameter
47: C      ( 'F': float, 'I': integer )
48: C NMVPPR : number of symbolic parameters
49: C PARAV(1:nstep,*) : data value of symbolic parameter
50: C
51:      character*16 PARAMN(MAXSPR)
52:      character*1 PARATP(MAXSPR)
53:      real*8 PARAV(MXSTEP,MAXSPR)
54:      integer NMVPPR
55:      integer JSVMVP(MXSTEP)
56: C
57:      character*4 CASEID
58:      character*2 SID1
59:      character*(*) DIRIN
60: C
61:      include 'INC/_CBURNC1'
62:      include 'INC/_CBURNC2'
63:      include 'INC/_CBURNC3'
64: C
65:      real INSCR, INTCR

```

```

66:      common /MEMOCK/ MEUSED, MEMMAX
67:      common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
68:      &          ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
69:      &          EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
70:      &          INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
71:      &          FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
72:      &          FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
73:      &          FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
74:      &          NBATCH(MXSTEP)
75: C2005
76:      &          ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
77:      &          RMONIT(MXSTEP), EMONIT(MXSTEP)
78: Cend
79: C
80: Ckuni 2003/12/22
81: Cdel character*72 DDNAME
82: Cmod character*8 MEMBER, MEMBR0, MEMBR1,MEMBR2
83: C      character*8 MEMBER, MEMBR0, MEMBR1
84: Cend
85: C2003
86:      character*12 NUCID
87:      character*2 STNM
88: C
89: C
90:      character*72 MVPDAT1, MVPDAT2, LINE
91:      character*8 IDATE, STIME
92: C
93:      character*2 STEPNM
94:      external STEPNM
95: C
96: C-----
97:      MEMBER = 'OPENING'
98:      call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, DUMMY, 3, NOUT1 )
99: C
100: C *** READ MATERIAL DATA
101: C
102: C
103:      call RWMATD( 'READ', DIRIN, CASEID, NMAT, KNMAX, NISO(1), TEMP(1),
104:      &              VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1), MTCARD(1,1),
105: C2005&              IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) )
106:      &              IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) ,
107:      &              MATREG(1) )
108: C
109: C ... input from stepwise "HIST" member (from version 2.0)
110: C
111:      MEMBER = CASEID// 'HT'//SID1
112: C2003 ... check member size
113:      call PDSIO( 'SEARCH',DIRIN,MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
114:      if (LENG.gt.MEMORY) then
115:          write(NOUT1,*) 'XXX(MVPINP) Size of PDS member ',MEMBER,
116:      &          ' (=' ,LENG,') exceeds buffer size (=' ,MEMORY,')'
117:          stop 999
118:      else
119:          MTMP = MEMMAX - MEMORY + LENG
120:          if (MTMP.gt.MEUSED) MEUSED = MTMP
121:      end if
122: C2003 ... end
123:      LENG = 0
124:
125:      call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, IARRAY, LENG, NOUT1 )
126:
127:      call PCTRW( IARRAY, MEMBER, IRET )
128:      call UNPKND( IARRAY, 'STEP#', 'I4', ISWSTP, 1, NDATA2, IRET )
129: C
130: C ... get number density only ...

```

src/mvpburn/mvpinp.f

```

131: c
132:   call PCTLB( IARRAY, 'number density', 'DNBURN', IRET )
133:   call UNPKND( IARRAY, 'number density', 'R4', DNBURN, NTNUC*NMAT,
134:   &          NDATA2, IRET )
135:   ISW      = LENG
136:   if ( IBEDIT.gt.2 ) then
137:     write(NOUT1,*) ' ** ISW ISWSTP LENG : ', ISW, ISWSTP, LENG
138:   end if
139: C
140: C *** open base MVP input member and MVP input data for step SID1
141: C
142:   MEMBR0 = CASEID//'MVPI'
143:   IMVPI   = 61
144:   call PDSFIL( 'READ', DIRIN, MEMBR0, 'TEXT', IMVPI, NOUT1, IIERR )
145:   if ( IIERR.ne.0 ) then
146:     write(NOUT1,6888)
147:     write(NOUT1,*) ' failed to open member <','MEMBR0,>'
148:     stop 999
149:   end if
150:
151:   MEMBER = CASEID//'VI'//SID1
152:   IMVPO   = 63
153:   call PDSFIL( 'WRITE', DIRIN, MEMBER, 'TEXT', IMVPO, NOUT1, IIERR
154:   &          )
155:   if ( IIERR.ne.0 ) then
156:     write(NOUT1,6888)
157:     write(NOUT1,*) ' failed to open member <','MEMBER,>'
158:     stop 999
159:   end if
160: C
161: C ... read first 2 lines of MVP input
162: C
163:   read(IMVPI,'(a)') MVPDAT1
164:   read(IMVPI,'(a)') MVPDAT2
165: C
166:   call UDATE( IDATE )
167:   call JIKAN( STIME )
168:   MVPDAT2(40:) = '# '//IDATE//'# '//STIME//' :STEP '//SID1
169: C
170: C ... Second line of the title is memorized in TITL2 to use
171: C   as a matching tag of input and result file in burnup phase
172: C
173:   TITL2 = MVPDAT2
174: C
175:   write(IMVPO,'(A72,A8)') MVPDAT1, MEMBER
176:   write(IMVPO,'(A72)') MVPDAT2
177: C
178: C ... this flag is set to non-zero if "###FISSIONFILE"
179: C   is found
180: C
181:   JJSIN = 0
182: C
183: C ... this flag is set to non-zero if "SOURCE-OUTPUT"
184: C
185:   JSOUOT = 0
186: C
187: C ... write input as is until the first material line ...
188: C
189:   LAST = MTCARD(1,1) - 1
190:   do 100 I = 3, LAST
191:     read(IMVPI,'(a)') MVPDAT1
192:     .... symbolic parameter modification
193:     if ( NMVPPR.gt.0.and.MVPDAT1(1:1).eq.'%' ) then
194:       call MVPSYM( MVPDAT1, ISTEP,
195:       &          PARANM,PARATP,PARAV,NMVPPR,MAXSPR, MXSTEP,

```

```

196:       &          NOUT1)
197:     end if
198:     write(IMVPO,'(A72)') MVPDAT1
199:     if ( MVPDAT1(1:14).eq.'###FISSIONFILE') JJSIN = JJSIN + 1
200:     if ( MVPDAT1(1:1).ne.'*') then
201: C
202: C SOURCE-OUTPUT & NO-SOURCE-OUTPUT OPTION CHECK
203: C
204:       call SOUCHK(MVPDAT1,JSOUOT)
205:     end if
206:   100 continue
207: C
208: C ... SOURCE-FILE OUTPUT OPTION CHECK , FISSIONFILE CASE
209: C
210:   if (JJSIN.ne.0) then
211:     if (JSOUOT.eq.0) then
212:       write(NOUT1,6888)
213:       write(NOUT1,'(A,A)') ' fission source option(###FISSIONFILE)',
214:       &          ' is selected '
215:       write(NOUT1,'(A,A)') ' but SOURCE-OUTPUT option is not',
216:       &          ' selected in MVP input.'
217:     stop 888
218:   end if
219: end if
220: C
221:   do 150 M = 1, NMAT
222:     MMK = NISO(M)
223:     IST = MTBURN(1,M)
224:     IEND = MTBURN(2,M)
225:     RTEMP = TEMP(M)
226:   Ckuni 2003/12/22
227:     MTYP = MTBURN(3,M)
228:     call ICLEA( KPBURN, MXNUC , 0 )
229:     do 105 I = IST, IEND
230:       ISW = 0
231:       do 103 K = 1, MMK
232:         if ( IPBURN(K,M).eq.I ) ISW = K
233:       103 continue
234: C
235:       if( MTYP.eq.1.and.I.le.LASTFP) then
236:         KPBURN(I) = 1
237:       else
238:         if(ISW.gt.0) then
239:           KPBURN(I) = 1
240:         endif
241:       endif
242:   105 continue
243: Cend
244:   do 120 I = IST, IEND
245:     DSAVE = DNBURN(I,M)
246:     if ( DSAVE.le.0.0 ) DSAVE = BZERO
247: C2003- To avoid too small reaction rate during burnup (by KSK)
248:     if ( DSAVE.le.BZERO) DSAVE = BZERO
249: C2003+
250:     NUCID = ' '
251:     ISW = 0
252:     do 110 K = 1, MMK
253:       if ( IPBURN(K,M).eq.I ) ISW = K
254:     110 continue
255:   Ckuni 2003/12/22
256:   if ( ISW.eq.0.and.KPBURN(I).eq.0 ) go to 120
257: Cend
258:   if ( ISW.eq.0 ) then
259: C2003   NUCID(1:4) = SRACID(I) (2:4) //'0'
260: C2003 ... make nuclide ID from NCODE

```

src/mvpburn/mvpinp.f

```

261:      call ZZMMM(1,1,NCODE(I),NUCID)
262:      if(NUCID(6:6).eq.' ') NUCID(6:6) = '0'
263:      if(NUCID(6:6).eq.'M') NUCID(6:6) = '1'
264:      NUCID(7:10) = '0000'
265: C2003 ... end
266:      call SETMID( NUCID, RTEMP )
267:      else
268:      NUCID = IDENT(ISW,M)
269:      end if
270: C
271:      LINE = ' '
272:      write(LINE,7000) NUCID, DSAVE
273:      call SCTOLC( LINE, LEN(LINE) )
274:      write(IMVPO,'(A)') LINE(:ICLEN2(LINE))
275: 120 continue
276: C
277: 7000 format(2X,A,'( ',1P,E12.5,' )')
278: C
279:      do 130 I = 1, MMK
280:      if ( IPBURN(I,M).eq.0 ) then
281:      LINE = ' '
282:      write(LINE,7000) IDENT(I,M), DNINIT(I,M)
283:      call SCTOLC( LINE, LEN(LINE) )
284:      write(IMVPO,'(A)') LINE(:ICLEN2(LINE))
285:      end if
286: 130 continue
287: C
288: C ... output MVP base input among material lines as is ...
289: C
290:      if ( M.lt.NMAT ) then
291:      IST = MTCARD(2,M) + 1
292:      IEND = MTCARD(1,M+1) - 1
293:      do 140 I = LAST + 1, IEND
294:      read(IMVPI,'(A)') MVPDAT1
295:      if ( I.ge.IST ) then
296:      ... symbolic parameter modification
297:      if ( NMVPPR.gt.0.and.MVPDAT1(1:1).eq.'%' ) then
298:      call MVPSYM( MVPDAT1, ISTEP,
299:      &      PARAMN,PARATP,PARAV,NMVPPR,MAXSPR, MXSTEP,
300:      &      NOUT1)
301:      end if
302:      write(IMVPO,'(A)') MVPDAT1(:ICLEN2(MVPDAT1))
303:      end if
304: 140 continue
305:      LAST = IEND
306:      end if
307: 150 continue
308: C
309: C ... output MVP base input as is ...
310: C
311:      IST = MTCARD(2,NMAT) + 1
312:      IEND = NCARD
313:      do 160 I = LAST + 1, IEND
314:      read(IMVPI,'(A)') MVPDAT1
315:      if ( I.ge.IST ) then
316: C
317: C ... symbolic parameter modification
318:      if ( NMVPPR.gt.0.and.MVPDAT1(1:1).eq.'%' ) then
319:      call MVPSYM( MVPDAT1, ISTEP,
320:      &      PARAMN,PARATP,PARAV,NMVPPR,MAXSPR, MXSTEP,
321:      &      NOUT1)
322:      end if
323:      write(IMVPO,'(A)') MVPDAT1(:ICLEN2(MVPDAT1))
324: Cdel if ( index(MVPDAT1,'*#SOURCE-').eq.1) JJSIN = JJSIN + 1
325:      end if

```

```

326:      160 continue
327: C
328:      call RWIND( IMVPO )
329:      call PDSFIL( 'CLOSE', DIRIN, MEMBR0, 'TEXT', IMVPI, NOUT1, IIERR
330:      &      )
331: C
332: C=====
333: C *** Replace $SOURCE block with fission file input if possible ***
334: C
335:      JSIN = 0
336: C
337:      if ( JJSIN.gt.0.and.(ISTEP.gt.1.or.ISTEP.eq.1.and.ISPC.eq.2)) then
338: C
339: C ... second substep of PC method mode (step i+1/2):
340: C      try to use source file
341: C      from the first substep (step i)
342: C
343: C      JSIN is 2 when the member exist and not empty.
344: C
345:      if ( ISPC.eq.2 ) then
346:      stnm = stepnm(istep,0)
347:      ITEMP = JSVMVP(ISTEP)/100
348:      ITEMP = MOD(ITEMP,10)
349:      if ( ITEMP.eq.0 ) then
350:      write(NOUT1,6666)
351:      write(NOUT1,'(A,A,A,A)') ' source file is not',
352:      &      ' saved in the previous step (' ,STNM,' ) in SAVE-MVP-OUTPUT.'
353:      write(NOUT1,'(A,A)') ' fission source option is not used',
354:      &      ' in this step.'
355:      else
356:      MEMBR1 = CASEID// 'VS' // Stnm
357:      call PDSIO('EXIST',DIRIN, MEMBR1,MEMBR1,dummy,iflag,NOUT1)
358:      if ( iflag.gt.0 ) then
359:      call PDSFIL('READ',DIRIN,MEMBR1,'BINARY',62,nout1,iier2)
360:      if( iier2.eq.0 .or. iier2.eq.1 ) then
361:      read(62,end=1234)
362: C
363: C ... non empty source file found
364:      JSIN = 2
365:      goto 1235
366: C
367: 1234 write(NOUT1,6666)
368:      write(NOUT1,*) ' Member <',MEMBR1,> is empty.'
369:      write(NOUT1,'(A,A)') ' fission source option is not',
370:      &      ' used in this step.'
371: C
372: 1235 call PDSFIL('CLOSE',DIRIN,MEMBR1,'BINARY',62,
373:      &      nout1,iier2)
374:      if( JSIN.eq.2 ) goto 1311
375:      else
376:      write(NOUT1,6666)
377:      write(NOUT1,*) ' Member <',MEMBR1,> is empty.'
378:      write(NOUT1,'(A,A)') ' fission source option is not',
379:      &      ' used in this step.'
380:      call PDSFIL('CLOSE',DIRIN,MEMBR1,'BINARY',62,
381:      &      nout1,iier2)
382:      if( JSIN.eq.2 ) goto 1311
383:      end if
384:      end if
385:      end if
386:      end if
387: C
388: C
389: C ... try to use source file from previous step
390: C

```

src/mvpburn/mvpinp.f

```

391: C      JSIN is 1 when the member exist and not empty.
392: C
393:       if ( ISPC.eq.1 ) then
394:         stnm = stepnm(istep-1,0)
395:         ITEMP = JSVMVP(ISTEP-1)/100
396:         ITEMP = MOD(ITEMP,10)
397:         if (ITEMP.eq.0) then
398:           write(NOUT1,6666)
399:           write(NOUT1,'(A,A,A,A)') ' source file is not',
400: &         ' saved in the previous step (' ,STNM,') in SAVE-MVP-OUTPUT.'
401:           write(NOUT1,'(A,A)') ' fission source option is not used',
402: &         ' in this step.'
403:         else
404:           MEMBR1 = CASEID//VS//STNM
405:           call PDSIO('EXIST',DIRIN, MEMBR1,MEMBR1,dummy,iflag,NOUT1)
406:           if ( iflag.eq.1 ) then
407:             call PDSFIL('READ',DIRIN,MEMBR1,'BINARY',62,nout1,iier2)
408:             if( iier2.eq.0 ) then
409:               read(62,end=1236)
410: C
411: C      ... non empty source file found
412: C      JSIN = 1
413: C      goto 1237
414: C
415: 1236      write(NOUT1,6666)
416:           write(NOUT1,*) ' Member <','MEMBR1,'> is empty.'
417:           write(NOUT1,'(A,A)') ' fission source option is not',
418: &         ' used in this step.'
419: C
420: 1237      call PDSFIL('CLOSE',DIRIN,MEMBR1,'BINARY',62,
421: &         nout1,iier2)
422:           if( JSIN.eq.1 ) goto 1311
423:           end if
424:         else
425:           write(NOUT1,6666)
426:           write(NOUT1,*) ' Member <','MEMBR1,'> is not found.'
427:           write(NOUT1,'(A,A)') ' fission source option is not',
428: &         ' used in this step.'
429:           end if
430:         end if
431:       end if
432: C
433: 1311      continue
434: C
435:       if ( JSIN .ne.0 ) then
436: C
437: C      ... check "$SOURCE"/"$END SOURCE" pair
438: C      and insert sampling function #FISSIONFILE.
439: C
440:         IMVPI2 = 93
441:         call rwind(IMVPI2)
442:         ISRC = 0
443:         NSSRC = 0
444: C
445: C===== 121 loop for source block =====
446: C
447: 121      LINE = ' '
448:           read(IMVPO,'(a)',end=1222) LINE
449:           if(LINE(1:7).eq. '$SOURCE') then
450: CKSK can I assume one $SOURCE block ? Now no checking.
451:             ISRC = 1
452:             NSSRC= NSSRC + 1
453:             write(IMVPI2,'(a)') '*** Source Replaced by MVP-BURN ***'
454:             write(IMVPI2,'(a)') '$SOURCE'
455:             write(IMVPI2,'(a)') '&'

```

```

456:           write(IMVPI2,'(a)') ' NEUTRON'
457:           write(IMVPI2,'(a)') ' RATIO(1.0)'
458:           write(IMVPI2,'(a)')
459: &         '@(X Y Z A B C W E) = #FISSIONFILE() ;'
460:           write(IMVPI2,'(a)') '$END SOURCE'
461:         endif
462: C
463: C      ... original source lines are left as comment lines
464: C      if(ISRC.EQ.1) then
465: C      if( LINE(1:4).eq. '$END' .and.
466: &         index(LINE(5:72),'SOURCE').gt.0 ) then
467: C      ISRC = 0
468: C      endif
469: C      write(IMVPI2,'(a)') '*/LINE(:ICLEN2(LINE))
470: C      else
471: C      write(IMVPI2,'(a)') LINE(:ICLEN2(LINE))
472: C      end if
473: C
474: C      goto 121
475: CKSK=====
476: C
477: 1222      continue
478: C
479:       if ( NSSRC.gt.1 ) then
480:         write(NOUT1,6888)
481:         write(NOUT1,*) ' Number of source blocks($SOURCE)',
482: &         ' more than 1 is not considered in MVP-BURN'
483:         write(NOUT1,*) ' for fission source replacement'
484:         stop 888
485:       end if
486: C
487: C      ...out of mvp-burn business ( may be mvp will check)
488: CKSK      if ( NSSRC.gt.0 .and. ISRC.ne.0 ) then
489: CKSK        write(NOUT1,6888)
490: CKSK        write(NOUT1,*) ' $SOURCE was found',
491: CKSK &        ' but source block is not closed by $SOURCE END.'
492: CKSK        stop 888
493: CKSK      end if
494: C
495:       if ( NSSRC.eq.0 ) then
496: C
497: C      ... no replacement of source
498: C
499:         JSIN = 0
500:       else
501: C
502: C      ... copy back data on IMVPO
503: C
504:         call rwind(IMVPI2)
505:         call rwind(IMVPO)
506:         LINE = ' '
507: CCCCC      read(IMVPO,'(a)',end=1229) LINE
508:           read(IMVPI2,'(a)',end=1229) LINE
509:           write(IMVPO,'(a)') LINE(:iclen2(LINE))
510:           goto 129
511: 1229      call rwind(IMVPO)
512:           end if
513:         end if
514:       end if
515: C
516: C *** PRINT OUT
517: C
518:       if (IBEDIT .gt. 0) then
519:         write(NOUT1,7020) CASEID, TITLE, SID1, NOWSTP
520:       end if

```


src/mvpburn/mvpinp.f

```
521: 7020 format('1',15X,' << MVPINP STEP INFORMATION LIST >>'//1X,
522: & 'CASEID:',A4/1X,'TITLE :',A72/1X,' :',
523: & A72/1X,' REQUESTED BURNUP STEP ID IS <',A,'>'//1X,
524: & ' BURNUP STEP NUMBER IN PDS-FILE IS ',I3//)
525: IF(IBC(3).LT.2) goto 9000
526: C
527: C ... echo created MVP input data ...
528: C
529: write(NOUT1,7030)
530: write(NOUT1,7040)
531: ICNT = 0
532: 170 continue
533: read(MVPO,'(A72)',end =180) LINE
534: ICNT = ICNT + 1
535: write(NOUT1,7060) ICNT, LINE, ICNT
536: go to 170
537: 180 continue
538: write(NOUT1,7080)
539: C
540: 9000 call PDSFIL( 'CLOSE', DIRIN, MEMBER, 'TEXT', IMVPO, NOUT1, IIERR
541: & )
542: C
543: 7030 format(/1X,10X,
544: & ' ** GENERATED INPUT DATA LIST FOR MVP CODE **'//)
545: 7040 format(12X,'...*...1...*...2...*...3...*...4',
546: & '...*...5...*...6...*...7...*...8')
547:
548: 7060 format(I8,4X,A72,I8)
549: 7080 format(/' ',12X,'*** INPUT DATA END ***')
550: C
551: C *** UPDATE caseREST MEMBER IF NOREQ < NOWSTP
552: C
553: if ( NOREQ.lt.NOWSTP.and.IWRITE.eq.1 ) then
554:
555: NOWSTP = NOREQ
556:
557: c MEMBER = CASEID//'REST'
558: c LENG = LENG4
559: c call PDSIO( 'WRITE', DDNAME, MEMBER, MEMBER, RSDUMY,
560: c & LENG, NOUT1 )
561:
562: call RWREST( 'WRITE', DIRIN, CASEID )
563:
564: end if
565: C
566: 6888 format( //' XXX(MVPINP) ERROR-STOP :' )
567: 6666 format( //' !!!(MVPINP) WARNING :' )
568: C
569: C *** END OF PROCESS
570: C
571: return
572: end
```

src/mvpburn/mvprun.f

```

1:      subroutine MVPRUN( ISTEP, ISPC,  DIRIN, DIROUT,CASEID,SID1,
2:      &                  MVPEX, MVPIDX, JSIN, NOUT1, IBC )
3:      C=====
4:      C MVP code execution in MVP-Burn code :
5:      C
6:      C-----
7:      C arguments (i=input o=output w=work)
8:      C i ISTEP : burnup step # (counted from 1)
9:      C i ISPC  : sub-step # for predictor-corrector mode ( 1 or 2 )
10:     C i DIRIN : input PDS name
11:     C i DIROUT: input PDS name
12:     C i CASEID: problem case ID
13:     C i SID1  : burnup step ID
14:     C i MVPEX : MVP execution command string.
15:     C i MVPIDX: MVP cross section library index file name
16:     C i JSIN  : fission source file input if non-zero
17:     C i NOUT1 : message printout I/O unit
18:     C i IBC   : flags
19:     C-----
20:     C/#IF MS_VISUAL
21:     C
22:     C ... This interface block is for CUTIL & MS-Visual tools. ...
23:     C
24:     *      interface
25:     *      subroutine FTSYSTEM( INT1, CHA1, INT2 )
26:     *      integer          INT1,INT2
27:     *      character*962 CHA1
28:     *CDEC$      ATTRIBUTES C :: FTSYSTEM
29:     *CDEC$      ATTRIBUTES REFERENCE :: INT1, CHA1, INT2
30:     *      end subroutine
31:     *      end interface
32:     C/#ENDIF
33:     character*(*) DIRIN, DIROUT
34:     character*4 CASEID
35:     character*2 SID1
36:     character*(*) MVPEX
37:     character*(*) MVPIDX
38:     C
39:     C ... local variable ...
40:     CKSK character*450 COMMAND
41:     CKSK too short when path name is too long or too many arguments
42:     CKSK 64*15 lines + 2
43:     character*962 COMMAND
44:     character*256 FILENM
45:     character*256 CWRK
46:     C
47:     character*2 STNM
48:     character*8 MEMBER
49:     logical NMD
50:     C2003 ... IBC is added as an argument
51:     integer IBC(20)
52:     C
53:     character*2 STEPNM
54:     external STEPNM
55:     C
56:     C-----
57:     C
58:     COMMAND = ' '
59:     ICL      = 1
60:     call CATST2( COMMAND, ICL, ' ', MVPEX(:ICLEN2(MVPEX)), IERR0 )
61:     C
62:     C-----
63:     C MVP input file (as virtual member of PDS DIRIN )
64:     C-----
65:     MEMBER = CASEID//'VI'//SID1

```

```

66:     call PDSNM( DIRIN, MEMBER, FILENM, NLEN, IERR )
67:     CWRK = ' /5rf://FILENM(:NLEN)
68:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
69:     C
70:     C-----
71:     C MVP index file
72:     C-----
73:     CWRK = ' /25rf://MVPIDX
74:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
75:     C
76:     C-----
77:     C MVP binary output file (can be opened as virtual member of PDS)
78:     C-----
79:     MEMBER = CASEID//'VR'//SID1
80:     call PDSNM( DIROUT, MEMBER, FILENM, NLEN, IERR )
81:     CWRK = ' /30://FILENM(:NLEN)
82:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
83:     C
84:     C-----
85:     C MVP source output file (can be opened as virtual member of PDS)
86:     C-----
87:     MEMBER = CASEID//'VS'//SID1
88:     call PDSNM( DIROUT, MEMBER, FILENM, NLEN, IERR )
89:     CWRK = ' /9://FILENM(:NLEN)
90:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
91:     C
92:     C-----
93:     C MVP source input file (can be opened as virtual member of PDS)
94:     C-----
95:     if ( JSIN.ne.0 ) then
96:     C
97:     if ( JSIN.eq.2 ) then
98:         STNM = STEPNM(ISTEP,0)
99:     else
100:         STNM = STEPNM(ISTEP-1,0)
101:     end if
102:     MEMBER = CASEID//'VS'//STNM
103:     call PDSNM( DIROUT, MEMBER, FILENM, NLEN, IERR )
104:     CWRK = ' /8r://FILENM(:NLEN)
105:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
106:     end if
107:     C
108:     C-----
109:     C MVP restart output file
110:     C-----
111:     call CATSTR( COMMAND, ICL, ' /20', IERR0 )
112:     C
113:     C-----
114:     C Other working files
115:     C-----
116:     C-----
117:     C
118:     CWRK = ' /15f /16 /55f /66f /75f'
119:     call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
120:     C-----
121:     C Explicit end of command line arguments (for MPICH)
122:     C-----
123:     call CATSTR( COMMAND, ICL, ' @@', IERR0 )
124:     C
125:     C-----
126:     C MVP printout file (can be opened as virtual member of PDS DIRIN )
127:     C-----
128:     MEMBER = CASEID//'VP'//SID1
129:     call PDSNM( DIROUT, MEMBER, FILENM, NLEN, IERR )
130:     CWRK = ' > //FILENM(:NLEN)

```

src/mvpburn/mvprun.f

```
131:      call CATSTR( COMMAND, ICL, CWRK(:ICLEN2(CWRK)), IERR0 )
132: C
133: C=====
134: C   Execute MVP
135: C=====
136:      write(NOUT1,7000) ISTEP,SID1
137:      write(NOUT1,7010)
138: C
139:      do 100 I = 1, ICL, 64
140:         I2      = MIN(ICL,I+64-1)
141:         write(NOUT1,7020) COMMAND(I:I2)
142: 100 continue
143: C
144:      call CPUTM2( TCPU0 )
145: C
146: CDebug stop
147: C      write(nout1,*) ' debug check then mvp not run (mvprun)'
148: C      stop
149: CDebug
150: C
151: C2003 ... skip MVP calculation in debug mode .
152:      if ( IBC(20).eq.0 ) then
153:         call FTSYSTEM( IERR1, COMMAND, ICLEN2(COMMAND) )
154:      else
155:         write(NOUT1,*) ' .... skip MVP run in debug mode.'
156:      end if
157: C
158:      call CPUTM2( TCPU1 )
159:      write(NOUT1,7040) IERR1, (TCPU1 - TCPU0)/60.0
160: C
161: C=====
162: 7000 format(/15X,'xxxxxxxxxxxxxxxxxxxxxx'
163:      &      /15X,'x      MVPRUN-MODE      x      PRESENT STEP NO. : ',I3,
164:      &      ' (',A2,')'
165:      &      /15X,'xxxxxxxxxxxxxxxxxxxxxx' / )
166: 7010 format(/1X,' === MVP EXECUTION COMMAND :'/)
167: 7020      format(3X,'< ',A,' >')
168: 7040 format(/1X,' === MVP CALCULATION FINISHED. CODE = ',I4,
169:      &      ' USED CPU TIME ',1P,E12.5,' MIN.')
170: C
171:      return
172:      end
```

src/mvpburn/mvpsym.f

```
1:      subroutine MVPSYM( LINE,  ISTEP,  PARANM,PARATP,PARAV, NMVPPR,  
2:      &                MAXSPR,MXSTEP,NOUT )  
3: C=====
```

4: C MVP symbolic parameter modifier for MVP-BURN

```
5: C=====
```

6: C

7: character*(*) LINE

8: C

9: character*16 PARANM(MAXSPR)

10: character*1 PARATP(MAXSPR)

11: real*8 PARAV(MXSTEP,MAXSPR)

12: integer NMVPPR

13: C

14: C local data

15: C

16: character*128 CWORK

17: character*128 NAME

18: C

19: C -----

20: C

21: if (NMVPPR.eq.0) return

22: if (LINE(1:1).ne.'%') return

23: C

24: C find NAME of "% NAME = ..."

25: C

26: K = INDEX(LINE,'=')

27: C

28: if (K.eq.0) return

29: C

30: N = 0

31: do 100 I = 2, K - 1

32: if (LINE(I:I).ne.' ') then

33: N = N + 1

34: NAME(N:N) = LINE(I:I)

35: end if

36: 100 continue

37: C

38: if (N.eq.0) return

39: C

40: do 110 II = 1, NMVPPR

41: if (NAME(1:N).eq.PARANM(II)) go to 120

42: 110 continue

43: C

44: return

45: C

46: 120 continue

47: C

48: CWORK = ' '

49: if (PARATP(II).eq.'F') then

50: write(CWORK,7000) PARANM(II) (:ICLEN2(PARANM(II))),

51: & PARAV(ISTEP,II), ISTEP

52: else

53: write(CWORK,7020) PARANM(II) (:ICLEN2(PARANM(II))),

54: & NINT(PARAV(ISTEP,II)), ISTEP

55: end if

56: 7000 format('% ',A,' = ',1P,D17.9,' /** REPLACED FOR STEP',I4)

57: 7020 format('% ',A,' = ',1I2,' /** REPLACED FOR STEP',I4)

58: C

59: write(NOUT,7040) LINE(:ICLEN2(LINE)), CWORK(:ICLEN2(CWORK))

60: 7040 format(/1X,'>>> SYMBOLIC PARAMETER DEFINITION IS CHANGED:'/1X,

61: & ' ORIGINAL LINE: ',A/1X,' CHANGED LINE: ',A)

62: C

63: LINE = CWORK

64: return

65: end

src/mvpburn/namrst.f

```
1:      subroutine NAMRST( NAMXXX,NAMMT )
2: C-----
3: C   restore {zzzmmmn}{r} form of string for NAMFIS and NAMFER
4: C-----
5:      character*8 NAMMT
6: C
7:      NAM      = NAMXXX/10
8:      MT       = NAMXXX - NAM*10
9: C
10:     write(NAMMT(1:7), '(I7)') NAM
11: C
12:     if ( MT.eq.1 ) NAMMT(8:8) = 'F'
13:     if ( MT.eq.2 ) NAMMT(8:8) = 'C'
14:     if ( MT.eq.4 ) NAMMT(8:8) = 'A'
15:     if ( MT.eq.0 ) NAMMT(8:8) = 'P'
16:     if ( MT.eq.3 ) NAMMT(8:8) = 'N'
17:     if ( MT.eq.5 ) NAMMT(8:8) = 'D'
18: C
19:     return
20: end
```

src/mvpburn/packet.f

```

1: C      parameter( nn = 10000 )
2: C      real      a(nn),b(nn)
3: C      integer   ia(nn),ib(nn)
4: C      real*8    d(nn),bd(nn)
5: C      equivalence (a(1),d(1),ia(1))
6: C      equivalence (b(1),bd(1),ib(1))
7: C
8: C      include 'INC/_WORDL'
9: C
10: C      character*128 ch
11: C
12: C      character*4 type
13: C      integer   iret(100)
14: C
15: C      mword = 2
16: C      mword = 1
17: C
18: C      call pack00( a, 'PACKET CONTAINER TEST', nn, iret(0))
19: C
20: C      n = 100
21: C      do 100 i=1,n
22: C          b(i) = i
23: C          ib(i+n) = i
24: C          bd(i+2*n) = i
25: C 100 continue
26: C
27: C      ch = '[PACKET OF CHARACTER TYPE] '//char(0)
28: C      lch = index(ch,char(0))-1
29: C      call packnd(a,'A', 'R4', b(1), n, iret(1) )
30: C      call packnd(a,'IA', 'I4', ib(n+1), n, iret(2) )
31: C      call packlb(a,'BEFORE D', 'BEFORE D', iret(3) )
32: C      call packnd(a,'D', 'R8', bd(2*n+1), n, iret(4) )
33: C      call packcs(a,'CH', ch(:lch), iret(5) )
34: C
35: C      write(6,*) 'IRETS ',(iret(i),i=1,5)
36: C
37: C      call pctcls(a,'CLOSE', nn2, nsize, ir )
38: C      write(6,*) ' close nn2 nsize ir ',nn2,nsize,ir
39: C
40: C      do 200 i=1,nn
41: C          b(i) = 0
42: C 200 continue
43: C
44: C      ... non existent label ...
45: C      call pctlb(a,'LABEL', 'INVALID',ir )
46: C      write(6,*) ' ir ',ir
47: C      call pctlb(a,'BEFORE D', 'BEFORE D', ir )
48: C      write(6,*) ' ir ',ir
49: C
50: C      call pctchk(a,'D?', type, lp, ndata, ir )
51: C      write(6,*) ir,' type ',type,' lp ', lp,' ndata ',ndata
52: C
53: C      call unpknd(a,'D', 'R8', bd, n, ir )
54: C      write(6,*) ir,' bd ',(bd(i),i=1,n)
55: C
56: C      call unpkpt(a,'CH?', type, lp, ndata, ir )
57: C      write(6,*) ir,' type ',type,' ndata ',ndata
58: C
59: C      call pctrw(a,'REWIND',ir)
60: C      write(6,*) 'rewind ',ir
61: C
62: C      call unpkpt(a,'a?', type, lp, ndata, ir )
63: C      write(6,*) ir,' type ',type,' lp ', lp,' ndata ',ndata
64: C      write(6,*) ir,' a ',(a(i),i=lp,lp+ndata-1)
65: C

```

```

66: C      call unpknd(a, 'ia?', 'I4', ib(1),n ,ndata, ir )
67: C      write(6,*) ir,' ndata ',ndata
68: C      write(6,*) ' ib ',(ib(i),i=1,ndata)
69: C
70: C      call unpknd(a, 'bd?', 'R8', bd(1),n ,ndata, ir )
71: C      write(6,*) ir,' ndata ',ndata
72: C      write(6,*) ' bd ',(bd(i),i=1,ndata)
73: C
74: C      call unpkcs(a, 'ch?',ch,ndata, ir )
75: C      write(6,*) ir,' ndata ',ndata
76: C      write(6,*) ' ch ',ch(:ndata)
77: C
78: C      end
79: C=<MVP/GMVP>=====
80: C purpose : data packing and unpacking into/from a data packet-container
81: C      A data-packet-container is a contiguous memory area on which
82: C      various types of data are stored as "packets" in arbitrary order
83: C      and length.
84: C calls: mvctoi, mvitoc, cmpchi
85: C=====
86: C
87: C      * data packet must begin on a double word boundary in
88: C      32 bit / word  architecture.
89: C
90: C      * structure of a "packet"
91: C
92: C*****
93: C*** Obsolete ***** after June 1997 *****
94: C*****
95: C***      Packet header in 2 integer (greater than 30 bit precision !!):
96: C***
97: C***      1. <data-type>*2**24 + <packet-size>
98: C***
99: C***          <data-type>  1  integer    2  real
100: C***                      3  real*8    4  character
101: C***                      5  label
102: C***          <packet size> number of data elements (type 1,2, 3)
103: C***                      or number of charaters (type 4 or 5)
104: C***
105: C***      This data is negative for newly packed packet next time.
106: C***      ( it means current end point of container also )
107: C***      -1 : able to add new packet
108: C***      -2 : cannot add new packet (closed container !)
109: C***
110: C***
111: C***      2. pointer (array index to 'packet' array) of the
112: C***      header of the next packet.
113: C*****
114: C
115: C      Packet header in NPHEAD = 4 integer:
116: C
117: C      1. data-type
118: C
119: C          <data-type>  1  integer    2  real
120: C***                  3  real*8    4  character
121: C***                  5  label
122: C
123: C      This data is negative for newly packed packet next time.
124: C      ( it means current end point of container also )
125: C      -1 : able to add new packet
126: C      -2 : cannot add new packet (closed container !)
127: C
128: C      2. packet data size
129: C
130: C          number of data elements (type 1,2, 3)

```

src/mvpburn/packet.f

```

131: C          or number of charaters (type 4 or 5)
132: C
133: C          3. pointer (array index to 'packet' array) of the
134: C          header of the next packet.
135: C          (-1 for the last packet)
136: C
137: C          4. pointer (array index to 'packet' array) of the
138: C          header of the previous packet.
139: C          (-1 for the first packet)
140: C
141: C          Data body immediately follows this header. Real*8 data should
142: C          begin on a double word memory boundary.
143: C
144: C
145: C          Routine (ENTRY) s
146: C
147: C          < packing >
148: C
149: C          pack00 : initialize a data packet container.
150: C          packri : re-initialize a data packet container to enable
151: C                  appending more packets.
152: C          packnd : pack numeric data( I4, R4, R8 )
153: C          packnn : pack a numeric data array repeatedly ( I4, R4, R8 )
154: C          packpt : create a specified size of packet and return position
155: C                  of data instead of storing data(I4,R4,R8)
156: C          packcs : pack a character string
157: C          packlb : put a label-packet
158: C
159: C          < positioning etc.>
160: C
161: C          pctrw : place pointer at the top of a packet container
162: C                  and reset status for use
163: C          pctlb : locate at a packet by giving a label
164: C          pctlb2 : locate at a packet by giving a label (silent)
165: C          pctlbr : locate at a packet by backward label search
166: C          pctcls : close a packet container
167: C          pctchk : get information of current packet
168: C
169: C          < unpacking by copying data >
170: C
171: C          Data packets should be retrieved in order of packing in valid
172: C          data types. Positionig by label may be useful to avoid ambiguity.
173: C
174: C          unpknd : unpack numeric data ( I4, R4, R8 )
175: C          unpkcs : unpack a character string
176: C
177: C          < unpacking or check by getting a pointer >
178: C
179: C          unpkpt : return pointer to packed data & proceed to the next
180: C                  packet
181: C
182: C          < checking status >
183: C
184: C          pcterr : return number of errors
185: C
186: C          < dump >
187: C
188: C          pctdmp : printout contents of container.
189: C
190: C-----
191: C
192: C
193: C          subroutine PACK00( PACKET,MESSAGE,          NPSIZE,IRET )
194: C-----
195: C          Packet container initialization:

```

```

196: C
197: C          packet : memory area used as packet container
198: C                  (it must begin on a double word boundary in 32 bit
199: C                  architecture.)
200: C          npsize : initial size of packet container in word.
201: C                  (on closing container, size information of the container
202: C                  is set to the effective length.)
203: C          iret : return code
204: C                  0 = initialized succesfully
205: C                  1 = not start from double word boundary (not implimented)
206: C                  2 = npsize is insufficient to consturct container.
207: C-----
208: C
209: C          ... packet ....
210: C
211: C          integer PACKET(*)
212: C
213: C          character*(*) MESSAGE
214: C
215: C          ... word length of container header ...
216: C
217: C          parameter( NPHEAD = 4 )
218: C          parameter( LPERR = 5 )
219: C          parameter( LPCUR = 6 )
220: C          parameter( NCHEAD = LPCUR )
221: C
222: C          include 'INC/_IOUNIT'
223: C          include 'INC/_WORDL'
224: C-----
225: C
226: C          ... internal static data ...
227: C
228: C          integer NCON
229: C          data NCON /0/
230: C-----
231: C
232: C          if ( NPSIZE.lt.NCHEAD+NPHEAD ) then
233: C              write(IPR,*) 'XXX(PACK00):needs more memory to prepare data',
234: C              & ' packet container.(container no. ', PACKET(1), ') <',
235: C              & MESSAGE, '>'
236: C              write(IPR,*) ' size = ', NPSIZE
237: C              IRET = 2
238: C              return
239: C          end if
240: C
241: C          NCON = NCON + 1
242: C          PACKET(1) = NCON
243: C          PACKET(2) = NPSIZE
244: C          ... total number of packets ...
245: C          PACKET(3) = 0
246: C          ... current packet number ...
247: C          PACKET(4) = 0
248: C
249: C          ... lperr : position of error counter ...
250: C
251: C          PACKET(LPERR) = 0
252: C
253: C          ... current packet position ...
254: C
255: C          lpcur : position of current pointer !! ...
256: C
257: C          PACKET(LPCUR) = NCHEAD + 1
258: C
259: C          ... positioning to the first packet ...
260: C

```

src/mvpburn/packet.f

```

261:     PACKET(PACKET(LPCUR)) = -1
262:     PACKET(PACKET(LPCUR)+1) = 0
263:     PACKET(PACKET(LPCUR)+2) = -1
264:     PACKET(PACKET(LPCUR)+3) = -1
265: C
266:     IRET = 0
267:     return
268: end
269: C
270: C
271:     subroutine PACKRI( PACKET,MESSAGE, NPSIZE,IRET )
272: C=====
273: C   Packet container initialization for already close container:
274: C
275: C   packet : memory area used as packet container
276: C   (it must begin on a double word boundary in 32 bit
277: C   architecture.)
278: C   npsize : new initial size of packet container in word.
279: C   (on closing container, size information of the container
280: C   was set to the effective length.)
281: C   iret : return code
282: C   0 = initialized succesfully
283: C   1 = not start from double word boundary (not implimented)
284: C   2 = npsize is insufficient to consturuct container.
285: C=====
286: C
287: C   ... packet ....
288: C
289:     integer PACKET(*)
290: C
291:     character*(*) MESSAGE
292: C
293: C   ... word length of container header ...
294: C
295:     parameter( NPHEAD = 4 )
296:     parameter( LPERR = 5 )
297:     parameter( LPCUR = 6 )
298:     parameter( NCHEAD = LPCUR )
299: C
300:     include 'INC/_IOUNIT'
301:     include 'INC/_WORDL'
302: C-----
303:     IRET = 0
304:     if( NPSIZE.lt.PACKET(2) ) then
305:         write(IPR,*)
306:         & 'XXX(PACKRI):failed to re-initilize packet container',
307:         & ' new size ('NPSIZE,') is smaller than current size ('
308:         & PACKET(2),') ',
309:         & ' <',MESSAGE,'>'
310:         IRET = 1
311:         return
312:     end if
313: C
314: C   ... search the last packet checking container integrity ...
315: C
316:     call PCTRW( PACKET,MESSAGE, IRET )
317:     LN = 0
318:     100 LL = PACKET(LPCUR)
319:     JST = PACKET(LL)
320:     LN = LN + 1
321:     if( JST.ne.-2 ) then
322:         if ( JST.eq.-1 ) then
323:             write(IPR,*)
324:             & 'XXX(PACKRI):found a writable packet position.',
325:             & ' this packet container seems not closed.',

```

```

326:         & ' <',message,'>'
327:         write(IPR,*) ' Number of packets encountered ',LN
328:         IRET = 2
329:         return
330:     end if
331: C
332:     LL0 = LL
333:     LL = PACKET(LL+2)
334: C
335:     if( PACKET(LL+3).ne.LL0 ) then
336:         write(IPR,*)
337:         & 'XXX(PACKRI):packet position link is destroyed?.'
338:         write(IPR,*)
339:         & ' Current packet position: ',ll0
340:         write(IPR,*)
341:         & ' next packet position: ',ll
342:         write(IPR,*)
343:         & ' Recorded current packet pos. in next packet: ',
344:         & packet(ll+3)
345:         write(IPR,*) ' PACKRI is Called with message <',message,'>'
346:         IRET = 3
347:         return
348:     end if
349:     PACKET(LPCUR) = LL
350:     PACKET(4) = PACKET(4) + 1
351:     goto 100
352: endif
353: C
354: C   ... reset the last packet as writable
355: C
356:     PACKET(PACKET(LPCUR)) = -1
357: C
358:     PACKET(2) = NPSIZE
359: C
360:     IRET = 0
361:     return
362: end
363: C-----
364: C
365: C   < packing >
366: C
367:     subroutine PACKND( PACKET,MESSAGE, TYPE, DATA, NDATA, IRET )
368: C=====
369: C   : pack numeric data( I4, R4, R8 )
370: C   message : message string (not stored in container, but for
371: C   error message etc.)
372: C   type : data type packed
373: C   'I4' : integer 'R4' : real 'R8' : real*8
374: C   data : packed data ( declared as integer )
375: C   ndata : number of data packed
376: C   iret : return code
377: C   0 = succesful
378: C   1 = cannot pack. Container size insufficient
379: C   2 = cannot pack. Container already closed
380: C   3 = unsupported data type
381: C   4 = too long data to pack
382: C=====
383: C   ... packet ....
384: C
385:     integer PACKET(*)
386: C
387: C   ... data packed / unpcked ...
388: C
389:     integer DATA(*)
390:     character*(*) MESSAGE

```


src/mvpburn/packet.f

```

391: C
392:     character(*) TYPE
393: C
394: C ... word length of container header ...
395: C
396:     parameter( NPHEAD = 4 )
397: Cccc parameter( LPCUR = 5 ) ! Sep 1995
398:     parameter( LPERR = 5 )
399:     parameter( LPCUR = 6 )
400:     parameter( NCHEAD = LPCUR )
401: C
402:     include 'INC/_IOUNIT'
403:     include 'INC/_WORDL'
404: C-----
405:     LL = PACKET(LPCUR)
406:     LH1 = PACKET(LL)
407: C
408:     if ( LH1.eq.-2 ) then
409:         write(IPR,*)
410:         & 'XXX(PACKND):you are trying to add new packet to closed',
411:         & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
412:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
413:         IRET = 2
414:         PACKET(LPERR) = PACKET(LPERR) + 1
415:         return
416:     end if
417: C
418:     LP = LL + NPHEAD
419:     ND = NDATA
420:     if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
421:         ND = NDATA*MWORD
422:         if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
423:     end if
424: C
425:     LDS = LP + ND
426:     LDLAST = LDS + NPHEAD - 1
427:     if ( LDLAST.gt.PACKET(2) ) then
428:         write(IPR,*) 'XXX(PACKND):packet container overflow',
429:         & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
430:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
431:         IRET = 1
432:         PACKET(LPERR) = PACKET(LPERR) + 1
433:         return
434:     end if
435: C
436:     if ( TYPE.eq.'I4' ) then
437:         ITYP = 1
438:     else if ( TYPE.eq.'R4' ) then
439:         ITYP = 2
440:     else if ( TYPE.eq.'R8' ) then
441:         ITYP = 3
442:     else
443:         write(IPR,*)
444:         & 'XXX(PACKND):tried to pack data of invalid type to the',
445:         & ' data container no. ', PACKET(1), ' <', MESSAGE, '>'
446:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
447:         IRET = 3
448:         PACKET(LPERR) = PACKET(LPERR) + 1
449:         return
450:     end if
451:     PACKET(LL) = ITYP
452:     PACKET(LL+1) = NDATA
453:     PACKET(LL+2) = LDS
454: C
455:     PACKET(3) = PACKET(3) + 1

```

```

456:     PACKET(4) = PACKET(4) + 1
457:     PACKET(LPCUR) = LDS
458: C
459: C ... for next packet
460: C
461:     PACKET(LDS) = -1
462:     PACKET(LDS+1) = 0
463:     PACKET(LDS+2) = -1
464:     PACKET(LDS+3) = LL
465: C
466:     do 100 I = 1, ND
467:         PACKET(LP-1+I) = DATA(I)
468:     100 continue
469: C
470:     IRET = 0
471:     return
472: end
473: C
474:     subroutine PACKNN( PACKET,MESSAGE, TYPE, DATA, NDATA, NC, IRET )
475: C=====
476: C : pack a numeric array DATA(NDATA) NC times in a packet
477: C ( I4, R4, R8 )
478: C
479: C Used to create a packet having repeated data structure without
480: C copying data in caller side.
481: C
482: C message : message string (not stored in container, but for
483: C error message etc.)
484: C type : data type packed
485: C 'I4' : integer 'R4' : real 'R8' : real*8
486: C data : packed data ( declared as integer )
487: C ndata : size of data array used as a repeated packing pattern.
488: C nc : data repetition count
489: C iret : return code
490: C 0 = succesful
491: C 1 = cannot pack. Container size insufficient
492: C 2 = cannot pack. Container already closed
493: C 3 = unsupported data type
494: C 4 = too long data to pack
495: C 5 = repetition count is negative
496: C=====
497: C ... packet ....
498: C
499:     integer PACKET(*)
500: C
501: C ... data packed / unpcked ...
502: C
503:     integer DATA(*)
504:     character(*) MESSAGE
505: C
506:     character(*) TYPE
507: C
508: C ... word length of container header ...
509: C
510:     parameter( NPHEAD = 4 )
511: Cccc parameter( LPCUR = 5 ) ! Sep 1995
512:     parameter( LPERR = 5 )
513:     parameter( LPCUR = 6 )
514:     parameter( NCHEAD = LPCUR )
515: C
516:     include 'INC/_IOUNIT'
517:     include 'INC/_WORDL'
518: C-----
519:     LL = PACKET(LPCUR)
520:     LH1 = PACKET(LL)

```

src/mvpburn/packet.f

```

521: C
522:   if ( LHL.eq.-2 ) then
523:     write(IPR,*)
524:   &   'XXX(PACKNN):you are trying to add new packet to closed',
525:   &   ' container no. ', PACKET(1), ' <', MESSAGE, '>'
526:   write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
527:   IRET = 2
528:   PACKET(LPERR) = PACKET(LPERR) + 1
529:   return
530: end if
531: C
532:   if ( NC.lt.0 ) then
533:     write(IPR,*)
534:   &   'XXX(PACKNN):data repetetion count for packed data',
535:   &   ' is negative. NC=',NC, ' <', MESSAGE, '>'
536:   IRET = 5
537:   return
538: end if
539: C
540:   LP = LL + NPHEAD
541:   ND = NDATA * NC
542:   if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
543:     ND = NDATA*MWORD*NC
544:   if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
545: end if
546: C
547:   LDS = LP + ND
548:   LDLAST = LDS + NPHEAD - 1
549:   if ( LDLAST.gt.PACKET(2) ) then
550:     write(IPR,*) 'XXX(PACKNN):packet container overflow.',
551:   &   ' container no. ', PACKET(1), ' <', MESSAGE, '>'
552:   write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
553:   IRET = 1
554:   PACKET(LPERR) = PACKET(LPERR) + 1
555:   return
556: end if
557: C
558:   if ( TYPE.eq.'I4' ) then
559:     ITYP = 1
560:   else if ( TYPE.eq.'R4' ) then
561:     ITYP = 2
562:   else if ( TYPE.eq.'R8' ) then
563:     ITYP = 3
564:   else
565:     write(IPR,*) 'XXX(PACKNN):tried to pack data of invalid type',
566:   &   ' to the data container no. ', PACKET(1), ' <', MESSAGE, '>'
567:     write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
568:     IRET = 3
569:     PACKET(LPERR) = PACKET(LPERR) + 1
570:     return
571:   end if
572:   PACKET(LL) = ITYP
573:   PACKET(LL+1) = NDATA
574:   PACKET(LL+2) = LDS
575: C
576:   PACKET(3) = PACKET(3) + 1
577:   PACKET(4) = PACKET(4) + 1
578:   PACKET(LPCUR) = LDS
579: C
580: C ... for next packet
581: C
582:   PACKET(LDS) = -1
583:   PACKET(LDS+1) = 0
584:   PACKET(LDS+2) = -1
585:   PACKET(LDS+3) = LL

```

```

586: C
587:   JJ = 0
588:   if( NC.gt.0 ) then
589:     do 110 J = 1, NC
590:       do 100 I = 1, ND/NC
591:         PACKET(LP-1+JJ+I) = DATA(I)
592:       100 continue
593:       JJ = JJ + ND/NC
594:     110 continue
595:   end if
596: C
597:   IRET = 0
598:   return
599: end
600: C
601:   subroutine PACKPT( PACKET,MESSAGE, TYPE, LP, NDATA, IRET )
602: C=====
603: C : get an index pointer to store numeric data( I4, R4, R8 ) as
604: C a packet.
605: C
606: C Data placement on the data area should be done afterwards
607: C and take care not to destroy data structure.
608: C This is useful for a case to place data directly from a file;
609: C
610: C ex:
611: C call packpt( packet, 'integer', 'R4', LP, 1000, IRET )
612: C if( IRET.eq.0 ) read(10) (packet(lp+i-1),i=1,1000)
613: C
614: C An index pointer for 'R8' data is returned as an index for
615: C PACKET(*) array declared as double word type.
616: C
617: C ex.
618: C
619: C real PACKET(100000)
620: C real*8 DPACKET(100000)
621: C equivalence (PACKET,DPACKET)
622: C ...
623: C call packpt( packet, 'integer', 'R4', LP1, 1000, IRET )
624: C if( IRET.eq.0 ) read(10) (PACKET(LP1+I-1),I=1,1000)
625: C call packpt( packet, 'integer', 'R8', LP2, 200, IRET )
626: C if( IRET.eq.0 ) read(10) (DPACKET(LP2+I-1),I=1,200)
627: C
628: C .....
629: C
630: C message : message string (not stored in container, but for
631: C error message etc.)
632: C type : data type packed
633: C 'I4' : integer 'R4' : real 'R8' : real*8
634: C lp : index pointer of data to be packed afterwards.
635: C ndata : number of data packed
636: C iret : return code
637: C 0 = succesful
638: C 1 = cannot pack. Container size insufficient
639: C 2 = cannot pack. Container already closed
640: C 3 = unsupported data type
641: C 4 = too long data to pack
642: C=====
643: C ... packet ....
644: C
645: C integer PACKET(*)
646: C
647: C ... data packed / unpacked ...
648: C
649: C ccccc integer DATA(*)
650: C character*(*) MESSAGE

```

src/mvpburn/packet.f

```

651: C
652:     character(*) TYPE
653: C
654: C ... word length of container header ...
655: C
656:     parameter( NPHEAD = 4 )
657: Cccc parameter( LPCUR = 5 ) ! Sep 1995
658:     parameter( LPERR = 5 )
659:     parameter( LPCUR = 6 )
660:     parameter( NCHEAD = LPCUR )
661: C
662:     include 'INC/_IOUNIT'
663:     include 'INC/_WORDL'
664: C-----
665:     LL = PACKET(LPCUR)
666:     LH1 = PACKET(LL)
667: C
668:     if ( LH1.eq.-2 ) then
669:         write(IPR,*) 'XXX(PACKPT):you are trying to add new packet ',
670: & 'to closed container no. ', PACKET(1), ' <', MESSAGE, '>'
671:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
672:         IRET = 2
673:         PACKET(LPERR) = PACKET(LPERR) + 1
674:         return
675:     end if
676: C
677:     LP = LL + NPHEAD
678:     ND = NDATA
679:     if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
680:         ND = NDATA*MWORD
681:         if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
682:     end if
683: C
684:     LDS = LP + ND
685:     LDLAST = LDS + NPHEAD - 1
686:     if ( LDLAST.gt.PACKET(2) ) then
687:         write(IPR,*) 'XXX(PACKPT):packet container overflow.',
688: & ' container no. ', PACKET(1), ' <', MESSAGE, '>'
689:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
690:         IRET = 1
691:         PACKET(LPERR) = PACKET(LPERR) + 1
692:         return
693:     end if
694: C
695:     if ( TYPE.eq.'I4' ) then
696:         ITYP = 1
697:     else if ( TYPE.eq.'R4' ) then
698:         ITYP = 2
699:     else if ( TYPE.eq.'R8' ) then
700:         ITYP = 3
701:     else
702:         write(IPR,*) 'XXX(PACKPT):tried to pack data of invalid type',
703: & ' to the data container no. ', PACKET(1), ' <', MESSAGE, '>'
704:         write(IPR,*) ' TYPE ', TYPE, ' ndata = ', NDATA
705:         IRET = 3
706:         PACKET(LPERR) = PACKET(LPERR) + 1
707:         return
708:     end if
709:     PACKET(LL) = ITYP
710:     PACKET(LL+1) = NDATA
711:     PACKET(LL+2) = LDS
712: C
713:     PACKET(3) = PACKET(3) + 1
714:     PACKET(4) = PACKET(4) + 1
715:     PACKET(LPCUR) = LDS

```

```

716: C
717:     PACKET(LDS) = -1
718:     PACKET(LDS+1) = 0
719:     PACKET(LDS+2) = -1
720:     PACKET(LDS+3) = LL
721: C
722:     if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
723:         LP = LP/MWORD + MWORD - 1
724:     end if
725: C
726:     IRET = 0
727:     return
728:     end
729: C=====
730: C packcs : pack a character string
731: C pcklb : tuck a label-packet
732: C message : message string (not stored in container, but for
733: C error message etc.)
734: C chstr : packed character string or label string (assumed length)
735: C iret : return code
736: C 0 = succesful
737: C 1 = cannot pack. Container size insufficient
738: C 2 = cannot pack. Container already closed
739: C 4 = too long data to pack
740: C=====
741:     subroutine PACKCS( PACKET,MESSAGE, CHSTR, IRET )
742: C
743: C ... packet ....
744: C
745: C integer PACKET(*)
746: C
747: C character*(*) MESSAGE
748: C character*(*) CHSTR
749: C
750: C ... word length of container header ...
751: C
752:     parameter( NPHEAD = 4 )
753: CCCCC parameter( LPCUR = 5 ) ! Sep 1995
754:     parameter( LPERR = 5 )
755:     parameter( LPCUR = 6 )
756:     parameter( NCHEAD = LPCUR )
757: C
758:     include 'INC/_IOUNIT'
759:     include 'INC/_WORDL'
760: C-----
761:     ITYP = 4
762: C
763:     LL = PACKET(LPCUR)
764:     LH1 = PACKET(LL)
765: C
766:     if ( LH1.eq.-2 ) then
767:         write(IPR,*) 'XXX(PACKCS):you are trying to add new packet ',
768: & 'to closed container no. ', PACKET(1), ' <', MESSAGE, '>'
769:         write(IPR,*) ' STRING <', CHSTR, '>'
770:         IRET = 2
771:         PACKET(LPERR) = PACKET(LPERR) + 1
772:         return
773:     end if
774: C
775:     LP = LL + NPHEAD
776: C
777:     KWD = 8/MWORD
778:     ND = (LEN(CHSTR)+KWD-1)/KWD
779:     LDS = LP + ND
780:     LDLAST = LDS + NPHEAD - 1

```

src/mvpcburn/packet.f

```

781:      if ( LDLAST.gt.PACKET(2) ) then
782:        write(IPR,*) 'XXX(PACKCS):packet container overflow.',
783:        &      ' container no. ', PACKET(1), ' <', MESSAGE, '>'
784:        write(IPR,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
785:        &      LEN(CHSTR)
786:        IRET = 1
787:        PACKET(LPERR) = PACKET(LPERR) + 1
788:        return
789:      end if
790: C
791:      PACKET(LL) = ITYP
792:      PACKET(LL+1) = LEN(CHSTR)
793:      PACKET(LL+2) = LDS
794: C
795:      PACKET(3) = PACKET(3) + 1
796:      PACKET(4) = PACKET(4) + 1
797:      PACKET(LPCUR) = LDS
798: C
799:      PACKET(LDS) = -1
800:      PACKET(LDS+1) = 0
801:      PACKET(LDS+2) = -1
802:      PACKET(LDS+3) = LL
803: C
804: C ... data transfer by internal read-write may cause compilation or
805: C execution error for too large data.
806: C
807:      call MVCTOI( LEN(CHSTR), CHSTR, PACKET(LP) )
808: C
809:      IRET = 0
810:      return
811:    end
812: C=====
813:      subroutine PACKLB( PACKET,MESSAGE, CHSTR, IRET )
814: C
815: C ... packet ....
816: C
817:      integer PACKET(*)
818: C
819:      character*(*) MESSAGE
820:      character*(*) CHSTR
821: C
822: C ... word length of container header ...
823: C
824:      parameter( NPHEAD = 4 )
825: Cccc parameter( LPCUR = 5 ) ! Sep 1995
826:      parameter( LPERR = 5 )
827:      parameter( LPCUR = 6 )
828:      parameter( NCHEAD = LPCUR )
829: C
830:      include 'INC/_IOUNIT'
831:      include 'INC/_WORDL'
832: C-----
833:      ITYP = 5
834:      LL = PACKET(LPCUR)
835:      LH1 = PACKET(LL)
836: C
837:      if ( LH1.eq.-2 ) then
838:        write(IPR,*) 'XXX(PACKLB):you are trying to add new packet ',
839:        &      'to closed container no. ', PACKET(1), ' <', MESSAGE, '>'
840:        write(IPR,*) ' STRING <', CHSTR, '>'
841:        IRET = 2
842:        PACKET(LPERR) = PACKET(LPERR) + 1
843:        return
844:      end if
845: C

```

```

846:      LP = LL + NPHEAD
847: C
848:      KWD = 8/MWORD
849:      ND = (LEN(CHSTR)+KWD-1) /KWD
850:      LDS = LP + ND
851:      LDLAST = LDS + NPHEAD - 1
852:      if ( LDLAST.gt.PACKET(2) ) then
853:        write(IPR,*) 'XXX(PACKLB):packet container overflow.',
854:        &      ' container no. ', PACKET(1), ' <', MESSAGE, '>'
855:        write(IPR,*) ' STRING(1:72) <', CHSTR(1:72), '>', ' LENGTH ',
856:        &      LEN(CHSTR)
857:        IRET = 1
858:        PACKET(LPERR) = PACKET(LPERR) + 1
859:        return
860:      end if
861: C
862:      PACKET(LL) = ITYP
863:      PACKET(LL+1) = LEN(CHSTR)
864:      PACKET(LL+2) = LDS
865: C
866:      PACKET(3) = PACKET(3) + 1
867:      PACKET(4) = PACKET(4) + 1
868:      PACKET(LPCUR) = LDS
869: C
870:      PACKET(LDS) = -1
871:      PACKET(LDS+1) = 0
872:      PACKET(LDS+2) = -1
873:      PACKET(LDS+3) = LL
874: C
875:      call MVCTOI( LEN(CHSTR), CHSTR, PACKET(LP) )
876: C
877:      IRET = 0
878:      return
879:    end
880: C-----
881: C
882: C < positioning etc.>
883: C
884:      subroutine PCTRW( PACKET,MESSAGE, IRET )
885: C=====
886: C : place pointer at the top of a packet container and reset status
887: C for use
888: C=====
889: C
890: C ... packet ....
891: C
892:      integer PACKET(*)
893: C
894:      character*(*) MESSAGE
895: C
896: C ... word length of container header ...
897: C
898:      parameter( NPHEAD = 4 )
899:      parameter( LPERR = 5 )
900:      parameter( LPCUR = 6 )
901:      parameter( NCHEAD = LPCUR )
902: C
903:      include 'INC/_IOUNIT'
904:      include 'INC/_WORDL'
905: C-----
906:      PACKET(4) = 0
907:      PACKET(LPCUR) = NCHEAD + 1
908:      IRET = 0
909:      return
910:    end

```

src/mvpburn/packet.f

```

911: C-----
912:      subroutine PCTLB( PACKET,MESSAGE,      CHSTR, IRET )
913: C=====
914: C   : locate at a packet by giving a label
915: C      chstr : label
916: C      iret  : return code
917: C          0 = found record successfully
918: C          1 = no such a label
919: C-----
920: C
921: C   ... packet ....
922: C
923: C      integer PACKET(*)
924: C
925: C      character*(*) MESSAGE
926: C      character*(*) CHSTR
927: C      call PCTLB0( PACKET,MESSAGE,      CHSTR, 0, IRET )
928: C      return
929: C      end
930: C-----
931:      subroutine PCTLB2( PACKET,MESSAGE,      CHSTR, IRET )
932: C=====
933: C   : locate at a packet by giving a label (do not print error message
934: C       if the label is not found )
935: C      chstr : label
936: C      iret  : return code
937: C          0 = found record successfully
938: C          1 = no such a label
939: C-----
940: C
941: C   ... packet ....
942: C
943: C      integer PACKET(*)
944: C
945: C      character*(*) MESSAGE
946: C      character*(*) CHSTR
947: C      call PCTLB0( PACKET,MESSAGE,      CHSTR, 1, IRET )
948: C      return
949: C      end
950: C-----
951:      subroutine PCTLB0( PACKET,MESSAGE,      CHSTR, isilnt, IRET )
952: C=====
953: C   : locate at a packet by giving a label
954: C      chstr : label
955: C      isilnt : 1 = do not print error message if specified label is
956: C                not found.
957: C                0 = print error message
958: C      iret  : return code
959: C          0 = found record successfully
960: C          1 = no such a label
961: C-----
962: C
963: C   ... packet ....
964: C
965: C      integer PACKET(*)
966: C
967: C      character*(*) MESSAGE
968: C      character*(*) CHSTR
969: C
970: C   ... word length of container header ...
971: C
972: C      parameter( NPHEAD = 4 )
973: C      Cccc parameter( LPCUR = 5 ) ! Sep 1995
974: C      parameter( LPERR = 5 )
975: C      parameter( LPCUR = 6 )

```

```

976:      parameter( NCHEAD = LPCUR )
977: C
978: C      include 'INC/_IOUNIT'
979: C      include 'INC/_WORDL'
980: C-----
981: C      LC      = PACKET(LPCUR)
982: C      IP      = PACKET(4)
983: C      LROT    = 0
984: C      100 continue
985: C      LL      = PACKET(LPCUR)
986: C      LH1     = PACKET(LL)
987: C      if ( LH1.gt.0 ) then
988: C          ITYP = LH1
989: C
990: C      if ( ITYP.eq.5 ) then
991: C          IM    = 0
992: C          ND    = PACKET(LL+1)
993: C          if ( ND.eq.LEN(CHSTR) )
994: C      &      call CMPCHI( ND, IM, CHSTR, PACKET(LL+NPHEAD) )
995: C
996: C      .... label found ....
997: C
998: C      if ( IM.eq.1 ) then
999: C          IRET = 0
1000: C          PACKET(4) = PACKET(4) + 1
1001: C          PACKET(LPCUR) = PACKET(LL+2)
1002: C          return
1003: C      end if
1004: C      end if
1005: C      PACKET(4) = PACKET(4) + 1
1006: C      PACKET(LPCUR) = PACKET(LL+2)
1007: C      go to 100
1008: C      else
1009: C
1010: C      ... not found ...
1011: C
1012: C      if ( LROT.ne.0 ) then
1013: C          IRET = 1
1014: C          if ( ISILNT.eq.0 ) then
1015: C              PACKET(LPERR) = PACKET(LPERR) + 1
1016: C              write(IPR,*) 'XXX(PACKRW):can not find label <', CHSTR,
1017: C      &      '> in data packet ', ' container No. ', PACKET(1),
1018: C      &      ' <', MESSAGE, '>'
1019: C          end if
1020: C          PACKET(4) = IP
1021: C          PACKET(LPCUR) = LC
1022: C          return
1023: C      else
1024: C          LROT = 1
1025: C          PACKET(4) = 1
1026: C          PACKET(LPCUR) = NCHEAD + 1
1027: C          go to 100
1028: C      end if
1029: C      end if
1030: C      IRET = 0
1031: C      return
1032: C      end
1033: C-----
1034:      subroutine PCTLBR( PACKET,MESSAGE,      CHSTR, isilnt, IRET )
1035: C=====
1036: C   : locate at a packet by giving a label and make backward search from
1037: C      current position.
1038: C      chstr : label
1039: C      isilnt : 1 = do not print error message if specified label is
1040: C                not found.

```

src/mvpburn/packet.f

```

1041: C          0 = print error message
1042: C      iret : return code
1043: C          0 = found record successfully
1044: C          1 = no such a label
1045: C=====
1046: C
1047: C      ... packet ....
1048: C
1049: C      integer PACKET(*)
1050: C
1051: C      character*(*) MESSAGE
1052: C      character*(*) CHSTR
1053: C
1054: C      ... word length of container header ...
1055: C
1056: C      parameter( NPHEAD = 4 )
1057: C      parameter( LPERR  = 5 )
1058: C      parameter( LPCUR  = 6 )
1059: C      parameter( NCHEAD = LPCUR )
1060: C
1061: C      include 'INC/_IUNIT'
1062: C      include 'INC/_WORDL'
1063: C-----
1064: C      LC      = PACKET(LPCUR)
1065: C      IP      = PACKET(4)
1066: C      LROT    = 0
1067: C      100 continue
1068: C      LL      = PACKET(LPCUR)
1069: C      ITYP    = PACKET(LL)
1070: C      if ( ITYP.eq.5 ) then
1071: C          IM      = 0
1072: C          ND      = PACKET(LL+1)
1073: C          if ( ND.eq.LEN(CHSTR) ) then
1074: C              call CMPCHI( ND, IM, CHSTR, PACKET(LL+NPHEAD) )
1075: C          end if
1076: C
1077: C      .... label found ....
1078: C
1079: C      if ( IM.eq.1 ) then
1080: C          IRET     = 0
1081: C          PACKET(4) = PACKET(4) + 1
1082: C          PACKET(LPCUR) = PACKET(LL+2)
1083: C          return
1084: C      end if
1085: C      end if
1086: C
1087: C      PACKET(4) = PACKET(4) - 1
1088: C      PACKET(LPCUR) = PACKET(LL+3)
1089: C
1090: C      if(PACKET(4).gt.0.and.PACKET(LPCUR).gt.0) go to 100
1091: C
1092: C      ... not found ...
1093: C
1094: C      if( PACKET(4).le.0 ) then
1095: C          IRET     = 1
1096: C          if ( ISILNT.eq.0 ) then
1097: C              PACKET(LPERR) = PACKET(LPERR) + 1
1098: C              write(IPR,*) 'XXX(PCTLBR):can not find label <', CHSTR,
1099: C      &              '>' in data packet ', ' container No. ', PACKET(1),
1100: C      &              ' <', MESSAGE, '>'
1101: C          end if
1102: C          PACKET(4) = IP
1103: C          PACKET(LPCUR) = LC
1104: C          return
1105: C      end if

```

```

1106: C
1107: C      IRET     = 0
1108: C      return
1109: C      end
1110: C-----
1111: C      subroutine PCTCLS( PACKET,MESSAGE,      NPSIZE,NSIZE, IRET )
1112: C-----
1113: C      : close a packet container , and return effective size of container.
1114: C
1115: C      npsize : given size in initialization (output)
1116: C      nsize  : effective size (output)
1117: C-----
1118: C
1119: C      ... packet ....
1120: C
1121: C      integer PACKET(*)
1122: C
1123: C      character*(*) MESSAGE
1124: C
1125: C      ... word length of container header ...
1126: C
1127: C      parameter( NPHEAD = 4 )
1128: C      Cccc parameter( LPCUR = 5 ) ! Sep 1995
1129: C      parameter( LPERR  = 5 )
1130: C      parameter( LPCUR  = 6 )
1131: C      parameter( NCHEAD = LPCUR )
1132: C
1133: C      include 'INC/_IUNIT'
1134: C      include 'INC/_WORDL'
1135: C-----
1136: C      LL      = PACKET(LPCUR)
1137: C      IP      = PACKET(4)
1138: C      100 continue
1139: C      if ( PACKET(LL).gt.0 ) then
1140: C          LL      = PACKET(LL+2)
1141: C          IP      = IP + 1
1142: C          go to 100
1143: C      end if
1144: C
1145: C      if ( PACKET(LL).eq.-1 ) then
1146: C          NPSIZE = PACKET(2)
1147: C          PACKET(2) = LL + NPHEAD - 1
1148: C          NSIZE = PACKET(2)
1149: C          PACKET(LL) = -2
1150: C          IRET     = 0
1151: C          return
1152: C      else if ( PACKET(LL).eq.-2 ) then
1153: C          write(IPR,*) 'XXX(PCTCLS):can not close data packet container ',
1154: C      &          PACKET(1), ' <', MESSAGE, '>'
1155: C          write(IPR,*) ' LOCATION ', LL, ' header ', PACKET(LL),
1156: C      &          PACKET(LL+1),PACKET(LL+2),PACKET(LL+3),
1157: C      &          ' PACKET # ', IP
1158: C          IRET     = 1
1159: C          PACKET(LPERR) = PACKET(LPERR) + 1
1160: C          return
1161: C      else
1162: C          write(IPR,*) 'XXX(PCTCLS):invalid packet header ', PACKET(LL),
1163: C      &          ' <',MESSAGE, '>'
1164: C          IRET     = 999
1165: C          PACKET(LPERR) = PACKET(LPERR) + 1
1166: C      end if
1167: C
1168: C      return
1169: C      end
1170: C-----

```

src/mvpcburn/packet.f

```

1171: C
1172: C < get information of currently positioned packet >
1173: C
1174: C subroutine PCTCHK( PACKET,MESSAGE, TYPE, LPT, NDATA, IRET
1175: C & )
1176: C=====
1177: C : check contents of currently positioned packet
1178: C returns pointer & data size but does not proceed to next packet.
1179: C (does not skip label packet )
1180: C
1181: C type : return data type of packet
1182: C 'I4','R4','R8','CH','LB' or blank for end of container
1183: C lpt : pointer to data portion of packet. ( packet(ipt) )
1184: C for 'R8' type, lpt is array index to real*8 array
1185: C having the same starting address with 'packet'.
1186: C ndata : number of data packed in data packet.
1187: C iret : return code
1188: C 0 : successful
1189: C 99 : invalid packet!
1190: C=====
1191: C
1192: C ... packet ...
1193: C
1194: C integer PACKET(*)
1195: C
1196: C character*(*) MESSAGE
1197: C character*(*) TYPE
1198: C
1199: C ... word length of container header ...
1200: C
1201: C parameter( NPHEAD = 4 )
1202: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1203: C parameter( LPERR = 5 )
1204: C parameter( LPCUR = 6 )
1205: C parameter( NCHEAD = LPCUR )
1206: C
1207: C include 'INC/_IOUNIT'
1208: C include 'INC/_WORDL'
1209: C-----
1210: C NDATA = 0
1211: C LPT = 0
1212: C LL = PACKET(LPCUR)
1213: C
1214: C LH1 = PACKET(LL)
1215: C if ( LH1.lt.0 ) then
1216: C IRET = 0
1217: C TYPE = ' '
1218: C return
1219: C end if
1220: C
1221: C ITYP = PACKET(LL)
1222: C
1223: C if ( ITYP.lt.1 .or. ITYP.gt.5 ) then
1224: C write(IPR,*) 'XXX(PCTCHK):unsupported type of data in packet ',
1225: C & ' packet container ', PACKET(1), ' packet # ',
1226: C & PACKET(4), ' <', MESSAGE, '>'
1227: C write(IPR,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1228: C IRET = 99
1229: C PACKET(LPERR) = PACKET(LPERR) + 1
1230: C return
1231: C end if
1232: C
1233: C if ( ITYP.eq.1 ) TYPE = 'I4'
1234: C if ( ITYP.eq.2 ) TYPE = 'R4'
1235: C if ( ITYP.eq.3 ) TYPE = 'R8'

```

```

1236: C if ( ITYP.eq.4 ) TYPE = 'CH'
1237: C if ( ITYP.eq.5 ) TYPE = 'LB'
1238: C
1239: Cccc NDATA = LH1 - ITYP*2**24
1240: C NDATA = PACKET(LL+1)
1241: C LPT = LL + NPHEAD
1242: C
1243: C if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
1244: C if ( MOD(LPT,MWORD).ne.1 ) LPT = LPT + 1
1245: C LPT = LPT/MWORD + MWORD - 1
1246: C end if
1247: C
1248: C IRET = 0
1249: C return
1250: C end
1251: C-----
1252: C
1253: C < return the number of error >
1254: C
1255: C subroutine PCTERR( PACKET,MESSAGE, NERR, IRET )
1256: C=====
1257: C : return the number of error
1258: C
1259: C nerr :
1260: C iret : return code
1261: C 0 : successful
1262: C-----
1263: C
1264: C ... packet ...
1265: C
1266: C integer PACKET(*)
1267: C
1268: C character*(*) MESSAGE
1269: Cccc character*(*) TYPE
1270: C
1271: C ... word length of container header ...
1272: C
1273: C parameter( NPHEAD = 4 )
1274: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1275: C parameter( LPERR = 5 )
1276: C parameter( LPCUR = 6 )
1277: C parameter( NCHEAD = LPCUR )
1278: C
1279: C-----
1280: C IRET = 0
1281: C NERR = PACKET(LPERR)
1282: C return
1283: C end
1284: C-----
1285: C
1286: C < unpacking by copying data >
1287: C
1288: C Data packets should be retrieved in order of packing with valid
1289: C data types. Positionig by label may be useful to avoid ambiguity.
1290: C
1291: C-----
1292: C subroutine UNPKND( PACKET,MESSAGE, TYPE, DATA, NDATA,
1293: C & NDATA2,IRET )
1294: C=====
1295: C : unpack numeric data ( I4, R4, R8 )
1296: C message : message string (not stored in container, but for
1297: C error message etc.)
1298: C type : data type packed
1299: C 'I4' : integer 'R4' : real 'R8' : real*8
1300: C data : packed data ( declared as integer )

```

src/mvppburn/packet.f

```

1301: C      ndata : number of data to be unpacked
1302: C      ndata2 : number of data really unpacked
1303: C      iret : return code
1304: C      0 = successful ( ndata2 <= ndata )
1305: C      1 = no more data packet in this container.
1306: C      2 = data packet includes more data than ndata.
1307: C      3 = packet itself is invalid.
1308: C      -N = data type mismatch.
1309: C      N is type of data in packet
1310: C      ( 1/2/3/4 = I4/R4/R8/CHAR )
1311: C-----
1312: C
1313: C      ... packet ....
1314: C
1315: C      integer PACKET(*)
1316: C
1317: C      character*(*) MESSAGE
1318: C      character*(*) TYPE
1319: C      integer DATA(*)
1320: C
1321: C      ... word length of container header ...
1322: C
1323: C      parameter( NPHEAD = 4 )
1324: C      parameter( LPCUR = 5 ) ! Sep 1995
1325: C      parameter( LPERR = 5 )
1326: C      parameter( LPCUR = 5 )
1327: C      parameter( NCHEAD = LPCUR )
1328: C
1329: C      include 'INC/_IOUNIT'
1330: C      include 'INC/_WORDL'
1331: C-----
1332: C      IRET = 0
1333: C      NDATA2 = 0
1334: C      100 LL = PACKET(LPCUR)
1335: C
1336: C      LH1 = PACKET(LL)
1337: C      if ( LH1.lt.0 ) then
1338: C      IRET = 1
1339: C      PACKET(LPERR) = PACKET(LPERR) + 1
1340: C      write(IPR,*) 'XXX(UNPKND):can not find data <', MESSAGE,
1341: C      & ' > of type <',TYPE, '> in data packet ', ' container No. ',
1342: C      & PACKET(1), ' ', end of container.'
1343: C      return
1344: C      end if
1345: C
1346: C      ITYP = LH1
1347: C
1348: C      ... skip label ...
1349: C
1350: C      if ( ITYP.eq.5 ) then
1351: C      PACKET(LPCUR) = PACKET(LL+2)
1352: C      PACKET(4) = PACKET(4) + 1
1353: C      go to 100
1354: C      end if
1355: C
1356: C      if ( ITYP.lt.1 .or. ITYP.gt.4 ) then
1357: C      write(IPR,*) 'XXX(UNPKND):unsupported type of data in packet ',
1358: C      & ' packet container ', PACKET(1), ' packet # ',
1359: C      & PACKET(4), ' <', MESSAGE, '>'
1360: C      write(IPR,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1361: C      IRET = 3
1362: C      PACKET(LPERR) = PACKET(LPERR) + 1
1363: C      PACKET(LPCUR) = PACKET(LL+2)
1364: C      PACKET(4) = PACKET(4) + 1
1365: C      return

```

```

1366: C      end if
1367: C
1368: C      if ( TYPE.eq.'I4'.and.ITYP.ne.1 .or. TYPE.eq.'R4'
1369: C      & .and.ITYP.ne.2 .or. TYPE.eq.'R8'.and.ITYP.ne.3 ) then
1370: C      IRET = -ITYP
1371: C      PACKET(LPERR) = PACKET(LPERR) + 1
1372: C      write(IPR,*) 'XXX(UNPKND):data <', MESSAGE, '> of type <', TYPE,
1373: C      & ' > required in data packet ', ' container No. ',
1374: C      & PACKET(1), ' but stored type is ', ITYP,
1375: C      & ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1376: C      PACKET(LPCUR) = PACKET(LL+2)
1377: C      PACKET(4) = PACKET(4) + 1
1378: C      return
1379: C      end if
1380: C
1381: C      ND0 = PACKET(LL+1)
1382: C      NDATA2 = MIN(ND0,NDATA)
1383: C      LP = LL + NPHEAD
1384: C      ND = NDATA2
1385: C      if ( TYPE(1:2).eq.'R8'.and.MWORD.gt.1 ) then
1386: C      ND = NDATA2*MWORD
1387: C      if ( MOD(LP,MWORD).ne.1 ) LP = LP + 1
1388: C      ..unnecessary
1389: C      end if
1390: C
1391: C      do 110 I = 1, ND
1392: C      DATA(I) = PACKET(LP+I-1)
1393: C      110 continue
1394: C
1395: C      if ( NDATA.lt.ND0 ) then
1396: C      IRET = 2
1397: C      PACKET(LPERR) = PACKET(LPERR) + 1
1398: C      write(IPR,*) 'XXX(UNPKND):data <', MESSAGE, '> of type <', TYPE,
1399: C      & ' > stored with length ', ND0, ' in data packet ',
1400: C      & ' container No. ', PACKET(1), ' but input buffer is ',
1401: C      & 'small to restore all data (', NDATA, ') '
1402: C      end if
1403: C
1404: C      PACKET(LPCUR) = PACKET(LL+2)
1405: C      PACKET(4) = PACKET(4) + 1
1406: C      IRET = 0
1407: C      return
1408: C      end
1409: C-----
1410: C      subroutine UNPKCS( PACKET,MESSAGE, CHSTR, NDATA2,IRET )
1411: C=====
1412: C : unpack a character string
1413: C      message : message string (not stored in container, but for
1414: C      error message etc.)
1415: C      chstr : unpacked string
1416: C      ndata2 : number of data really unpacked
1417: C      iret : return code
1418: C      0 = successful ( ndata2 <= ndata )
1419: C      1 = no more data packet in this container.
1420: C      2 = data packet includes more data than len(chstr).
1421: C      99 = packet itself is invalid.
1422: C      -N = data type mismatch.
1423: C      N is type of data in packet
1424: C      ( 1/2/3/4 = I4/R4/R8/CHAR )
1425: C-----
1426: C
1427: C      ... packet ....
1428: C
1429: C      integer PACKET(*)
1430: C

```


src/mvpcburn/packet.f

```

1431:      character*(*) MESSAGE
1432:      character*(*) CHSTR
1433: C
1434: C ... word length of container header ...
1435: C
1436:      parameter( NPHEAD = 4 )
1437: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1438:      parameter( LPERR = 5 )
1439:      parameter( LPCUR = 6 )
1440:      parameter( NCHEAD = LPCUR )
1441: C
1442:      include 'INC/_IOUNIT'
1443:      include 'INC/_WORDL'
1444: C-----
1445:      NDATA2 = 0
1446: 100 LL = PACKET(LPCUR)
1447: C
1448:      LH1 = PACKET(LL)
1449:      if ( LH1.lt.0 ) then
1450:      IRET = 1
1451:      PACKET(LPERR) = PACKET(LPERR) + 1
1452:      write(IPR,*) 'XXX(UNPKCS):can not find data <', MESSAGE,
1453: & ' > of string type in data packet ', ' container No. ',
1454: & PACKET(1), ' ', end of container.'
1455:      return
1456:      end if
1457: C
1458:      ITYP = LH1
1459: C
1460: C ... skip label ...
1461: C
1462:      if ( ITYP.eq.5 ) then
1463:      PACKET(LPCUR) = PACKET(LL+2)
1464:      PACKET(4) = PACKET(4) + 1
1465:      go to 100
1466:      end if
1467: C
1468:      if ( ITYP.lt.1 .or. ITYP.gt.4 ) then
1469:      write(IPR,*) 'XXX(UNPKCS):unsupported type of data in packet ',
1470: & ' CONTAINER NO. ', PACKET(1), ' packet # ', PACKET(4),
1471: & ' <', MESSAGE, '>'
1472:      write(IPR,*) ' TYPE ', ITYP, ' location ', PACKET(LPCUR)
1473:      IRET = 99
1474:      PACKET(LPERR) = PACKET(LPERR) + 1
1475:      PACKET(LPCUR) = PACKET(LL+2)
1476:      PACKET(4) = PACKET(4) + 1
1477:      return
1478:      end if
1479: C
1480:      if ( ITYP.ne.4 ) then
1481:      IRET = -ITYP
1482:      PACKET(LPERR) = PACKET(LPERR) + 1
1483:      write(IPR,*) 'XXX(UNPKCS):data <', MESSAGE,
1484: & ' > of type <CH> required in data packet ',
1485: & ' container No. ', PACKET(1), ' but stored type is ',
1486: & ITYP, ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1487:      PACKET(LPCUR) = PACKET(LL+2)
1488:      PACKET(4) = PACKET(4) + 1
1489:      return
1490:      end if
1491: C
1492:      ND0 = PACKET(LL+1)
1493:      NDATA2 = MIN(ND0,LEN(CHSTR))
1494:      LP = LL + NPHEAD
1495: C

```

```

1496:      call MVITOC( NDATA2, CHSTR, PACKET(LP) )
1497: C
1498:      if ( NDATA2.lt.ND0 ) then
1499:      IRET = 2
1500:      PACKET(LPERR) = PACKET(LPERR) + 1
1501:      write(IPR,*) 'XXX(UNPKCS):data <', MESSAGE,
1502: & ' > of type <CH> stored with length ', ND0,
1503: & ' in data packet ', ' container No. ', PACKET(1),
1504: & ' but input buffer is ', 'small to restore all data (',
1505: & LEN(CHSTR), ') '
1506:      end if
1507: C
1508:      PACKET(LPCUR) = PACKET(LL+2)
1509:      PACKET(4) = PACKET(4) + 1
1510:      IRET = 0
1511:      return
1512:      end
1513: C-----
1514: C
1515: C < unpacking by getting pointer to data >
1516: C
1517:      subroutine UNPKPT( PACKET,MESSAGE, TYPE, LPT, NDATA, IRET
1518: & )
1519: C=====
1520: C : unpack numeric/character data by returning pointer ( I4, R4, R8 )
1521: C returns pointer & data size and proceed to next packet.
1522: C (does not skip label packet )
1523: C
1524: C type : data type
1525: C lpt : pointer to data portion of packet. ( packet(ipt) )
1526: C for 'R8' type, lpt is array index to real*8 array
1527: C having the same starting address with 'packet'.
1528: C ndata : number of data packed in data packet.
1529: C iret : return code
1530: C 0 : successful
1531: C 1 : no more packet
1532: C 99 : invalid packet!
1533: C -N : data type mismatch.
1534: C N is type of data in packet
1535: C ( 1/2/3/4/5 = I4/R4/R8/CHAR/LB )
1536: C=====
1537: C
1538: C ... packet ....
1539: C
1540: C integer PACKET(*)
1541: C
1542: C character*(*) MESSAGE
1543: C character*(*) TYPE
1544: C
1545: C ... word length of container header ...
1546: C
1547:      parameter( NPHEAD = 4 )
1548: Cccc parameter( LPCUR = 5 ) ! Sep 1995
1549:      parameter( LPERR = 5 )
1550:      parameter( LPCUR = 6 )
1551:      parameter( NCHEAD = LPCUR )
1552: C
1553:      include 'INC/_IOUNIT'
1554:      include 'INC/_WORDL'
1555: C-----
1556:      NDATA = 0
1557:      LPT = 0
1558:      LL = PACKET(LPCUR)
1559: C
1560:      LH1 = PACKET(LL)

```

src/mvppburn/packet.f

```

1561:      if ( LH1.lt.0 ) then
1562:         IRET = 1
1563:         PACKET(LPERR) = PACKET(LPERR) + 1
1564:         write(IPR,*) 'XXX(UNPKPT):cannot find data <', MESSAGE,
1565: &         '> of type <',
1566: &         'TYPE, '> in data packet ', ' container No. ',
1567: &         PACKET(1), ', end of container.'
1568:         return
1569:      end if
1570: C
1571:      ITYP = LH1
1572: C
1573:      if ( ITYP.lt.1 .or. ITYP.gt.5 ) then
1574:         write(IPR,*) 'XXX(UNPKPT):unsupported type of data in packet ',
1575: &         ' packet container ', PACKET(1), ' packet # ',
1576: &         PACKET(4), ' <', MESSAGE, '>'
1577:         write(IPR,*) 'TYPE ', ITYP, ' location ', PACKET(LPCUR)
1578:         IRET = 99
1579:         PACKET(LPERR) = PACKET(LPERR) + 1
1580:         PACKET(LPCUR) = PACKET(LL+2)
1581:         PACKET(4) = PACKET(4) + 1
1582:         return
1583:      end if
1584: C
1585:      if ( ITYP.eq.1.and.TYPE.ne.'I4' .or. ITYP.eq.2
1586: & .and.TYPE.ne.'R4' .or. ITYP.eq.3.and.TYPE.ne.'R8' .or. ITYP.eq.
1587: & 4.and.TYPE.ne.'CH' .or. ITYP.eq.5.and.TYPE.ne.'LB' ) then
1588:         IRET = -ITYP
1589:         PACKET(LPERR) = PACKET(LPERR) + 1
1590:         write(IPR,*) 'XXX(UNPKPT):data <', MESSAGE, '> of type <', TYPE,
1591: &         '> required in data packet ', ' container No. ',
1592: &         PACKET(1), ' but stored type is ', ITYP,
1593: &         ' (1/2/3/4/5=I4/R4/R8/CH/LB )'
1594:         PACKET(LPCUR) = PACKET(LL+2)
1595:         PACKET(4) = PACKET(4) + 1
1596:         return
1597:      end if
1598: C
1599:      NDATA = PACKET(LL+1)
1600:      LPT = LL + NPHEAD
1601: C
1602:      if ( TYPE.eq.'R8'.and.MWORD.gt.1 ) then
1603:         if ( MOD(LPT,MWORD).ne.1 ) LPT = LPT + 1
1604:         LPT = LPT/MWORD + MWORD - 1
1605:      end if
1606: C
1607:      PACKET(LPCUR) = PACKET(LL+2)
1608:      PACKET(4) = PACKET(4) + 1
1609:      IRET = 0
1610:      return
1611: C
1612:      end
1613: C
1614: C-----
1615: C-----
1616: C
1617:      subroutine PCTDMP( IPR, PACKET,RPCT, DPCT )
1618: C=====
1619: C      dump data packet container
1620: C=====
1621:      integer PACKET(*)
1622: C
1623: C      ... rpct & dpct must be equivalent to packet.
1624: C
1625:      real RPCT(*)

```

```

1626:      real*8 DPCT(*)
1627: C
1628:      character*4 TYPE
1629:      character*512 CBUFF
1630: C
1631:      write(IPR,'(//lx,a/)') '***** DATA PACKET CONTAINER DUMP *****'
1632: C
1633:      call PCTRW( PACKET, 'START', IRET )
1634: C
1635:      IP = 0
1636: C
1637: 100 TYPE = ' '
1638:      call PCTCHK( PACKET, 'CHECK', TYPE, LPT, NDATA, IRET )
1639: C
1640:      if ( TYPE.eq.' ' ) go to 120
1641: C
1642:      IP = IP + 1
1643: C
1644:      write(IPR,'(/a,i4,3a)') '-- packet # ',IP,' --- type <',TYPE,>'
1645:      write(IPR,*) ' pointer ', LPT, ' number of data ', NDATA
1646: C
1647: C
1648:      if ( TYPE.eq.'LB' .or. TYPE.eq.'CH' ) then
1649: C
1650:         if ( NDATA.gt.LEN(CBUFF) ) then
1651:            write(IPR,*) ' (character length ', NDATA, ' is longer ',
1652: &            'than buffer size. output truncated.)'
1653:         end if
1654: C
1655:         if ( TYPE.eq.'CH' ) then
1656:            call UNPKCS( PACKET, 'CH', CBUFF, NDATA, IRET )
1657:         else
1658:            call UNPKPT( PACKET, 'LB', TYPE, LPT, NDATA, IRET )
1659:            NDATA = MIN(NDATA,LEN(CBUFF))
1660:            call MVITOC( NDATA, CBUFF, PACKET(LPT) )
1661:         end if
1662: C
1663:         do 110 I = 1, NDATA, 64
1664:            I2 = MIN(I+63,NDATA)
1665:
1666:            write(IPR,'(lx,i3,'''',i3,''' [ ''a, '' ]'')') I, I2,
1667: &            CBUFF(I:I2)
1668: 110      continue
1669: C
1670:         else if ( TYPE.eq.'I4' ) then
1671: C
1672:            call UNPKPT( PACKET, 'INT4', 'I4', LP, ND, IRET )
1673:            write(IPR,'(lx,5x,9i7)') (PACKET(LP+I),I=0,ND-1)
1674: C
1675:         else if ( TYPE.eq.'R4' ) then
1676: C
1677:            call UNPKPT( PACKET, 'REAL4', 'R4', LP, ND, IRET )
1678:            write(IPR,'(lx,lp,5e14.5)') (RPCT(LP+I),I=0,ND-1)
1679: C
1680:         else if ( TYPE.eq.'R8' ) then
1681: C
1682:            call UNPKPT( PACKET, 'REAL8', 'R8', LP, ND, IRET )
1683:            write(IPR,'(lx,lp,5d14.5)') (DPCT(LP+I),I=0,ND-1)
1684: C
1685:         end if
1686: C
1687:         go to 100
1688: C
1689: 120 write(IPR,'(//lx,a//)') '***** END OF DATA PACKET CONTAINER *****'
1690:      return

```

src/mvpburn/packet.f

1691: end



src/mvpsburn/pdsfil.f

```

1:      subroutine PDSFIL(MODE, DIRNAM, MEM1, FORM, IUNIT, IOUT, IERR)
2: C=====
3: C PDS UTILITY ROUTINE to open/close "virtual" member.
4: C
5: C A "virtual" PDS member is an arbitrary text or binary file
6: C created in a PDS directory using naming rule similar to ordinary
7: C PDS members.
8: C
9: C This type of member cannot be accessed by "READ" or "WRITE" mode
10: C of PDSIO routine. "SEARCH" and "DELETE" operations are possible
11: C if the file is not OPEN.
12: C
13: C To get the full path name of the file, users can use PDSNM routine.
14: C
15: C
16: C Mode:
17: C
18: C OPEN : open virtual PDS member on I/O unit IUNIT as read/write
19: C mode.
20: C WRITE : open virtual PDS member on I/O unit IUNIT as write
21: C mode.
22: C READ : open virtual PDS member on I/O unit IUNIT as readonly file.
23: C CLOSE : close PDS. Doing nothing in UNIX.
24: C
25: C DIRNAM, MEM1 : PDS directory and member name.
26: C
27: C FORM : 'TEXT' or 'BINARY' ('T' or 'B' is effective)
28: C
29: C IUNIT : I/O unit of the file. Must be given by caller.
30: C
31: C IOUT : message output I/O unit.
32: C
33: C IERR : return code.
34: C
35: C =0 : succesfully opened or closed
36: C =1 : I/O unit is already opened in OPEN mode
37: C or closing unoped I/O unit in CLOSE mode.
38: C =2 : failed to open/close the file
39: C =4 : invalid mode
40: C =8 : unsupported file form
41: C
42: C=====
43:      character(*) MODE
44:      character(*) FORM
45:      character(*) DIRNAM
46:      character*8 MEM1
47: C
48:      logical OPD
49:      character*256 FILNAM
50:      character*8 ACTION
51:      logical EX
52:      common /PDSPDS/ IPDSCK
53: C
54: C-----
55: C
56:      IERR = 0
57: C
58:      NLENG = 0
59:      call PDSNM( DIRNAM, MEM1, FILNAM, NLENG, IERR )
60: C
61:      IRC = 0
62:      if ( IERR.ne.0 ) then
63:          if ( IERR.eq.1 ) then
64:              IRC = 2
65:              write(IOUT,6888)

```

```

66:          write(IOUT,*) ' member name is invalid: <', MEM1,
67:          &      '>'
68:          else if ( IERR.eq.2 ) then
69:              IRC = 3
70:              write(IOUT,6888)
71:              write(IOUT,*) ' directory name is invalid: <',
72:              &      DIRNAM(:iclen2(DIRNAM)), '>'
73:          end if
74:      end if
75: C
76:      if ( IRC.gt.0 ) then
77:          write(IOUT,*)
78:          &      ' can not construct file name for member <',
79:          &      MEM1, '> of PDS <', DIRNAM(:ICLEN2(DIRNAM)), '>'
80:          stop 888
81:      end if
82: C
83: C *** OPEN
84: C
85:      if ( MODE.eq.'OPEN' .or. MODE.eq.'WRITE' .or. MODE.eq.'READ' )
86:          &      then
87: C
88:          if ( MODE.eq.'READ' ) then
89:              ACTION='READ'
90:              inquire(file=FILNAM(:ICLEN2(FILNAM)),EXIST=EX)
91:              if (.NOT.EX) then
92:                  write(IOUT,6888)
93:                  write(IOUT,*) ' file (',FILNAM(:ICLEN2(FILNAM)),
94:                  &      ' ) dose not exist'
95:                  IERR = 1
96:                  stop 999
97:              end if
98:          else
99:              ACTION='WRITE'
100:          end if
101: C
102:          inquire(IUNIT,opened =OPD)
103:          if ( OPD ) then
104:              write(IOUT,*) 'XXX(PDSFIL) ERROR :tried to open virtual',
105:              &      ' member <',MEM1,> on I/O unit ', IUNIT,
106:              &      ' but it is already opened.'
107:              IERR = 1
108:              return
109:          end if
110:
111:          if ( FORM(1:1).eq.'B' ) then
112:
113:              call OPENF( IUNIT,FILNAM(:ICLEN2(FILNAM)),'UNFORMATTED',
114:              &      'UNKNOWN', ACTION, IOS )
115:
116:          else if ( FORM(1:1).eq.'T' ) then
117:
118:              call OPENF( IUNIT,FILNAM(:ICLEN2(FILNAM)),'FORMATTED',
119:              &      'UNKNOWN', ACTION, IOS )
120:
121:          else
122:              write(IOUT,*) 'XXX(PDSFIL) ERROR :tried to open virtual',
123:              &      ' member <',
124:              &      MEM1,> in unsupported file form <', FORM, '>'
125:              IERR = 8
126:              return
127:          end if
128:
129:          if ( IOS.ne.0 ) then
130:              write(IOUT,*) 'XXX(PDSFIL) ERROR :failed to open virtual',

```

src/mvpburn/pdsfil.f

```
131:      &          ' member <',
132:      &          MEM1,'> open I/O error code ', IOS
133:      IERR      = 2
134:      return
135:  end if
136: C
137:      if ( IPDSCK.eq.1 ) then
138:      if ( MODE.eq.'WRITE' ) then
139:      write(IOUT,7000) MEM1, IUNIT, MODE
140:      end if
141:      else if ( IPDSCK.gt.1 ) then
142:      write(IOUT,7000) MEM1, IUNIT, MODE
143:      end if
144: C
145: C *** close
146: C
147:      else if ( MODE.eq.'CLOSE' ) then
148:      inquire(IUNIT,opened =OPD)
149:      if ( .not.OPD ) then
150:      write(IOUT,*) 'XXX(PDSFIL) ERROR :tried to close virtual',
151:      &          ' member <',
152:      &          MEM1,'> on I/O unit ', IUNIT,
153:      &          ' but it is not opened.'
154:      IERR      = 1
155:      return
156:      end if
157:
158:      close( IUNIT, iostat =IOS )
159:      if ( IOS.ne.0 ) then
160:      write(IOUT,*)
161:      &          'XXX(PDSFIL) ERROR :failed to close virtual member',
162:      &          ' <', MEM1,
163:      &          '>' close I/O error code ', IOS
164:      IERR      = 2
165:      return
166:      end if
167: C
168:      if ( IPDSCK.gt.1 ) then
169:      write(IOUT,7020) MEM1, IUNIT
170:      end if
171:      else
172:      write(IOUT,*) 'XXX(PDSFIL) ERROR :tried to open virtual',
173:      &          ' member <',
174:      &          MEM1,'> in unsupported mode <', MODE, '>'
175:      IERR      = 4
176:      return
177:      end if
178: C
179: 6888 format(// 'XXX(PDSFIL) ERROR-STOP : ' )
180: 7000 format(2X,'MEMBER "',A,'" IS OPENED AS VIRTUAL MEMBER',
181:      &          ' ON I/O UNIT ',I2,' IN <',a,'> MODE.')
182: 7020 format(2X,'MEMBER "',A,'" IS CLOSED AS VIRTUAL MEMBER',
183:      &          ' ON I/O UNIT ',I2)
184:      end
```

src/mvpburn/pdsio.f

```

1:      subroutine PDSIO( MODE, DIRNAM, MEM1, MEM2, ARRAY, IDATA, IOUT )
2: C=====
3: C      PDS UTILITY ROUTINE
4: C-----
5: C
6: C      Mode:
7: C
8: C      OPEN : open PDS. Doing nothing in UNIX assuming target directory
9: C              already exist.
10: C      CLOSE : close PDS. Doing nothing in UNIX.
11: C      READ : read data from member MEM1.
12: C      WRITE : write data on member MEM1.
13: C      SEARCH : check existence of member MEM1 and get number of data IDATA.
14: C              Return negative IDATA when the member does not exist.
15: C      EXIST : check only existence of the member.
16: C              IDATA is set to 1 when the member exists and set to negative
17: C              when it does not exist.
18: C      DELETE : delete member MEM1.
19: C
20: C=====
21:      character*(*) MODE
22:      character*(*) DIRNAM
23:      character*8 MEM1, MEM2
24: C
25:      real ARRAY(*)
26: C
27:      common /MACHTY/ IFACOM
28:      common /PDSPDS/ IPDSCK
29: C
30:      character*256 FILNAM
31:      character*128 DDNAME
32: C
33:      logical EXST
34: C
35: C-----
36: C
37: C *** START OF PROCESS
38: C
39:      IRC      = 0
40:      LENG     = 0
41: C
42: C *** copy dirnam to ddname (PDS* routines may write something on it)
43: C
44:      LDIR     = ICLEN2(DIRNAM)
45:      if ( IFACOM.ne.0 ) then
46:          LDIR     = 8
47:      end if
48:      DDNAME   = DIRNAM(:LDIR)
49: C
50: C *** OPEN PDS-FILE
51: C
52:      if ( MODE(1:4).eq.'OPEN' ) then
53:          if ( IPDSCK .gt. 1 ) THEN
54:              write(IOUT,7000) DDNAME(:LDIR)
55:          end if
56:          return
57:      end if
58: C
59: C *** CLOSE PDS-FILE
60: C
61:      if ( MODE(1:4).eq.'CLOS' ) then
62:          if ( IPDSCK .gt. 1 ) THEN
63:              write(IOUT,7020) DDNAME(:LDIR)
64:          end if
65:          return

```

```

66:      end if
67: C
68: C *** SET FILE NAME USING DDNAME & MEMBER NAME
69: C
70:      IERR     = 0
71:      NLENG     = 0
72: C
73:      call PDSNM( DDNAME(:LDIR), MEM1, FILNAM, NLENG, IERR )
74:      if ( IERR.ne.0 ) then
75:          if ( IERR.eq.1 ) then
76:              IRC      = 2
77:              write(IOUT,*) 'XXX(PDSIO) MEMBER NAME IS INVALID: <', MEM1,
78:                  &          '>'
79:          else if ( IERR.eq.2 ) then
80:              IRC      = 3
81:              write(IOUT,*) 'XXX(PDSIO) DIRECTORY NAME IS INVALID: <',
82:                  &          DDNAME(:LDIR), '>'
83:          end if
84:      end if
85: C
86:      if ( IRC.gt.0 ) go to 100
87: C
88: C *** READ PDS-FILE
89: C
90:      if ( MODE(1:4).eq.'READ' ) then
91:          IERR     = 0
92:          call PDSSR( FILNAM, IERR )
93:          if ( IERR.eq.1 ) then
94:              IRC      = 8
95:              go to 100
96:          end if
97:          call PDSLN( FILNAM, IDATA, IRC, IOUT )
98:          if ( IRC.gt.0 ) go to 100
99:          call PDSRD( FILNAM, ARRAY, IDATA, IRC )
100:         if ( IRC.gt.0 ) go to 100
101:         if ( IPDSCK .gt. 1 ) THEN
102:             write(IOUT,7040) MEM1, IDATA
103:         end if
104:     end if
105: C
106: C *** WRITE PDS-FILE
107: C
108:     if ( MODE(1:4).eq.'WRIT' ) then
109:         IERR     = 0
110:         call PDSSR( FILNAM, IERR )
111:         if ( IERR.eq.0 ) then
112:             call PDSRM( FILNAM, IRC, IOUT )
113:             if ( IRC.gt.0 ) go to 100
114:         end if
115:         IRC      = 0
116:         call PDSWT( FILNAM, ARRAY, IDATA, IRC )
117:         if ( IRC.gt.0 ) go to 100
118:         if ( IPDSCK .gt. 0 ) THEN
119:             write(IOUT,7060) MEM1, IDATA
120:         end if
121:     end if
122: C
123: C *** SEARCH PDS-FILE
124: C
125:     if ( MODE(1:4).eq.'SEAR' ) then
126:         IERR     = 0
127:         call PDSSR( FILNAM, IERR )
128:         if ( IERR.eq.1 ) then
129:             IDATA   = -1
130:             return

```

src/mvppburn/pdsio.f

```

131:         end if
132:         call PDSLN( FILNAM, IDATA, IRC, IOUT )
133:         if ( IRC.gt.0 ) go to 100
134:         end if
135: C
136: C *** EXISTENCE CHECK ONLY PDS-FILE
137: C
138:         if ( MODE(1:4).eq.'EXIS' ) then
139:             IERR = 0
140:             inquire(file =FILNAM(:ICLEN2(FILNAM)),exist =EXST)
141:             if ( EXST ) then
142:                 IDATA = 1
143:             else
144:                 IDATA = -1
145:             end if
146:             return
147:         end if
148: C
149: C *** RENAME PDS-FILE
150: C *** IN UNIX SYSTEM , THIS MODE IS NOT SUPPORTED !!!
151: C
152:         if ( MODE(1:4).eq.'RENA' ) then
153: CDEL          CALL PDSREN( DDNAME , MEM1 , MEM2 , IRC)
154:             IRC = 999
155:             if ( IRC.gt.0 ) go to 100
156:             if ( IPDSCK .gt. 0) THEN
157:                 write(IOUT,7080) MEM1, MEM2
158:             end if
159:         end if
160: C
161: C *** DELETE PDS-FILE
162: C
163:         if ( MODE(1:4).eq.'DELE' ) then
164:             IERR = 0
165:             call PDSSR( FILNAM, IERR )
166:             if ( IERR.eq.1 ) then
167:                 write(IOUT,7180) MEM1
168:                 return
169:             end if
170:             call PDSSRM( FILNAM, IRC, IOUT )
171:             if ( IRC.gt.0 ) go to 100
172:             if ( IPDSCK .gt. 0) THEN
173:                 write(IOUT,7100) MEM1
174:             end if
175:         end if
176: C
177:         return
178: C
179: 100 continue
180:         write(IOUT,6888)
181:         write(IOUT,7140) IRC, MODE, MEM1
182:         stop 999
183: C
184: 6888 format(/// XXX(PDSIO) ERROR-STOP : ' )
185: 7000 format(/1X, ' ** DIRECTORY IS OPENED (',A,') ** ' )
186: 7020 format(/1X, ' ** DIRECTORY IS CLOSED (',A,') ** ' )
187: 7040 format(2X,'MEMBER "',A,'" OF LENGTH ',I10,' IS READ FROM PDS')
188: 7060 format(2X,'MEMBER "',A,'" OF LENGTH ',I10,' IS STORED IN PDS')
189: 7080 format(2X,'MEMBER "',A,'" IS RENAMED TO ',A8,' IN PDS')
190: 7100 format(2X,'MEMBER "',A,'" IS DELETED FROM PDS')
191: 7140 format(1X,'error at pds I/O operation'/1X,
192:             &         'return code is ',I4,' in mode ',A,' !!!',5X,
193:             &         'member name is "',A,'"')
194: 7180 format(2X,'MEMBER "',A,'" IS NOT FOUND IN PDS ',
195:             &         'IN DELETE MODE (NORMAL RETURN !!!)')

```

```

196: C
197:         end

```

src/mvpburn/pdsln.f

```
1: C*****
2: C UTILITY PROGRAM TO GET LENGTH OF PDS DATA      *
3: C*****
4:
5:       subroutine PDSLN( FILNAM,LENG,  IRC,   IOUT )
6: C
7: C-----
8: C     IRC : ERROR CODE
9: C         = 0 NORMAL END
10: C        = 5 PDS OPEN ERROR
11: C        = 6 PDS CLOSE ERROR
12: C        = 7 PDS READ ERROR
13: C-----
14: C
15:       character FILNAM*(*)
16: C
17:       IOPDS   = 60
18:       IRC     = 0
19: C
20:       open( unit =IOPDS, file =FILNAM(:ICLEN2(FILNAM)), err =100,
21: &         status ='UNKNOWN', access ='SEQUENTIAL', form =
22: &         'UNFORMATTED', iostat =IOS )
23:       read(unit =IOPDS,err =120,iostat =IOS) LENG
24:       close( unit =IOPDS, err =110, status ='KEEP', iostat =IOS )
25:       go to 130
26: C
27: 100 IRC     = 5
28:       write(IOUT,*) ' XXX PDS OPEN ERROR IN ', FILNAM
29:       return
30: 110 IRC     = 6
31:       write(IOUT,*) ' XXX PDS CLOSE ERROR IN ', FILNAM
32:       return
33: 120 IRC     = 7
34:       write(IOUT,*) ' XXX PDS READ ERROR IN ', FILNAM
35: C
36: 130 return
37:       end
```


src/mvpburn/pdsnm.f

```
1:      subroutine PDSNM(DIRNAM, MEMBER, FILNAM, NLENG, IERR)
2: C=====
3: C UTILITY PROGRAM TO GIVE FULL NAME OF PDS MEMBER
4: C FILNAM = DIRNAM/MEMBER
5: C=====
6:      character DIRNAM*(*)
7:      character MEMBER*8
8:      character FILNAM*(*)
9: C
10: C-----INPUT-----
11: C      DIRNAM      : DIRECTORY NAME (A72) OF PDS : /XXX/XXX/MACRO
12: C      MEMBER      : PDS MEMBER NAME (A8) : CASEA010
13: C-----OUTPUT-----
14: C      FILNAM      : FULL FILE NAME (A80) OF PDS MEMBER
15: C                  /XXX/XXX/MACRO/CASEA010
16: C      NLENG       : LENGTH OF FILE NAME
17: C      IERR        : ERROR CODE =0 : NORMAL END
18: C                  =1 : MEMBER NAME IS EMPTY OR INVALID
19: C                  =2 : DIRECTORY NAME IS EMPTY OR INVALID
20: C-----
21: C
22:      IERR = 0
23:      if ( MEMBER(1:1).eq.' ' ) then
24:          IERR = 1
25:          return
26:      end if
27:      if ( DIRNAM(1:1).eq.' ' ) then
28:          IERR = 2
29:          return
30:      end if
31: C
32:      FILNAM = ' '
33:      NLENG = 0
34:      do I = 1, ICLEN2(DIRNAM)
35:          if ( DIRNAM(I:I).ne.' ' ) then
36:              NLENG = NLENG + 1
37:          else
38:              go to 110
39:          end if
40:      end do
41: 110 if ( NLENG.eq.0 ) then
42:      IERR = 2
43:      return
44:  end if
45: C
46:      MLENG = 0
47:      do I = 1, 8
48:          if ( MEMBER(I:I).ne.' ' ) then
49:              MLENG = MLENG + 1
50:          else
51:              go to 130
52:          end if
53:      end do
54: C
55: 130 if ( NLENG.eq.0 ) then
56:      IERR = 1
57:      return
58:  end if
59:      NLENGW = NLENG
60:      NLENG = NLENG + 1 + MLENG
61:
62: C/#IF FILENAME( DOS )
63: C
64: C CHAR(92) is '\ ' in ASC-II character code.
65: C '\ ' may be taken as "escape code" characters in charcter string
```

```
66: C on some Fortran compilers. so such an expression is used.
67: C
68: *      FILNAM(1:NLENG) = DIRNAM(1:NLENGW) //CHAR(92)//MEMBER
69: C
70: C/#ELSEIF FILENAME( MACOS )
71: C
72: *      FILNAM(1:NLENG) = DIRNAM(1:NLENGW) //' '//MEMBER
73: C
74: C/#ELSE
75: C
76:      FILNAM(1:NLENG) = DIRNAM(1:NLENGW) //' '//MEMBER
77: C
78: C/#ENDIF
79:
80:      return
81:      end
```

src/mvpburn/pdsrd.f

```
1:      subroutine PDSRD( FILNAM,WORK, LENG, IERR )
2: C*****
3: C UTILITY PROGRAM TO READ CONTENTS OF PDS MEMBER
4: C*****
5: C
6:      real WORK(1)
7:      character FILNAM*(*)
8: C
9: C-----INPUT-----
10: C   FILNAM      : FULL PDS MEMBER NAME : /XXX/XXX/MACRO/CASEA010
11: C-----OUTPUT-----
12: C   WORK        : CONTENTS OF MEMBER
13: C   LENG        : LENGTH OF DATA IN MEMBER (EXCEPT LENG ITSELF)
14: C   IERR        : ERROR CODE =0 : NORMAL END
15: C               =1 : OPEN ERROR
16: C               =2 : CLOSE ERROR
17: C               =3 : READ ERROR
18: C-----
19: C
20:      IERR      = 0
21:      IOPDS     = 60
22: C
23:      open( unit =IOPDS, file =FILNAM(:ICLEN2(FILNAM)), err =100,
24: &         status ='UNKNOWN', access ='SEQUENTIAL', form =
25: &         'UNFORMATTED', iostat =IOS )
26:      read(unit =IOPDS,err =120,iostat =IOS) LENG, (WORK(I),I=1,LENG)
27:      close( unit =IOPDS, err =110, status ='KEEP', iostat =IOS )
28:      return
29: 100 IERR      = 1
30: C   WRITE(6,*) ' OPEN ERROR , IOSTAT=',IOS
31:      return
32: 110 IERR      = 2
33: C   WRITE(6,*) ' CLOSE ERROR , IOSTAT=',IOS
34:      return
35: 120 IERR      = 3
36: C   WRITE(6,*) ' READ ERROR , IOSTAT=',IOS
37:      return
38:      end
```

src/mvpsburn/pdsrm.f

```
1:      subroutine PDSRM( FILNAM,IRC,  IOUT )
2: C*****
3: C UTILITY PROGRAM TO DELETE A PDS MEMBER
4: C*****
5: C
6:      character FILNAM*(*)
7: C
8: C-----INPUT-----
9: C      FILNAM      : FULL PDS MEMBER NAME : /XXX/XXX/MACRO/CASEA010
10: C-----OUTPUT-----
11: C      IRC          : ERROR CODE
12: C                  = 0 NORMAL END
13: C                  = 1 MEMBER NOT FOUND IN READ MODE (NOT EFFECTIVE)
14: C                  = 2 MEMBER NAME IS EMPTY OR INVALID (NOT EFFECTIVE)
15: C                  = 3 DIRECTORY NAME IS EMPTY OR INVALID (NOT EFFECTIVE)
16: C                  = 4 MEMBER ALREADY EXIST IN WRITE MODE (NOT EFFECTIVE)
17: C                  = 5 PDS OPEN ERROR
18: C                  = 6 PDS CLOSE ERROR
19: C                  = 7 PDS READ ERROR (NOT EFFECTIVE)
20: C                  = 8 PDS WRITE ERROR (NOT EFFECTIVE)
21: C-----
22: C
23:      IRC          = 0
24:      IOPDS        = 60
25: C
26:      open( unit =IOPDS, file =FILNAM(:ICLEN2(FILNAM)), err =100,
27: &         iostat =IOS )
28:      close( unit =IOPDS, err =110, status = 'DELETE', iostat =IOS )
29:      return
30: 100 IRC          = 5
31:      write(IOUT,*) ' XX PDS OPEN ERROR(PDSRM) IN ', FILNAM
32:      return
33: 110 IRC          = 6
34:      write(IOUT,*) ' XX PDS CLOSE ERROR(PDSRM) IN ', FILNAM
35:      return
36:      end
```

src/mvpburn/pdssr.f

```
1:      subroutine PDSSR( FILNAM,IEXST )
2: C*****
3: C UTILITY PROGRAM TO SEARCH PDS MEMBER          *
4: C FILNAM = DIRNAM/MEMBER                        *
5: C*****
6: C
7:      character FILNAM*(*)
8: C
9: C-----INPUT-----
10: C      FILNAM      : FULL FILE NAME (A80) OF PDS MEMBER
11: C                   /XXX/XXX/MACRO/CASEA010
12: C-----OUTPUT-----
13: C      IEXST       : = 0  EXIST
14: C                   = 1  NOT EXIST
15: C-----
16: C
17:      logical EX
18:
19:      inquire(file =FILNAM(:ICLEN2(FILNAM)),exist =EX)
20:      if ( .not EX ) then
21:          IEXST = 1
22:      else
23:          IEXST = 0
24:      end if
25:      return
26:      end
```

src/mvpburn/pdswt.f

```
1:      subroutine PDSWT( FILNAM,WORK, LENG, IERR )
2: C
3: C*****
4: C UTILITY PROGRAM TO WRITE CONTENTS OF PDS MEMBER      *
5: C*****
6: C
7:      real WORK(LENG)
8:      character FILNAM*(*)
9: C
10: C-----INPUT-----
11: C      FILNAM      : FULL PDS MEMBER NAME : /XXX/XXX/MACRO/CASEA010
12: C      WORK        : CONTENTS OF MEMBER
13: C      LENG        : LENGTH OF DATA IN MEMBER (EXCEPT LENG ITSELF)
14: C-----OUTPUT-----
15: C      IERR        : ERROR CODE =0 : NORMAL END
16: C                  =1 : OPEN ERROR
17: C                  =2 : CLOSE ERROR
18: C                  =3 : WRITE ERROR
19: C-----
20: C
21:      IERR      = 0
22:      IOPDS     = 60
23: C
24:      open( unit =IOPDS, file =FILNAM(:ICLEN2(FILNAM)), err =100,
25: &          status = 'UNKNOWN', access = 'SEQUENTIAL', form =
26: &          'UNFORMATTED', iostat =IOS )
27:      write(unit =IOPDS,err =120,iostat =IOS) LENG, (WORK(I),I=1,LENG)
28:      close( unit =IOPDS, err =110, status = 'KEEP', iostat =IOS )
29:      return
30: 100 IERR      = 1
31: C      WRITE(6,*) ' OPEN ERROR , IOSTAT=',IOS
32:      return
33: 110 IERR      = 2
34: C      WRITE(6,*) ' CLOSE ERROR , IOSTAT=',IOS
35:      return
36: 120 IERR      = 3
37: C      WRITE(6,*) ' WRITE ERROR , IOSTAT=',IOS
38:      return
39:      end
```

src/mvpburn/prstop.f

```
1:      subroutine PRSTOP( NERR, MESSG )
2: C=====
3: C Purpose: To print summary input-data error when terminating
4: C   before completing input data processing etc.
5: C
6: C   << This routine is necessary to use "FREAD" routines,
7: C       from MVP-BURN v2.0 and after>>
8: C
9: C       nerr : increment count fatal error 'nerr' times.
10: C
11: C CALLED IN: anywhere
12: C CALLS PRNERR
13: C=====
14:      character*(*) MESSG
15: C
16: C ... I/O unit of printout specific to MVP-BURN ....
17: C
18:      common /IOPRNT/  NOUT1,  NOUT2
19: C
20: C-----
21: C
22:      IPR = NOUT1
23: C
24:      if ( NERR.gt.0 ) then
25:        do 100 I = 1, NERR
26:          call CNTERR( 'FATAL' )
27: 100    continue
28:      end if
29: C
30:      call PRNERR( IPR, ' ' )
31: C
32:      write(IPR,7000)
33: 7000 format('1'/5X,5('XXXXXXXXXXXX')//6X,
34:      &      ' Your input data has one or more fatal errors which make '
35:      &      '//6X,' it impossible to process all of your input data.'//
36:      &      5X,5('XXXXXXXXXXXX')//)
37: C
38:      if ( MESSG.ne.' ' ) then
39:        write(IPR,7020) MESSG
40: 7020 format(/5X,' MESSAGE FROM SUBROUTINE : '//5X,A/)
41:      end if
42:
43: C/#IF PARA( PVM )
44: *      call pvmfexit(info)
45: C/#ELSEIF PARA( MPI )
46: *      call mpi_finalize(info)
47: C/#ENDIF
48:
49: C/#IF UNIX
50: *      call flushstd()
51: C/#ENDIF
52:      return
53:      end
```

src/mvpburn/redmic.f

```
1:      subroutine REDMIC( RATMIC,NTREG, NUC,      NEMIC, KIND,  ANS,  ERR,
2:      &                  MPOS,  INUC,  IR )
3: C
4:      real*8 RATMIC(NTREG,NUC,NEMIC,KIND)
5:      real*8 ANS, ERR
6: C
7: C      ... Error is converted to absolute value from percent.
8: C
9:      ANS      = RATMIC(MPOS,INUC,IR,1)
10:     ERR      = RATMIC(MPOS,INUC,IR,2)
11:     ERR      = ERR*ANS*1.0000D-2
12: C
13:     return
14:     end
```

src/mvpburn/redmvp.f

```

1:      subroutine REDMVP( A,      MEMORY,NFILE, NMAT,  NTNUC,  VOLM,
2:      &      MATMVP,RATMIC,DENTAB,IDENT, AKEFF, EKEFF,
3:      &      NHIST0,KBATCH,MTBURN,TITL2, NOUT1, NOUT2,
4:      Ckuni&      IBC )
5:      C2005&      IBC      ,KPBURN )
6:      &      IBC      ,KPBURN,ISWRAT,IDMONI,RATNOW,ERRNOW )
7:      C=<MVPBURN>=====
8:      C Purpose : Read microscopic reaction rate (RATMIC) and eigenvalue
9:      C      (AKEF) from output file of MVP
10:     C
11:     C Microscopic reaction rate is normalized to per volume and per atom
12:     C in this routine so number density table DENTAB should match that used
13:     C in MVP calculation. (take care in PC method!!)
14:     C=====
15:     real*8 A(MEMORY)
16:     C
17:     real VOLM(NMAT), RATMIC(2,4,NTNUC,NMAT)
18:     real DENTAB(NTNUC,NMAT)
19:     C2003
20:     character*12 IDENT(MINUC,NMAT)
21:     integer MATMVP(NMAT), MTBURN(3,NMAT)
22:     character*72 TITL2
23:     C
24:     parameter (MXNUCS=1000)
25:     C
26:     real*8 WSUM, AKEFF, EKEFF, FACT, ANS, ERR
27:     character*72 TITLE(2), LINE*64
28:     C2003
29:     character*12 NUCID(MXNUCS), NUCNM
30:     integer LMIC(20), LMAC(20)
31:     character*32 TAG
32:     C2003 .... added an array argument IBC.
33:     integer IBC(20)
34:     C2003
35:     character*4 DUMYC4
36:     Ckuni 2003/12/22
37:     integer KPBURN(NTNUC,NMAT)
38:     C2005
39:     real*8 RRATE, ERATE1, ERATE2
40:     C/#IF INTEGER8
41:     C integer*8 NTHIST ! This delaration will be active in the future.
42:     C/#ELSE
43:     integer NTHIST
44:     C/#ENDIF
45:     C
46:     C-----
47:     C
48:     write(NOUT1,*)
49:     &      ' ---- GETTING ENERGY INTEGRATED MICRO. REACTION RATE',
50:     &      ' FROM MVP-BINARY FILE ---- '
51:     C
52:     call RWIND( NFILE )
53:     read(NFILE,err =160,end =160) LINE
54:     C
55:     write(NOUT1,'(1X,A64)') LINE
56:     C
57:     C.... OUTPUT FILE TYPE (too old MVP is not available)
58:     IFTP = 0
59:     if ( INDEX(LINE,'TYPE 1 ').ne.0 ) then
60:       IFTP = 1
61:     else if ( INDEX(LINE,'TYPE 2.0').ne.0 ) then
62:       IFTP = 1
63:     else if ( INDEX(LINE,'TYPE 3.0 ').ne.0 ) then
64:       IFTP = 1
65:     else if ( INDEX(LINE,'TYPE 3.01').ne.0 ) then

```

```

66:       IFTP = 1
67:     else if ( INDEX(LINE,'TYPE 3.02').ne.0 ) then
68:       IFTP = 1
69:     else if ( INDEX(LINE,'TYPE 3.03').ne.0 ) then
70:       IFTP = 1
71:     else if ( INDEX(LINE,'TYPE 3.04').ne.0 ) then
72:       IFTP = 1
73:     else
74:       IFTP = 0
75:     end if
76:     C
77:     if ( IFTP.ne.0 ) then
78:       write(NOUT1,6888)
79:       write(NOUT1,*) ' your using MVP is too old',
80:       &      ' MVP-BURN can not treat this MVP version'
81:       write(NOUT1,*) ' replace it by new MVP with binary file type',
82:       &      ' is 3.05 or more'
83:       stop 999
84:     end if
85:     C
86:     read(NFILE) LINE(1:27)
87:     write(NOUT1,'(1X,A)') LINE(1:27)
88:     read(NFILE) TITLE
89:     C
90:     write(NOUT1,7000) TITLE
91:     7000 format(/1X,'TITLE :',A72/1X,'      :',A72/)
92:     C
93:     C ... check the 2'nd line of the title ...
94:     C
95:     CKSK Set IADMIN = 1 to execute dummy MVP *****
96:     C      IADMIN = 1
97:     C      IADMIN = 0
98:     C2003 ... DEBUG mode. skip check of title2.
99:     if ( IBC(20).eq.1 ) IADMIN = 1
100:    if ( IADMIN.eq.1 ) goto 123
101:    CKSK*****
102:    if ( TITLE(2).ne.TITL2 ) then
103:      write(NOUT1,6888)
104:      write(NOUT1,7020) TITLE(2), TITL2
105:      stop 999
106:    end if
107:    123 continue
108:    CKSK*****
109:    C
110:    C Version 3.05 or later
111:    read(NFILE) TAG, WSUM, NTHIST, NBATCH, NGROUP, NREG, NRESP, NSKIP,
112:    &      NSTAL, NGP1, NGP2, NTREG, NPDET, NTIME, NSTALY,
113:    &      JNEUT, JPHOT, JRESP,
114:    &      JEIGN, JADJM, JTIME
115:    NREG = NTREG
116:    C
117:    NPKIND = 1
118:    if ( JNEUT*JPHOT.ne.0 ) NPKIND = 2
119:    C
120:    C-----
121:    C-----
122:    C NFILE: I/O UNIT
123:    C-----
124:    C
125:    C
126:    KBATCH = NBATCH - NSKIP
127:    NHIST0 = WSUM
128:    C
129:    C-----
130:    C -----0-----*---1-----*---2-----*---3--

```


src/mvpburn/redmvp.f

```

131:      TAG      = 'MICRO & MACRO TALLY DATA'
132: C -----
133:      call SRTAG( NFILE, TAG, IRC )
134: C
135:      if ( IRC.eq.0 ) then
136:        write(NOUT1,6888)
137:        write(NOUT1,*) ' no micro & macro tally data in MVP binary file'
138:        stop 999
139:      end if
140: C
141: C
142:      read(NFILE) TAG, NUC, NEMIC, NEMAC, JRTTR, JRTCL
143: C
144:      if ( NUC.gt.MXNUCS ) then
145:        write(NOUT1,6888)
146:        write(NOUT1,*)
147:        & ' too many number of nuclides treated in MVP'
148:        write(NOUT1,*) ' requested number of nuclides = ',NUC
149:        write(NOUT1,*) ' limit for number of nuclides = ',MXNUCS
150:        write(NOUT1,*)
151:        & ' change parameter value(MXNUCS) in this subroutine'
152:        stop 999
153:      end if
154: C
155: C2003 read(NFILE) TAG, (LMIC(L),L=1,16), (LMAC(L),L=1,16),
156: C2003& (NUCID(N),N=1,NUC)
157:      read(NFILE) TAG, (LMIC(L),L=1,16), (LMAC(L),L=1,16),
158:      & (NUCID(N)(1:8),N=1,NUC)
159: C2003 ... read 'NUCLIDE ID' record
160: C -----0-----*-----1-----*-----2-----*-----3---
161:      TAG      = 'NUCLIDE ID'
162: C      (16 characters in 3.05)
163: C -----
164:      call SRTAG( NFILE, TAG, IRC )
165:      read(NFILE) TAG, LC,(NUCID(N),DUMYC4,N=1,NUC)
166: C2003 ... end
167: C
168: C ...MICROSCOPIC : READ NUCLIDE # OR ID .....
169: C
170: C -----0-----*-----1-----*-----2-----*-----3---
171:      TAG      = 'MICROSCOPIC REACTION RATE'
172: C -----
173: C
174:      call SRTAG( NFILE, TAG, IRC )
175: C
176:      if ( IRC.eq.0 ) then
177:        write(NOUT1,6888)
178:        write(NOUT1,*) ' no micro reaction rate data in MVP binary file'
179:        stop 999
180:      end if
181: C
182: C
183:      read(NFILE) TAG
184:      LENG      = NTREG*NUC*NEMIC*2
185: C
186:      if ( LENG.gt.MEMORY ) then
187:        write(NOUT1,6888)
188:        write(NOUT1,*) ' memory over'
189:        write(NOUT1,*) ' requested memory size = ', LENG, ' words.'
190:        write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
191:        write(NOUT1,*) ' change parameter value in include file'
192:        stop 999
193:      end if
194: C
195:      read(NFILE) TAG, (A(I),I=1,LENG)

```

```

196: C
197: C *** CHECK ONE-GROUP REACTION RATE ( FISSION , CAPTURE , (N,2N) )
198: C
199:      do 100 MT = 1, 3
200:        IREAC = 3
201:        if ( MT.eq.2 ) IREAC = 5
202:        if ( MT.eq.3 ) IREAC = 7
203:        IR      = LMIC(IREAC)
204:        if ( IR.eq.0 ) then
205:          write(NOUT1,6888)
206:          write(NOUT1,*) ' reaction ', IREAC,
207:          & ' does not exist in MVP output binary file.'
208:          write(NOUT1,*) ' <reactions in file> ', (LMIC(L),L=1,8)
209:          stop 999
210:        end if
211:      100 continue
212: C
213: C *** STORE ONE-GROUP REACTION RATE ( FISSION , CAPTURE , (N,2N) )
214: C
215:      do 140 M = 1, NMAT
216:        MPOS = MATMVP(M)
217:        do 130 K = MTBURN(1,M), MTBURN(2,M)
218:          Ckuni 2003/12/22
219:          if( KPBURN(K,M).gt.0 ) then
220:            Cend
221:            NUCNM = IDENT(K,M)
222:            do 110 N = 1, NUC
223:              INUC = N
224:              C2003 if ( NUCID(N).eq.NUCNM ) go to 120
225:              C2003 ... temporary ... NUCID has only 8 character information in
226:              C2003 binary output file.
227:              CCCCCCCCCC if ( NUCID(N)(1:12).eq.NUCNM(1:8) ) go to 120
228:              if ( NUCID(N).eq.NUCNM ) go to 120
229:            110 continue
230:          C
231:          write(NOUT1,6888)
232:          write(NOUT1,*) ' nuclide ',NUCNM,
233:          & ' does not exist',
234:          & ' in MVP output binary file.'
235:          write(NOUT1,*) ' <nuclides in file> ', (NUCID(N),N=1,NUC)
236:          stop 999
237:          C
238:          120 continue
239:          FACT = VOLM(M)*DENTAB(K,M)
240:          if ( FACT.gt.0.0 ) FACT = 1.000/FACT
241:          C ***** GET FISSION RATE
242:          IR      = LMIC(3)
243:          call REDMIC( A, NTREG, NUC, NEMIC, 2, ANS, ERR, MPOS, INUC,
244:          & IR )
245:          RATMIC(1,1,K,M) = ANS*FACT
246:          RATMIC(2,1,K,M) = ERR*FACT
247:          C ***** GET CAPTURE RATE
248:          IR      = LMIC(5)
249:          call REDMIC( A, NTREG, NUC, NEMIC, 2, ANS, ERR, MPOS, INUC,
250:          & IR )
251:          RATMIC(1,2,K,M) = ANS*FACT
252:          RATMIC(2,2,K,M) = ERR*FACT
253:          C ***** GET (N,2N) RATE
254:          IR      = LMIC(7)
255:          call REDMIC( A, NTREG, NUC, NEMIC, 2, ANS, ERR, MPOS, INUC,
256:          & IR )
257:          RATMIC(1,3,K,M) = ANS*FACT
258:          RATMIC(2,3,K,M) = ERR*FACT
259:          C ***** SET ABSORPTION RATE
260:          RATMIC(1,4,K,M) = RATMIC(1,1,K,M) + RATMIC(1,2,K,M)

```

src/mvpburn/redmvp.f

```

261:      RATMIC(2,4,K,M) = RATMIC(2,1,K,M) + RATMIC(2,2,K,M)
262: Ckuni 2003/12/22
263:      endif
264: Cend
265:      130 continue
266:      140 continue
267: C-----
268: C      EIGENVALUE
269: C-----
270: C
271: C2005 ... added
272:      if ( JEIGN.eq.0 ) then
273:          AKEFF = 0.0
274:          EKEFF = 0.0
275:          go to 155
276:      endif
277: C2005 ... end
278: C      0----*----1----*----2----*----3--
279:      TAG = 'K-EFF: MAXIMUM LIKELYHOOD '
280: C      0----*----1----*----2----*----3--
281: C
282: C
283:      call SRTAG( NFILE, TAG, IRC )
284: C
285: C2005 removed
286: CKSK-- fixed source problem is allowed (may be not general)
287: Cdel if ( IRC.eq.0 ) then
288: Cdel      write(NOUT1,6666)
289: Cdel      write(NOUT1,*) ' no eigenvalue data in MVP output binary file'
290: Cdel      write(NOUT1,*) ' eigen value and its error are set 1.0 '
291: Cdel &      ' and calculation is continued'
292: Cdel      AKEFF = 1.0
293: Cdel      EKEFF = 1.0
294: Cdel      goto 155
295: Cdel end if
296: Cend
297: C
298: C      REC. POS= 1 : TRACK LENGTH (PRODUCTION + BALANCE)
299: C                2 : COLLISION (PRODUCTION + BALANCE)
300: C                3 : ANALOG (PRODUCTION + BALANCE)
301: C                4 : PRODUCTION (TRK+COL+ANALOG)
302: C                5 : BALANCE (TRK+COL+ANALOG)
303: C                6 : ALL ESTIMATORS
304: C
305:      AKEFF = 0.0
306:      EKEFF = 0.0
307: C ***** GET ALL ESTIMATORS K-EFF
308:      do 150 I = 1, 5
309:          read(NFILE)
310:      150 continue
311:      read(NFILE) TAG, (A(I),I=1,NBATCH-1), (A(NBATCH+I),I=1,NBATCH-1)
312:      AKEFF = A(NSKIP+1)
313: C
314: C      .... standard deviation (%)
315:      EKEFF = A(NSKIP+1+NBATCH)
316: C
317: C *** PRINT OUT
318: C
319: C2005 ... modified
320: C      write(NOUT1,7040) NUC, JRTTR, JRTCL, AKEFF, EKEFF
321:      155 continue
322:      write(NOUT1,7040) NUC, JRTTR, JRTCL, JEIGN, AKEFF, EKEFF
323: C2005 ... end
324: C
325:      7040 format(/1X,5X,

```

```

326:      &      'NUC : NO OF NUCLIDE IN REACTION RATE RESULT ... ',I3/
327:      &      1X,5X,
328:      &      'JRTTR : 1/0=TRACK LENGTH/COLLISION ESTIMATOR .... ',I3/
329:      &      1X,5X,
330:      &      'JRTCL : 1/0=COLLISION/TRACK LENGTH ESTIMATOR .... ',I3/
331: C2005 ... added
332:      &      1X,5X,
333:      &      'JEIGN : 1/0=EIGEN-VALUE/FIXED-SOURCE ..... ',I3/
334: C2005 ... end
335:      &      1X,5X,
336:      &      'KEFF : K-EFFECTIVE BY ALL ESTIMATORS ..... ',F10.6
337:      &      /1X,5X,
338:      &      'EKEFF : 1 SIGMA RELATIVE ERROR ..... ',
339:      &      F10.6, ' %'//)
340: C2005
341:      if (ISWRAT.eq.0) return
342: C
343: C      0----*----1----*----2----*----3--
344:      TAG = 'TALLIES SPECIAL '
345: C      0----*----1----*----2----*----3--
346: C
347: C
348:      call SRTAG( NFILE, TAG, IRC )
349: C
350:      if ( IR.eq.0 ) then
351:          write(NOUT1,6888)
352:          write(NOUT1,*) ' XXX NO SPECIAL TALLY DATA !! STOP.'
353:          stop 999
354:      end if
355: C
356:      read(NFILE) TAG,NETALY
357: C
358: Cm      write(NOUT1,*) ' TAG : ',TAG
359: Cm      write(NOUT1,*) ' NETALY is ',NETALY
360: Cm      write(NOUT1,*) ' IDMONI is ',IDMONI
361: C
362:      call STLDR2(NFILE, IDMONI , NOUT1 ,
363:      &      NETALY, RRATE, ERATE1, ERATE2 )
364: C
365:      RATNOW = RRATE
366:      ERRNOW = ERATE1
367: C
368:      return
369: C
370:      160 continue
371:      write(NOUT1,6888)
372:      write(NOUT1,*) ' error at reading MVP binary file'
373:      write(NOUT1,*) ' probably MVP run is abnormal ended.'
374:      stop 999
375: C
376:      6888 format(/// ' XXX(REDMVP) ERROR-STOP : ' )
377:      6666 format(/// ' !!!(REDMVP) WARNING : ' )
378:      7020 format(' TITLE(2) in MVP output binary file ',
379:      &      'does not match that marked in input data.'/
380:      &      1X, ' :From file <',A,'>/1X, ' :On input <',A,'>/1X,
381:      &      ' probably current MVP calculation has not completed.')
382: C
383:      end

```

src/mvpburn/rwchan.f

```

1:      subroutine RWCHAN( MODE, PDS, CASEID,NPAR, NTNUC, NTFISS,
2:      &                   NUCLP, GAM, NBIC, PBIC )
3: C=====
4: C Read/write chain data to/from PDS member
5: C=====
6: C
7: C ---- Arguments -----
8: C
9:      character*(*) MODE
10:     character*(*) PDS
11:     character*4 CASEID
12: C
13:     integer NUCLP(NPAR,NTNUC)
14:     real GAM(NTFISS,NTNUC)
15:     integer NBIC(NPAR,NTNUC)
16:     real PBIC(NPAR,NTNUC)
17: C
18:     common /IOPRNT/ NOUT1, NOUT2
19: C
20: C ---- local data -----
21: C
22:     character*8 MEMBER
23:     character*32 TAG
24: C
25: C -----
26: C
27:     MEMBER = 'OPENING'
28:     call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, RARRAY, 3, NOUT1 )
29: C
30:     MEMBER = CASEID// 'CHAN'
31:
32:     if ( MODE.eq.'WRITE' ) then
33:
34:         call PDSFIL( 'WRITE',PDS,MEMBER, 'BINARY', 62, NOUT1, IERR0
35:         &
36:         if (IERR0.ne.0) then
37:             write(NOUT1,6888)
38:             write(NOUT1,*) ' writing chain data member(caseCHAN) failed'
39:             goto 100
40:         end if
41:
42:         TAG = 'SIZE'
43:         write(62) TAG, NPAR, NTNUC, NTFISS
44:         TAG = 'NUCLP'
45:         write(62) TAG,((NUCLP(J,I),J=1,NPAR),I=1,NTNUC)
46:         TAG = 'GAM'
47:         write(62) TAG,((GAM(J,I),J=1,NTFISS),I=1,NTNUC)
48:         TAG = 'NBIC'
49:         write(62) TAG,((NBIC(J,I),J=1,NPAR),I=1,NTNUC)
50:         TAG = 'PBIC'
51:         write(62) TAG,((PBIC(J,I),J=1,NPAR),I=1,NTNUC)
52:
53:         call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
54:         & IERR0 )
55: C
56:     else if ( MODE.eq.'READ' ) then
57:
58:         call PDSFIL( 'READ', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0
59:         &
60:         if (IERR0.ne.0) then
61:             write(NOUT1,6888)
62:             write(NOUT1,*) ' reading chain data member(caseCHAN) failed'
63:             goto 100
64:         end if
65:

```

```

66:         read(62) TAG,NPAR0, NTNUC0, NTFISS0
67:
68:         if ( NPAR0.ne.NPAR .or. NTNUC0.ne.NTNUC .or. NTFISS0.ne.NTFISS
69:         &
70:         ) then
71:             write(NOUT1,6666)
72:             write(NOUT1,*) ' array size in PDS member <',
73:             & MEMBER, '> differ from those of arguments.'
74:             write(NOUT1,*) ' NPAR,NTNUC,NTFISS in PDS: ', NPAR0,
75:             & NTNUC0, NTFISS0
76:             write(NOUT1,*) ' NPAR,NTNUC,NTFISS on argument: ', NPAR,
77:             & NTNUC, NTFISS
78:         end if
79:
80:         read(62) TAG,((NUCLP(J,I),J=1,NPAR),I=1,NTNUC)
81:
82:         read(62) TAG,((GAM(J,I),J=1,NTFISS),I=1,NTNUC)
83:
84:         read(62) TAG,((NBIC(J,I),J=1,NPAR),I=1,NTNUC)
85:         read(62) TAG,((PBIC(J,I),J=1,NPAR),I=1,NTNUC)
86:         call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
87:         & IERR0 )
88:
89:     else
90:         write(NOUT1,6888)
91:         write(NOUT1,*) ' may be programming error : ',
92:         & 'called with unexpected mode <',MODE,'>'
93:         write(NOUT1,*) ' PDS <', PDS(:ICLEN2(PDS)), '>'
94:         write(NOUT1,*) ' CASEID <', CASEID, '>'
95:         stop 999
96:     end if
97: C
98:     return
99: C
100: write(NOUT1,*) ' error in access to PDS member <', MEMBER,
101: & '> in <', MODE, '> mode. flag = ', IERR0
102: C
103: stop 999
104: C
105: 6666 format(/// '!!!(RWCHAN) WARNING :' )
106: 6888 format(/// 'XXX(RWCHAN) ERROR-STOP :' )
107: end

```

src/mvpburn/rwcom1.f

```

1:      subroutine RWCOM1(MODE, PDS, CASEID)
2: C=====
3: C Read/Write common /BURNCL/ to/from a PDS member
4: C=====
5:      character*(*) MODE
6:      character*(*) PDS
7:      character*4 CASEID
8:      include 'INC/_BURNP'
9: C -----
10: C
11:      include 'INC/_CBURNCL'
12: C
13:      common /IOPRNT/  NOUT1,  NOUT2
14: C
15: C ..... local data
16: C
17:      character*8 MEMBER
18:      character*32 TAG
19: C -----
20: C
21: C
22:      MEMBER = 'OPENING'
23:      call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, RARRAY, 3, NOUT1 )
24: C
25:      MEMBER = CASEID// 'COM1'
26:      LENG1  = 25 + 3*MXSTEP
27:      LENG   = LENG1
28: C
29: C
30:      if ( MODE.eq.'WRITE' ) then
31:
32:          call PDSFIL( 'WRITE',PDS,MEMBER,'BINARY', 62, NOUT1, IERR0
33:      &
34: C
35:          if (IERR0.ne.0) then
36:              write(NOUT1,6888)
37:              write(NOUT1,*) ' writing member(caseCOM1) failed'
38:              goto 100
39:          end if
40: C
41:          TAG = 'MAXIMUM'
42:          write(62) TAG,MXSTEP
43:          TAG = 'IBC'
44:          write(62) TAG,IBC
45:          TAG = 'SIZE'
46:          write(62) TAG,NEP, NEP1, LENG2, LENG3, LENG4
47:          TAG = 'PERIOD'
48:          write(62) TAG,(PERIOD(KK),KK=1,MXSTEP)
49:          TAG = 'POWERL'
50:          write(62) TAG,(POWERL(KK),KK=1,MXSTEP)
51:          TAG = 'NSTP'
52:          write(62) TAG,(NSTP(KK),KK=1,MXSTEP)
53:          TAG = 'EXTSRC'
54:          write(62) TAG,(EXTSRC(KK),KK=1,MXSTEP)
55:          TAG = 'VMONIT'
56:          write(62) TAG,IDMONI,(VMONIT(KK),KK=1,MXSTEP)
57:          TAG = 'REGPOW'
58:          write(62) TAG,(REGPOW(KK),KK=1,MXSTEP)
59:          call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
60:      &
61: C
62: C
63:      else if ( MODE.eq.'READ' ) then
64:
65:          call PDSFIL( 'READ',PDS,MEMBER,'BINARY', 62, NOUT1, IERR0

```

```

66:      &
67: C
68:          if (IERR0.ne.0) then
69:              write(NOUT1,6888)
70:              write(NOUT1,*) ' reading member(caseCOM1) failed'
71:              goto 100
72:          end if
73: C
74:          read(62) TAG,MXSTEP0
75: C
76:          if ( MXSTEP0.ne.MXSTEP ) then
77:              write(NOUT1,6666)
78:              write(NOUT1,*) ' array size in PDS member <',
79:      &
80:              write(NOUT1,*) ' MXSTEP in PDS: ', MXSTEP0
81:              write(NOUT1,*) ' MXSTEP on caller: ', MXSTEP
82:          end if
83: C
84:
85:          MXSTEP1 = min(MXSTEP,MXSTEP0)
86: C
87:          read(62) TAG,IBC
88:          read(62) TAG,NEP, NEP1, LENG2, LENG3, LENG4
89:          read(62) TAG,(PERIOD(KK),KK=1,MXSTEP1)
90:          read(62) TAG,(POWERL(KK),KK=1,MXSTEP1)
91:          read(62) TAG,(NSTP(KK),KK=1,MXSTEP1)
92:          read(62) TAG,(EXTSRC(KK),KK=1,MXSTEP1)
93:          read(62) TAG,IDMONI,(VMONIT(KK),KK=1,MXSTEP1)
94:          read(62) TAG,(REGPOW(KK),KK=1,MXSTEP1)
95:          call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
96:      &
97:          IERR0 )
98:      else
99:          write(NOUT1,6888)
100:          write(NOUT1,*) ' may be programming error : ',
101:      &
102:          'called with unexpected mode <',MODE,'>'
103:          write(NOUT1,*) ' PDS <', PDS(:ICLEN2(PDS)), '>'
104:          write(NOUT1,*) ' CASEID <', CASEID, '>'
105:          stop 999
106: C
107:      end if
108: C
109:      return
110: C
111:      100 write(NOUT1,*) ' error in opening member <', MEMBER,
112:      &
113:      '> in <', MODE, '> mode. flag = ', IERR0
114:      stop 999
115: C
116:      end

```

src/mvpburn/rwcom2.f

```

1:      subroutine RWCOM2( MODE, PDS, CASEID )
2: C=====
3: C Read/Write common /BURN2/ to/from a PDS member
4: C=====
5: C
6:      character*(*) MODE
7:      character*(*) PDS
8:      character*4 CASEID
9: C
10:     include 'INC/_BURNP'
11: C -----
12:     include 'INC/_CBURN2'
13: C
14: C
15:     common /IOPRNT/ NOUT1, NOUT2
16: C
17: C ..... local data
18: C
19:     character*8 MEMBER
20:     character*32 TAG
21: C -----
22: C
23: C
24:     MEMBER = 'OPENING'
25: C
26:     call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, RANRAY, 3, NOUT1 )
27: C
28:     MEMBER = CASEID//'.COM2'
29: C
30:     if ( MODE.eq.'WRITE' ) then
31:       call PDSFIL( 'WRITE',PDS,MEMBER,'BINARY', 62, NOUT1, IERR0
32: &
33:       if (IERR0.ne.0) then
34:         write(NOUT1,6888)
35:         write(NOUT1,*) ' writing member caseCOM2 failed'
36:         goto 100
37:       end if
38:
39:       TAG = 'MAXIMUM'
40:       write(62) TAG,MXFISS, MXNUC
41:       TAG = 'SIZE'
42:       write(62) TAG,NCARD, NMAT, KNMAX, METHPC, IVOID, ST235, TWTHVY,
43: &      EVTOJ, ABOGA, BZERO, LASTFP, IDU235, IBUNIT, IBEDIT,
44: &      NTNUC, NTFISS, NPAR, NCHA, LCHA, IDXE35, IDI135,
45: &      IDSM49, IDPM49, NFIS, NFER
46:       TAG = 'IFISDN'
47:       write(62) TAG,(IFISDN(I),I=1,MXFISS)
48:       TAG = 'FISFCT'
49:       write(62) TAG,(FISFCT(I),I=1,MXFISS)
50:       TAG = 'IFRTDN'
51:       write(62) TAG,(IFRTDN(I),I=1,MXFISS)
52:       TAG = 'FRTFCT'
53:       write(62) TAG,(FRTFCT(I),I=1,MXFISS)
54:       TAG = 'REACEV'
55:       write(62) TAG,(REACEV(1,I),REACEV(2,I),I=1,MXNUC)
56:       TAG = 'IFISS'
57:       write(62) TAG,(IFISS(I),I=1,MXNUC)
58:       TAG = 'AMASS'
59:       write(62) TAG,(AMASS(I),I=1,MXNUC)
60:       TAG = 'NUCL'
61:       write(62) TAG,(NUCL(I),I=1,MXNUC)
62:       TAG = 'FACT2N'
63:       write(62) TAG,(FACT2N(I),I=1,MXNUC)
64:       TAG = 'NCH'
65:       write(62) TAG,(NCH(I),I=1,MXNUC)

```

```

66:       TAG = 'RAMDA'
67:       write(62) TAG,(RAMDA(I),I=1,MXNUC)
68: C2003 ...
69:       TAG = 'NCODE'
70:       write(62) TAG,(NCODE(I),I=1,MXNUC)
71: C2003
72:       TAG = 'STDNUC'
73:       write(62) TAG,STDNUC
74:       TAG = 'NAMFIS'
75:       write(62) TAG,(NAMFIS(I),I=1,MXFISS)
76:       TAG = 'NAMFER'
77:       write(62) TAG,(NAMFER(I),I=1,MXFISS)
78: C2003 ... end
79: C2005
80:       TAG = 'YLDNUC'
81:       write(62) TAG,NCXE35, NCII35, NCSM49, NCPM49
82: C2005 ... end
83: C
84:       call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
85: &      IERR0 )
86: C
87:       else if ( MODE.eq.'READ' ) then
88:         call PDSFIL( 'READ', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0
89: &
90:         if (IERR0.ne.0) then
91:           write(NOUT1,6888)
92:           write(NOUT1,*) ' reading member caseCOM2 failed'
93:           goto 100
94:         end if
95:
96:         read(62) TAG,MXFISS0, MXNUC0
97:         if ( MXFISS0.ne.MXFISS .or. MXNUC0.ne.MXNUC ) then
98:           write(NOUT1,6666)
99:           write(NOUT1,*) ' array size in PDS member <',
100: &      MEMBER, '> differ from those of calling routine.'
101:           write(NOUT1,*) ' MXFISS,MXNUC in PDS: ', MXFISS0,MXNUC0
102:           write(NOUT1,*) ' MXFISS,MXNUC on caller: ', MXFISS,MXNUC
103:         end if
104:
105:         read(62) TAG,NCARD, NMAT, KNMAX, METHPC, IVOID, ST235, TWTHVY,
106: &      EVTOJ, ABOGA, BZERO, LASTFP, IDU235, IBUNIT, IBEDI0,
107: &      NTNUC, NTFISS, NPAR, NCHA, LCHA, IDXE35, IDI135,
108: &      IDSM49, IDPM49, NFIS, NFER
109:         MXFISS1 = min( MXFISS,MXFISS0 )
110:         MXNUC1 = min( MXNUC, MXNUC0 )
111:
112:         read(62) TAG,(IFISDN(I),I=1,MXFISS1)
113:         read(62) TAG,(FISFCT(I),I=1,MXFISS1)
114:         read(62) TAG,(IFRTDN(I),I=1,MXFISS1)
115:         read(62) TAG,(FRTFCT(I),I=1,MXFISS1)
116:         read(62) TAG,(REACEV(1,I),REACEV(2,I),I=1,MXNUC1)
117:         read(62) TAG,(IFISS(I),I=1,MXNUC1)
118:         read(62) TAG,(AMASS(I),I=1,MXNUC1)
119:         read(62) TAG,(NUCL(I),I=1,MXNUC1)
120:         read(62) TAG,(FACT2N(I),I=1,MXNUC1)
121:         read(62) TAG,(NCH(I),I=1,MXNUC1)
122:         read(62) TAG,(RAMDA(I),I=1,MXNUC1)
123: C2003
124:         read(62) TAG,(NCODE(I),I=1,MXNUC1)
125: C2003
126:         read(62) TAG,STDNUC
127:         read(62) TAG,(NAMFIS(I),I=1,MXFISS1)
128:         read(62) TAG,(NAMFER(I),I=1,MXFISS1)
129: C2003 ... end
130: C2005

```

src/mvpburn/rwcom2.f

```
131:      read(62) TAG,NCXE35, NCI135, NCSM49, NCPM49
132: C2005 ... end
133: C
134:      call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
135:      &          IERR0 )
136: C
137:      else
138:      write(NOUT1,6888)
139:      write(NOUT1,*) ' may be programming error : ',
140:      &          'called with unexpected mode <',MODE,'>'
141:      write(NOUT1,*) ' PDS      <', PDS(:ICLEN2(PDS)), '>'
142:      write(NOUT1,*) ' CASEID <', CASEID, '>'
143:      stop 999
144:      end if
145: C
146:      return
147: C
148: 100 write(NOUT1,*) ' error in opening member <', MEMBER,
149:      &          '> in <', MODE, '> mode.  flag = ', IERR0
150:      stop 999
151: C
152: 6666 format(/// '!!!(RWCOM2) WARNING :' )
153: 6888 format(/// 'XXX(RWCOM2) ERROR-STOP :' )
154:      end
```

src/mvpburn/rwcom3.f

```

1:      subroutine RWCOM3( MODE, PDS, CASEID )
2: C=====
3: C Read/Write common /BURNC3/ to/from a PDS member
4: C=====
5: C
6:      character*(*) MODE
7:      character*(*) PDS
8:      character*4 CASEID
9:      include 'INC/_BURNP'
10: C
11:      include 'INC/_CBURNC3'
12: C
13:      common /IOPRNT/ NOUT1, NOUT2
14: C
15: C ..... local data
16: C
17:      character*8 MEMBER
18:      character*32 TAG
19: C
20: C -----
21: C
22:      MEMBER = 'OPENING'
23: C
24:      call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, RARRAY, 3, NOUT1 )
25: C
26:      MEMBER = CASEID// 'COM3'
27: C
28:      if ( MODE.eq.'WRITE' ) then
29:
30:          call PDSFIL( 'WRITE',PDS,MEMBER,'BINARY', 62, NOUT1, IERR0
31:      &
32:          if (IERR0.ne.0) then
33:              write(NOUT1,6888)
34:              write(NOUT1,*) ' writing member caseCOM3 failed'
35:              goto 100
36:          end if
37:
38:          TAG = 'MAXIMUM'
39:          write(62) TAG,MXFISS, MXNUC
40: C2003
41: CCC      TAG = 'STDNUC'
42: CCC      write(62) TAG,STDNUC, CASBRN, CASPCM
43: CCC      TAG = 'CASBRN'
44: CCC      write(62) TAG, CASBRN, CASPCM
45: C2003
46: CCC      TAG = 'NAMFIS'
47: CCC      write(62) TAG,(NAMFIS(I),I=1,MXFISS)
48: CCC      TAG = 'NAMFER'
49: CCC      write(62) TAG,(NAMFER(I),I=1,MXFISS)
50: C2005
51: CCC      TAG = 'SRACID'
52: CCC      write(62) TAG,(SRACID(I),I=1,MXNUC)
53: CCC      TAG = 'TITLE'
54: CCC      write(62) TAG,(TITLE(I),I=1,2)
55: C2003
56: CCC      TAG = 'IHOL'
57: CCC      write(62) TAG,(IHOL(I),I=1,MXNUC)
58:
59:      call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
60:      &
61:          IERR0 )
62:
63:      else if ( MODE.eq.'READ' ) then
64:
65:          call PDSFIL( 'READ', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0
66:      &
67:          if (IERR0.ne.0) then
68:              write(NOUT1,6888)
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102: C
103:
104: C
105:
106:
107:
108: C
109:
110:
111: C
112:

```

```

66:      write(NOUT1,*) ' reading member caseCOM3 failed'
67:      goto 100
68:      end if
69:
70:      read(62) TAG,MXFISS0, MXNUC0
71:      if ( MXFISS0.ne.MXFISS .or. MXNUC0.ne.MXNUC ) then
72:          write(NOUT1,6666)
73:          write(NOUT1,*) ' array size in PDS member <',
74:      &
75:          MEMBER, '> differ from those of calling routine.'
76:          write(NOUT1,*) ' MXFISS,MXNUC in PDS: ', MXFISS0,MXNUC0
77:          write(NOUT1,*) ' MXFISS,MXNUC on caller: ', MXFISS,MXNUC
78:      end if
79:
80:      MXFISS1 = min( MXFISS,MXFISS0 )
81:      MXNUC1 = min( MXNUC, MXNUC0 )
82: C2003
83: read(62) TAG,STDNUC, CASBRN, CASPCM
84: C2003
85: read(62) TAG,(NAMFIS(I),I=1,MXFISS1)
86: C2003
87: read(62) TAG,(NAMFER(I),I=1,MXFISS1)
88: C2005
89: read(62) TAG,(SRACID(I),I=1,MXNUC1)
90: read(62) TAG,(TITLE(I),I=1,2)
91: C2003
92: read(62) TAG,(IHOL(I),I=1,MXNUC1)
93:
94:      call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
95:      &
96:          IERR0 )
97:
98:      else
99:          write(NOUT1,6888)
100:          write(NOUT1,*) ' may be programming error : ',
101:      &
102:          'called with unexpected mode <',MODE,'>'
103:          write(NOUT1,*) ' PDS <', PDS(:ICLEN2(PDS)), '>'
104:          write(NOUT1,*) ' CASEID <', CASEID, '>'
105:          stop 999
106:      end if
107:
108: C
109:
110: C
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:

```

src/mvpburn/rwmatd.f

```

1:      subroutine RWMATD( MODE, PDS, CASEID,NMAT, KNMAX, NISO, TEMP,
2:      &                  VOLM, TRGNAM,LENTRG,MTBURN,MTCARD,IDENT,
3:      C2005&            DNINIT,MATMVP,IPBURN )
4:      &                  DNINIT,MATMVP,IPBURN,MATREG )
5:      C=====
6:      C Read/Write material data to/from a PDS member
7:      C=====
8:      C
9:      character*(*) MODE
10:     character*4 CASEID
11:     character*(*) PDS
12:   C
13:   C
14: C2003
15:     character*12 IDENT(KNMAX,NMAT)
16:     character*12 TRGNAM(NMAT)
17:   C
18:     integer NISO(NMAT), LENTRG(NMAT)
19:     integer IPBURN(KNMAX,NMAT), MATMVP(NMAT)
20:     integer MTBURN(3,NMAT), MTCARD(2,NMAT)
21:   C
22:     real TEMP(NMAT), VOLM(NMAT)
23:     real DNINIT(KNMAX,NMAT)
24: C2005
25:     integer MATREG(NMAT)
26: Cend
27:   C
28:     common /IOPRNT/ NOUT1, NOUT2
29:   C
30:   C ... local variables
31:   C
32:     character*8 MEMBER
33:     character*32 TAG
34:   C
35:   C-----
36:   C
37:     MEMBER = 'OPENING'
38:     call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, A, 3, NOUT1 )
39:   C
40:     MEMBER = CASEID// 'MATD'
41:   C
42:     if ( MODE.eq.'WRITE' ) then
43:       call PDSFIL( 'WRITE',PDS,MEMBER,'BINARY',62,NOUT1, IERR0
44: &
45:       if (IERR0.ne.0) then
46:         write(NOUT1,6888)
47:         write(NOUT1,*) ' writing member caseMATD failed'
48:         goto 100
49:       end if
50:
51:       TAG = 'SIZE'
52:       write(62) TAG,NMAT, KNMAX
53:       TAG = 'NISO'
54:       write(62) TAG,(NISO(I),I=1,NMAT)
55:       TAG = 'TEMP'
56:       write(62) TAG,(TEMP(I),I=1,NMAT)
57:       TAG = 'VOLM'
58:       write(62) TAG,(VOLM(I),I=1,NMAT)
59:       TAG = 'TRGNAM'
60:       write(62) TAG,(TRGNAM(I),I=1,NMAT)
61:       TAG = 'LENTRG'
62:       write(62) TAG,(LENTRG(I),I=1,NMAT)
63:       TAG = 'MTBURN'
64:       write(62) TAG,((MTBURN(J,I),J=1,3),I=1,NMAT)
65:       TAG = 'MTCARD'

```

```

66:       write(62) TAG,((MTCARD(J,I),J=1,2),I=1,NMAT)
67:       TAG = 'IDENT'
68:       write(62) TAG,((IDENT(J,I),J=1,KNMAX),I=1,NMAT)
69:       TAG = 'DNINIT'
70:       write(62) TAG,((DNINIT(J,I),J=1,KNMAX),I=1,NMAT)
71:       TAG = 'MATMVP'
72:       write(62) TAG,(MATMVP(I),I=1,NMAT)
73:       TAG = 'IPBURN'
74:       write(62) TAG,((IPBURN(J,I),J=1,KNMAX),I=1,NMAT)
75: C2005
76:       TAG = 'MATREG'
77:       write(62) TAG,(MATREG(I),I=1,NMAT)
78: Cend
79:       call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
80: &                  IERR0 )
81:
82:       else if ( MODE.eq.'READ' ) then
83:
84:         call PDSFIL( 'READ', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0
85: &
86:         if (IERR0.ne.0) then
87:           write(NOUT1,6888)
88:           write(NOUT1,*) ' reading member caseMATD failed'
89:           goto 100
90:         end if
91:
92:         read(62) TAG,NMAT0, KNMAX0
93:         if ( NMAT0.ne.NMAT .or. KNMAX0.ne.KNMAX ) then
94:           write(NOUT1,6666)
95:           write(NOUT1,*) ' array size in PDS member <',
96: &
97:           MEMBER, '> differ from those of calling routine.'
98:           write(NOUT1,*) ' NMAT,KNMAX in PDS: ', NMAT0, KNMAX0
99:           write(NOUT1,*) ' NMAT,KNMAX on caller: ', NMAT, KNMAX
100:         end if
101:
102:         NMAT1 = min( NMAT,NMAT0 )
103:         KNMAX1 = min( KNMAX, KNMAX0 )
104:
105:         read(62) TAG,(NISO(I),I=1,NMAT1)
106:         read(62) TAG,(TEMP(I),I=1,NMAT1)
107:         read(62) TAG,(VOLM(I),I=1,NMAT1)
108:         read(62) TAG,(TRGNAM(I),I=1,NMAT1)
109:         read(62) TAG,(LENTRG(I),I=1,NMAT1)
110:         read(62) TAG,((MTBURN(J,I),J=1,3),I=1,NMAT1)
111:         read(62) TAG,((MTCARD(J,I),J=1,2),I=1,NMAT1)
112:         read(62) TAG,((IDENT(J,I),J=1,KNMAX1),I=1,NMAT1)
113:         read(62) TAG,((DNINIT(J,I),J=1,KNMAX1),I=1,NMAT1)
114:         read(62) TAG,(MATMVP(I),I=1,NMAT1)
115:         read(62) TAG,((IPBURN(J,I),J=1,KNMAX1),I=1,NMAT1)
116: C2005
117:         read(62) TAG,(MATREG(I),I=1,NMAT1)
118: Cend
119:         call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1,
120: &                  IERR0 )
121:
122:       else
123:         write(NOUT1,*) ' may be programming error : ',
124: &
125:         'called with unexpected mode <',MODE,'>'
126:         write(NOUT1,*) ' PDS <', PDS(:ICLEN2(PDS)), '>'
127:         write(NOUT1,*) ' CASEID <', CASEID, '>'
128:         stop 999
129:       end if
130:   C
131:   C
132:   C
133:   C
134:   C
135:   C
136:   C
137:   C
138:   C
139:   C
140:   C
141:   C
142:   C
143:   C
144:   C
145:   C
146:   C
147:   C
148:   C
149:   C
150:   C
151:   C
152:   C
153:   C
154:   C
155:   C
156:   C
157:   C
158:   C
159:   C
160:   C
161:   C
162:   C
163:   C
164:   C
165:   C
166:   C
167:   C
168:   C
169:   C
170:   C
171:   C
172:   C
173:   C
174:   C
175:   C
176:   C
177:   C
178:   C
179:   C
180:   C
181:   C
182:   C
183:   C
184:   C
185:   C
186:   C
187:   C
188:   C
189:   C
190:   C
191:   C
192:   C
193:   C
194:   C
195:   C
196:   C
197:   C
198:   C
199:   C
200:   C
201:   C
202:   C
203:   C
204:   C
205:   C
206:   C
207:   C
208:   C
209:   C
210:   C
211:   C
212:   C
213:   C
214:   C
215:   C
216:   C
217:   C
218:   C
219:   C
220:   C
221:   C
222:   C
223:   C
224:   C
225:   C
226:   C
227:   C
228:   C
229:   C
230:   C
231:   C
232:   C
233:   C
234:   C
235:   C
236:   C
237:   C
238:   C
239:   C
240:   C
241:   C
242:   C
243:   C
244:   C
245:   C
246:   C
247:   C
248:   C
249:   C
250:   C
251:   C
252:   C
253:   C
254:   C
255:   C
256:   C
257:   C
258:   C
259:   C
260:   C
261:   C
262:   C
263:   C
264:   C
265:   C
266:   C
267:   C
268:   C
269:   C
270:   C
271:   C
272:   C
273:   C
274:   C
275:   C
276:   C
277:   C
278:   C
279:   C
280:   C
281:   C
282:   C
283:   C
284:   C
285:   C
286:   C
287:   C
288:   C
289:   C
290:   C
291:   C
292:   C
293:   C
294:   C
295:   C
296:   C
297:   C
298:   C
299:   C
300:   C
301:   C
302:   C
303:   C
304:   C
305:   C
306:   C
307:   C
308:   C
309:   C
310:   C
311:   C
312:   C
313:   C
314:   C
315:   C
316:   C
317:   C
318:   C
319:   C
320:   C
321:   C
322:   C
323:   C
324:   C
325:   C
326:   C
327:   C
328:   C
329:   C
330:   C
331:   C
332:   C
333:   C
334:   C
335:   C
336:   C
337:   C
338:   C
339:   C
340:   C
341:   C
342:   C
343:   C
344:   C
345:   C
346:   C
347:   C
348:   C
349:   C
350:   C
351:   C
352:   C
353:   C
354:   C
355:   C
356:   C
357:   C
358:   C
359:   C
360:   C
361:   C
362:   C
363:   C
364:   C
365:   C
366:   C
367:   C
368:   C
369:   C
370:   C
371:   C
372:   C
373:   C
374:   C
375:   C
376:   C
377:   C
378:   C
379:   C
380:   C
381:   C
382:   C
383:   C
384:   C
385:   C
386:   C
387:   C
388:   C
389:   C
390:   C
391:   C
392:   C
393:   C
394:   C
395:   C
396:   C
397:   C
398:   C
399:   C
400:   C
401:   C
402:   C
403:   C
404:   C
405:   C
406:   C
407:   C
408:   C
409:   C
410:   C
411:   C
412:   C
413:   C
414:   C
415:   C
416:   C
417:   C
418:   C
419:   C
420:   C
421:   C
422:   C
423:   C
424:   C
425:   C
426:   C
427:   C
428:   C
429:   C
430:   C
431:   C
432:   C
433:   C
434:   C
435:   C
436:   C
437:   C
438:   C
439:   C
440:   C
441:   C
442:   C
443:   C
444:   C
445:   C
446:   C
447:   C
448:   C
449:   C
450:   C
451:   C
452:   C
453:   C
454:   C
455:   C
456:   C
457:   C
458:   C
459:   C
460:   C
461:   C
462:   C
463:   C
464:   C
465:   C
466:   C
467:   C
468:   C
469:   C
470:   C
471:   C
472:   C
473:   C
474:   C
475:   C
476:   C
477:   C
478:   C
479:   C
480:   C
481:   C
482:   C
483:   C
484:   C
485:   C
486:   C
487:   C
488:   C
489:   C
490:   C
491:   C
492:   C
493:   C
494:   C
495:   C
496:   C
497:   C
498:   C
499:   C
500:   C
501:   C
502:   C
503:   C
504:   C
505:   C
506:   C
507:   C
508:   C
509:   C
510:   C
511:   C
512:   C
513:   C
514:   C
515:   C
516:   C
517:   C
518:   C
519:   C
520:   C
521:   C
522:   C
523:   C
524:   C
525:   C
526:   C
527:   C
528:   C
529:   C
530:   C
531:   C
532:   C
533:   C
534:   C
535:   C
536:   C
537:   C
538:   C
539:   C
540:   C
541:   C
542:   C
543:   C
544:   C
545:   C
546:   C
547:   C
548:   C
549:   C
550:   C
551:   C
552:   C
553:   C
554:   C
555:   C
556:   C
557:   C
558:   C
559:   C
560:   C
561:   C
562:   C
563:   C
564:   C
565:   C
566:   C
567:   C
568:   C
569:   C
570:   C
571:   C
572:   C
573:   C
574:   C
575:   C
576:   C
577:   C
578:   C
579:   C
580:   C
581:   C
582:   C
583:   C
584:   C
585:   C
586:   C
587:   C
588:   C
589:   C
590:   C
591:   C
592:   C
593:   C
594:   C
595:   C
596:   C
597:   C
598:   C
599:   C
600:   C
601:   C
602:   C
603:   C
604:   C
605:   C
606:   C
607:   C
608:   C
609:   C
610:   C
611:   C
612:   C
613:   C
614:   C
615:   C
616:   C
617:   C
618:   C
619:   C
620:   C
621:   C
622:   C
623:   C
624:   C
625:   C
626:   C
627:   C
628:   C
629:   C
630:   C
631:   C
632:   C
633:   C
634:   C
635:   C
636:   C
637:   C
638:   C
639:   C
640:   C
641:   C
642:   C
643:   C
644:   C
645:   C
646:   C
647:   C
648:   C
649:   C
650:   C
651:   C
652:   C
653:   C
654:   C
655:   C
656:   C
657:   C
658:   C
659:   C
660:   C
661:   C
662:   C
663:   C
664:   C
665:   C
666:   C
667:   C
668:   C
669:   C
670:   C
671:   C
672:   C
673:   C
674:   C
675:   C
676:   C
677:   C
678:   C
679:   C
680:   C
681:   C
682:   C
683:   C
684:   C
685:   C
686:   C
687:   C
688:   C
689:   C
690:   C
691:   C
692:   C
693:   C
694:   C
695:   C
696:   C
697:   C
698:   C
699:   C
700:   C
701:   C
702:   C
703:   C
704:   C
705:   C
706:   C
707:   C
708:   C
709:   C
710:   C
711:   C
712:   C
713:   C
714:   C
715:   C
716:   C
717:   C
718:   C
719:   C
720:   C
721:   C
722:   C
723:   C
724:   C
725:   C
726:   C
727:   C
728:   C
729:   C
730:   C
731:   C
732:   C
733:   C
734:   C
735:   C
736:   C
737:   C
738:   C
739:   C
740:   C
741:   C
742:   C
743:   C
744:   C
745:   C
746:   C
747:   C
748:   C
749:   C
750:   C
751:   C
752:   C
753:   C
754:   C
755:   C
756:   C
757:   C
758:   C
759:   C
760:   C
761:   C
762:   C
763:   C
764:   C
765:   C
766:   C
767:   C
768:   C
769:   C
770:   C
771:   C
772:   C
773:   C
774:   C
775:   C
776:   C
777:   C
778:   C
779:   C
780:   C
781:   C
782:   C
783:   C
784:   C
785:   C
786:   C
787:   C
788:   C
789:   C
790:   C
791:   C
792:   C
793:   C
794:   C
795:   C
796:   C
797:   C
798:   C
799:   C
800:   C
801:   C
802:   C
803:   C
804:   C
805:   C
806:   C
807:   C
808:   C
809:   C
810:   C
811:   C
812:   C
813:   C
814:   C
815:   C
816:   C
817:   C
818:   C
819:   C
820:   C
821:   C
822:   C
823:   C
824:   C
825:   C
826:   C
827:   C
828:   C
829:   C
830:   C
831:   C
832:   C
833:   C
834:   C
835:   C
836:   C
837:   C
838:   C
839:   C
840:   C
841:   C
842:   C
843:   C
844:   C
845:   C
846:   C
847:   C
848:   C
849:   C
850:   C
851:   C
852:   C
853:   C
854:   C
855:   C
856:   C
857:   C
858:   C
859:   C
860:   C
861:   C
862:   C
863:   C
864:   C
865:   C
866:   C
867:   C
868:   C
869:   C
870:   C
871:   C
872:   C
873:   C
874:   C
875:   C
876:   C
877:   C
878:   C
879:   C
880:   C
881:   C
882:   C
883:   C
884:   C
885:   C
886:   C
887:   C
888:   C
889:   C
890:   C
891:   C
892:   C
893:   C
894:   C
895:   C
896:   C
897:   C
898:   C
899:   C
900:   C
901:   C
902:   C
903:   C
904:   C
905:   C
906:   C
907:   C
908:   C
909:   C
910:   C
911:   C
912:   C
913:   C
914:   C
915:   C
916:   C
917:   C
918:   C
919:   C
920:   C
921:   C
922:   C
923:   C
924:   C
925:   C
926:   C
927:   C
928:   C
929:   C
930:   C
931:   C
932:   C
933:   C
934:   C
935:   C
936:   C
937:   C
938:   C
939:   C
940:   C
941:   C
942:   C
943:   C
944:   C
945:   C
946:   C
947:   C
948:   C
949:   C
950:   C
951:   C
952:   C
953:   C
954:   C
955:   C
956:   C
957:   C
958:   C
959:   C
960:   C
961:   C
962:   C
963:   C
964:   C
965:   C
966:   C
967:   C
968:   C
969:   C
970:   C
971:   C
972:   C
973:   C
974:   C
975:   C
976:   C
977:   C
978:   C
979:   C
980:   C
981:   C
982:   C
983:   C
984:   C
985:   C
986:   C
987:   C
988:   C
989:   C
990:   C
991:   C
992:   C
993:   C
994:   C
995:   C
996:   C
997:   C
998:   C
999:   C

```


src/mvpburn/rwmatd.f

```
131: 100 write(NOUT1,*) ' error in access to PDS member <', MEMBER,  
132: & '> in <', MODE, '> mode. flag = ', IERR0  
133: stop 999  
134: C  
135: 6666 format(/// !!!(RWMTD) WARNING :' )  
136: 6888 format(/// XXX(RWMTD) ERROR-STOP :' )  
137: C  
138: end
```

SAFE

src/mvpburn/rwrest.f

```

1:      subroutine RWREST( MODE, PDS, CASEID )
2: C=====
3: C Read/Write common /REST/ to/from a PDS member
4: C=====
5: C
6:      character*(*) MODE
7:      character*(*) PDS
8:      character*4 CASEID
9:      include 'INC/_BURNP'
10: C
11:      real INSCR, INTCR
12:      common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
13: & ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
14: & EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
15: & INTCR(MXSTEP), EINSKR(MXSTEP), EINTCR(MXSTEP),
16: & FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
17: & FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
18: & FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
19: & NBATCH(MXSTEP)
20: C2005
21: & ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
22: & RMONIT(MXSTEP), EMONIT(MXSTEP)
23: Cend
24: C
25:      common /IOPRNT/ NOUT1, NOUT2
26: C
27: C ..... local data
28: C
29:      character*8 MEMBER
30:      character*32 TAG
31: C
32: C -----
33: C
34:      MEMBER = 'OPENING'
35:      call PDSIO( 'OPEN', PDS, MEMBER, MEMBER, RARRAY, 3, NOUT1 )
36: C
37:      MEMBER = CASEID//'.REST'
38: C
39:      if ( MODE.eq.'WRITE' ) then
40:
41:          call PDSFIL( 'WRITE', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
42:          if (IERR0.ne.0) then
43:              write(NOUT1,6888)
44:              write(NOUT1,*) ' writing member(caseREST) failed'
45:              goto 120
46:          end if
47:
48:          TAG = 'MAXIMUM'
49:          write(62) TAG, MXSTEP
50:
51:          TAG = 'STEP'
52:          write(62) TAG, NOWSTP, IBEND
53:          TAG = 'AKEFF'
54:          write(62) TAG, (AKEFF(I),I=1,MXSTEP)
55:          TAG = 'ERRKEF'
56:          write(62) TAG, (ERRKEF(I),I=1,MXSTEP)
57:          TAG = 'DAYS'
58:          write(62) TAG, (DAYS(I),I=1,MXSTEP)
59:          TAG = 'U235F'
60:          write(62) TAG, (U235F(I),I=1,MXSTEP)
61:          TAG = 'EXPST'
62:          write(62) TAG, (EXPST(I),I=1,MXSTEP)
63:          TAG = 'CUMMWD'
64:          write(62) TAG, (CUMMWD(I),I=1,MXSTEP)
65:          TAG = 'INSCR'

```

```

66:          write(62) TAG, (INSCR(I),I=1,MXSTEP)
67:          TAG = 'INTCR'
68:          write(62) TAG, (INTCR(I),I=1,MXSTEP)
69:          TAG = 'EINSKR'
70:          write(62) TAG, (EINSKR(I),I=1,MXSTEP)
71:          TAG = 'EINTCR'
72:          write(62) TAG, (EINTCR(I),I=1,MXSTEP)
73:          TAG = 'FLXNRM'
74:          write(62) TAG, (FLXNRM(I),I=1,MXSTEP)
75:          TAG = 'FACNRM'
76:          write(62) TAG, (FACNRM(I),I=1,MXSTEP)
77:          TAG = 'FISABS'
78:          write(62) TAG, (FISABS(I),I=1,MXSTEP)
79:          TAG = 'FRTCAP'
80:          write(62) TAG, (FRTCAP(I),I=1,MXSTEP)
81:          TAG = 'EFISAB'
82:          write(62) TAG, (EFISAB(I),I=1,MXSTEP)
83:          TAG = 'EFRTCA'
84:          write(62) TAG, (EFRTCA(I),I=1,MXSTEP)
85:          TAG = 'FDECAY'
86:          write(62) TAG, (FDECAY(I),I=1,MXSTEP)
87:          TAG = 'CDECAY'
88:          write(62) TAG, (CDECAY(I),I=1,MXSTEP)
89:          TAG = 'NHIST'
90:          write(62) TAG, (NHIST(I),I=1,MXSTEP)
91:          TAG = 'NBATCH'
92:          write(62) TAG, (NBATCH(I),I=1,MXSTEP)
93: C2005
94:          TAG = 'ERRNRM'
95:          write(62) TAG, (ERRNRM(I),I=1,MXSTEP)
96:          TAG = 'POWERW'
97:          write(62) TAG, (POWERW(I),I=1,MXSTEP)
98:          TAG = 'POWERE'
99:          write(62) TAG, (POWERE(I),I=1,MXSTEP)
100:          TAG = 'RMONIT'
101:          write(62) TAG, (RMONIT(I),I=1,MXSTEP)
102:          TAG = 'EMONIT'
103:          write(62) TAG, (EMONIT(I),I=1,MXSTEP)
104: Cend
105:          call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
106: C
107:          else if ( MODE.eq.'READ' ) then
108: C00/01/27
109:              call PDSIO( 'EXIST', PDS, MEMBER, MEMBER, RARRAY, IDATA,
110: & NOUT1 )
111:              if ( IDATA.lt.0 ) then
112:                  write(NOUT1,6666)
113:                  write(NOUT1,*)
114:                  & ' member ',CASEID,'.REST member is not found in pds file.'
115:                  return
116:              end if
117:
118:              call PDSFIL( 'READ', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
119:              if (IERR0.ne.0) then
120:                  write(NOUT1,6888)
121:                  write(NOUT1,*) ' reading member(caseREST) failed'
122:                  goto 120
123:              end if
124:
125:              read(62,err =100) TAG, MXSTEP0
126:              go to 110
127: C
128: C ... compatibility with earliest version (no TAG in MXSTEP record!)
129:          100      backspace 62
130:              read(62) MXSTEP0

```

src/mvpburn/rwrest.f

```
131: 110 continue
132: C
133: if ( MXSTEP0.ne.MXSTEP ) then
134: write(NOUT1,6666)
135: write(NOUT1,*) ' array size in PDS member <',
136: & MEMBER, '> differ from those of caller rourine.'
137: write(NOUT1,*) ' MXSTEP in PDS: ', MXSTEP0
138: write(NOUT1,*) ' MXSTEP on caller: ', MXSTEP
139: end if
140:
141: MXSTEP1 = MIN(MXSTEP,MXSTEP0)
142:
143: read(62) TAG, NOWSTP, IBEND
144: C
145: read(62) TAG, (AKERF(I),I=1,MXSTEP1)
146: read(62) TAG, (ERRKEF(I),I=1,MXSTEP1)
147: read(62) TAG, (DAYS(I),I=1,MXSTEP1)
148: read(62) TAG, (U235F(I),I=1,MXSTEP1)
149: read(62) TAG, (EXPST(I),I=1,MXSTEP1)
150: read(62) TAG, (CUMMWD(I),I=1,MXSTEP1)
151: read(62) TAG, (INSCR(I),I=1,MXSTEP1)
152: read(62) TAG, (INTCR(I),I=1,MXSTEP1)
153: read(62) TAG, (EINSCR(I),I=1,MXSTEP1)
154: read(62) TAG, (EINTCR(I),I=1,MXSTEP1)
155: read(62) TAG, (FLXNRM(I),I=1,MXSTEP1)
156: read(62) TAG, (FACNRM(I),I=1,MXSTEP1)
157: read(62) TAG, (FISABS(I),I=1,MXSTEP1)
158: read(62) TAG, (FRTCAP(I),I=1,MXSTEP1)
159: read(62) TAG, (EFISAB(I),I=1,MXSTEP1)
160: read(62) TAG, (EFRTCA(I),I=1,MXSTEP1)
161: read(62) TAG, (FDECAY(I),I=1,MXSTEP1)
162: read(62) TAG, (CDECAY(I),I=1,MXSTEP1)
163: read(62) TAG, (NHIST(I),I=1,MXSTEP1)
164: read(62) TAG, (NBATCH(I),I=1,MXSTEP1)
165: C2005
166: read(62) TAG, (ERRNRM(I),I=1,MXSTEP1)
167: read(62) TAG, (POWERW(I),I=1,MXSTEP1)
168: read(62) TAG, (POWERY(I),I=1,MXSTEP1)
169: read(62) TAG, (RMONIT(I),I=1,MXSTEP1)
170: read(62) TAG, (EMONIT(I),I=1,MXSTEP1)
171: Cend
172: call PDSFIL( 'CLOSE', PDS, MEMBER, 'BINARY', 62, NOUT1, IERR0 )
173:
174: else
175: write(NOUT1,6888)
176: write(NOUT1,*) ' may be programming error : ',
177: & 'called with unexpected mode <',MODE,'>'
178: write(NOUT1,*) ' PDS <', PDS(:ICLEN2(PDS)), '>'
179: write(NOUT1,*) ' CASEID <', CASEID, '>'
180: stop 999
181: end if
182: C
183: return
184: C
185: 120 write(NOUT1,*) ' error in access to PDS member <', MEMBER,
186: & '> in <', MODE, '> mode. flag = ', IERR0
187: stop 999
188:
189: 6666 format(// ' !!!(RWREST) WARNING : ' )
190: 6888 format(// ' XXX(RWREST) ERROR-STOP : ' )
191: C
192: end
```

src/mvpburn/sctolc.f

```
1: C*****
2: C*   Change small character to large character
3: C*
4: C*   ICHAR : character string (arbitrary length)
5: C*   M     : change case up to M characters
6: C*****
7:   subroutine SCTOLC( ICH, M )
8: C
9:   character*(*) ICH
10: C
11:   character SMALL*26, LARGE*26
12: C
13:   SMALL = 'abcdefghijklmnopqrstuvwxyz'
14:   LARGE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
15: C
16:   do 100 I = 1, MIN(M,LEN(ICH))
17:     K = INDEX(SMALL,ICH(I:I))
18:     if ( K.gt.0 ) ICH(I:I) = LARGE(K:K)
19: 100 continue
20:   return
21:   end
```

src/mvpburn/setmid.f

```
1:      subroutine SETMID( ID,      TEMP )
2: C
3:      character*(*) ID
4: C
5:      ITEMP      = TEMP + 0.01
6: C2003 WRITE(ID(5:8),'(I4.4)') ITEMP
7: C2003 IF(ID(4:4).EQ.' ') ID(4:4) = '0'
8: C2003 ... for new nuclide ID
9:      do 100 LL = LEN(ID), 1, -1
10:         if ( ID(LL:LL).ne.' ' ) go to 110
11:      100 continue
12:      110 continue
13:      write(ID(LL-3:LL),'(I4.4)') ITEMP
14:      if ( ID(LL:LL).eq.' ' ) ID(LL:LL) = '0'
15: C2003 ... end
16: C
17:      return
18:      end
```

src/mvpburn/smryin.f

```

1:      subroutine SMRYIN( MODE, ARRAY, MEMORY,
2:      &                  TRGNAM, IDENT, IDWORK, MAXTRG, MAXISO,
3:      &                  NOUT1, NOUT2, DIRIN, DIROUT, CASEID )
4: C=====
5: C  Input & Control routine of $SUMMARY mode.
6: C=====
7: C
8:      real ARRAY(MEMORY)
9:      character*12 TRGNAM(MAXTRG)
10: C2003
11:      character*12 IDENT(MAXISO,MAXTRG)
12: C2003
13:      character*12 IDWORK(MAXISO,MAXTRG)
14: C
15:      character*(*) DIRIN, DIROUT
16: C
17:      common /REAMIO/  NOUT6,  NDTLS
18: C
19: C --- array sizes -----
20: C
21:      include 'INC/_BURNP'
22: C
23: C -- Common data -----
24: C
25:      include 'INC/_CBURN1'
26:      include 'INC/_CBURN2'
27:      include 'INC/_CBURN3'
28: C
29:      real INSCR, INTCR
30:      common /BNREST/  RSDUMY(3),      NOWSTP, IBEND,  AKEFF(MXSTEP),
31:      &                ERRKEF(MXSTEP),  DAYS(MXSTEP),  U235F(MXSTEP),
32:      &                EXPST(MXSTEP),   CUMMWD(MXSTEP), INSCR(MXSTEP),
33:      &                INTCR(MXSTEP),   EINSR(MXSTEP), EINTCR(MXSTEP),
34:      &                FLXNRM(MXSTEP),  FACNRM(MXSTEP), FISABS(MXSTEP),
35:      &                FRTCAP(MXSTEP),  EFISAB(MXSTEP), EFRTCA(MXSTEP),
36:      &                FDECAY(MXSTEP),  CDECAY(MXSTEP), NHIST(MXSTEP),
37:      &                NBATCH(MXSTEP)
38: C2005
39:      &                ,ERRNRM(MXSTEP),  POWERW(MXSTEP), POWERE(MXSTEP),
40:      &                RMONIT(MXSTEP),  EMONIT(MXSTEP)
41: Cend
42:      common /PDSPDS/  IPDSCK
43:      integer IOTSTP(MXSTEP)
44: C
45: C -----
46: C
47: C  Variables of Fixed size
48: C
49:      character*8 MODE
50:      character*4 CASEID
51: C2005 character*4 CASREF
52: C2005 character*8 IDATE
53: C2005 character*8 STIME
54: C
55: C -----
56: C
57:      character*72 VNAME
58: C
59: C
60: C ---- variables to create inputs to modules -----
61: C
62: C2005 character*128 CWRK
63: C
64:      character*2 STEPNM
65:      external STEPNM

```

```

66: C -----
67: C
68:      NIN      = NDTLS
69:      NOUT6    = NOUT1
70: C
71: C ***** Clear common area BURN1, BURN2, BURN3 & BNREST
72: C
73:      call CLRCOM
74: C
75: C *****
76: C
77:      CASEID = ' '
78:      TITLE(1) = ' '
79:      TITLE(2) = ' '
80: C
81:      NEP      = 0
82:      ISTART   = 1
83:      ISTOP    = 0
84: C
85:      NERR     = 0
86:      IPDSCK   = 1
87:      NOTSTP   = 0
88:      do 100 I = 1,MXSTEP
89:          IOTSTP(I) = I
90:      100 continue
91: C
92: C -----
93: C  Read data in $SUMMARY block
94: C -----
95: C
96:      130 continue
97: C
98:      call FREADS( VNAME, NLEN, IEND )
99:      if ( IEND.ne.0 ) then
100:          write(NOUT1,6888)
101:          write(NOUT1,*)
102:          &          ' unexpected end of input in $SUMMARY block'
103:          write(NOUT1,*) ' probably no "$END SUMMARY" line.'
104:          stop 888
105:      end if
106: C
107: C ..... TITLES .....
108: C
109:      if ( VNAME(1:NLEN).eq.'TITLE1' ) then
110:          call CHREAD( VNAME(1:NLEN), TITLE(1), ' ', INLEN, ITERM, IEND )
111:          if ( ITERM.eq.0 ) then
112:              write(NOUT1,6888)
113:              write(NOUT1,7160) VNAME(1:NLEN)
114:              stop 888
115:          end if
116:          if ( IEND.ne.0 ) go to 250
117: C
118:      else if ( VNAME(1:NLEN).eq.'TITLE2' ) then
119:          call CHREAD( VNAME(1:NLEN), TITLE(2), ' ', INLEN, ITERM, IEND )
120:          if ( ITERM.eq.0 ) then
121:              write(NOUT1,6888)
122:              write(NOUT1,7160) VNAME(1:NLEN)
123:              stop 888
124:          end if
125:          if ( IEND.ne.0 ) go to 250
126: C
127: C ..... CASEID .....
128: C
129:      else if ( VNAME(1:NLEN).eq.'CASEID' ) then
130:          call CHREAD( VNAME(1:NLEN), CASEID, ' ', INLEN, ITERM, IEND )

```

src/mvpburn/smryin.f

```

131:         if ( ITERM.eq.0 ) then
132:             write(NOUT1,6888)
133:             write(NOUT1,7160) VNAME(1:NLEN)
134:             stop 888
135:         end if
136:         if ( IEND.ne.0 ) go to 250
137: C
138: C ..... PDS or PDSIN .....
139: C
140:         else if ( VNAME(1:NLEN).eq.'PDS' ) then
141:             call CHREAD( VNAME(1:NLEN), DIRIN, ' ', INLEN, ITERM, IEND )
142:             if ( ITERM.eq.0 ) then
143:                 write(NOUT1,6888)
144:                 write(NOUT1,7160) VNAME(1:NLEN)
145:                 stop 888
146:             end if
147:             if ( IEND.ne.0 ) go to 250
148:         else if ( VNAME(1:NLEN).eq.'PRINT' ) then
149:             call I4READ( VNAME(1:NLEN), IBC(3), NA, 1, IEND )
150:             if ( IEND.ne.0 ) go to 250
151:             IBEDIT = IBC(3)
152: C
153: C ..... PDSCHK .....
154: C
155:         else if ( VNAME(1:NLEN).eq.'PDSCHK' ) then
156:             call I4READ( VNAME(1:NLEN), IPDSCH, NA, 1, IEND )
157:             if ( IPDSCK.le.0 ) IPDSCK = 0
158:             if ( IEND.ne.0 ) go to 250
159: C
160: C ..... OUT-STEPS .....
161: C
162:         else if ( VNAME(1:NLEN).eq.'OUT-STEPS' ) then
163:             call I4READ( VNAME(1:NLEN), IOTSTP, NA, -MXSTEP, IEND )
164:             if ( IEND.ne.0 ) go to 250
165:             NOTSTP = NA
166: C
167: C ..... $END SUMMARY .....
168: C
169:         else if ( VNAME(1:NLEN).eq.'$END' ) then
170:             call RESET
171:             go to 160
172: C
173: C ..... unknown data .....
174: C
175:         else
176:             write(NOUT1,6888)
177:             write(NOUT1,7120) VNAME(1:NLEN)
178:             CALL CNTERR('FATAL')
179:             call DMREAD( VNAME(:NLEN), NA, NB, IEND )
180:         end if
181: C
182:         go to 130
183: C
184: 160 continue
185: C-----
186: C ..... CHECK INPUT OR PROCESSING ERROR .....
187: C-----
188: C
189: C     call PRNERR( NOUT1, 'AFTER DATA INPUT' )
190: C     call CHKERR( 'WARNING', KK )
191: C     if ( KK.gt.0 ) then
192: C         write(NOUT1,7125)
193: C         format(' *** some WARNING messages have been issued. ****'/
194: C             & ' I recommend you checking your input data',
195: C             & ' once more.'/' WARNING message is printed ',

```

```

196:         & 'with "!!!" symbol on the head of printed line.'/)
197:     end if
198:     call CHKERR( 'FATAL', KK )
199: C
200:     if ( KK.gt.0 ) then
201:         write(NOUT1,7145)
202: 7145     format(' ', '*** fatal ERRORS have been detected',
203:         & ' in input-phase. stop execution. ****'/
204:         & ' message for fatal ERROR is printed ',
205:         & 'with "XXX" symbol on the head of printed line.'/
206:         & ' you must check your input data', ' carefully !!')
207:     stop 888
208:     end if
209: C
210:     if ( CASEID.eq.' ' ) then
211:         write(NOUT1,6888)
212:         write(NOUT1,*)
213:         & ' no specification of CASEID ',
214:         & 'in ', mode(:iclen2(mode)), ' block.'
215:         CALL CNTERR('FATAL')
216:     end if
217: C
218:     if ( DIRIN.eq.' ' ) then
219:         write(NOUT1,6888)
220:         write(NOUT1,*)
221:         & ' no specification of input PDS ',
222:         & 'in ', mode(:iclen2(mode)), ' block.'
223:         CALL CNTERR('FATAL')
224:     end if
225:     if ( DIROUT .eq. ' ' ) then
226:         DIROUT = DIRIN
227:     end if
228: C
229:     call CHKERR( 'FATAL', KK )
230: C
231:     if ( KK.gt.0 ) then
232:         write(NOUT1,7145)
233:         stop 888
234:     end if
235: C
236: C ***** output summary *****
237: C
238:     call CPUTM2( T00 )
239:     call SUMRY0( MEMORY, NOUT1, NOUT2, T00, DIROUT, ARRAY,
240:         & TRGNAM, IDENT, MAXTRG, MAXISO,
241:         & CASEID, NOTSTP, IOTSTP )
242: C
243:     return
244: C
245: 250 write(NOUT1,6888)
246:     write(NOUT1,7180) VNAME(:NLEN)
247:     stop 999
248: C=====
249: 6888 format(// ' XXX(SMRYIN) ERROR-STOP : ' )
250: 7120 format(1X, ' unknown data name <', A,
251:     & ' >. Skipping ...')
252: 7160 format(1X, ' string data <', A,
253:     & ' > has no closing ")" ?'/1X,
254:     & 'or quotation mark is missing.')
255: 7180 format(1X, ' end of input data in reading data <', A,
256:     & ' > in $SUMMARY block.')
257: C
258:     end

```

src/mvpburn/souchk.f

```
1:      subroutine SOUCHK(LINE,JSOUOT)
2: C
3: C SOURCE-OUTPUT & NO-SOURCE-OUTPUT CHECK
4: C
5:      character*(*) line
6: C
7:      ISV = 0
8:      INO = 0
9:      IST = 1
10: 100 continue
11:      IEND = IST+12
12:      if (IEND.gt.72) go to 110
13:      ICK = index(line(IST:IEND),'SOURCE-OUTPUT')
14:      if (ICK.eq.1) then
15:          if (IST.gt.2) then
16:              if (LINE(IST-1:IST-1).ne.'-') then
17:                  ISV = IST
18:                  else
19:                      ISV = 0
20:                  end if
21:              else
22:                  ISV = IST
23:              end if
24:              IST = IEND + 1
25:          else
26:              IST = IST + 1
27:          end if
28:          go to 100
29: 110 continue
30:      IST = 1
31: 120 continue
32:      IEND = IST + 15
33:      if (IEND.gt.72) go to 130
34:      ICK = index(LINE(IST:IEND),'NO-SOURCE-OUTPUT')
35:      if (ICK.eq.1) then
36:          INO = IST
37:          IST = IEND+1
38:      else
39:          IST = IST + 1
40:      end if
41:      go to 120
42: 130 continue
43:      if (ISV.gt.0.and.INO.gt.0) then
44:          if (ISV.gt.INO) then
45:              JSOUOT = 1
46:          else
47:              JSOUOT = 0
48:          end if
49:      else if (ISV.gt.0) then
50:          JSOUOT = 1
51:      else if (INO.gt.0) then
52:          JSOUOT = 0
53:      end if
54:      return
55:      end
```


src/mvpburn/srtag2.f

```
1:      subroutine SRTAG2( NFILE, TAG,   IRC )
2: C=====
3: C  Find a record having specified TAG string in MVP/GMVP binary output.
4: C  Rewind and retry from beginning of the file if record is not found.
5: C
6: C      IRC  = 0 : RECORD WITH TAG NOT FOUND
7: C      IRC  = 1 : RECORD WITH TAG IS  FOUND
8: C=====
9:      character*(*) TAG
10:     character*32 STAG
11: C
12:     IRW      = 0
13:     IRC      = 0
14: 100 read(NFILE,end =110) STAG
15:     if ( STAG.eq.TAG ) then
16:         IRC      = 1
17:         backspace NFILE
18:         return
19:     end if
20:     go to 100
21: C
22: C ... retry if not rewound once
23: C
24: 110 if ( IRW.eq.0 ) then
25:     IRW = 1
26:     rewind(NFILE)
27:     goto 100
28: end if
29: C
30:     return
31: end
```

src/mvpburn/srtag.f

```
1:      subroutine SRTAG( NFILE, TAG,   IRC )
2: C=====
3: C   Find a record having specified TAG string in MVP/GMVP binary output.
4: C
5: C       IRC   = 0 : RECORD WITH TAG NOT FOUND
6: C       IRC   = 1 : RECORD WITH TAG IS   FOUND
7: C=====
8:      character*32 TAG
9:      character*32 STAG
10: C
11:      IRC      = 0
12: 100 read(NFILE,end =110) STAG
13:      if ( STAG.eq.TAG ) then
14:          IRC      = 1
15:          backspace NFILE
16:          return
17:      end if
18:      go to 100
19: C
20: 110 return
21:      end
```

src/mvpburn/stepnm.f

```
1:      function STEPNM(ISTEP,JPC)
2: c=====
3: c Return burn-up step number string attached to PDS member
4: c-----
5: c  istep : step #
6: c  jpc   : intermediate step of PC method if non-zero.
7: c-----
8: c   When jpc is non-zero, lowest digit is changed to 'ABCDEFGHJIJ' from
9: c   '0123456789'.
10: c=====
11:      character*2 STEPNM
12: c
13:      character*10 ALP
14:      character*2 CWRK
15: c
16:      ALP = 'ABCDEFGHJIJ'
17: c
18:      write(CWRK,'(i2)') ISTEP
19: CKSK write(CWRK,'(i2)') ISTEP - 1
20: c
21:      if ( JPC.ne.0 ) then
22: CMod      K      = MOD(ISTEP,10) + 1
23:           K      = MOD(ISTEP,10)
24:           if ( K.eq.0 ) then
25:             K = 10
26:           end if
27: CKSK      K      = MOD(ISTEP-1,10) + 1
28:           CWRK(2:2) = ALP(K:K)
29:           end if
30:           if ( CWRK(1:1).eq.' ' ) CWRK(1:1) = '0'
31: c
32:           STEPNM = CWRK
33: c
34:           return
35:       end
```

src/mvpburn/stlrd2.f

```

1:      subroutine STL2RD2(NFILE, ISTNO,  NOUT1  ,
2:      &                    NETALY, RRATE, ERATE1, ERATE2 )
3: C=====
4: C   CALLED FROM STALLY for VERSN:3.05
5: C=====
6: C   NFILE : unit of input file
7: C   ISTNO : SPECIAL TALLY ID #   (>0)
8: C=====
9:
10:      real*8  RRATE , ERATE1 ,ERATE2
11: C
12: C
13: C   ... local data ...
14: C
15:      integer INFO(10)
16: C
17:      character*6   TAG
18:      character*32  TAGS
19: C
20:      character*32  PLAB
21: C
22:      character  TLABEL*128
23: C
24:      character  RNM*128
25:      character  DIMLBL*16
26: C
27:      character  REGNAM*128
28:      character  DIMNAM*16
29: C
30: C
31: C
32:      do 1000 N = 1, NETALY
33: C
34:          TAG      = ' '
35:          write(TAG,('( '.i5)') N
36:          call CCOMP( TAG, LEN(TAG), TAG, IIF )
37:          LTAG = ICLEN(TAG)
38: C
39:          PLAB      = 'TALLY'//TAG(:LTAG)
40:          TAGS      = PLAB
41:          CALL      SRTAG( NFILE, TAGS, IRC)
42: C
43:          IF(IRC.EQ.0) THEN
44:              write(NOUT1,6888)
45:              write(NOUT1,*) ' XXX NO SPECIAL TALLY DATA !! STOP '
46:          &                  , TAGS
47:              stop 999
48:          endif
49: C
50:          read(NFILE) TAGS,NN,INFO(4),JRV
51:          read(NFILE) TAGS,L,TLABEL
52:          read(NFILE) TAGS,KK,(INFO(J),J=1,KK)
53: C
54:          write(NOUT1,*) ' INFO=',INFO
55: C
56:          if( INFO(1).eq.ISTNO ) go to 2000
57: 1000      continue
58: C
59:          write(NOUT1,6888)
60:          write(NOUT1,*) ' XXX SPECIFIED SPECIAL TALLY ID # (',ISTNO,
61:      &                  ' ) IS INCORRECT !! STOP.'
62:          write(NOUT1,*) ' PLEASE SET CORRECT MONITOR ID.'
63:          stop 999
64: C
65: C

```

```

66: 2000 continue
67: C
68: Cm      write(NOUT1,'(A,I3,A,I5,A)')
69: Cm      &      'Special tally No.', N, ' (ID=',INFO(1),')'
70: Cm      write(NOUT1,'(A,I4)') ' No of tally dimensions is ',INFO(4)
71: C
72:          LEN0 = ICLEN2(TLABEL)
73: C
74: Cm      write(NOUT1,7000) TLABEL(1:LEN0)
75: C7000      format(1X,'LABEL: <',A,'>')
76: C
77:          NSIZ      = 1
78: C
79:          do 100 J = 1, INFO(4)
80:              read(NFILE) TAGS,DIMLBL,NDIME
81: C
82:              if( DIMLBL(1:3).eq.'REG' ) then
83:                  read(NFILE) TAGS,L,RNM
84: C
85:                  else
86:                      read(NFILE) TAGS
87:                      RNM = ' '
88:                  endif
89: C
90:          NSIZ      = NSIZ*NDIME
91: C
92:          if (J.eq.1) then
93:              DIMNAM = DIMLBL
94:              REGNAM = RNM
95:          endif
96: 100      continue
97: C
98: C
99: C
100:      read(NFILE) TAGS,L,RRATE
101: C
102:      if( NSIZ.ne.L ) then
103:          write(NOUT1,*) ' XXX ',TAGS(1:10),', NSIZ(',L,') is wrong.'
104:      endif
105: C
106:      read(NFILE) TAGS,L,ERATE1
107: C
108:      if( NSIZ.ne.L ) then
109:          write(NOUT1,*) ' XXX ',TAGS(1:11),', NSIZ(',L,') is wrong.'
110:      endif
111: C
112:      if ( JRV.ne.0 ) then
113:          read(NFILE) TAGS,L,ERATE2
114: C
115:          if( NSIZ.ne.L ) then
116:              write(NOUT1,*) ' XXX ',TAGS(1:15),', NSIZ(',L,') is wrong.'
117:          endif
118: C
119:          else
120:              ERATE2 = 0.0
121:          endif
122: C
123:          LEN1 = ICLEN2(DIMNAM)
124:          LEN2 = ICLEN2(REGNAM)
125: C
126:          write(NOUT1,*) ' '
127:          write(NOUT1,*) ' << Requested Special Tally data >> '
128:          write(NOUT1,*) ' Tally ID no. is ',INFO(1)
129:          write(NOUT1,*) ' Tally Label is ',TLABEL(1:LEN0)
130:          write(NOUT1,*) ' Dimension Name is ',DIMNAM(1:LEN1)

```

src/mvpburn/stlrd2.f

```
131:      write(NOUT1,*) '      Region      Name is ',REGNAM(1:LEN2)
132:      write(NOUT1,111) ' Tally is ',RRATE
133:      write(NOUT1,112) ' Error is ',ERATE1,' % '
134:      write(NOUT1,112) ' Error is ',ERATE2,' % (real varinace)'
135:      write(NOUT1,*) ' '
136: C
137:      111 format(4X,A,1PE12.5)
138:      112 format(4X,A,F8.4,A)
139: C
140:      if ( RRATE.le.0.0 ) then
141:         write(NOUT1,6888)
142:         write(NOUT1,*) ' XXX SPECIFIED SPECIAL TALLY DATA is Zero ',
143:         & ' ' '!!! STOP.'
144:         write(NOUT1,*) ' PLEASE CHECK MVP SPECIAL TALLY INPUT !!!'
145:         stop 999
146:      endif
147: C
148:      6888 format(///' XXX(STLRD2) ERROR-STOP :' )
149: C
150:      return
151:      end
```

src/mvpburn/sumry0.f

```

1:      subroutine SUMRY0( MEMORY,NOUT1, NOUT2, T00,   DIRIN, A,
2:      &                TRGNAM, IDENT, MAXTRG, MAXISO,
3:      &                CASEID, NOTSTP, IOTSTP )
4: C=====
5:      character*(*) DIRIN
6:      character*4 CASEID
7:      real A(MEMORY)
8:      character*12 TRGNAM(MAXTRG)
9: C2003
10:     character*12 IDENT(MAXISO,MAXTRG)
11: C
12:     include 'INC/_BURNP'
13: C
14:     include 'INC/_CBURNCL'
15:     include 'INC/_CBURNCL2'
16:     include 'INC/_CBURNCL3'
17: C
18: C
19:     real INSCR, INTCR
20:     common /MEMOCK/ MEUSED
21:     common /BNREST/ RSDUMY(3),   NOWSTP, IBEND, AKEFF(MXSTEP),
22:     &      ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
23:     &      EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
24:     &      INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
25:     &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
26:     &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
27:     &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
28:     &      NBATCH(MXSTEP)
29: C2005
30:     &      ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
31:     &      RMONIT(MXSTEP), EMONIT(MXSTEP)
32: Cend
33: C
34:     integer IOTSTP(MXSTEP)
35:     character*8 MEMBER
36: C
37: C -----
38: C
39:     IWRITE = 0
40:     write(NOUT1,7000) CASEID, TITLE
41: C
42:     7000 format(/15X,'xxxxxxxxxxxxxxxxxxxxx'
43:     &          /15X,'x SUMMARY-MODE x'
44:     &          /15X,'xxxxxxxxxxxxxxxxxxxxx'
45:     &          /1X,'CASEID:',A4
46:     &          /1X,'TITLE :',A72
47:     &          /1X,'      :',A72/ )
48: C
49: C---- read PDS
50: C
51:     call RWCOM1('READ', DIRIN, CASEID )
52:     call RWCOM2('READ', DIRIN, CASEID )
53:     call RWCOM3('READ', DIRIN, CASEID )
54:
55:     MEMBER = 'OPENING'
56:     call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, A, 3, NOUT1 )
57: C
58: C *** SET VAIRABLE DIMENSION
59: C
60:     LOC01 = 1
61: C --- MTYP
62:     LOC02 = LOC01 + NMAT
63: C --- NISO
64:     LOC03 = LOC02 + NMAT
65: C --- TEMP

```

```

66:     LOC04 = LOC03 + NMAT
67: C --- VOLM
68:     LOC05 = LOC04 + NMAT
69: C --- TRGNAM
70: C2005 LOC06 = LOC05
71: C --- MATREG
72:     LOC06 = LOC05 + NMAT
73: C --- LENTRG
74:     LOC07 = LOC06 + NMAT
75: C --- MTBURN
76:     LOC08 = LOC07 + NMAT*3
77: C --- MTCARD
78:     LOC09 = LOC08 + NMAT*2
79: C --- IDENT
80:     LOC10 = LOC09
81: C --- DNINIT
82:     LOC11 = LOC10 + KNMAX*NMAT
83: C --- MATMVP
84:     LOC12 = LOC11 + NMAT
85: C --- IPBURN
86:     LOC13 = LOC12 + KNMAX*NMAT
87: C --- MVPDAT
88:     LOC14 = LOC13
89: C --- NUCLP
90:     LOC15 = LOC14 + NPAR*NTNUC
91: C --- GAM
92:     LOC16 = LOC15 + NTFISS*NTNUC
93: C --- NBIC
94:     LOC17 = LOC16 + NPAR*NTNUC
95: C --- PBIC
96: C ***** LOCATION FOR CASEBNUP *****
97:     LOC18 = LOC17 + NPAR*NTNUC
98: C --- POWRZN
99:     LOC19 = LOC18 + NMAT*NEP1
100: C --- EXPSZN
101:     LOC20 = LOC19 + NMAT*NEP1
102: C --- HMINV
103:     LOC21 = LOC20 + NMAT*NEP1
104: C --- DMWZON
105:     LOC22 = LOC21 + NMAT*NEP1
106: C --- GAMAVG
107:     LOC23 = LOC22 + NMAT*NEP1
108: C --- YLDXE5
109:     LOC24 = LOC23 + NMAT*NEP1
110: C --- YLDI35
111:     LOC25 = LOC24 + NMAT*NEP1
112: C --- YLDSM9
113:     LOC26 = LOC25 + NMAT*NEP1
114: C --- YLDPM9
115:     LOC27 = LOC26 + NMAT*NEP1
116: C --- DENTAB
117:     LOC28 = LOC27 + NTNUC*NMAT*NEP1
118: C --- RATMIC
119:     LAST = LOC28 + 2*4*NTNUC*NMAT*NEP1
120: C -----
121:     if (LAST.gt.MEUSED) MEUSED = LAST
122:     LEMORY = MEMORY - LAST + 1
123: C
124: C notstp < 0 : called from autobn.f (last of burnup mode)
125: C
126:     if ( LEMORY.lt.0 ) then
127:       if( NOTSTP.lt.0 ) then
128:         write(NOUT1,*)
129:         write(NOUT1,*) ' SUMMARY MODE WAS SKIPPED HERE BECAUSE OF ',
130:         &              ' INSUFFICIENT MEMORY'

```

src/mvpburn/sumry0.f

```

131:      write(NOUT1,*)
132:      return
133:    end if
134:    write(NOUT1,6888)
135:    write(NOUT1,*) ' memory over'
136:    write(NOUT1,*) ' requested memory size = ', LAST, ' words.'
137:    write(NOUT1,*) ' reserved memory size = ', MEMORY, ' words.'
138:    write(NOUT1,*) ' change parameter value in include file'
139:    stop 999
140:  end if
141: C
142:  if ( NMAT.gt.MAXTRG ) then
143:    write(NOUT1,6888)
144:    write(NOUT1,*) ' too many tally regions'
145:    write(NOUT1,*) ' requested number of materials = ',NMAT
146:    write(NOUT1,*) ' limit for number of tally regions',
147:  &      ' (MAXTRG)=',MAXTRG
148:    write(NOUT1,*) ' change parameter value in include file'
149:    stop 999
150:  end if
151: C
152:  if ( KNMAX.gt.MAXISO ) then
153:    write(NOUT1,6888)
154:    write(NOUT1,*) ' too many nuclides per tally region'
155:    write(NOUT1,*) ' requested number of nuclides = ',KNMAX
156:    write(NOUT1,*) ' limit for number of nuclides per ',
157:  &      'tally region(MAXISO)=',MAXISO
158:    write(NOUT1,*) ' change parameter value in include file'
159:    stop 999
160:  end if
161: CKSK--add on 21 Aug. 2002 ---
162:  if( NTNUC.gt.MXNUC ) then
163:    write(NOUT1,6888)
164:    write(NOUT1,*) ' too many depleting nuclides'
165:    write(NOUT1,*) ' requested number of nuclides = ',NTNUC
166:    write(NOUT1,*) ' limit for number of depleting nuclides',
167:  &      ' (MXNUC)=',MXNUC
168:    write(NOUT1,*) ' change parameter value in include file'
169:    stop 999
170:  end if
171: C
172:  call CLEA( A, LAST, 0.0 )
173: C
174:  call SUMRY1( NOUT1, NOUT2, LEMORY, DIRIN, CASEID, IWRITE
175:  &      A(LOC01), A(LOC02), A(LOC03), A(LOC04), TRGNAM ,
176:  &      A(LOC06), A(LOC07), A(LOC08), IDENT , A(LOC10),
177:  &      A(LOC11), A(LOC12), A(LOC14), A(LOC15),
178:  &      A(LOC16), A(LOC17), A(LOC18), A(LOC19), A(LOC20),
179:  &      A(LOC21), A(LOC22), A(LOC23), A(LOC24), A(LOC25),
180:  &      A(LOC26), A(LOC27), A(LOC28), A(LAST), A(LAST) ,
181:  &      C2005& NOTSTP , IOTSTP )
182:  &      NOTSTP , IOTSTP , A(LOC05) )
183: C
184: C --- CLOSE PDS
185: C
186:  MEMBER = 'CLOSING'
187:  call PDSIO( 'CLOSE', DIRIN, MEMBER, MEMBER, A, 0, NOUT1 )
188: C
189:  call CPUTM2( T1 )
190:  if (IBEDIT .gt. 0) then
191:    write(NOUT1,7040) T1 - T00
192:  end if
193: C
194: 6888 format(/// XXX(SUMRY0) ERROR-STOP :')
195: 7040 format(/1X,12X,'CPU ',1P,E12.5,' SEC.')

```

```

196: C
197: C *** END OF PROCESS
198: C
199:      return
200:    end

```

src/mvpburn/sumry1.f

```

1:      subroutine SUMRY1( NOUT1, NOUT2, MEMORY,DIRIN, CASEID,
2:      &                  IWRITE,MTYP, NISO, TEMP, VOLM, TRGNAM,
3:      &                  LENTRG,MTBURN,MTCARD,IDENT, DNINIT,MATMVP,
4:      &                  IPBURN,NUCLP, GAM, NBIC, PBIC, POWRZN,
5:      &                  EXPSZN,HMINV, DMWZON,GAMAVG,YLDXE5,YLDI35,
6:      &                  YLDSM9,YLDPM9,DENTAB,RATMIC,IARRAY,RARRAY ,
7: C2005&                  NOTSTP,IOTSTP
8:      &                  NOTSTP,IOTSTP,MATREG
9: C=====
10:     character*(*) DIRIN
11:     character*4 CASEID
12: C
13: C2003
14:     character*12 IDENT(KNMAX,NMAT)
15:     character*12 TRGNAM(NMAT)
16: C
17:     integer MTYP(NMAT), NISO(NMAT), LENTRG(NMAT)
18:     integer IPBURN(KNMAX,NMAT), MATMVP(NMAT)
19:     integer MTBURN(3,NMAT), MTCARD(2,NMAT)
20: C
21:     real TEMP(NMAT), VOLM(NMAT)
22:     real DNINIT(KNMAX,NMAT)
23: C
24:     integer NUCLP(NPAR,NTNUC)
25:     real GAM(NTFISS,NTNUC)
26:     integer NBIC(NPAR,NTNUC)
27:     real PBIC(NPAR,NTNUC)
28: C
29: C
30:     real POWRZN(NMAT,NEP1)
31:     real EXPSZN(NMAT,NEP1)
32:     real HMINV(NMAT,NEP1)
33:     real DMWZON(NMAT,NEP1)
34:     real YLDXE5(NMAT,NEP1)
35:     real YLDI35(NMAT,NEP1)
36:     real YLDSM9(NMAT,NEP1)
37:     real YLDPM9(NMAT,NEP1)
38:     real GAMAVG(NMAT,NEP1)
39: C
40:     real DENTAB(NTNUC,NMAT,NEP1)
41:     real RATMIC(2,4,NTNUC,NMAT,NEP1)
42:     integer IARRAY(MEMORY)
43:     real RARRAY(MEMORY)
44: C2005
45:     integer MATREG(NMAT)
46: Cend
47:     include 'INC/_BURNP'
48: C
49:     include 'INC/_CBURN1'
50:     include 'INC/_CBURN2'
51:     include 'INC/_CBURN3'
52: C
53:     real INSCR, INTCR
54:     common /MEMOCK/ MEUSED, MEMMAX
55:     common /BNREST/ RSDUMY(3), NOWSTP, IBEND, AKEFF(MXSTEP),
56:     &      ERRKEF(MXSTEP), DAYS(MXSTEP), U235F(MXSTEP),
57:     &      EXPST(MXSTEP), CUMMWD(MXSTEP), INSCR(MXSTEP),
58:     &      INTCR(MXSTEP), EINSR(MXSTEP), EINTCR(MXSTEP),
59:     &      FLXNRM(MXSTEP), FACNRM(MXSTEP), FISABS(MXSTEP),
60:     &      FRTCAP(MXSTEP), EFISAB(MXSTEP), EFRTCA(MXSTEP),
61:     &      FDECAY(MXSTEP), CDECAY(MXSTEP), NHIST(MXSTEP),
62:     &      NBATCH(MXSTEP)
63: C2005
64:     &      ,ERRNRM(MXSTEP), POWERW(MXSTEP), POWERE(MXSTEP),
65:     &      RMONIT(MXSTEP), EMONIT(MXSTEP)

```

```

66: Cend
67: C
68: C .... local variables ....
69: C
70:     character*72 MLINE
71:     character*8 MEMBER
72:     integer IOTSTP(MXSTEP)
73: C
74: C .... external function ...
75: C
76:     character*2 STEPNM
77:     external STEPNM
78: C
79: C-----
80: C
81:     MEMBER = 'OPENING'
82:     call PDSIO( 'OPEN', DIRIN, MEMBER, MEMBER, A, 1, NOUT1 )
83: C
84: C *** READ MATERIAL DATA
85: C
86:     call RWMATD( 'READ', DIRIN, CASEID, NMAT, KNMAX, NISO(1), TEMP(1),
87:     &      VOLM(1), TRGNAM(1), LENTRG(1), MTBURN(1,1), MTCARD(1,1),
88: C2005&      IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) )
89:     &      IDENT(1,1), DNINIT(1,1), MATMVP(1), IPBURN(1,1) ,
90:     &      MATREG(1) )
91: C
92: C *** READ CHAIN DATA
93: C
94: C
95:     call RWCHAN( 'READ', DIRIN, CASEID, NPAR, NTNUC, NTFISS,
96:     &      NUCLP(1,1), GAM(1,1), NBIC(1,1), PBIC(1,1) )
97: C
98: C
99: C2005 LENG4 = 5 + 20*MXSTEP
100:     LENG4 = 5 + 25*MXSTEP
101:     call CLEA( RSDUMY, LENG4, 0.0 )
102: C
103:     NOWSTP = 0
104: C
105: c notstp > 0 : output step is selected for user input
106: c      notstp+1 dummy step add for full out-put
107: c (notstp<0 : called from autobn.f : last step of burnup mode)
108: c
109: Cdel if (NOTSTP.gt.0) then
110: Cdel MSTEP = 0
111: Cdel do 102 ISTEP = 1,NOTSTP
112: Cdel MSTEP = MAX0(IOTSTP(ISTEP),MSTEP)
113: Cdel 102 continue
114: Cdel NOTSTP = NOTSTP+1
115: Cdel IOTSTP(NOTSTP) = MSTEP+1
116: Cdel end if
117: C
118:     LOP = 0
119:     IWRT = 0
120:     do 110 LOP1 = 1, NEP1
121: C
122:     if (NOTSTP.gt.0) then
123:     do 105 ISTEP = 1,NOTSTP
124:     if (LOP1.eq.IOTSTP(ISTEP)) go to 115
125: 105 continue
126: go to 110
127: end if
128: 115 continue
129: C MEMBER = CASEID// 'HT' //STEPNM(LOP,0)
130: MEMBER = CASEID// 'HT' //STEPNM(LOP1,0)

```


src/mvpburn/sumry1.f

```

131: c
132:   call PDSIO('EXIST',DIRIN, MEMBER, MEMBER, dummy, iflag, NOUT1)
133:   if ( iflag.lt.0 ) then
134:     if ( IWRT.le.0 ) write(NOUT1,6000)
135:     write(NOUT1,6100) LOP1, MEMBER
136:     IWRT = IWRT + 1
137:     go to 110
138:   end if
139:   LOP = LOP + 1
140: C
141: C2003 ... check member size
142: C
143:   call PDSIO( 'SEARCH',DIRIN, MEMBER, MEMBER, RARRAY(1), LENGHT,
144:   & NOUT1 )
145:   if (LENGHT.gt.MEMORY) then
146:     write(NOUT1,*) 'XXX(SUMRY1) Size of PDS member ', MEMBER,
147:   & ' (', LENGHT, ') exceeds buffer size ( =', MEMORY, ' )'
148:     stop 999
149: C
150:   else
151:     MTMP = MEMMAX - MEMORY + LENGHT
152:     if (MTMP.gt.MEUSED) MEUSED = MTMP
153:   end if
154: C
155: C2003 ... end
156: C
157:   call PDSIO( 'READ', DIRIN, MEMBER, MEMBER, RARRAY(1), LENGHT,
158:   & NOUT1 )
159: C
160: C ... NOWSTP is set to max step # having 'HT' member
161: C
162:   NOWSTP = LOP
163: C
164:   call PCTRW( IARRAY, MEMBER, IRET )
165: C
166:   call UNPKND( IARRAY, 'STEP#', 'I4', ISWSTP, 1, NDATA2, IRET )
167: C
168:   call PCTLB( IARRAY, MEMBER, 'HMINV', IRET )
169:   call UNPKND( IARRAY, 'HMINV', 'R4', HMINV(1,LOP), NMAT, NDATA2,
170:   & IRET )
171: C
172:   call PCTLB( IARRAY, MEMBER, 'EXPSZN', IRET )
173:   call UNPKND( IARRAY, 'EXPSZN', 'R4', EXPSZN(1,LOP), NMAT,
174:   & NDATA2, IRET )
175: C
176:   call PCTLB( IARRAY, MEMBER, 'DMWZON', IRET )
177:   call UNPKND( IARRAY, 'DMWZON', 'R4', DMWZON(1,LOP), NMAT,
178:   & NDATA2, IRET )
179: C
180:   call PCTLB( IARRAY, MEMBER, 'DNBURN', IRET )
181:   call UNPKND( IARRAY, 'DNBURN', 'R4', DENTAB(1,1,LOP), NTNUC*
182:   & NMAT, NDATA2, IRET )
183: C
184:   call PCTLB2( IARRAY, MEMBER, 'INTEGER', IRET )
185:   if ( IRET.eq.0 ) then
186:     call UNPKPT( IARRAY, 'INTEGER', 'I4', LP, NPK, IRET1 )
187:     NHIST(LOP) = IARRAY(LP)
188:     NBATCH(LOP) = IARRAY(LP+1)
189:   end if
190: C
191:   call PCTLB2( IARRAY, MEMBER, 'REAL', IRET )
192:   if ( IRET.eq.0 ) then
193:     call UNPKPT( IARRAY, 'REAL', 'R4', LP, NPK, IRET1 )
194:     AKEFF(LOP) = RARRAY(LP)
195:     ERRKEF(LOP) = RARRAY(LP+1)

```

```

196:   DAYS(LOP) = RARRAY(LP+2)
197:   U235F(LOP) = RARRAY(LP+3)
198:   EXPST(LOP) = RARRAY(LP+4)
199:   CUMMWD(LOP) = RARRAY(LP+5)
200:   INSCR(LOP) = RARRAY(LP+6)
201:   INTCR(LOP) = RARRAY(LP+7)
202:   EINSR(LOP) = RARRAY(LP+8)
203:   EINTCR(LOP) = RARRAY(LP+9)
204:   FLXNRM(LOP) = RARRAY(LP+10)
205:   FACNRM(LOP) = RARRAY(LP+11)
206:   FISABS(LOP) = RARRAY(LP+12)
207:   FRTCAP(LOP) = RARRAY(LP+13)
208:   EFISAB(LOP) = RARRAY(LP+14)
209:   EFRTCA(LOP) = RARRAY(LP+15)
210:   FDECAY(LOP) = RARRAY(LP+16)
211:   CDECAY(LOP) = RARRAY(LP+17)
212: C2005 ... added
213:   ERRNRM(LOP) = RARRAY(LP+18)
214:   POWERW(LOP) = RARRAY(LP+19)
215:   POWERE(LOP) = RARRAY(LP+20)
216:   RMONIT(LOP) = RARRAY(LP+21)
217:   EMONIT(LOP) = RARRAY(LP+22)
218: C2005 ... ended
219:   end if
220: C
221:   call PCTLB2( IARRAY, MEMBER, 'POWRZN', IRET )
222:   if ( IRET.eq.0 ) then
223:     call UNPKND( IARRAY, 'POWRZN', 'R4', POWRZN(1,LOP), NMAT,
224:   & NDATA2, IRET )
225:   else
226:     if ( IWRT.le.0 ) write(NOUT1,6000)
227:     write(NOUT1,6200) LOP, MEMBER, 'POWRZN'
228:     IWRT = IWRT + 1
229:   end if
230: C
231:   call PCTLB2( IARRAY, MEMBER, 'GMAVG', IRET )
232:   if ( IRET.eq.0 ) then
233:     call UNPKND( IARRAY, 'GMAVG', 'R4', GAMAVG(1,LOP), NMAT,
234:   & NDATA2, IRET )
235:   else
236:     if ( IWRT.le.0 ) write(NOUT1,6000)
237:     write(NOUT1,6200) LOP, MEMBER, 'GMAVG'
238:     IWRT = IWRT + 1
239:   end if
240: C
241:   call PCTLB2( IARRAY, MEMBER, 'YLDXE5', IRET )
242:   if ( IRET.eq.0 ) then
243:     call UNPKND( IARRAY, 'YLDXE5', 'R4', YLDXE5(1,LOP), NMAT,
244:   & NDATA2, IRET )
245:   else
246:     if ( IWRT.le.0 ) write(NOUT1,6000)
247:     write(NOUT1,6200) LOP, MEMBER, 'YLDXE5'
248:     IWRT = IWRT + 1
249:   end if
250: C
251:   call PCTLB2( IARRAY, MEMBER, 'YLDI35', IRET )
252:   if ( IRET.eq.0 ) then
253:     call UNPKND( IARRAY, 'YLDI35', 'R4', YLDI35(1,LOP), NMAT,
254:   & NDATA2, IRET )
255:   else
256:     if ( IWRT.le.0 ) write(NOUT1,6000)
257:     write(NOUT1,6200) LOP, MEMBER, 'YLDI35'
258:     IWRT = IWRT + 1
259:   end if
260: C

```

src/mvpburn/sumry1.f

```

261:      call PCTLB2( IARRAY, MEMBER, 'YLD9M9', IRET )
262:      if ( IRET.eq.0 ) then
263:        call UNPKND( IARRAY, 'YLD9M9', 'R4', YLD9M9(1,LOP), NMAT,
264:          &          NDATA2, IRET )
265:      else
266:        If ( IWRT.le.0) write(NOUT1,6000)
267:        write(NOUT1,6200) LOP, MEMBER, 'YLD9M9'
268:        IWRT = IWRT + 1
269:      end if
270: C
271:      call PCTLB2( IARRAY, MEMBER, 'YLDPM9', IRET )
272:      if ( IRET.eq.0 ) then
273:        call UNPKND( IARRAY, 'YLDPM9', 'R4', YLDPM9(1,LOP), NMAT,
274:          &          NDATA2, IRET )
275:      else
276:        If ( IWRT.le.0) write(NOUT1,6000)
277:        write(NOUT1,6200) LOP, MEMBER, 'YLDPM9'
278:        IWRT = IWRT + 1
279:      end if
280: C
281:      call PCTLB2( IARRAY, MEMBER, 'RATMIC', IRET )
282:      if ( IRET.eq.0 ) then
283:        do 100 M = 1, NMAT
284:          call UNPKND( IARRAY, 'RATMIC', 'R4', RATMIC(1,1,1,M,LOP),
285:            &          2*4*NTNUC, NDATA2, IRET )
286:        100 continue
287:      else
288:        If ( IWRT.le.0) write(NOUT1,6000)
289:        write(NOUT1,6200) LOP, MEMBER, 'RATMIC'
290:        IWRT = IWRT + 1
291:      end if
292: C
293:      110 continue
294: C
295:      do 120 M = 1, NMAT
296:        MTYP(M) = MTBURN(3,M)
297:      120 continue
298: C
299: C **** Copy original MVP input data to I/O unit IMVPIN
300: C
301:      IMVPIN = 93
302:      MEMBER = CASEID// 'MVPI'
303: C
304:      IMVPI = 61
305: C2003 ... check existence of caseMVPI by SEARCH
306:      call PDSIO( 'EXIST',DIRIN, MEMBER, MEMBER, DUMY, IDATA, NOUT1 )
307: C2003 call PDSFIL( 'READ', DIRIN, MEMBER, 'TEXT', IMVPI, NOUT1, IIERR )
308: C
309: C2003 if ( IIERR.ne.0 ) then
310: C
311:      if ( IDATA.lt.0 ) then
312:        write(NOUT1,*) '!!!(SUMRY1) failed to open member <', MEMBER,
313:          &          '>'
314:        IMVPIN = -1
315:      else
316: C
317:        call PDSFIL( 'READ', DIRIN, MEMBER, 'TEXT', IMVPI, NOUT1, IIERR )
318:        call RWIND( IMVPI )
319:        call RWIND( IMVPIN )
320:        do 130 I = 1, NCARD
321:          read(IMVPI, '(a)') MLINE
322:          write(IMVPIN, '(A)') MLINE(:ICLEN2(MLINE))
323:      130 continue
324:        call RWIND( IMVPIN )
325: C

```

```

326:      call PDSFIL( 'CLOSE', DIRIN, MEMBER, 'TEXT', IMVPI, NOUT1,
327:        &          IIERR )
328:      end if
329: C
330: C
331: C
332: C
333: C2003 .... add DENERR and flag JDENER are added (not used)
334: C      JDENER = 0 : not print density errors
335: C      DENERR : dummy array here
336: C
337: C      JDENER = 0
338: C
339: C      call BURN9M( LENG9M, MEMORY, NOUT1, NOUT2, CASEID, MTYP,
340:        &          IMVPIN, TEMP, VOLM, TRGNAM, MTBURN, POWRZN, EXPSZN, HMINV,
341:        &          DMWZON, GAMAVG, YLDXE5, YLDI35, YLD9M9, YLDPM9, DENTAB,
342:        &          DENERR, JDENER,
343:        &          RATMIC, GAM, IARRAY, RARRAY, IOTSTP, NOTSTP )
344: C
345: C2003 .... end
346: C
347: C *** END OF PROCESS
348: C
349: 6000 format(/1X,9('=',) ' INFORMATION ON MEMBERS IN PDS ',9('='))
350: 6100 format(1X,'STEP',I3,' : MEMBER <',A,'> DOSE NOT EXIST IN PDS')
351: 6200 format(1X,'STEP',I3,' : MEMBER <',A,'> LACKS DATA <',A,'>')
352: C
353:      return
354:      end

```

src/mvpburn/udate.f

```
1:      subroutine UDATE( ADATE )
2: C
3: C=====
4: C      DATE IS FACOM BUILTIN SERVICE ROUTINE TO RETURN CURRENT DATE
5: C      ADATE(OUTPUT) : CURRENT DATE IN 8-BYTE
6: C                      EX. '95-02-09' IF WRITE ADATE IN A8 FORMAT
7: C=====
8:      character*8 ADATE
9:      character*12 XDATE
10: C
11:      character*3 MONTH(12)
12:      data MONTH /'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
13:      &          'Aug', 'Sep', 'Oct', 'Nov', 'Dec'/
14: C
15: C
16:      XDATE = ' '
17:      call HIZUKE( XDATE )
18: C
19: C convert DD-MMM-YY or DD-MMM-YYYY
20: C      to YY-MM-DD      ... aha ... ;-)
21: C
22:      I1      = INDEX(XDATE,'-')
23:      I2      = I1 + INDEX(XDATE(I1+1:),'-')
24:      do 100 LL = LEN(XDATE), 1, -1
25:      if ( XDATE(LL:LL).ne.' ') go to 110
26: 100 continue
27: 110 continue
28: C
29: C ... Year (only two digit)
30: C
31:      ADATE(1:3) = XDATE(LL-1:LL) //'-'
32:      ADATE(4:5) = ' '
33: C
34: C ... Month
35: C
36:      do 120 IM = 1, 12
37:      if ( XDATE(I1+1:I2-1).eq.MONTH(IM) ) then
38:      if ( IM.ge.10 ) then
39:      write(ADATE(4:5),'(i2)') IM
40:      else
41:      write(ADATE(4:5),'(''0'',i1)') IM
42:      end if
43:      go to 130
44:      end if
45: 120 continue
46: 130 continue
47: C
48: C ... Day
49: C
50:      if ( XDATE(1:1).eq.' ' ) XDATE(1:1) = '0'
51:      ADATE(6:8) = '-'//XDATE(1:2)
52: C
53:      return
54:      end
```

src/mvpburn/wdbl.f

```
1:      subroutine WDBL
2: C=====
3: C purpose: define constant MWORD which shows size of double precision
4: C      data (real*8) in word of the architecture.
5: C update:
6: C 30 Oct 1997: changed to subroutine from block data because
7: C      block data information may be lost if linked as lib*.a archive.
8: C=====
9:      include 'INC/_WORDL'
10: C
11: C/#IF WORD(64)
12: C ... 64 BIT MACHINES (CRAY or other pure 64 bit architecture)
13: *      MWORD = 1
14: C/#ELSE
15: C ... 32 BIT MACHINES (VAX SX ETC)
16:      MWORD = 2
17: C/#ENDIF
18:      end
```

src/mvpburn/zzmmm.f

```

1: C-----1-----2-----3-----4-----5-----6-----7--
2: C   MAIN PROGRAM TO TEST SUBROUTINE ZZMMM (update 11,JLY,2005)
3: C-----1-----2-----3-----4-----5-----6-----7--
4: *   INTEGER IC,NUM(10)
5: *   CHARACTER*6 SYMB(10)
6: C
7: *   NISO = 6
8: *   IC=0
9: *   SYMB(1) = 'D0002'
10: *   SYMB(2) = 'T-003'
11: *   SYMB(3) = 'C-000'
12: *   SYMB(4) = 'XE135'
13: *   SYMB(5) = 'AM241M'
14: *   SYMB(6) = 'ZZ115'
15: *   call ZZMMM(IC,NISO,NUM,SYMB)
16: *   do I=1,NISO
17: *     write(6,*) 'No. SYMB NUM =', i, ' ',SYMB(i), NUM(i)
18: *   end do
19: C
20: *   IC = 1
21: *   call ZZMMM(IC,NISO,NUM,SYMB)
22: *   write(6,*) '***** IC=+1 *****'
23: *   write(6,600) (SYMB(i),i=1,NISO)
24: * 600 format(10(A6,X))
25: C
26: *   IC = -1
27: *   call ZZMMM(IC,NISO,NUM,SYMB)
28: *   write(6,*) '***** IC=-1 *****'
29: *   write(6,600) (SYMB(i),i=1,NISO)
30: C
31: *   IC = 2
32: *   call ZZMMM(IC,NISO,NUM,SYMB)
33: *   write(6,*) '***** IC=+2 *****'
34: *   write(6,600) (SYMB(i),i=1,NISO)
35: C
36: *   IC = -2
37: *   call ZZMMM(IC,NISO,NUM,SYMB)
38: *   write(6,*) '***** IC=-2 *****'
39: *   write(6,600) (SYMB(i),i=1,NISO)
40: C
41: *   stop
42: *   end
43: C*****
44: SUBROUTINE ZZMMM(IC,NISO,NUM,SYMB)
45: C*****
46: C Conversion between Nuclide symbol(A6) and nuclide number(7 digits)
47: C by Keisuke OKUMURA (JAEA) Last modified : 21 Nov 2006
48: C*****
49: C
50: C ZZMMML <=> zz*10000+mmm*10+1
51: C e.g. U-235 : 922350
52: C AM242M : 952421
53: C AM242L : 952421
54: C H-001 : 10010
55: C D0000 : 10020
56: C C-000 : 60000 (Nat.)
57: C MT266M :1092661
58: C ZZ1010 : 1010 (one of a Psudo F.P. for fast reactor)
59: C
60: C IC : CONTROL INTEGER (INPUT)
61: C IC = 0 : from Symbol to Number
62: C "U0" or "U-" are treated as the same input
63: C An output symbol is not changed
64: C Different symbols of Hydrogen(H,D,T) are acceptable
65: C IC != 0 : from Number to Symbol

```

```

66: C = 1 : Symbol "0" is used for a blank character (U0235)
67: C --1 : ibid but Different symbols(H/D/T) of Hydrogen are
68: C used according to mass number (D002 in SRAC)
69: C = 2 : Symbol "-" is used for a blank character (U-235)
70: C --2 : ibid but Symbols(H/D/T) of Hydrogen are used
71: C Memo : IC=-1(for SRAC) and IC=1(for MVP-BURN)
72: C
73: C NISO: NUMBER OF NUCLIDES
74: C
75: C SYMB(NISO):NUCLIDE SYMBOL (INPUT when IC=0, OUTPUT when IC !=0)
76: C SYMBOL = ZZ//MMM//L
77: C ZZ : Chemical symbol (U-, H0, PU,...)
78: C MMM: Mass number (001-999)
79: C L : '=' (ground state), ='M'(excited state)
80: C Exceptionals:
81: C 1) Pseudo F.P. nuclides (ZZsnn0)
82: C ZZ : Pseudo FP
83: C s : =0 for thermal reactor / =1 for fast reactor
84: C nn : ID number for Pseudo types (chain model types)
85: C
86: C NUM(NISO) :NUCLIDE NUMBERS (OUTPUT when IC=0, INPUT when IC!=0)
87: C zzzmmml ( zz*10000 + mmm*10 + 1 )
88: C zz : Atomic number
89: C mmm: Mass number
90: C 1 : =0(ground sate), =1 (excited state)
91: C Exceptionals: from 0 to 9999
92: C 1) Pseudo FP (from 1 to 1999)
93: C zz=0
94: C mmm = snn (ID number of Pseudo types)
95: C 10 - 990 : Pseudo for thermal reactors
96: C 1010 - 1990 : Pseudo for fast reactors
97: C
98: C*****
99: C
100: C CHARACTER*6 SYMB(*)
101: C INTEGER I,NUM(*),N1,N2,N3
102: C CHARACTER*2 S1
103: C CHARACTER*3 S2
104: C CHARACTER*1 S3
105: C CHARACTER*2 ZSYMB(110)
106: C DATA ZSYMB/'H-','HE','LI','BE','B-','C-','N-','O-','F-','NE',
107: C 1 'NA','MG','AL','SI','P-','S-','CL','AR','K-','CA',
108: C 2 'SC','TI','V-','CR','MN','FE','CO','NI','CU','ZN',
109: C 3 'GA','GE','AS','SE','BR','KR','RB','SR','Y-','ZR',
110: C 4 'NB','MO','TC','RU','RH','PD','AG','CD','IN','SN',
111: C 5 'SB','TE','I-','XE','CS','BA','LA','CE','PR','ND',
112: C 6 'PM','SM','EU','GD','TB','DY','HO','ER','TM','YB',
113: C 7 'LU','HF','TA','W-','RE','OS','IR','PT','AU','HG',
114: C 8 'TL','PB','BI','PO','AT','RN','FR','RA','AC','TH',
115: C 9 'PA','U-','NP','PU','AM','CM','BK','CF','ES','FM',
116: C & 'MD','NO','LR','RF','DB','SG','BH','HS','MT','ZZ'/
117: C
118: C Note : ZSYMB(110)=ZZ : Symbol for Pseudo Fission Products
119: C
120: C=====
121: C
122: C MAXZ = 110
123: C IF (IC.NE.0) GOTO 2000
124: C
125: C-----IC=0 : SYMBOL => NUMBER -----
126: C
127: C DO 1100 J=1,NISO
128: C S1=SYMB(J)(1:2)
129: C S2=SYMB(J)(3:5)
130: C S3=SYMB(J)(6:)

```

src/mvpburn/zmmmm.f

```

131:      IF(S1(2:2).EQ.'0') S1(2:2)='- '
132:      IF(S1.EQ.'D-' .OR. S1.EQ.'T-' ) THEN
133:          S1 = 'H-'
134:      ENDIF
135:      N1 = 0
136:      DO 100 I=1,MAXZ
137:          IF (S1.EQ.ZSYMB(I)) N1=I
138: 100    CONTINUE
139:      IF(N1.EQ.0) THEN
140:          WRITE(6,6000) SYMB(J)
141:          STOP 999
142:      ENDIF
143:      IF(N1.EQ.110) N1=0
144: C
145:      READ(S2,'(I3)') N2
146: C2003  IF (S3.EQ.' ') THEN
147:      IF (S3.EQ.' ' .or. S3.EQ.'0' ) THEN
148:          N3=0
149: C2005  ELSEIF (S3.EQ.'M') THEN
150:      ELSEIF (S3.EQ.'M' .or. S3.EQ.'m' .or. S3.EQ.'1') THEN
151:          N3=1
152:      ELSE
153: C2003  WRITE(6,6000) SYMB(J)
154: C2003  STOP 999
155:          NUM(J) = -1
156:          return
157: C2003  ... end
158:      ENDIF
159:      NUM(J) = N1*10000+N2*10+N3
160: 1100  CONTINUE
161:      GOTO 9999
162: C
163: C6000  FORMAT(/1X,'<<<  ERROR STOP (ZZZMM)  >>>' /1X,
164: 6000  FORMAT(/1X,'XXX(ZZMMM):',
165:      &'Input nuclide symbol(' ,A,') can not be identified')
166: C
167: C-----IC!=0 : NUMBER => SYMBOL -----
168: C
169: 2000  CONTINUE
170:      IF (IABS(IC).GT.2) THEN
171:          WRITE(6,6100) IC
172:          STOP 999
173:      ENDIF
174: C
175:      DO 2100 J=1,NISO
176:          N1=INT(NUM(J)/10000)
177:          N2=INT((NUM(J)-N1*10000)/10)
178:          N3=MOD(NUM(J),10)
179: C
180:          IF (N1.GT.0 .AND. N1.LT.MAXZ) THEN
181:              S1=ZSYMB(N1)
182:          ELSEIF (N1.EQ.0) THEN
183:              S1=ZSYMB(110)
184:          ELSE
185:              WRITE(6,6200) NUM(J)
186:              STOP 999
187:          ENDIF
188:          IF(IABS(IC).EQ.1 .AND. S1(2:2).EQ.'-' ) S1(2:2)='0'
189: C
190:          WRITE(S2,'(I3.3)') N2
191:          IF(IC.LT.0 .AND. N2.EQ.2) S1(1:1)='D'
192:          IF(IC.LT.0 .and. N1.EQ.1 .and. N2.EQ.3) S1(1:1)='T'
193: C
194:          IF (N3.EQ.0) THEN
195:              S3=' '
196:          ELSEIF (N3.EQ.1) THEN
197:              S3='M'
198:          ELSE
199:              WRITE(6,6200) NUM(J)
200:              STOP 999
201:          ENDIF
202:          SYMB(J)=S1//S2//S3
203: C
204: 2100  continue
205: C
206: C6100  FORMAT(/1X,'<<<  ERROR STOP (ZZZMM)  >>>' /1X,
207: 6100  FORMAT(/1X,'XXX(ZZMMM):',
208:      &'Input control integer (' ,I10,') is invalid')
209: C6200  FORMAT(/1X,'<<<  ERROR STOP (ZZZMM)  >>>' /1X,
210: 6200  FORMAT(/1X,'XXX(ZZMMM):',
211:      &'Input nuclide number(' ,I10,') can not be identified')
212: C
213: 9999  RETURN
214:      END

```

src/mvpburn/INC/_AVERG

```
1: C
2: C this file is INC/_AVERG
3: C
4: C     MAXCAS : limit for number of cases in averaging mode
5: C
6:     PARAMETER ( MAXCAS = 50 )
```

SAFE

src/mvpburn/INC/_BURNP

```
1: C
2: C this file is INC/_BURNP
3: C
4: C     MXSTEP : limit for number of burnnup steps ( < 100 )
5: C     MXNUC  : limit for number of depleting nuclides in material
6: C     MXFISS : limit for number of fissile nuclides
7: C
8: CKSK  PARAMETER ( MXSTEP = 99 , MXNUC  = 100 , MXFISS = 30 )
9:        PARAMETER ( MXSTEP = 99 , MXNUC  = 160 , MXFISS = 30 )
```

SAFE

src/mvpburn/INC/_CBURN1

```
1:      common /BURN1/  IBC(20),      NEP,      NEP1,  LENG2,  LENG3,  
2:      &      LENG4,      PERIOD(MXSTEP), POWERL(MXSTEP), NSTP(MXSTEP)  
3: C2005  
4:      &      ,EXTSRC(MXSTEP), IDMONI , VMONIT(MXSTEP),  REGPOW(MXSTEP)
```

SAFE

src/mvpburn/INC/_CBURNC2

```
1: C
2: C2003 ... add NCODE and STDNUC,NAMFIS, NAMFER
3:   integer STDNUC
4:   common /BURNC2/ BNDUMY(5),      NCARD,  NMAT,   KNMAX,  METHPC,
5:   &      IVOID,  ST235,  TWTHVY, EVTOJ,  ABOGA,  BZERO,  LASTFP,
6:   &      IDU235, IBUNIT, IBEDIT, NTNUC,   NTFISS, NPAR,   NCHA,
7:   &      LCHA,   IDXE35, IDI135, IDSM49, IDPM49, NFIS,   NFER,
8:   &      IFISDN(MXFISS), FISFCT(MXFISS), IFRTDN(MXFISS),
9:   &      FRTFCT(MXFISS), REACEV(2,MXNUC),      IFISS(MXNUC),
10:  &      AMASS(MXNUC),   NUCL(MXNUC),   FACT2N(MXNUC),
11:  &      NCH(MXNUC),     RAMDA(MXNUC),  NCODE(MXNUC),
12:  &      STDNUC,  NAMFIS(MXFISS),  NAMFER(MXFISS)
13: C2005
14:  &      ,NCXE35, NCI135, NCSM49,  NCPM49
```

src/mvpburn/INC/_CBURN3

```
1: C2003 character*4 NAMFIS, NAMFER, CASBRN, SRACID, STDNUC, CASPCM
2: C2005 character*4 CASBRN, SRACID, CASPCM
3:   character*4 CASBRN, CASPCM
4:   character*72 TITLE
5: C2003 ... add IHOL and move NAMFIS,NAMFER and STDNUC
6:   character*8 IHOL
7: C2003 common /BURNC3/  NAMFIS(MXFISS), NAMFER(MXFISS), SRACID(MXNUC),
8: C2003&          STDNUC,  CASBRN, CASPCM, TITLE(2), IHOL(MXNUC)
9: C2005 common /BURNC3/  SRACID(MXNUC), IHOL(MXNUC), CASBRN, CASPCM,
10: C2005&          TITLE(2)
11:   common /BURNC3/  IHOL(MXNUC), CASBRN, CASPCM, TITLE(2)
```

SAFE

src/mvpburn/INC/_IOUNIT

```
1: C
2: C This file is INC/_IOUNIT
3: C
4: C*
5: C* I/O unit parameters used in PACKET routines in MVP-BURN:
6: C*
7: C* This is a simplified version of src/shared/INC/_IOUNIT for MVP/GMVP
8: C*
9: C* IPR : standard output (printout)
10: C*
11:     PARAMETER ( IPR = 6 )
```

SAE

src/mvpburn/INC/_MEMORY

```
1: C
2: C this file is INC/_MEMORY
3: C
4: C MAXMEM : limit of memory size for variable dimension of BURN part)
5: C
6: CKSK  PARAMETER ( MAXMEM   = 10000000 )
7: CKSK  PARAMETER ( MAXMEM   = 80000000 )    ! for VPP5000 in JAERI !
8:      PARAMETER ( MAXMEM   = 100000000 )
```

JAERI

src/mvpburn/INC/_MVPBRN

```
1: C
2: C This file is INC/_MVPBRN
3: C
4: C MAXTRG : limit for number of tally regions (materials)
5: C MAXISO : limit for number of nuclides in a tally region (material)
6: C MAXSPR : limit for number of step-dependent MVP symbolic parameters
7: C
8: CKSK parameter( MAXTRG = 200, MAXISO=80 , MAXSPR = 8 )
9: CKSK parameter( MAXTRG = 200, MAXISO=160 , MAXSPR = 8 )
10: parameter( MAXTRG = 1000, MAXISO=160 , MAXSPR = 8 )
```

SAFE

src/mvpburn/INC/_WORDL

```
1: C
2: C This file is INC/_WORDL
3: C
4: C*
5: C*--- WORD LENGTH PARAMETER ( MWORD = 2 / 1 ... VP,SX/CRAY )
6: C*
7: C* (WORD LENGTH OF 8 BYTE REAL OR 'CHARACTER*8' VARIABLES)
8: C* SEE BLOCKDATA WDBL
9: C*
10: COMMON /WORDL/ MWORD
11: C*
12: C MWORD 14 Word length for 8-byte floating point number on users$
13: C $ machine.
14: C*
15: C =2 : 32 bit machines.
16: C =1 : 64 bit machines such as CRAY.
17: C*
```

src/utils/adtoidx.c

```

1: /*****
2: *
3: *   Memory allocation routine for FORTRAN compilers not supporting
4: *   pointer handling.
5: *
6: *   Using malloc() function which is defined in both POSIX & ANSI-C.
7: *
8: *       3 Feb 1995 M.Sasaki
9: *       25 Sep 1997 M.Sasaki
10: *       15 Sep 1998 M.Sasaki
11: *
12: *   $Id: adtoidx.c,v 1.2 1998/10/08 08:31:26 sasaki Exp $
13: *
14: *****/
15: /*****
16: *   Fortran called C subroutine :
17: *
18: *   UNIX vendors have their own convention or rule for external
19: *   names of FORTRAN coded subprograms or commons.
20: *
21: *   We prepare equivalent sources for C coded routines
22: *
23: *       1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
24: *       2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
25: *       3. Uppercase string of FORTRAN name. ( Ncube etc. )
26: *       4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
27: *
28: *****/
29:
30: #include <stdio.h>
31: #include <stdlib.h>
32:
33: void address_to_index( ) ;
34:
35: /*****
36: *   Usage from Fortran program :
37: *
38: *       common /array/ ii(1)
39: *
40: *       nsize = 1000000
41: *       call adtoidx( ii(1), nsize, lstart, jerr )
42: *
43: *       ! LSTART is starting array index of allocated memory.
44: *
45: *       if( jerr.eq. 0 ) then
46: *           call sub( ii(lstart), nsize )
47: *       else
48: *           write(*,*) 'memory allocation failed !'
49: *           stop
50: *       endif
51: *
52: *****/
53: void adtoidx ( ii, nii, lii, lerr )
54: int * ii ; /** starting address of buffer array **/
55: int * nii ; /** memory size allocated as "int" type (word) **/
56: int * lii ; /** starting "array index" of allocated area **/
57: int * lerr ; /** error flag. failed to allocate if non-zero***/
58: {
59:     address_to_index( ii, nii, lii, lerr ) ;
60: }
61: void adtoidx_ ( ii, nii, lii, lerr )
62: int * ii ; /** starting address of buffer array **/
63: int * nii ; /** memory size allocated as "int" type (word) **/
64: int * lii ; /** starting "array index" of allocated area **/
65: int * lerr ; /** error flag. failed to allocate if non-zero***/

```

```

66: {
67:     address_to_index( ii, nii, lii, lerr ) ;
68: }
69: void ADTOIDX ( ii, nii, lii, lerr )
70: int * ii ; /** starting address of buffer array **/
71: int * nii ; /** memory size allocated as "int" type (word) **/
72: int * lii ; /** starting "array index" of allocated area **/
73: int * lerr ; /** error flag. failed to allocate if non-zero***/
74: {
75:     address_to_index( ii, nii, lii, lerr ) ;
76: }
77: void _ADTOIDX ( ii, nii, lii, lerr )
78: int * ii ; /** starting address of buffer array **/
79: int * nii ; /** memory size allocated as "int" type (word) **/
80: int * lii ; /** starting "array index" of allocated area **/
81: int * lerr ; /** error flag. failed to allocate if non-zero***/
82: {
83:     address_to_index( ii, nii, lii, lerr ) ;
84: }
85: void _adtoidx ( ii, nii, lii, lerr )
86: int * ii ; /** starting address of buffer array **/
87: int * nii ; /** memory size allocated as "int" type (word) **/
88: int * lii ; /** starting "array index" of allocated area **/
89: int * lerr ; /** error flag. failed to allocate if non-zero***/
90: {
91:     address_to_index( ii, nii, lii, lerr ) ;
92: }
93:
94:
95: void address_to_index( ii, nii, lii, lerr )
96: int * ii ; /** starting address of buffer array **/
97: int * nii ; /** memory size allocated as "int" type (word) **/
98: int * lii ; /** starting "array index" of allocated area **/
99: int * lerr ; /** error flag. failed to allocate if non-zero***/
100: {
101:     int * pii ;
102:
103:     *lerr = 0 ;
104:
105:     pii = malloc( *nii * sizeof(int) ) ;
106:     if ( pii == 0 )
107:     {
108:         printf("XXX Failed to allocate required memory.\n") ;
109:         *lii = 0 ;
110:         *lerr = 1 ;
111:     }
112:     else
113:     {
114:         /*****
115:         printf(
116:             "    == Absolute address of allocated memory: %x\n",
117:             pii ) ;
118:         *****/
119:
120:         *lii = ( pii - ii ) + 1 ;
121:     }
122: }

```


src/utils/cctimdat.c

```

1:
2: /*****
3:  * Fortran callable C subroutine :
4:  *
5:  *   UNIX venders have their own convention or rule for external
6:  *   names of FORTRAN coded subprograms or commons.
7:  *
8:  *   We prepare equivalent sources for C coded routines
9:  *
10:  *   1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
11:  *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
12:  *   3. Uppercase string of FORTRAN name. ( Ncube etc. )
13:  *   4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
14:  *
15:  *****/
16:
17: /*****
18:  *
19:  *   Get local time and day and pass them to Fortran program
20:  *   ( ANSI C & POSIX )
21:  *
22:  *   16 Sep 1998: add 1900 to year.
23:  *
24:  *****/
25:
26: #include <stdio.h>
27: #include <sys/types.h>
28: #include <time.h>
29:
30: void cc_time_date();
31:
32: void cctimdat( i_hour, i_minute, i_sec, i_day, i_month, i_year )
33: {
34:     int * i_hour ;          /* hour */
35:     int * i_minute ;        /* minute */
36:     int * i_sec ;           /* second */
37:     int * i_day ;           /* hour */
38:     int * i_month ;         /* month */
39:     int * i_year ;          /* year */
40:
41:     {
42:         cc_time_date(i_hour, i_minute, i_sec, i_day, i_month, i_year );
43:     }
44:
45: void cctimdat_( i_hour, i_minute, i_sec, i_day, i_month, i_year )
46: {
47:     int * i_hour ;          /* hour */
48:     int * i_minute ;        /* minute */
49:     int * i_sec ;           /* second */
50:     int * i_day ;           /* hour */
51:     int * i_month ;         /* month */
52:     int * i_year ;          /* year */
53:
54:     {
55:         cc_time_date(i_hour, i_minute, i_sec, i_day, i_month, i_year );
56:     }
57:
58: void CCTIMDAT( i_hour, i_minute, i_sec, i_day, i_month, i_year )
59: {
60:     int * i_hour ;          /* hour */
61:     int * i_minute ;        /* minute */
62:     int * i_sec ;           /* second */
63:
64:     {
65:         cc_time_date(i_hour, i_minute, i_sec, i_day, i_month, i_year );
66:     }
67:
68: void _CCTIMDAT( i_hour, i_minute, i_sec, i_day, i_month, i_year )
69: {
70:     int * i_hour ;          /* hour */
71:     int * i_minute ;        /* minute */
72:     int * i_sec ;           /* second */

```

```

73: int * i_day ;              /* hour */
74: int * i_month ;            /* month */
75: int * i_year ;             /* year */
76: {
77:     {
78:         cc_time_date(i_hour, i_minute, i_sec, i_day, i_month, i_year );
79:     }
80:
81: void cc_time_date( i_hour, i_minute, i_sec, i_day, i_month, i_year )
82: {
83:     int * i_hour ;          /* hour */
84:     int * i_minute ;        /* minute */
85:     int * i_sec ;           /* second */
86:     int * i_day ;           /* hour */
87:     int * i_month ;         /* month */
88:     int * i_year ;          /* year */
89:
90:     {
91:         time_t th ;
92:         struct tm *lt ;
93:
94:         if( -1 != time( &th ) )
95:         {
96:             lt = localtime( &th ) ;
97:
98:             *i_hour      = lt->tm_hour ;
99:             *i_minute    = lt->tm_min ;
100:             *i_sec       = lt->tm_sec ;
101:             *i_day       = lt->tm_mday ;
102:             *i_month     = lt->tm_mon + 1 ;
103:             *i_year      = lt->tm_year + 1900 ;
104:         }
105:     }
106:
107: else
108: {
109:     *i_hour      = 0 ;
110:     *i_minute    = 0 ;
111:     *i_sec       = 0 ;
112:     *i_day       = 0 ;
113:     *i_month     = 0 ;
114:     *i_year      = 0 ;
115: }
116:
117: /**
118: main() {
119:     int i_hour, i_minute, i_sec, i_day, i_month, i_year ;
120:     cc_time_date(&i_hour, &i_minute, &i_sec, &i_day, &i_month, &i_year ) ;
121:     printf(" %d %d %d %d %d %d\n",
122:         i_hour, i_minute, i_sec, i_day, i_month, i_year ) ;
123: }
124:
125: */

```

src/utls/cdummyf.f

```
1: C
2: C==<MVP/GMVP>=====
3: C
4: C Fortran dummy routines for C-coded functions
5: C
6: C     Most of these functions/routines should never be called in
7: C     MVP/GMVP Fortran source.
8: C     These are for convenience of installation on systems having
9: C     no C-language compiler.
10: C
11: C << C-coded functions >> ( Sep 1998 )
12: C
13: C adtoidx( ii, nii, lii, lerr )
14: C
15: C     Allocate memory and calculate offset from Fortran variable.
16: C
17: C cctimdat( i_hour, i_minute, i_sec, i_day, i_month, i_year )
18: C
19: C     Get time and date.
20: C
21: C cputime(ttt):
22: C cputime2(ttt):
23: C
24: C     Get Process CPU time (double)
25: C
26: C wallclock( ttt )
27: C
28: C     Get wallclock time (double)
29: C
30: C fsigset():
31: C fsigusr():
32: C fgetusrsig( signum, flag )
33: C
34: C     Signal handler setting.
35: C
36: C flushstd():
37: C
38: C     Flush standard output stream.
39: C
40: C ftgetenv( name,[namlen,] value [,vallen] )
41: C
42: C     Get environment variable.
43: C
44: C ftsystem( code, command, [length] )
45: C
46: C     Execute command.
47: C
48: C funbuf()
49: C
50: C     Set standard output stream to line buffered mode.
51: C
52: C function lfmalloc ( nbytes )
53: C
54: C     Works like "malloc"
55: C -----
56: C history: Programmed by M.Sasaki (Sep 1998)
57: C=====
58: C
59: C/#IF NOCC
60: C
61: C.....
62: C
63: *     subroutine adtoidx( ii, nii, lii, lerr )
64: C
65: C     Allocate memory and calculate offset from Fortran variable.
```

```
66: C
67: *     write(*,*) 'XXX Dummy routine for ADTIDX is called. XXXXXXXX'
68: *     write(*,*) 'XXX But it must not be called as dummy routine!!'
69: *     write(*,*) 'XXX Compilation setting must be modified.'
70: *     stop
71: *     end
72: C
73: C.....
74: C
75: *     subroutine cctimdat( ihour,iminute,isec,iday,imonth,iyear )
76: C
77: C     Get time and date.
78: C
79: *     ihour = 0
80: *     iminute = 0
81: *     isec = 0
82: *     iday = 0
83: *     imonth = 0
84: *     iyear = 0
85: *     return
86: *     end
87: C
88: C.....
89: C
90: C
91: C     Get Process CPU time (double)
92: C
93: *     subroutine cputime(ttt)
94: *     real*8 ttt
95: *     ttt = 0.0d0
96: *     return
97: *     end
98: C
99: C.....
100: C
101: *     subroutine cputime2(ttt)
102: *     real*8 ttt
103: *     ttt = 0.0d0
104: *     return
105: *     end
106: C
107: C.....
108: C
109: *     subroutine wallclock( ttt )
110: C
111: C     Get wallclock time (double)
112: C
113: *     real*8 ttt
114: *     ttt = 0.0d0
115: *     return
116: *     end
117: C
118: C.....
119: C
120: C     Signal handler setting.
121: C
122: *     subroutine fsigset()
123: *     write(*,*) '<< Dummy routine for FSIGSET is called. Do nothing>>'
124: *     return
125: *     end
126: C
127: *     subroutine fsigusr()
128: *     write(*,*) '<< Dummy routine for FSIGUSR is called. Do nothing>>'
129: *     return
130: *     end
```

src/utls/cdummy.f

```
131: C
132: *      subroutine fgetusrsig(isignum,iflag)
133: *      write(*,*)
134: *      &      '<< Dummy routine for fgetusrsig is called. Do nothing>>'
135: *      return
136: *      end
137: C
138: C.....
139: C
140: *      subroutine flushstd()
141: C
142: C      Flush standard output stream.
143: C
144: *      write(*,*) '<< Dummy routine for FSIGSET is called. Do nothing>>'
145: *      return
146: *      end
147: C
148: C.....
149: C
150: *      subroutine ftgetenv( name, value )
151: C
152: C      Get environment variable.
153: C
154: *      character*(*) name, value
155: *      write(*,*) '<< Dummy routine for FTGETENV is called. Do nothing>>'
156: *      return
157: *      end
158: C
159: C.....
160: C
161: *      subroutine ftsystem( icode, command )
162: C
163: C      Execute command.
164: C
165: *      character*(*) command
166: *      write(*,*) '<< Dummy routine for FTSYSTEM is called. Do nothing>>'
167: *      return
168: *      end
169: C
170: C.....
171: C
172: *      subroutine funbuf()
173: C
174: C      Set standard output stream to line buffered mode.
175: C
176: *      write(*,*) '<< Dummy routine for FUNBUF is called. Do nothing>>'
177: *      return
178: *      end
179: C
180: C.....
181: C
182: *      function lfmalloc ( nbytes )
183: C
184: C      Works like "malloc"
185: C
186: *      write(*,*) 'XXX Dummy function for LFMALLOC is called. XXXXXXXX'
187: *      write(*,*) 'XXX But it must not be called as dummy function!!'
188: *      write(*,*) 'XXX Compilation setting must be modified.'
189: *      stop
190: *      end
191: C
192: C/#ENDIF
193: C
194: *      subroutine cdumyf()
195: *      return
```

196: end

src/utls/cplin.f

```
1:      subroutine CPLIN( IUIN,  LINE,  NCHAR, IEND )
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C PURPOSE: INPUT A LINE FROM UNIT IUIN.
7: C           (SKIP COMMENT LINE & COMMENT CHARACTERS)
8: C           IEND = -1 / 0 : END OF INPUT / NO
9: C           NCHAR : EFFECTIVE CHARACTERS AS INPUT. ( 0 TO LEN(LINE))
10: C
11: C   This routine is almost identical to 'GETLIN' routine but does not
12: C   interpret parameter definition line ('%' line) and only copy them
13: C   to buffer 'LINE'. And cannot handle virtual file.
14: C
15: C=====
16: C   character*(*) LINE
17: C
18: C       IEND      = 0
19: C       NCHAR      = 0
20: C       LINE       = ' '
21: C
22: C   100 read(IUIN,'(A)',end =110) LINE
23: C
24: C       .... COMMENT ....
25: C
26: C       if ( LINE(1:1).eq.'*' ) go to 100
27: C
28: C       K          = INDEX(LINE,'/*')
29: C       if ( K.eq.0 ) then
30: C           NCHAR   = LEN(LINE)
31: C       else
32: C           NCHAR   = K - 1
33: C           LINE(K:) = ' '
34: C       end if
35: C       if ( NCHAR.eq.0 ) go to 100
36: C       return
37: C
38: C   110 IEND      = -1
39: C       return
40: C   end
```

src/utlis/cputime2.c

```

1: #ifndef NO_CPUTIME_C
2: /*****
3:  *
4:  *   Elapsed CPU time monitor (parent + child process)
5:  *
6:  *   by using POSIX system call "times".
7:  *
8:  *   Programmed by M.Sasaki (the Japan Research Institute Ltd.)
9:  *
10: *   Used with mutually exclusive cpp options;
11: *   -DPOSIX_C, -DSUN_C and -DANSI_C.
12: *
13: *****/
14: /*****
15:  * First version 25 May 1998
16: *****/
17:
18: #include <stdio.h>
19:
20: #ifdef SUN_C
21:
22: #include <unistd.h>
23: #include <sys/types.h>
24: #include <sys/times.h>
25:
26: #elif POSIX_C
27:
28: #include <unistd.h>
29: #include <sys/times.h>
30:
31: #elif ANSI_C
32:
33: #include <time.h>
34:
35: #endif
36:
37: /*****
38:  * Fortran callable C subroutine :
39:  *
40:  *   UNIX vendors have their own convention or rule for external
41:  *   names of FORTRAN coded subprograms or commons.
42:  *
43:  *   We prepare equivalent sources for C coded routines
44:  *
45:  *   1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
46:  *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
47:  *   3. Uppercase string of FORTRAN name. ( Ncube etc. )
48:  *   4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
49:  *
50:  *
51: *****/
52:
53: #ifdef POSIX_C
54: /*****
55:  *
56:  *   For environments having POSIX library routine "times".
57:  *   ( the function "times" is defined in POSIX but not in ANSI-C)
58:  *
59:  *
60: *****/
61:
62: static int init = 0 ;
63:
64: void cputime2_( ttt )
65:     double * ttt ;

```

```

66: {
67:     static struct tms tt ;
68:     static double resolution ;
69:
70:     if( init == 0 )
71:     {
72:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
73:         times( &tt ) ;
74:         init = 1 ;
75:     }
76:     times( &tt ) ;
77:     *ttt = (double)((tt.tms_utime + tt.tms_stime +
78:                     tt.tms_cutime + tt.tms_cstime)*resolution) ;
79: }
80:
81: void cputime2 ( ttt )
82:     double * ttt ;
83: {
84:     static struct tms tt ;
85:     static double resolution ;
86:
87:     if( init == 0 )
88:     {
89:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
90:         times( &tt ) ;
91:         init = 1 ;
92:     }
93:     times( &tt ) ;
94:     *ttt = (double)((tt.tms_utime + tt.tms_stime +
95:                     tt.tms_cutime + tt.tms_cstime)*resolution) ;
96: }
97: void CPUTIME2( ttt )
98:     double * ttt ;
99: {
100:     static struct tms tt ;
101:     static double resolution ;
102:
103:     if( init == 0 )
104:     {
105:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
106:         times( &tt ) ;
107:         init = 1 ;
108:     }
109:     times( &tt ) ;
110:     *ttt = (double)((tt.tms_utime + tt.tms_stime +
111:                     tt.tms_cutime + tt.tms_cstime)*resolution) ;
112: }
113: void _CPUTIME2( ttt )
114:     double * ttt ;
115: {
116:     static struct tms tt ;
117:     static double resolution ;
118:
119:     if( init == 0 )
120:     {
121:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
122:         times( &tt ) ;
123:         init = 1 ;
124:     }
125:     times( &tt ) ;
126:     *ttt = (double)((tt.tms_utime + tt.tms_stime +
127:                     tt.tms_cutime + tt.tms_cstime)*resolution) ;
128: }
129:
130: #endif /** end of POSIX_C version */

```

src/utls/cputime2.c

```

131:
132: #ifdef SUN_C
133:
134: /*****
135:  *
136:  *   Elapsed CPU time monitor on old SUN-OS's whose clock tic
137:  *   may be 1/60 sec.
138:  *
139:  *   Programmed by M.Sasaki (the Japan Research Institute Co. Ltd.)
140:  *
141:  *   1st version:  20 Jan 1993
142:  *
143:  *****/
144:
145:
146: static int init =0 ;
147:
148: void cputime2( ttt )
149:     double * ttt ;
150: {
151:     static struct tms tt ;
152:     static double resolution ;
153:
154:     if( init == 0 )
155:     {
156: #ifdef _SC_CLK_TCK
157:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
158: #else
159:         resolution = 1.0/(double)60.0 ;
160: #endif
161:         times( &tt ) ;
162:         init = 1 ;
163:     }
164:
165:     times( &tt ) ;
166:     *ttt = (double)((tt.tms_utime + tt.tms_stime +
167:         tt.tms_cutime + tt.tms_cstime)*resolution) ;
168: }
169:
170: #endif /** end of SUN_C version **/
171:
172: #ifdef ANSI_C
173:
174: /*****
175:  * Accumulated "CPU" (?) time using ANSI C function clock().
176:  *
177:  * For systems whose clock tic is microsecond and clock_t is
178:  * 32 bit integer:
179:  *
180:  * Calling intervals longer than 36 minutes may give inaccurate
181:  * timings, and it is impossible to correct timings for calling
182:  * intervals more than 72 minutes :-).
183:  *
184:  * On non UNIX plathome, the time returned by "clock" is not
185:  * a "CPU time" of the process.
186:  *
187:  *****/
188:
189: static int init =0 ;
190: void MY_CPUTIME2() ;
191:
192: void cputime2( ttt )
193:     double * ttt ;
194: {
195:     MY_CPUTIME2(ttt) ;

```

```

196: }
197: void cputime2_( ttt )
198:     double * ttt ;
199: {
200:     MY_CPUTIME2(ttt) ;
201: }
202: void CPUTIME2( ttt )
203:     double * ttt ;
204: {
205:     MY_CPUTIME2(ttt) ;
206: }
207: void _CPUTIME2( ttt )
208:     double * ttt ;
209: {
210:     MY_CPUTIME2(ttt) ;
211: }
212:
213: void MY_CPUTIME2( ttt )
214:     double * ttt ;
215: {
216:     static clock_t tt, t1;
217:     static double resolution, tttd ;
218:     static double accum_time = 0.0 ;
219:
220: #define MIN_36 35.791394133333 /** 2^31 / 1000,000 / 60 **/
221:
222:     if( init == 0 )
223:     {
224:         tt = clock() ;
225:         accum_time = (double)tt / CLOCKS_PER_SEC ;
226:
227:         if ( CLOCKS_PER_SEC == 1000000 )
228:         {
229:             init = 1 ; /** clock tic is micro second **/
230:         }
231:         else
232:             init = 2 ;
233:
234:         resolution = 1.0/(double)CLOCKS_PER_SEC ;
235:     }
236:
237:     t1 = clock() ;
238:     accum_time += (double)(t1 - tt)*resolution ;
239:
240: /*****
241:  * Trying to correct timings for calling intervals
242:  * longer than 36 minutes.
243:  *****/
244:     if( init == 1 && t1 < tt ) accum_time += MIN_36 ;
245:
246:     tt = t1 ;
247:     *ttt = (double)accum_time ;
248: }
249: #endif /** end of ANSI_C version **/
250:
251: #else
252:
253: void cputime2_( ttt )
254:     double * ttt ;
255: {
256:     *ttt = 0.0 ;
257: }
258: void cputime2 ( ttt )
259:     double * ttt ;
260: {

```

src/Utils/cputime2.c

```
261:     *ttt = 0.0 ;
262: }
263: void CPUTIME2( ttt )
264:     double * ttt ;
265: {
266:     *ttt = 0.0 ;
267: }
268: void _CPUTIME2( ttt )
269:     double * ttt ;
270: {
271:     *ttt = 0.0 ;
272: }
273:
274: #endif /** end of ifndef **/
```

WAVE

src/utls/cputime.c

```
1: #ifndef NO_CPUTIME_C
2: /*****
3:  *
4:  *   Elapsed CPU time monitor
5:  *
6:  *   Programmed by M.Sasaki (the Japan Research Institute Ltd.)
7:  *
8:  *   Used with mutually exclusive cpp options;
9:  *   -DPOSIX_C, -DSUN_C and -DANSI_C.
10:  *
11:  *****/
12: /*****
13:  *   1st version: 15 Oct 1992
14:  *   2nd version: 7 Mar 1996
15:  *   For non POSIX but ANSI compilers, enable monitoring
16:  *   using "clock" function ( it may return bogus value
17:  *   if a calling interval is more than ~36 minutes
18:  *   when unit of the time tick value is microsecond.
19:  *   10 Mar 1996: change argument type from single float to double.
20:  *****/
21:
22: #include <stdio.h>
23:
24: #ifdef SUN_C
25:
26: #include <unistd.h>
27: #include <sys/types.h>
28: #include <sys/times.h>
29:
30: #elif POSIX_C
31:
32: #include <unistd.h>
33: #include <sys/times.h>
34:
35: #elif ANSI_C
36:
37: #include <time.h>
38:
39: #endif
40:
41: /*****
42:  *   Fortran called C subroutine :
43:  *
44:  *   UNIX venders have their own convention or rule for external
45:  *   names of FORTRAN coded subprograms or commons.
46:  *
47:  *   We prepare equivalent sources for C coded routines
48:  *
49:  *   1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
50:  *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
51:  *   3. Uppercase string of FORTRAN name. ( Ncube etc. )
52:  *   4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
53:  *
54:  *****/
55: /*****
56:
57: #ifdef POSIX_C
58:
59: /*****
60:  *
61:  *   For environments having POSIX library routine "times".
62:  *   ( the function "times" is defined in POSIX but not in ANSI-C)
63:  *
64:  *****/
65:
```

```
66: static int init =0 ;
67:
68: void cputime_( ttt )
69: {
70:     static struct tms tt ;
71:     static double resolution ;
72:
73:     if( init == 0 )
74:     {
75:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
76:         times( &tt ) ;
77:         init = 1 ;
78:     }
79:
80:     times( &tt ) ;
81:     /*return (tt.tms_utime + tt.tms_stime)*resolution **/;
82:     *ttt = (double)((tt.tms_utime + tt.tms_stime)*resolution) ;
83: }
84:
85: void cputime ( ttt )
86: {
87:     static struct tms tt ;
88:     static double resolution ;
89:
90:     if( init == 0 )
91:     {
92:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
93:         times( &tt ) ;
94:         init = 1 ;
95:     }
96:
97:     times( &tt ) ;
98:     *ttt = (double)((tt.tms_utime + tt.tms_stime)*resolution) ;
99: }
100: void CPUTIME( ttt )
101: {
102:     static struct tms tt ;
103:     static double resolution ;
104:
105:     if( init == 0 )
106:     {
107:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
108:         times( &tt ) ;
109:         init = 1 ;
110:     }
111:
112:     times( &tt ) ;
113:     *ttt = (double)((tt.tms_utime + tt.tms_stime)*resolution) ;
114: }
115: void _CPUTIME( ttt )
116: {
117:     static struct tms tt ;
118:     static double resolution ;
119:
120:     if( init == 0 )
121:     {
122:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
123:         times( &tt ) ;
124:         init = 1 ;
125:     }
126:
127:     times( &tt ) ;
128:     *ttt = (double)((tt.tms_utime + tt.tms_stime)*resolution) ;
129: }
130:
```


src/utls/cputime.c

```
131: #endif /** end of POSIX_C version **/
132:
133: #ifdef SUN_C
134:
135: /*****
136:  *
137:  *   Elapsed CPU time monitor on old SUN-OS's whose clock tic
138:  *   may be 1/60 sec.
139:  *
140:  *   Programmed by M.Sasaki (the Japan Research Institute Co. Ltd.)
141:  *
142:  *   1st version: 20 Jan 1993
143:  *
144:  *****/
145:
146:
147: static int init = 0 ;
148:
149: void cputime_( ttt )
150:     double * ttt ;
151: {
152:     static struct tms tt ;
153:     static double resolution ;
154:
155:     if( init == 0 )
156:     {
157: #ifdef _SC_CLK_TCK
158:         resolution = 1.0/(double)sysconf( _SC_CLK_TCK ) ;
159: #else
160:         resolution = 1.0/(double)60.0 ;
161: #endif
162:         times( &tt ) ;
163:         init = 1 ;
164:     }
165:
166:     times( &tt ) ;
167:     *ttt = (double)((tt.tms_etime + tt.tms_stime)*resolution) ;
168: }
169:
170: #endif /** end of SUN_C version **/
171:
172: #ifdef ANSI_C
173:
174: /*****
175:  * Accumulated "CPU" (?) time using ANSI C function clock().
176:  *
177:  * For systems whose clock tic is microsecond and clock_t is
178:  * 32 bit integer:
179:  *
180:  * Calling intervals longer than 36 minutes may give inaccurate
181:  * timings, and it is impossible to correct timings for calling
182:  * intervals more than 72 minutes :-).
183:  *****/
184:
185: static int init = 0 ;
186: void MY_CPUTIME() ;
187:
188: void cputime( ttt )
189:     double * ttt ;
190: {
191:     MY_CPUTIME(ttt) ;
192: }
193: void cputime_( ttt )
194:     double * ttt ;
195: {
196:     MY_CPUTIME(ttt) ;
197: }
198: void CPUTIME( ttt )
199:     double * ttt ;
200: {
201:     MY_CPUTIME(ttt) ;
202: }
203: void _CPUTIME( ttt )
204:     double * ttt ;
205: {
206:     MY_CPUTIME(ttt) ;
207: }
208:
209: void MY_CPUTIME( ttt )
210:     double * ttt ;
211: {
212:     static clock_t tt, t1 ;
213:     static double resolution, tttd ;
214:     static double accum_time = 0.0 ;
215:
216: #define MIN_36 35.791394133333 /** 2^31 / 1000,000 / 60 **/
217:
218:     if( init == 0 )
219:     {
220:         tt = clock() ;
221:         accum_time = (double)tt / CLOCKS_PER_SEC ;
222:
223:         if ( CLOCKS_PER_SEC == 1000000 )
224:         {
225:             init = 1 ; /** clock tic is micro second **/
226:         }
227:         else
228:             init = 2 ;
229:
230:         resolution = 1.0/(double)CLOCKS_PER_SEC ;
231:     }
232:
233:     t1 = clock() ;
234:     accum_time += (double)(t1 - tt)*resolution ;
235:
236: /*****
237:  * Trying to correct timings for calling intervals
238:  * longer than 36 minutes.
239:  *****/
240:     if( init == 1 && t1 < tt ) accum_time += MIN_36 ;
241:
242:     tt = t1 ;
243:     *ttt = (double)accum_time ;
244: }
245: #endif /** end of ANSI_C version **/
246:
247: #endif /** end of ifndef **/
```

src/utls/cputm2.f

```
1:      subroutine CPUTM2( X )
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1998)
4: C-----
5: C  Returns CPU time (or wall clock time) of caller process and
6: C  its child processes if possible as single precision floating
7: C  point number in second.
8: C
9: C  Do not use for precise CPU time monitoring.
10: C
11: C=====
12: C/#IF CUTIL.AND.MS_VISUAL
13: C
14: C ... This interface block is for CUTIL & MS-Visual tools. ...
15: C
16: *   interface
17: *     subroutine CPUTIME2(D)
18: *       real*8  D
19: *CDEC$    ATTRIBUTES C :: CPUTIME2
20: *CDEC$    ATTRIBUTES REFERENCE :: D
21: *     end subroutine
22: *   end interface
23: C/#ENDIF
24:      real  X
25: C
26: C ... C-langage coded cputime2 function if available ...
27: C
28: C/#IF UNIX .OR. CUTIL
29: C
30:      real*8 D
31:      call CPUTIME2(D)
32:      X = D
33: C
34: C/#ELSE
35: C
36: C ... No method to get parent/child combined CPU time ...
37: C      return that of current process only
38: C
39: *     call CPUTM(X)
40: C/#ENDIF
41:      return
42:      end
```

src/utls/cputm.f

```
1:      subroutine CPUTM( X )
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C  Returns CPU time (or wall clock time) as single precision floating
6: C  point number in second.
7: C  Do not use for precise CPU time monitoring.
8: C
9: C=====
10: C/#IF CUTIL.AND.MS_VISUAL
11: C
12: C ... This interface block is for CUTIL & MS-Visual tools. ...
13: C
14: *      interface
15: *          subroutine CPUTIME(D)
16: *              real*8 D
17: *CDEC$      ATTRIBUTES C :: CPUTIME
18: *CDEC$      ATTRIBUTES REFERENCE :: D
19: *          end subroutine
20: *      end interface
21: C/#ENDIF
22:      real X
23: C
24: C ... C-language coded cputime function ...
25: C
26: C/#IF SYSTEM(HP* SUN* MIPS* NEC* SGI* ) .OR. CUTIL
27:
28:      real*8 D
29:      call CPUTIME(D)
30:      X = D
31:
32: C/#ELSEIF SYSTEM(FACOM*)
33: *
34: *      call CLOCK(X,0,1)
35: *
36: C/#ELSEIF SYSTEM( DECOSF* PARAGON*)
37: *
38: *      external ETIME
39: *      real TARRAY(2)
40: *      X = ETIME( TARRAY )
41: *
42: C/#ELSEIF SYSTEM(SX* ACOS*)
43: *
44: *      real*8 D
45: *      call CLOCK(D)
46: *      X = D
47: *
48: C/#ELSEIF SYSTEM(CRAY*)
49: *
50: *      call SECOND(X)
51: *
52: C/#ELSEIF SYSTEM(IBMRS* AIX*)
53: *
54: *      X = MCLOCK()/100.0
55: *
56: C/#ELSE
57: C
58: C ... no way to get process CPU time. use wallclock time.
59: C
60: *
61: *      call TOKEI(X,0)
62: *
63: C/#ENDIF
64:      return
65:      end
```

src/utlils/dentak.f

```

1: *VOCL TOTAL,SCALAR
2: C
3: C   MVP/GMVP UTILITY
4: C
5: C=====
6: C PURPOSE: Translation of a character string to a numerical value.
7: C   (Works like a pocket or desktop calculator, and it is called
8: C   "den-taku" in Japanese :-))
9: C
10: C   function DENTAK( LINE, IUPR, IERR )
11: C
12: C       LINE: character string including expression
13: C       IUPR: message printout I/O unit
14: C       IERR: error code (returns non-zero on error)
15: C
16: C   subroutine CDENTK( LINE, IUPR, IERR,
17: C       ELINE, ISTMT0, IPFLO, ISTKM0, IVECN0)
18: C
19: C       LINE: character string including expression
20: C       IUPR: message printout I/O unit
21: C       IERR: error code (returns non-zero on error)
22: C       ELINE: expression in which symbolic parameters are expanded
23: C
24: C       ISTMT0 : number of individual statements ...
25: C       IPFLO  : number of symbolic parameter reference
26: C       ISTKM0 : maximum stack depth
27: C       IVECN0 : number of vector variable reference ...
28: C
29: C   subroutine EVCHAR, DVCHAR
30: C
31: C       Auxiliary routines to expand a numeric value to a string.
32: C
33: C   << possible expressions >>
34: C
35: C   * numeric constant:
36: C       integer constant expression
37: C       floating constant expression ( exponent with E or D )
38: C   * numeric expression with "+", "-", "*", "/" and "( )"
39: C   * power operator "**" as fortran and numeric functions
40: C       like fortran intrinsics;
41: C
42: C       SQRT(x), SIN(x), COS(x), TAN(x), ATAN2(x,y),
43: C       ASIN(x), ACOS(x), ATAN(x),
44: C       LOG(x), LOG10(x), EXP(x), MAX(x,y,z,...), MIN(x,y,z,...)
45: C       MOD(x,y), INX(x), NINT(x)
46: C       SINH(x), COSH(x),TANH(x) (hyperpolc functions)
47: C
48: C   and other numeric functions;
49: C
50: C       SIND(x),COSD(x),TAND(x) (trigonometric functions in degree)
51: C
52: C   * logical functions; (false if zero, true if non-zero)
53: C
54: C       IF( a, b, c ) := b if a is true(ne.0) else = c
55: C       AND( a,b,c,... ), OR(a,b,c,...): returns 1 (true) or 0
56: C       NOT( a ) : returns 1 if a.eq.0 else returns 0
57: C       EQ(a,b ) : returns 1 if a.eq.b else returns 0
58: C       NE(a,b ) : returns 1 if a.ne.b else returns 0
59: C       GT(a,b,c,... ) : returns 1 if a>b>c>... else returns 0
60: C       GE(a,b,c,... ) : returns 1 if a>=b>c>=... else returns 0
61: C       LT(a,b,c,... ) : returns 1 if a<b<c<... else returns 0
62: C       LE(a,b,c,... ) : returns 1 if a<=b<=c<=... else returns 0
63: C
64: C   * symbolic parameter (variable) definition;
65: C

```

```

66: C       name = expression [, name = expression,...]
67: C
68: C   * reference of symbolic parameters as constants.
69: C
70: C   * mode control or special command: a string beginning with "&&"
71: C
72: C       &&SILENT ON : output no error message.
73: C       &&REDEFINE ON : enable re-definition of symbolic parameter value
74: C       &&ECHO ON : output expression and result on I/O unit 66
75: C       &&TRACE ON : produce debug output
76: C       &&PRINT : printout all symbolic parameters
77: C       &&CLEAR : clear and undefine all symbolic parameters
78: C
79: C   Modes with "ON" are disabled by the same commands with "OFF".
80: C
81: C   << Bugs >>
82: C
83: C       Too long as single subprogram ???
84: C
85: C-----
86: C CALLED FROM: FREAD,PARA
87: C CALLS: most of fortran's numeric intrinsic functions
88: C       DVCHAR.
89: C=====
90: C
91: C-----
92: C       function DENTAK(LINE,IUPR,IERR)
93: C-----
94: C
95: C       character*(*) LINE
96: C       real*8 DENTAK, DENTK0
97: C
98: C       character*4 CDUMMY
99: C
100: C       JCHECK = 0
101: C       DENTAK = DENTK0(LINE,JCHECK,IUPR,IERR,CDUMMY,
102: C       & ISTMT0,IPFLO,ISTKM0,IVECN0)
103: C
104: C       return
105: C       end
106: C
107: C-----
108: C       subroutine CDENTK( LINE, IUPR, IERR, ELINE, ISTMT0,IPFLO, ISTKM0,
109: C       & IVECN0 )
110: C-----
111: C
112: C=====
113: C CDENTK:
114: C
115: C   Does not get result, but only checks syntax and expands
116: C   symbolic parameter values for use in vector version of DENTAK
117: C   (VDENTK routine).
118: C
119: C   * replace symbolic parameter reference to numeric character string.
120: C       LINE --> ELINE
121: C   * check number of statements: ISTMT
122: C   * check number of symbolic parameter reference : IPFL
123: C   * check maximum value stack depth : ISTKM
124: C   * check number of vector variable (@...) reference : IVECN
125: C
126: C=====
127: C
128: C       character*(*) LINE, ELINE
129: C       real*8 DENTK0, D
130: C

```

src/utls/dentak.f

```

131:      JCHECK = 1
132:
133:      D = DENTK0(LINE,JCHECK,IUPR,IERR,ELINE,
134:      &          ISTMT0,IPFL0,ISTKM0,IVECN0)
135:
136:      return
137:      end
138: C
139: C-----
140: C
141: C === Working body of DENTAK & CDENTK ===
142: C
143: C-----
144: C
145: C-----
146:      function DENTK0(LINE,JCHECK,IUPR,IERR,ELINE,
147:      &          ISTMT0,IPFL0,ISTKM0,IVECN0)
148: C-----
149: C
150:      implicit real*8(D)
151: C
152:      real*8 DENTK0
153: C
154: cccc include 'INC/_IOUNIT'
155: C
156:      character LINE*(*)
157:      character ELINE*(*)
158: C
159: C .... STACK. ....
160: C
161: C      STK : numeric data(& function #) stack (counter: ISTN )
162: C      ISTK : operator # stack (counter: ISTO )
163: C      LHSTK : stack for lefthand-side variables (counter: ISTLH )
164: C      NARG : function-argument number counter (counter: ISTF )
165: C
166: C
167:      parameter( MS = 128 )
168:      real*8 STK(MS)
169:      integer ISTK(MS)
170:      integer LHSTK(MS)
171:      integer NARG(MS)
172: C
173: C ... TEMPORARY ...
174: C
175:      real*8 TVALUE
176: C
177: C
178:      character OPERTR*9, NUM*11
179:      character CHARW*8, VCHAR*32
180: C
181: C .... WORKING BUFFER
182: C
183: C      LIN: input 'LINE' is compressed to this array.
184: C      IPLIN: original position in 'LINE'
185: C
186:      parameter( MAXCOL = 256 )
187:      character LIN*(MAXCOL)
188:      integer IPLIN(MAXCOL)
189: C
190: C .... FORMAT STRING FOR INTERNAL INPUT ....
191: C
192:      character FORMT*32
193: C
194: C .... FUNCTION SYMBOLS FUNC ) & ARGUMRNT NUMBERS(NFARG) ....
195: C      ( NFARG < 0 : at least -NFARG arguments required)

```

```

196: C
197:      parameter( NFUNC = 33 )
198:      character*6 FUNC(NFUNC)
199:      integer NFARG(NFUNC)
200: C
201: C =====
202: C .... SYMBOLIC PARAMETERS & VALUES OF THEM ....
203: C =====
204: C
205: C      These data must be saved call to call.
206: C
207: C      PNAME: names of symbolic parameters
208: C      VALUE: numerical values of symbolic parameters
209: C      JPSTAT: status of symbolic parameters
210: C              0 = undefined, 1 = numeric value,
211: C              2 = string (not supported)
212: C              3- = ?
213: C      NP : number of symbolic parameters.
214: C
215: C
216:      parameter( MAXP = 4096 )
217:      character*8 PNAME(MAXP)
218:      real*8 VALUE(MAXP)
219:      integer JPSTAT(MAXP)
220:      integer NP
221: C
222:      parameter( DPAI = 3.1415926535897932D+00/180.0D0 )
223: C
224: C
225:      parameter( KFUNC = 0 )
226:      parameter( KEND = 1 )
227: C
228: C ... operator #'s : <position in 'OPRTR' string> + 1
229: C
230:      parameter( KLBRC = 2 )
231:      parameter( KRBRC = 3 )
232:      parameter( KPLUS = 4 )
233:      parameter( KMINUS = 5 )
234:      parameter( KMULT = 6 )
235:      parameter( KDIV = 7 )
236:      parameter( KPOW = 8 )
237:      parameter( KCOMMA = 9 )
238:      parameter( KEQUAL = 10 )
239:      parameter( KUNMNS = 11 )
240:      parameter( NMOPR = KUNMNS )
241: C
242:      parameter( IECHO = 66 )
243: C
244: C .... TABLE OF CALCULATION RULES ....
245: C
246:      integer ICALC(0:NMOPR,0:NMOPR)
247: C
248: C -----
249: C Data initialization
250: C -----
251: C
252: C
253:      data PNAME(1) /'          ', VALUE /MAXP*0.0d0/,
254:      &          JPSTAT /MAXP*0/, NP /0/
255: C
256: C
257: C .... OPERATOR, NUMBER ....
258: C      ( POWER OPERATOR '***' WILL BE REPLACED BY '#' )
259: C
260:      data OPERTR / '()+-*/#,,=' /, NUM /'0123456789.'/

```

src/utlils/dentak.f

```

261: C
262:   data (FUNC(I),I=1,NFUNC) /
263:   &      'EXP'  'LOG'  'INT'  'SIN'  'COS'  '
264:   &      'TAN'  'ASIN' 'ACOS' 'ATAN' 'SQRT' '
265:   &      'ABS'  'MAX'  'MIN'  'MOD'  'LOG10' '
266:   &      'SIND' 'COSD' 'TAND' '
267:   &      'IF'   'AND'  'OR'   '
268:   &      'NOT'  'EQ'   'NE'   'GT'   'GE'   '
269:   &      'LT'   'LE'   'NINT' 'ATAN2' '
270:   &      'SINH' 'COSH' 'TANH' '/'
271: C
272: C   NFARG : function argument number required
273: C
274:   data (NFARG(I),I=1,NFUNC) /
275:   &      1, 1, 1, 1, 1, 1,
276:   &      1, 1, 1, 1, 1, 1,
277:   &      1, -2, -2, 2, 1,
278:   &      1, 1, 1,
279:   &      3, -2, -2,
280:   &      1, 2, 2, -2, -2,
281:   &      -2, -2, 1, 2,
282:   &      1, 1, 1/
283: C
284: C   .... STATE TABLE ....
285: C
286:   data      ICALC /
287:   Z  2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 0
288:   1  1, 4, 1, 5, 1, 1, 1, 1, 1, 7, 1, 1
289:   2  1, 5, 1, 3, 1, 1, 1, 1, 1, 6, 1, 1
290:   3  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
291:   4  1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 5, 0
292:   5  1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 5, 0
293:   6  1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 5, 0
294:   7  1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 5, 0
295:   8  1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 5, 0
296:   9  0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
297:   A  1, 8, 1, 5, 1, 1, 1, 1, 1, 8, 1, 1
298:   B  1, 8, 1, 8, 8, 8, 8, 8, 8, 8, 0, 0
299:   X  /
300: C
301: C
302: C   F END ( ) + - * / ** , = -
303: C
304: C 0 F 2 2 1 2 2 2 2 2 2 2 1 0
305: C 1 END 1 4 1 5 1 1 1 1 1 1 7 1 1
306: C 2 ( 1 5 1 3 1 1 1 1 1 1 6 1 1
307: C 3 ) 0 0 0 0 0 0 0 0 0 0 0 0 0
308: C 4 + 1 2 1 2 2 2 2 1 1 1 2 5 0
309: C 5 - 1 2 1 2 2 2 2 1 1 1 2 5 0
310: C 6 * 1 2 1 2 2 2 2 2 2 1 2 5 0
311: C 7 / 1 2 1 2 2 2 2 2 2 1 2 5 0
312: C 8 ** 1 2 1 2 2 2 2 2 2 2 2 5 0
313: C 9 , 0 5 0 0 0 0 0 0 0 0 0 0 0
314: C 10 = 1 8 1 5 1 1 1 1 1 1 8 1 1
315: C 11 - 1 8 1 8 8 8 8 8 8 8 8 0 0
316: C
317: C
318:   data INIT /1/
319: C
320: C   ... operation mode flags ( 1/0 = ON/OFF )
321: C
322: C   &&SILENT OFF
323: C
324:   data JSILNT /0/
325: C

```

```

326: C   &&REDEFINE OFF
327: C
328:   data JREDEF /0/
329: C   &&ECHO ON
330:   data JECHO, IECHK /0, 0/
331: C
332: C   &&CHECK OFF
333: Ccc DATA JCHECK /0/
334: C
335: C   &&TRACE ON
336:   data JTRACE /0/
337: C
338: C   ... commands ... (added July 1996)
339: C   &&PRINT : printout all symbolic parameters
340: C   &&CLEAR : clear and undefine all symbolic parameters
341: C
342: C -----
343: C
344: C
345: C   ---- initialization ( number of parameter = 0 )
346: C
347:   if ( INIT.eq.1 ) then
348:     NP = 0
349:     do 100 I = 1, MAXP
350:       PNAME(I) = ' '
351:       VALUE(I) = 0.0D0
352:       JPSTAT(I) = 0
353: 100 continue
354:     INIT = 0
355:   end if
356: C
357: C   OPERTR = '()+-*/#,%='
358: C   NUM = '0123456789.'
359: C
360:   DENTK0 = -999.0
361: C
362:   IST = 1
363:   IEND = LEN(LINE)
364:   IERR = 0
365: C
366: C   .... REMOVE BLANK & < > AND STORE RESULT IN 'LIN' ....
367: C
368:   NL = 0
369:   IEEE = IST
370:   do 120 I = IST, IEND
371:     if ( INDEX('<>',LINE(I:I)).eq.0 ) then
372: C
373:       NL = NL + 1
374: C
375:       if ( NL.gt.LEN(LIN) ) then
376:         write(IUPR,*) 'XXX(CALCULATOR) too long input !!',
377:         &      ' ( MAX=', LEN(LIN), ' non-blank characters) XXX'
378:         IERR = 5
379:         do 110 JJ = 1, LEN(LINE), 72
380:           KK = MIN(JJ+71,LEN(LINE))
381:           write(IUPR,'(1X ,'' LINE:''A)') LINE(JJ:KK)
382: 110 continue
383:           return
384:         end if
385: C
386:         LIN(NL:NL) = LINE(I:I)
387:         IPLIN(NL) = I
388:       end if
389:       if ( LINE(I:I).ne.' ' ) IEEE = I
390: 120 continue

```

src/utlils/dentak.f

```

391: C
392: C
393:   if( JTRACE.ne.0 ) then
394:     write(IECHO,'(1X,A,A)') 'input: ',LIN(:NL)
395:   end if
396: C
397: C ... operation mode commands ....
398: C
399:   if ( LIN(1:2).eq.'&&' ) then
400:     if ( INDEX(LIN,'&&SILENT').eq.1 ) then
401:       K = LEN('&&SILENT')
402:       if ( LIN(K+1:K+2).eq.'ON' ) JSILNT = 1
403:       if ( LIN(K+1:K+3).eq.'OFF' ) JSILNT = 0
404:     else if ( INDEX(LIN,'&&REDEFINE').eq.1 ) then
405:       K = LEN('&&REDEFINE')
406:       if ( LIN(K+1:K+2).eq.'ON' ) JREDEF = 1
407:       if ( LIN(K+1:K+3).eq.'OFF' ) JREDEF = 0
408:     else if ( INDEX(LIN,'&&ECHO').eq.1 ) then
409:       K = LEN('&&ECHO')
410:       if ( LIN(K+1:K+2).eq.'ON' ) JECHO = 1
411:       if ( LIN(K+1:K+3).eq.'OFF' ) JECHO = 0
412:     else if (index(lin,'&&CHECK').eq.1) then
413:       Ccc k = len('&&CHECK')
414:       Ccc if(lin(k+1:k+2).eq.'ON') JCHECK = 1
415:       Ccc if(lin(k+1:k+3).eq.'OFF') JCHECK = 0
416:       Ccc
417:     else if( INDEX(LIN,'&&TRACE').eq.1 ) then
418:       K = LEN('&&TRACE')
419:       if(LIN(K+1:K+2).eq.'ON') JTRACE = 1
420:       if(LIN(K+1:K+3).eq.'OFF') JTRACE = 0
421:     else if( LIN(:NL).eq.'&&PRINT' ) then
422:       C write(IUPR,*) '>>>(CALCULATOR) PRINT CONTENTS OF SYMBOLIC',
423:       & ' PARAMETERS.'
424:       & do 122 I=1,NP
425:       &   write(IUPR,
426:       &     '(1X,'<'<',A,'>' VALUE = ',D23.16, ' STATUS ',I3)')
427:       &   PNAME(I),VALUE(I),JPSTAT(I)
428:       &   continue
429:       &   else if( LIN(:NL).eq.'&&CLEAR' ) then
430:       &     write(IUPR,*) '>>>(CALCULATOR) CLEAR ALL SYMBOLIC',
431:       &     ' PARAMETER DEFINITIONS.'
432:       &     NP = 0
433:       &   end if
434:       &   return
435:       &   end if
436:       &   else
437:       &     write(IUPR,*) 'XXX(CALCULATOR) invalid mode command :',
438:       &     LIN(:NL)
439:       &     IERR = 1
440:       &   end if
441:       &   return
442:       &   end if
443:       &   ... initialize echoback message file ....
444:       &   ...
445:       &   if ( IECHK.eq.0.and.JECHO.eq.1 ) then
446:       &     IECHK = 1
447:       &     write(IECHO,'(''1'',A/A)')
448:       &     ' === MONITOR OUTPUT OF <DENTAK> === ',
449:       &     (CALCULATOR UNIT / VERSION 3.1 JUL 1993 ) '
450:       &   end if
451:       &   ... numerical data stack pointer ...
452:       &   ISTN = 0

```

```

456: C
457: C .... function number stack pointer ...
458: C
459: C ISTF = 0
460: C
461: C .... operator number stack pointer ...
462: C
463: C (DUMMY OPERATOR '1' ( END ) is pushed )
464: C
465: C ISTO = 1
466: C ISTK(ISTO) = KEND
467: C ISTLH = 0
468: C
469: C .... MEMORIZE MAXIMUM VALUE-STACK DEPTH & PARAMETER REFERENCE
470: C
471: C ISTKM = 0
472: C IPFL = 0
473: C
474: C .... number of substitution ...
475: C
476: C ISUBN = 0
477: C ISUBNO = 0
478: C
479: C .... number of individual statements ...
480: C
481: C ISTMT = 1
482: C
483: C .... number of vector variable reference ...
484: C
485: C IVECN = 0
486: C
487: C .... for check and parameter expansion mode
488: C if ( JCHECK.ne.0 ) then
489: C   ELINE = ' '
490: C end if
491: C LELIN = 0
492: C JPFLG = 0
493: C
494: C ... IS = current position in translation ...
495: C
496: C IS = 1
497: C
498: C ... the following label is common returning point ...
499: C
500: C 1000 continue
501: C
502: C
503: C
504: C
505: C if ( IS.gt.NL ) then
506: C   K = INDEX(OPERTR,LIN(NL:NL)) + 1
507: C   if ( K.gt.1.and.K.ne.KRBRC ) then
508: C
509: C ... last character is operator and not ')' ...
510: C
511: C   if ( JSILNT.eq.0 ) then
512: C     write(IUPR,*)
513: C     & 'XXX(CALCULATOR) SYNTAX ERROR:',
514: C     & ' last character is operator and not ")"'
515: C   end if
516: C   IERR = 1
517: C   go to 9000
518: C end if
519: C K = KEND
520: C go to 280

```

src/utlils/dentak.f

```

521:      end if
522: C
523: C ..... FIND NEXT OPERATOR .....
524: C
525: C      IE : end position of operand in LIN
526: C      IEO : end position of operator in LIN
527: C
528: C      K : operator number
529: C
530: C      parameter ( kfunc = 0 )
531: C      parameter ( kend = 1 )
532: C      parameter ( klbrc = 2 )
533: C      parameter ( krbrc = 3 )
534: C      parameter ( kplus = 4 )
535: C      parameter ( kminus = 5 )
536: C      parameter ( kmult = 6 )
537: C      parameter ( kdiv = 7 )
538: C      parameter ( kpow = 8 )
539: C      parameter ( kcomma = 9 )
540: C      parameter ( kequal = 10 )
541: C      parameter ( kunmns = 11 )
542: C
543: C      0 : single/multi argument function
544: C
545: C      1 : END of input or stack empty
546: C      2 : (
547: C      3 : )
548: C      4 : +
549: C      5 : -
550: C      6 : *
551: C      7 : /
552: C      8 : ** or # (power)
553: C      9 : ,
554: C      10 : =
555: C      11 : - (unary minus)
556: C
557: C      JN      = INDEX(NUM,LIN(IS:IS))
558: C      IEO      = IS
559: C
560: C      do 140 I = IS, NL
561: C          K      = INDEX(OPERTR,LIN(I:I)) + 1
562: C          IEO      = I
563: C
564: C      .. operator found ..
565: C
566: C          if ( K.gt.1 ) then
567: C
568: C          .... +- in 3.0E-5 0.9D+12 etc. These are not operator ....
569: C
570: C              if ( JN.gt.0.and.INDEX('+-',LIN(I:I)).ne.0.and.I.gt.IS
571: C                  &.and.(INDEX('ED',LIN(I-1:I-1)).ne.0
572: C                  &.or.INDEX('ed',LIN(I-1:I-1)).ne.0 ) ) go to 140
573: C
574: C          .... '***' : power operator
575: C
576: C              if ( I.lt.NL.and.LIN(I:I+1).eq.'***' ) then
577: C                  K      = KPOW
578: C                  IEO      = IEO + 1
579: C              end if
580: C              go to 150
581: C          end if
582: C      140 continue
583: C
584: C      ... end of input ...
585: C

```

```

586:      K      = KEND
587: C
588: C      ... found operator other than "end of input" ...
589: C
590: C      150 IE      = I - 1
591: C
592: C          if( JTRACE .ne. 0 ) then
593: C              write(IECHO,*) 'part: <',LIN(IS:IE),'> operator ',K
594: C          end if
595: C
596: C
597: C      ..... GET VALUES AND PUSH TO STACK (STK) .....
598: C
599: C
600: C          if ( INDEX(NUM,LIN(IS:IS)).ne.0 ) then
601: C
602: C          ... numeric constant ...
603: C
604: C              if ( K.eq.KEQUAL ) then
605: C                  if ( JSILNT.eq.0 ) then
606: C                      write(IUPR,*)
607: C                      &'XXX(CALCULATOR) numeric data is on the',
608: C                      &' lefthand-side of "=" !!! '
609: C                  end if
610: C                  IERR      = 1
611: C                  go to 9000
612: C              end if
613: C
614: C              if ( K.eq.KLBRC ) then
615: C                  if ( JSILNT.eq.0 ) then
616: C                      write(IUPR,*)
617: C                      &'XXX(CALCULATOR) NUMERIC DATA IS ON THE LEFT OF "(" !!! '
618: C                  end if
619: C                  IERR      = 1
620: C                  go to 9000
621: C              end if
622: C
623: C              ISTN      = ISTN + 1
624: C              ISTKM      = MAX(ISTKM,ISTN)
625: C
626: C          .... STACK OVERFLOW .....
627: C
628: C          if ( ISTN.gt.MS ) go to 490
629: C
630: C          .... numeric constant "1.0E" can be taken as "1.0E0" in
631: C          many systems on formatted read, but such an expression
632: C          is not allowed here.
633: C
634: C          if ( LIN(IE:IE).eq.'E' .or. LIN(IE:IE).eq.'D' ) then
635: C              IERR      = 1
636: C              if ( JSILNT.eq.0 ) then
637: C                  write(IUPR,*)
638: C                  &'XXX(CALCULATOR) Invalid numeric ', 'constant <',
639: C                  &LIN(IS:IE), '> (no exponent) XXX'
640: C              end if
641: C              go to 9000
642: C          end if
643: C
644: C          .... GET VALUE .....
645: C
646: C          IDGT      = IE - IS + 1
647: C          if ( IDGT.ge.10 ) then
648: C              write(FORMT,'(''(D'',I2,''.0)'')' ) IDGT
649: C          else
650: C              write(FORMT,'(''(D'',I1,''.0)'')' ) IDGT

```


src/utils/dentak.f

```

651:      end if
652: C
653:      read(LIN(IS:IE),fmt =FORMT,iostat =IOSS) STK(ISTN)
654:      if ( IOSS.ne.0 ) then
655:          IERR = 1
656:          if ( JSILNT.eq.0 ) then
657:              write(IUPR,*)
658:          &      'XXX(CALCULATOR) Invalid numeric ', 'constant <',
659:          &      LIN(IS:IE), '>' XXX'
660:          end if
661:          go to 9000
662:      end if
663: C
664:      if( JTRACE.ne.0 ) then
665:          write(IECHO,'(1X,A,D14.5,A,I3)') 'push: <',STK(ISTN),
666:          &      '>' stack ',ISTN
667:      end if
668: C
669: C ..... FUNCTION OR NAMED-PARAMETER ??? .....
670: C
671: C
672:      else if ( IE.ge.IS ) then
673: C
674: C ..... FUNCTION .....
675: C
676:          if ( IE.lt.NL.and.LIN(IE+1:IE+1).eq.'(' ) then
677: C
678: C
679:              do 160 IFF = 1, NFUNC
680:                  if ( LIN(IS:IE).eq.FUNC(IFF) ) go to 170
681: 160          continue
682:
683:              if ( JSILNT.eq.0 ) then
684:                  write(IUPR,*)
685:          &      'XXX(CALCULATOR) Unsupported function ', ' ('
686:          &      LIN(IS:IE), ') XXX'
687:              end if
688:              IERR = 1
689:              go to 9000
690: C
691: C ..... push function number as operand (obsolete)....
692: C
693: 170          continue
694:
695: C 117          ISTN = ISTN + 1
696: C          IF(ISTN.GT.MS) GOTO 9999
697: C          STK(ISTN) = IFF
698: C
699: C .... reset function argument counter ...
700: C
701:          ISTF = ISTF + 1
702: Cjuly95      NARG(ISTF) = 1
703:          NARG(ISTF) = 0
704: C
705: Ccccccccc   K = KFUNC
706: C
707: C ... set -<function number> as operator.
708: C
709:          K = -IFF
710: C
711:          if( JTRACE.ne.0 ) then
712:              write(IECHO,'(1X,A,A,A,I3)') 'function: <',
713:          &      FUNC(ABS(K)),
714:          &      '>' F-stack ',ISTF
715:          end if

```

```

716: C
717: C ..... substitution .....
718: C
719:      else if ( K.eq.KEQUAL ) then
720: C
721:          ISUBN = ISUBN + 1
722: C
723:          if ( LIN(IS:IS).eq.'%' ) then
724:              if ( JSILNT.eq.0 ) then
725:                  write(IUPR,*)
726:          &      'XXX(CALCULATOR) In value definition',
727:          &      ' of a symbolic parameter, a lefthand-side of type',
728:          &      '%VAL appeared !!'
729:              end if
730:              IERR = 1
731:              go to 9000
732: C
733: C ... vector variable (@...) is only checked ...
734: C
735:          else if ( LIN(IS:IS).eq.'@' ) then
736:              if ( JCHECK.eq.0 ) then
737:                  write(IUPR,*) 'XXX(CALCULATOR) ',
738:          &      ' Vector variable is not allowed.'
739:              IERR = 1
740:              go to 9000
741:          else
742:              IP = 0
743:              go to 200
744:          end if
745:          end if
746:
747:          LP = IE - IS + 1
748:
749:          if ( LP.gt.LEN(PNAME(1)) ) then
750:              if ( JSILNT.eq.0 ) then
751:                  write(IUPR,*)
752:          &      'XXX(CALCULATOR) Too long symbolic parameter',
753:          &      ' name <', LIN(IS:IE), '>' (MAX=', LEN(PNAME(1))),
754:          &      ') XXX'
755:              end if
756:              IERR = 1
757:              go to 9000
758:          end if
759: C
760:          if ( JREDEF.eq.1 ) then
761:              do 180 IP = 1, NP
762:                  if ( LIN(IS:IS+LP-1).eq.PNAME(IP) ) go to 200
763: 180          continue
764:          else
765:              do 190 IP = 1, NP
766:                  if ( LIN(IS:IS+LP-1).eq.PNAME(IP) ) then
767:                      if ( JSILNT.eq.0 ) then
768:                          write(IUPR,*)
769:          &      'XXX(CALCULATOR) Parameter value redefinition',
770:          &      ' in "REDEFINE OFF" mode : ', PNAME(IP)
771:                      end if
772:                      IERR = 8
773:                      go to 9000
774:                  end if
775: 190          continue
776:          end if
777: C
778: C .... new symbolic parameter ....
779: C
780:          NP = NP + 1

```

src/utls/dentak.f

```

781:         if ( NP.gt.MAXP ) then
782:             if ( JSILNT.eq.0 ) then
783:                 write(IUPR,*)
784:             &         'XXX(CALCULATOR) Too many symbolic ',
785:             &         'parameters!! (MAX=', MAXP, ') '
786:             end if
787:             IERR      = 6
788:             go to 9000
789:         end if
790:         PNAME(NP)    = LIN(IS:IS+LP-1)
791:         JPSTAT(NP)   = 0
792: C
793:         if( JTRACE.ne.0 ) then
794:             write(IECHO,'(lx,A,A,A)') 'new parameter: <',
795:             &         PNAME(NP) '>'
796:         end if
797: C
798:         IP           = NP
799: C
800: C         .... push parameter or variable # ....
801: C
802: C 126         ISTN      = ISTN + 1
803: C         IF(ISTN.GT.MS) GOTO 9999
804: C         STK(ISTN) = IP
805: C
806: C 200         ISTLH     = ISTLH + 1
807: C         if ( ISTLH.gt.MS ) go to 490
808: C         LHSTK(ISTLH) = IP
809: C
810: C
811:         else
812: C
813: C         ..... PARAMETER REFERENCE .....
814: C
815: C         JNONP      = 0
816: C
817: C         .... SPECIAL PARAMETER HAVING '%' HEADER ...
818: C         (REFER PROGRAM VARIABLE VALUE BY NAME )
819: C
820:         if ( LIN(IS:IS).eq.'%' ) then
821:             call GTVVAL( LIN(IS+1:IE), TVALUE, IERRRR )
822:             if ( IERRRR.eq.1 ) then
823:                 if ( JSILNT.eq.0 ) then
824:                     write(IUPR,'(/a,a,a,a/)')
825:                     &         'XXX(CALCULATOR) Value of <', LIN(IS+1:IE),
826:                     &         '> is not allowed to refer or variable is ',
827:                     &         'non-existent in program !!'
828:                 end if
829:                 IERR      = 3
830:                 go to 9000
831:             else
832:                 go to 220
833:             end if
834: C
835: C         .... @xxxxx : vector parameter. invalid other than check mode
836: C
837:         else if ( LIN(IS:IS).eq.'@' ) then
838:             if ( JCHECK.eq.0 ) then
839:                 write(IUPR,*) 'XXX(CALCULATOR) Reference of ',
840:                 &         'vector variable <', LIN(IS:IE),
841:                 &         '> is not allowed in this context.'
842:                 IERR      = 3
843:                 go to 9000
844:             else
845:                 IVECN     = IVECN + 1

```

```

846:             JNONP      = 1
847:             TVALUE     = 0.0D0
848:             go to 220
849:         end if
850:     else
851:         do 210 I = 1, NP
852:             if ( LIN(IS:IE).eq.PNAME(I).and.JPSTAT(I).eq.1 ) then
853:                 TVALUE = VALUE(I)
854:                 go to 220
855:             end if
856: 210         continue
857: C
858: C         .... UNDEFINED PARAMETER ....
859: C
860:             if ( JSILNT.eq.0 ) then
861:                 write(IUPR,*)
862:                 &         'XXX(CALCULATOR) Symbolic parameter <', LIN(IS:IE),
863:                 &         '> is not defined.'
864:             end if
865:             IERR      = 3
866:             go to 9000
867:         end if
868: C
869: C         .... PUSH PARAMETER VALUE TO STACK ....
870: C
871: C 220         ISTN      = ISTN + 1
872: C         ISTKM      = MAX(ISTKM,ISTN)
873: C
874:         if ( ISTN.gt.MS ) go to 490
875:         STK(ISTN)    = TVALUE
876: C
877: C
878: C
879:         JPFLG      = 0
880: C
881:         if ( JNONP.eq.0 ) then
882: C
883: C         ..... PARAMETER REFERENCE COUNTER .....
884: C
885:             IPFL     = IPFL + 1
886: C
887: C         .... expand (characterize) parameter value for checking mode ...
888: C
889:             if ( JCHECK.eq.1 ) then
890:                 JPFLG = 1
891: C
892:                 VCHAR = ' '
893:                 if ( TVALUE.lt.0.0 ) then
894:                     VCHAR(1:1) = '('
895:                     call DVCHAR( TVALUE, VCHAR(2:), LLC, IERR )
896:                     if ( IERR.ne.0 ) then
897:                         IERR = 9
898:                         go to 9000
899:                     end if
900:                     LLCHAR = 1 + LLC + 1
901:                     VCHAR(LLCHAR:LLCHAR) = ')'
902:                 else
903:                     call DVCHAR( TVALUE, VCHAR, LLC, IERR )
904:                     if ( IERR.ne.0 ) then
905:                         IERR = 9
906:                         go to 9000
907:                     end if
908:                 end if
909:             end if
910: C
911:             VCHAR     = ' '

```

src/utils/dentak.f

```

911: CC      LLCHAR = 0
912: CC      if ( TVALUE.lt.0.0 ) then
913: CC          LLCHAR = 1
914: CC          VCHAR(1:1) = '('
915: CC      end if
916: CC      write(VCHAR(LLCHAR+1:),fmt='(1P,D28.15)',iostat =IOS)
917: CC      &          TVALUE
918: CC      if ( IOS.ne.0 ) then
919: CC          IERR = 9
920: CC          go to 9000
921: CC      end if
922: C
923: C      ... llchar : effective length of character string
924: C      ... lesig : the last occurrence position of +- sign
925: C
926: CC      LESIG = 0
927: CC      do 230 I = LLCHAR + 1, LEN(VCHAR)
928: CC          if ( VCHAR(I:I).ne.' ' ) then
929: CC              LLCHAR = LLCHAR + 1
930: CC              CHARW(1:1) = VCHAR(I:I)
931: CC              if ( CHARW(1:1).eq.'E'.or. CHARW(1:1).eq.'e'
932: CC      &          .or. CHARW(1:1).eq.'d' ) CHARW(1:1) = 'D'
933: CC              VCHAR(LLCHAR:LLCHAR) = CHARW(1:1)
934: CC              if ( INDEX('+-',VCHAR(LLCHAR:LLCHAR)).ne.0 )
935: CC      &          LESIG = LLCHAR
936: CC      end if
937: CC      continue
938: CC      if ( LLCHAR.lt.LEN(VCHAR) ) VCHAR(LLCHAR+1:) = ' '
939: CC      KD = INDEX(VCHAR,'D')
940:
941: C      .... missing D or E ...
942:
943: CC      if ( KD.eq.0 ) then
944: CC          CHARW = VCHAR(LESIG:LLCHAR)
945: CC          VCHAR(LESIG:LESIG) = 'D'
946: CC          VCHAR(LESIG+1:LLCHAR+1) = CHARW
947: CC          KD = LESIG
948: CC          LLCHAR = LLCHAR + 1
949: CC      end if
950:
951: C      .... check exponent is zero or not
952:
953: CC      do 240 I = KD + 1, LLCHAR
954: CC          if ( INDEX('123456789',VCHAR(I:I)).ne.0 )
955: CC              go to 250
956: CC      continue
957:
958: CC      LLCHAR = KD - 1
959: CC      KD = LLCHAR + 1
960:
961: CC      LLNE = KD - 1
962:
963: C      .... check non-zero part of fraction part
964:
965: CC      do 260 I = KD - 1, 1, -1
966: CC          if ( VCHAR(I:I).ne.'0' ) then
967: CC              LLNE = I
968: CC              if ( VCHAR(I:I).eq.'.' ) LLNE = LLNE - 1
969: CC              go to 270
970: CC          end if
971: CC      continue
972:
973: CC      if ( KD.gt.LLCHAR ) then
974: CC          VCHAR(LLNE+1:) = ' '
975: CC      else

```

```

976: CC          CHARW = VCHAR(KD:LLCHAR)
977: CC          VCHAR(LLNE+1:) = ' '
978: CC          VCHAR(LLNE+1:) = CHARW
979: CC      end if
980: C
981: CC          LLCHAR = INDEX(VCHAR,' ' ) - 1
982: CC          if ( LLCHAR.lt.0 ) LLCHAR = LEN(VCHAR)
983: CC          if ( TVALUE.lt.0.0 ) then
984: CC              VCHAR(LLCHAR+1:LLCHAR+1) = ')'
985: CC              LLCHAR = LLCHAR + 1
986: CC          end if
987: CC      end if
988: CC      end if
989: CC      end if
990: C
991: C      .... no operand before operator .....
992: C
993: C
994: C
995: C      else if ( IE.lt.IS ) then
996: C
997: C      < unary operator ? >
998: C
999: C      ..... + OR - IMMEDIATELY AFTER OPERATER (,=
1000: C
1001: C      if ( K.eq.KPLUS .or. K.eq.KMINUS ) then
1002: C          if ( IS.eq.1
1003: C      &          .or. (IS.gt.1.and.INDEX('=',',',LIN(IS-1:IS-1)).ne.0) )
1004: C      &          then
1005: C
1006: C          .... unary plus means doing nothing.
1007: C
1008: C          if ( K.eq.KPLUS ) then
1009: C              IS = IS + 1
1010: C              go to 1000
1011: C          else
1012: C              K = KUNMNS
1013: C          end if
1014: C      else if ( LIN(IS-1:IS-1).ne.' ' ) then
1015: C          if ( JSILNT.eq.0 ) then
1016: C              write(IUPR,*)
1017: C              'XXX(CALCULATOR) SYNTAX ERROR: ',
1018: C      &              ' invalid use of "+" or "-'.
1019: C          end if
1020: C          IERR = 1
1021: C          go to 9000
1022: C      end if
1023: C
1024: C      .... syntax error other than "),"
1025: C
1026: C      else if ( K.eq.KCOMMA ) then
1027: C          if ( IE.le.0 .or. (IE.gt.0.and.LIN(IE:IE).ne.' ' ) ) then
1028: C              if ( JSILNT.eq.0 ) then
1029: C                  write(IUPR,*)
1030: C                  'XXX(CALCULATOR) SYNTAX ERROR: ',
1031: C      &                  'invalid use of comma.'
1032: C              end if
1033: C              IERR = 1
1034: C              go to 9000
1035: C          end if
1036: C
1037: C      .... equal operator needs lefthand side ...
1038: C
1039: C      else if ( K.eq.KEQUAL ) then
1040: C          if ( JSILNT.eq.0 ) then

```

src/utils/dentak.f

```

1041:      write(IUPR,*)
1042:      &      'XXX(CALCULATOR) SYNTAX ERROR: invalid use of "="'
1043:      end if
1044:      IERR      = 1
1045:      go to 9000
1046: C
1047: C      .... ) ( ...
1048: C
1049:      else if ( K.eq.KLBRC ) then
1050:      if ( IS.gt.1.and.LIN(IS-1:IS-1).eq.'') then
1051:      if ( JSILNT.eq.0 ) then
1052:      write(IUPR,*)
1053:      &      'XXX(CALCULATOR) SYNTAX ERROR: ',
1054:      &      'invalid use of bracket.'
1055:      end if
1056:      IERR      = 1
1057:      go to 9000
1058:      end if
1059:      end if
1060:      end if
1061: C
1062:      if ( JCHECK.eq.1 ) then
1063:      if ( JPFLG.eq.0 ) then
1064:      LEL0      = LELIN + IEO - IS + 1
1065:      if ( LEL0.gt.LEN(ELINE) ) then
1066:      IERR      = 9
1067:      go to 9000
1068:      end if
1069:      ELINE(LELIN+1:LEL0) = LIN(IS:IEO)
1070:      LELIN      = LEL0
1071:      else
1072:      JPFLG      = 0
1073:      LEL0      = LELIN + LLCHAR + IEO - IE
1074:      if ( LEL0.gt.LEN(ELINE) ) then
1075:      IERR      = 9
1076:      go to 9000
1077:      end if
1078:      if ( IE.lt.IEO ) then
1079:      ELINE(LELIN+1:LEL0) = VCHAR(:LLCHAR) // LIN(IE+1:IEO)
1080:      else
1081:      ELINE(LELIN+1:LEL0) = VCHAR(:LLCHAR)
1082:      end if
1083:      LELIN      = LEL0
1084:      end if
1085:      end if
1086: C
1087: C
1088: C ..... CALCULATION .....
1089: C
1090: C      A / K
1091: C      1 2 3 4 5 6 7 8 9
1092: C      END ( ) + - * / ** F
1093: C      1 END 4 1 5 1 1 1 1 1 1
1094: C      2 ( 5 1 3 1 1 1 1 1 1
1095: C      3 ) 0 0 0 0 0 0 0 0 0
1096: C      4 + 2 1 2 2 2 1 1 1 1
1097: C      5 - 2 1 2 2 2 1 1 1 1
1098: C      6 * 2 1 2 2 2 2 2 1 1
1099: C      7 / 2 1 2 2 2 2 2 1 1
1100: C      8 ** 2 1 2 2 2 2 2 2 1
1101: C      9 F 2 1 2 2 2 2 2 2 2
1102: C
1103: C
1104: C
1105: C

```

```

1106: C
1107: C      A / K
1108: C      0 1 2 3 4 5 6 7 8 9 10 11
1109: C
1110: C      F END ( ) + - * / ** , = -
1111: C
1112: C      0 F 2 2 1 2 2 2 2 2 2 1 0
1113: C      1 END 1 4 1 5 1 1 1 1 1 1
1114: C      2 ( 1 5 1 3 1 1 1 1 1 6 1
1115: C      3 ) 0 0 0 0 0 0 0 0 0 0 0
1116: C      4 + 1 2 1 2 2 2 1 1 1 2 5
1117: C      5 - 1 2 1 2 2 2 1 1 1 2 5
1118: C      6 * 1 2 1 2 2 2 2 1 2 5 0
1119: C      7 / 1 2 1 2 2 2 2 1 2 5 0
1120: C      8 ** 1 2 1 2 2 2 2 2 2 5 0
1121: C      9 , 0 5 0 0 0 0 0 0 0 0 0
1122: C      10 = 1 8 1 5 1 1 1 1 8 1 1
1123: C      11 - 1 8 1 8 8 8 8 8 8 0 0
1124: C
1125: C      A ... TOP OF OPERATION STACK <ISTK>
1126: C      K ... NEW OPERATOR
1127: C
1128: C      < OPERATIONS >
1129: C
1130: C      0 : non-existent (A,K) pair
1131: C      1 : PUSH K TO OPERATION STACK <ISTK>
1132: C      2 : * OPERATION BY A.
1133: C      * POP OPERATION STACK <ISTK> (discard)
1134: C      * CONTINUE (A,K) OPERATION
1135: C      3 : POP OPERATION STACK <ISTK>
1136: C      4 : END OF CALCULATION. RETURN TOP OF VALUE STACK <STK>
1137: C      5 : ERROR
1138: C      6 : increment function-argument-counter
1139: C      7 : discard K & top of <STK> return to input
1140: C      8 : like 2. but operator is unary.
1141: C
1142: C      Operator K is discarded and goto next operator search (goto 1000)
1143: C      except 2 and 8.
1144: C
1145: C      280 KK      = K
1146: C      if ( KK.lt.0 ) K      = KFUNC
1147: C      KA      = ISTK(ISTO)
1148: C      if ( KA.lt.0 ) KA      = KFUNC
1149: C
1150: C      if( JTRACE.ne.0 ) then
1151: C      write(IECHO,*) 'new operator & stack top: ',
1152: C      &      K, KA, ' : stack depth ', ISTO
1153: C      end if
1154: C
1155: C      go to(111,222,333,444,555,666,777,888) ICALC(K,KA)
1156: C
1157: C      write(IUPR,*) 'XXX(CALCULATOR) Operation error(may be a bug)',
1158: C      &      ' Please take contact to authors!!!'
1159: C      write(IUPR,*) (ISTK(I),I=1,ISTN)
1160: C      write(IUPR,*) (STK(I),I=1,ISTN)
1161: C      stop 666
1162: C
1163: C      ..... PUSH .....
1164: C
1165: C      111 ISTO      = ISTO + 1
1166: C      if ( ISTO.gt.MS ) go to 490
1167: C      ISTK(ISTO) = KK
1168: C      if ( K.eq.KFUNC ) then
1169: C      K      = KLBRC
1170: C      go to 280

```

src/utlils/dentak.f

```

1171:      end if
1172:      IS      = IEO + 1
1173:      go to 1000
1174: C
1175: C      ..... OPERATION BY AN OPERATOR AT THE TOP OF STACK ....
1176: C
1177:      222 IOP      = ISTK(ISTO)
1178:      if ( IOP.lt.0 ) IOP = KFUNC
1179: C
1180: C      two operands
1181: C
1182:      if ( IOP.ne.KFUNC ) then
1183:      if ( ISTN.le.1 ) then
1184:      if ( JSILNT.eq.0 ) then
1185:      write(IUPR,*)
1186:      &      'XXX(CALCULATOR) SYNTAX ERROR: ',
1187:      &      'Stack conflict.'
1188:      end if
1189:      IERR      = 1
1190:      go to 9000
1191:      end if
1192:      DB      = STK(ISTN)
1193:      DA      = STK(ISTN-1)
1194: C
1195:      if ( JCHECK.ne.0 ) then
1196:      DA = 0.0
1197:      go to 310
1198:      end if
1199: C
1200:      if ( IOP.eq.KPLUS ) then
1201:      DA      = DA + DB
1202:      else if ( IOP.eq.KMINUS ) then
1203:      DA      = DA - DB
1204:      else if ( IOP.eq.KMULT ) then
1205:      DA      = DA*DB
1206:      else if ( IOP.eq.KDIV ) then
1207:      DA      = DA/DB
1208:      else if ( IOP.eq.KPOW ) then
1209:      if ( DA.lt.0.0 ) then
1210:      if ( DB.ne.0.0.and.ABS(NINT(DB)-DB)/DB.lt.1.0D-5 ) then
1211:      DA      = (1-2*MOD(NINT(DB),2))*(ABS(DA)**NINT(DB))
1212:      else if ( DB.eq.0.0 ) then
1213:      DA      = 1.0
1214:      else
1215:      if ( JSILNT.eq.0 ) then
1216:      write(IUPR,*)
1217:      &      'XXX(CALCULATOR) Unpermitted power',
1218:      &      ' for negative number : (', DA, ')**(', DB, ')'
1219:      end if
1220:      IERR      = 2
1221:      return
1222:      end if
1223:      else
1224:      if ( ABS(DB).le.2**30.and.DBLE(NINT(DB)).eq.DB ) then
1225:      DA      = DA**(NINT(DB))
1226:      else
1227:      DA      = DA**DB
1228:      end if
1229:      end if
1230: C
1231: C      .... substitution (obsolete. treated as a unary operator)...
1232: C
1233: C      ELSE IF(IOP.EQ.kequal) THEN
1234: C      VALUE( LHSTK(ISTLH) = DB
1235: C      ISTLH = ISTLH - 1

```

```

1236: Ccccccccccc DA      = DB
1237: C
1238:      end if
1239: C
1240: C      ..... POP NUMERICAL STACK .....
1241: C
1242:      310      ISTN      = ISTN - 1
1243: C
1244: C      ..... substitute by new value .....
1245: C
1246:      STK(ISTN)      = DA
1247: C
1248: C      ..... POP OPERATOR STACK .....
1249: C
1250:      ISTO      = ISTO - 1
1251: C
1252: C      ..... FUNCTION .....
1253: C
1254:      else
1255: C
1256: C      KF      = ISTN-NARG(ISTF)
1257: C      if( istf.le.0 .or. kf .le. 0 ) goto 2000
1258: C      KF      = int(STK(kf))
1259: C
1260:      if ( ISTF.le.0 ) go to 280
1261:      KF      = -ISTK(ISTO)
1262: C
1263:      if ( NFARG(KF).gt.0 ) then
1264: C
1265:      if ( NARG(ISTF).ne.NFARG(KF) ) then
1266:      if ( JSILNT.eq.0 ) then
1267:      write(IUPR,*)
1268:      &      'XXX(CALCULATOR) Function <', FUNC(KF), '> requires ',
1269:      &      NFARG(KF), ' arguments but given ', NARG(ISTF)
1270:      end if
1271:      IERR      = 1
1272:      go to 9000
1273:      end if
1274:      else if ( NFARG(KF).lt.0 ) then
1275:      if ( NARG(ISTF).lt.ABS(NFARG(KF)) ) then
1276:      if ( JSILNT.eq.0 ) then
1277:      write(IUPR,*)
1278:      &      'XXX(CALCULATOR) Function ', FUNC(KF),
1279:      &      ' requires at least ', ABS(NFARG(KF)),
1280:      &      ' arguments but given only ', NARG(ISTF)
1281:      end if
1282:      IERR      = 1
1283:      go to 9000
1284:      end if
1285:      end if
1286: Cjuly95
1287:      if(ISTN-NARG(ISTF).lt.0) then
1288:      if ( JSILNT.eq.0 ) then
1289:      write(IUPR,*)
1290:      &      'XXX(CALCULATOR) FUNCTION: stack conflict.'
1291:      end if
1292:      IERR      = 1
1293:      go to 9000
1294:      end if
1295: C
1296:      if ( JCHECK.ne.0 ) then
1297:      DA = 0.0
1298:      go to 400
1299:      end if
1300: C

```

src/utlils/dentak.f

```

1301:      if ( KF.eq.1 ) then
1302:          DA      = EXP(STK(ISTN))
1303:      else if ( KF.eq.2 ) then
1304:          if ( STK(ISTN).le.0.0 ) then
1305:              if ( JSILNT.eq.0 ) then
1306:                  write(IUPR,*)
1307:              &          'XXX(CALCULATOR) Non-positive argument for log ',
1308:              &          STK(ISTN)
1309:          end if
1310:          IERR      = 2
1311:          go to 9000
1312:      end if
1313:      DA      = LOG(STK(ISTN))
1314:      else if ( KF.eq.3 ) then
1315:          DA      = INT(STK(ISTN))
1316:      else if ( KF.eq.4 ) then
1317:          DA      = SIN(STK(ISTN))
1318:      else if ( KF.eq.5 ) then
1319:          DA      = COS(STK(ISTN))
1320:      else if ( KF.eq.6 ) then
1321:          DA      = TAN(STK(ISTN))
1322:      else if ( KF.eq.7 .or. KF.eq.8 ) then
1323:          if ( ABS(STK(ISTN)).gt.1.0 ) then
1324:              if ( JSILNT.eq.0 ) then
1325:                  write(IUPR,*)
1326:              &          'XXX(CALCULATOR) Argument invalid',
1327:              &          ' in ARCSIN,ARCCOS ', STK(ISTN)
1328:          end if
1329:          IERR      = 2
1330:          go to 9000
1331:      end if
1332:      if ( KF.eq.7 ) DA      = ASIN(STK(ISTN))
1333:      if ( KF.eq.8 ) DA      = ACOS(STK(ISTN))
1334:      else if ( KF.eq.9 ) then
1335:          DA      = ATAN(STK(ISTN))
1336:      else if ( KF.eq.10 ) then
1337:          if ( STK(ISTN).lt.0.0 ) then
1338:              if ( JSILNT.eq.0 ) then
1339:                  write(IUPR,*) 'XXX(CALCULATOR)',
1340:              &          ' Negative argument in squareroot ', STK(ISTN)
1341:          end if
1342:          IERR      = 2
1343:          go to 9000
1344:      end if
1345:      DA      = SQRT(STK(ISTN))
1346:      else if ( KF.eq.11 ) then
1347:          DA      = ABS(STK(ISTN))
1348:      else if ( KF.eq.12 ) then
1349:          DA      = STK(ISTN-NARG(ISTF)+1)
1350:          do 320 I = ISTN - NARG(ISTF) + 2, ISTN
1351:              DA      = MAX(DA,STK(I))
1352: 320      continue
1353:      else if ( KF.eq.13 ) then
1354:          DA      = STK(ISTN-NARG(ISTF)+1)
1355:          do 330 I = ISTN - NARG(ISTF) + 2, ISTN
1356:              DA      = MIN(DA,STK(I))
1357: 330      continue
1358:      else if ( KF.eq.14 ) then
1359:          DA      = MOD(STK(ISTN-1),STK(ISTN))
1360:      else if ( KF.eq.15 ) then
1361:          DA      = LOG10(STK(ISTN))
1362:      else if ( KF.eq.16 ) then
1363:          DA      = SIN(DPAI*STK(ISTN))
1364:      else if ( KF.eq.17 ) then
1365:          DA      = COS(DPAI*STK(ISTN))

```

```

1366:      else if ( KF.eq.18 ) then
1367:          DA      = TAN(DPAI*STK(ISTN))
1368: C
1369: C      .... functions for logical operation ...
1370: C
1371: C      &          'IF      ','AND      ','OR      ','
1372: C      &          'NOT      ','EQ      ','NE      ','GT      ','GE      ','
1373: C      &          'LT      ','LE      '/'
1374: C
1375: C      ... IF ...
1376:      else if ( KF.eq.19 ) then
1377:          if ( STK(ISTN-2).ne.0.0 ) then
1378:              DA      = STK(ISTN-1)
1379:          else
1380:              DA      = STK(ISTN)
1381:          end if
1382: C      ... AND ...
1383:      else if ( KF.eq.20 ) then
1384:          DA      = 1
1385:          do 340 I = ISTN - NARG(ISTF) + 1, ISTN
1386:              if ( STK(I).eq.0.0D0 ) DA      = 0.0
1387: 340      continue
1388: C      ... OR ...
1389:      else if ( KF.eq.21 ) then
1390:          DA      = 0
1391:          do 350 I = ISTN - NARG(ISTF) + 1, ISTN
1392:              if ( STK(I).ne.0.0D0 ) DA      = 1.0
1393: 350      continue
1394: C      ... NOT ...
1395:      else if ( KF.eq.22 ) then
1396:          DA      = 0
1397:          if ( STK(ISTN).eq.0.0D0 ) DA      = 1
1398: C      ... EQ ...
1399:      else if ( KF.eq.23 ) then
1400:          DA      = 0
1401:          if ( STK(ISTN-1).eq.STK(ISTN) ) DA      = 1
1402: C      ... NE ...
1403:      else if ( KF.eq.24 ) then
1404:          DA      = 0
1405:          if ( STK(ISTN-1).ne.STK(ISTN) ) DA      = 1
1406: C      ... GT ...
1407:      else if ( KF.eq.25 ) then
1408:          DA      = 1
1409:          do 360 I = ISTN - NARG(ISTF) + 2, ISTN
1410:              if ( STK(I-1).le.STK(I) ) DA      = 0
1411: 360      continue
1412: C      ... GE ...
1413:      else if ( KF.eq.26 ) then
1414:          DA      = 1
1415:          do 370 I = ISTN - NARG(ISTF) + 2, ISTN
1416:              if ( STK(I-1).lt.STK(I) ) DA      = 0
1417: 370      continue
1418: C      ... LT ...
1419:      else if ( KF.eq.27 ) then
1420:          DA      = 1
1421:          do 380 I = ISTN - NARG(ISTF) + 2, ISTN
1422:              if ( STK(I-1).ge.STK(I) ) DA      = 0
1423: 380      continue
1424: C      ... LE ...
1425:      else if ( KF.eq.28 ) then
1426:          DA      = 1
1427:          do 390 I = ISTN - NARG(ISTF) + 2, ISTN
1428:              if ( STK(I-1).gt.STK(I) ) DA      = 0
1429: 390      continue
1430: C

```

src/utlils/dentak.f

```

1431: C ... nint
1432:     else if ( KF.eq.29 ) then
1433:         DA      = NINT(STK(ISTN))
1434: C
1435: C ... atan2
1436:     else if ( KF.eq.30 ) then
1437:         DA      = ATAN2(STK(ISTN-1),STK(ISTN))
1438: C
1439: C ... sinh
1440:     else if ( KF.eq.31 ) then
1441:         DA      = sinh(STK(ISTN))
1442: C
1443: C ... cosh
1444:     else if ( KF.eq.32 ) then
1445:         DA      = cosh(STK(ISTN))
1446: C
1447: C ... tanh
1448:     else if ( KF.eq.33 ) then
1449:         DA      = tanh(STK(ISTN))
1450:     end if
1451: C
1452: C
1453: Ccccc   istn = istn - narg(istf)
1454: C
1455: 400     ISTN     = ISTN - NARG(ISTF) + 1
1456:         STK(ISTN) = DA
1457:         ISTF     = ISTF - 1
1458: C
1459:         ISTO     = ISTO - 1
1460:     end if
1461: C
1462:     go to 280
1463: C
1464: C ..... POP operator stack .....
1465: C
1466: 333 ISTO     = ISTO - 1
1467:     IS       = IEO + 1
1468: C
1469: C ... increment function argument counter here (July 1995)
1470: C
1471:     if( istf.gt.0.and.isto.gt.0 .and. istk(isto).lt.0 ) then
1472:         NARG(ISTF) = NARG(ISTF) + 1
1473:     end if
1474:     go to 1000
1475: C
1476: C ..... TERMINATE CALCULATION .....
1477: C
1478: 444 go to 470
1479: C
1480: C ..... SYNTAX ERROR !! .....
1481: C
1482: 555 if ( JSILNT.eq.0 ) then
1483:     write(IUPR,*) 'XXX(CALCULATOR) SYNTAX ERROR:',
1484:     & ' operator conflict.'
1485:     end if
1486:     IERR     = 1
1487:     go to 9000
1488: C
1489: C
1490: 666 if ( ISTO.gt.1.and.ISTK(ISTO-1).ge.0 ) then
1491:     if ( JSILNT.eq.0 ) then
1492:         write(IUPR,*)
1493:         & 'XXX(CALCULATOR) SYNTAX ERROR: misused comma!!'
1494:     end if
1495:     IERR     = 1

```

```

1496:         go to 9000
1497:     else
1498:         NARG(ISTF) = NARG(ISTF) + 1
1499:     end if
1500:     IS       = IEO + 1
1501:     go to 1000
1502: C
1503: C ... end of a comma separated statement block
1504: C
1505: 777 ISTN     = ISTN - 1
1506:     if ( ISUBN.le.ISUBN0 ) then
1507:         write(IUPR,*)
1508:         & 'XXX(CALCULATOR)',
1509:         & ' A comma separated statement block has no parameter definition.'
1510:         write(IUPR,*) ' Probably a misuse of "," for "." ...'
1511:         IERR     = 1
1512:         go to 9000
1513:     end if
1514:     IS       = IEO + 1
1515:     ISTMT    = ISTMT + 1
1516:     ISUBN0   = ISUBN
1517:     go to 1000
1518: C
1519: C
1520: C ... unary operators ...
1521: C
1522: 888 if ( ISTK(ISTO).eq.KUNMNS ) then
1523:     STK(ISTN) = -STK(ISTN)
1524:     ISTO      = ISTO - 1
1525:     go to 280
1526: C
1527: C ..... substitution ...
1528: C
1529:     else if ( ISTK(ISTO).eq.KEQUAL ) then
1530:         if ( JCHECK.eq.0 ) then
1531:             VALUE(LHSTK(ISTLH)) = STK(ISTN)
1532:             JPSTAT(LHSTK(ISTLH)) = 1
1533:         end if
1534:         ISTLH = ISTLH - 1
1535:         ISTO  = ISTO - 1
1536:         go to 280
1537:     end if
1538: C
1539: C ..... END OF PROCESSING
1540: C
1541: 470 DENTK0 = STK(ISTN)
1542: C
1543:     if ( JECHO.eq.1.and.(ISTKM.gt.1.or.IPFL.gt.0) ) then
1544:         write(IECHO,('( ' ',A)') LINE(IST:IEEE)
1545:         do 480 I = 1, ISTN
1546:             write(IECHO,'(1x,a,1p,d22.15,1x,e15.7)')
1547:             & ' ==> ', STK(I), REAL(STK(I))
1548: 480     continue
1549:     end if
1550: C
1551: C =====
1552: C
1553:     if ( JCHECK.ne.0 ) then
1554:         ISTMT0 = ISTMT
1555:         IPFL0  = IPFL
1556:         ISTKM0 = ISTKM
1557:         IVECN0 = IVECN
1558: C
1559: Ccccccccc CDENTK = 0.0d0
1560: C

```

src/utls/dentak.f

```

1561:      end if
1562:      return
1563: C
1564: C ..... ERROR : STACK IS FULL
1565: C
1566:      490 if ( JSILNT.eq.0 ) then
1567:          write(IUPR,*)
1568:          & 'XXX(CALCULATOR) Stack is full XXX '
1569:          end if
1570:          IERR = 4
1571:          go to 9000
1572: C
1573: C ..... PRINTOUT POSITION OF ERROR .....
1574: C
1575:      9000 if ( JSILNT.eq.0 ) then
1576:          LK = LEN(LINE)
1577:          do 510 I = LEN(LINE), 1, -1
1578:              if ( LINE(I:I).ne.' ' ) then
1579:                  LK = I
1580:                  go to 520
1581:              end if
1582:          510 continue
1583:          520 write(IUPR,'(1X,' LINE: <'',A,'>')' ) LINE(:LK)
1584:          FORMT = ' '
1585:          if ( IS.gt.NL ) then
1586:              IPP = NL + 1
1587:              ILL = 1
1588:          else
1589:              IPP = IPLIN(IS)
1590:              ILL = MAX(1,IPLIN(IEO)-IPLIN(IS)+1)
1591:          end if
1592:          write(FORMT,'(''(1X,T'',I3,'',''',I3,'(''*****')')')')
1593:          & 9 + IPP, ILL
1594:          write(IUPR,fmt =FORMT)
1595:          end if
1596:          return
1597:          end
1598: C
1599: C
1600:      subroutine DVCHAR( TVALUE,VCHAR, LLCHAR,IERR )
1601: C=====
1602: C Purpose: convert a double precision numeric value to a numeric
1603: C          character string as short as possible keeping 15 digit precision.
1604: C Called in: everywhere
1605: C Calls: none
1606: C History: extracted from DENTAK routine on 12 July 1998
1607: C Update:
1608: C 30 Aug 1998: "character(*) VCHAR" "character(*) VCHAR"
1609: C-----
1610: C arguments ( i=input, o=output, w=work)
1611: C i TVALUE : a double precision floating point data
1612: C o VCHAR(*) : a character string in which translated string is stored.
1613: C          << Must be longer than 15+8 characters >>
1614: C o LLCHAR : effective length of translated string.
1615: C o IERR : returns non-zero value if error occurred.
1616: C=====
1617:      real*8 TVALUE
1618:      character*(*) VCHAR
1619: C
1620: C ..... local variables
1621: C
1622:      character*8 CHARW
1623: C
1624: C-----
1625: C
1626:      IERR = 0
1627:      VCHAR = ' '
1628:      LLCHAR = 0
1629: C
1630:      write(VCHAR(LLCHAR+1:),fmt ='(1P,D23.15)',iostat =IOS) TVALUE
1631:      if ( IOS.ne.0 ) then
1632:          IERR = 9
1633:          return
1634:      end if
1635: C
1636: C ... llchar : effective length of character string
1637: C ... lesig : the last occurrence position of +- sign
1638: C
1639:      LESIG = 0
1640:      do 100 I = LLCHAR + 1, LEN(VCHAR)
1641:          if ( VCHAR(I:I).ne.' ' ) then
1642:              LLCHAR = LLCHAR + 1
1643:              CHARW(1:1) = VCHAR(I:I)
1644:              if ( CHARW(1:1).eq.'E' .or. CHARW(1:1).eq.'e'
1645:                  & .or. CHARW(1:1).eq.'d' ) then
1646:                  CHARW(1:1) = 'D'
1647:              end if
1648:              VCHAR(LLCHAR:LLCHAR) = CHARW(1:1)
1649:              if ( INDEX('+-',CHARW(1:1)).ne.0 ) LESIG = LLCHAR
1650:          end if
1651:      100 continue
1652:      if ( LLCHAR.lt.LEN(VCHAR) ) VCHAR(LLCHAR+1:) = ' '
1653:      KD = INDEX(VCHAR,'D')
1654: C
1655: C ... missing D or E ...
1656:      if ( KD.eq.0 ) then
1657:          CHARW = VCHAR(LESIG:LLCHAR)
1658:          VCHAR(LESIG:LESIG) = 'D'
1659:          VCHAR(LESIG+1:LLCHAR+1) = CHARW
1660:          KD = LESIG
1661:          LLCHAR = LLCHAR + 1
1662:      end if
1663: C
1664: C ... check exponent is zero or not
1665: C
1666:      do 110 I = KD + 1, LLCHAR
1667:          if ( INDEX('123456789',VCHAR(I:I)).ne.0 ) go to 120
1668:      110 continue
1669: C
1670:      LLCHAR = KD - 1
1671:      KD = LLCHAR + 1
1672: C
1673:      120 LLNE = KD - 1
1674: C
1675: C ... check non-zero part of fraction part
1676: C
1677:      do 130 I = KD - 1, 1, -1
1678:          if ( VCHAR(I:I).ne.'0' ) then
1679:              LLNE = I
1680:              if ( VCHAR(I:I).eq.'.' ) LLNE = LLNE - 1
1681:              go to 140
1682:          end if
1683:      130 continue
1684: C
1685:      140 if ( KD.gt.LLCHAR ) then
1686:          VCHAR(LLNE+1:) = ' '
1687:      else
1688:          CHARW = VCHAR(KD:LLCHAR)
1689:          VCHAR(LLNE+1:) = ' '
1690:          VCHAR(LLNE+1:) = CHARW

```


src/utils/dentak.f

```

1691:      end if
1692: C
1693:      LLCHAR = INDEX(VCHAR,' ') - 1
1694:      if ( LLCHAR.lt.0 ) LLCHAR = LEN(VCHAR)
1695: C
1696:      return
1697:      end
1698: C
1699:      subroutine EVCHAR( TVALUE,VCHAR, LLCHAR,IERR )
1700: C=====
1701: C Purpose: convert a single precision numeric value to a numeric
1702: C character string as short as possible keeping 8 digit precision.
1703: C << This is a single precision version of DVCHAR above >>
1704: C=====
1705:      real TVALUE
1706:      character*(*) VCHAR
1707: C
1708: C ..... local variables
1709: C
1710:      character*8 CHARW
1711: C
1712: C-----
1713: C
1714:      IERR = 0
1715:      VCHAR = ' '
1716:      LLCHAR = 0
1717: C
1718:      write(VCHAR(LLCHAR+1:),fmt='(1P,e16.7)',iostat=IOS) TVALUE
1719:      if ( IOS.ne.0 ) then
1720:          IERR = 9
1721:          return
1722:      end if
1723: C
1724: C ... llchar : effective length of character string
1725: C ... lesig : the last occurrence position of +- sign
1726: C
1727:      LESIG = 0
1728:      do 100 I = LLCHAR + 1, LEN(VCHAR)
1729:          if ( VCHAR(I:I).ne.' ' ) then
1730:              LLCHAR = LLCHAR + 1
1731:              CHARW(1:1) = VCHAR(I:I)
1732:              if ( CHARW(1:1).eq.'D' .or. CHARW(1:1).eq.'e'
1733:                  & .or. CHARW(1:1).eq.'d' ) then
1734:                  CHARW(1:1) = 'E'
1735:              end if
1736:              VCHAR(LLCHAR:LLCHAR) = CHARW(1:1)
1737:              if ( INDEX('+-',CHARW(1:1)).ne.0 ) LESIG = LLCHAR
1738:          end if
1739:      100 continue
1740:      if ( LLCHAR.lt.LEN(VCHAR) ) VCHAR(LLCHAR+1:) = ' '
1741:      KD = INDEX(VCHAR,'E')
1742: C
1743: Cc ..... missing D or E ...
1744:      if ( KD.eq.0 ) then
1745:          CHARW = VCHAR(LESIG:LLCHAR)
1746:          VCHAR(LESIG:LESIG) = 'E'
1747:          VCHAR(LESIG+1:LLCHAR+1) = CHARW
1748:          KD = LESIG
1749:          LLCHAR = LLCHAR + 1
1750:      end if
1751: C
1752: C ..... check exponent is zero or not
1753: C
1754:      do 110 I = KD + 1, LLCHAR
1755:          if ( INDEX('123456789',VCHAR(I:I)).ne.0 ) go to 120
1756:      110 continue
1757: C
1758:      LLCHAR = KD - 1
1759:      KD = LLCHAR + 1
1760: C
1761:      120 LLNE = KD - 1
1762: C
1763: C .... check non-zero part of fraction part
1764: C
1765:      do 130 I = KD - 1, 1, -1
1766:          if ( VCHAR(I:I).ne.'0' ) then
1767:              LLNE = I
1768:              if ( VCHAR(I:I).eq.'.' ) LLNE = LLNE - 1
1769:              go to 140
1770:          end if
1771:      130 continue
1772: C
1773:      140 if ( KD.gt.LLCHAR ) then
1774:          VCHAR(LLNE+1:) = ' '
1775:      else
1776:          CHARW = VCHAR(KD:LLCHAR)
1777:          VCHAR(LLNE+1:) = ' '
1778:          VCHAR(LLNE+1:) = CHARW
1779:      end if
1780: C
1781:      LLCHAR = INDEX(VCHAR,' ') - 1
1782:      if ( LLCHAR.lt.0 ) LLCHAR = LEN(VCHAR)
1783: C
1784:      return
1785:      end

```

src/utls/envexp.f

```

1:      subroutine ENVEXP( STRING,EXPAND,IOPR,  IERR )
2: C=====
3: C Purpose: expand string "$xxxx" by corresponding environment variable
4: C         if possible.
5: C
6: C  "xxxx" is a string composed of alphanumeric characters and
7: C  underscore.
8: C
9: C -----
10: C <arguments> (i=input, o=output)
11: C
12: C i STRING : input string
13: C o EXPAND : expanded string
14: C i IOPR : message printout I/O unit
15: C o IERR : non-zero return value on error.
16: C=====
17:      character*(*) STRING
18:      character*(*) EXPAND
19: c
20:      character*256 CWRK
21:      character CHARS*63
22: c
23: c -----
24: c
25:      CHARS = 'abcdefghijklmnopqrstuvwxyz'//
26:      &      'ABCDEFGHIJKLMNOPQRSTUVWXYZ'//'/0123456789'//'_-'
27: c
28:      IERR = 0
29:      IP = 1
30:      IE = 1
31: c
32: 100 IK = INDEX(STRING(IP:),'$')
33:      if ( IK.gt.0 ) then
34:          if ( IK.gt.1 ) then
35:              EXPAND(IE:) = STRING(IP:IP+IK-2)
36:              IE = IE + IK - 1
37:          end if
38:          IP = IP + IK
39:          do 110 IC = IP, LEN(STRING)
40:              if ( INDEX(CHARS,STRING(IC:IC)).eq.0 ) then
41:                  go to 120
42:              end if
43: 110      continue
44: 120      continue
45:              if ( IC.gt.IP ) then
46:                  call ENVGET( STRING(IP:IC-1), CWRK )
47:                  if ( CWRK.eq.' ' ) then
48:                      IERR = IERR + 1
49:                      write(IOPR,*) '!!!(ENVEXP) Environment variable <',
50:      &      STRING(IP:IC-1), '> is not defined.'
51:                  else
52:                      IJ = ICLEN2(CWRK)
53:                      EXPAND(IE:) = CWRK(:IJ)
54:                      IE = IE + IJ
55:                  end if
56:              else
57:                  IERR = IERR + 1
58:                  write(IOPR,*) '!!!(ENVEXP) No effective string after "$"'
59:              end if
60:              IP = IC
61:              if ( IP.le.LEN(STRING) ) go to 100
62:          else
63:              EXPAND(IE:) = STRING(IP:)
64:          end if
65: c
66:      return
67:      end

```

src/utils/envget.f

```
1:      subroutine ENVGET( EVAR, VALUE )
2:
3: C=<MVP>=====
4: C   Purpose: Check enviromnet variables
5: C=====
6: C   argument :
7: C
8: C i   EVAR : environment variable name
9: C o   VALUE : environment variable value. Return blank if EVAR is
10: C      not defined
11: C
12: C=====
13: C/#IF CUTIL.AND.MS_VISUAL
14: C
15: C ... This interface block is for CUTIL & MS-Visual tools. ...
16: C
17: *   interface
18: *       subroutine FTGETENV( EVAR, LL, VALUE, LEN )
19: *           character EVAR
20: *           character VALUE
21: *           integer LL
22: *           integer LEN
23: *CDEC$   ATTRIBUTES C :: FTGETENV
24: *CDEC$   ATTRIBUTES REFERENCE :: EVAR, VALUE, LL, LEN
25: *       end subroutine
26: *   end interface
27: C/#ENDIF
28: C
29:       character*(*) EVAR
30:       character*(*) VALUE
31: C
32: C/#IF NOGETENV
33: C
34:       VALUE = ' '
35: C
36: C/#ELSE
37: C
38:       VALUE = ' '
39:       LL = ICLEN2(EVAR)
40: C
41: C/# IF SYSTEM( CRAY* )
42: *       call PXFGETENV( EVAR(:LL), LL, VALUE, LLVAL, IERR )
43: C/# ELSE MS_VISUAL.AND. .NOT.CUTIL
44: *       call GETENV(EVAR(:LL), VALUE)
45: C/# ELSE
46: C/# IF PASSSTRING( NONULL )
47: *       call FTGETENV( EVAR(:LL), LL, VALUE, LEN(VALUE) )
48: C/# ELSE
49: *       call FTGETENV( EVAR(:LL)//char(0), LL, VALUE, LEN(VALUE) )
50: C/# ENDIF
51: C/# ENDIF
52:
53: C/#ENDIF
54: C
55:       return
56:       end
```

src/utils/erf.f

```

1: C=====
2: C Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
3: C-----
4: C .... error function prepared for
5: C systems which does not have it as an intrinsic function.
6: C
7: C Reference: "Numeric Calculation Software with Fortran77,"
8: C T.Watanabe,M.Natori,T.Oguni ed., Maruzen, 1989 (in Japanese)
9: C
10: C-----
11: C
12: C .... single precision ....
13: C
14: C function ERF(B)
15: C real*8 A, ERFD
16: C A = B
17: C ERFD = ERFD(A)
18: C return
19: C end
20: C
21: C
22: C .... DOUBLE PRECISION ERROR FUNCTION :
23: C
24: C
25: C function ERFD(X)
26: C
27: C by calculating the incomplete gamma function.
28: C
29: C / x
30: C erf(x) = 2/sqrt(PI)* | exp( -t**2 ) dt
31: C / 0
32: C
33: C GH = GAMMA(1/2) = SQRT(PI)
34: C
35: C implicit real*8(A-H,O-Z)
36: C parameter( GH = 1.7724538509055159D0 )
37: C parameter( RH = 1.4142135623730951D0 )
38: C parameter( GR = 1.0D0/(RH*GH) )
39: C
40: C X2 = X*X
41: C
42: C if ( X2.le.1.0 ) then
43: C A0 = 1.0
44: C B0 = 1.0
45: C do 100 I = 1, 20
46: C A0 = A0*X2/(0.5D0+DBLE(I))
47: C B0 = B0 + A0
48: C if ( A0.le.1.0D-18 ) go to 110
49: C 100 continue
50: C 110 ERFD = 2.0D0*X*EXP(-X2)*B0/GH
51: C return
52: C else if ( ABS(X).gt.10.0 ) then
53: C ERFD = 1.0D0
54: C return
55: C else
56: C IT = 80.0D0/X2 + 10.D0
57: C A0 = 0.0
58: C A1 = 1.0D-30
59: C B0 = A1
60: C B1 = X2*B0
61: C do 120 I = 1, IT
62: C
63: C c = a1/b1
64: C
65: C A = DBLE(I) - 0.5D0

```

```

66: C A0 = A1 + A*A0
67: C B0 = B1 + A*B0
68: C A1 = X2*A0 + I*A1
69: C B1 = X2*B0 + I*B1
70: C if ( I.gt.50 ) then
71: C A0 = A0/I
72: C B0 = B0/I
73: C A1 = A1/I
74: C B1 = B1/I
75: C end if
76: C 120 continue
77: C
78: C 130 ERFD = 1.0 - ABS(X)*EXP(-X2) /B1*A1/GH
79: C if ( X.lt.0.0 ) ERFD = -ERFD
80: C
81: C write(6,*) i
82: C
83: C return
84: C end if
85: C end

```

src/utils/error.f

```
1:      subroutine error
2: c
3: c
4:      write(6,*) ' XXXX ERROR STOP XXXX (ERROR ROUTINE CALLED) '
5:      stop
6:      end
```

SAFE

src/utls/flushstd.c

```
1: /*****
2: *
3: *   To flush standard output by SIGINT ( kill -INT pid etc.)
4: *
5: *   Programmed by M.Sasaki ( The Japan Research Inst. Co. Ltd.)
6: *
7: *   fsigset should be called somewhere to enable interrrupt.
8: *   fsigset is disabled in ANSI-C mode.
9: *
10: *****/
11:
12: #include <stdio.h>
13:
14: #ifndef ANSI_C
15: #define SIGINT 0
16: #define signal(a,b) " "
17: #else
18: #include <signal.h>
19: #endif
20:
21: /*****
22: *   Fortran called C subroutine :
23: *
24: *   UNIX venders have their own convention or rule for external
25: *   names of FORTRAN coded subprograms or commons.
26: *
27: *   We prepare equivalent sources for C coded routines
28: *
29: *   1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
30: *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
31: *   3. Uppercase string of FORTRAN name. ( Ncube etc. )
32: *   4. "_" + uppercase string of FORTRAN name. ( HI/OSF. )
33: *
34: *****/
35:
36: void fsigset_( )
37: {
38:     void FLUSH_STD ( ) ;
39:     signal( SIGINT, FLUSH_STD ) ;
40: }
41: void fsigset ( )
42: {
43:     void FLUSH_STD ( ) ;
44:     signal( SIGINT, FLUSH_STD ) ;
45: }
46: void FSIGSET ( )
47: {
48:     void FLUSH_STD ( ) ;
49:     signal( SIGINT, FLUSH_STD ) ;
50: }
51: void _FSIGSET ( )
52: {
53:     void FLUSH_STD ( ) ;
54:     signal( SIGINT, FLUSH_STD ) ;
55: }
56:
57: void FLUSH_STD ( sig )
58:     int sig ;
59: {
60:     fflush( stdout ) ;
61:     signal( SIGINT, FLUSH_STD ) ;
62: }
63:
64: /*****
65: *   flushstd called directly from FORTRAN program.
```

```
66: *****/
67:
68: void flushstd_( )
69: {
70:     fflush( stdout ) ;
71: }
72: void flushstd()
73: {
74:     fflush( stdout ) ;
75: }
76: void FLUSHSTD()
77: {
78:     fflush( stdout ) ;
79: }
80: void _FLUSHSTD()
81: {
82:     fflush( stdout ) ;
83: }
```

src/utils/fread.f

```

1: *VOCL TOTAL,SCALAR
2: C
3: C  JAERI MONTE CARLO CODE UTILITY
4: C
5: C=<MVP/GMVP>=====
6: C
7: C  Routines for MVP/GMVP type input-data handling.
8: C
9: C      FREADN,FREADS,FREADL,FREADB,FPROBE  (search data name etc.)
10: C      FREAD0  (real body of FREADN,FREADS,FREADL & FREADB)
11: C
12: C      I4READ,R4READ,R8READ,I8READ,DMREAD  (input numerical data array)
13: C      VRREAD0  (real body of I4READ,R4READ,R8READ,I8READ & DMREAD)
14: C
15: C      GETSTR, CHREAD  (input character string)
16: C      RESET, ENTRY RIUNIT  (input file status handling)
17: C      VIFILE, ENTRY VIFURG  ("virtual file" handling)
18: C      NFIMP <FUNCTION>  (check number of data in the last input)
19: C      READI4  (convert a character string to a number(integer))
20: C      READR4  (convert a character string to a number(real))
21: C      FRELIN  (print invalid input line)
22: C
23: C  Related routines:
24: C
25: C      GETLIN      : get a line from input file
26: C      PARA        : symbolic parameter line handling
27: C      DENTAK (function) : calculator
28: C      ICLEN2 (function) : get position of last non-blank character
29: C
30: C  Routines to be supplied by user to use FREAD routines:
31: C
32: C      * subroutine GTVVAL(NAME, VALUE, IERR):
33: C          character*(*) NAME
34: C          real*8      VALUE
35: C          integer      IERR
36: C
37: C      Return a value of of specified NAME on VALUE.
38: C      If an error ocured, a non-zero IERR were returned.
39: C
40: C      * subroutine CNTERR(TYPE):
41: C          character*(*) TYPE
42: C
43: C      Count number of errors of type TYPE.
44: C      Current effective error type is 'FATAL', 'WARNING' and 'MESSAGE'.
45: C
46: C      * subroutine PRSTOP(IECNT, MESSAGE):
47: C          integer      IECNT
48: C          character*(*) MESSAGE
49: C
50: C      Stop current program with message string MESSAGE.
51: C      IECNT can be used to count 'FATAL' error IECNT times in PRSTOP
52: C      routine.
53: C
54: C  Global (COMMON) data :
55: C
56: C      COMMON /FRDATA/ : common data for these routines (numeric data)
57: C      COMMON /FRDATC/ : common data for these routines (character data)
58: C      COMMON /VFREAD/ COMMON /VFILE/ : virtual file
59: C
60: C=====
61: C
62: C
63: C
64: C
65: C      subroutine FRINIT( IPPIN, IPPR, IMXERR )

```

```

66: C
67: C=====
68: C  FRINIT: initialize static data shared among 'FREAD' group of routines.
69: C=====
70: C  i IPPIN : data input I/O unit set as default.
71: C  i IPPR  : message printout I/O unit set as default.
72: C  i IMXERR : allowable number of fatal input data error set to MXCERR.
73: C=====
74: C
75: C      common /FRDATA/ NAA, IS, IUIN, IUPR, NCHAR, MXCERR,
76: C      & NCERR
77: C
78: C      ..... INPUT BUFFER ....
79: C      MAXCOL : EFFECTIVE CHARACTERS IN AN INPUT-LINE
80: C
81: C      parameter( MAXCOL = 72 )
82: C      character LINE*(MAXCOL)
83: C
84: C      ..... CHARACTER LIST .... ( ALPHABETS & NUMBERS ) .....
85: C
86: C      character*46 ALPNUM
87: C      character*26 ALPLOW
88: C
89: C      common /FRDATC/ LINE, ALPNUM, ALPLOW
90: C
91: C      ..... data for GETLIN routine
92: C
93: C      GLPRPT : prompt string (effective until last non-blank character)
94: C      JPRPT : output prompt string (=1 with a blank, =2: no blank)
95: C      JECHO : =1 echoback input string (=2,with prompt)
96: C
97: C      common /GLDATC/ GLPRPT
98: C      character*32 GLPRPT
99: C      common /GLDATN/ JPRPT, JECHO
100: C-----
101: C
102: C      data ALPNUM /'!@#$%?:_-ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'/
103: C      data IS /0/, NCHAR /0/
104: C      data IUIN /5/
105: C      data LINE /' '/
106: C
107: C      ... initial value of allowable fatal error count & counter ...
108: C
109: C      data MXCERR /10/
110: C      data NCERR /0/
111: C
112: C      ALPNUM = '!@#$%?:_-ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
113: C      ALPLOW = 'abcdefghijklmnopqrstuvwxyz'
114: C      IS = 0
115: C      NCHAR = 0
116: C      IUIN = IPPIN
117: C      IUPR = IPPR
118: C      LINE = ' '
119: C
120: C      MXCERR = IMXERR
121: C      NCERR = 0
122: C
123: C      GLPRPT = ' '
124: C      JPRPT = 0
125: C      JECHO = 0
126: C
127: C      return
128: C      end
129: C
130: C

```

```

=====
LEADS:
Like 'FREADN', but users does not need to specify length of VNAME.
Returns number of characters stored in VNAME on NLEN.
( never pass non-character variable as VNAME !! )
=====
subroutine FREADS( VNAME, NLEN, IEND )

character*(*) VNAME

JBLANK = 0
call FREAD0( VNAME, 'FREADS', NLEN, JBLANK, IEND )
if ( IEND.ne.0 ) NLEN = 0
return
end

=====
LEADL:
Like 'FREADS', but accept lowercase character in VNAME
=====
subroutine FREADL( VNAME, NLEN, IEND )

character*(*) VNAME

JBLANK = 0
call FREAD0( VNAME, 'FREADL', NLEN, JBLANK, IEND )
if ( IEND.ne.0 ) NLEN = 0
return
end

=====
LEADB:
Like 'FREADS', but allow leading blank characters and fill rest of line
with asterisks.
=====

```

```
195:      if ( IEND.ne.0 ) NLEN = 0
```

[illegible]

src/utls/fread.f

```

261: Cccccccc if ( IEND.eq.1 ) go to 160
262:         if ( IEND.ne.0 ) go to 160
263: C
264: C .... SPECIAL TREATMENT FOR FREADB ....
265: C
266:         if ( LINE(1:NCHAR).eq.' '.and.JBLANK.ne.0 ) then
267:             if ( IEND.eq.0 ) IEND = 1
268:             return
269:         end if
270:         IS = 1
271:     end if
272: C
273: C
274: C ... VARIABLE NAMES MUST BEGIN WITH AN ALPHABET OR CHARACTERS
275: C '!', '@', '$', '&', '#', '*' OR '?'.
276: C
277: C
278:         LLP0 = INDEX(ALPNUM,'0') - 1
279: C
280: C (modified on 11 Oct 1998: warning for non-data-name string)
281: C
282:         I = IS
283: 110 continue
284:         if ( I.le. NCHAR ) then
285:             if ( INDEX(ALPNUM(:LLP0),LINE(I:I)).ne.0.or.
286:             & (ENAME.eq.'FREADL'.and.INDEX(ALPLOW,LINE(I:I)).ne.0) ) then
287:                 go to 120
288:             else if ( LINE(I:I).eq.'/' ) then
289: C
290: C ..... STOP INPUT IF / IS FOUND.
291: C
292:                 IEND = 1
293:                 IS = 0
294:                 return
295: C
296: C ... string ignored in FREAD* ...
297: C
298:             else if ( LINE(I:I).ne.' ' ) then
299:                 I2 = INDEX(LINE(I:NCHAR),' ')
300:                 if( I2.ne.0 ) then
301:                     I2 = I + I2 - 1
302:                 else
303:                     I2 = NCHAR
304:                 end if
305:                 write(IUPR,'(1X,A,A,A,A)')
306:             & '!!!('FREAD0) string <' ,LINE(I:I2), '> ',
307:             & 'is not a "data name" string. Ignored.'
308:                 call FRELIN( IUPR, LINE, I, I2 )
309:                 call CNTERR( 'WARNING' )
310:                 I = I2
311:             end if
312:             I = I + 1
313:             go to 110
314:         end if
315: C
316: C ..... GET THE TAIL POSITION OF VARIABLE NAME .....
317: C
318: 120 if ( I.le.NCHAR ) then
319:         IS = I
320:         do 130 IE = IS, NCHAR
321:             if ( INDEX(' ',LINE(IE:IE)).ne.0 ) go to 140
322: 130 continue
323: 140 IEE = IE
324: C
325: C ---- REMOVE BLANK FROM LINE(IS:IEE) AND STORE IN VNAME. ---

```

```

326: C
327:         VNAME = ' '
328:         NLENA = 0
329:         do 150 I = IS, IE - 1
330:             if ( LINE(I:I).ne.' ' ) then
331:                 NLENA = NLENA + 1
332:                 if ( NLENA.le.LN ) VNAME(NLENA:NLENA) = LINE(I:I)
333:             end if
334: 150 continue
335: C
336:         if ( NLENA.gt.LN ) then
337:             write(IUPR,'(1X,A,A,A,I4,A,A,A,A,A)') '!!!(' , ENAME,
338:             & ' ) VARIABLE LENGTH SHOULD BE LESS THAN ', LN + 1,
339:             & ' ', LINE(IS:IE-1), ' IS TAKEN AS <',
340:             & LINE(IS:IS+LN-1), '>'
341:             call FRELIN( IUPR, LINE, IS, IE-1 )
342:             call CNTERR( 'WARNING' )
343: C
344:             IEE = IS + LN
345:             NLENA = LN
346:         end if
347: C
348:         IS = IE
349: C
350: C
351: C ..... REST OF THE LINE IS BLANK OR CAN'T BE INTERPRETED. ....
352: C
353: C
354:         else if ( I.gt.NCHAR ) then
355:             if ( LINE(IS:NCHAR).ne.' ' ) then
356:                 write(IUPR,'(1X,a,a,a)') '!!!(' , ENAME,
357:             & ' ) I CAN'T INTERPRET THIS LINE !'
358:                 write(IUPR,'(1X,a,a)') ' LINE: ', LINE
359:                 call CNTERR( 'WARNING' )
360:             end if
361: C
362:             IS = 0
363:         end if
364: C
365:         if ( IS.eq.0 ) go to 100
366:         if ( IS.gt.NCHAR ) IS = 0
367: C
368:         if ( ENAME.eq.'FREADS' .or. ENAME.eq.'FREADB'
369:         & .or. ENAME.eq.'FREADL' ) NLEN = NLENA
370: C
371:         return
372: C
373: C ..... THE END OF INPUT DATA IS ENCOUNTERED. ....
374: C
375: C 160 IEND = 1
376: 160 continue
377:         return
378:     end
379: C
380: C
381: C
382: C
383:         subroutine FPROBE( IPP, CHAR, CGET )
384: C=====
385: C FPROBE: To check the first character that is not included in
386: C 'CHAR' in the current input buffer 'LINE'(from IS). Returns
387: C characters begining with other characters than 'CHAR'
388: C in 'CGET' and returns position in the buffer on 'IPP'.
389: C IPP=0 if end of buffer.
390: C Buffer pointer 'IS' remains unchanged.

```

[illegible][illegible]

```
src/utls/fread.f
```

```

521: C
522: Cccc include 'INC/_IOUNIT'
523: C
524: Cccc ENAME = 'GETSTR'
525: ITERM = 0
526: NLEN = 0
527: LL = NLEN
528: C
529: MODEL = MODE
530: if ( INDEX(CHAR,' ').ne.0 ) MODEL = 0
531: JBLANK = 0
532: IERR = 0
533: IEND = 0
534: C
535: 100 continue
536: C
537: if ( IS.eq.0 ) then
538: call GETLIN( IUIN, IUPR, LINE, NCHAR, IEND )
539: if ( IEND.ne.0 ) return
540: IS = 1
541: end if
542: C
543: do 110 I = IS, NCHAR
544: C
545: KI = INDEX(CHAR,LINE(I:I))
546: C
547: if ( KI.ne.0 ) then
548: ITERM = KI
549: IS = I + 1
550: go to 120
551: else
552: if ( MODEL.eq.1 ) then
553: CM2016 if ( LINE(I:I).ne.' ' .or. LINE(I:I).eq.' '
554: CM & .and.JBLANK.eq.0 ) then
555: if ( LINE(I:I).ne.' ' .or.
556: & (LINE(I:I).eq.' ' .and.JBLANK.eq.0) ) then
557: if ( LINE(I:I).eq.' ' ) then
558: JBLANK = 1
559: else
560: JBLANK = 0
561: end if
562: NLEN = LL + 1
563: end if
564:
565: CM2016 if ( MODEL.eq.2 ) then
566: else if ( MODEL.eq.2 ) then
567: if ( LINE(I:I).ne.' ' ) NLEN = LL + 1
568: CM2016 end if
569: else
570: NLEN = LL + 1
571: end if
572:
573: if ( NLEN.gt.LL ) then
574: if ( NLEN.le.LEN(CGET) ) then
575: CGET(NLEN:NLEN) = LINE(I:I)
576: else
577: NLEN = NLEN - 1
578: IERR = 1
579: IS = I
580: go to 120
581: end if
582: end if
583: LL = NLEN
584: end if
585: 110 continue

```

[illegible]

```
src/utls/fread.f
```

[illegible]

```

716:      else
717:        if ( LINE(I:I).ne.QUOT ) then
718:          LL      = LL + 1
719:          if ( LL.le.LEN(CGET) ) then
720:            NLEN   = NLEN + 1
721:            CGET(NLEN:NLEN) = LINE(I:I)
722:          end if
723:        else
724:          IS      = I + 1
725:          go to 120
726:        end if
727:      end if
728: 110 continue
729:      IS      = 0
730:      if ( NLEN.eq.0 .or. JQUOT.ne.0 ) go to 100
731: C
732: 120 if ( LL.gt.NLEN ) then
733:   write(IUPR,'(1x,a,a,a/1x,a,i4,a,i4,a)')
734:   &      '!!!(CHREAD) IN CHARACTER STRING INPUT,',
735:   &      ' LENGTH OF INPUT STRING (', LL,
736:   &      ') EXCEEDS THAT OF CHARACTER VARIABLES (', LEN(CGET),
737:   &      ')'
738:   if ( JQUOT.ne.0 ) then
739:     write(IUPR,'(1x,a/)'')
740:   &      'Please check the position of a closing quotation mark!!'
741:   end if
742:   call FRELIN( IUPR, LINE, IS, IS+LL-1 )
743:   call CNTERR( 'WARNING' )
744: end if
745: C
746: C ..... SEARCH NEXT NON-BLANK OR TERMINATION CHARACTER ...
747: C
748: if ( ITERM.eq.0 ) then
749: 130 continue
750:   if ( IS.eq.0 .or. IS.gt.NCHAR ) then
751:     call GETLIN( IUIN, IUPR, LINE, NCHAR, IEND )
752:     if ( IEND.ne.0 ) return
753:     IS      = 1
754:   end if
755:
756:   if ( LINE(IS:NCHAR).eq.' ' ) then
757:     IS      = 0
758:     go to 130
759:   end if
760: C
761:   do 140 I = IS, NCHAR
762:     if ( LINE(I:I).ne.' ' ) then
763:       ITERM  = INDEX(CHAR,LINE(I:I))
764:       IS      = I
765:       if ( ITERM.ne.0 ) IS  = IS + 1
766:       go to 150
767:     end if
768: 140 continue
769:   end if
770: C
771: 150 if ( IS.gt.NCHAR ) IS  = 0
772: C
773:   return
774: end
775: C
776: C
777: C=====
778: C
779: C Entries for input of free form numeric value.
780: C

```

src/utils/fread.f

```

781: C   IA,RA,R8A,CHA : arrays or character string
782: C   NA : number of data elements which have been input.
783: C   NB : >0      expected data number. Ignore from NB+1'th
784: C           to the last input data.
785: C           =0     No expected data number. store all input data.
786: C           (may cause memory destruction !!)
787: C           <0     No expected data number. store upto abs(NB) input data.
788: C           (to prevent memory destruction)
789: C
790: C***** OBSOLETE *****
791: C** DATA TYPE 1: ONLY ONE VALUE IS ASSIGNED. PUT BLANK AFTER VNAME
792: C** AND PUT THE VALUE.
793: C** EX.
794: C**      VNAME 2.0      NGROUP 125
795: C*****
796: C
797: C DATA TYPE 2: values of arbitrary number are assigned.
798: C sandwich values by parentheses.
799: C ex.
800: C      VNAME( 2.0 1.0E-11 ) VNAME2( 0.0 10(20.0) )
801: C=====
802: C
803: C
804: C
805: C
806: C      subroutine I4READ( VNAME, IA, NA, NB, IEND )
807: C=====
808: C PURPOSE: Read integer data into array IA
809: C=====
810: C
811: C      character*(*) VNAME
812: C      integer IA(*)
813: C
814: C/#IF INTEGER8
815: C *      integer*8 I8DUMY
816: C/#ELSE
817: C      integer I8DUMY
818: C/#ENDIF
819: C      real R4DUMY
820: C      real*8 R8DUMY
821: C
822: C      call VREAD0( 'I4READ', VNAME, IA, I8DUMY, R4DUMY, R8DUMY, NA, NB,
823: C & IEND )
824: C      return
825: C      end
826: C
827: C
828: C
829: C
830: C      subroutine I8READ( VNAME, I8A, NA, NB, IEND )
831: C=====
832: C PURPOSE: Read integer*8 data into array I8A
833: C=====
834: C
835: C      character*(*) VNAME
836: C/#IF INTEGER8
837: C *      integer*8 I8A(*)
838: C/#ELSE
839: C      integer I8A(*)
840: C/#ENDIF
841: C
842: C      integer I4DUMY
843: C      real R4DUMY
844: C      real*8 R8DUMY
845: C

```

```

846: C      call VREAD0( 'I8READ', VNAME, I4DUMY, I8A, R4DUMY, R8DUMY, NA, NB,
847: C & IEND )
848: C      return
849: C      end
850: C
851: C
852: C
853: C
854: C      subroutine R4READ( VNAME, RA, NA, NB, IEND )
855: C=====
856: C PURPOSE: Read real data into array RA
857: C=====
858: C
859: C      character*(*) VNAME
860: C      real RA(*)
861: C
862: C      integer I4DUMY
863: C/#IF INTEGER8
864: C *      integer*8 I8DUMY
865: C/#ELSE
866: C      integer I8DUMY
867: C/#ENDIF
868: C      real*8 R8DUMY
869: C
870: C      call VREAD0( 'R4READ', VNAME, I4DUMY, I8DUMY, RA, R8DUMY, NA, NB,
871: C & IEND )
872: C      return
873: C      end
874: C
875: C
876: C
877: C
878: C      subroutine R8READ( VNAME, R8A, NA, NB, IEND )
879: C=====
880: C PURPOSE: Read real*8 data into array R8A
881: C=====
882: C
883: C      character*(*) VNAME
884: C      real*8 R8A(*)
885: C
886: C      integer I4DUMY
887: C/#IF INTEGER8
888: C *      integer*8 I8DUMY
889: C/#ELSE
890: C      integer I8DUMY
891: C/#ENDIF
892: C      real R4DUMY
893: C
894: C      call VREAD0( 'R8READ', VNAME, I4DUMY, I8DUMY, R4DUMY, R8A, NA, NB,
895: C & IEND )
896: C      return
897: C      end
898: C
899: C
900: C
901: C
902: C      subroutine DMREAD( VNAME, NA, NB, IEND )
903: C
904: C=====
905: C PURPOSE: Dummy read (read but no transfer on memory )
906: C=====
907: C
908: C      character*(*) VNAME
909: C
910: C      integer I4DUMY

```

```

end
=====
subroutine VREAD0( ENAME, VNAME, IA, I8A, RA, R8A, NA, NB, IEND )

character*(*) ENAME, VNAME
integer IA(*)
integer*8 I8A(*)
integer*8 I8AA
integer I8A(*)
integer I8AA
real RA(*)
real*8 R8A(*)

common /FRDATA/ NAA, IS, IUIN, IUPR, NCHAR, MXCERR,
& NCERR

..... INPUT BUFFER .....
MAXCOL : EFFECTIVE CHARACTERS IN AN INPUT-LINE

parameter( MAXCOL = 72 )
character LINE*(MAXCOL)

..... CHARACTER LIST .... ( ALPHABETS & NUMBERS ) .....

character*46 ALPNUM
character*26 ALPLOW

common /FRDATC/ LINE, ALPNUM, ALPLOW

```

```

976: C/#ELSE
977:         integer R8TOI8
978: C/#ENDIF
979:         external  R8TOI8
980: C
981: C      ..... Stack/flag for nested paraensises treatment .....
982: C
983: C      LCNT(L) : repetition count or "function" mode flag(-1999) of
984: C                nest level L
985: C      KCNT(L) : number of data stored in nest level L
986: C      JZEROC(L) : set to 1 if nest level L or upper level has
987: C                LCNT()==0 and data read in this level may be counted but
988: C                should not be stored on memory.
989: C
990:         parameter( MAXLVL   = 15 )
991:         integer LCNT(MAXLVL), KCNT(MAXLVL), JZEROC(MAXLVL)
992:         parameter( MAXFNL   = 128 )
993: C
994: C      .... for function
995:         character*(MAXFNL) PFUNC(MAXLVL)
996:         character*500 PPFUNC
997:         character*32 VCHAR
998: C
999:         real*8 R8AA
1000: C
1001: C-----
1002: C
1003:         if ( ENAME.eq.'I4READ' ) then
1004:             IRC      = 1
1005:         else if ( ENAME.eq.'R4READ' ) then
1006:             IRC      = 2
1007:         else if ( ENAME.eq.'R8READ' ) then
1008:             IRC      = 3
1009:         else if ( ENAME.eq.'DMREAD' ) then
1010:             IRC      = 4
1011:         else if ( ENAME.eq.'I8READ' ) then
1012:             IRC      = 5
1013:         else
1014:             write(IUPR,*) 'XXX(I4/I8/R4/R8/DM-READ) INVALID MODE <', ENAME,
1015: &                '>'
1016:             stop 666
1017:         end if
1018: C
1019: C
1020:         100 IFLAG   = 0
1021:             IEND    = 0
1022:             NBA     = ABS(NB)
1023:         CCCC NNX    = 0
1024:             NNX     = -1
1025:             LV      = INDEX(VNAME,' ') - 1
1026:             if ( LV.lt.0 ) LV = LEN(VNAME)
1027: C
1028: C
1029: C =====
1030: C ===== LABEL 110 =====
1031: C ===== BEGINNING POINT OF OUTERMOST IMPLICIT LOOP =====
1032: C ===== ( FIND OUTERMOST LEFT PARENSIS ) =====
1033: C =====
1034: C =====
1035: C
1036: C
1037:         110 continue
1038: C
1039: C      ..... IS = 0 ! NOW I MUST READ-IN NEW LINE. ....
1040: C

```

src/utlils/fread.f

```

1041:      if ( IS.eq.0 ) then
1042:        call GETLIN( IUIN, IUPR, LINE, NCHAR, IEND )
1043:        if ( IEND.ne.0 ) then
1044:          IS      = 0
1045:          return
1046:        end if
1047:        IS      = 1
1048:      end if
1049: C
1050: C
1051: C ..... FIND THE START POSITION OF DATA .....
1052: C
1053: C IFLAG = 1 : "(" of "VNAME(...)" is found.
1054: C IFLAG = 2 : data head of ")" of "VNAME(...)" is found.
1055: C
1056:      if ( IFLAG.ne.2 ) then
1057:        do 120 I = IS, NCHAR
1058:          CM2016 if ( IFLAG.eq.0.and.LINE(I:I).eq.'(' ) IFLAG = 1
1059:          CM2016 if ( LINE(I:I).ne.' ' .and.LINE(I:I).ne.'(' ) then
1060:            C      if ( IFLAG.eq.0 ) then
1061:              C        if( LINE(I:I).eq.'(' ) IFLAG = 1
1062:              if ( IFLAG.eq.0 .and. LINE(I:I).eq.'(' ) then
1063:                IFLAG = 1
1064:              else if ( LINE(I:I).ne.' ' ) then
1065:                IS      = I
1066:                IFLAG   = 2
1067:                go to 130
1068:              end if
1069:            120 continue
1070:            IS      = 0
1071:            go to 110
1072:          end if
1073: C
1074: C
1075: C ..... FIND THE FIRST DATA ELEMENT IN VNAME(...)
1076: C
1077: C
1078: C 130 continue
1079: C
1080: C
1081: C ..... START DATA INPUT !! .....
1082: C
1083: C
1084: C      NA      = 0
1085: C      LEVEL   = 1
1086: C      do 140 L = 1, MAXLVL
1087: C      CCCC      LCNT(L) = 0
1088: C      .... default repetition cout is set to 1 (from Nov 1999)
1089: C
1090: C      LCNT(L) = 1
1091: C      KCNT(L) = 0
1092: C      JZEROC(L) = 0
1093: C 140 continue
1094: C
1095: C =====
1096: C =====
1097: C ===== BEGINNING POINT OF DATA INPUT LOOP =====
1098: C ===== ( FIND DATA-TOKEN ) =====
1099: C =====
1100: C =====
1101: C
1102: C 150 continue
1103: C
1104: C      if ( IS.eq.0 ) then
1105: C        call GETLIN( IUIN, IUPR, LINE, NCHAR, IEND )

```

```

1106:      if ( IEND.ne.0 ) then
1107:        IS      = 0
1108:        return
1109:      end if
1110:      IS      = 1
1111:    end if
1112: C
1113: C =====
1114: C =====
1115: C ===== RETURNING POINT OF TOKEN-QUEST-LOOP =====
1116: C =====
1117: C =====
1118: C
1119: C 160 continue
1120: C
1121: C ..... FIND A NON-BLANK CHARACTER .....
1122: C
1123: C      do 170 I = IS, NCHAR
1124: C        if ( LINE(I:I).ne.' ' ) go to 180
1125: C 170 continue
1126: C      IS      = 0
1127: C      go to 150
1128: C
1129: C 180 IS      = I
1130: C
1131: C =====
1132: C ===== RETURN HERE WHEN CLOSING ")" IS FOUND =====
1133: C =====
1134: C
1135: C 190 continue
1136: C
1137: C
1138: C      if ( LINE(IS:IS).ne.' ' ) then
1139: C
1140: C
1141: C ..... READ IN A VALUE .....
1142: C
1143: C IFLAG2 = 1 : SIMPLE VALUE (SINGLE DATA ITEM)
1144: C 2 : VALUE IS REPETITION COUNT ( NN(...) )
1145: C 3 : VALUE IS AT THE END OF A DATA BLOCK ( ( ... DATA) )
1146: C 4 : VALUE IS INCREMENT COUNT ( NN(...) )
1147: C 5 : Function exporession {variable;expression}( ...
1148: C 0 : DATA NOT FOUND IN CURRENT INPUT LINE.
1149: C
1150: C CHECK %%%
1151: C write(*,*) '%%(fread(1)) is ',is,' ',line(is:is),' '
1152: C %%%
1153: C
1154: C IFLAG2 = 0
1155: C
1156: C PAR      = ' '
1157: C PAR(1:1) = LINE(IS:IS)
1158: C if ( IS.lt.NCHAR ) PAR(2:2) = LINE(IS+1:IS+1)
1159: C
1160: C ..... IN THE CASE OF {var:function} ( data ) ....
1161: C or <:var:function:> ( data ) ....
1162: C
1163: C LPF      = 0
1164: C if ( PAR(1:1).eq.'{' .or. PAR.eq.'<:' ) then
1165: C   PPFUNC = ' '
1166: C   if ( PAR(1:1).eq.'{' ) then
1167: C     PAR2 = '}'
1168: C     do 200 I = IS + 1, NCHAR
1169: C       CM2016 if ( LINE(I:I).eq.PAR2 ) IFLAG2 = 1
1170: C       CM2016 if ( IFLAG2.ne.0.and.LINE(I:I).eq.'(' ) then

```

src/utlils/fread.f

```

1171:         if ( IFLAG2.eq.0 ) then
1172:             if ( LINE(I:I).eq.PAR2 ) IFLAG2 = 1
1173: C
1174:         else if ( LINE(I:I).eq.'(' ) then
1175: C
1176:             if ( IFLAG2.gt.1 ) then
1177:                 write(IUPR,'(lx,a,a,a,a)') 'XXX(', ENAME,
1178: &                 ' ) Unexpected characters are found ',
1179: &                 'between "> (" function expression.'
1180:                 call FRELIN( IUPR, LINE, IFLAG2, IE )
1181:                 call CNTERR( 'FATAL' )
1182:                 NCERR = NCERR + 1
1183:                 if ( NCERR.gt.MXCERR ) call FRDERR
1184:             end if
1185: C
1186:             IFLAG2 = 5
1187:             go to 220
1188: C
1189:         else if ( LINE(I:I).ne.' ' ) then
1190:             if ( IFLAG2.eq.1 ) IFLAG2 = I
1191:             IE = I
1192:         end if
1193: C
1194:         if ( IFLAG2.eq.0 ) then
1195:             LPF = LPF + 1
1196:             PPFUNC(LPF:LPF) = LINE(I:I)
1197:         end if
1198: 200         continue
1199:     else
1200:         PAR2 = ':>'
1201:         do 210 I = IS + 2, NCHAR
1202:             if ( I.lt.NCHAR.and.LINE(I:I+1).eq.PAR2 ) IFLAG2 = 1
1203:             if ( IFLAG2.ne.0.and.LINE(I:I).eq.'(' ) then
1204:                 if ( IFLAG2.eq.0 ) then
1205:                     if ( I.lt.NCHAR.and.LINE(I:I+1).eq.PAR2 )
1206: &                     IFLAG2 = 1
1207: C
1208:                 else if ( LINE(I:I).eq.'(' ) then
1209: C
1210:                     if ( IFLAG2.gt.1 ) then
1211:                         write(IUPR,'(lx,a,a,a,a)') 'XXX(', ENAME,
1212: &                         ' ) Unexpected characters are found ',
1213: &                         'between "> (" function expression.'
1214:                         call FRELIN( IUPR, LINE, IFLAG2, IE )
1215:                         call CNTERR( 'FATAL' )
1216:                         NCERR = NCERR + 1
1217:                         if ( NCERR.gt.MXCERR ) call FRDERR
1218:                     end if
1219: C
1220:                     IFLAG2 = 5
1221:                     go to 220
1222: C
1223:                 else if ( LINE(I:I).ne.' ' ) then
1224:                     if ( IFLAG2.eq.1 ) IFLAG2 = I
1225:                     IE = I
1226:                 end if
1227: C
1228:                 if ( IFLAG2.eq.0 ) then
1229:                     LPF = LPF + 1
1230:                     PPFUNC(LPF:LPF) = LINE(I:I)
1231:                 end if
1232: 210         continue
1233:             end if
1234: C
1235: c##<2007/03/14:PN3:

```

```

1236: c##         write(IUPR,'(lx,a,a)') 'XXX(', ENAME,
1237:         write(IUPR,'(lx,a,a,a)') 'XXX(', ENAME,
1238: c##>
1239: &         ' ) Function expression (+ "(") is not closing in a line.'
1240:         IE = NCHAR
1241:         call FRELIN( IUPR, LINE, IS, IE )
1242:         call CNTERR( 'FATAL' )
1243:         NCERR = NCERR + 1
1244:         if ( NCERR.gt.MXCERR ) call FRDERR
1245:         IS = 0
1246:         go to 150
1247: C
1248: 220         go to 250
1249: C
1250: C ..... IN THE CASE OF <.....> DATA ....
1251: C
1252:         else if ( PAR(1:1).eq.'<' ) then
1253:             do 230 I = IS, NCHAR
1254:                 if ( LINE(I:I).eq.'>' ) IFLAG2 = 1
1255:                 if ( IFLAG2.ne.0.and.LINE(I:I).eq.'(' ) IFLAG2 = 2
1256:                 if ( IFLAG2.ne.0.and.LINE(I:I).eq.')' ) IFLAG2 = 3
1257:                 if ( IFLAG2.ne.0.and.LINE(I:I).eq.':') IFLAG2 = 4
1258:                 if ( IFLAG2.eq.1.and.(LINE(I:I).eq.' ' .or.I.eq.NCHAR) )
1259: &                 go to 250
1260:                 if ( IFLAG2.gt.1 ) go to 250
1261: 230         continue
1262:         write(IUPR,'(lx,a,a,a)') 'XXX(', ENAME,
1263: &         ' ) Expression <...> is not closing in a line.'
1264:         IE = NCHAR
1265:         call FRELIN( IUPR, LINE, IS, IE )
1266:         call CNTERR( 'FATAL' )
1267:         NCERR = NCERR + 1
1268:         if ( NCERR.gt.MXCERR ) call FRDERR
1269:         IS = 0
1270:         go to 150
1271: C
1272: C ..... data not in form <...> ....
1273: C
1274: C
1275:         else
1276:             do 240 I = IS, NCHAR
1277:                 if ( LINE(I:I).eq.' ' ) IFLAG2 = 1
1278:                 if ( LINE(I:I).eq.'(' ) IFLAG2 = 2
1279:                 if ( LINE(I:I).eq.')' ) IFLAG2 = 3
1280:                 if ( LINE(I:I).eq.':') IFLAG2 = 4
1281:                 if ( IFLAG2.ne.0 ) go to 250
1282: 240         continue
1283:             IFLAG2 = 1
1284:             end if
1285: C
1286: 250         IE = I - 1
1287:         CHECK %%%
1288:         write(*,*) '%%(fread(2)) iflag2',iflag2, ' ie "',line(is:ie),'"'
1289: C %%%%%%%%%
1290: C ... read next line ...
1291: C
1292:         if ( IFLAG2.eq.0 ) then
1293:             IS = 0
1294:             go to 150
1295: C
1296: C ... simple data ...
1297: C
1298:         else if ( IFLAG2.eq.1 .or. IFLAG2.eq.3 ) then
1299: C
1300: CCCCC         if ( NNX.eq.0 ) then

```


src/utls/fread.f

```

1301: C
1302: C      Increment count NNX = -1 means not in increment mode
1303: C      and NNX = 0 is "zero-time" increment !!!
1304: C      (behaviour changed in Nov 1999)
1305: C
1306: C      if ( NNX.eq.-1 ) then
1307: C      if ( JZEROC(LEVEL).eq.0 ) then
1308: C      KCNT(LEVEL) = KCNT(LEVEL) + 1
1309: C      NA = NA + 1
1310: C      end if
1311: C
1312: C      if ( IE.ge.IS.and.LINE(IS:IE).ne.' '
1313: C      .and.(NB.eq.0.or.NA.le.NBA) ) then
1314: C      if ( LINE(IS:IS).ne.' ' .or. LINE(IE:IE).ne.' ' )
1315: C      then
1316: C      IER = 0
1317: C
1318: C      if ( JZEROC(LEVEL).eq.1 ) then
1319: C      R8AA = DENTAK(LINE(IS:IE),IUPR,IER)
1320: C      else
1321: C
1322: C ----- IF(IRC.EQ.1) READ(LINE(IS:IE),*) IA(NA)
1323: C
1324: C      if ( IRC.eq.1 ) then
1325: C      IA(NA) = INT(DENTAK(LINE(IS:IE),IUPR,IER))
1326: C      end if
1327: C
1328: C ----- IF(IRC.EQ.5) READ(LINE(IS:IE),*) I8A(NA)
1329: C
1330: C      if ( IRC.eq.5 ) then
1331: C      I8A(NA) = R8TOI8(DENTAK(LINE(IS:IE),IUPR,IER))
1332: C      end if
1333: C
1334: C ----- IF(IRC.EQ.2) READ(LINE(IS:IE),*) RA(NA)
1335: C
1336: C      if ( IRC.eq.2 ) then
1337: C      RA(NA) = DENTAK(LINE(IS:IE),IUPR,IER)
1338: C      end if
1339: C
1340: C ----- IF(IRC.EQ.3) READ(LINE(IS:IE),*) R8A(NA)
1341: C
1342: C      if ( IRC.eq.3 ) then
1343: C      R8A(NA) = DENTAK(LINE(IS:IE),IUPR,IER)
1344: C      end if
1345: C      end if
1346: C
1347: C      if ( IER.ne.0 ) then
1348: C      write(IUPR,'(1x,a,a)') 'XXX(', ENAME,
1349: C      ' ) DATA READ ERROR.'
1350: C      call FRELIN( IUPR, LINE, IS, IE )
1351: C      call CNTERR( 'FATAL' )
1352: C      NCERR = NCERR + 1
1353: C      if ( NCERR.gt.MXCERR ) call FRDERR
1354: C      end if
1355: C
1356: C ----- CHARACTER STRING -----
1357: C      (probably meaningless)
1358: C
1359: C      else
1360: C      C8 = LINE(IS+1:IE-1)
1361: C      if ( IRC.eq.1 ) read(C8,'(A4)') IA(NA)
1362: C      if ( IRC.eq.2 ) read(C8,'(A4)') RA(NA)
1363: C      if ( IRC.eq.3 ) read(C8,'(A8)') R8A(NA)
1364: C      CM2016
1365: C      if ( IRC.eq.5 ) read(C8,'(A8)') I8A(NA)

```

```

1366: C      end if
1367: C      end if
1368: C
1369: C ----- CASE OF INCREMENT ----- ( NNX >= 0 ) ----
1370: C      NNX : NUMBER OF VALUES TO BE MADE BY INCREMENT
1371: C
1372: C      else
1373: C      IER = 0
1374: C      if ( IRC.eq.1 ) then
1375: C      IAA = INT(DENTAK(LINE(IS:IE),IUPR,IER))
1376: C      if ( JZEROC(LEVEL).eq.0.and.NNX.gt.0 ) then
1377: C      do 260 I = 1, NNX
1378: C      if ( NB.eq.0 .or. NA+I.le.NBA ) then
1379: C      IA(NA+I) = IA(NA) + I*IAA
1380: C      end if
1381: C      260 continue
1382: C      end if
1383: C
1384: C      else if ( IRC.eq.2 ) then
1385: C      RAA = DENTAK(LINE(IS:IE),IUPR,IER)
1386: C      if ( JZEROC(LEVEL).eq.0.and.NNX.gt.0 ) then
1387: C      do 270 I = 1, NNX
1388: C      if ( NB.eq.0 .or. NA+I.le.NBA ) then
1389: C      RA(NA+I) = RA(NA) + I*RAA
1390: C      end if
1391: C      270 continue
1392: C      end if
1393: C
1394: C      else if ( IRC.eq.3 ) then
1395: C      R8AA = DENTAK(LINE(IS:IE),IUPR,IER)
1396: C      if ( JZEROC(LEVEL).eq.0.and.NNX.gt.0 ) then
1397: C      do 280 I = 1, NNX
1398: C      if ( NB.eq.0 .or. NA+I.le.NBA ) then
1399: C      R8A(NA+I) = R8A(NA) + I*R8AA
1400: C      end if
1401: C      280 continue
1402: C      end if
1403: C      CM2016
1404: C
1405: C      else if ( IRC.eq.5 ) then
1406: C      I8AA = R8TOI8(DENTAK(LINE(IS:IE),IUPR,IER))
1407: C      if ( JZEROC(LEVEL).eq.0.and.NNX.gt.0 ) then
1408: C      do
1409: C      if ( NB.eq.0 .or. NA+I.le.NBA ) then
1410: C      I8A(NA+I) = I8A(NA) + I*I8AA
1411: C      end if
1412: C      end do
1413: C      end if
1414: C
1415: C      if ( IER.ne.0 ) then
1416: C      write(IUPR,'(1x,a,a)') 'XXX(', ENAME,
1417: C      ' ) DATA READ ERROR.'
1418: C      &
1419: C      call FRELIN( IUPR, LINE, IS, IE )
1420: C      call CNTERR( 'FATAL' )
1421: C      NCERR = NCERR + 1
1422: C      if ( NCERR.gt.MXCERR ) call FRDERR
1423: C      end if
1424: C
1425: C      if ( JZEROC(LEVEL).eq.0 ) then
1426: C      KCNT(LEVEL) = KCNT(LEVEL) + NNX
1427: C      NA = NA + NNX
1428: C      end if
1429: C      end if
1430: C      ... reset to non increment mode

```

src/utlils/fread.f

```

1431: C
1432:         NNX      = -1
1433: C
1434: C         ..... " DATA)" ...
1435: C
1436:         if ( IFLAG2.eq.3 ) then
1437:             IS      = IE + 1
1438:             go to 190
1439:         end if
1440: C
1441: C         ..... " DATA " ...
1442: C
1443: CM2016 ... to effect because IFLAG is always 2 here,
1444: CM         and "goto 320" is incorrect if IFLAG is replaced with IFLAG2.
1445: CM         if ( IFLAG.eq.1 ) then
1446: CM             IS      = IE + 2
1447: CM             go to 320
1448: CM         end if
1449: C
1450: C         ..... N:X TYPE MEANS MAKING N VALUES BY INCREMENT X .....
1451: C
1452:         else if ( IFLAG2.eq.4 ) then
1453: C
1454: C         -----
1455: C         << behaviour for "N:data" is changed in Nov 1999 >>
1456: C
1457: C         ... NNX = -1 means not in increment mode
1458: C         NNX = 0 means "zero-count" increment
1459: C         i.e. : "data" of N:data is read but not used when
1460: C         N=0 or NNX is set to zero by some reason
1461: C         (probably by read error for N).
1462: C
1463: C         Formerly, at least one increment is made even though
1464: C         NNX is not positive.
1465: C         -----
1466: C
1467: CM2016 Feb.17
1468:         if ( NNX.ne.-1 ) then
1469:             write(IUPR,'(1x,a,a,a)') 'XXX(', ENAME,
1470:             &             ' ) 2nd ":" is found before processing previous one.'
1471:             IE1      = IE + 1
1472:             call FRELIN( IUPR, LINE, IS, IE1 )
1473:             call CNTERR( 'FATAL' )
1474:             NCERR     = NCERR + 1
1475:             if ( NCERR.gt.MXCERR ) call FRDERR
1476:             NNX      = -1
1477:         end if
1478:         if ( NA.eq.0 ) then
1479:             write(IUPR,'(1x,a,a,a)') 'XXX(', ENAME,
1480:             &             ' ) ":" is found when NA=0 (no data read in so far).'
1481:             IE1      = IE + 1
1482:             call FRELIN( IUPR, LINE, IS, IE1 )
1483:             call CNTERR( 'FATAL' )
1484:             NCERR     = NCERR + 1
1485:             if ( NCERR.gt.MXCERR ) call FRDERR
1486:         end if
1487: CM         NNX      = -1
1488:         if ( IRC.le.3 ) then
1489:             if ( IE.ge.IS ) then
1490:                 NNX      = INT(DENTAK(LINE(IS:IE),IUPR,IER))
1491:                 if ( IER.ne.0 ) then
1492:                     NNX      = 0
1493:                     write(IUPR,'(1x,a,a,a)') 'XXX(', ENAME,
1494:                     &                     ' ) Data read error.'
1495:                     call FRELIN( IUPR, LINE, IS, IE )

```

```

1496:             call CNTERR( 'FATAL' )
1497:             NCERR     = NCERR + 1
1498:             if ( NCERR.gt.MXCERR ) call FRDERR
1499:             else if ( NNX.eq.0 ) then
1500:                 write(IUPR,'(1x,a,a)') '!!!(DATA INPUT)',
1501:                 &                 ' Increment count ( N of "N:data" ) is zero.'
1502:                 call FRELIN( IUPR, LINE, IS, IE )
1503:                 call CNTERR( 'WARNING' )
1504:             else if ( NNX.lt.0 ) then
1505:                 NNX      = 0
1506:                 write(IUPR,'(1x,a,a,a,i8,a)') 'XXX(DATA INPUT)',
1507:                 &                 ' Increment count ( N of "N:data" ) is negative.',
1508:                 &                 ' ( ', NNX, ' )'
1509:                 call FRELIN( IUPR, LINE, IS, IE )
1510:                 call CNTERR( 'FATAL' )
1511:                 NCERR     = NCERR + 1
1512:                 if ( NCERR.gt.MXCERR ) call FRDERR
1513:             end if
1514:         else
1515:             NNX      = 0
1516:             write(IUPR,'(1x,a,a/1x,a,i4)') '!!!(DATA INPUT)',
1517:             &             ' Increment count ( N of "N:data" ) is null.'
1518:             call FRELIN( IUPR, LINE, IE+1, IE+1 )
1519:             call CNTERR( 'WARNING' )
1520:         end if
1521:     end if
1522: C
1523: C         ..... READ IN REPETITION COUNT. ( N of N(...) ) .....
1524: C
1525:         else if ( IFLAG2.eq.2 ) then
1526:             LEVEL     = LEVEL + 1
1527:             if ( LEVEL.gt.MAXLVL ) then
1528:                 write(IUPR,'(1x,a,a,i8,a)') 'XXX(DATA INPUT)',
1529:                 &                 ' Too deeply nested (((...))) (MAXLVL=', MAXLVL,
1530:                 &                 ' )'
1531:                 call FRELIN( IUPR, LINE, IS, IE )
1532:                 call CNTERR( 'FATAL' )
1533:                 NCERR     = NCERR + 1
1534:                 if ( NCERR.gt.MXCERR ) call FRDERR
1535:             end if
1536: C
1537: C         LCNT(LEVEL) = 1
1538: C         JZEROC(LEVEL) = 0
1539: C
1540: CM2016         if ( IRC.le.3 ) then
1541:             if ( IRC.le.3 .or. IRC.eq.5 ) then
1542:                 if ( IE.ge.IS ) then
1543: C
1544: C         ..... R(...) : REPEAT TO THE END OF ARRAY .....
1545: C
1546:                 if ( LINE(IS:IE).eq.'R' ) then
1547:                     LCNT(LEVEL) = -999
1548:                     if ( LEVEL.ne.2 ) then
1549:                         write(IUPR,'(1x,a,a,a,a)') 'XXX(', ENAME,
1550:                         &                         ' ) R(...) is in an ',
1551:                         &                         'invalid nesting level. INPUT ERROR !!'
1552:                         call CNTERR( 'FATAL' )
1553:                         NCERR     = NCERR + 1
1554:                         if ( NCERR.gt.MXCERR ) call FRDERR
1555:                     end if
1556:                 else
1557:                     LCNT(LEVEL) = INT(DENTAK(LINE(IS:IE),IUPR,IER))
1558:                     if ( IER.ne.0 ) then
1559:                         write(IUPR,'(1x,a,a)') 'XXX(', ENAME,
1560:                         &                         ' ) Data read error.'

```

src/utlils/fread.f

```

1561:      call FRELIN( IUPR, LINE, IS, IE )
1562:      call CNTERR( 'FATAL' )
1563:      NCERR = NCERR + 1
1564:      if ( NCERR.gt.MXCERR ) call FRDERR
1565:      LCNT(LEVEL) = 0
1566:      else if ( LCNT(LEVEL).eq.0 ) then
1567:        write(IUPR,'(1x,a,a)') '!!!(DATA INPUT)',
1568:      &      ' Repetition count ( N of "N(...)" ) is zero.'
1569:      call FRELIN( IUPR, LINE, IS, IE )
1570:      call CNTERR( 'WARNING' )
1571:      else if ( LCNT(LEVEL).lt.0 ) then
1572:        LCNT(LEVEL) = 0
1573:        write(IUPR,'(1x,a,a,i8,a)') 'XXX(DATA INPUT)',
1574:      &      ' Repetition count (N of "N(...)" ) is negative.',
1575:      &      ' ( LCNT(LEVEL), ' ) '
1576:      call FRELIN( IUPR, LINE, IS, IE )
1577:      call CNTERR( 'FATAL' )
1578:      NCERR = NCERR + 1
1579:      if ( NCERR.gt.MXCERR ) call FRDERR
1580:      end if
1581:    end if
1582: C
1583: C    ... no characters before "("
1584: C
1585: C    It is explicitly treated as "one-time repetition"
1586: C    from 17 Nov 1999
1587: C
1588: C    else
1589: CCCCCC      LCNT(LEVEL) = 0
1590:      LCNT(LEVEL) = 1
1591:      write(IUPR,'(1x,a,a/5x,a)') '!!!(DATA INPUT)',
1592:      &      ' Repetition count (N of "N(...)" ) is null.',
1593:      &      ' Treated as repetition count is equal to 1.'
1594:      call FRELIN( IUPR, LINE, IE+1, IE+1 )
1595:      call CNTERR( 'WARNING' )
1596:    end if
1597: C
1598: C    ... character input mode does not allow repetition
1599: C    else if ( IRC.eq.4 ) then
1600: C      LCNT(LEVEL) = 1
1601: C    end if
1602: C
1603: C    if ( LCNT(LEVEL).eq.0 ) JZEROC(LEVEL) = 1
1604: C
1605: C    ... but data in this level should be discarded if
1606: C    zero repetition count is set in upper level ...
1607: C
1608: C    if ( JZEROC(LEVEL-1).eq.1 ) JZEROC(LEVEL) = 1
1609: C
1610: C ..... Check function .....
1611: C
1612: C    else if ( IFLAG2.eq.5 ) then
1613: C      LEVEL = LEVEL + 1
1614: C      if ( LEVEL.gt.MAXLVL ) then
1615: C        write(IUPR,'(1x,a,a,i8,a)') 'XXX(DATA INPUT)',
1616: C      &      ' Too deeply nested (((...))) (MAXLVL=', MAXLVL,
1617: C      &      ' )'
1618: C      call FRELIN( IUPR, LINE, IS, IE )
1619: C      call CNTERR( 'FATAL' )
1620: C      NCERR = NCERR + 1
1621: C      if ( NCERR.gt.MXCERR ) call FRDERR
1622: C    end if
1623: C    LCNT(LEVEL) = -1999
1624: C    if ( INDEX(PPFUNC,';').eq.0 ) then
1625: C      write(IUPR,'(1x,a,a,a)') 'XXX(DATA INPUT)',

```

```

1626: C      &      ' Function should be given as "variable;expression"',
1627: C      &      ' but lacks ";".'
1628: C      call FRELIN( IUPR, LINE, IS, IE )
1629: C      call CNTERR( 'FATAL' )
1630: C      NCERR = NCERR + 1
1631: C      if ( NCERR.gt.MXCERR ) call FRDERR
1632: C    end if
1633: C
1634: C      PFUNC(LEVEL) = PPFUNC
1635: C      JZEROC(LEVEL) = 0
1636: C
1637: C      ... but data in this level should be discarded if
1638: C      zero repetition count is set in upper level ...
1639: C
1640: C      if ( JZEROC(LEVEL-1).eq.1 ) JZEROC(LEVEL) = 1
1641: C    end if
1642: C
1643: C      IS = IE + 2
1644: C      if ( IS.gt.NCHAR ) then
1645: C        IS = 0
1646: C        go to 150
1647: C      end if
1648: C      go to 160
1649: C
1650: C
1651: C ..... ' )' IS END OF DATA GROUP SANDWICHED BY ( ). .....
1652: C
1653: C
1654: C***** IF(LINE(IS:IS).EQ.')') THEN
1655: C
1656: C    else
1657: C
1658: C ..... END OF ALL INPUT DATA .....
1659: C
1660: C    if ( LEVEL.eq.1 ) then
1661: C      IS = IS + 1
1662: C      if ( IS.gt.NCHAR ) IS = 0
1663: C      go to 320
1664: C    end if
1665: C
1666: C      LL = LCNT(LEVEL)
1667: C      KK = KCNT(LEVEL)
1668: C
1669: C ..... apply function to data of the level ....
1670: C
1671: C    if ( LL.eq.-1999 ) then
1672: C      PPFUNC = PFUNC(LEVEL)
1673: C      if ( IRC.eq.1 ) then
1674: C        call APFNCI( IA(NA-KK+1), KK, PPFUNC, IUPR, NERR1 )
1675: C      else if ( IRC.eq.2 ) then
1676: C        call APFNCR( RA(NA-KK+1), KK, PPFUNC, IUPR, NERR1 )
1677: C      else if ( IRC.eq.3 ) then
1678: C        call APFNCD( R8A(NA-KK+1), KK, PPFUNC, IUPR, NERR1 )
1679: C      CM2016
1680: C    else if ( IRC.eq.5 ) then
1681: C      do K = 1, KK
1682: C        R8A(1) = I8A(NA-KK+K)
1683: C        call APFNCD( R8A(1), 1, PPFUNC, IUPR, NERR1 )
1684: C        I8A(NA-KK+K) = R8TOI8( R8A(1) )
1685: C      end do
1686: C    else
1687: C      NERR1 = 0
1688: C    end if
1689: C    if ( NERR1.ne.0 ) then
1690: C      call CNTERR( 'FATAL' )

```

```
src/utls/fread.f
```

```

1691:      write(IUPR,'(1x,a)') 'XXX(VREAD0) ERRORS IN APFNC?'
1692:      NCERR = NCERR + NERR1
1693:      if ( NCERR.gt.MXCERR ) call FRDERR
1694:    end if
1695:    KCNT(LEVEL-1) = KCNT(LEVEL-1) + KCNT(LEVEL)
1696:    LCNT(LEVEL) = 0
1697:    KCNT(LEVEL) = 0
1698:    LEVEL = LEVEL - 1
1699:    IS = IS + 1
1700:    go to 160
1701:  else
1702: C ..... REPEAT INPUT DATA OF THE PREVIOUS LEVEL .....
1703: C .....
1704: C      NL = NA + (LL-1)*KK
1705: C ..... R(...) OPTION .....
1706: C .....
1707: C      if ( LL.eq.-999 ) then
1708: C        if ( NB.ge.0.and.NBA.le.NA ) then
1709: C          call CNTERR( 'WARNING' )
1710: C
1711: C          write(IUPR,'(1x,a,a,a,a,i5,a/4x,a,i5,a/)' ) '!!!('
1712: C            & ENAME,') R(...) is ignored ',
1713: C            & ' because expected number of data (',NB,
1714: C            & ',) is greater than or equal to',
1715: C            & ' the number of data read in so far (',NA,
1716: C            & ') '
1717: C
1718: C          NL = NA
1719: C        else
1720: C          NL = NBA
1721: C        end if
1722: C      end if
1723: C
1724: C      if ( IRC.eq.1 ) then
1725: C        do 290 K = NA + 1, NL
1726: C          if ( NB.eq.0 .or. K.le.NBA ) IA(K) = IA(K-KK)
1727: C          continue
1728: C        else if ( IRC.eq.2 ) then
1729: C          do 300 K = NA + 1, NL
1730: C            if ( NB.eq.0 .or. K.le.NBA ) RA(K) = RA(K-KK)
1731: C            continue
1732: C          else if ( IRC.eq.3 ) then
1733: C            do 310 K = NA + 1, NL
1734: C              if ( NB.eq.0 .or. K.le.NBA ) R8A(K) = R8A(K-KK)
1735: C              continue
1736: C            else if ( IRC.eq.5 ) then
1737: C              do K = NA + 1, NL
1738: C                if ( NB.eq.0 .or. K.le.NBA ) I8A(K) = I8A(K-KK)
1739: C                end do
1740: C              end if
1741: C            end if
1742: C          end if
1743: C
1744: C      NA = NL
1745: C      if( NL.ge.NA ) NA = NL
1746: C      KCNT(LEVEL-1) = KCNT(LEVEL-1) + LCNT(LEVEL)*KCNT(LEVEL)
1747: C      LCNT(LEVEL) = 0
1748: C      LCNT(LEVEL) = 1
1749: C      KCNT(LEVEL) = 0
1750: C      LEVEL = LEVEL - 1
1751: C      IS = IS + 1
1752: C
1753: C***** IF( IFLAG.EQ.1 .AND. NA.GE.NB ) RETURN
1754:

```

[illegible]

[illegible][illegible]

```

LINE      = LNSAVE(0)
IS        = ISSAVE
IUIN      = IOSAVE
ISSAVE    = 0
IOSAVE    = 0
NVLINE    = 0
return

end

=====
FRDERR:
Stop program execution in this routine if number of fatal
errors in this routine exceeds limit.
=====

subroutine FRDERR

include 'INC/_IOUNIT'
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
common /FRDATA/ NAA, IS, IUIN, IUPR, NCHAR, MXCERR,
& NCERR
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

write(IUPR,7000)
format('1',1X,8('XXXXXXXXXX'))//2X,
& 'Too many errors in data input routine. Please check your'
& ' input data.'//1X,
& ' Probable causes of errors are as follows:'//2X,
& ' * Unclosed (...) or <...> pairs.'//2X,
& ' * Use of undefined symbolic parameters or misused',
& ' parameter name.'//2X,' * Invalid use of function.'//2X,
& ' * Invalid numeric data.'//2X,8('XXXXXXXXXX'))

call PRSTOP( 0, 'Too many error in free-form input.' )
stop 888
end
```

```
call PRSTOP( 1, 'INTERNAL ERROR IN FREE FORM I/O ROUTINE.' )
stop 999
end if

LINE      = LNSAVE(0)
IS        = ISSAVE
IUIN      = IOSAVE
ISSAVE    = 0
IOSAVE    = 0
NVLINE    = 0
return

end

=====
FRDERR:
Stop program execution in this routine if number of fatal
errors in this routine exceeds limit.
=====

subroutine FRDERR

include 'INC/_IOUNIT'
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
common /FRDATA/ NAA, IS, IUIN, IUPR, NCHAR, MXCERR,
& NCERR
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

write(IUPR,7000)
format('1',1X,8('XXXXXXXXXX'))//2X,
& 'Too many errors in data input routine. Please check your'
& ', input data.'//1X,
& ' Probable causes of errors are as follows:'//2X,
& ' * Unclosed (...) or <...> pairs.'//2X,
& ' * Use of undefined symbolic parameters or misused',
& ' parameter name.'//2X,' * Invalid use of function.'//2X,
& ' * Invalid numeric data.'//2X,8('XXXXXXXXXX'))
```

```

2026:      call CNTERR( 'FATAL' )
2027:
2028: Ccccccc   ncerr = ncerr + 1
2029: Ccccccc   if( ncerr .gt. mxcerr ) call frderr
2030:
2031:           end if
2032:           return
2033:       end
2034: C
2035: C
2036: C
2037: C
2038:         subroutine READR4( LINE2, IST,    IEND, RET )
2039: C=====
2040: C READR4:   Read real   data and put it in RET
2041: C=====
2042:             character*(*) LINE2
2043:             real*8 DENTAK
2044: C
2045: Cccc include 'INC/_IOUNIT'
2046: C <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
2047:             common /FRDATA/ NAA,     IS,        IUIN,   IUPR,  NCHAR, MXCERR,
2048:             &              NCERR
2049: C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2050: C
2051:             RET    = REAL(DENTAK(LINE2(IST:IEND),IUPR,IER))
2052:             if ( IER.ne.0 ) then
2053:                 write(IUPR,'(lx,a)') 'XXX(READR4) DATA READ ERROR.'
2054:                 call FRELIN( IUPR, LINE2, IST, IEND )
2055:                 call CNTERR( 'FATAL' )
2056:
2057: Ccccccc   ncerr = ncerr + 1
2058: Ccccccc   if( ncerr .gt. mxcerr ) call frderr
2059:
2060:           end if
2061:           return
2062:       end
2063: C
2064: C
2065: C
2066: C
2067:         subroutine FRELIN( IOUT, LINE, IS,    IE )
2068: C=====
2069: C Frelin:   Print line having invalid/suspicious data.
2070: C History: Programmed by M.Sasaki ( 21 May 1995 )
2071: C Update:
2072: C 10 Jul 1998: LINE may be longer than 72 characters.
2073: C-----
2074: C   IOUT : printout I/O unit
2075: C   LINE : input line
2076: C   IS/IEND : start/end column position marked as invalid.
2077: C=====
2078:             character*(*) LINE
2079: C
2080:             parameter( MAXCOL = 72 )

```

src/utils/fread.f

```

2081:      character*(MAXCOL) MARK
2082: C
2083:      I1      = MIN(IS,IE)
2084:      I2      = MAX(IS,IE)
2085:      LL      = LEN(MARK)
2086:      NN      = MAX(ICLEN2(LINE),I2)
2087:      do 110 J = 1, NN, LL
2088:          MM      = MIN(J+LL-1,NN)
2089:          MARK      = ' '
2090:          do 100 I = J, MM
2091:              MARK(I-J+1:I-J+1) = ' '
2092:              if ( I.ge.I1.and.I.le.I2 ) MARK(I-J+1:I-J+1) = 'V'
2093: 100      continue
2094:          write(IOUT,'(/2X,' ' ' 'A')' ) MARK(:MM-J+1)
2095:          write(IOUT,'(2X,' 'LINE: ' 'A/)' ) LINE(J:MM)
2096: 110      continue
2097: C
2098: C
2099:      return
2100:      end
2101: C
2102: C
2103: C
2104: C
2105:      subroutine APFUNC( IRC, IA, RA, DA, N, PPFUNC,IUPR,
2106:      & NERR1 )
2107: C=====
2108: C Subrutines to apply "function" to array data
2109: C
2110: C PPFUNC must have the folloing form:
2111: C
2112: C      "variable-name ; expression"
2113: C
2114: C "variable-name" strings in "expression" are replaced with data in
2115: C array IA (IRC=1),RA (IRC=2),DA (IRC=3) and passed to
2116: C DENTAK routine. Data on array IA,RA,DA are replaced by resulting
2117: C values.
2118: C
2119: C History: Programmed by M.Sasaki (12 July 1998)
2120: C Update:
2121: C=====
2122:      integer IA(N)
2123:      real RA(N)
2124:      real*8 DA(N)
2125:      character*(*) PPFUNC
2126: C
2127: C ... local data ...
2128:      character*500 ELINE
2129:      parameter( MAXXP = 512 )
2130:      integer IPOS(MAXXP)
2131:      integer JVC(MAXXP)
2132: C
2133:      real*8 DD
2134:      character*16 VN
2135:      character*32 VCHAR
2136:      character*63 ALPN
2137:      character*1 CW
2138: C
2139:      real*8 DENTAK
2140:      external DENTAK
2141: C
2142:      data ALPN /
2143:      &'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-' /
2144: C
2145: C -----

```

```

2146: C
2147:      NERR1      = 0
2148:      LP      = ICLEN2(PPFUNC)
2149:      K      = INDEX(PPFUNC,';')
2150:      if ( K.eq.0 ) then
2151:          write(IUPR,*) 'XXX(APFUNC) Function string has no ";" to ',
2152:      & ' separate variable name and expression.'
2153:          write(IUPR,*) ' function <', PPFUNC(:LP), '>'
2154:          NERR1      = NERR1 + 1
2155:          return
2156:      end if
2157:      VN      = PPFUNC(:K-1)
2158:      LVN      = 0
2159:      do 100 I = 1, K - 1
2160:          if ( VN(I:I).ne.' ' ) then
2161:              LVN      = LVN + 1
2162:              CW      = VN(I:I)
2163:              VN(LVN:LVN) = CW
2164:          end if
2165: 100      continue
2166: C
2167:      if ( LVN.eq.0 ) then
2168:          write(IUPR,*)
2169:      & 'XXX(APFUNC) Function string has no variable name',
2170:      & ' before ";".'
2171:          write(IUPR,*) ' function <', PPFUNC(:LP), '>'
2172:          NERR1      = NERR1 + 1
2173:          return
2174:      end if
2175: C
2176: C ... split PPFUNC into parts :
2177: C IPOS(i) : start position of i'th part in PPFUNC
2178: C JVC(i) : i'th part is replaced by expanded data value string
2179: C when JVC(i) = 1
2180: C
2181:      IP      = K + 1
2182:      NP      = 0
2183:      IPOS(1) = IP
2184: 110      if ( IP.le.LP ) then
2185:          KK      = INDEX(PPFUNC(IP:LP),VN(:LVN))
2186:          if ( KK.ne.0 ) then
2187:              K1      = IP + KK - 1
2188:              K2      = K1 + LVN - 1
2189:
2190:              if ( (K1.gt.1.and.INDEX(ALPN,PPFUNC(K1-1:K1-1)).ne.0)
2191:      & .or. (K2.lt.LP.and.INDEX(ALPN,PPFUNC(K2+1:K2+1)).ne.0) )
2192:      & then
2193:                  IP      = IP + 1
2194:                  go to 110
2195:              end if
2196:              if ( K1.gt.IPOS(NP+1) ) then
2197:                  NP      = NP + 1
2198:                  JVC(NP) = 0
2199:                  IPOS(NP+1) = K1
2200:              end if
2201:              NP      = NP + 1
2202:              JVC(NP) = 1
2203:              IP      = K2 + 1
2204:              IPOS(NP+1) = IP
2205:              go to 110
2206:          else
2207:              NP      = NP + 1
2208:              IPOS(NP+1) = LP + 1
2209:              JVC(NP) = 0
2210:          end if

```

src/utls/fread.f

```

2211:      end if
2212: Check %%%%%%%%%%
2213: C      write(*,*) '%%%check (APFUNC) n ',n,' np = ', NP
2214: C      write(*,*) ' variable = <', VN(:LVN), '>'
2215: C      do I = 1, NP
2216: C          write(*,*) 'part ', I, JVC(I), ' from ', IPOS(I), ' to ',
2217: C      &          IPOS(I+1) - 1, ' <', PPFUNC(IPOS(I):IPOS(I+1)-1), '>'
2218: C      end do
2219: C %%%%%%%%%%%%%%
2220: C
2221:      do 130 K = 1, N
2222:      if ( IRC.eq.1 ) then
2223:      call DVCHAR( DBLE(IA(K)), VCHAR, LVV, IERR )
2224:      else if ( IRC.eq.2 ) then
2225:      call EVCHAR( RA(K), VCHAR, LVV, IERR )
2226:      else
2227:      call DVCHAR( DA(K), VCHAR, LVV, IERR )
2228:      end if
2229: C
2230:      if ( IERR.ne.0 ) then
2231:      write(IUPR,*) 'XXX(APFUNC) Failed to convert numeric data ',
2232:      &          ' to a character string in applying function to ',
2233:      &          K, ''th data.'
2234:      write(IUPR,*) ' function <', PPFUNC(:ICLEN2(PPFUNC)), '>'
2235:      NERR1 = NERR1 + 1
2236:      go to 130
2237:      end if
2238: C
2239:      LE = 0
2240:      do 120 I = 1, NP
2241:      if ( JVC(I).eq.0 ) then
2242:      I1 = IPOS(I)
2243:      I2 = IPOS(I+1) - 1
2244:      LK = IPOS(I+1) - IPOS(I)
2245:      ELINE(LE+1:LE+LK) = PPFUNC(I1:I2)
2246:      else
2247:      LK = LVV
2248:      ELINE(LE+1:LE+LK) = VCHAR(:LVV)
2249:      end if
2250:      LE = LE + LK
2251: 120 continue
2252: C
2253:      DD = DENTAK(ELINE(:LE),IUPR,IERR)
2254: C
2255:      if ( IERR.ne.0 ) then
2256:      write(IUPR,*) 'XXX(APFUNC) Failed to apply function',
2257:      &          ' to ', K, ''th data. ', VN(:LVN), ' = ',
2258:      &          VCHAR(:LVV)
2259:      write(IUPR,*) ' <', ELINE(:LE), '>'
2260:      NERR1 = NERR1 + 1
2261:      go to 130
2262:      end if
2263: C
2264:      if ( IRC.eq.1 ) then
2265:      IA(K) = NINT(DD)
2266:      else if ( IRC.eq.2 ) then
2267:      RA(K) = REAL(DD)
2268:      else
2269:      DA(K) = DD
2270:      end if
2271: 130 continue
2272: C
2273:      return
2274:      end
2275: C=====

```

```

2276:      subroutine APFNCI( IA,      N,      PPFUNC,IUPR,  NERR1 )
2277:      integer IA(N)
2278:      character*(*) PPFUNC
2279:      real*8 DDUM
2280:      IRC = 1
2281:      call APFUNC( IRC, IA, DUMMY, DDUM, N, PPFUNC, IUPR, NERR1 )
2282:      return
2283:      end
2284: C=====
2285:      subroutine APFNCR( RA,      N,      PPFUNC,IUPR,  NERR1 )
2286:      real RA(N)
2287:      character*(*) PPFUNC
2288:      real*8 DDUM
2289:      IRC = 2
2290:      call APFUNC( IRC, IDUMMY, RA, DDUM, N, PPFUNC, IUPR, NERR1 )
2291:      return
2292:      end
2293: C=====
2294:      subroutine APFNCD( DA,      N,      PPFUNC,IUPR,  NERR1 )
2295:      real*8 DA(N)
2296:      character*(*) PPFUNC
2297:      IRC = 3
2298:      call APFUNC( IRC, IDUMMY, DUMMY, DA, N, PPFUNC, IUPR, NERR1 )
2299:      return
2300:      end

```


src/utlils/fsigusr.c

```

1: /*****
2: *
3: *   To catch user defined signal 1 & 2 ( kill -USR[1|2] pid etc.)
4: *
5: *   Programmed by M.Sasaki ( The Japan Research Inst. Co. Ltd.)
6: *
7: *   fsigusr should be called somewhere to enable signal catching.
8: *
9: *****/
10:
11: #include <stdio.h>
12:
13: #ifdef __STDC__
14: #include <signal.h>
15:
16: /* Only for Fujitsu C Version 2.110 for Windows */
17: #ifndef SIGUSR1
18: #define SIGUSR1 0
19: #endif
20: #ifndef SIGUSR2
21: #define SIGUSR2 0
22: #endif
23:
24: #else
25: /** do not use signal for non-ANSI C **/
26: #define SIGUSR1 0
27: #define SIGUSR2 0
28: #define signal(a,b) " "
29: #endif
30:
31: /*****
32: *   Fortran callable C function :
33: *
34: *   UNIX vendors have their own convention or rule for external
35: *   names of FORTRAN coded subprograms or commons.
36: *
37: *   We prepare equivalent sources for C coded routines.
38: *
39: *   1. Attach underscore to FORTRAN name. ( SUN & other BSD UNIX)
40: *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
41: *   3. Uppercase string of FORTRAN name. ( Ncube etc.)
42: *   4. "_" + uppercase string of FORTRAN name. ( HI/USF. )
43: *
44: *****/
45:
46: /*****
47: *   Flags toggled zero/non-zero by user defined signals.
48: *****/
49:
50: static usr1_flag = 0 ;
51: static usr2_flag = 0 ;
52:
53: void fsigusr( )
54: {
55:     void FSIG_USR1 ( ) ;
56:     void FSIG_USR2 ( ) ;
57:     signal( SIGUSR1, FSIG_USR1 ) ;
58:     signal( SIGUSR2, FSIG_USR2 ) ;
59: }
60: void fsigusr_( )
61: {
62:     void FSIG_USR1 ( ) ;
63:     void FSIG_USR2 ( ) ;
64:     signal( SIGUSR1, FSIG_USR1 ) ;
65:     signal( SIGUSR2, FSIG_USR2 ) ;

```

```

66: }
67: void FSIGUSR( )
68: {
69:     void FSIG_USR1 ( ) ;
70:     void FSIG_USR2 ( ) ;
71:     signal( SIGUSR1, FSIG_USR1 ) ;
72:     signal( SIGUSR2, FSIG_USR2 ) ;
73: }
74: void _FSIGUSR( )
75: {
76:     void FSIG_USR1 ( ) ;
77:     void FSIG_USR2 ( ) ;
78:     signal( SIGUSR1, FSIG_USR1 ) ;
79:     signal( SIGUSR2, FSIG_USR2 ) ;
80: }
81:
82: /*****
83: *
84: *   Signal handler
85: *
86: *****/
87:
88: void FSIG_USR1 ( sig )
89:     int sig ;
90: {
91:     fflush( stdout ) ;
92:     printf("\n *** Received signal SIGUSR1 ***\n\n") ;
93:     fflush( stdout ) ;
94:     usr1_flag = usr1_flag ? 0 : 1 ;
95:     signal( SIGUSR1, FSIG_USR1 ) ;
96: }
97:
98: void FSIG_USR2 ( sig )
99:     int sig ;
100: {
101:     fflush( stdout ) ;
102:     printf("\n *** Received signal SIGUSR2 ***\n\n") ;
103:     fflush( stdout ) ;
104:     usr2_flag = usr2_flag ? 0 : 1 ;
105:     signal( SIGUSR2, FSIG_USR2 ) ;
106: }
107:
108: /*****
109: *
110: *   Functions to get flag-value of user defined signals in fortran
111: *
112: *****/
113: void FGETUSR_C( ) ;
114:
115: void fgetusrsig( signum, flag )
116:     int *signum ;
117:     int *flag ;
118: {
119:     FGETUSR_C( signum, flag ) ;
120: }
121:
122: void fgetusrsig_( signum, flag )
123:     int *signum ;
124:     int *flag ;
125: {
126:     FGETUSR_C( signum, flag ) ;
127: }
128:
129: void FGETUSRSIG( signum, flag )
130:     int *signum ;

```

src/utls/fsigusr.c

```
131: int *flag ;
132: {
133:     FGETUSR_C( signum, flag ) ;
134: }
135:
136: void _FGETUSRSIG( signum, flag )
137: int *signum ;
138: int *flag ;
139: {
140:     FGETUSR_C( signum, flag ) ;
141: }
142:
143:
144:
145: void FGETUSR_C( signum, flag )
146: int *signum ; /* user signal whose flag is get. 1/2 = USR1/USR2 */
147: int *flag ;
148: {
149:     *flag = 0 ;
150:     switch( *signum )
151:     {
152:         case 1 :
153:             *flag = usr1_flag ;
154:             break ;
155:
156:         case 2 :
157:             *flag = usr2_flag ;
158:             break ;
159:
160:         default :
161:             fflush( stdout ) ;
162:             printf("\n *** Tried to get undefined user signal %d.\n",
163:                 *signum );
164:     }
165: }
```

src/utls/ftgetenv.c

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4: /*****
5: *
6: * Get value of an environment variable from a Fortran program.
7: *
8: * Programmed by M.Sasaki (The Japan Research Inst.Ltd.) 1996
9: *
10: *
11: * Example:
12: *
13: *   character*128 value
14: *   ...
15: *   call ftgetenv( 'SRAC_IS_NOT_ZRAC', 16, value, len(value) )
16: *
17: *****/
18:
19: void FT_GETENV();
20:
21: void ftgetenv_( name, namelen, value, valuelen )
22:   char *name ;
23:   int *namelen ;
24:   char *value ;
25:   int *valuelen ;
26: {
27:   FT_GETENV( name, namelen, value, valuelen );
28: }
29: void ftgetenv( name, namelen, value, valuelen )
30:   char *name ;
31:   int *namelen ;
32:   char *value ;
33:   int *valuelen ;
34: {
35:   FT_GETENV( name, namelen, value, valuelen );
36: }
37: void FTGETENV( name, namelen, value, valuelen )
38:   char *name ;
39:   int *namelen ;
40:   char *value ;
41:   int *valuelen ;
42: {
43:   FT_GETENV( name, namelen, value, valuelen );
44: }
45: void _FTGETENV( name, namelen, value, valuelen )
46:   char *name ;
47:   int *namelen ;
48:   char *value ;
49:   int *valuelen ;
50: {
51:   FT_GETENV( name, namelen, value, valuelen );
52: }
53:
54: void FT_GETENV( name, namelen, value, valuelen )
55:   char *name ;
56:   int *namelen ;
57:   char *value ;
58:   int *valuelen ;
59: {
60:   char inquiry[512] ;
61:   char *vp ;
62:   int i, nl ;
63:
64:   memcpy( inquiry, name, *namelen ) ;
65:   inquiry[*namelen] = '\0' ;
```

```
66:
67:   vp = getenv( inquiry ) ;
68:   if ( vp == NULL )
69:   {
70:     /*** Fill with blanks if target env.variable i not defined **/
71:     memset( value, ' ', *valuelen ) ;
72:   }
73:   else
74:   {
75:     nl = strlen(vp) ;
76:     if( nl <= *valuelen )
77:     {
78:       memcpy( value, vp, nl) ;
79:       for( i=nl; i< *valuelen; ++i ) value[i] = ' ' ;
80:     }
81:     else
82:       memcpy( value, vp, *valuelen) ;
83:   }
84: }
```

src/utls/ftsystem.c

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4:
5: /*****
6: *
7: * Execute a shell command from a Fortran program.
8: *
9: * Programmed by M.Sasaki (The Japan Research Inst.Ltd.) 1996
10: *
11: * Using ANSI-C function system().
12: *
13: * Example:
14: *
15: * character*128 value
16: * ..
17: * call ftsystem( icode, 'banner MAKO', 11 )
18: *
19: *****/
20:
21: #define COMBUFFER 1024
22:
23: static char comsave[COMBUFFER] ;
24:
25: void FT_SYSTEM();
26:
27: void ftsystem( code, command, length )
28:   int *code ;
29:   char *command ;
30:   int *length ;
31: {
32:   FT_SYSTEM( code, command, length ) ;
33: }
34: void ftsystem_( code, command, length )
35:   int *code ;
36:   char *command ;
37:   int *length ;
38: {
39:   FT_SYSTEM( code, command, length ) ;
40: }
41: void FTSYSTEM( code, command, length )
42:   int *code ;
43:   char *command ;
44:   int *length ;
45: {
46:   FT_SYSTEM( code, command, length ) ;
47: }
48: void FTSYSTEM_( code, command, length )
49:   int *code ;
50:   char *command ;
51:   int *length ;
52: {
53:   FT_SYSTEM( code, command, length ) ;
54: }
55:
56: /*****
57:
58: void FT_SYSTEM( code, command, length )
59:   int *code ;
60:   char *command ;
61:   int *length ;
62: {
63:   if ( *length >= COMBUFFER ) {
64:     printf("xxx(FT_SYSTEM) too long command (limit is %d).\n",
65:     COMBUFFER) ;
```

```
66:     exit(1) ;
67:   }
68:
69:   memcpy( comsave, command , (size_t)(*length)) ;
70:   comsave[*length] = '\0' ;
71:   *code = system( comsave ) ;
72: }
```

src/utls/funbuf.c

```
1: /*****
2: *
3: *   To set standard output unbuffered using ANSI-C function setvbuf.
4: *
5: *   (Set as "line-buffered" saying exactly.
6: *   The real "unbuffered" mode may produce a confusing output
7: *   in parallel processing mode.)
8: *
9: *
10: *   Programmed by M.Sasaki ( The Japan Research Inst. Co. Ltd.)
11: *   ( July 1997 )
12: *
13: *****/
14:
15: #include <stdio.h>
16:
17: /*****
18: *   Fortran called C subroutine :
19: *
20: *   UNIX vendors have their own convention or rule for external
21: *   names of FORTRAN coded subprograms or commons.
22: *
23: *   We prepare equivalent sources for C coded routines
24: *
25: *   1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
26: *   2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
27: *   3. Uppercase string of FORTRAN name. ( Ncube etc. )
28: *   4. "_" + uppercase string of FORTRAN name. ( HI/OSF. )
29: *
30: *****/
31:
32: void funbuf_( )
33: {
34:     setvbuf( stdout, NULL, _IOLBF, 0 ) ;
35: }
36:
37: void funbuf( )
38: {
39:     setvbuf( stdout, NULL, _IOLBF, 0 ) ;
40: }
41:
42: void FUNBUF( )
43: {
44:     setvbuf( stdout, NULL, _IOLBF, 0 ) ;
45: }
46:
47: void _FUNBUF( )
48: {
49:     setvbuf( stdout, NULL, _IOLBF, 0 ) ;
50: }
```

src/utlils/getlin.f

```

1:      subroutine GETLIN( IUIN, IUPR, LINE,  NCHAR, IEND )
2:      C
3:      C  GMVP/MVP UTILITY
4:      C
5:      C=====
6:      C PURPOSE: INPUT A LINE FROM UNIT IUIN.
7:      C          (SKIP COMMENT LINE & COMMENT CHARACTERS)
8:      C          IUPR : message printout I/O unit
9:      C          IEND = -1 / 0 : end of input / no
10:     C          NCHAR : effective characters as input. ( 0 TO LEN(LINE))
11:     C CALLED IN : FREAD & MANY PLACE
12:     C=====
13:     character*(*) LINE
14:     C
15:     C ..... data for GETLIN routine
16:     C
17:     C GLPRPT : prompt string (effective until last non-blank character)
18:     C JPRPT : output prompt string (=1 with a blank, =2: no blank)
19:     C JECHO : =1 echoback input string (=2,with prompt)
20:     C
21:     common /GLDATC/ GLPRPT
22:     character*32 GLPRPT
23:     common /GLDATN/ JPRPT, JECHO
24:     C
25:     C .... DATA FOR VIRTUAL FILE ...
26:     C ( THE SAME COMMON DATA IN FREAD ROUTINE )
27:     C
28:     parameter( MXVLIN  = 32, MAXCOL  = 72 )
29:     common /VFREAD/ IOSAVE, ISSAVE, NVLINE, IVLINE
30:     common /VFILE/  LNSAVE
31:     character*(MAXCOL) LNSAVE(0:MXVLIN)
32:     C
33:     C-----
34:     C
35:     IEND  = 0
36:     NCHAR = 0
37:     LINE  = ' '
38:     C
39:     100 if ( IUIN.gt.0 ) then
40:     C
41:         if ( JPRPT.eq.1 ) then
42:         C---F90: no newline
43:         C      write(unit=IUPR,fmt='(1x,a,1x)',advance='NO')
44:         C      write(IUPR,fmt='(1x,a,1x)')
45:         C      &      GLPRPT(:ICLEN2(GLPRPT))
46:         C      else if ( JPRPT.eq.2 ) then
47:         C---F90: no newline
48:         C      write(unit=IUPR,fmt='(1x,a)',advance='NO')
49:         C      write(IUPR,fmt='(1x,a)')
50:         C      &      GLPRPT(:ICLEN2(GLPRPT))
51:         C      end if
52:     C
53:     read(IUIN,'(A)',end =110) LINE
54:     C
55:     if ( JECHO.eq.1 ) then
56:         write(IUPR,'(1x,a)') line(:iclen2(line))
57:     else if ( JECHO.eq.2 ) then
58:         if ( JPRPT.eq.1 ) then
59:             write(IUPR,'(1x,a,1x,a)')
60:         &      GLPRPT(:ICLEN2(GLPRPT)), line(:iclen2(line))
61:         else if ( JPRPT.eq.2 ) then
62:             write(IUPR,'(1x,a,a)')
63:         &      GLPRPT(:ICLEN2(GLPRPT)), line(:iclen2(line))
64:         end if
65:     end if

```

```

66:     C
67:     else
68:         IVLINE = IVLINE + 1
69:         if ( IVLINE.gt.NVLINE ) go to 110
70:         LINE   = LNSAVE(IVLINE)
71:     end if
72:     C
73:     C .... COMMENT .....
74:     C
75:     if ( LINE(1:1).eq.'*' ) go to 100
76:     C
77:     C .... SYMBOLIC PARAMETER DEFINITION ....
78:     C
79:     if ( LINE(1:1).eq.'%' ) then
80:         call PARA( LINE, IUPR )
81:         go to 100
82:     end if
83:     C
84:     K      = INDEX(LINE,'*')
85:     if ( K.eq.0 ) then
86:         NCHAR = LEN(LINE)
87:     else
88:         NCHAR = K - 1
89:         LINE(K:) = ' '
90:     end if
91:     if ( NCHAR.eq.0 ) go to 100
92:     return
93:     C
94:     110 IEND  = -1
95:     return
96:     end
97:     C
98:     C=====
99:     C
100:    subroutine GTLSET( PROMPT, JJPRPT, JJECHO )
101:    C=====
102:    C set prompt and echoback flag
103:    C=====
104:    character*(*) PROMPT
105:    C
106:    C ..... data for GETLIN routine
107:    C
108:    GLPRPT : prompt string (effective until last non-blank character)
109:    JPRPT : output prompt string (=1 with a blank, =2: no blank)
110:    JECHO : =1 echoback input string (=2,with prompt)
111:    C
112:    common /GLDATC/ GLPRPT
113:    character*32 GLPRPT
114:    common /GLDATN/ JPRPT, JECHO
115:    C
116:    GLPRPT = PROMPT
117:    JPRPT  = JJPRPT
118:    JECHO  = JJECHO
119:    C
120:    return
121:    end

```

src/utlils/header.f

```

1:      subroutine HEADER( IO,      CHAR )
2:      C
3:      C=<MVP/GMVP>=====
4:      C PAGE HEADER PRINTOUT
5:      C-----
6:      C argument :
7:      C i IO : printout I/O unit #
8:      C i CHAR : character string to be printed.
9:      C=====
10:     C
11:     character*(*) CHAR
12:     character*130 LINE, LINE2
13:     character*12 DT
14:     character*8 TM
15:     C
16:     character*32 BLANK
17:     C
18:     common /CMHEAD/ V, LV, RDT, LRDT
19:     character*32 V
20:     character*32 RDT
21:     Ccc data V/'
22:     Ccc data LV /8/
23:     C
24:     C=====
25:     C = = = = =
26:     C = MVP V.1 (CHARACTERS) = DATE = TIME =
27:     C = = = = =
28:     C=====
29:     C
30:     C
31:     write(IO,(''1'',13(''===== '')))
32:     C
33:     .... get version description ...
34:     C
35:     BLANK = ' '
36:     LINE = ' '//V(:LV) //' = '//CHAR
37:     LINE2 = ' '//BLANK(:LV) //' = '
38:     C
39:     call HIZUKE( DT )
40:     call JIKAN( TM )
41:     C
42:     do 100 I = LEN(DT), 1, -1
43:       if ( DT(I:I).ne.' ' ) go to 110
44:     100 continue
45:     110 LDT = I
46:     C
47:     C
48:     L = LEN(' ') + LDT + LEN(' ') + LEN(TM) + LEN(' ')
49:     LINE(130-L+1:) = ' '//DT(1:LDT) //' = '//TM//' = '
50:     C
51:     do 120 I = 130 - L + 1, 130
52:       if ( LINE(I:I).eq.' ' ) LINE2(I:I) = ' '
53:     120 continue
54:     C
55:     write(IO,('1X,A')) LINE2
56:     write(IO,('1X,A')) LINE
57:     write(IO,('1X,A')) LINE2
58:     C
59:     write(IO,('' ',13(''===== '')))
60:     C
61:     return
62:     end
63:     C
64:     C=====
65:     C purpose: set version description
66:     C (must be called before any calling of HEADER )
67:     C history: subroutine made from an entry of 'HEADER' routine.
68:     C (17 Sep 1994)
69:     C update:
70:     C 1 Feb 1999: accept release date string
71:     C=====
72:     subroutine HVERSN( VERSN, DATEST )
73:     C
74:     character*(*) VERSN
75:     character*(*) DATEST
76:     common /CMHEAD/ V, LV, RDT, LRDT
77:     character*32 V
78:     character*32 RDT
79:     C
80:     LV = LEN(VERSN)
81:     V = VERSN
82:     LRDT = LEN(DATEST)
83:     RDT = DATEST
84:     return
85:     end
86:     C=====
87:     C purpose: get version description and release date string
88:     C (must be called after any calling of HEADER )
89:     C history: Jan 1999
90:     C=====
91:     subroutine GETVER( VERSN, DATEST )
92:     C
93:     character*(*) VERSN
94:     character*(*) DATEST
95:     common /CMHEAD/ V, LV, RDT, LRDT
96:     character*32 V
97:     character*32 RDT
98:     C
99:     VERSN = V
100:    DATEST = RDT
101:    return
102:    end

```

src/utls/hizuke.f

```
1:      subroutine HIZUKE(XDATE)
2: C=====
3: C   Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C
5: C   In Japanese, "Hizuke" means the date.
6: C   ( Bug: should be "HIDUKE" exactly )
7: C-----
8: C
9: C RETURN DATE IN DD-MMM-YY (YY: YEAR  MMM: MONTH  DD: DAY)
10: C
11: C
12: C=====
13: C/IF CUTIL AND MS_VISUAL
14: C
15: C ... This interface block is for CUTIL & MS-Visual tools. ...
16: C
17: C   interface
18: *      subroutine CCTIMDAT(IH,IMIMN,ISEC,ID,IM,IY)
19: *      integer IH,IMIMN,ISEC,ID,IM,IY
20: *CDEC$  ATTRIBUTES C :: CCTIMDAT
21: *CDEC$  ATTRIBUTES REFERENCE :: IH,IMIMN,ISEC,ID,IM,IY
22: *      end subroutine
23: *      end interface
24: C/ENDIF
25:
26:      character*(*) XDATE
27:      character*3 AMONTH
28:
29: C/IF F90( DATE_AND_TIME )
30: C
31: C ..... When you can use standard intrinsic function DATE_AND_TIME
32: C      of FORTRAN90 ...
33: C
34:      character (LEN=8) DT
35:      call DATE_AND_TIME( DATE=dt )
36:      read(DT(5:6),'(I2)') IM
37:      XDATE = DT(7:8)//'- '//AMONTH(IM)//'- '//DT(1:4)
38:
39: C/#ELSEIF CUTIL
40: *      XDATE = ' '
41: *      call CCTIMDAT(IH,IMIMN,ISEC,ID,IM,IY)
42: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
43: C/#ELSEIF SYSTEM(FACOM* HIOSF)
44: *
45: C
46: C ----- IN FUJITU (FACOM) OR Hitachi SYSTEM --- YY-MM-DD
47: C
48: C --- can't get 4 digit year in this form ...
49: C
50: *      character*8 TDATE
51: *      XDATE = ' '
52: *      call DATE(TDATE)
53: *      read(TDATE,'(I2,1X,I2,1X,I2)') IY,IM,ID
54: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
55: *
56: C
57: C
58: C/#ELSEIF SYSTEM(HP*)
59: C
60: C ----- IN VMS & HP-9000 SERIES --- DD-MMM-YY
61: C
62: *$NOSTANDARD SYSTEM
63: *      XDATE = ' '
64: *      call IDATE(IM,ID,IY)
65: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
```

```
66: *
67: C/#ELSEIF SYSTEM(SGI*)
68: C
69: C ----- IN IRIX
70: C
71: *      XDATE = ' '
72: *      call IDATE(IM,ID,IY)
73: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
74: C
75: C/#ELSEIF SYSTEM(DECOSF* PARAGON*)
76: C
77: *      integer IAA(3)
78: *      XDATE = ' '
79: *      call IDATE(IAA(1),IAA(2),IAA(3))
80: *      write(XDATE,'(I2,''-''',a3,''-''',I4)')
81: *      & IAA(2),AMONTH(IAA(1)),LDGT24(IAA(3))
82: C
83: C/#ELSEIF SYSTEM(MIPS*)
84: C
85: *      integer IAA(3)
86: *      XDATE = ' '
87: *      call IDATE(IAA)
88: *      write(XDATE,'(I2,''-''',a3,''-''',I4)')
89: *      & IAA(1),AMONTH(IAA(2)),LDGT24(IAA(3))
90: C
91: C/#ELSEIF SYSTEM(VMS NECEWS*)
92: *
93: C
94: C ----- IN VMS --- DD-MMM-YY
95: C
96: C --- can't get 4 digit year in this form ...
97: C
98: *      XDATE = ' '
99: *      call DATE(XDATE(1:9))
100: *
101: C/#ELSEIF SYSTEM(SUN*)
102: *
103: *      call IDATE( ID, IM, IY )
104: *      XDATE = ' '
105: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
106: *
107: C/#ELSEIF SYSTEM(IBMRS* AIX*)
108: *
109: C
110: C --- IBM AIX --- EX. SUN SEP 16 01:32:52 1973
111: C .....+.....+.....+..... (24 CHARACTERS)
112: C
113: C      DD-MMM-YY
114: C
115: *      character*24 TT
116: *      call FDATE_(TT)
117: *      XDATE = TT(9:10)//'- '//TT(5:7)//'- '//TT(21:24)
118: *
119: C/#ELSEIF SYSTEM(CRAY*)
120: *
121: C
122: C ----- CRAY ----- MM/DD/YY
123: C
124: C --- can't get 4 digit year in this form ...
125: C
126: *      XDATE = ' '
127: *      call DATE(XDATE(1:8))
128: *      read(XDATE(1:8),'(I2,1X,I2,1X,I2)') IM,ID,IY
129: *      write(XDATE,'(I2,''-''',A3,''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
130: *
```


src/utls/hizuke.f

```
131: C/#ELSEIF SYSTEM(SX* ACOS)
132: *
133: C
134: C ----- NEC - SX or ACOS ----- MM/DD/YY
135: C
136: C
137: *      character*8      DUMMY
138: *      XDATE = ' '
139: *      call DATIM(XDATE(1:8),DUMMY,1)
140: *      read(XDATE(1:8),'(I2,1X,I2,1X,I2)') IY,IM,ID
141: *      write(XDATE,'(I2,'''-''',A3,'''-''',I4)') ID,AMONTH(IM),LDGT24(IY)
142: *
143: C/#ELSEIF SYSTEM(IBMVS)
144: *
145: C
146: C----- IBM VS-FORTRAN -----
147: C
148: *      integer IT(8)
149: *      call DATIM(IT)
150: *      XDATE = ' '
151: *      write(XDATE,'(I2,'''-''',A3,'''-''',I4)') IT(6),AMONTH(IT(7)),
152: *      &      LDGT24(IT(8))
153: *
154: C/#ELSE
155: *
156: *      XDATE = 'DATE?????'
157: *
158: C/#ENDIF
159:      return
160:      end
161: C
162: C ==== month ====
163: C
164:      character*3 function AMONTH(I)
165:      character*3 MONTH(0:12)
166:      data month /'???' , 'Jan' , 'Feb' , 'Mar' , 'Apr' , 'May' , 'Jun' , 'Jul' , 'Aug' ,
167:      &      'Sep' , 'Oct' , 'Nov' , 'Dec' /
168:      AMONTH = MONTH(I)
169:      return
170:      end
171: C
172: C ==== set two or three digit year to four digit. ====
173: C
174:      function LDGT24(IY)
175:      LDGT24 = IY
176:      if( IY.lt.1900 ) then
177: C
178: C      ... only last two digit of year and over year 2000
179:      if ( IY.lt.98 ) then
180:          LDGT24 = IY + 2000
181: C      ... year - 1900
182:      else
183:          LDGT24 = IY + 1900
184:      end if
185:      end if
186:      return
187:      end
```

src/utils/iclen2.f

```
1:      function ICLEN2(CH)
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C  Get position of the last non-blank character in a character string.
6: C=====
7:      character*(*) CH
8:      do 100 I = LEN(CH), 1, -1
9:          if ( CH(I:I).ne.' ' ) then
10:             ICLEN2 = I
11:             return
12:          end if
13: 100 continue
14:      ICLEN2 = 0
15:      return
16:      end
```

src/utls/iclen.f

```
1:      function ICLEN(CH)
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C  GET LENGTH OF FIRST NON-BLANK PART OF CHARACTER STRING.
6: C=====
7:      character*(*) CH
8:      ICLEN   = INDEX(CH,' ') - 1
9:      if ( ICLEN.lt.0 ) ICLEN = LEN(CH)
10:     return
11:     end
```

JAERI

src/utils/imatch.f

```
1:      function IMATCH(PATRN,STR)
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C purpose: function interface to 'MATCH' routine.
6: C-----
7: C arguments(i=input)
8: C i PATRN : matching pattern string.
9: C i STR   : string to be checked.
10: C-----
11:      character*(*) PATRN, STR
12: C
13:      call MATCH( IM, PATRN, STR )
14:      IMATCH = IM
15:      return
16:      end
```

src/utils/int8.f

```
1: C/#IF INTEGER8
2: *   integer*8 function INT8( III )
3: C/#ELSE
4:   integer   function INT8( III )
5: C/#ENDIF
6: C=====
7: C   Utility routine for MVP/GMVP system (JAERI)
8: C-----
9: C   Change integer declaration to integer*8.
10: C=====
11:   integer III
12:   INT8 = III
13:   return
14: end
15: C
16: C
17: C/#IF INTEGER8
18: *   integer*8 function R8TOI8( DDD )
19: C/#ELSE
20:   integer   function R8TOI8( DDD )
21: C/#ENDIF
22: C=====
23: C   Utility routine for MVP/GMVP system (JAERI)
24: C-----
25: C   Change real*8 declaration to integer*8.
26: C=====
27:   real*8 DDD
28:   R8TOI8 = DDD
29:   return
30: end
31: C
32: C
33:   integer   function I8TOI4( III8 )
34: C=====
35: C   Utility routine for MVP/GMVP system (JAERI)
36: C-----
37: C   Change integer*8 declaration to integer*4.
38: C=====
39: C/#IF INTEGER8
40: *   integer*8 III8
41: C/#ELSE
42:   integer   III8
43: C/#ENDIF
44:   I8TOI4 = III8
45:   return
46: end
47:
```

src/utls/jfopen.f

```

1: C =====
2: C
3: C Automatic file open/close function ( for UNIX systems )
4: C
5: C =====
6: C
7: C Interpret an input-character-string
8: C and open appropriate file by specified I/O unit number.
9: C
10: C string = /n[control characters][:file-name]
11: C
12: C where;
13: C
14: C n = I/O unit number (always necessary. 1 to 99)
15: C
16: C control-characters:
17: C
18: C < FORM parameter >
19: C
20: C n or U : open as an unformatted (binary?) file
21: C (default)
22: C
23: C f or F : open as a formatted (text) file
24: C
25: C < STATUS parameter >
26: C
27: C Default status is 'UNKNOWN'.
28: C
29: C s or S : scratch file (deleted on close operation)
30: C When an existing file name is specified, does
31: C not delete the file.
32: C
33: C r or R : Read only ('OLD' status)
34: C
35: C n or N : New file.
36: C
37: C When both control characters and filename are omitted,
38: C open an scratch file.
39: C
40: C Examples:
41: C
42: C /41:keno67.lib (or /41U:keno67.lib )
43: C
44: C ---> open file keno67.lib as unformatted file on unit 41
45: C
46: C /7F:text.dat
47: C
48: C ---> open file text.dat as formatted file on unit 7
49: C
50: C /10s or /10
51: C
52: C ---> open a scratch file on unit 10
53: C
54: C =====
55: C
56: C Sample program:
57: C
58: C
59: C Specify parameters on command line.
60: C
61: C /**** Main routine of a program *****/
62: C
63: C Program aaaaaa /** PROGRAM statement is necessary **/
64: C character*256 str
65: C k = iargc() /** get number of arguments by iargc **/
66: C
67: C do 100 i=2,k
68: C call getarg(i, str) /** get i'th argument **/
69: C j = jfopen( str )
70: C write(6,*) ' return code :',j
71: C 100 continue
72: C
73: C call prog /** The execution program !! **/
74: C
75: C j = jclose( 0 ) /** Close all file opened by jfopen **/
76: C stop
77: C end
78: C
79: C =====
80: C
81: C integer function JFOPEN(STRNG)
82: C character*(*) STRNG
83: C
84: C ===== memory of opened I/O units =====
85: C
86: C integer IOUNIT(100), NOPEN
87: C
88: C character NUM*10, FORMT*32
89: C
90: C
91: C character STATUS*16, FORM*16, IOMODE*16
92: C character HEAD
93: C logical OPD, LEXIST
94: C
95: C ===== data initialization =====
96: C
97: C data IOUNIT /100*0/, NOPEN /0/
98: C data NUM /'0123456789'/
99: C data HEAD /''/
100: C
101: C
102: C
103: C ===== execution starts here !! =====
104: C
105: C
106: C
107: C JFOPEN = 0
108: C
109: C ===== Effective data begin with '/' (HEAD). =====
110: C
111: C LNG = LEN(STRNG)
112: C if ( STRNG(1:1).ne.HEAD .or. LNG.lt.2 ) then
113: C write(6,*) ' == INVALID CHARACTERS PASSED TO JFOPEN !!'
114: C write(6,'(1X,A,A)') ' --> ', STRNG
115: C JFOPEN = -1
116: C return
117: C end if
118: C
119: C
120: C ===== get I/O unit # =====
121: C
122: C
123: C ..... find tail of numeric characters .....
124: C
125: C do 100 IN = 2, LNG
126: C if ( INDEX(NUM,STRNG(IN:IN)).eq.0 ) go to 110
127: C 100 continue
128: C 110 IN = IN - 1
129: C
130: C FORMT = ' '

```

src/utils/jfopen.f

```

131:      write(FORMT,'(''('I'',I1,'')'')'') IN - 1
132:      read(STRNG(2:IN),fmt =FORMT) IUNT
133: C
134:      if ( IUNT.lt.1 .or. IUNT.gt.99 ) then
135:        write(6,'(lx,a,i8,a)'') ' == INVALID I/O UNIT # ', IUNT,
136:      &      ' PASSED TO JFOPEN !!'
137:        write(6,'(lx,A,A)'') ' --> ', STRNG
138:        JFOPEN = -2
139:        return
140:      end if
141: C
142: C
143: C ===== Get control characters =====
144: C
145: C
146: C .... Filename starts after ':', if any. ....
147: C
148:      do 120 IC = IN + 1, LNG
149:        if ( STRNG(IC:IC).eq.':') go to 130
150: 120 continue
151: 130 IC = IC - 1
152: C
153: C
154: C
155: C ===== Set I/O control options =====
156: C
157: C .... default parameters ....
158: C
159:      STATUS = 'UNKNOWN'
160:      FORM = 'UNFORMATTED'
161:      IOMODE = 'READWRITE'
162: C
163: C
164:      if ( IC.gt.IN ) then
165:        do 140 I = IN + 1, IC
166: C
167:          if ( STRNG(I:I).eq.'f' .or. STRNG(I:I).eq.'F' ) then
168:            FORM = 'FORMATTED'
169: C
170:          else if ( STRNG(I:I).eq.'s' .or. STRNG(I:I).eq.'S' ) then
171:            STATUS = 'SCRATCH'
172: C
173:          else if ( STRNG(I:I).eq.'r' .or. STRNG(I:I).eq.'R' ) then
174:            STATUS = 'OLD'
175:            IOMODE = 'READONLY'
176: C
177:          else if ( STRNG(I:I).eq.'n' .or. STRNG(I:I).eq.'N' ) then
178:            STATUS = 'NEW'
179:          end if
180: 140 continue
181:        end if
182:        JJ = IC
183: C
184: C
185: C ===== Get file name (start / end position in strng ) ==
186: C
187: C
188:      IFS = JJ + 2
189:      IFE = 0
190:      if ( IFS.le.LNG ) then
191:        do 150 IFE = IFS, LNG
192:          if ( STRNG(IFE:IFE).eq.' ' ) go to 160
193: 150 continue
194: 160 IFE = IFE - 1
195:      end if

```

```

196: C
197: C .... No file name specified .....
198: C
199:      if ( IFE.lt.IFS ) STATUS = 'SCRATCH'
200: C
201: C
202: C ===== Open the file !! =====
203: C
204: C
205:      LLS = INDEX(STATUS,' ') - 1
206:      if ( LLS.lt.0 ) LLS = LEN(STATUS)
207:      LLF = INDEX(FORM,' ') - 1
208:      if ( LLF.lt.0 ) LLF = LEN(FORM)
209:      if ( IFE.ge.IFS ) then
210: C
211: C
212: C .... do not scratch existing file !! ...
213: C
214:        if ( STATUS.eq.'SCRATCH' ) then
215:          inquire(file =STRNG(IFS:IFE),exist =LEXIST)
216:          if ( LEXIST ) then
217:            STATUS = 'OLD'
218:            write(6,'(lx,a,a,a)'')
219:      &      '*** Does not scratch existing file <',
220:      &      STRNG(IFS:IFE), '>'
221:          end if
222:          LLS = INDEX(STATUS,' ') - 1
223:          if ( LLS.lt.0 ) LLS = LEN(STATUS)
224:        end if
225:
226:        write(6,'(lx,a,a,a,i3,a,a,a,a)'') '== OPEN FILE ',
227:      &      STRNG(IFS:IFE), ' ON I/O UNIT ', IUNT, ' (STATUS=',
228:      &      STATUS(:LLS), ' FORM=', FORM(:LLF), ' )'
229: C
230:      else
231:        write(6,'(lx,a,i3,a,a,a)'') '== OPEN SCRATCH FILE ON I/O UNIT ',
232:      &      IUNT, ' (FORM=', FORM(:LLF), ' )'
233:      end if
234: C
235: C
236: C
237:      if ( IFE.ge.IFS ) then
238: C
239: C/#IF READONLY( ACTION )
240: C
241: C ... read-only mode by ACTION='READ' ...
242: C ( SX3 ACOS FACOM CRAY etc. )
243: C
244:      if ( IOMODE.eq.'READ' ) then
245:        open( IUNT, form=FORM, status=STATUS, file=STRNG(IFS:IFE),
246:      &      iostat =IOS, action = 'READ' )
247:      else
248:        open( IUNT, form=FORM, status=STATUS, file=STRNG(IFS:IFE),
249:      &      iostat =IOS )
250:      end if
251: C
252: C/#ELSEIF READONLY(DEC)
253: C
254: C ... read-only mode by READONLY parameter (DEC VAX STYLE)...
255: C
256:      if ( IOMODE.eq.'READ' ) then
257:        open( IUNT, form =FORM, status =STATUS, file=STRNG(IFS:IFE),
258:      &      iostat =IOS, readonly )
259:      else
260:        open( IUNT, form =FORM, status =STATUS, file=STRNG(IFS:IFE),

```

src/utls/jfopen.f

```

261:      &          iostat =IOS )
262:      end if
263:
264: C/#ELSEIF READONLY(MODE)
265: C
266: C ... read-only mode by MODE parameter (Microsoft Fortran)...
267: C
268:      if ( IOMODE.eq.'READ' ) then
269:          open( IUNT, form =FORM, status =STATUS, file=STRNG(IFS:IFE),
270:              &          iostat =IOS, mode ='READ' )
271:      else
272:          open( IUNT, form =FORM, status =STATUS, file=STRNG(IFS:IFE),
273:              &          iostat =IOS )
274:      end if
275: C
276: C/#ELSE
277: C
278:      open( IUNT, form =FORM, status =STATUS, file =STRNG(IFS:IFE),
279:          &          iostat =IOS )
280: C
281:      if ( IOMODE.eq.'READ' ) then
282:          write(6,'(lx,a,a/lx,a)')
283:          &          '**** Currently READ-ONLY mode is really not effective',
284:          &          ' in the program.',
285:          &          '      Opened only in "OLD" status.'
286:      end if
287: C/#ENDIF
288:
289:      else
290:
291:          open( IUNT, form =FORM, status =STATUS, iostat =IOS )
292: C
293:          if ( IOMODE.eq.'READ' ) then
294:              write(6,'(lx,a,a,i3,a)')
295:          &          '**** READ-ONLY mode is meaningless for unnamed file.',
296:          &          ' (unit ', IUNT, ' )'
297:          end if
298: C
299:      end if
300: C
301: C
302:      if ( IOS.eq.0 ) then
303:          NOPEN = NOPEN + 1
304:          IOUNIT(NOPEN) = IUNT
305:      else
306:          write(6,7000) '*****'
307:          write(6,7020) '** FAILED TO OPEN FILE ON UNIT ', IUNT, ' ***'
308:          write(6,7020) '** ( ERROR CODE : ', IOS, ' ) **'
309:          write(6,7000) '*****'
310:      7000      format(lx,A)
311:      7020      format(lx,A,i4,A)
312: C
313:      if ( IFE.ge.IFS ) then
314:          inquire(file =STRNG(IFS:IFE),exist =LEXIST)
315:          if ( .not.LEXIST ) then
316:              write(6,'(lx,a,a,a)')
317:          &          '*** Specified file does not exist <',
318:          &          STRNG(IFS:IFE), '>'
319:          end if
320:      end if
321: C
322:      end if
323: C
324: C
325: C

```

```

326:      return
327: C
328: C ==== close specified I/O unit ===
329: C
330:      entry JCLOSE(IUNIT)
331: C
332: C iunit = 0 : close all units
333: C > 0 : close unit iunit
334: C
335:      JCLOSE = 0
336:      if ( IUNIT.gt.0 ) then
337:          close( IUNIT )
338:          write(6,'(lx,a,i3,a)') ' == I/O UNIT ', IUNIT, ' IS CLOSED.'
339:      else if ( IUNIT.eq.0 ) then
340:          do 170 I = 1, NOPEN
341:              inquire(IUNIT,opened =OPD)
342: C
343:              if ( OPD ) then
344:                  close( IOUNIT(I) )
345:                  write(6,'(lx,a,i3,a)') ' == I/O UNIT ', IOUNIT(I),
346:                  &          ' IS CLOSED. '
347:              end if
348:          170 continue
349:      end if
350:      return
351: C
352: C
353: C
354:      end

```


src/utls/jikan.f

```

1:      subroutine JIKAN( TIME8 )
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C  RETURN TIME  ( TIME8 = HOUR:MINUTE:SECOND )
6: C
7: C=====
8: C/#IF CUTIL.AND.MS_VISUAL
9: C
10: C ... This interface block is for CUTIL & MS-Visual tools. ...
11: C
12: C      interface
13: C      subroutine CCTIMDAT( IH, IMN, ISEC, ID, IM, IY )
14: C      integer IH, IMN, ISEC, ID, IM, IY
15: C*DEC$  ATTRIBUTES C :: CCTIMDAT
16: C*DEC$  ATTRIBUTES REFERENCE :: IH, IMN, ISEC, ID, IM, IY
17: C      and subroutine
18: C      end interface
19: C/#ENDIF
20: C      character*(*) TIME8
21: C
22: C/#IF F90( DATE_AND_TIME )
23: C
24: C      .... When you can use standard intrinsic function DATE_AND_TIME
25: C      of FORTRAN90 ...
26: C
27: C      character (LEN=10) TT
28: C      call DATE_AND_TIME( TIME=TT )
29: C      TIME8 = TT(1:2)//':'//TT(3:4)//':'//TT(5:6)
30: C
31: C/#ELSEIF CUTIL
32: C      call CCTIMDAT( IH, IMN, ISEC, ID, IM, IY )
33: C      write(TIME8,'(I2,':'',I2,':'',I2)') IH, IMN, ISEC
34: C      do 100 I=1,LEN(TIME8)
35: C          if( TIME8(I:I).eq.' ' ) TIME8(I:I) = '0'
36: C      100 continue
37: C
38: C/#ELSEIF SYSTEM(HP* VAX)
39: C
40: C      HH:MM:SS
41: C
42: C      character*8 TT
43: C      call TIME(TT)
44: C      TIME8 = TT
45: C
46: C/#ELSEIF SYSTEM(DECOSF* SGI* PARAGON*)
47: C
48: C --- DEC Alpha OSF/1 ---
49: C
50: C      integer IT(3)
51: C      call ITIME( IT )
52: C      write(TIME8,'(I2,':'',I2,':'',I2)') IT(1),IT(2),IT(3)
53: C      do 100 I=1,LEN(TIME8)
54: C          if( TIME8(I:I).eq.' ' ) TIME8(I:I) = '0'
55: C      100 continue
56: C
57: C/#ELSEIF SYSTEM(SUN* MIPS*)
58: C
59: C --- SUN --- EX. SUN SEP 16 01:32:52 1973
60: C      ..... (24 CHARACTERS)
61: C
62: C      character*24 TT
63: C      call FDATE(TT)
64: C      TIME8 = TT(12:19)
65: C

```

```

66: C/#ELSEIF SYSTEM(IBMRS* AIX*)
67: C
68: C --- IBM AIX --- EX. SUN SEP 16 01:32:52 1973
69: C      ..... (24 CHARACTERS)
70: C
71: C      character*24 TT
72: C      call FDATE_(TT)
73: C      TIME8 = TT(12:19)
74: C
75: C/#ELSEIF SYSTEM(NECEWS*)
76: C
77: C --- NEC EWS --- hh:mm:ss
78: C
79: C      call TIME(TIME8)
80: C
81: C/#ELSEIF SYSTEM(CRAY*)
82: C
83: C----- CRAY ----- HH:MM:SS
84: C
85: C
86: C      character*8 TT
87: C      call CLOCK(TT)
88: C      TIME8 = TT
89: C
90: C/#ELSEIF SYSTEM(ACOS SX*)
91: C
92: C
93: C----- NEC-SX ----- HH:MM:SS
94: C
95: C
96: C      character*8 DT
97: C      character*8 TT
98: C      call DATIM(DT,TT,1)
99: C      TIME8 = TT
100: C
101: C/#ELSEIF SYSTEM(FACOM*)
102: C
103: C
104: C----- FACOM ----- SYSTEM TIME IN MILLISECOND
105: C
106: C
107: C      call TIME(IITIME)
108: C      IITIME = IITIME/1000
109: C      IHOURL = IITIME/3600
110: C      IMINT = (IITIME - IHOURL*3600)/60
111: C      ISEC = IITIME - IHOURL*3600 - IMINT*60
112: C      write(TIME8,'(G2.2,':'',G2.2,':'',G2.2)') IHOURL,IMINT,ISEC
113: C
114: C/#ELSEIF SYSTEM(IBMVS)
115: C
116: C
117: C----- IBM VS-FORTRAN ---
118: C
119: C      integer IT(8)
120: C      call DATIM(it)
121: C      write(TIME8,'(I2,':'',I2,':'',I2)') IT(5),IT(4),IT(3)
122: C      do 100 I=1,LEN(TIME8)
123: C          if( TIME8(I:I).eq.' ' ) TIME8(I:I) = '0'
124: C      100 continue
125: C
126: C/#ELSEIF SYSTEM(HIOSF)
127: C
128: C
129: C----- Hitachi S series ---
130: C

```

src/utils/jikan.f

```
131: *      character*12 TT
132: *      call CLOCK(TT,1)
133: *      TIME8 = TT(1:2)///': '//TT(4:5)///': '//TT(7:8)
134: C/#ELSE
135: *
136: *      TIME8 = '00:00:00'
137: *
138: C/#ENDIF
139: C
140:      return
141:      end
```

SAE

src/utls/label.f

```
1:      subroutine LABEL( IO,      STR )
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C  PURPOSE : PRINT ITEM LABELS (STR) ON UNIT IO.
6: C  CALLED IN: EVERYWHERE
7: C  CALLS   : (NONE)
8: C=====
9:      character*(*) STR
10: C
11:      character*1 BORDR
12:      data BORDR /' ' /
13:      data MERGN /6/
14: C
15:      L = LEN(STR)
16:      K = 1 + MERGN + L + MERGN + 1
17: C
18: C ..... LABEL WRITTEN .....
19: C
20: C
21: C *****
22: C *      STRINGS      *
23: C *****
24: C
25: C
26:      write(IO, '(/)')
27: C
28:      write(IO,7000) (BORDR,I=1,K)
29: 7000 format(' ',4X,120(:A1))
30: C
31:      write(IO,7020) BORDR, (' ',I=1,MERGN), STR, (' ',I=1,MERGN), BORDR
32: 7020 format(' ',4X,30(:A))
33: C
34:      write(IO,7000) (BORDR,I=1,K)
35: C
36:      write(IO, '(/)')
37: C
38:      return
39:      end
```

src/utlis/lfmalloc.c

```
1: #ifdef LFMALLOC_C
2:
3: #include <stdlib.h>
4:
5: /*****
6: *
7: *   Interface routine to the 'malloc' function from FORTRAN.
8: *
9: *   Programmed by M.Sasaki ( Mar 1994 )
10: *
11: *****/
12:
13: /*****
14: *   Fortran called C subroutine :
15: *
16: *   UNIX vendors have their own convention or rule for external
17: *   names of FORTRAN coded subprograms or commons.
18: *
19: *   We prepare equivalent sources for C coded routines
20: *
21: *       1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
22: *       2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
23: *       3. Uppercase string of FORTRAN name. ( Ncube etc. )
24: *       4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
25: *
26: *****/
27:
28:
29: void * lfmalloc_ ( nbytes )
30:     size_t * nbytes ;
31: {
32:     return (void *) malloc( *nbytes ) ;
33: }
34:
35: void * lfmalloc ( nbytes )
36:     size_t * nbytes ;
37: {
38:     return (void *) malloc( *nbytes ) ;
39: }
40:
41: void * LFMALLOC ( nbytes )
42:     size_t * nbytes ;
43: {
44:     return (void *) malloc( *nbytes ) ;
45: }
46:
47: void * _LFMALLOC ( nbytes )
48:     size_t * nbytes ;
49: {
50:     return (void *) malloc( *nbytes ) ;
51: }
52:
53: #else
54:
55: void * lfmalloc_ ( nbytes )
56:     int * nbytes ;
57: {
58:     return (void *) 0 ;
59: }
60:
61: #endif
```

src/utlils/match.f

```

1:      subroutine MATCH( IM,   PATTRN,STR )
2: C=====
3: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C PURPOSE: STRING PATTERN MATCHING
6: C-----
7: C arguments(i=input, o=output)
8: C
9: C o IM      : returns 1 if matched else 0
10: C i PATTRN : matching pattern string.
11: C i STR     : string to be checked.
12: C=====
13: C
14: C  PATTRN & STR CAN INCLUDE BLANKS BUT THEY MUST BE PLACED AFTER ANY
15: C  NON-BLANK CHARACERS. ( ignore characters after blank !! )
16: C
17: C      IM = 1 / 0 ... match / unmatch
18: C
19: C
20: C << WILDCARD CHARACTERS >>
21: C
22: C  '?' : ANY ONE CHARACTER ON THE PLACE OF '?'.
23: C  '*' : ANY CHARACTER STRING ( length >= 0 ).
24: C
25: C=====
26: C      character*(*) PATTRN
27: C      character*(*) STR
28: C
29: C      .... default is unmatch ...
30: C
31: C      IM      = 0
32: C
33: C      LP      = INDEX(PATTRN,' ') - 1
34: C      if ( LP.lt.0 ) LP = LEN(PATTRN)
35: C
36: C      LS      = INDEX(STR,' ') - 1
37: C      if ( LS.lt.0 ) LS = LEN(STR)
38: C      if ( LS.eq.0 ) return
39: C
40: C
41: C
42: C      .... pattern has no '*' .....
43: C
44: C
45: C
46: C      if ( INDEX(PATTRN(:LP),'*').eq.0 ) then
47: C          if ( LP.ne.LS ) return
48: C          do 100 I = 1, LP
49: C              if ( PATTRN(I:I).ne.'?' .and. PATTRN(I:I).ne.STR(I:I) ) return
50: C          100 continue
51: C          IM      = 1
52: C          return
53: C      end if
54: C
55: C
56: C      .... pattern has '*' ....
57: C
58: C
59: C << Logic of pattern matching with wild-card character '*' >>
60: C
61: C
62: C
63: C 0. Set "current pattern" to the whole pattern string.
64: C   And set "object string" to the string checked.
65: C

```

```

66: C 1. Split the current-pattern string to two sub-patterns by '*'.
67: C
68: C      pattern = L * R
69: C             or L      ( if the current pattern has no '*'. )
70: C
71: C      L : left-side sub-pattern (does not include '*'. can be
72: C        null string)
73: C      R : right-side sub-pattern (may include '*')
74: C
75: C 2. If the left-side sub-pattern is not null string,
76: C    check matching with the left-side sub-pattern.
77: C
78: C    If the sub-pattern does not match any substring of the object
79: C    string (the leftmost substring for the first time), terminate as
80: C    'unmatching'.
81: C
82: C    Else cut off the matching substring and characters on the left
83: C    of the substring from the object string and proceed to step 4.
84: C
85: C 3. If the pattern has no '*',
86: C    check matching between the current pattern and the rightmost
87: C    substring of the object string.
88: C
89: C    If matched ,terminate matching as 'matching',else terminate
90: C    matching as 'unmatching'.
91: C
92: C 4. Set the right-side sub-pattern as the current pattern and
93: C    repeat the process from 1.
94: C
95: C
96: C      IP      = 1
97: C      IS      = 1
98: C
99: C 110 continue
100: C
101: C
102: C      K      = INDEX(PATTRN(IP:LP),'*')
103: C
104: C
105: C ==== check matching with the left side of current '*'
106: C
107: C
108: C      if ( K.gt.1 ) then
109: C
110: C          IS1      = LS - K + 2
111: C
112: C      .... for the first matching, the sub-pattern must match
113: C      at the leftmost position.
114: C
115: C      if ( IP.eq.1 ) IS1 = IS
116: C      do 130 ISS = IS, IS1
117: C          do 120 I = 0, K - 2
118: C              if ( PATTRN(IP+I:IP+I).ne.'?'
119: C                & .and. PATTRN(IP+I:IP+I).ne.STR(ISS+I:ISS+I) ) go to 130
120: C          120 continue
121: C
122: C      ---- left side matches ----
123: C
124: C          IS      = ISS + I
125: C          go to 150
126: C
127: C 130 continue
128: C
129: C ---- unmatched ----
130: C

```

src/utls/match.f

```
131:         return
132: C
133: C
134: C  === check the rightmost string if no '*' remains in pattern ===
135: C
136: C
137:         else if ( K.eq.0 ) then
138:             if ( LS-IS.lt.LP-IP ) return
139:             do 140 I = IP, LP
140:                 if ( PATTRN(I:I).ne.'?'
141:                     & .and.PATTRN(I:I).ne.STR(LS+I-LP:LS+I-LP) ) return
142:             140 continue
143:             IM      = 1
144:             return
145:         end if
146: C
147: C  (if the left-side sub-pattern is null, do only the following
148: C  step.)
149: C
150:         150 IP      = IP + K
151:         if ( IP.le.LP ) go to 110
152: C
153: C  ---- matched !! ---
154: C
155:         IM      = 1
156: C
157:         return
158:     end
```

src/utills/msort.f

```

1:      subroutine MSORT( NN,      R,      I1,      RWRK, IWRK )
2: C=====
3: C purpose: sorting of array. (recursive merge sort)
4: C=====
5: C arguments ( i = input, o=output, w = work )
6: C
7: C i NN : length of array sorted.
8: C io r(NN) : array r(*) and I1(*) is sorted in ascending order of R(*).
9: C io il(NN) : sorted in ascending order of R(*).
10: C w rwrk(NN): working array
11: C w iwrk(NN): working array
12: C=====
13: C
14:      real R(NN)
15:      real RWRK(NN)
16: C
17:      integer I1(NN)
18:      integer IWRK(NN)
19: C
20: C-----
21: C Sorting by a recursive merge sort.
22: C Order of processing time is expected to be O(n*log(n)).
23: C-----
24: C
25:      MM      = 1
26: C
27:      100 continue
28: C
29:      M0      = MM
30:      MM      = MM*2
31:      if ( M0.lt.NN ) then
32:      do 140 K = 1, NN, MM
33:          L      = K
34:          M      = K + M0
35:          L2     = MIN(M-1,NN)
36:          M2     = MIN(K+MM-1,NN)
37:          KK     = K
38:      110      continue
39: C
40: C-----
41: C already sorted two sequences R(L:L2) & R(M:M2) are merged
42: C-----
43: C
44:      if ( L.le.L2.and.M.le.M2 ) then
45:      if ( R(L).gt.R(M) ) then
46:          RWRK(KK) = R(M)
47:          IWRK(KK) = I1(M)
48:          M        = M + 1
49:      else
50:          RWRK(KK) = R(L)
51:          IWRK(KK) = I1(L)
52:          L        = L + 1
53:      end if
54:      KK          = KK + 1
55:      if ( KK.le.M2 ) go to 110
56: C
57:      else if ( L.gt.L2.and.M.le.M2 ) then
58: C
59:      do 120 I = M, M2
60:          RWRK(KK+I-M) = R(I)
61:          IWRK(KK+I-M) = I1(I)
62:      120      continue
63: C
64:      else if ( L.le.L2.and.M.gt.M2 ) then
65: C

```

```

66:      do 130 I = L, L2
67:          RWRK(KK+I-L) = R(I)
68:          IWRK(KK+I-L) = I1(I)
69:      130      continue
70:      end if
71: C
72:      140      continue
73: C
74:      do 150 I = 1, NN
75:          R(I) = RWRK(I)
76:          I1(I) = IWRK(I)
77:      150      continue
78: C
79:      go to 100
80:      end if
81: C
82:      return
83:      end
84: C
85:      subroutine MSORTD( NN,      R,      I1,      RWRK, IWRK )
86: C=====
87: C purpose: sorting of double precision array. (recursive merge sort)
88: C history: programmed by M.Sasaki (July 1997)
89: C=====
90: C arguments ( i = input, o=output, w = work )
91: C
92: C i NN : length of array sorted.
93: C io R(NN) : array R(*) and I1(*) is sorted in ascending order of R(*).
94: C io il(NN) : sorted in ascending order of R(*).
95: C w rwrk(NN): working array
96: C w iwrk(NN): working array
97: C=====
98: C
99:      real*8 R(NN)
100:      real*8 RWRK(NN)
101: C
102:      integer I1(NN)
103:      integer IWRK(NN)
104: C
105: C-----
106: C Sorting by a recursive merge sort.
107: C Order of processing time is expected to be O(n*log(n)).
108: C-----
109: C
110:      MM      = 1
111: C
112:      100 continue
113: C
114:      M0      = MM
115:      MM      = MM*2
116:      if ( M0.lt.NN ) then
117:      do 140 K = 1, NN, MM
118:          L      = K
119:          M      = K + M0
120:          L2     = MIN(M-1,NN)
121:          M2     = MIN(K+MM-1,NN)
122:          KK     = K
123:      110      continue
124: C
125: C-----
126: C already sorted two sequences R(L:L2) & R(M:M2) are merged
127: C-----
128: C
129:      if ( L.le.L2.and.M.le.M2 ) then
130:      if ( R(L).gt.R(M) ) then

```

src/utils/msort.f

```

131:          RWRK(KK) = R(M)
132:          IWRK(KK) = I1(M)
133:          M = M + 1
134:        else
135:          RWRK(KK) = R(L)
136:          IWRK(KK) = I1(L)
137:          L = L + 1
138:        end if
139:        KK = KK + 1
140:        if ( KK.le.M2 ) go to 110
141: C
142:        else if ( L.gt.L2.and.M.le.M2 ) then
143: C
144:          do 120 I = M, M2
145:            RWRK(KK+I-M) = R(I)
146:            IWRK(KK+I-M) = I1(I)
147:          120 continue
148: C
149:        else if ( L.le.L2.and.M.gt.M2 ) then
150: C
151:          do 130 I = L, L2
152:            RWRK(KK+I-L) = R(I)
153:            IWRK(KK+I-L) = I1(I)
154:          130 continue
155:        end if
156: C
157:        140 continue
158: C
159:        do 150 I = 1, NN
160:          R(I) = RWRK(I)
161:          I1(I) = IWRK(I)
162:        150 continue
163: C
164:        go to 100
165:      end if
166: C
167:      return
168:    end
169: C
170:    subroutine MSORTI( NN, IR, I1, IRWRK, IWRK )
171: C=====
172: C purpose: sorting of integer array. (recursive merge sort)
173: C history: programmed by M.Sasaki (July 1997)
174: C=====
175: C arguments ( i = input, o=output, w = work )
176: C
177: C i NN : length of array sorted.
178: C io IR(NN) : array IR(*) and I1(*) is sorted in ascending order of
179: C IR(*).
180: C io i1(NN) : sorted in ascending order of IR(*).
181: C w irwrk(NN): working array
182: C w iwrk(NN): working array
183: C=====
184: C
185: C integer IR(NN)
186: C integer IRWRK(NN)
187: C
188: C integer I1(NN)
189: C integer IWRK(NN)
190: C
191: C-----
192: C Sorting by a recursive merge sort.
193: C Order of processing time is expected to be O(n*log(n)).
194: C-----
195: C

```

```

196:          MM = 1
197: C
198:        100 continue
199: C
200:          M0 = MM
201:          MM = MM*2
202:          if ( M0.lt.NN ) then
203:            do 140 K = 1, NN, MM
204:              L = K
205:              M = K + M0
206:              L2 = MIN(M-1,NN)
207:              M2 = MIN(K+MM-1,NN)
208:              KK = K
209:            110 continue
210: C
211: C -----
212: C already sorted two sequences IR(L:L2) & IR(M:M2) are merged
213: C -----
214: C
215:          if ( L.le.L2.and.M.le.M2 ) then
216:            if ( IR(L).gt.IR(M) ) then
217:              IRWRK(KK) = IR(M)
218:              IWRK(KK) = I1(M)
219:              M = M + 1
220:            else
221:              IRWRK(KK) = IR(L)
222:              IWRK(KK) = I1(L)
223:              L = L + 1
224:            end if
225:            KK = KK + 1
226:            if ( KK.le.M2 ) go to 110
227: C
228:          else if ( L.gt.L2.and.M.le.M2 ) then
229: C
230:            do 120 I = M, M2
231:              IRWRK(KK+I-M) = IR(I)
232:              IWRK(KK+I-M) = I1(I)
233:            120 continue
234: C
235:          else if ( L.le.L2.and.M.gt.M2 ) then
236: C
237:            do 130 I = L, L2
238:              IRWRK(KK+I-L) = IR(I)
239:              IWRK(KK+I-L) = I1(I)
240:            130 continue
241:          end if
242: C
243:          140 continue
244: C
245:          do 150 I = 1, NN
246:            IR(I) = IRWRK(I)
247:            I1(I) = IWRK(I)
248:          150 continue
249: C
250:          go to 100
251:        end if
252: C
253:      return
254:    end

```


src/utlils/mtxinv.f

```

1:      subroutine MTXINV( A,      N,      IP,      W,      EPS,      IERR )
2: c=====
3: C   Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: c-----
5: C PURPOSE: MATRIX INVERSION ( DOUBLE PRECISION )
6: C COMMENT: INVERSION BY SIMPLE LU DECOMPOSITION WITH GAUSS ELIMINATION.
7: C           USE FOR LOW ORDER MATRIX ONLY!!
8: C
9: c=====
10:      implicit real*8(A-H,O-Z)
11: C
12: C   == (N,N) MATRIX INVERTED ==
13: C
14:      real*8 A(N,N)
15: C
16: C   == (N,N) MATRIX FOR WORKING SPACE ==
17: C
18:      real*8 W(N,N)
19: C
20: C   == WORKING ARRAY FOR PIVOTTING ==
21: C
22:      integer IP(N)
23: C
24:      IERR      = 0
25:      if ( EPS.lt.0.0 ) EPS      = 1.0D-20
26: C
27: C -----
28: C   ( I,J ) -- (ROW, COLUMN)
29: C
30: C   AX = B      OR  A A = E
31: C
32: C   PA X = PB   ( P : ROW EXCHANGE MATRIX )
33: C
34: C   PA  = LU   ( LU DECOMPOSITION )
35: C
36: C   -1      -1      -1      -1      -1
37: C   A      = ( P LU )      = U L P
38: C
39: C   L      --> A  --|      -1
40: C   U      --> W  --+-----> A  --> A
41: C
42: C -----
43: C   LU DECOMPOSITION FOR A
44: C -----
45: C
46: C   do 160 K = 1, N - 1
47: C
48: C
49: C
50: C
51: C   .... SELECT PIVOTTING ROW  IP(K)  ....
52: C
53: C   DMX      = 0.0
54: C   IPPP      = K
55: C
56: C   do 100 I = K, N
57: C       DD      = ABS(A(I,K))
58: C       if ( DD.gt.DMX ) then
59: C           IPPP      = I
60: C           DMX      = DD
61: C       end if
62: C   100 continue
63: C   IP(K)      = IPPP
64: C
65: C   .... EXCHANGE ROW K AND PIVOTTING ROW ....

```

```

66: C
67: C   if ( IP(K).ne.K ) then
68: C       do 110 J = 1, N
69: C           DT      = A(IPPP,J)
70: C           A(IPPP,J) = A(K,J)
71: C           A(K,J)   = DT
72: C   110 continue
73: C   end if
74: C
75: C   .... SIMPLE GAUSS ELIMINATION ....
76: C
77: C   DP      = A(K,K)
78: C   if ( ABS(DP).le.EPS ) then
79: C       IERR      = 1
80: C       return
81: C   end if
82: C
83: C   do 120 I = K + 1, N
84: C       A(I,K)    = -A(I,K) /DP
85: C   120 continue
86: C
87: C   do 140 J = 1, N
88: C       if ( J.ne.K ) then
89: C           DM      = A(K,J)
90: C           do 130 I = K + 1, N
91: C               A(I,J) = A(I,J) + A(I,K)*DM
92: C           130 continue
93: C       end if
94: C   140 continue
95: C
96: C   do 150 J = K, N
97: C       W(K,J)    = A(K,J)
98: C       A(K,J)    = 0.0
99: C   150 continue
100: C   A(K,K)      = 1.0
101: C
102: C   160 continue
103: C
104: C   W(N,N)      = A(N,N)
105: C   A(N,N)      = 1.0
106: C   IP(N)       = N
107: C
108: C -----
109: C -----
110: C   -1 -1      -1 -1
111: C   U L = W A  ---> A
112: C -----
113: C
114: C   do 200 K = N, 1, -1
115: C
116: C   if ( ABS(W(K,K)).le.EPS ) then
117: C       IERR      = 2
118: C       return
119: C   end if
120: C
121: C   do 170 J = 1, N
122: C       A(K,J)    = A(K,J) /W(K,K)
123: C   170 continue
124: C
125: C   do 190 I = 1, K - 1
126: C
127: C       do 180 J = 1, N
128: C           A(I,J) = A(I,J) - A(K,J)*W(I,K)
129: C       180 continue
130: C

```

src/utls/mtxinv.f

```
131: 190 continue
132: C
133: 200 continue
134: C
135: C-----
136: C      -1      -1      -1      -1 -1
137: C      A      =  (P LU)      = U L P
138: C-----
139: C
140:      do 220 K = N, 1, -1
141:      if ( IP(K).ne.K ) then
142:      do 210 I = 1, N
143:      DD      = A(I,IP(K))
144:      A(I,IP(K)) = A(I,K)
145:      A(I,K) = DD
146: 210 continue
147:      end if
148: 220 continue
149:
150:      return
151:      end
```

src/utlils/mvctoi.f

```
1:      subroutine MVCTOI( N,      CHAR,  INTA )
2: C==<MVP/GMVP>=====
3: C purpose: byte-wise data transfer from character string to integer.
4: C called in: data packet handling routines
5: C=====
6:
7: C/#IF CHARARG(LOOSE)
8: C
9: C .....
10: C   For compilers using loose character argument passing rule
11: C   which allows passing a non-character real argument as a character
12: C   real argument or vice versa.)
13: C .....
14: C
15: C      character*1 CHAR(*)
16: C      character*1 INTA(*)
17: C      do 100 I = 1, N
18: C          INTA(I) = CHAR(I)
19: C      100 continue
20: C
21: C/#ELSE
22: C
23: C .....
24: C   For compilers using rigorous character argument passing rule
25: C   which does not allow passing a non-character real argument as
26: C   for a character dummy argument or vice versa.
27: C .....
28: C
29: C      character*(*) CHAR
30: C      integer INTA(*)
31: C
32: C   ... CWRK must have the same byte size as that of single word data
33: C   of your system.
34: C
35: C/#IF WORD(64)
36: C      parameter( LBWORD = 8 )
37: C/#ELSE
38: C      parameter( LBWORD = 4 )
39: C/#ENDIF
40: C      character*(LBWORD) CWRK
41: C
42: C      character*6 FORMT
43: C -----
44: C
45: C      FORMT = ' '
46: C      write(FORMT, '( '(a'',il,'')' )' ) LBWORD
47: C      IC = 0
48: C      do 100 K = 1, N, LBWORD
49: C          K2 = MIN(N,K+LBWORD-1)
50: C          IC = IC + 1
51: C          CWRK = CHAR(K:K2)
52: C          read(CWRK,fmt =FORMT) INTA(IC)
53: C      100 continue
54: C
55: C/#ENDIF
56: C
57: C      return
58: C      end
```

src/utls/mvitoc.f

```
1:      subroutine MVITOC( N,      CHAR,  INTA )
2: C==<MVP/GMVP>=====
3: C purpose: byte-wise data transfer from integer to character string .
4: C called in: data packet handling routine
5: C=====
6:
7: C/#IF CHARARG(LOOSE)
8: C
9: C .....
10: C   For compilers using loose character argument passing rule
11: C   which allows passing a non-character argument as a character
12: C   argument or vice versa.)
13: C .....
14: C
15: C   character*1 CHAR(*)
16: C   character*1 INTA(*)
17: C   do 100 I = 1, N
18: C       CHAR(I) = INTA(I)
19: C 100 continue
20: C
21: C/#ELSE
22: C
23: C .....
24: C   For compilers using rigorous character argument passing rule
25: C   which does not allow passing a non-character argument as
26: C   a character dummy argument or vice versa.
27: C
28: C   Someone may say that putting the character dummy argument at the end
29: C   of argument list remedy this situation, but this routine has been
30: C   used in many places of MVP/GMVP and related programs...
31: C .....
32: C
33: C   character*(*) CHAR
34: C   integer INTA(*)
35: C
36: C   ... CWRK must have the same byte size as that of single word data
37: C   of your system.
38: C
39: C/#IF WORD(64)
40: C   parameter( LBWORD = 8 )
41: C/#ELSE
42: C   parameter( LBWORD  = 4 )
43: C/#ENDIF
44: C   character*(LBWORD) CWRK
45: C
46: C   character*6 FORMT
47: C -----
48: C
49: C   FORMT = ' '
50: C   write(FORMT, '(a', il, ',') ) LBWORD
51: C   IC = 0
52: C   do 100 K = 1, N, LBWORD
53: C       K2 = MIN(N,K+LBWORD-1)
54: C       IC = IC + 1
55: C       CWRK = ' '
56: C       write(CWRK,fmt =FORMT) INTA(IC)
57: C       CHAR(K:K2) = CWRK
58: C 100 continue
59: C
60: C/#ENDIF
61: C
62: C   return
63: C   end
```

src/utlils/noblkn.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine NOBLNK( LINE,  IS,    IE,    NL )
3: C
4: C   JAERI MONTE CARLO CODE UTILITY
5: C
6: C=====
7: C   PURPOSE: FIND NONBLANK CHARACTER STRING IN 'LINE'.
8: C
9: C       SEARCH LINE(IS:NL)
10: C   AND RETURNS STARTING POSITION AND END POSITION OF NON-BLANK STRING.
11: C   IN IS & IE RESPECTIVELY.
12: C
13: C   IF no non-blank character atom is found, IS=IE > NL are returned.
14: C
15: C
16: C   CALLED IN: KONEIN   ETC.
17: C=====
18:   character*(*) LINE
19:   if ( IS.gt.NL ) then
20:     IE      = IS
21:     return
22:   end if
23:   ISS      = 0
24:   do 100 I = IS, NL
25:     if ( LINE(I:I).ne.' ' .and.ISS.eq.0 ) ISS      = I
26:     if ( LINE(I:I).eq.' ' .and.ISS.ne.0 ) go to 110
27:   100 continue
28:   if ( ISS.eq.0 ) then
29:     IS      = I
30:     IE      = I
31:     return
32:   end if
33:  110 IS      = ISS
34:     IE      = I - 1
35:   return
36:   end
```

src/utls/openf.f

```
1:      subroutine OPENF(IO, FILE, FORM, STATUS, ACTION, IERR)
2: C
3: C   GMVP/MVP UTILITY
4: C
5: C=====
6: C PURPOSE:  Open a file on specified I/O unit with parameters.
7: C   Users can use this routine in stead of writing the OPEN
8: C   statements in their program source.
9: C   A merit of this routine is that enables use of system dependent
10: C   open-statement parameters without changing all many OPEN staements.
11: C
12: C   File that can be opened:
13: C       Non-direct access  AND
14: C       * Named file  ( status = NEW ,OLD , UNKNOWN , SCRATCH )
15: C       * Non-named scratch-file
16: C
17: C   ACTION = 'READ' or other ('WRITE' etc.)
18: C
19: C-----
20: C
21:      character*(*) FILE, FORM, STATUS, ACTION
22:      character*16 STAT
23:
24:      IERR      = 0
25: C
26: C----- FILE NAME IS GIVEN -----
27: C
28:      if ( FILE.ne.' ' ) then
29:
30:          STAT      = STATUS
31:
32: C/# IF SYSTEM(FACOM*)
33: C .... FACOM ONLY ...
34: C *      if( STAT .eq. 'OLD' ) STAT = 'SHR'
35: C/# ENDDIF
36:
37:      if ( ACTION.eq.'READ' ) then
38:
39: C/#IF READONLY( DEC )
40: C
41: C ... readonly specification by DEC type ...
42: C
43: C *      open(IO, file =FILE, status =STAT, form =FORM, iostat =
44: C *      &      IERR, readonly)
45: C
46: C/#ELSEIF READONLY(ACTION)
47: C
48: C ... readonly specification by ACTION keyword ...
49: C
50: C *      open(IO, file =FILE, status =STAT, form =FORM, iostat =
51: C *      &      IERR, action ='READ')
52: C
53: C/#ELSEIF READONLY(MODE)
54: C
55: C ... readonly specification by MODE keyword ...
56: C
57: C *      open(IO, file =FILE, status =STAT, form =FORM, iostat =
58: C *      &      IERR, mode ='READ')
59: C
60: C/#ELSE
61: C
62: C ... no readonly specification ...
63: C
64:      open(IO, file =FILE, status =STAT, form =FORM, iostat =IERR)
65: C/#ENDDIF
```

```
66:      else
67:          open(IO, file =FILE, status =STAT, form =FORM, iostat =IERR)
68:      end if
69:  else
70: C
71: C----- FILE NAME IS NOT GIVEN -> TAKEN AS SCRATCH FILE -----
72: C
73:      if ( STATUS.eq.'SCRATCH' ) then
74:          open( IO, status ='SCRATCH', form =FORM, iostat =IERR )
75:      end if
76:  end if
77:  return
78:  end
```

src/utils/para.f

```
1: *VOCL TOTAL,SCALAR
2:   subroutine PARA( LINE, IUPR )
3: C=====
4: C PURPOSE:   Processing of parameter line in input
5: C CALLED FROM: GETLIN
6: C CALLS:    DENTAK
7: C
8: C   <INPUT EXAMPLE>
9: C
10: C      %   A= 123 , B= 0.1, C=B-SQRT(4*A)      /*  COMMENT
11: C
12: C=====
13:   character LINE*(*)
14:   real*8 DENTAK, D
15: cccc include 'INC/_IOUNIT'
16: C
17: C ===== CHARACTERS AFTER /* ARE MEANINGLESS =====
18: C
19:   LP      = LEN(LINE)
20:   K       = INDEX(LINE,'/*')
21:   if ( K.ne.0 ) LP      = K - 1
22: C
23: C == START INTERPRETATION FROM 2'ND CHARACTER IF '%' IS ON 1'ST COLUMN.
24: C
25:   IS      = 1
26:   if ( LINE(1:1).eq.'%' ) IS = 2
27: C
28: C
29:   D       = DENTAK(LINE(IS:LP),IUPR,IERR)
30: C
31:   if ( IERR.ne.0 ) then
32:     write(IUPR,*) 'XXX FATAL ERROR IN CALCULATOR'
33:     call CNTERR( 'FATAL' )
34:   end if
35: C
36: C
37:   return
38: end
```

src/utils/ranu2.f

```

1: C/#IF SSL2
2: C
3: C ... WHEN YOUR SYSTEM HAS SSL2-RANU2 ROUTINE ...
4: C
5: C ... create double precision random number using RANU2
6: C
7: C SUBROUTINE DRANU2( IX, D, N, ierr )
8: C
9: C real*8 D(N)
10: C
11: C PARAMETER (NN = 2048)
12: C real WORK(0:NN)
13: C do 100 i=1,N,NN
14: C i2 = min(nn,i+nn-1)
15: C NR = i2 - i + 1
16: C call ranu2(ix, work, nr, ierr )
17: C do 200 j=0,nr-1
18: C D(i+j) = WORK(j)
19: C 200 continue
20: C 100 continue
21: C return
22: C end
23: C
24: * subroutine RNINIT( NRSTEP )
25: * return
26: C end
27: C
28: C/#ELSEIF LCG(SIMPLE)
29: C
30: C=====
31: C This algorithm can be used for power-of-2 moduli.
32: C=====
33: * subroutine RANU2( IRAND, R, N, IERR )
34: *
35: * implicit none
36: *
37: * include '../shared/INC/_RAND'
38: *
39: C/# IF PARA( SX* )
40: * local common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
41: C/# ELSEIF PARA( CRAY* )
42: * task common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
43: C/# ELSE
44: * common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
45: C/# ENDIF
46: *
47: * integer*8 AAA, CCC
48: * real SMALL, RNEAR1
49: * integer INIT
50: C
51: C ... dummy arguments ...
52: C
53: * integer IRAND, N, IERR
54: * real R(N)
55: *
56: C ..... 63-bit LCG : multiplier = 9219741426499971445, increment = 1
57: C period = 2**63
58: *
59: * integer*8 AAA0, CCC0
60: * integer*8 MASK
61: * real*8 MODLS, NORM
62: *
63: C/#IF LCGMOD( 31BIT )
64: C ..... 31bit LCG
65: * parameter( MODLS = 2.d0**31 )

```

```

66: * parameter( AAA0 = 32771 )
67: * parameter( CCC0 = 1234567891 )
68: C 2**31 - 1 = 2147483647
69: * parameter( MASK = 2147483647 )
70: C/#ELSE
71: C ..... 63bit LCG
72: * parameter( MODLS = 2.d0**63 )
73: * parameter( AAA0 = 9219741426499971445_8 )
74: * parameter( CCC0 = 1 )
75: C 2**63 - 1 = 9223372036854775807
76: * parameter( MASK = 9223372036854775807_8 )
77: C/#ENDIF
78: * parameter( NORM = 1.d0/MODLS )
79: C
80: C ... local variables ...
81: C
82: * integer I, IB
83: * real MEPS, RNO
84: *
85: * integer*8 SEED
86: * data SEED / 1 /
87: * save SEED
88: *
89: C data RNCTR / 0.0d0 /
90: *
91: * if ( RNCTR.eq.0.0d0 ) then
92: * SEED = IRAND
93: * else
94: * SEED = IRNSU*(2_8**42) + IRNSM*(2_8**21) + IRNSL
95: * end if
96: C
97: C --- initialization stage ---
98: C
99: * if ( INIT.eq.1 ) then
100: * INIT = 0
101: * AAA = AAA0
102: * CCC = CCC0
103: C-----
104: C Calculate machine epsilon or the smallest number (SMALL) &
105: C and the nearest number to unity.
106: C-----
107: C
108: C ==== RNEAR1 :
109: C
110: C/#IF RANDOM( BOUND )
111: * RNEAR1 = 0.999999
112: * RNO = RNEAR1
113: * MEPS = 1.0
114: * do 110 IB = 1, 64
115: * MEPS = MEPS*0.5
116: * RNO = 1.0 - MEPS
117: * if ( RNO.eq.1.0 ) go to 120
118: * RNEAR1 = RNO
119: * 110 continue
120: * 120 continue
121: C/#ENDIF
122: C
123: C ==== SMALL : value setting is currently in "trial and error" stage.
124: C
125: C ... possible smallest random number
126: C
127: C SMALL = 1.0D0/MODLS
128: * SMALL = NORM
129: C
130: C ... computer epsilon

```


src/utils/ranu2.f

```

131: C
132: ccccc SMALL = 1.0 - RNEAR1
133: *
134: *   end if
135: *
136: *   do 100 I = 1, N
137: *       SEED = iand( AAA*SEED+CCC, MASK )
138: *       R(I) = SEED*NORM
139: C/#IF RANDOM( BOUND )
140: *       R(I) = MIN( RNEAR1, MAX(SMALL,R(I)))
141: C/#ELSEIF .NOT.RANDOM( ALLOWZERO )
142: *       if ( R(I).eq.0.0 ) R(I) = 1.0E-30
143: C/#ENDIF
144: *       RNCTR = RNCTR + 1.0d0
145: *   100 continue
146: *
147: c =<memo>= The seed value is saved as common variables
148: c           for a restart calculation. This treatment will be
149: c           modified in the future.
150: *
151: *       IRNSU = int(SEED/2_8**42)
152: *       IRNSM = int((SEED - IRNSU*2_8**42)/2_8**21)
153: *       IRNSL = SEED - IRNSU*2_8**42 - IRNSM*2_8**21
154: *
155: c ... print the number of used RNS ...
156: c   write(*,*) ' RANU2 : N=',N,' RNCTR=',RNCTR
157: *
158: *   return
159: *   end
160: C
161: *   subroutine RNINIT( NRSTEP )
162: *
163: *   implicit none
164: *
165: *   include '../shared/INC/_RAND'
166: *
167: C/# IF PARA( SX* )
168: *   local common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
169: C/# ELSEIF PARA( CRAY* )
170: *   task common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
171: C/# ELSE
172: *   common /CMRAND/ AAA, CCC, SMALL, RNEAR1, INIT
173: C/# ENDIF
174: *
175: *   integer*8 AAA, CCC
176: *   real      SMALL, RNEAR1
177: *   integer   INIT
178: *
179: *   integer*8 AAA0, CCC0
180: *   integer*8 MASK
181: *   real*8     MODLS
182: *
183: *   integer   NRSTEP
184: *
185: *   integer   K
186: *   integer*8 SUMA, PWRA
187: *
188: C/#IF LCGMOD( 31BIT )
189: c ..... 31bit LCG
190: *   parameter( MODLS = 2.d0**31 )
191: *   parameter( AAA0 = 32771 )
192: *   parameter( CCC0 = 1234567891 )
193: c   2**31 - 1 = 2147483647
194: *   parameter( MASK = 2147483647 )
195: C/#ELSE

```

```

196: c ..... 63bit LCG
197: *   parameter( MODLS = 2.d0**63 )
198: *   parameter( AAA0 = 9219741426499971445_8 )
199: *   parameter( CCC0 = 1 )
200: c   2**63 - 1 = 9223372036854775807
201: *   parameter( MASK = 9223372036854775807_8 )
202: C/#ENDIF
203: *
204: *   INIT = 0
205: *   RNCTR = 0.d0
206: *
207: *   SUMA = 1
208: *   PWRA = 1
209: *
210: *   if ( NRSTEP.ne.1 ) then
211: *       do 100 K = 1, NRSTEP-1
212: *           PWRA = iand( PWRA*AAA0, MASK )
213: *           SUMA = iand( SUMA+PWRA, MASK )
214: *       100 continue
215: *   end if
216: *
217: *   AAA = iand( PWRA*AAA0, MASK )
218: *   CCC = iand( SUMA*CCC0, MASK )
219: *
220: *   return
221: C   end
222: C/#ELSE
223: C
224: C=====
225: C Vectorizable random number generator routine.
226: C-----
227: C
228: C=====
229: C
230: C The subroutine coded here is designed to simulate the "RANU2" routine
231: C of the SSL-II subroutine library of Fujitsu Ltd.
232: C
233: C-----
234: C Function:
235: C
236: C To give a series of random numbers uniformly distributed between
237: C zero and 1.0 by the mixed congruential method.
238: C
239: C-----
240: C Arguments:
241: C
242: C   irand : A non-negative integer smaller than 2**31
243: C           used as a seed of random numbers .
244: C           Value of an integer random number that gives
245: C           the last floating point random number
246: C           is given .
247: C
248: C   r ,dr : an array of floating point number in which
249: C           generated random numbers are stored.
250: C   n      : number of random numbers to be generated.
251: C           users must prepare a memory of (at least) n-words
252: C           as the array r by himself !!)
253: C   ierr   : Error code (meaningless in this routine)
254: C
255: C-----
256: C
257: C   subroutine RANU2( IRAND, R, N, IERR )
258: C
259: c   implicit real*8(A-H,O-Z)
260: c   implicit none

```

src/utils/ranu2.f

```

261:
262:      include '../shared/INC/_RAND'
263: c
264: c ... dummy arguments ...
265: c
266:      integer IRAND, N, IERR
267:      real R(N)
268: c
269: c
270: c ... parameters ...
271: c
272:      integer NV
273:      parameter( NV = 2048 )
274: c
275: c
276: c -----
277: c
278: c << Properties of this random number generator >>
279: c
280: c * Generation parameters:
281: c
282: c      xmulti   : 32771 ( prime number )
283: c      xadd      : 1234567891 ( prime number )
284: c
285: c      2**31 - xadd = 912915757 = 13*43*1633123
286: c
287: c
288: c * Measured periods:
289: c
290: c      irand(period) = irand(0)
291: c
292: c -----
293: c      Initial      Period
294: c -----
295: c      0            1,073,741,824 (2**30)
296: c      1            1,073,741,824 (2**30)
297: c      1633123      1,073,741,824 (2**30)
298: c      19940401     1,073,741,824 (2**30)
299: c -----
300: c
301: c -----
302: c
303: c
304: c /#IF LCGMOD( 31BIT )
305: c
306: c      MVP/GMVP original RNG = Fujitsu SSL-II
307: c ... 31-bit LCG : multiplier = 32771, increment = 1234567891
308: c
309: c      integer MM, KK
310: c      real*8 DPM, DK, DMK
311: c      parameter( MM = 31, KK = 16 )
312: c      parameter( DPM = (2.0D0)**MM )
313: c      parameter( DK = (2.0D0)**KK )
314: c      parameter( DMK = (2.0D0)**(MM-KK) )
315: c
316: c      real*8 XMULTI, XADD
317: c      parameter( XMULTI = 32771D0 )
318: c      parameter( XADD = 1234567891D0 )
319: c /#ELSE
320: c
321: c ... 63-bit LCG : multiplier = 9219741426499971445, increment = 1
322: c      period = 2**63
323: c
324: c /# IF INTEGER8( RANU2 )
325: c      real*8 DPM

```

```

326: c      integer*8 AAA, CCC, MASK
327: c      real*8 NORM
328: c
329: c ... 63bit LCG ...
330: c      parameter( DPM = (2.0d0)**63 )
331: c      parameter( AAA = 9219741426499971445_8 )
332: c      parameter( CCC = 1 )
333: c      2**63 - 1 = 9223372036854775807
334: c      parameter( MASK = 9223372036854775807_8 )
335: c      parameter( NORM = 1.d0/2.d0**63 )
336: c
337: c ... 16bit LCG ...
338: c      parameter( DPM = (2.0d0)**16 )
339: c      parameter( AAA = 18389 )
340: c      parameter( CCC = 1 )
341: c      2**16 - 1 = 65535
342: c      parameter( MASK = 65535 )
343: c      parameter( NORM = 1.d0/DPM )
344: c
345: c ... 8bit LCG ...
346: c      parameter( DPM = (2.0d0)**8 )
347: c      parameter( AAA = 141 )
348: c      parameter( CCC = 1 )
349: c      2**8 - 1 = 255
350: c      parameter( MASK = 255 )
351: c      parameter( NORM = 1.d0/DPM )
352: c /# ELSE
353: c      real*8 DK1, DK2, DPM
354: c      parameter( DK1 = (2.0d0)**21 )
355: c      parameter( DK2 = (2.0d0)**42 )
356: c      parameter( DPM = (2.0d0)**63 )
357: c
358: c      real*8 AU, AM, AL, CU, CM, CL
359: c      parameter( AU = 2096326 )
360: c      parameter( AM = 1037627 )
361: c      parameter( AL = 1829237 )
362: c      parameter( CU = 0 )
363: c      parameter( CM = 0 )
364: c      parameter( CL = 1 )
365: c /# ENDDIF
366: c /#ENDDIF
367: c
368: c
369: c /# IF PARA( SX* )
370: c
371: c /# IF LCGMOD( 31BIT )
372: c      local common /CMRAND/ X0, X1, Y, SMALL, RNEAR1, INIT
373: c /# ELSE
374: c /# IF INTEGER8( RANU2 )
375: c      local common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
376: c /# ELSE
377: c      local common /CMRAND/ XU, XM, XL, YU, YM, YL,
378: c      & SMALL, RNEAR1, INIT
379: c /# ENDDIF
380: c /# ENDDIF
381: c
382: c /# ELSEIF PARA( CRAY* )
383: c
384: c /# IF LCGMOD( 31BIT )
385: c      task common /CMRAND/ X0, X1, Y, SMALL, RNEAR1, INIT
386: c /# ELSE
387: c /# IF INTEGER8( RANU2 )
388: c      task common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
389: c /# ELSE
390: c      task common /CMRAND/ XU, XM, XL, YU, YM, YL,

```

src/utils/ranu2.f

```

391: *      &                SMALL, RNEAR1, INIT
392: C/#      ENDIF
393: C/#      ENDIF
394: *
395: C/# ELSE
396:
397: C/#      IF LCGMOD( 31BIT )
398: *      common /CMRAND/ X0,      X1,      Y,      SMALL, RNEAR1, INIT
399: C/#      ELSE
400: C/#      IF INTEGER8( RANU2 )
401: *      common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
402: C/#      ELSE
403: *      common /CMRAND/ XU, XM, XL, YU, YM, YL, SMALL, RNEAR1, INIT
404: C/#      ENDIF
405: C/#      ENDIF
406:
407: C/# ENDIF
408: C
409: c
410: c ... common variables & arrays ...
411: c
412: C/#IF LCGMOD( 31BIT )
413: *      real*8 X0(NV), X1(NV), Y(NV)
414: C/#ELSE
415: C/# IF INTEGER8( RANU2 )
416: *      integer*8 X(NV), Y(NV)
417: C/# ELSE
418: *      real*8 XU(NV), XM(NV), XL(NV), YU(NV), YM(NV), YL(NV)
419: C/# ENDIF
420: C/#ENDIF
421: *      real      SMALL, RNEAR1
422: *      integer INIT
423: c
424: c ... local variables ...
425: c
426: C/#IF LCGMOD( 31BIT )
427: *      real*8 IX1, IX0
428: *      real*8 DX, DD, DI
429: C/#ELSE
430: C/# IF INTEGER8( RANU2 )
431: *      integer*8 SEED
432: *      data      SEED / 1 /
433: *      save      SEED
434: *      integer*8 TSEED, TTT
435: C/# ELSE
436: *      real*8 SU, SM, SL
437: *      save      SU, SM, SL
438: C/# ENDIF
439: C/#ENDIF
440: *      real      MEPS, RN0
441:
442: *      integer I, IB, K, J, M, K1
443:
444: c      data      RNCTR / 0.0d0 /
445:
446: C/#IF .NOT.LCGMOD( 31BIT )
447: C/# IF .NOT.INTEGER8( RANU2 )
448: *      real*8 VAL1X, QQQ1X, RRR1X, VAL2X, QQQ2X, RRR2X,
449: *      &      VAL3X, QQQ3X, RRR3X
450: *      real*8 VAL1Y, QQQ1Y, RRR1Y, VAL2Y, QQQ2Y, RRR2Y,
451: *      &      VAL3Y, QQQ3Y, RRR3Y
452: *      real*8 VAL1S, QQQ1S, RRR1S, VAL2S, QQQ2S, RRR2S,
453: *      &      VAL3S, QQQ3S, RRR3S
454: C/# ENDIF
455: C/#ENDIF

```

```

456:
457: C/#IF .NOT.LCGMOD( 31BIT )
458: c ... IRAND is split into 3 parts. ...
459: c      IRAND = IRNSU*2**42 + IRNSM*2**21 + IRNSL
460:
461: C/# IF INTEGER8( RANU2 )
462: *      SEED      = IRNSU*(2_8**42) + IRNSM*(2_8**21) + IRNSL
463: C/# ELSE
464: *      SU        = IRNSU
465: *      SM        = IRNSM
466: *      SL        = IRNSL
467: C/# ENDIF
468:
469: C/#ENDIF
470: C
471: C --- initialization stage ----
472: C
473: *      if ( INIT.eq.1 ) then
474: *      INIT      = 0
475: C/#IF LCGMOD( 31BIT )
476: *      X1(1)     = INT(XMULTI/DK)
477: *      X0(1)     = XMULTI - DK*X1(1)
478: *      Y(1)      = XADD
479: C/#ELSE
480: C/# IF INTEGER8( RANU2 )
481: *      X(1)      = AAA
482: *      Y(1)      = CCC
483: C/# ELSE
484: *      XU(1)     = AU
485: *      XM(1)     = AM
486: *      XL(1)     = AL
487: *      YU(1)     = CU
488: *      YM(1)     = CM
489: *      YL(1)     = CL
490: C/# ENDIF
491: C/#ENDIF
492:
493: C-----
494: C
495: C ==== Random Number Generation Logic (vector) for 31-bit LCG ====
496: C
497: C      (dk = 2**16)
498: C
499: C      x(0) = 1
500: C      x(i) = x1(i)*dk + x0(i) == (x1(i-1)*dk + x0(i-1))*xmulti (mod 2**31)
501: C
502: C      y(0) = 0
503: C      y(i) == y(i-1)*xmulti + xadd (mod 2**31)
504: C
505: C
506: C      and r(i) = MOD (seed * x(i) + y(i), 2**31) / 2**31
507: C
508: C-----
509:
510: *      do 100 I = 2, NV
511: C/#IF LCGMOD( 31BIT )
512: *      DD      = X0(I-1)*XMULTI
513: *      DI      = INT(DD/DK)
514: *      X0(I)   = DD - DI*DK
515: *      X1(I)   = MOD(MOD(X1(I-1)*XMULTI,DMK)+DI,DMK)
516: *      Y(I)    = MOD(MOD(Y(I-1)*XMULTI,DPM)+XADD,DPM)
517: C/#ELSE
518: C/# IF INTEGER8( RANU2 )
519: *      X(I)    = iand( AAA*X(I-1), MASK )
520: *      Y(I)    = iand( AAA*Y(I-1)+CCC, MASK )
521: C/# ELSE

```

src/utils/ranu2.f

```

521:      VAL1X = AL*XL(I-1)
522:      QQQ1X = int(VAL1X/DK1)
523:      RRR1X = VAL1X - DK1*QQQ1X
524:
525:      VAL2X = AM*XL(I-1) + AL*XM(I-1) + QQQ1X
526:      QQQ2X = int(VAL2X/DK1)
527:      RRR2X = VAL2X - DK1*QQQ2X
528:
529:      VAL3X = AU*XL(I-1) + AM*XM(I-1) + AL*XU(I-1) + QQQ2X
530:      QQQ3X = int(VAL3X/DK1)
531:      RRR3X = VAL3X - DK1*QQQ3X
532:
533:      XU(I) = RRR3X
534:      XM(I) = RRR2X
535:      XL(I) = RRR1X
536:
537:      VAL1Y = AL*YL(I-1) + CL
538:      QQQ1Y = int(VAL1Y/DK1)
539:      RRR1Y = VAL1Y - DK1*QQQ1Y
540:
541:      VAL2Y = AM*YL(I-1) + AL*YM(I-1) + CM + QQQ1Y
542:      QQQ2Y = int(VAL2Y/DK1)
543:      RRR2Y = VAL2Y - DK1*QQQ2Y
544:
545:      VAL3Y = AU*YL(I-1) + AM*YM(I-1) + AL*YU(I-1) + CU + QQQ2Y
546:      QQQ3Y = int(VAL3Y/DK1)
547:      RRR3Y = VAL3Y - DK1*QQQ3Y
548:
549:      YU(I) = RRR3Y
550:      YM(I) = RRR2Y
551:      YL(I) = RRR1Y
552: C/# ENDIF
553: C/#ENDIF
554:
555:      100      continue
556:
557: C
558: C-----
559: C Calculate machine epsilon or the smallest number (SMALL) &
560: C and the nearest number to unity.
561: C-----
562: C
563: C ==== RNEAR1 :
564: C
565: C/#IF RANDOM( BOUND )
566: *      RNEAR1 = 0.999999
567: *      RN0 = RNEAR1
568: *      MEPS = 1.0
569: *      do 110 IB = 1, 64
570: *          MEPS = MEPS*0.5
571: *          RN0 = 1.0 - MEPS
572: *          if ( RN0.eq.1.0 ) go to 120
573: *          RNEAR1 = RN0
574: * 110      continue
575: * 120      continue
576: C/#ENDIF
577: C
578: C ==== SMALL : value setting is currently in "trial and error" stage.
579: C
580: C ... possible smallest random number
581: C
582:      SMALL = 1.0D0/DPM
583: C
584: C ... computer epsilon
585: C

```

```

586: ccccc      SMALL = 1.0 - RNEAR1
587: C
588:      end if
589:
590:      do 140 K = 1, N, NV
591:          M = MIN(N-K+1,NV)
592:          K1 = K - 1
593: C
594: C ----- It is desirable to declare ix1 & ix0 as double precision
595: C for vector supercomputers such as VP, SX . -----
596: C
597: C/#IF LCGMOD( 31BIT )
598: *          IX1 = INT(IRAND/DK)
599: *          IX0 = IRAND - IX1*DK
600: C/#ENDIF
601:
602:      do 130 J = 1, M
603: C
604: C/#IF LCGMOD( 31BIT )
605: C
606: C-----
607: C .... REAL*8 data must have at least 47 bit precision in mantissa
608: C to keep 31 bit precision in the following calculation.
609: C
610: C Mantissa precision in various 64 bit floating point number expressions
611: C
612: C      IEEE      : 53-54 bits
613: C      IBM       : 56-53 bits (hexa)
614: C      Cray      : 48-49 bits (Default REAL)
615: C      VAX D_FLOATING : 55-56 bits
616: C      VAX G_FLOATING : 52-53 bits
617: C-----
618: C
619: *          DX = (Y(J)+IX0*X0(J)) /DPM + (IX0*X1(J)+IX1*X0(J)) /DMK
620: *          DX = DX - INT(DX)
621: *          R(K1+J) = DX
622: C/#ELSE
623: C/# IF INTEGER8( RANU2 )
624: *          TSEED = iand( X(J)*SEED+Y(J), MASK )
625: *          R(K1+J) = dble(TSEED)*NORM
626: C/# ELSE
627:          VAL1S = XL(J)*SL + YL(J)
628:          QQQ1S = int(VAL1S/DK1)
629:          RRR1S = VAL1S - DK1*QQQ1S
630:
631:          VAL2S = XM(J)*SL + XL(J)*SM + YM(J) + QQQ1S
632:          QQQ2S = int(VAL2S/DK1)
633:          RRR2S = VAL2S - DK1*QQQ2S
634:
635:          VAL3S = XU(J)*SL + XM(J)*SM + XL(J)*SU + YU(J) + QQQ2S
636:          QQQ3S = int(VAL3S/DK1)
637:          RRR3S = VAL3S - DK1*QQQ3S
638:
639:          R(K1+J) = RRR3S/DK1 + RRR2S/DK2 + RRR1S/DPM
640: C/# ENDIF
641: C/#ENDIF
642:
643: C/#IF RANDOM( BOUND )
644: *          R(K1+J) = MIN( RNEAR1, MAX(SMALL,R(K1+J)))
645: C/#ELSEIF .NOT.RANDOM( ALLOWZERO )
646: *          if ( R(K1+J).eq.0.0 ) R(K1+J) = 1.0E-30
647: C/#ENDIF
648:
649:      RNCTR = RNCTR + 1.0d0
650:

```

src/utils/ranu2.f

```

651:      130      continue
652:
653:      C/#IF LCGMOD( 31BIT )
654:      *          IRAND      = INT(DX*DPM)
655:      C/#ELSE
656:      C/# IF INTEGER8( RANU2 )
657:      *          SEED       = TSEED
658:      C/# ELSE
659:      cc          IRAND      = int(RRR3S*DK2,8) + int(RRR2S*DK1,8) + int(RRR1S,8)
660:      SU          = RRR3S
661:      SM          = RRR2S
662:      SL          = RRR1S
663:      C/# ENDIF
664:      C/#ENDIF
665:
666:      140 continue
667:
668:      c =<memo>= The seed value is saved as common variables
669:      c          for a restart calculation. This treatment will be
670:      c          modified in the future.
671:
672:      C/#IF LCGMOD( 31BIT )
673:      *          IRNSU = 0
674:      *          IRNSM = int(IRAND/2**21)
675:      *          IRNSL = IRAND - IRNSM*2**21
676:      C/#ELSE
677:      C/# IF INTEGER8( RANU2 )
678:      *          IRNSU = int(SEED/2_8**42)
679:      *          IRNSM = int((SEED - IRNSU*2_8**42)/2_8**21)
680:      *          IRNSL = SEED - IRNSU*2_8**42 - IRNSM*2_8**21
681:      C/# ELSE
682:      IRNSU = SU
683:      IRNSM = SM
684:      IRNSL = SL
685:      C/# ENDIF
686:      C/#ENDIF
687:
688:      c ... print the number of used RNS ...
689:      c      write(*,*) ' RANU2 : N=',N,' RNCTR=',RNCTR
690:
691:      return
692:      end
693: C
694: C=====
695: C .... initialization to use every 'NRSTEP' th random number ...
696: C=====
697: C
698:      subroutine RNINIT( NRSTEP )
699: C
700: C Initialize X0,X1 & Y parameter to generate <i>nrstep + k</i>'th
701: C random number is generated by RANU2 call. ( i = 0,1,2,3, ... )
702: C where IRAND = <k>'th random number.
703: C
704: c      implicit real*8(A-H,O-Z)
705: c      implicit none
706:
707:      include '../shared/INC/_RAND'
708: c
709: c ... dummy argument ...
710: c
711:      integer NRSTEP
712: c
713: c ... parameters ...
714: c
715:      integer NV

```

```

716:      parameter( NV      = 2048 )
717: c
718: C/#IF LCGMOD( 31BIT )
719: c      MVP/GMVP original RNG = Fujitsu SSL-II
720: c ... 31-bit LCG : multiplier = 32771, increment = 1234567891
721: c
722: *      integer MM, KK
723: *      parameter( MM      = 31, KK      = 16 )
724: *      real*8 DPM, DK, DMK
725: *      parameter( DPM      = (2.0D0 )**MM )
726: *      parameter( DK       = (2.0D0 )**KK )
727: *      parameter( DMK      = (2.0D0 )**(MM-KK) )
728: c
729: *      real*8 XMULTI, XADD
730: *      parameter( XMULTI   = 32771D0 )
731: *      parameter( XADD     = 1234567891D0 )
732: c
733: C/#ELSE
734: c
735: c ... 63-bit LCG : multiplier = 9219741426499971445, increment = 1
736: c
737: C/# IF INTEGER8( RANU2 )
738: *      integer*8 AAA, CCC, MASK
739: c ... 63bit LCG ...
740: *      parameter( AAA      = 9219741426499971445_8 )
741: *      parameter( CCC      = 1 )
742: c      2**63 - 1 = 9223372036854775807
743: *      parameter( MASK     = 9223372036854775807_8 )
744: c ... 16bit LCG ...
745: c      parameter( AAA      = 18389 )
746: c      parameter( CCC      = 1 )
747: c      2**16 - 1 = 65535
748: c      parameter( MASK     = 65535 )
749: c ... 8bit LCG ...
750: c      parameter( AAA      = 141 )
751: c      parameter( CCC      = 1 )
752: c      2**8 - 1 = 255
753: c      parameter( MASK     = 255 )
754: C/# ELSE
755:      real*8 DK1, DK2, DPM
756:      parameter( DK1      = (2.0d0)**21 )
757:      parameter( DK2      = (2.0d0)**42 )
758:      parameter( DPM      = (2.0d0)**63 )
759:
760:      real*8 AU, AM, AL, CU, CM, CL
761:      parameter( AU      = 2096326 )
762:      parameter( AM      = 1037627 )
763:      parameter( AL      = 1829237 )
764:      parameter( CU      = 0 )
765:      parameter( CM      = 0 )
766:      parameter( CL      = 1 )
767: C/# ENDIF
768: C/#ENDIF
769: c
770: c ... common variables & arrays ...
771: c
772: C/#IF LCGMOD( 31BIT )
773: *      real*8 X0(NV), X1(NV), Y(NV)
774: C/#ELSE
775: C/# IF INTEGER8( RANU2 )
776: *      integer*8 X(NV), Y(NV)
777: C/# ELSE
778:      real*8 XU(NV), XM(NV), XL(NV), YU(NV), YM(NV), YL(NV)
779: C/# ENDIF
780: C/#ENDIF

```

src/utils/ranu2.f

```

781:      real    SMALL, RNEAR1
782:      integer INIT
783:
784: C/#IF PARA( SX* )
785: *
786: C/# IF LCGMOD( 31BIT )
787: *   local common /CMRAND/ X0, X1, Y, SMALL, RNEAR1, INIT
788: C/# ELSE
789: C/#   IF INTEGER8( RANU2 )
790: *     local common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
791: C/# ELSE
792: *     local common /CMRAND/ XU, XM, XL, YU, YM, YL,
793: *     &              SMALL, RNEAR1, INIT
794: C/#   ENDIF
795: C/# ENDIF
796: *
797: C/#ELSEIF PARA( CRAY* )
798: *
799: C/# IF LCGMOD( 31BIT )
800: *   task common /CMRAND/ X0, X1, Y, SMALL, RNEAR1, INIT
801: C/# ELSE
802: C/#   IF INTEGER8( RANU2 )
803: *     task common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
804: C/# ELSE
805: *     task common /CMRAND/ XU, XM, XL, YU, YM, YL,
806: *     &              SMALL, RNEAR1, INIT
807: C/#   ENDIF
808: C/# ENDIF
809: *
810: C/#ELSE
811:
812: C/# IF LCGMOD( 31BIT )
813: *   common /CMRAND/ X0,      X1,      Y,      SMALL, RNEAR1, INIT
814: C/# ELSE
815: C/#   IF INTEGER8( RANU2 )
816: *     common /CMRAND/ X, Y, SMALL, RNEAR1, INIT
817: C/# ELSE
818: *     common /CMRAND/ XU, XM, XL, YU, YM, YL, SMALL, RNEAR1, INIT
819: C/#   ENDIF
820: C/# ENDIF
821:
822: C/#ENDIF
823: C
824: c
825: c ... local variables ...
826: c
827:      integer NS, I
828: C/#IF LCGMOD( 31BIT )
829: *   real*8 DX1, DX0, DY, DD, DI
830: C/#ELSE
831: C/# IF INTEGER8( RANU2 )
832: *   integer*8 DX, DY
833: C/# ELSE
834:      real*8 DXU, DXM, DXL, DYU, DYM, DYL
835:      real*8 VAL1X, QQQ1X, RRR1X, VAL2X, QQQ2X, RRR2X,
836:      &      VAL3X, QQQ3X, RRR3X
837:      real*8 VAL1Y, QQQ1Y, RRR1Y, VAL2Y, QQQ2Y, RRR2Y,
838:      &      VAL3Y, QQQ3Y, RRR3Y
839: C/#   ENDIF
840: C/#ENDIF
841:
842:      INIT = 0
843:      RNCTR = 0.d0
844:
845: C/#IF LCGMOD( 31BIT )

```

```

846: *   DX1 = INT(XMULTI/DK)
847: *   DX0 = XMULTI - DK*DX1
848: *   DY = XADD
849: C/#ELSE
850: C/# IF INTEGER8( RANU2 )
851: *   DX = AAA
852: *   DY = CCC
853: C/# ELSE
854:      DXU = AU
855:      DXM = AM
856:      DXL = AL
857:      DYU = CU
858:      DYM = CM
859:      DYL = CL
860: C/#   ENDIF
861: C/#ENDIF
862:
863: C
864:      NS = 0
865:      do 100 I = 1, NV*NRSTEP
866:
867:          if ( MOD(I,NRSTEP).eq.0 ) then
868:              NS = NS + 1
869:
870: C/#IF LCGMOD( 31BIT )
871: *   X0(NS) = DX0
872: *   X1(NS) = DX1
873: *   Y(NS) = DY
874: C/#ELSE
875: C/# IF INTEGER8( RANU2 )
876: *   X(NS) = DX
877: *   Y(NS) = DY
878: C/# ELSE
879:      XU(NS) = DXU
880:      XM(NS) = DXM
881:      XL(NS) = DXL
882:      YU(NS) = DYU
883:      YM(NS) = DYM
884:      YL(NS) = DYL
885: C/#   ENDIF
886: C/#ENDIF
887:          end if
888:
889: C/#IF LCGMOD( 31BIT )
890: *   DD = DX0*XMULTI
891: *   DI = INT(DD/DK)
892: *   DX0 = DD - DI*DK
893: *   DX1 = MOD(MOD(DX1*XMULTI,DMK)+DI,DMK)
894: *   DY = MOD(MOD(DY*XMULTI,DPM)+XADD,DPM)
895: C/#ELSE
896: C/# IF INTEGER8( RANU2 )
897: *   DX = iand( AAA*DX, MASK )
898: *   DY = iand( AAA*DY+CCC, MASK )
899: C/# ELSE
900:      VAL1X = AL*DXL
901:      QQQ1X = int(VAL1X/DK1)
902:      RRR1X = VAL1X - DK1*QQQ1X
903:
904:      VAL2X = AM*DXL + AL*DXM + QQQ1X
905:      QQQ2X = int(VAL2X/DK1)
906:      RRR2X = VAL2X - DK1*QQQ2X
907:
908:      VAL3X = AU*DXL + AM*DXM + AL*DXU + QQQ2X
909:      QQQ3X = int(VAL3X/DK1)
910:      RRR3X = VAL3X - DK1*QQQ3X

```

src/utls/ranu2.f

```
911:
912:      DXU   = RRR3X
913:      DXM   = RRR2X
914:      DXL   = RRR1X
915:
916:      VAL1Y = AL*DYL + CL
917:      QQQ1Y = int(VAL1Y/DK1)
918:      RRR1Y = VAL1Y - DK1*QQQ1Y
919:
920:      VAL2Y = AM*DYL + AL*DYM + CM + QQQ1Y
921:      QQQ2Y = int(VAL2Y/DK1)
922:      RRR2Y = VAL2Y - DK1*QQQ2Y
923:
924:      VAL3Y = AU*DYL + AM*DYM + AL*DYU + CU + QQQ2Y
925:      QQQ3Y = int(VAL3Y/DK1)
926:      RRR3Y = VAL3Y - DK1*QQQ3Y
927:
928:      DYU   = RRR3Y
929:      DYM   = RRR2Y
930:      DYL   = RRR1Y
931: C/#  ENDIF
932: C/#ENDIF
933:
934:      100 continue
935:      return
936: C/#ENDIF
937:      end
```

src/utils/rwind.f

```
1: C=====
2: C  Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
3: C-----
4: C    .... REWIND OPERATION IN SAFE :
5: C
6: C
7: C    Rewind operation for an un-opened file may cause error-stop
8: C    in some systems.
9: C=====
10:    subroutine RWIND( IUNIT )
11: C
12:    logical OPD
13:    inquire(unit =IUNIT,opened =OPD)
14:    if ( OPD ) rewind(IUNIT)
15:    return
16:    end
```


src/utils/tokei.f

```

1:      subroutine TOKEI( TMM, MODE )
2: C=====
3: C   Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
4: C-----
5: C   RETURN TIME   IN SECONDS.
6: C
7: C   << arguments >>
8: C
9: C       tmm : Wallclock time measured from an arbitrary time point that
10: C           depends on user's system, or measured from the last reset
11: C           action ( mode = 1 ).
12: C
13: C       mode : = 1   reset clock ( set the base of time to the calling
14: C                   time ). returns tmm = 0.0.
15: C               other than 1   --- no reset action.
16: C
17: C   << comment >>
18: C
19: C   This routine aims to get clock value in sub-second precision, but
20: C   user's system might not allow to get clock value in such a precision.
21: C
22: C=====
23: C/#IF CUTIL.AND.MS_VISUAL
24: C
25: C ... This interface block is for CUTIL & MS-Visual tools. ...
26: C
27: C   interface
28: C       subroutine WALLCLOCK( TTT8 )
29: C           real*8 TTT8
30: C       *CDEC$ ATTRIBUTES C :: WALLCLOCK
31: C       *CDEC$ ATTRIBUTES REFERENCE :: TTT8
32: C       end subroutine
33: C   end interface
34: C/#ENDIF
35: C
36: C       real TMM
37: C
38: C       real*8 TTT8
39: C
40: C   ... current basis of time measurement
41: C
42: C       real*8 TTT0
43: C
44: C
45: C       integer IARRAY(10)
46: C
47: C       data TTT0 /0.0D0/
48: C
49: C/#IF F90(SYSTEM_CLOCK)
50: C
51: C       call SYSTEM_CLOCK( ICOUNT, IRATE, ICNTMX )
52: C       if ( IRATE.ne.0 ) TTT8 = dble(ICOUNT)/dble(IRATE)
53: C
54: C/#ELSEIF CUTIL
55: C
56: C   ... for ordinary UNIX systems.
57: C       call C coded subroutine WALLCLOCK.
58: C
59: C       call WALLCLOCK( TTT8 )
60: C
61: C/#ELSEIF SYSTEM(HP* SUN* DECOSF* NECEWS* IBMRS* AIX* HIOSF* SGI*)
62: C
63: C       call WALLCLOCK( TTT8 )
64: C
65: C/#ELSEIF SYSTEM(PARAGON*)

```

```

66:
67: C       call WALLCLOCK( TTT8 )
68: C
69: C/#ELSEIF SYSTEM(MIPS*)
70: C
71: C       call ITIME(IARRAY)
72: C       TTT8 = IARRAY(3) + 60* (IARRAY(2) + 60* IARRAY(1) )
73: C
74: C/#ELSEIF SYSTEM(CRAY*)
75: C
76: C----- CRAY -----
77: C
78: C       TTT8 = RTC()
79: C       TTT8 = TIMEF()/1000.0
80: C
81: C/#ELSEIF SYSTEM(SX*)
82: C
83: C----- NEC-SX -----
84: C
85: C       call ETIME(TTT8)
86: C
87: C/#ELSEIF SYSTEM(FACOM*)
88: C
89: C----- FACOM -----
90: C
91: C       call TIME(IITIME)
92: C       TTT8 = dble(IITIME)/1000
93: C
94: C/#ELSEIF SYSTEM(IBMVS)
95: C
96: C----- IBM VS-FORTRAN ---
97: C
98: C       integer IT(8)
99: C       CALL DATIM(IT)
100: C       TTT8 = IT(2)/1000D0+IT(3)+60D0*(IT(4)+60D0*(IT(5)+24D0*IT(6)))
101: C
102: C/#ELSE
103: C
104: C       TTT8 = 0.0D0
105: C
106: C/#ENDIF
107: C
108: C       if( MODE.eq.1 ) TTT0 = TTT8
109: C       TMM = TTT8 - TTT0
110: C
111: C       return
112: C       end

```

src/utils/wallcloc.c

```

1: #ifndef NO_WALLCLOCK_C
2:
3: /*****
4:  * Fortran called C subroutine :
5:  *
6:  * UNIX venders have their own convention or rule for external
7:  * names of FORTRAN coded subprograms or commons.
8:  *
9:  * We prepare equivalent sources for C coded routines
10:  *
11:  * 1. Attach underscore to FORTRAN name. ( SUN & Other BSD UNIX)
12:  * 2. Same as FORTRAN name (lowercase). (HP-UX, IBM-AIX etc.)
13:  * 3. Uppercase string of FORTRAN name. ( Ncube etc. )
14:  * 4. "_" + uppercase string of FORTRAN name. ( HI/OSF )
15:  *
16:  *****/
17:
18: /*****
19:  *
20:  * Wall clock time since midnight January 1, 1970. ;
21:  *
22:  * This clock has a nominal precision of microsecond but
23:  * the actual precision depends on settings of users system.
24:  *
25:  * Using BSD origined system call "gettimeofday" which is probably
26:  * supported on most UN*X systems.
27:  * For non UN*X systems, use ANSI-C function "times".
28:  *
29:  * Programmed by M.Sasaki (the Japan Research Institute Co. Ltd.)
30:  *
31:  * 1st version: 6 Oct 1993
32:  * 19 Jul 1996 : use time() for non-POSIX and ANSI-C compilers.
33:  * May 1997: gettimeofday() is not in POSIX. It is BSD system
34:  * call...
35:  *
36:  * * works on HP-UX, SunOS, DEC/OSF, Sysytem_V etc.
37:  *
38:  *****/
39:
40: #include <stdio.h>
41:
42: #ifdef ANSI_C
43: /*** Use time function for ANSI-C (non BSD) mode ***/
44: #include <time.h>
45: time_t ttt0 ;
46:
47: double wall_clock()
48: {
49:     return (double)time( &ttt0 ) ;
50: }
51:
52: #else /*** UN*X: use gettimeofday ***/
53:
54: #include <sys/types.h>
55: #include <sys/time.h>
56:
57: double wall_clock()
58: {
59:     struct timeval tv;
60:     struct timezone tz;
61: #ifdef SX /*** Super UX lacks tz argument of gettimeofday ***/
62:     gettimeofday( &tv ) ;
63: #else
64:     gettimeofday( &tv, &tz ) ;
65: #endif
66:     return (double)tv.tv_sec + (double)(tv.tv_usec)/1000000 ;
67: }
68: #endif
69:
70: void wallclock( ttt )
71:     double *ttt ;
72: {
73:     *ttt = wall_clock();
74: }
75:
76: void wallclock_( ttt )
77:     double *ttt ;
78: {
79:     *ttt = wall_clock();
80: }
81:
82: void WALLCLOCK( ttt )
83:     double *ttt ;
84: {
85:     *ttt = wall_clock();
86: }
87:
88: void _WALLCLOCK( ttt )
89:     double *ttt ;
90: {
91:     *ttt = wall_clock();
92: }
93:
94: main()
95: {
96:     double t0, t1 ;
97:     tokei( &t0 ) ;
98:
99:     while( getchar() != EOF )
100:     {
101:         tokei( &t1 ) ;
102:         printf( "%lf \n", t1-t0 ) ;
103:     }
104: }

```

src/tools/bio2.f

```

1: C
2: C BIORHYTHM DAYO
3: C
4:     parameter( IMAX = 20 )
5:     parameter( KKY  = 1957, KKM = 8, KKD  = 20, KKW = 2 )
6: C
7:     character*(2*IMAX+1) LINE, WAKU
8:     character*5 YOBI(7)
9:     dimension MONTH(12)
10: C
11:     logical LPY
12:     logical LEAPY
13: C
14:     data MONTH /31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
15:     data YOBI /'(SUN)', ' MON ', ' TUE ', ' WED ', ' THR ', ' FRI '
16:     & ' SAT ' /
17: C
18: -----
19: C
20:     LEAPY(IY) = MOD(IY,4) .eq.0
21:     & .and.(MOD(IY,400).eq.0.or.MOD(IY,100).ne.0)
22: C
23: -----
24: C
25:     WAKU = ' '
26:     IM = IMAX/5
27:     do 100 I = 0, 10, 5
28:         WAKU(I*IM+1:I*IM+1) = ' '
29:     100 continue
30: C
31:     110 write(6,*) ' TANJOBI ( Y M D ) ==> '
32:     read(5,*) IBY, IBM, IBD
33:     if ( IBY.eq.0 ) go to 130
34:     write(6,*) ' OMEATE ( Y M D ) ==> '
35:     read(5,*) JBY, JBM, JBD
36:     JDAY = IDAY(IBY,IBM,IBD,JBY,JBM,JBD,MONTH)
37:     KDAY = IDAY(KKY,KKM,KKD,JBY,JBM,JBD,MONTH)
38: CCCCC WRITE(6,*) 'KDAY =',KDAY
39:     if ( KDAY.ge.0 ) then
40:         IBW = MOD(KDAY,7) + KKW
41:     else
42:         IBW = -MOD(-KDAY,7) + KKW
43:         if ( IBW.lt.0 ) IBW = IBW + 7
44:     end if
45:     KY = JBY
46:     KM = JBM
47:     KD = JBD
48:     do 120 I = 1, 30
49:         P = SIN((FLOAT(MOD(JDAY,23))-0.5)/23.0*2.0*3.141592653)
50:         S = SIN((FLOAT(MOD(JDAY,28))-0.5)/28.0*2.0*3.141592653)
51:         XI = SIN((FLOAT(MOD(JDAY,33))-0.5)/33.0*2.0*3.141592653)
52:         LINE = WAKU
53:         IP = IMAX + 1 + IFIX(P*IMAX)
54:         IS = IMAX + 1 + IFIX(S*IMAX)
55:         IXI = IMAX + 1 + IFIX(XI*IMAX)
56:         LINE(IP:IP) = 'P'
57:         LINE(IS:IS) = 'S'
58:         LINE(IXI:IXI) = 'I'
59:         write(6,'(1X,I2,1X,I2,1X,A5,1X,A)') KM, KD, YOBI(IBW+1), LINE
60:         JDAY = JDAY + 1
61:         IBW = MOD(IBW+1,7)
62:         KD = KD + 1
63:         LPY = LEAPY(KY)
64:         if ( KM.ne.2.and.KD.gt.MONTH(KM) .or. KM.eq.2.and..not.LPY
65:         & .and.KD.gt.28 .or. KM.eq.2.and.LPY.and.KD.gt.29 ) then

```

```

66:         KD = 1
67:         KM = KM + 1
68:         if ( KM.gt.12 ) then
69:             KM = 1
70:             KY = KY + 1
71:         end if
72:     end if
73:     120 continue
74: C
75:     go to 110
76:     130 stop
77:     end
78: C
79:     function IDAY(IY,IM,ID,JY,JM,JD,MONTH)
80:     dimension MONTH(12)
81:     logical LEAPY
82:     LEAPY(IY) = MOD(IY,4) .eq.0
83:     & .and.(MOD(IY,400).eq.0.or.MOD(IY,100).ne.0)
84:     IBY = IY
85:     IBM = IM
86:     IBD = ID
87:     JBY = JY
88:     JBM = JM
89:     JBD = JD
90:     IS = 1
91:     if ( IBY.gt.JBY .or. IBY.eq.JBY.and.IBM.gt.JBM .or. IBY.eq.JBY
92:     & .and.IBM.eq.JBM.and.IBD.gt.JBD ) then
93:         IBY = JY
94:         IBM = JM
95:         IBD = JD
96:         JBY = IY
97:         JBM = IM
98:         JBD = ID
99:         IS = -1
100:     end if
101:     IDAY = MONTH(IBM) - IBD
102: C
103:     if ( IBM.eq.2.and.LEAPY(IBY) ) IDAY = IDAY + 1
104:     if ( IBM.eq.JBM.and.IBY.eq.JBY ) go to 110
105: C
106:     100 IBM = IBM + 1
107:     if ( IBM.gt.12 ) then
108:         IBY = IBY + 1
109:         IBM = 1
110:     end if
111:     IDAY = IDAY + MONTH(IBM)
112:     if ( IBY.ge.JBY.and.IBM.ge.JBM ) go to 110
113:     if ( IBM.eq.2.and.LEAPY(IBY) ) IDAY = IDAY + 1
114:     go to 100
115: C
116:     110 IDAY = IS*(IDAY-(MONTH(IBM)-JBD))
117:     return
118:     end

```

src/tools/cat80.f

```
1: C
2: C
3: C   Padding textfile with blank to make all lines to 80 columns
4: C
5: C
6:       character*80 line
7:       line = ' '
8:   1000 read(5,'(a)',end=999) line
9:       write(6,'(a80)') line
10:      goto 1000
11: c
12:   999 stop
13:      end
```

DATA

src/tools/chuck.c

```

1: /*
2:  * Tool for MVP/GMVP system
3:  *
4:  * Check parenthesis ( ) & bracket <> in MVP/GMVP input data.
5:  *
6:  * June 1995, by Bunei Furutaku (furutaku@tyo.sci.jri.co.jp)
7:  * July 1995, modified by M.Sasaki (sasaki@tyo.sci.jri.co.jp)
8:  *
9:  */
10:
11: /*****
12:  Common headers
13:  *****/
14: #include <stdio.h>
15:
16: #ifdef BSD_STRINGS_H /** BSD type strings.h for index(),rindex() */
17: #include <strings.h>
18: #define HAVE_RINDEX
19: #else /** for ANSI type string functions */
20: #include <string.h>
21: #endif
22:
23: /*****
24:  Global data
25:  *****/
26:
27: /** input line buffer */
28: #define MAX_COLUMN 128
29: #define MAX_BUFFERED_LINE 1000
30: static char line_buf[MAX_BUFFERED_LINE][MAX_COLUMN] ;
31:
32: static int nline = 0 ; /** current line # */
33: static int bline = 0 ; /** position of currnet line in line_buf */
34: static int ncolumn = 0 ; /** current column number */
35:
36:
37: /*****
38:  function : getc_line
39:  Return a character from line input buffer as "getc" function.
40:  *****/
41: char getc_line( fp, iline, icolumn )
42: FILE *fp ;
43: int *iline;
44: int *icolumn;
45: {
46:     if( ncolumn == 0 || line_buf[bline][ncolumn] == '\0' )
47:     {
48:
49:         if( bline == MAX_BUFFERED_LINE-1 )
50:         {
51:             bline = 0 ;
52:         }
53:         else
54:         {
55:             bline++;
56:         }
57:
58:         if( fgets( line_buf[bline],MAX_COLUMN,fp) == NULL )
59:         {
60:             bline--;
61:             return EOF ;
62:         }
63:         nline++;
64:         ncolumn = 0 ;
65:     }
66:
67:     ncolumn++ ;
68:
69:     *iline = nline ;
70:     *icolumn = ncolumn ;
71:     return line_buf[bline][ncolumn-1] ;
72: }
73: /*****
74:  function : non_blank( )
75:  return column the position of the first non-blank character
76:  in the current line. Return a value greater than strlen() for
77:  blank lines.
78:  *****/
79: int non_blank( )
80: {
81:     int i ;
82:     char * cp = &( line_buf[bline][0] ) ;
83:     for( i = 0 ; cp[i] == ' ' ; i++ ) ;
84:     return ++i ;
85: }
86:
87: /*****
88:  function : print_marked_line
89:  Print the line input buffer with mark at a specific column position.
90:  *****/
91:
92: int print_marked_line( iline, column )
93: int iline;
94: int column;
95: {
96:     int i, j;
97:
98:     /* check whether specified line exists in line_buf */
99:     if ( nline - iline >= MAX_BUFFERED_LINE )
100:     {
101:         return 1 ;
102:     }
103:
104:     j = bline - (nline - iline) ;
105:     if( j < 0 ) j = MAX_BUFFERED_LINE + j ;
106:
107:     fputs("          ",stderr);
108:     if( column > 0 ) {
109:         for( i=1 ; i < column ; i++ ) fputc( '-',stderr);
110:         fputc('V',stderr);
111:         fputc('\n',stderr);
112:     }
113:
114:     fprintf(stderr,"%5d: %s\n",iline,line_buf[j]);
115: }
116:
117: /*****
118:  Main function
119:  *****/
120:
121: main(argc,argv)
122: int argc;
123: char *argv[];
124: {
125:     char a,b,dl;
126:     static char icross[200];
127:     int i,icolumn,iline,ierror;
128:     int istk,icount,ibcount;
129:     static int iltmp[100],irtmp[100];
130:     static int ibltmp[100],ibrtmp[100];

```

src/tools/chuck.c

```
131: FILE *fp;
132:
133: if(argc<2)
134: {
135:     fprintf(stderr,"usage : %s file\n",argv[0]);
136:     exit();
137: }
138: if((fp=fopen(argv[1],"r"))==NULL)
139: {
140:     fprintf(stderr,"Can't open file\n");
141:     exit();
142: }
143:
144: icount=0;
145: ibcount=0;
146: icolumn=0;
147: iline=1;
148: istk=0;
149: ierror=0;
150:
151: while(1)
152: {
153:     a=getc_line(fp, &iline, &icolumn);
154:
155:     if(a==EOF) break;
156:
157:     /* checking whether there is a word after column 72 of each line */
158:     if(icolumn>72)
159:     {
160:         int non_blank = 0 ;
161:         while(a!='\n') {
162:             if( a != ' ' ) non_blank=1 ;
163:             a=getc_line(fp,&iline,&icolumn);
164:         }
165:         if( non_blank ) {
166:             fprintf(stderr,"line %d : Something after 73th column\n",iline
167: );
168:             ierror++;
169:             print_marked_line(iline, 73);
170:         }
171:     }
172:
173:     /* checking whether there is '#' or '$' in the middle of line. */
174:     if(icolumn>1 && (a=='#' || a=='$' || a=='*' || a=='&' || a=='%'))
175:     {
176:         if( non_blank() == icolumn )
177:         {
178:             fprintf(stderr,
179: "line %d, column %d : '%c' is the first non-blank character
180: and not on the 1st column.\n",
181: iline,icolumn,a);
182:             ierror++;
183:             print_marked_line(iline,icolumn);
184:         }
185:
186:         /* neglecting comment line which starts from '/*'. */
187:         if(icolumn==1 && a=='*')
188:         {
189:             while(a!='\n') a=getc_line(fp,&iline,&icolumn);
190:         }
191:
192:         /* neglecting rest of column after '/*'. */
193:         if(icolumn>1 && a=='/')
194:
195:         {
196:             a=getc_line(fp,&iline,&icolumn);
197:             if(a=='*') while(a!='\n') a=getc_line(fp,&iline,&icolumn);
198:         }
199:
200:         /* Main part of this program */
201:         switch(a)
202:         {
203:             case '(':
204:                 icount++;
205:                 istk++;
206:                 icross[istk]='(';
207:                 iltmp[icount-1]=iline;
208:                 irtmp[icount-1]=icolumn;
209:                 break;
210:             case ')':
211:                 icount--;
212:                 if(istk==0) break;
213:                 if(icross[istk]!='(')
214:                 {
215:                     ierror++;
216:                     fprintf(stderr,
217: "line %d, column %d '<' and line %d, column %d ')' --- Cros
218: sing\n",
219: ibltmp[ibcount-1],ibrtmp[ibcount-1],iline,icolumn);
220:
221:                     print_marked_line(ibltmp[ibcount-1],ibrtmp[ibcount-1]);
222:                     print_marked_line(iline,icolumn);
223:                 }
224:                 else
225:                     istk--;
226:                 break;
227:             case '<':
228:                 ibcount++;
229:                 istk++;
230:                 icross[istk]='<';
231:                 ibltmp[ibcount-1]=iline;
232:                 ibrtmp[ibcount-1]=icolumn;
233:                 break;
234:             case '>':
235:                 ibcount--;
236:                 if(istk==0) break;
237:                 if(icross[istk]!='<')
238:                 {
239:                     fprintf(stderr,
240: "line %d, column %d '<' and line %d, column %d '>' --- Cros
241: sing\n",
242: iltmp[icount-1],irtmp[icount-1],iline,icolumn);
243:                     ierror++;
244:                     print_marked_line(iltmp[icount-1],irtmp[icount-1]);
245:                     print_marked_line(iline,icolumn);
246:                 }
247:                 else
248:                     istk--;
249:                 break;
250:             case '\n':
251:                 iline++;
252:                 icolumn=0;
253:                 break;
254:             default:
255:                 break;
256:         }
257:         if(icount<0)
258:         {
259:             fprintf(stderr,"line %d, column %d, no corresponding '('\n",
```

src/tools/chuck.c

```
257:         iline, icolumn);
258:         icount=0;
259:         ierror++;
260:         print_marked_line(iline, icolumn);
261:     }
262:     if(ibcount<0)
263:     {
264:         fprintf(stderr, "line %d, column %d, no corresponding '<'\n",
265:             iline, icolumn);
266:         ibcount=0;
267:         ierror++;
268:         print_marked_line(iline, icolumn);
269:     }
270:
271:     b=a;
272: }
273:
274: for(i=0; i<icount; i++)
275: {
276:     fprintf(stderr, "line %d, column %d, no corresponding '*''\n",
277:         iltmp[i], irtmp[i]);
278:     ierror++;
279:     print_marked_line(iltmp[i], irtmp[i]);
280: }
281: for(i=0; i<ibcount; i++)
282: {
283:     fprintf(stderr, "line %d, column %d, no corresponding '>'\n",
284:         ibltmp[i], ibrtmp[i]);
285:     ierror++;
286:     print_marked_line(ibltmp[i], ibrtmp[i]);
287: }
288: fprintf(stderr, "\n%d errors detected\n\n", ierror);
289: fclose(fp);
290: }
```

src/tools/dtk.f

```

1: C
2: C   DENTAK : portable calculator simulator
3: C
4: C   ORIGINALLY PROGRAMMED ON FACOM M SERIES BY M.SASAKI
5: C
6: C
7: C
8: C ... main program of DENTAK : ...
9: c
10: C   (compile with dentak.f of MVP/GMVP source)
11: c
12: c
13: C
14:   parameter( MAXP = 20 )
15:   implicit real*8(D)
16:   character LINE*72, FF*25
17:   real*8 DENTAK
18: C
19:   write(6,*) '-----'
20:   write(6,*) '  Portable calculator  ( MVP/GMVP tool )-----'
21:   write(6,*) '-----'
22:   write(6,*) '  PROGRAMMED BY M.SASAKI ( JUL 1993 ) -----'
23:   write(6,*) '-----'
24:   write(6,*) '-----'
25:   write(6,*) '-- predefined variables (constant) -----'
26:   write(6,*) '-----'
27:   write(6,*) '-- %PAI = 3.141592653589793238D+00 -----'
28:   write(6,*) '-- %E = 2.718281828459045235D+00 -----'
29:   write(6,*) '-- %G = 0.5772156649015328606D+00 (Euler const.)--'
30:   write(6,*) '-----'
31:   write(6,*) '-----'
32:   write(6,*) ' == DO YOU NEED A LOG-LIST ? (1/0 = YES /NO)'
33:   read(5,*) IPR
34: C
35:   if ( IPR.eq.1 ) then
36:     write(*,*) ' == INPUT LOG-LIST FILE NAME =='
37:     read(5, '(A)') FF
38:     write(6,*) ' === LIST : ', FF
39:     open( 8, file =FF, status = 'UNKNOWN' )
40:   end if
41: C
42:   write(*,*) ' Type "END" or "QUIT" to exit.'
43: C
44: 100 read(5,fmt = '(A)',end =110) LINE
45:   if ( IPR.eq.1 ) write(8, '(1x,A)') LINE
46:   if ( LINE(1:1).eq.'*' .or. LINE.eq.' ' ) go to 100
47: C
48: C ..... FILTER : SMALL CHARACTER TO CAPITAL ....
49: C
50:   call CAPITAL( LINE, 72 )
51: C
52:   if ( LINE(1:3).ne.'END' .and. LINE(1:4).ne.'QUIT' ) then
53: C
54:     DD      = DENTAK(LINE,6,IERR)
55:     write(6, '(1x, '' ANSWER = '',1PE13.5,2X,D25.18)') DD, DD
56:     if ( IPR.eq.1 )
57:       &      write(8, '(1x, '' ANSWER = '',1PE13.5,2X,D25.18)') DD, DD
58:     go to 100
59: C
60:   end if
61: C
62: 110 if ( IPR.eq.1 ) then
63:   close( 8 )
64:   write(6,*) ' === LIST : ', FF
65:   end if

```

```

66: C
67:   write(6,*) 'END OF CALCULATION'
68:   stop
69:   end
70: C -----
71: C
72: C ===== SMALL TO CAPITAL =====
73: C
74:   subroutine CAPITAL( STR,   LEN )
75: C
76:   character*(*) STR
77:   character*26 SMALL, CAPITL
78:   data SMALL /'abcdefghijklmnopqrstuvwxyz'/
79:   data CAPITL /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
80: C
81:   do 100 I = 1, LEN
82:     K      = INDEX(SMALL,STR(I:I))
83:     if ( K.ne.0 ) STR(I:I) = CAPITL(K:K)
84: 100 continue
85: C
86:   return
87:   end
88: C -----
89:   subroutine GTVVAL( VARNM, VAL,   IER )
90:   character*(*) VARNM
91:   real*8 VAL
92:   IER      = 0
93:   if ( VARNM.eq.'PAI' ) then
94:     VAL     = 3.141592653589793238D+00
95:   else if ( VARNM.eq.'E' ) then
96:     VAL     = 2.718281828459045235D+00
97:   else if ( VARNM.eq.'G' ) then
98:     VAL     = 0.5772156649015328606D+00
99:   else
100:     IER     = 1
101:     VAL     = 0
102:   end if
103:   return
104:   end

```


src/tools/fat.f

```
1: C
2: C  FAT :  Fortran source Assimilation Tool :
3: C
4: C
5: C      Author: M.Sasaki   (the Japan Research Institute Co. Ltd.)
6: C      First version :  Feb 1993
7: C
8: C=====
9: C  Assimilate a fortran source into user's machine, compiler, OS etc.
10: C=====
11: C
12: C -----
13: C 1. Requirements for fortran sources assimilated by this program.
14: C -----
15: C
16: C * Dependences on system, compilers, etc. are described as follows:
17: C
18: C      C/#IF  expression
19: C           source lines
20: C      C/#ELSEIF expression
21: C           source lines
22: C      C/#ELSE
23: C           source lines
24: C      C/#ENDIF
25: C
26: C      ( 'C/#' on column 1 to 3. ELSEIF & ELSE blocks are optional.
27: C        can be put blanks after #. )
28: C
29: C 'expression' is a logical expression:
30: C
31: C      expression := logical-operator expression
32: C                  := expression logical-operator expression
33: C                  := ( expression )
34: C                  := parameter-name
35: C                  := parameter-name( specifications )
36: C
37: C      logical-operator := .NOT.      (unary)
38: C                       := .AND.
39: C                       := .OR.
40: C
41: C      parameter-name : a character string set by 'SET' command or
42: C                       not set (described below).
43: C
44: C      specification : specification characters string given to a
45: C                       parameter name in 'SET' command (described below).
46: C
47: C
48: C      (Order of intensity is .NOT. > .AND. > .OR. .)
49: C
50: C * Expressions are continued to the next C/# line by '&'.
51: C   (yet not in use)
52: C
53: C * Expressions are finally translated 'ON' or 'OFF' status
54: C   as follows;
55: C
56: C 1. expression := parameter-name, 'ON' if parameter-name is set
57: C    by SET command (described below) else 'OFF'.
58: C
59: C 2. expression := parameter-name( specifications ), 'ON' if
60: C    the parameter-name is 'ON' and is given
61: C    specific-characters which match at least one
62: C    specification-characters in '(specifications)'.
63: C
64: C Parameters and specifications in expression can include wildcard
65: C characters '*' and '?' any times. An '*' can matches any strings
```

```
66: C      including null string, and a '?' matches any one character.
67: C
68: C * IF ... ENDIF(ELSE) or ELSEIF ... ENDIF (ELSE) blocks can be
69: C   nested. ( upto 'MAXNST' levels )
70: C
71: C
72: C
73: C -----
74: C 2. How to set parameters, file names etc. with ASSIM commands.
75: C -----
76: C
77: C ASSIM can interpret the following commands;
78: C
79: C      SET, UNSET, INPUT, OUTPUT, SOURCE, DIRECTIVE, MONITOR
80: C
81: C These commands are put on standard input (or between source lines
82: C for SET,UNSET and DIRECTIVE).
83: C
84: C * Parameter names or specifications are given in source files or
85: C   given from standard input by the following commands..
86: C
87: C      [C/#]SET  parameter-name[(specification-strings)]
88: C      [C/#]UNSET parameter-name[(specification-strings)]
89: C
90: C * Filenames of input source program or processed output source file
91: C   can be given with the following commands;
92: C
93: C      INPUT  input-file <standard input only>
94: C      OUTPUT output-file <standard input only>
95: C
96: C * Program source can be given in standard input stream after
97: C   'SOURCE' command.
98: C
99: C      SOURCE <standard input only>
100: C      .... fortran source lines to the end of the input stream ...
101: C
102: C * Specify "directives" on line-head that should be remain unchanged
103: C   in activation of 'ON' status lines. Effective only directives
104: C   having '*' charcter on the first column.
105: C
106: C      [C/#]DIRECTIVE directive
107: C
108: C << Rules for ASSIM commands >>
109: C
110: C * 'C/#' on top of a line is necessary in fortran source.
111: C * '(specification-characters)' are optional.
112: C * 'SOURCE' can be used only in standard input and means that
113: C   inputs after this command are source lines and source input
114: C   other than standard input is ignored.
115: C * 'SET' command sets status of parameter-name 'ON' and gives
116: C   specification characters to the parameter-name if specified.
117: C   'SET' can be used more than once for the same parameter-name.
118: C * 'UNSET' command sets status of parameter-name 'OFF' if no
119: C   specifications are given or parameter-name is not set
120: C   previously.
121: C   When specifications are given in the 'UNSET' command,
122: C   the specifications are removed from the parameter-name if the
123: C   specifications are given previously with 'ON' status,
124: C   otherwise no change on parameter status & specification.
125: C
126: C Parameter names or specication strings are non-blank character
127: C strings whose lengthes are less than 'MAXLEN' (parameter defined
128: C in this program).
129: C
130: C * select I/O unit of monitoring output by MONITOR command;
```

src/tools/fat.f

```
131: C
132: C      MONITOR   I/O-unit
133: C
134: C      Default monitoring output is I/O unit 0 (standard error for most
135: C      UNIX systems but In HP-UX, standard output is unit 7.)
136: C      Cannot specify unit 5,6,10 or 20 for monitoring.
137: C
138: C      In source lines 'C/# SET', 'C/# UNSET' and 'C/# DIRECTIVE' can be
139: C      under control of 'C/#IF', 'C/#ELSEIF', 'C/#ELSE' and 'C/#ENDIF'
140: C
141: C -----
142: C 3. How does this program work.
143: C -----
144: C
145: C 1. Interpret 'SET' or 'UNSET' command from standard input or in
146: C    a fortran source (on unit 10 or specified by 'INPUT' command).
147: C
148: C 2. When 'IF', 'ELSEIF' command line is encountered, evaluate
149: C    expression and activate the following block if result is 'ON'
150: C    or deactivate if result is 'OFF'.
151: C
152: C    <Activation>      Replace '*' character on the first column
153: C                      with a blank. If a character other than a numeric character or a
154: C                      blank is on the second column and the first column is '*',
155: C                      remove the first column and shift the line to the left by one
156: C                      column ( this is for the case that the source block includes
157: C                      compiler directives headed by '$' '@' '*' etc.)
158: C
159: C    <Deactivation>   Replace a blank character on the first column of
160: C                      lines in the source file with '*'. When the first column is
161: C                      neither 'c', 'C' nor a blank, shift the line to the right by one
162: C                      column and '*' is placed on the first column (to save compiler
163: C                      directives).
164: C
165: C -----
166: C 4. Input & Output
167: C -----
168: C
169: C *Standard input :
170: C
171: C      * 'SET', 'UNSET' commands.
172: C      * 'INPUT', 'OUTPUT' commands.
173: C      * 'DIRECTIVE' command.
174: C      * Fortran source after SOURCE command line.
175: C
176: C *Source input :
177: C
178: C      I/O unit 10. ( or specified by 'INPUT' command )
179: C
180: C      ASSIM command with 'C/#' on line-head.
181: C
182: C      * 'IF', 'ELSE', 'ELSEIF', 'ENDIF'
183: C      * 'SET', 'UNSET' commands.
184: C      * 'DIRECTIVE' command.
185: C
186: C *Source output :
187: C
188: C      I/O unit 20. ( or specified by 'OUTPUT' command )
189: C
190: C *Standard output :
191: C
192: C      The total number of activated or deactivated lines is output.
193: C      Users can use this value to overwrite original source file by
194: C      the processed source files.
195: C
```

```
196: C      Ex.1 (HP-UX with C-shell, command input is HP.set)
197: C
198: C      echo 'SET SYSTEM(HP9000)' > HP.set
199: C      echo 'SET ROUNDOFF(NEAREST)' >> HP.set
200: C      echo 'UNSET ARGSAVE' >> HP.set
201: C
202: C      setenv ftn10 $1          # input source file
203: C      setenv ftn20 $$temp.f    # temporary output source file
204: C      set F = 'assim < HP.set' # number of processed lines
205: C      if( $F > 0 ) then
206: C          cp $1 $1$.OLD        # save the original file for safety
207: C          mv $$temp.f $1
208: C          echo processed $F lines for $1
209: C      else
210: C          echo $1 is not changed '
211: C      endif
212: C
213: C      Ex.2 (SUN with C-shell, command input is SUN.set)
214: C
215: C      echo 'SET SYSTEM(SUN)' > SUN.set
216: C      echo 'SET ROUNDOFF(NEAREST)' >> SUN.set
217: C      echo INPUT $1 >> SUN.set
218: C      echo OUTPUT $$temp.f >> SUN.set
219: C
220: C      set F = 'assim < SUN.set' # number of processed lines
221: C      if( $F > 0 ) then
222: C          cp $1 $1$.OLD        # save file for safety
223: C          mv $$temp.f $1
224: C          echo processed $F lines for $1
225: C      else
226: C          echo $1 is not changed '
227: C      endif
228: C
229: C -----
230: C
231: C -----
232: C
233: C -----
234: C
235: C      program FAT
236: C
237: C      parameter( NCLINE = 72 )
238: C      character*(NCLINE) LINE, LINEU
239: C      character*(NCLINE) FILE
240: C
241: C      parameter( MXNEST = 64 )
242: C      integer IFEND(MXNEST)
243: C      logical LOGIF(MXNEST), LCUR
244: C
245: C      logical LDENTK, LEV
246: C
247: C      parameter( MAXDRV = 16 )
248: C      character*8 DIRCTV(MAXDRV)
249: C
250: C      character*8 FORMT
251: C
252: C      ---- nchg : number of lines changed .
253: C
254: C      NCHG = 0
255: C
256: C      ---- number of directives --
257: C
258: C      NDRV = 0
259: C
260: C
```

src/tools/fat.f

```

261: C
262: C ---- I/O unit of monitoring output ---
263: C
264: C     IPR = 7
265: C     IPR      = 0
266: C
267: C
268: C     Get SETting information ( & SOURCE if any ) from std. input.
269: C
270: C     ISTD      = 5
271: C     IIN       = ISTD
272: C     JIN       = 10
273: C     JOUT      = 20
274: C
275: C
276: C
277: C 100 call INLINE( IIN, LINE, LENG, IERR, IEND )
278: C
279: C
280: C     if ( IEND.ne.0 ) go to 140
281: C     if ( IERR.ne.0 ) then
282: C         write(6,*) - 999
283: C         stop 666
284: C     end if
285: C
286: C     if ( LINE(:LENG).eq.' ' ) go to 100
287: C
288: C ... lower case -> uppercase .
289: C
290: C     LINEU = LINE
291: C     call UPPER( LINE(:LENG) )
292: C
293: C     if ( LINE(1:3).eq.'SET' ) then
294: C
295: C         call SETPRM( LINE(4:LENG), IERR2 )
296: C
297: C     else if ( LINE(1:5).eq.'UNSET' ) then
298: C
299: C         call UNSETP( LINE(6:LENG), IERR2 )
300: C
301: C     else if ( LINE(1:6).eq.'SOURCE' ) then
302: C
303: C         JIN      = ISTD
304: C         go to 140
305: C
306: C     else if ( LINE(1:5).eq.'INPUT' ) then
307: C
308: C         call CCOMP( FILE, LEN(FILE), LINEU(6:), INN, IFFF )
309: C         open( JIN, form='FORMATTED', file=FILE(:INN), status='OLD',
310: C &             iostat=IOS )
311: C         if ( IOS.ne.0 ) then
312: C             write(IPR,*) 'SOURCE INPUT FILE OPEN ERROR: IOS =', IOS
313: C             write(6,*) -999
314: C             stop 666
315: C         end if
316: C
317: C     else if ( LINE(1:6).eq.'OUTPUT' ) then
318: C
319: C         call CCOMP( FILE, LEN(FILE), LINEU(7:), INN, IFFF )
320: C         open( JOUT, form='FORMATTED', file=FILE(:INN),
321: C &             status='UNKNOWN', iostat=IOS )
322: C         if ( IOS.ne.0 ) then
323: C             write(IPR,*) 'SOURCE OUTPUT FILE OPEN ERROR: IOS =', IOS
324: C             write(6,*) - 999
325: C         stop 666

```

```

326: C     end if
327: C
328: C     else if ( LINE(1:9).eq.'DIRECTIVE' ) then
329: C
330: C         IS      = 10
331: C         call NOBLNK( LINE, IS, IE, LENG )
332: C         if ( IE-IS+1.gt.LEN(DIRCTV(1)) ) then
333: C             write(IPR,*) ' too long directive <', LINE(IS:IE), '>'
334: C             write(IPR,*) ' truncate to ', LEN(DIRCTV(1)), ' characters'
335: C             IE      = IS + LEN(DIRCTV(1)) - 1
336: C         end if
337: C         do 110 I = 1, NDRV
338: C             if ( DIRCTV(I).eq.LINE(IS:IE) ) go to 120
339: C 110         continue
340: C
341: C         NDRV      = NDRV + 1
342: C         if ( NDRV.gt.MAXDRV ) then
343: C             write(IPR,*) ' too many directives !! max = ', MAXDRV
344: C             write(6,*) - 999
345: C             stop 666
346: C         end if
347: C         DIRCTV(NDRV) = LINE(IS:IE)
348: C
349: C 120         continue
350: C
351: C     else if ( LINE(1:7).eq.'MONITOR' ) then
352: C
353: C         IS      = 8
354: C         call NOBLNK( LINE, IS, IE, LENG )
355: C         if ( IS.le.LENG ) then
356: C             write(FORMT(:5), ' ((' (I'',i2,'') ')) ' ) IE - IS + 1
357: C             read(LINE(IS:IE),fmt=FORMT(:5),err=130) IIPR
358: C             if ( IIPR.eq.ISTD .or. IIPR.eq.6 .or. IIPR.eq.JIN
359: C &                 .or. IIPR.eq.JOUT ) then
360: C                 write(IPR,*) 'MONITORING OUTPUT UNIT CANNOT BE ', IIPR
361: C             else
362: C                 IPR      = IIPR
363: C             end if
364: C             go to 100
365: C 130         write(IPR,*) 'INVALID MONITORING OUTPUT UNIT <',
366: C &                 LINE(IS:IE), '>'
367: C             go to 100
368: C         end if
369: C
370: C     else if ( LINE(1:1).ne.'*' ) then
371: C
372: C         write(IPR,*) 'INVALID COMMAND'
373: C         write(IPR,*) LINE(:LENG)
374: C         write(6,*) -999
375: C         stop 999
376: C     end if
377: C     go to 100
378: C
379: C
380: C
381: C 140 continue
382: C
383: C     if( jin .ne. istd ) then
384: C         open(jin,form='FORMATTED',iostat=ios)
385: C         if(ios.ne.0) then
386: C             write(ipr,*) 'Source input file open error: ios =',ios
387: C             write(6,*) -999
388: C             stop 666
389: C         endif
390: C     endif

```

src/tools/fat.f

```

391: C
392: C   open(jout,form='FORMATTED',status='UNKNOWN',iostat=ios)
393: C   if(ios.ne.0) then
394: C       write(ipr,*) 'Source output file open error: ios =',ios
395: C       write(6,*) -999
396: C       stop 666
397: C   endif
398: C
399: C   NLINE   = 0
400: C   IFNEST  = 0
401: C
402: C 150 call INLINE( JIN, LINE, LENG, IERR, IEND )
403: C
404: C   if ( IEND.ne.0 ) go to 270
405: C   if ( IERR.ne.0 ) then
406: C       write(6,*) - 999
407: C       stop 666
408: C   endif
409: C
410: C   if ( LENG.eq.0 ) go to 240
411: C
412: C   NLINE   = NLINE + 1
413: C   NCHGO   = NCHG
414: C
415: C
416: C
417: C ==== C/# lines =====
418: C
419: C
420: C   LINEU(1:3) = LINE(1:3)
421: C   call UPPER( LINEU(1:3) )
422: C
423: C   if ( LINEU(1:3).eq.'C/#' ) then
424: C       LINEU = LINE(1:LENG)
425: C       call UPPER( LINEU )
426: C
427: C
428: C   ... ignore ' ' & '#' between 'C/#' and assim command.
429: C
430: C   MM      = 4
431: C   do 160 M = MM, LENG
432: C       if ( LINEU(M:M).ne.' ' .and.LINEU(M:M).ne.'#' ) then
433: C           MM      = M
434: C           go to 170
435: C       endif
436: C 160 continue
437: C   MM      = LENG
438: C
439: C 170 call NOBLNK( LINEU, MM, IE, LENG )
440: C   if ( MM.gt.LENG ) go to 240
441: C
442: C
443: C
444: C   if ( LINEU(MM:MM+2).eq.'SET' ) then
445: C
446: C       if ( IFNEST.le.0 .or. IFNEST.gt.0.and.LCUR )
447: C       &       call SETPRM( LINEU(MM+3:LENG), IERR2 )
448: C
449: C   else if ( LINEU(MM:MM+4).eq.'UNSET' ) then
450: C
451: C       if ( IFNEST.le.0 .or. IFNEST.gt.0.and.LCUR )
452: C       &       call UNSETP( LINEU(MM+5:LENG), IERR2 )
453: C
454: C   else if ( LINEU(MM:MM+8).eq.'DIRECTIVE' ) then
455: C

```

```

456: C   if ( IFNEST.le.0 .or. IFNEST.gt.0.and.LCUR ) then
457: C       IS      = 10
458: C       call NOBLNK( LINEU, IS, IE, LENG )
459: C
460: C       if ( IE-IS+1.gt.LENG(DIRCTV(1)) ) then
461: C           write(IPR,*) ' too long directive <', LINEU(IS:IE),
462: C               '>'
463: C           write(IPR,*) ' truncate to ', LENG(DIRCTV(1)),
464: C               ' characters'
465: C           IE      = IS + LENG(DIRCTV(1)) - 1
466: C       endif
467: C
468: C   do 180 I = 1, NDRV
469: C       if ( DIRCTV(I).eq.LINEU(IS:IE) ) go to 190
470: C 180 continue
471: C
472: C   NDRV      = NDRV + 1
473: C   if ( NDRV.gt.MAXDRV ) then
474: C       write(IPR,*) ' too many directives !! max = ', MAXDRV
475: C       write(6,*) -999
476: C       stop 666
477: C   endif
478: C   DIRCTV(NDRV) = LINEU(IS:IE)
479: C
480: C 190 continue
481: C   endif
482: C
483: C   .... IF ...
484: C
485: C   else if ( LINEU(MM:MM+2).eq.'IF '
486: C       &       .or. LINEU(MM:MM+6).eq.'ELSEIF '
487: C       &       .or. LINEU(MM:MM+4).eq.'ELSE '
488: C       &       .or. LINEU(MM:MM+5).eq.'ENDIF ' ) then
489: C
490: C       if ( LINEU(MM:MM+2).eq.'IF ' ) then
491: C           IFNEST = IFNEST + 1
492: C           if ( IFNEST.gt.MXNEST ) go to 280
493: C
494: C   .... nest level to restore when ENDIF appeared ...
495: C
496: C   IFEND(IFNEST) = IFNEST - 1
497: C
498: C   if ( LENG.le.MM+3 ) then
499: C       write(IPR,*) 'NO EXPRESSION AFTER "IF":'
500: C       write(IPR,*) NLINE
501: C       write(6,*) - 999
502: C       stop 666
503: C   endif
504: C
505: C ccc      CALL LPEVAL( LINEU(MM+3:LENG), LEV, IERR3 )
506: C
507: C   LEV      = LDENTK(LINEU(MM+3:LENG),IERR3)
508: C
509: C   if ( IERR3.ne.0 ) then
510: C       write(IPR,*) 'INVALID EXPRESSION :'
511: C       write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
512: C       write(6,*) - 999
513: C       stop 666
514: C   endif
515: C
516: C   LOGIF(IFNEST) = LEV
517: C
518: C   .... ELSEIF ...
519: C
520: C   else if ( LINEU(MM:MM+6).eq.'ELSEIF ' ) then

```

src/tools/fat.f

```

521:         if ( IFNEST.eq.0 ) then
522:             write(IPR,*) ' ELSEIF BEFORE IF OR ELSEIF'
523:             write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
524:             write(6,*) - 999
525:             stop 666
526:         end if
527:         LOGIF(IFNEST) = .not.LOGIF(IFNEST)
528: C
529:         IFNEST = IFNEST + 1
530:         if ( IFNEST.gt.MXNEST ) go to 280
531: C
532: C .... nest level to restore when ENDIF appeared ...
533: C
534:         IFEND(IFNEST) = IFEND(IFNEST-1)
535: C
536:         if ( LENG.le.MM+7 ) then
537:             write(IPR,*) 'NO EXPRESSION AFTER "ELSEIF":'
538:             write(IPR,*) NLINE
539:             write(6,*) - 999
540:             stop 666
541:         end if
542: C
543: Ccccc         CALL LPEVAL( LINEU(MM+7:LENG), LEV, IERR3 )
544: C
545:         LEV = LDENTK( LINEU(MM+7:LENG), IERR3 )
546: C
547:         if ( IERR3.ne.0 ) then
548:             write(IPR,*) 'INVALID EXPRESSION : '
549:             write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
550:             write(6,*) - 999
551:             stop 666
552:         end if
553: C
554:         LOGIF(IFNEST) = LEV
555: C
556: C .... ELSE ...
557: C
558:         else if ( LINEU(MM:MM+4).eq.'ELSE ' ) then
559:             if ( IFNEST.eq.0 ) then
560:                 write(IPR,*) ' ELSE BEFORE IF OR ELSEIF'
561:                 write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
562:                 write(6,*) - 999
563:                 stop 666
564:             end if
565:             LOGIF(IFNEST) = .not.LOGIF(IFNEST)
566: C
567: C .... ENDIF ...
568: C
569:         else if ( LINEU(MM:MM+5).eq.'ENDIF ' ) then
570:             if ( IFNEST.eq.0 ) then
571:                 write(IPR,*) ' ENDIF BEFORE IF OR ELSE'
572:                 write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
573:                 write(6,*) - 999
574:                 stop 666
575:             end if
576:             IFNEST = IFEND(IFNEST)
577:         end if
578: C
579:         LCUR = .true.
580:         if ( IFNEST.gt.0 ) then
581:             do 200 I = 1, IFNEST
582:                 LCUR = LCUR.and.LOGIF(I)
583:             200 continue
584:         end if
585: C

```

```

586: C
587: C
588: C
589:         else
590:             write(IPR,*) 'INVALID COMMAND'
591:             write(IPR,*) NLINE, ' <', LINEU(:LENG), '>'
592:             write(6,*) - 999
593:             stop 999
594:         end if
595: C
596: C
597: C ==== ordinary lines =====
598: C
599: C
600:         else
601: C
602:             if ( IFNEST.gt.0 ) then
603: C
604:                 .... '*' lines to recover ....
605: C
606: C
607:                 if ( LCUR ) then
608:                     if ( LINE(1:1).eq.'*' ) then
609: C
610:                         check directive or/not
611: C
612:                     if ( NDRV.gt.0 ) then
613:                         LINEU(:LENG) = LINE(:LENG)
614:                         call UPPER( LINEU(:LENG) )
615: C
616:                         do 210 I = 1, NDRV
617:                             LD = INDEX(DIRCTV(I)/// ' ', ' ') - 1
618:                             if ( DIRCTV(I)(:LD).eq.LINEU(:LD) ) then
619:                                 write(IPR,*) ' <DIRECTIVE>: ', LINE(:LENG)
620:                                 go to 240
621:                             end if
622:                         210 continue
623:                     end if
624: C
625: C
626: C
627:                     NCHG = NCHG + 1
628:                     LINE(1:1) = ' '
629: C
630: C .... some special character on 2nd column ....
631: C
632:                     if ( INDEX(' 0123456789',LINE(2:2)).eq.0 ) then
633:                         do 220 I = 1, LENG - 1
634:                             LINE(I:I) = LINE(I+1:I+1)
635:                         220 continue
636:                         LINE(LENG:LENG) = ' '
637:                         LENG = LENG - 1
638:                         write(IPR,*) ' <SPECIAL LINE>: ', LINE(:LENG)
639:                     end if
640:                 end if
641: C
642: C .... lines to commentize ...
643: C
644:         else if ( .not.LCUR ) then
645:             if ( INDEX('0123456789',LINE(1:1)).ne.0 ) then
646:                 write(IPR,*) 'NUMERIC CHARACTER ON FIRST COLUMN : '
647:                 write(IPR,*) NLINE, ' <', LINE(:LENG), '>'
648:                 write(6,*) - 999
649:                 stop 666
650:             end if

```

src/tools/fat.f

```

651: C
652: C      .... simply put '*' ....
653: C
654: C      if ( LINE(1:1).eq.' ' ) then
655: C          NCHG = NCHG + 1
656: C          LINE(1:1) = '*'
657: C
658: C      else if ( LINE(1:1).ne.'C'.and.LINE(1:1).ne.'c' ) then
659: C
660: C      .... shift 1 column & put '*' ....
661: C
662: C          if ( LINE(1:1).ne.'*'
663: C              .or.
664: C              (LINE(1:1).eq.'*'
665: C              .and.INDEX(' 0123456789',LINE(2:2)).eq.0 ) ) then
666: C              NCHG = NCHG + 1
667: C              do 230 I = MIN(LEN(LINE)-1,LENG), 1, -1
668: C                  LINE(I+1:I+1) = LINE(I:I)
669: C 230      continue
670: C          LENG = MIN(LEN(LINE),LENG+1)
671: C          LINE(1:1) = '*'
672: C      end if
673: C  end if
674: C  end if
675: C  end if
676: C  end if
677: C
678: C  ===== output lines =====
679: C
680: C 240 continue
681: C      do 250 I = LENG, 1, -1
682: C          if ( LINE(I:I).ne.' ' ) go to 260
683: C 250 continue
684: C 260 NL = I
685: C
686: C      write(JOUT,'(A)') LINE(:NL)
687: C
688: C      if ( LINE(1:3).eq.'C/#' .or. LINE(1:3).eq.'c/#' ) then
689: C          write(IPR,'(A)') LINE(:NL)
690: C          if ( IFNEST.gt.0 ) then
691: C              if ( LCUR ) then
692: C                  write(IPR,'(A)') ' << ON >>'
693: C              else
694: C                  write(IPR,'(A)') ' << OFF >>'
695: C              end if
696: C          end if
697: C      else if ( NCHG.gt.NCHG0 ) then
698: C          write(IPR,'(A)') LINE(:NL)
699: C      end if
700: C
701: C
702: C
703: C
704: C      go to 150
705: C
706: C
707: C 270 continue
708: C      if ( IFNEST.gt.0 ) then
709: C          write(IPR,*) 'INCONSISTENT C/#IF BLOCKS '
710: C          write(6,*) - 999
711: C          stop 666
712: C      end if
713: C      write(6,*) NCHG
714: C      stop
715: C

```

```

716: 280 write(IPR,*) 'TOO DEEP IF NEST (MAX=', MXNEST, ' )'
717:      write(6,*) - 999
718:      stop 999
719: C
720: C      end
721: C
722: C
723: C
724: C*****
725: C*****
726: C*****
727: C*****
728: C
729: C
730: C
731: C      subroutine INLINE( JIN, LINE, LENG, IERR, IEND )
732: C
733: C      parameter( NCLINE = 72 )
734: C      character*(NCLINE) LINE
735: C
736: C      IEND = 0
737: C      IERR = 0
738: C      LINE = ' '
739: C      read(JIN,err =110,iostat =IOS,end =100,fmt ='(A)') LINE
740: C      LENG = LEN(LINE)
741: C      return
742: C
743: C 100 IEND = 1
744: C      return
745: C 110 IERR = MAX(1,IOS)
746: C      return
747: C      end
748: C
749: C
750: C
751: C*****
752: C*****
753: C*****
754: C*****
755: C
756: C
757: C
758: C      subroutine SETPRM( STR, IERR )
759: C
760: C      character*(*) STR
761: C      logical LEV
762: C
763: C      PARAMETER ( IPR = 7 )
764: C
765: C      parameter( IPR = 0 )
766: C
767: C
768: C      =====
769: C      .... SYMBOLIC PARAMETERS & VALUES OF THEM ....
770: C      =====
771: C
772: C      These data must be saved call to call.
773: C
774: C      parameter( MAXP = 32, MAXSP = 4*MAXP, LNAM = 16 )
775: C
776: C
777: C      character*(LNAM) PNAME(MAXP)
778: C      logical PONOFF(MAXP)
779: C      integer NPARM
780: C

```

src/tools/fat.f

```

781:      character*(LNAM) SPEC(MAXSP)
782:      logical SONOFF(MAXSP)
783:      integer NSPEC, IPARM(MAXSP)
784: C
785: C ... local data
786: C
787:      character*(LNAM) PARM
788:      character*(LNAM) SPC
789: C
790:      data PNAME /MAXP*' '/
791:      data PONOFF /MAXP*.false./
792:      data NPARM /0/
793: C
794:      data SPEC /MAXSP*' '/
795:      data SONOFF /MAXSP*.false./
796:      data NSPEC /0/
797:      data IPARM /MAXSP*0/
798: C
799:      ISET      = 1
800:      go to 100
801: C
802: C
803:      entry UNSETP(STR,IERR)
804: C
805: C
806:      ISET      = 0
807: C
808: 100 IS      = 1
809:      IERR      = 0
810: C
811: C .... get parameter name ...
812: C
813:      PARM      = ' '
814:      call STRTOK( STR, ' (', PARM, LP, IS )
815: C
816:      if ( LP.eq.0 ) then
817:          IERR    = 1
818:          return
819:      end if
820: C
821:      II        = INDEX(STR(IS:),'(')
822:      if ( II.gt.0.and.INDEX(STR(IS:),'').eq.0 ) then
823:          IERR    = 1
824:          return
825:      end if
826: C
827:      IS        = IS + II
828: C
829: C .... wild card character for parameter .
830: C
831:      if ( INDEX(PARM,'*').ne.0 .or. INDEX(PARM,'?').ne.0 ) then
832:          IERR    = 1
833:          return
834:      end if
835: C
836: C
837:      IM        = 0
838:      NEW       = 0
839:      do 110 I = 1, NPARM
840:          call MATCH( IM, PARM, PNAME(I) )
841:          if ( PNAME(I).eq.PARM ) then
842:              IPN      = I
843:              go to 120
844:          end if
845: 110 continue
846: C
847: C .... no matching parameter name ....
848: C
849:      if ( ISET.eq.1 ) then
850:          NPARM      = NPARM + 1
851:          if ( NPARM.gt.MAXP ) go to 170
852:          PNAME(NPARM) = PARM
853:          PONOFF(NPARM) = .true.
854:          IPN        = NPARM
855:          NEW        = 1
856:      else
857: C
858: C ... does nothing for UNSET ...
859: C
860:          IERR      = 0
861:          return
862:      end if
863: C
864: C .... without specification ....
865: C
866: 120 if ( II.eq.0 ) then
867:     if ( ISET.eq.1 ) then
868:         PONOFF(IPN) = .true.
869:     else
870:         PONOFF(IPN) = .false.
871:         do 130 I = 1, NSPEC
872:             if ( IPARM(I).eq.IPIN ) SONOFF(I) = .false.
873: 130         continue
874:         end if
875:         return
876: C
877: C .... with specification ....
878: C
879:     else if ( II.gt.0 ) then
880: C
881: 140         SPC      = ' '
882:         call STRTOK( STR, ' )', SPC, IKK, IS )
883: C
884:         if ( IKK.gt.0 ) then
885:             do 150 I = 1, NSPEC
886:                 if ( IPARM(I).eq.IPIN.and.SPEC(I).eq.SPC ) then
887:                     ISPN      = I
888:                     go to 160
889:                 end if
890: 150             continue
891: C
892:             if ( ISET.eq.1 ) then
893:                 NSPEC      = NSPEC + 1
894:                 if ( NSPEC.gt.MAXSP ) go to 170
895:                 SPEC(NSPEC) = SPC
896:                 IPARM(NSPEC) = IPN
897:                 SONOFF(NSPEC) = .true.
898:                 ISPN      = NSPEC
899:             else
900:                 go to 140
901:             end if
902: C
903: 160         if ( ISET.eq.1 ) then
904:             SONOFF(ISPN) = .true.
905:         else
906:             SONOFF(ISPN) = .false.
907:         end if
908:         go to 140
909:     end if
910: end if

```

src/tools/fat.f

```

911:      return
912: C
913: 170 write(IPR,*) 'XXX SET : PARAMETER MEMORY OVERFLOW ', ' NPARM ',
914:      &      NPARM, '>', MAXP, ' OR NSPEC ', NSPEC, '>', MAXSP
915:      IERR = 999
916:      return
917: C
918: C
919: C ===== logical evaluation =====
920: C
921:      entry LPEVAL(STR,LEV,IERR)
922: C
923:      LEV = .false.
924:      IERR = 0
925: C
926: C .... get parameter name ...
927: C
928:      IS = 1
929:      PARM = ' '
930:      call STRTOK( STR, '(', PARM, LP, IS )
931:      if ( LP.eq.0 ) then
932:          IERR = 1
933:          return
934:      end if
935: C
936:      II = INDEX(STR(1:), '(')
937:      if ( II.gt.0.and.INDEX(STR(1:), '(').eq.0 ) then
938:          IERR = 1
939:          return
940:      end if
941: C
942:      IS = IS + II
943: C
944: C .... wild card character for parameter name with specification .
945: C
946:      if ( II.gt.0.and.INDEX(PARM,'*').ne.0 .or. INDEX(PARM,'?').ne.0 )
947:      &      then
948:          IERR = 1
949:          return
950:      end if
951: C
952: C
953:      IM = 0
954:      NEW = 0
955:      do 180 I = 1, NPARM
956:          call MATCH( IM, PARM, PNAME(I) )
957:          if ( IM.ne.0 ) then
958:              IPN = I
959:              go to 190
960:          end if
961: 180 continue
962: C
963: C .... no matching parameter name ....
964: C
965:      LEV = .false.
966:      return
967: 190 continue
968: C
969: C .... without specification ....
970: C
971: C
972:      if ( II.eq.0 ) then
973:          LEV = PONOFF(IPN)
974:          return
975: C

```

```

976: C .... with specification ....
977: C
978:      else if ( II.gt.0 ) then
979: C
980: 200      SPC = ' '
981:          call STRTOK( STR, ' ', SPC, IKK, IS )
982: C
983:          if ( IKK.gt.0 ) then
984:              do 210 I = 1, NSPEC
985:                  if ( IPARM(I).eq.IPN ) then
986:                      call MATCH( IM, SPC, SPEC(I) )
987:                      if ( IM.ne.0.and.SONOFF(I) ) then
988:                          LEV = .true.
989:                          return
990:                      end if
991:                  end if
992: 210          continue
993:              go to 200
994:          end if
995:      end if
996: C
997:      return
998:      end
999: C
1000: C
1001: C
1002: C*****
1003: C*****
1004: C*****
1005: C*****
1006: C
1007: C
1008: C
1009:      subroutine STRTOK( STR1, STR2, TOKEN, ITLEN, IPOS )
1010: C
1011: C ***** THIS ROUTINE GET TOKEN FROM A STRING *****
1012: C
1013: C      EXTRACT A TOKEN THAT DOES NOT INCLUDE ANY CHARACTERS IN 'STR2'
1014: C      -----
1015: C      FROM STR1(IPOS: ) INTO 'TOKEN'.
1016: C      MOVE POINTER IPOS TO NEXT POSITION OF TOKEN IN STR1.
1017: C
1018: C < SAMPLE PROGRAM >
1019: C*****
1020: *      CHARACTER*72 LINE, TOKEN
1021: *      READ(5,*) LINE
1022: *      NT = 0
1023: *      IP = 1
1024: * 100 CALL STRTOK( LINE, ' .:;/', TOKEN, LT, IP )
1025: *      IF(LT.EQ.0) THEN
1026: *          WRITE(6,*) ' == END DATA ==   TOKEN NUMBER = ',LT
1027: *          GOTO 999
1028: *      ELSE
1029: *          NT = NT + 1
1030: *          WRITE(6,*) 'TOKEN ',NT,' <',TOKEN(1:LT),'>   LENGTH ',LT
1031: *          GOTO 100
1032: *      ENDIF
1033: C
1034: * 999 STOP
1035: ***** END
1036: C*****
1037: C
1038:      character*(*) STR1, TOKEN
1039:      character*(*) STR2
1040: C

```


src/tools/fat.f

```

1041:      if ( IPOS.gt.LEN(STR1) ) then
1042:          ITLEN = 0
1043:          return
1044:      end if
1045: C
1046: C ----- FIND TOKEN HEAD -----
1047: C
1048: C
1049:      do 100 I1 = IPOS, LEN(STR1)
1050:          if ( INDEX(STR2,STR1(I1:I1)).eq.0 ) go to 110
1051: 100 continue
1052: C
1053: C .... NO TOKEN FOUND ....
1054: C
1055:      IPOS = I1
1056:      ITLEN = 0
1057:      return
1058: C
1059: C ----- FIND TOKEN TAIL -----
1060: C
1061: 110 continue
1062:      do 120 I2 = I1, LEN(STR1)
1063:          if ( INDEX(STR2,STR1(I2:I2)).ne.0 ) go to 130
1064: 120 continue
1065: C
1066: 130 IPOS = I2
1067:      ITLEN = I2 - I1
1068:      TOKEN = STR1(I1:I2-1)
1069:      return
1070:      end
1071: C
1072: C
1073: C
1074: C*****
1075: C*****
1076: C*****
1077: C*****
1078: C
1079: C
1080: C
1081:      logicalfunction LDENTK(LINE,IERR)
1082: C
1083: C MVP/GMVP UTILITY
1084: C
1085: C=====
1086: C PURPOSE: INTERPRETATION OF LOGICAL EXPRESSION
1087: C .... STACK. ....
1088: C
1089: C      STK : logical value stack (counter: ISTN )
1090: C      ISTK : operator # stack (counter: ISTO )
1091: C=====
1092: Cc
1093:      character*(*) LINE
1094: C
1095:      parameter( IPR = 7 )
1096: C
1097:      parameter( MS = 64 )
1098:      logical STK(MS)
1099:      integer ISTK(MS)
1100: C
1101: C ... TEMPORARY ...
1102: C
1103:      logical LEV, LGA, LGB
1104:      character*32 FORMT
1105: C
1106: C
1107: C
1108: C =====
1109: C
1110: C .... OPERATOR, NUMBER ....
1111: C
1112:      parameter( KEND = 0 )
1113:      parameter( KLBR = 1 )
1114:      parameter( KRBR = 2 )
1115:      parameter( KOR = 3 )
1116:      parameter( KAND = 4 )
1117:      parameter( KNOT = 5 )
1118:      parameter( KXOR = 6 )
1119:      parameter( NMOPR = KXOR )
1120: C
1121: C .... LOGICAL OPERATORS & THEIR LENGTH IN CHARACTER ...
1122: C
1123:      character*8 OPERTR(NMOPR)
1124:      integer LOP(NMOPR)
1125: C
1126: C .... TABLE OF CALCULATION RULES .....
1127: C
1128:      integer ICALC(0:NMOPR,0:NMOPR)
1129: C
1130: C
1131: C .... operator and their lengths
1132: C
1133:      data(OPERTR(I),LOP(I),I=1,NMOPR) /
1134:      & ' ( ' , 1,
1135:      & ' ) ' , 1,
1136:      & '.OR.' , 4,
1137:      & '.AND.' , 5,
1138:      & '.NOT.' , 5,
1139:      & '.XOR.' , 5,
1140:      & /
1141: C
1142: C .... STATE TABLE ....
1143: C
1144:      data ICALC /
1145:      & 4, 1, 5, 1, 1, 1, 1,
1146:      & 5, 1, 3, 1, 1, 1, 1,
1147:      & 0, 0, 0, 0, 0, 0, 0,
1148:      & 2, 1, 2, 2, 1, 0, 2,
1149:      & 2, 1, 2, 2, 2, 0, 2,
1150:      & 6, 1, 6, 6, 6, 0, 6,
1151:      & 2, 1, 2, 2, 2, 0, 2 /
1152: C
1153: C
1154:      data INIT /1/
1155: C
1156: C -----
1157: C
1158: Ccc PARAMETER ( IECHO = 66 )
1159: C
1160: C ---- initialization ( number of parameter = 0 )
1161: C
1162:      IECHO = -1
1163:      if ( INIT.eq.1 ) then
1164: C
1165: C ... initialize echoback message file ....
1166: C
1167:      INIT = 0
1168:      if ( IECHO.ge.0 ) then
1169:          write(IECHO,('( '1',A/A/))')
1170:      & ' == MONITOR OUTPUT OF <LDENTK> == ' ,

```

src/tools/fat.f

```

1171:      &          '      (CALCULATOR UNIT / VERSION 3.0   FEB 1993 ) '
1172:      end if
1173:      end if
1174: C
1175:      LDENTK = .false.
1176: C
1177:      IERR   = 0
1178: C
1179:      .... logical data stack pointer ...
1180: C
1181:      ISTN   = 0
1182: C
1183:      .... operator number stack pointer ...
1184: C
1185:      (DUMMY OPERATOR kend ( END ) is pushed )
1186: C
1187:      ISTO   = 1
1188:      ISTK(ISTO) = KEND
1189: C
1190:      ..... MEMORIZE MAXIMUM VALUE-STACK DEPTH & PARAMETER REFERENCE
1191: C
1192:      ISTKM  = 0
1193:      IPFL   = 0
1194: C
1195: C ..... previous item : k0 = 1/2 = operator / parameter (0:initial)
1196: C
1197:      K0     = 0
1198:      LEND   = LEN(LINE)
1199:      IS     = 1
1200: C
1201: C ..... get parameter token or operator .....
1202: C
1203:      100 continue
1204:      ISO    = IS
1205: C
1206:      if ( IS.gt.LEND .or. IS.le.LEND.and.LINE(IS:).eq.' ' ) then
1207:      K      = KEND
1208:      K0     = 1
1209:      IS     = LEND + 1
1210:      go to 140
1211: else
1212:      II     = IRINDX(LINE(IS:),' ') + IS - 1
1213:      do 110 K = 1, NMOPR
1214:      if ( LINE(II:II+LOP(K)-1).eq.OPERTR(K)(:LOP(K)) ) then
1215:      IS     = II + LOP(K)
1216:      K0     = 1
1217:      go to 140
1218:      end if
1219:      110 continue
1220: C
1221: C Get string or string( strings ) type token.
1222: C
1223: C
1224:      IBB    = 0
1225:      IL     = 0
1226:      do 120 L = II, LEND
1227:      if ( IBB.eq.0 ) then
1228:      if ( LINE(L:L).eq.' ' .or. LINE(L:L).eq.'.' ) then
1229:      IL     = L - 1
1230:      go to 130
1231:      else if ( LINE(L:L).eq.'(' ) then
1232:      IBB    = 1
1233:      end if
1234:      else if ( LINE(L:L).eq.')' ) then
1235:      IL     = L
1236:
1237:      go to 130
1238:      end if
1239:      120 continue
1240:      IL     = L - 1
1241: C
1242: C .... evaluate logical value of parameter token ....
1243: C
1244: C
1245:      130 call LPEVAL( LINE(II:IL), LEV, IERR )
1246: C
1247:      if ( IERR.ne.0 ) return
1248: C
1249: C .... more than two successive parameters ....
1250: C
1251:      if ( K0.eq.2 ) then
1252:      IERR = 1
1253:      return
1254:      end if
1255:      IS   = IL + 1
1256: C
1257:      K0   = 2
1258: C
1259:      ISTN = ISTN + 1
1260:      ISTKM = MAX(ISTKM,ISTN)
1261: C
1262: C ..... STACK OVERFLOW .....
1263: C
1264:      if ( ISTN.gt.MS ) go to 230
1265: C
1266:      STK(ISTN) = LEV
1267: C
1268:      go to 100
1269: C
1270:      end if
1271: C
1272: C
1273: C
1274: C
1275: C parameter ( kend = 0 )
1276: C parameter ( klbr = 1 )
1277: C parameter ( krbr = 2 )
1278: C parameter ( kor  = 3 )
1279: C parameter ( kand = 4 )
1280: C parameter ( knot = 5 )
1281: C parameter ( kxor = 6 )
1282: C
1283: C A ... TOP OF OPERATION STACK <ISTK>
1284: C K ... NEW OPERATOR
1285: C
1286: C < OPERATIONS >
1287: C
1288: C 0 : non-existent (A,K) pair
1289: C 1 : PUSH K TO OPERATION STACK <ISTK>
1290: C 2 : * OPERATION BY A.
1291: C      * POP OPERATION STACK <ISTK> (discard)
1292: C      * CONTINUE (A,K) OPERATION
1293: C 3 : POP OPERATION STACK <ISTK>
1294: C 4 : END OF CALCULATION. RETURN TOP OF VALUE STACK <STK>
1295: C 5 : ERROR
1296: C 6 : like 2. but operator is unary.
1297: C
1298: C
1299: C A / K
1300: C

```

src/tools/fat.f

```

1301: C          END ( ) OR AND NOT XOR
1302: C
1303: C 0 END 4 1 5 1 1 1 1
1304: C 1 ( 5 1 3 1 1 1 1
1305: C 2 ) 0 0 0 0 0 0 0
1306: C 3 OR 2 1 2 2 1 0 2
1307: C 4 AND 2 1 2 2 2 0 2
1308: C 5 NOT 6 1 6 6 6 0 6
1309: C 6 XOR 2 1 2 2 2 0 2
1310: C
1311: C
1312: C 140 go to 150,160,170,180,190,200) ICALC(K,ISTK(ISTO))
1313: C
1314: C write(IPR,*) 'XXX(LOGICAL_CALCULATOR) OPERATION ERROR ',
1315: C & '(MAY BE A BUG)', ' PLEASE CONTACT AUTHORS!!! '
1316: C write(IPR,*) (ISTK(I),I=1,ISTN)
1317: C write(IPR,*) (STK(I),I=1,ISTN)
1318: C stop 666
1319: C
1320: C ..... PUSH .....
1321: C
1322: C 150 ISTO = ISTO + 1
1323: C if ( ISTO.gt.MS ) go to 230
1324: C ISTK(ISTO) = K
1325: C go to 100
1326: C
1327: C ..... OPERATION BY AN OPERATOR AT THE TOP OF STACK ....
1328: C
1329: C 160 IOP = ISTK(ISTO)
1330: C
1331: C two operands
1332: C
1333: C if ( ISTN.le.1 ) then
1334: C write(IPR,*) 'XXX(CALCULATOR) SYNTAX ERROR'
1335: C IERR = 1
1336: C go to 240
1337: C end if
1338: C LGA = STK(ISTN)
1339: C LGB = STK(ISTN-1)
1340: C if ( IOP.eq.KOR ) then
1341: C LGA = LGA .or. LGB
1342: C else if ( IOP.eq.KXOR ) then
1343: C LGA = LGA.neqv.LGB
1344: C else if ( IOP.eq.KAND ) then
1345: C LGA = LGA.and.LGB
1346: C end if
1347: C
1348: C ..... POP logical STACK .....
1349: C
1350: C ISTN = ISTN - 1
1351: C
1352: C ..... substitute by new value ....
1353: C
1354: C STK(ISTN) = LGA
1355: C
1356: C ..... POP OPERATOR STACK .....
1357: C
1358: C ISTO = ISTO - 1
1359: C
1360: C go to 140
1361: C
1362: C ..... POP operator stack .....
1363: C
1364: C 170 ISTO = ISTO - 1
1365: C go to 100
1366: C
1367: C ..... TERMINATE CALCULATION .....
1368: C
1369: C 180 go to 210
1370: C
1371: C ..... SYNTAX ERROR !! .....
1372: C
1373: C 190 write(IPR,*) 'XXX(CALCULATOR) SYNTAX ERROR '
1374: C IERR = 1
1375: C go to 240
1376: C
1377: C
1378: C
1379: C 200 if ( ISTK(ISTO).eq.KNOT ) then
1380: C STK(ISTN) = .not.STK(ISTN)
1381: C ISTO = ISTO - 1
1382: C end if
1383: C go to 140
1384: C
1385: C ..... END OF PROCESSING
1386: C
1387: C 210 LDENTK = STK(ISTN)
1388: C
1389: C
1390: C
1391: C if ( ISTKM.gt.1 .or. IPFL.gt.0 ) then
1392: C if ( IECHO.ge.0 ) then
1393: C write(IECHO,(' ' ,A)) LINE(IS:LEND)
1394: C do 220 I = 1, ISTN
1395: C write(IECHO,*) ' ==> ', STK(I)
1396: C 220 continue
1397: C end if
1398: C end if
1399: C
1400: C return
1401: C
1402: C ..... ERROR : STACK IS FULL
1403: C
1404: C 230 write(IPR,*) 'XXX(LDENTK) STACK IS FULL XXX '
1405: C IERR = 4
1406: C go to 240
1407: C
1408: C ..... PRINTOUT POSITION OF ERROR ....
1409: C
1410: C 240 write(IPR,('1X,' LINE: <' ,A,'>')) LINE
1411: C FORMT = ' '
1412: C write(FORMT,('('1X,T',I3,'',',I3,'('' '*'''))'))
1413: C & 9 + IS0, 9 + MAX(IS-1,LEN(LINE))
1414: C write(IPR,fmt =FORMT)
1415: C return
1416: C end
1417: C
1418: C
1419: C
1420: C *****
1421: C *****
1422: C *****
1423: C *****
1424: C
1425: C
1426: C
1427: C function IRINDX(STR,CHAR)
1428: C
1429: C Return position of character other than char.
1430: C (returns 0 if cannot find character)

```

src/tools/fat.f

```
1431: C
1432:     character*(*) STR
1433:     character*1 CHAR
1434: C
1435:     do 100 I = 1, LEN(STR)
1436:         if ( STR(I:I).ne.CHAR ) go to 110
1437:     100 continue
1438: C
1439:         IRINDX = 0
1440: C
1441:     110 IRINDX = I
1442:     return
1443: end
1444: C
1445: C
1446: C
1447: C*****
1448: C*****
1449: C*****
1450: C*****
1451: C
1452: C
1453: C
1454:     subroutine MATCH( IM,     PATTRN,STR )
1455: C
1456: C     MVP/GMVP UTILITY
1457: C
1458: C=====
1459: C PURPOSE: STRING PATTERN MATCHING
1460: C HISTORY: PROGRAMMED BY M.SASAKI (30 APR 1992)
1461: C UPDATE:
1462: C 7 JUL 1993: '*' matches any string including null string.
1463: C           More sophisticated matching logic that allows any
1464: C           combinations of wildcard meta-characters.
1465: C=====
1466: C
1467: C     PATTRN & STR CAN INCLUDE BLANKS BUT THEY MUST BE PLACED AFTER ANY
1468: C     NON-BLANK CHARACERS. ( ignore characters after blank !! )
1469: C
1470: C     IM = 1 / 0 ... match / unmatched
1471: C
1472: C
1473: C << WILDCARD CHARACTERS >>
1474: C
1475: C '?' : ANY ONE CHARACTER ON THE PLACE OF '?'.
1476: C '*' : ANY CHARACTER STRING ( length >= 0 ).
1477: C
1478: C=====
1479: C
1480:     character*(*) PATTRN
1481:     character*(*) STR
1482: C
1483: C     .... default is unmatched ...
1484: C
1485:     IM = 0
1486: C
1487:     LP = INDEX(PATTRN,' ') - 1
1488:     if ( LP.lt.0 ) LP = LEN(PATTRN)
1489: C
1490:     LS = INDEX(STR,' ') - 1
1491:     if ( LS.lt.0 ) LS = LEN(STR)
1492:     if ( LS.eq.0 ) return
1493: C
1494: C
1495: C
```

```
1496: C     .... pattern has no '*' .....
1497: C
1498: C
1499: C
1500:     if ( INDEX(PATTRN(:LP),'*').eq.0 ) then
1501:         if ( LP.ne.LS ) return
1502:         do 100 I = 1, LP
1503:             if ( PATTRN(I:I).ne.'?' .and. PATTRN(I:I).ne.STR(I:I) ) return
1504:     100 continue
1505:     IM = 1
1506:     return
1507: end if
1508: C
1509: C
1510: C     .... pattern has '*' ....
1511: C
1512: C
1513: C << Logic of pattern matching with wild-card character '*' >>
1514: C
1515: C
1516: C
1517: C 0. Set "current pattern" to the whole pattern string.
1518: C     And set "object string" to the string checked.
1519: C
1520: C 1. Split the current-pattern string to two sub-patterns by '*'.
1521: C
1522: C     pattern = L * R
1523: C             or L      ( if the current pattern has no '*' )
1524: C
1525: C     L : left-side sub-pattern (does not include '*'. can be
1526: C         null string)
1527: C     R : right-side sub-pattern (may include '*')
1528: C
1529: C 2. If the left-side sub-pattern is not null string,
1530: C     check matching with the left-side sub-pattern.
1531: C
1532: C     If the sub-pattern does not match any substring of the object
1533: C     string (the leftmost substring for the first time), terminate as
1534: C     'unmatching'.
1535: C
1536: C     Else cut off the matching substring and characters on the left
1537: C     of the substring from the object string and proceed to step 4.
1538: C
1539: C 3. If the pattern has no '*',
1540: C     check matching between the current pattern and the rightmost
1541: C     substring of the object string.
1542: C
1543: C     If matched ,terminate matching as 'matching',else terminate
1544: C     matching as 'unmatching'.
1545: C
1546: C 4. Set the right-side sub-pattern as the current pattern and
1547: C     repeat the process from 1.
1548: C
1549: C
1550: C     IP = 1
1551: C     IS = 1
1552: C
1553: C 110 continue
1554: C
1555: C
1556: C     K = INDEX(PATTRN(IP:LP),'*')
1557: C
1558: C
1559: C ==== check matching with the left side of current '*'
1560: C
```

src/tools/fat.f

```

1561: C
1562:     if ( K.gt.1 ) then
1563: C
1564:         IS1      = LS - K + 2
1565: C
1566: C     .... for the first matching, the sub-pattern must match
1567: C         at the leftmost position.
1568: C
1569:         if ( IP.eq.1 ) IS1 = IS
1570:         do 130 ISS = IS, IS1
1571:             do 120 I = 0, K - 2
1572:                 if ( PATTRN(IP+I:IP+I).ne.'?'
1573:                     & .and.PATTRN(IP+I:IP+I).ne.STR(ISS+I:ISS+I) ) go to 130
1574:             120 continue
1575: C
1576: C     ---- left side matches ----
1577: C
1578:         IS      = ISS + I
1579:         go to 150
1580: C
1581:     130 continue
1582: C
1583: C     ---- unmatched ----
1584: C
1585:         return
1586: C
1587: C
1588: C     ==== check the rightmost string if no '*' remains in pattern ====
1589: C
1590: C
1591:         else if ( K.eq.0 ) then
1592:             if ( LS-IS.lt.LP-IP ) return
1593:             do 140 I = IP, LP
1594:                 if ( PATTRN(I:I).ne.'?'
1595:                     & .and.PATTRN(I:I).ne.STR(LS+I-LP:LS+I-LP) ) return
1596:             140 continue
1597:             IM      = 1
1598:             return
1599:         end if
1600: C
1601: C     (if the left-side sub-pattern is null, do only the following
1602: C         step.)
1603: C
1604:     150 IP      = IP + K
1605:         if ( IP.le.LP ) go to 110
1606: C
1607: C     ---- matched !! ----
1608: C
1609:         IM      = 1
1610: C
1611:         return
1612:     end
1613: C
1614: C
1615: C
1616:     subroutine UPPER( STR )
1617: C
1618: C=====
1619: C     change string to uppercase
1620: C=====
1621: C
1622:     character*(*) STR
1623:     character*26 UPC, LWC
1624:     data UPC /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
1625:     data LWC /'abcdefghijklmnopqrstuvwxyz'/

```

```

1626: C
1627:     do 100 I = 1, LEN(STR)
1628:         K      = INDEX(LWC,STR(I:I))
1629:         if ( K.ne.0 ) STR(I:I) = UPC(K:K)
1630:     100 continue
1631:     return
1632: end
1633: C
1634: C
1635: C
1636: C
1637:     subroutine CCOMP( OUTCH, LN,      INCH, NN,      IFL )
1638: C
1639: C     GMVP/MVP UTILITY
1640: C
1641: C=====
1642: C     PURPOSE: 'COMPRESS' CHARACTER STRING INCH BY REMOVING BLANK
1643: C             INTO CHARACTER STRING 'OUTCH' UPTO LN CHARACTERS.
1644: C             ('OUTCH' & 'INCH' CAN OVERLAP IN MEMORY BUT
1645: C             THE STARTING ADDRESS OF 'OUTCH' MUST PRECEED OR BE SAME AS
1646: C             THAT OF 'INCH'. )
1647: C
1648: C             IFL : FLAG
1649: C                 0 = NORMAL END
1650: C                 1 = NUMBER OF NON-BLANK CHARACTERS IN INCH EXCEEDS LN.
1651: C
1652: C     HISTORY: PROGRAMMED BY M.SASAKI ( 18 MAR 1992 )
1653: C=====
1654: C
1655:     character*(*) OUTCH, INCH
1656:     character*1 CH1
1657: C
1658:     IFL      = 0
1659:     LL       = LEN(INCH)
1660:     NN       = 0
1661: C
1662:     do 100 I = 1, LL
1663:         if ( INCH(I:I).ne.' ' ) then
1664:             NN      = NN + 1
1665:             if ( NN.le.LN ) then
1666:                 CH1  = INCH(I:I)
1667:                 OUTCH(NN:NN) = CH1
1668:             end if
1669:         end if
1670:     100 continue
1671:     if ( NN.lt.LN ) OUTCH(NN+1:LN) = ' '
1672:     if ( NN.gt.LN ) IFL = 1
1673:     return
1674: end
1675: C
1676: C
1677: C
1678:     subroutine NOBLNK( LINE, IS,      IE,      NL )
1679: C
1680: C     JAERI MONTE CARLO CODE UTILITY
1681: C
1682: C=====
1683: C     PURPOSE: FIND NONBLANK CHARACTER STRING IN 'LINE'.
1684: C
1685: C     SEARCH LINE(IS:NL)
1686: C     AND RETURNS STARTING POSITION AND END POSITION OF NON-BLANK STRING.
1687: C     IN IS & IE RESPECTIVELY.
1688: C
1689: C     CALLED IN: ZONEIN ETC.
1690: C=====

```

src/tools/fat.f

```
1691:
1692:     character*(*) LINE
1693:     if ( IS.gt.NL ) then
1694:         IE      = IS
1695:         return
1696:     end if
1697:     ISS      = 0
1698:     do 100 I = IS, NL
1699:         if ( LINE(I:I).ne.' ' .and.ISS.eq.0 ) ISS      = I
1700:         if ( LINE(I:I).eq.' ' .and.ISS.ne.0 ) go to 110
1701: 100 continue
1702:     if ( ISS.eq.0 ) then
1703:         IS      = I
1704:         IE      = I
1705:         return
1706:     end if
1707: 110 IS      = ISS
1708:     IE      = I - 1
1709:     return
1710: end
```

src/tools/fbcheck.c

```

1: /*****
2:  *
3:  * Fotran binary file checker :
4:  *
5:  *   First version : 24 Jan 1995 ( M.Sasaki (J.R.I.) )
6:  *
7:  *****/
8:
9: #include <stdio.h>
10: #include <stdlib.h>
11: #include <unistd.h>
12: #include <fcntl.h>
13: #include <string.h>
14: #include <ctype.h>
15:
16: unsigned int swap_bytes ( ii )
17: unsigned int ii ;
18: {
19:     return ( ( ii & 0x000000ff ) << 24 ) |
20:            ( ( ii & 0x0000ff00 ) << 8 ) |
21:            ( ( ii & 0x00ff0000 ) >> 8 ) |
22:            ( ( ii & 0xff000000 ) >> 24 ) ;
23: }
24:
25:
26: main( argc, argv )
27: int argc ;
28: char * argv[] ;
29: {
30:     int filed ;
31:     char *file_name ;
32:
33:     unsigned int check_string , c2 ;
34:     unsigned char *cp;
35:     int endian ; /*** (sizeof(int)) * +-1 ;
36:                    +1 big-endian ; -1 little-endian **/
37:
38:     int record_count ;
39:     int record_length, record_length2 ;
40:     int nbytes;
41:     int i, nn, nn2 ;
42:
43: /*#define MAX_READ_BUF 240000
44: */
45: #define MAX_READ_BUF 20000
46:
47:     char read_buffer[ MAX_READ_BUF ] ;
48:
49: /*****
50:  Check endian of current machine
51:  *****/
52:
53:     endian = sizeof(int) ;
54:     if( endian == 4 ) {
55:         check_string = (('a'*256+'b')*256+'c')*256+'d';
56:         if ( memcmp("abcd", &check_string, (size_t)endian)) {
57:             endian *= -1 ;
58:         }
59:         c2 = swap_bytes(check_string) ;
60:         printf(" %4.4s %4.4s\n", &check_string, &c2);
61:
62:     } else if( endian == 8 ) {
63:         check_string =
64:             ((((((('a'*256+'b')*256+'c')*256+'d')*256+'e')*256+'f')*256+'g')*256+'
h';

65:         if ( memcmp("abcdefgh", &check_string, (size_t)endian)) {
66:             endian *= -1 ;
67:         }
68:     }
69:     printf(" Endian : %d\n\n",endian);
70:
71:     if( endian != 4 && endian != -4 ) {
72:         printf("This program does not work in architecture having int whose si
ze is not 4.\n");
73:         exit(1) ;
74:     }
75:
76: /*****
77:  Check arguments
78:  *****/
79:
80:     if( argc <= 1 )
81:     {
82:         printf("Usage: %s inputfile\n", argv[0] ) ;
83:         exit(1) ;
84:     }
85:     /**** get file name ****/
86:
87:     file_name = argv[1] ;
88:     printf( "File: %s\n\n",file_name);
89:
90:     if( ( filed = open( file_name, O_RDONLY ) ) == -1 )
91:     {
92:         fprintf( stderr, "XXX Failed to open file < %s>\n",
93:                 file_name ) ;
94:         exit(1) ;
95:     }
96:
97: /*****
98:  * Structure of Fortran binary file assumed:
99:  *
100:  *   (a) Record length in 4-byte integer
101:  *   (b) data in record
102:  *   (c) Record length in 4-byte integer
103:  *
104:  *   Data (a),(b),(c) is repeated for all records.
105:  *****/
106:
107: #define WORD_SIZE (sizeof(int))
108:
109:     record_count = 0 ;
110:
111:     while( ( nbytes = read(filed,(void *)&record_length, sizeof(int)) ) > 0 )
112:     {
113:         ++record_count ;
114:         printf( " * record %d : length %d bytes, %d words\n",
115:                 record_count, record_length,
116:                 (int)( record_length / sizeof(int)) );
117:
118:         /****
119:         if( record_length > MAX_READ_BUF )
120:         {
121:             fprintf(stderr,"Too long record!! %d\n",
122:                     record_length );
123:         }
124:         ****/
125:
126:         nbytes = 0 ;
127:         for ( i=0 ; i < record_length ; i+=MAX_READ_BUF ) {
128:             nn2 = i + MAX_READ_BUF < record_length ?

```

src/tools/fbcheck.c

```
129:             MAX_READ_BUF : record_length - i ;
130:
131:     nbytes += read( filed, read_buffer, nn2 ) ;
132:
133:     /*****
134:     Print upto 48 printable characters
135:     *****/
136:     if ( i == 0 ) {
137:         int jj;
138:         nn = 0 ;
139:         for ( jj=0 ; jj < 48 && jj < nbytes ; jj++ ) {
140:             if ( isprint( read_buffer[jj] ) ) {
141:                 printf("%c",read_buffer[jj]) ;
142:                 nn++ ;
143:             } else {
144:                 break ;
145:             }
146:         }
147:         if(nn) {
148:             printf(" <= printable character on record head\n") ;
149:         }
150:     }
151: }
152:
153: if ( nbytes != record_length )
154: {
155:     fprintf( stderr,
156:         "XXX record length invalid: expected %d read %d\n",
157:         record_length, nbytes ) ;
158:     exit(2) ;
159: }
160:
161: nbytes = read( filed, (void *)&record_length2, sizeof(int));
162: if( record_length2 != record_length )
163: {
164:     fprintf( stderr,
165:         "XXX invalid record length field: header %d tail %d\n",
166:         record_length, record_length2 ) ;
167:     exit(2) ;
168: }
169: }
170: printf(" -- Number of records : %d\n", record_count ) ;
171: }
```


src/tools/fsfreader.f

```

1: c
2: c ... Simple program to read MVP/GMVP fission source file ...
3: c
4: c   written by Y.Nagaya (2009 Jul 22)
5: c
6: c   program FSFREADER
7: c
8: c   character HEADER*32, TITLE1*72, TITLE2*72, DATE*64
9: c   character TAG*32
10: c   integer   NUSE
11: c
12: c   parameter(NSMAX=10000)
13: c   real*8     XXXF(NSMAX), YYF(NSMAX), ZZZF(NSMAX)
14: c
15: c ... I/O units ...
16: c
17: c   NSTDI = 5
18: c   NSTDO = 6
19: c   NFSCF = 1
20: c   NTXTO = 6
21: c
22: c ... open file ...
23: c
24: c   open(NFSCF, file='mvptest.fissionfile.srcout.fissrc',
25: c     & status='old',access='sequential',form='unformatted')
26: c
27: c ... read binary file ...
28: c
29: c   read(NFSCF) HEADER, NHREC
30: c   write(NTXTO, '(a,i2)') HEADER, NHREC
31: c
32: c   read(NFSCF) HEADER
33: c   write(NTXTO, '(a)') HEADER
34: c
35: c   read(NFSCF) TITLE1, TITLE2
36: c   write(NTXTO, '(a/a)') TITLE1, TITLE2
37: c
38: c   read(NFSCF) DATE
39: c   write(NTXTO, '(a)') DATE
40: c
41: c   ----- Number of source neutrons -----
42: c
43: c       0-----1-----2-----3--
44: c   TAG   = 'SIZES'
45: c       0-----1-----2-----3--
46: c   call SRTAG(NFSCF, TAG, IRC)
47: c
48: c   if (IRC.eq.0) then
49: c     write(NSTDO, *) 'XXX NO SIZE DATA !! STOP '
50: c     stop
51: c   end if
52: c
53: c   read(NFSCF) TAG
54: c   read(NFSCF) NUSE
55: c   write(NTXTO, *) TAG, NUSE
56: c
57: c   if (NUSE.gt.NSMAX) then
58: c     write(NSTDO, *) 'XXX EXTEND NSMAX !! STOP '
59: c     stop
60: c   end if
61: c
62: c   ----- X-COORDINATE OF SOURCE POINTS -----
63: c
64: c       0-----1-----2-----3--
65: c   TAG   = 'X'

```

```

66: c       0-----1-----2-----3--
67: c   call SRTAG(NFSCF, TAG, IRC)
68: c
69: c   if (IRC.eq.0) then
70: c     write(NSTDO, *) 'XXX NO X DATA !! STOP '
71: c     stop
72: c   end if
73: c
74: c   read(NFSCF) TAG
75: c   read(NFSCF) (XXXF(I),I=1,NUSE)
76: c
77: c   ----- Y-COORDINATE OF SOURCE POINTS -----
78: c
79: c       0-----1-----2-----3--
80: c   TAG   = 'Y'
81: c       0-----1-----2-----3--
82: c   call SRTAG(NFSCF, TAG, IRC)
83: c
84: c   if (IRC.eq.0) then
85: c     write(NSTDO, *) 'XXX NO Y DATA !! STOP '
86: c     stop
87: c   end if
88: c
89: c   read(NFSCF) TAG
90: c   read(NFSCF) (YYF(I),I=1,NUSE)
91: c
92: c   ----- Z-COORDINATE OF SOURCE POINTS -----
93: c
94: c       0-----1-----2-----3--
95: c   TAG   = 'Z'
96: c       0-----1-----2-----3--
97: c   call SRTAG(NFSCF, TAG, IRC)
98: c
99: c   if (IRC.eq.0) then
100: c     write(NSTDO, *) 'XXX NO Z DATA !! STOP '
101: c     stop
102: c   end if
103: c
104: c   read(NFSCF) TAG
105: c   read(NFSCF) (ZZZF(I),I=1,NUSE)
106: c
107: c ... Dump source coordinates ...
108: c
109: c   write(NTXTO, *) ' X, Y, Z'
110: c   do I = 1, NUSE
111: c     write(NTXTO, '(3f13.5)') XXXF(I), YYF(I), ZZZF(I)
112: c   end do
113: c
114: c ... close file ...
115: c
116: c   close(unit=NFSCF, status='keep')
117: c
118: c   stop
119: c   end
120: c
121: c ===== SEARCH RECORD TAG =====
122: c
123: c   subroutine SRTAG( NFILE, TAG, IRC )
124: c
125: c     character*32 TAG
126: c     character*32 STAG
127: c
128: c     IRC = 0 : RECORD WITH TAG NOT FOUND
129: c     IRC = 1 : RECORD WITH TAG IS FOUND
130: c

```

src/tools/fsfreader.f

```
131:      IRC   = 0
132: 1000 read( NFILE, END=999 ) STAG
133:      if ( STAG.eq.TAG ) then
134:          IRC = 1
135:          backspace NFILE
136:          return
137:      end if
138:      go to 1000
139: c
140: 999 return
141: end
```

SAFE

src/tools/gmvplbcv.f

```

1: C
2: C  GMVP/KENO/ANISN cross section library form converter
3: C
4: C      BINARY <---> TEXT
5: C
6: C=====
7: CCCC/#IF SYSTEM(HP*)
8: C/#IF FLOATHANDLER( ONSTMT )
9: C
10: C ---- TO PREVENT UNDERFLOW ABORT in trapping mode (+T option)---
11: C
12: *      ON REAL*4 UNDERFLOW CALL UNDERF4
13: *      ON REAL*8 UNDERFLOW CALL UNDERF8
14: C/#ENDIF
15: C
16: write(6,*) '*****'
17: write(6,*) '***  GMVP/KENO/ANISN library form conversion  ***'
18: write(6,*) '*****'
19: C
20: 100 write(6,*) ' conversion type : '
21: write(6,*) '      1. binary --> text'
22: write(6,*) '      2. text ---> binary'
23: write(6,*) '      INPUT > '
24: read(5,*) ITYPE
25: if ( ITYPE.eq.1 ) then
26:   call LB2TXT
27: else if ( ITYPE.eq.2 ) then
28:   call TXT2LB
29: else
30:   write(6,*) 'XXX conversion type ', ITYPE, ' is invalid.'
31:   go to 100
32: end if
33: end
34: C
35: C
36: C
37: CCCCC/#IF SYSTEM(HP*)
38: C/#IF FLOATHANDLER( ONSTMT )
39: C
40: C ... underflow handling ...
41: C
42: *      subroutine underf4( r4 )
43: *      real    r4
44: *      r4 = 0.0
45: *      return
46: *      end
47: C
48: C
49: *      subroutine underf8( r8 )
50: *      real*8 r8
51: *      r8 = 0.0d0
52: *      return
53: *      end
54: C/#ENDIF
55: C
56: C
57: C
58:   subroutine TXT2LB
59: C
60: C  FILE FORM CONVERTER FOR KENO/ANISN LIBRARY : TEXT TO BINARY
61: C
62: C  ASSUMES TEXT FILE MADE WITH LIB2TEXT PROGRAM.
63: C
64: C
65: C  PROGRAMMED BY M.SASAKI   DEC 12 1991

```

```

66: C                                     MAR 19 1993
67: C
68: C      INPUT TEXT LIBRARY   : UNIT 15
69: C      OUTPUT BINARY LIBRARY : UNIT 10
70: C
71: C
72: C  -- LIMIT PARAMETERS ---
73: C
74: C      NG : NUMBER OF ENERGY GROUPS
75: C      NDS : MAX. OF DOWNSCATTERS
76: C
77: C      MAXIHM : MAX. OF TABLE LENGTH FOR ANISN
78: C
79: C      parameter( NG   = 137 )
80: C      parameter( NDS  = 137 )
81: C      parameter( MAXIHM = 300 )
82: C
83: C      character*32 TYPE
84: C
85: C  ---- DATA FOR KENO ---
86: C
87: C      common /KENO/      E(NG+1),      F(NG),  T(NG),  AC(NG),
88: C      &      TSP(NDS,NG),      FIST(NG),      TSP1(NDS,NG),
89: C      &      DLETH(NG+1),      XST(18)
90: C
91: C  ---- DATA FOR GMVP/ANISN ( PL ) ---
92: C
93: C      common /ANISN/      ANSIG(MAXIHM,NG)
94: C
95: C      character*72 INFILE, OTFILE
96: C-----
97: C      write(6,*)' ====='
98: C      write(6,*)' == GMVP/KENO/ANISN Library Converter : Text to Bin. =='
99: C      write(6,*)' ====='
100: C      write(6,*)' == Programmed by M.SASAKI Dec 1991,Mar 1993 =='
101: C      write(6,*)' ====='
102: C      write(6,*)' ====='
103: C      write(6,*)' == Input: unit 15 ====='
104: C      write(6,*)' == Output: unit 10 ====='
105: C      write(6,*)' ====='
106: C      LD      = 10
107: C      LT      = 15
108: C
109: C----- READ CODE TYPE -----
110: C
111: C1000 WRITE(6,*)
112: C      & ' CODE TYPE (1 : KENO, 2 : GMVP/ANISN(PL) 3: ANISN(MGCL-MAIL)'
113: C      write(6,*)
114: C      & '      4 : DDX ) -> '
115: C      READ(5,*) ICODE
116: C      WRITE(6,*) ICODE
117: C
118: C----- READ FILENAMES -----
119: C
120: C      write(6,*) ' == INPUT (TEXT) LIBRARY --> '
121: C      read(5,'(A72)') INFILE
122: C      write(6,'(1X,A72)') INFILE
123: C
124: C      write(6,*) ' == OUTPUT (BINARY) LIBRARY --> '
125: C      read(5,'(A72)') OTFILE
126: C      write(6,'(1X,A72)') OTFILE
127: C
128: C      open( LD, file =OTFILE, status ='UNKNOWN', form ='UNFORMATTED' )
129: C
130: C/#IF READONLY(ACTION)

```

src/tools/gmvplbcv.f

```

131: *      OPEN(LT,FILE=INFILE, STATUS='OLD',FORM='FORMATTED',
132: *      &      ACTION='READ')
133: C/#ELSEIF READONLY(DEC)
134: *      OPEN(LT,FILE=INFILE, STATUS='OLD',FORM='FORMATTED', READONLY)
135: C/#ELSEIF READONLY(MODE)
136: *      OPEN(LT,FILE=INFILE, STATUS='OLD',FORM='FORMATTED', MODE='READ')
137: C/#ELSE
138:      open( LT, file =INFILE, status = 'OLD', form = 'FORMATTED' )
139: C/#ENDIF
140: C
141: C
142: C ... read cross section type ...
143: C
144:      read(LT,'(a)') TYPE
145: C
146:      write(6,*) ' === data type : <', TYPE, '>'
147: C
148:      if ( TYPE.eq.'KENO' ) then
149:          ICODE = 1
150:      else if ( TYPE.eq.'ANISN' ) then
151:          ICODE = 2
152:      else if ( TYPE.eq.'ANISN-MAIL' ) then
153:          ICODE = 3
154:      else if ( TYPE.eq.'DDX' ) then
155:          ICODE = 4
156:      else
157:          write(6,*) 'XXX invalid code type !! : <', TYPE, '>'
158:          stop 888
159:      end if
160: C
161:      if ( ICODE.eq.1 ) then
162: C
163: C
164: C ===== KENO =====
165: C
166:          IFORM = 0
167: C
168: Cccccccc READ(LT,'(3I10)') NNUC,NGP,NDS1
169:          read(LT,'(4I10)') NNUC, NGP, NDS1, IFORM
170: C
171: C ... check size limit ...
172: C
173:          if ( NGP.gt.NG .or. NDS1.gt.NDS ) then
174:              write(6,*) 'XXX Too many energy group or downscatter.'
175:              write(6,*) 'Energy group ', NGP, ' (program limit ', NG, ')'
176:              write(6,*) 'Downscatter ', NDS1, ' (program limit ', NDS,
177:              &          ' )'
178:              stop 999
179:          end if
180: C
181:          if ( IFORM.eq.0 ) then
182:              read(LT,'(1P,6E12.5)') (E(I),I=1,NGP+1),
183:              &          (DLETH(I),I=1,NGP+1)
184:          else
185:              read(LT,*) (E(I),I=1,NGP+1), (DLETH(I),I=1,NGP+1)
186:          end if
187: C
188:          write(LD) NNUC, NGP, NDS1, (E(I),I=1,NGP+1),
189:          &          (DLETH(I),I=1,NGP+1)
190: C
191: C
192:          do 120 II = 1, NNUC
193: C
194:              read(LT,'(18A4,I8)') (XST(L),L=1,18)
195:              read(LT,'(I10)') IDENT

```

```

196:          if ( IFORM.eq.0 ) then
197:              read(LT,'(1P,6E12.5)') (AC(L),F(L),T(L),FIST(L),L=1,NGP),
198:              &          ((TSP(K,J),K=1,NDS1),J=1,NGP),
199:              &          ((TSP1(K,J),K=1,NDS1),J=1,NGP)
200:          else
201:              read(LT,*) (AC(L),L=1,NGP), (F(L),L=1,NGP),
202:              &          (T(L),L=1,NGP), (FIST(L),L=1,NGP)
203: C
204:          do 100 J = 1, NGP
205:              read(LT,*) (TSP(K,J),I=K,NDS1)
206:          100 continue
207:          do 110 J = 1, NGP
208:              read(LT,*) (TSP1(K,J),I=K,NDS1)
209:          110 continue
210:          end if
211: C
212:          write(6,7000) IDENT, (XST(LK),LK=1,18)
213: C
214:          write(LD) (XST(L),L=1,18), IDENT,
215:          &          (AC(L),F(L),T(L),FIST(L),L=1,NGP),
216:          &          ((TSP(K,J),K=1,NDS1),J=1,NGP),
217:          &          ((TSP1(K,J),K=1,NDS1),J=1,NGP)
218: C
219: C
220:          7000 format(' NUCLEIDE = ',I10/3X,18A4)
221:          120 continue
222: C
223:          stop
224: C
225: C
226: C
227: C ===== GMVP / ANISN =====
228: C
229:          else if ( ICODE.eq.2 .or. ICODE.eq.3 ) then
230: C
231: CCCCC
232: Ccccc READ(LT,'(4I10)') NGP,IHM,IHT,IHS
233:          IFORM = 0
234: C
235:          read(LT,'(5I10)') NGP, IHM, IHT, IHS, IFORM
236: C
237:          write(6,*) ' NUMBER OF ENERGY GROUPS -> ', NGP
238:          write(6,*) ' IHM (TABLE LENGTH) -> ', IHM
239:          write(6,*) ' IHT (POSITION OF SIGT ) -> ', IHT
240:          write(6,*) ' IHS (POSITION OF SIG-GG ) -> ', IHS
241: C
242: C ... check size limit ...
243: C
244:          if ( NGP.gt.NG .or. IHM.gt.MAXIHM ) then
245:              write(6,*) 'XXX Too many energy group or table length.'
246:              write(6,*) 'Energy group ', NGP, ' (program limit ', NG, ')'
247:              write(6,*) 'Table leng. ', IHM, ' (program limit ', MAXIHM,
248:              &          ' )'
249:              stop 999
250:          end if
251: C
252: C
253: C
254:          130 read(LT,fmt='(4I10)',end=150) N1, N2, N3, N4
255:          read(LT,'(12A4)') (XST(L),L=1,12)
256: C
257:          write(LD) N1, N2, N3, N4, (XST(L),L=1,12)
258: C
259:          write(6,'(2X,4I10/1X,12A4)') N1, N2, N3, N4, (XST(L),L=1,12)
260: C

```

src/tools/gmvplbcv.f

```

261:      do 140 I = 1, NGP
262: C
263: C          ... written in fixed form
264: C
265:      if ( IFORM.eq.0 ) then
266:          read(LT,'(1P,6E12.5)') (ANSIG(J,I),J=1,IHM)
267:      else
268: C
269: C          ... written in free form
270: C
271:          read(LT,*) (ANSIG(J,I),J=1,IHM)
272: C
273:      end if
274:      if ( ICODE.eq.3 ) write(LD,*) (ANSIG(J,I),J=1,IHM)
275: 140 continue
276:      if ( ICODE.eq.2 ) write(LD,*) ((ANSIG(J,I),J=1,IHM),I=1,NGP)
277: C
278:      go to 130
279: C
280: 150 write(6,*) ' FILE END ! COMPLETED '
281: C
282:      stop
283: C
284:      else if ( ICODE.eq.4 ) then
285: C
286:          call DDXCV( 'TEXT-TO-BINARY', LD, LT )
287:      end if
288: C
289: 160 write(6,*) ' INVALID CODE TYPE : ICODE ', ICODE
290:      stop
291:      end
292: C
293: C
294: C
295: C
296: C
297:      subroutine LB2TXT
298: C
299: C  FILE FORM CONVERTER FOR KENO/ANISN LIBRARY :  BINARY TO TEXT
300: C
301: C  MADE BY M.SASAKI   DEC 12 1991
302: C                   MAR 19 1993
303: C
304: C  INPUT BINARY LIBRARY :  UNIT 10
305: C  OUTPUT TEXT LIBRARY  :  UNIT 11
306: C
307: C
308: C  -- LIMIT PARAMETERS ---
309: C
310: C  NG : NUMBER OF ENERGY GROUPS
311: C  NDS : MAX. OF DOWNSCATTERS
312: C
313: C  MAXIHM : MAX. OF TABLE LENGTH FOR ANISN
314: C
315:      parameter( NG   = 137 )
316:      parameter( NDS   = 137 )
317:      parameter( MAXIHM = 300 )
318: C
319: C ---- DATA FOR KENO --
320: C
321:      common /KENO/      E(NG+1),      F(NG),  T(NG),  AC(NG),
322:      &      TSP(NDS,NG),  FIST(NG),      TSP1(NDS,NG),
323:      &      DLETH(NG+1),  XST(18)
324: C
325: C ---- DATA FOR ANISN --

```

```

326: C
327:      common /ANISN/      ANSIG(MAXIHM,NG)
328: C
329:      character*72 INFILE, OTFILE
330: C
331:      character*32 TYPE
332: C
333: C
334:      parameter( NDIGIT   = 8 )
335: C
336: C-----
337:      write(6,*) '=====
338:      write(6,*) '== GMVP/KENO/ANISN LIBRARY CONVERTOR : BIN. TO TEXT ==
339:      write(6,*) '=====
340:      write(6,*) '== PROGRAMMED BY M.SASAKI   DEC 1991,MAR 1993 ==
341:      write(6,*) '=====
342:      write(6,*) '=====
343:      write(6,*) '== INPUT: UNIT 10 ==
344:      write(6,*) '== OUTPUT: UNIT 15 ==
345:      write(6,*) '=====
346:      LD      = 10
347:      LT      = 15
348: C
349: C----- READ CODE TYPE -----
350: C
351:      100 write(6,*)
352:      &      ' CODE TYPE (1: KENO, 2: GMVP(ANISN), 3: ANISN(MGCL-MAIL) '
353:      write(6,*) '              4: DDX ) -> '
354:      read(5,*) ICODE
355:      write(6,*) ICODE
356: C
357:      if ( ICODE.eq.1 ) then
358:          TYPE = 'KENO'
359:      else if ( ICODE.eq.2 ) then
360:          TYPE = 'ANISN'
361:      else if ( ICODE.eq.3 ) then
362:          TYPE = 'ANISN-MAIL'
363:      else if ( ICODE.eq.4 ) then
364:          TYPE = 'DDX'
365:      else
366:          write(6,*) 'XXX invalid code type !! : ', ICODE
367:          stop 888
368:      end if
369: C
370: C
371: C----- READ FILENAMES -----
372: C
373:      write(6,*) ' == INPUT (BINARY) LIBRARY --> '
374:      read(5,'(A72)') INFILE
375:      write(6,'(1X,A72)') INFILE
376: C
377:      write(6,*) ' == OUTPUT (TEXT) LIBRARY --> '
378:      read(5,'(A72)') OTFILE
379:      write(6,'(1X,A72)') OTFILE
380: C
381: C/#IF READONLY(ACTION)
382: *      OPEN(LD,FILE=INFILE, STATUS='OLD',FORM='UNFORMATTED',
383: *      &      ACTION='READ' )
384: C/#ELSEIF READONLY(READONLY)
385: *      OPEN(LD,FILE=INFILE, STATUS='OLD',FORM='UNFORMATTED',READONLY )
386: C/#ELSEIF READONLY(MODE)
387: *      OPEN(LD,FILE=INFILE, STATUS='OLD',FORM='UNFORMATTED',MODE='READ' )
388: C/#ELSE
389:      open( LD, file =INFILE, status ='OLD', form ='UNFORMATTED' )
390: C/#ENDIF

```

src/tools/gmvplbcv.f

```

391:
392:      open( LT, file =OTFILE, status = 'UNKNOWN', form = 'FORMATTED' )
393: C
394: C ... output cross section type on the first output line ...
395: C
396:      write(LT,'(a)') TYPE
397: C
398: C ... open I/O unit LT as a freeform output unit ...
399: C      (80 column output)
400: C
401:      call POPEN( LT, 80 )
402: C
403:      IFORM = 1
404: C
405: C ... iform : output form flag of cross section data (ANSSIG) ...
406: C
407: C      ... iform = 1 ( free format )
408: C      ... iform = 0 ( fixed format ) : to read data processed by
409: C      older version
410: C
411:      if ( ICODE.eq.1 ) then
412: C
413: C -----
414: C ===== KENO =====
415: C -----
416: C -----
417: C
418: C ... read only size parameters at first for size limit check ...
419: C
420: C 110      read(LD) NNUC, NGP, NDS1
421: C
422: C      ... check size limit ...
423: C
424:      if ( NGP.gt.NG .or. NDS1.gt.NDS ) then
425:          write(6,*) 'XXX Too many energy group or downscatter.'
426:          write(6,*) 'Energy group ', NGP, ' (program limit ', NG, ') '
427:          write(6,*) 'Downscatter ', NDS1, ' (program limit ', NDS,
428:          &          ' ) '
429:          stop 999
430:      end if
431: C
432:      backspace LD
433: C
434:      read(LD) NNUC, NGP, NDS1, (E(I),I=1,NGP+1),
435:      &      (DLETH(I),I=1,NGP+1)
436: C
437: C ... output iform on this record
438: C
439: Ccccccc WRITE(LT,'(3I10)') NNUC,NGP,NDS1
440:      write(LT,'(4I10)') NNUC, NGP, NDS1, IFORM
441: C
442: C
443:      if ( IFORM.eq.0 ) then
444:          write(LT,'(1P,6E12.5)') (E(I),I=1,NGP+1),
445:          &      (DLETH(I),I=1,NGP+1)
446:      else
447:          call EWRITE( E, NGP+1, NDIGIT, IWST )
448:          call EWRITE( DLETH, NGP+1, NDIGIT, IWST )
449:          call NXTLIN( IWST )
450:      end if
451: C
452:      do 140 II = 1, NNUC
453: C
454: C
455: C AC : absorption x-sec

```

```

456: C      F : nu * fission x-sec
457: C      T : total x-sec
458: C      FIST : fission spectrum
459: C
460: C
461:      read(LD) (XST(L),L=1,18), IDENT,
462:      &      (AC(L),F(L),T(L),FIST(L),L=1,NGP),
463:      &      ((TSP(K,J),K=1,NDS1),J=1,NGP),
464:      &      ((TSP1(K,J),K=1,NDS1),J=1,NGP)
465: C
466:      write(6,7000) IDENT, (XST(LK),LK=1,18)
467: C 7000      format(' NUCLIDE = ',I10/3X,18A4)
468: C
469:      write(LT,'(18A4,I8)') (XST(L),L=1,18)
470:      write(LT,'(I10)') IDENT
471: C
472:      if ( IFORM.eq.0 ) then
473: C
474:          write(LT,'(1P,6E12.5)')
475:          &      (AC(L),F(L),T(L),FIST(L),L=1,NGP),
476:          &      ((TSP(K,J),K=1,NDS1),J=1,NGP),
477:          &      ((TSP1(K,J),K=1,NDS1),J=1,NGP)
478: C
479:      else
480:          call EWRITE( AC, NGP, NDIGIT, IWST )
481:          call EWRITE( F, NGP, NDIGIT, IWST )
482:          call EWRITE( T, NGP, NDIGIT, IWST )
483:          call EWRITE( FIST, NGP, NDIGIT, IWST )
484:          call NXTLIN( IWST )
485: C
486:          do 120 J = 1, NGP
487:              call EWRITE( TSP(1,J), NDS1, NDIGIT, IWST )
488:              call NXTLIN( IWST )
489: C 120          continue
490: C
491:          do 130 J = 1, NGP
492:              call EWRITE( TSP1(1,J), NDS1, NDIGIT, IWST )
493:              call NXTLIN( IWST )
494: C 130          continue
495:          end if
496: C
497: C 140      continue
498: C
499:      stop
500: C
501: C -----
502: C ===== GMVP/ANISN =====
503: C -----
504: C
505:      else if ( ICODE.eq.2 .or. ICODE.eq.3 ) then
506: C
507:          write(6,*) ' NUMBER OF ENERGY GROUPS -> '
508:          read(5,*) NGP
509:          write(6,*) ' IHM (TABLE LENGTH) -> '
510:          read(5,*) IHM
511:          write(6,*) ' IHT (POSITION OF SIGT ) -> '
512:          read(5,*) IHT
513:          write(6,*) ' IHS (POSITION OF SIG-GG ) -> '
514:          read(5,*) IHS
515: C
516: C
517: Ccccccc WRITE(LT,'(4I10)') NGP,IHM,IHT,IHS
518:          write(LT,'(5I10)') NGP, IHM, IHT, IHS, IFORM
519: C
520: C      ... check size limit ...

```

src/tools/gmvplbcv.f

```

521: C
522:   if ( NGP.gt.NG .or. IHM.gt.MAXIHM ) then
523:     write(6,*) 'XXX Too many energy group or table length.'
524:     write(6,*) 'Energy group ', NGP, ' (program limit ', NG, ' )'
525:     write(6,*) 'Table leng. ', IHM, ' (program limit ', MAXIHM,
526:       &      ' )'
527:     stop 999
528:   end if
529: C
530: C
531: 150   read(LD,end =190) N1, N2, N3, N4, (XST(L),L=1,12)
532: C
533:     write(LT,'(4I10)') N1, N2, N3, N4
534:     write(LT,'(12A4)') (XST(L),L=1,12)
535: C
536:     write(6,'(2X,4I10,1X,12A4)') N1, N2, N3, N4, (XST(L),L=1,12)
537: C
538:   if ( ICODE.eq.2 ) then
539:     read(LD) ((ANSIG(J,I),J=1,IHM),I=1,NGP)
540:   else
541:     do 160 I = 1, NGP
542:       read(LD) (ANSIG(J,I),J=1,IHM)
543: 160   continue
544:   end if
545: C
546: C
547: C
548:   if ( IFORM.eq.0 ) then
549:     do 170 I = 1, NGP
550:       write(LT,'(1P,6E12.5)') (ANSIG(J,I),J=1,IHM)
551: 170   continue
552:   else
553: C
554: C   ... output cross sections in free form ...
555: C
556:     do 180 I = 1, NGP
557:       call EWRITE( ANSIG(1,I), IHM, NDIGIT, IWST )
558:       if ( IWST.ne.0 ) then
559:         write(6,*) 'XXX Data output error : code of EWRITE=',
560:           &      IWST
561:         stop 888
562:       end if
563:       call NXTLIN( IWST )
564: 180   continue
565:     end if
566: C
567:     go to 150
568: C
569: 190   write(6,*) ' FILE END ! '
570: C
571:     stop
572: C
573: C-----
574: C ===== DDX =====
575: C-----
576: C
577: C
578:   else if ( ICODE.eq.4 ) then
579: C
580:     call DDXCV( 'BINARY-TO-TEXT', LD, LT )
581:   end if
582: C
583: 200 write(6,*) ' INVALID CODE TYPE : ICODE ', ICODE
584:   stop
585:   end

```

```

586: C
587: C
588: C
589: C ===== Conversion of DDX library =====
590: C
591: C
592: C
593:   subroutine DDXCV( MODE, LD, LT )
594: C
595:   character*(*) MODE
596: C
597:   parameter( MAXG = 125, MAXMU = 20 )
598: C
599:   parameter( NDIGIT = 8 )
600: C
601: C
602: Cccc common /ANISN/ MAXSD(maxg),SIG(5,maxg),P(maxmu,maxg)
603: C   ... transpose dimension of SIG(i,j)
604: C
605:   common /ANISN/ MAXSD(MAXG), SIG(MAXG,5), P(MAXMU,MAXG)
606:   character*4 TITLE(12)
607: C
608: C
609: C
610:   if ( MODE.eq.'TEXT-TO-BINARY' ) then
611: C
612: C
613: C
614: 100   read(LT,7000,end =160) MAXI, IDUM1, IDUM2, MATNO,
615:     &      (TITLE(J),J=1,12)
616: C
617:   if ( MAXI.gt.0 ) then
618:     IFORM = 0
619:   else
620:     IFORM = 1
621:     MAXI = ABS(MAXI)
622:   end if
623: C
624:   write(6,7000) MAXI, IDUM1, IDUM2, MATNO, (TITLE(J),J=1,12)
625: 7000 format(4I6,12A4)
626: C
627:   if ( MAXI.gt.MAXG ) then
628:     write(6,*) 'XXX NUMBER OF ENERGY GROUPS EXCEED LIMIT',
629:       &      ' (MAXG = ', MAXG, ' )'
630:     stop 999
631:   end if
632: C
633:   write(LD) MAXI, IDUM1, IDUM2, MATNO, (TITLE(J),J=1,12)
634: C
635:   read(LT,7020) (MAXSD(J),J=1,MAXI)
636: 7020 format(12I6)
637:   write(LD) (MAXSD(J),J=1,MAXI)
638: C
639:   if ( IFORM.eq.0 ) then
640:     read(LT,7040) ((SIG(J,I),I=1,5),J=1,MAXI)
641: 7040 format(10E12.5)
642:   else
643:     read(LT,*) ((SIG(J,I),J=1,MAXI),I=1,5)
644:   end if
645: C
646:   write(LD) ((SIG(J,I),I=1,5),J=1,MAXI)
647: C
648:   do 110 K = 1, MAXI
649:     if ( IFORM.eq.0 ) then
650:       read(LT,7040) ((P(I,J),I=1,MAXMU),J=1,MAXSD(K))

```

src/tools/gmvplbcv.f

```

651:         else
652:         read(LT,*) ((P(I,J),I=1,MAXMU),J=1,MAXSD(K))
653:         end if
654:         write(LD) ((P(I,J),I=1,MAXMU),J=1,MAXSD(K))
655: 110      continue
656: C
657:         go to 100
658: C
659: C
660: C
661:         else if ( MODE.eq.'BINARY-TO-TEXT' ) then
662: C
663: C
664: C
665: C
666: C ... open I/O unit LT as a freeform output unit ...
667: C      (80 column output)
668: C
669: C      call POPEN( LT, 80 )
670: C
671: C      IFORM = 1
672: C
673: C ... iform : output form flag of cross section data (ANSSIG) ...
674: C
675: C      ... iform = 1 ( free format )
676: C      ... iform = 0 ( fixed format ) to read data processed by
677: C                      older version
678: C
679: C
680: 120      read(LD,end =160) MAXI, IDUM1, IDUM2, MATNO, (TITLE(J),J=1,12)
681:         write(6,7000) MAXI, IDUM1, IDUM2, MATNO, (TITLE(J),J=1,12)
682: C1000      FORMAT(4I6,12A4)
683: C
684:         if ( MAXI.gt.MAXG ) then
685:             write(6,*) 'XXX NUMBER OF ENERGY GROUPS EXCEED LIMIT',
686: &             ' (MAXG = ', MAXG, ' )'
687:             stop 999
688:         end if
689: C
690: C
691: C ... output negated MAXI when iform==1 ....
692: C
693: C
694:         if ( IFORM.eq.0 ) then
695:             write(LT,7000) MAXI, IDUM1, IDUM2, MATNO, (TITLE(J),J=1,12)
696:         else
697:             write(LT,7000) - MAXI, IDUM1, IDUM2, MATNO,
698: &             (TITLE(J),J=1,12)
699:         end if
700: C
701:         read(LD) (MAXSD(J),J=1,MAXI)
702: C2000      FORMAT(12I6)
703: C
704:         write(LT,7020) (MAXSD(J),J=1,MAXI)
705: C
706:         read(LD) ((SIG(J,I),I=1,5),J=1,MAXI)
707: C3000      FORMAT(10E12.5)
708: C
709: C
710:         if ( IFORM.eq.0 ) then
711:             write(LT,7040) ((SIG(J,I),I=1,5),J=1,MAXI)
712:         else
713:             do 130 I = 1, 5
714:                 call EWRITE( SIG(1,I), MAXI, NDIGIT, IWST )
715: 130      continue

```

```

716:         call NXTLIN( IWST )
717:         end if
718: C
719: C
720:         do 150 K = 1, MAXI
721:             read(LD) ((P(I,J),I=1,MAXMU),J=1,MAXSD(K))
722:             if ( IFORM.eq.0 ) then
723:                 write(LT,7040) ((P(I,J),I=1,MAXMU),J=1,MAXSD(K))
724:             else
725:                 do 140 J = 1, MAXSD(K)
726:                     call EWRITE( P(1,J), MAXMU, NDIGIT, IWST )
727:                 140      continue
728:                 call NXTLIN( IWST )
729:             end if
730:         150      continue
731: C
732:         go to 120
733: C
734:         end if
735: C
736: 160      stop
737:         end
738: C
739: C
740: C --- 'FREE-FORM' OUTPUT ROUTINES ----
741: C
742: C      PROGRAMMED BY M.SASAKI      ( JAN 1992 )
743: C
744: C
745: C      OUTPUT DATA (FLOAT, INT., CHARACTER ) INPUTTABLE BY
746: C      USING 'CREAD' SUBROUTINE PACKAGE MADE BY M.SASAKI.
747: C
748: C === CONTENTS ===
749: C
750: C -----
751: C      SUBROUTINE      POPEN( IUOUT, NBUF )
752: C -----
753: C      ASSIGN OUTPUT I/O UNIT TO IUOUT ASSUMING
754: C      NBUF CHARACTERS IN ONE LINE.
755: C -----
756: C
757: C      SUBROUTINE      EWRITE( EDATA, N, IDIGIT, IWST )
758: C      SUBROUTINE      DWRITE( DDATA, N, IDIGIT, IWST )
759: C -----
760: C      OUTPUT N FLOATING POINT DATA (SINGLE/DOUBLE PRECISION
761: C      RESPECTIVELY) ON UNIT IUOUT IN IDIGIT OF PRECISION
762: C      IN DECIMAL.
763: C -----
764: C
765: C      SUBROUTINE      CWRITE( STR, N, IWST )
766: C      SUBROUTINE      SWRITE( STR, N, IWST )
767: C -----
768: C
769: C      OUTPUT CHARACTER STRING STR ( N CHARACTERS. IF N=0
770: C      LEN(STR) CHARACTERS ). 'SWRITE' ADDS TERMINATION CHARACTERS
771: C      READABLE BY SREAD ROUTINE OF CREAD PACKAGE.
772: C -----
773: C
774: C      SUBROUTINE      IWRITE( IDATA, N , IWST)
775: C -----
776: C      OUTPUT N INTEGER DATA IN ARRAY.
777: C -----
778: C
779: C      SUBROUTINE      NXTLIN( IWST )
780: C -----

```


src/tools/gmvplbcv.f

```

781: C      FLUSH LINE INPUT BUFFER & GO TO NEXT LINE.
782: C
783: C=====
784:      subroutine POPEN( IO,      NDG )
785: C
786:      common /CWRT1/      LINE
787:      character*256 LINE
788:      common /CWRT2/      IUOUT, NBUF,      IPOS
789: C
790:      IUOUT = IO
791:      IPOS = 1
792: C
793:      if ( NDG.gt.LEN(LINE) ) then
794:        write(6,*) ' = OUTPUT BUFFER TOO LONG ! ( > ', LEN(LINE), ' )='
795:        stop 666
796:      end if
797:      NBUF = NDG
798:      LINE = ' '
799: C
800:      return
801:      end
802: C
803: C=====
804:      subroutine CWRITE( STR,      NL,      IWST )
805: C
806:      character*(*) STR
807: C
808:      common /CWRT1/      LINE
809:      character*256 LINE
810:      common /CWRT2/      IUOUT, NBUF,      IPOS
811: C
812: C
813:      M = 1
814:      N = NL
815:      if ( NL.eq.0 ) then
816:        M = 0
817:        do 100 I = 1, LEN(STR)
818:          if ( STR(I:I).ne.' ' ) go to 110
819:        continue
820:      110 M = I
821:        do 120 I = LEN(STR), 1, -1
822:          if ( STR(I:I).ne.' ' ) go to 130
823:        continue
824:      130 N = I
825:      end if
826: C
827:      IP2 = IPOS + N - M
828:      if ( IP2.gt.NBUF ) then
829:        call NXTLIN( ICON )
830:        IP2 = IPOS + N - M
831:      end if
832:      LINE(IPOS:IP2) = STR(M:N)
833: C
834:      IPOS = IP2 + 2
835:      return
836:      end
837: C
838: C=====
839: C
840:      subroutine NXTLIN( IWST )
841: C
842:      common /CWRT1/      LINE
843:      character*256 LINE
844:      common /CWRT2/      IUOUT, NBUF,      IPOS
845: C

```

```

846:      write(IUOUT,'(A)') LINE(1:MIN(NBUF,IPOS-1))
847: CHECK *****
848: ***** WRITE(6,*) IPOS,':',LINE(1:NBUF)
849:      LINE(1:NBUF) = ' '
850:      IPOS = 1
851:      return
852:      end
853: C
854: C=====
855: C
856:      subroutine SWRITE( STR,      NL,      IWST )
857: C
858:      character*(*) STR
859: C
860:      common /CWRT1/      LINE
861:      character*256 LINE
862:      common /CWRT2/      IUOUT, NBUF,      IPOS
863: C
864:      character*512 STR2
865:      character*10 TM
866:      character*1 T
867:      data TM /' '&:;?#@%-'/
868: C
869:      N = NL
870:      if ( NL.eq.0 ) N = LEN(STR)
871: C
872:      if ( N.gt.LEN(STR2)-2 ) then
873:        write(6,*) ' XXX TOO LONG STRAING! (SWRITE) XXX'
874:        write(6,*) STR(1:N)
875:        stop 666
876:      end if
877: C
878:      do 100 K = 1, 6
879:        if ( INDEX(STR(1:N),TM(K:K)).eq.0 ) then
880:          T = TM(K:K)
881:          go to 110
882:        end if
883:      100 continue
884:      write(6,*) 'XX CANNOT FIND APPROPREATE TERMINATION',
885:      & ' CHARACTER WITHIN<', TM, '>'
886:      write(6,*) ' STRING = <', STR(1:N), '>'
887:      stop 666
888: C
889: C
890:      110 STR2 = T//STR(1:N)//T
891:      call CWRITE( STR2, N+2, ICON )
892:      return
893:      end
894: C
895: C
896: C=====
897: C
898:      subroutine CCOMP( STR,      LN )
899:      character*(*) STR
900:      LN = 0
901:      do 100 I = 1, LEN(STR)
902:        if ( STR(I:I).ne.' ' ) then
903:          LN = LN + 1
904:          STR(LN:LN) = STR(I:I)
905:        end if
906:      100 continue
907:      if ( LN.lt.LEN(STR) ) STR(LN+1:LEN(STR)) = ' '
908:      return
909:      end
910: C

```

src/tools/gmvplbcv.f

```

911: C=====
912: C
913:       subroutine IWRITE( IDATA, NL,      IWST )
914: C
915:       integer IDATA(*)
916: C
917:       common /CWRT1/  LINE
918:       character*256 LINE
919:       common /CWRT2/  IUOUT,  NBUF,  IPOS
920: C
921:       character*32 BUF
922:       character*16 BUF2
923: C
924:       if ( NL.le.0 ) NL  = 1
925: C
926:       if ( NL.eq.1 ) then
927:         write(BUF,'(I12)') IDATA(1)
928:         call CWRITE( BUF, 0, IWST )
929:         return
930:       end if
931: C
932:       IWST  = 0
933:       I1    = 1
934:       do 100 I = 1, NL + 1
935:         if ( I.gt.NL .or. (I.le.NL.and.IDATA(I1).ne.IDATA(I)) ) then
936:           write(BUF2,'(I16)') IDATA(I1)
937:           call CCOMP( BUF2, LN )
938:           NN      = I - I1
939:           if ( NN.gt.1 ) then
940:             write(BUF,'(I7,','*',A)') NN, BUF2(1:LN)
941:             call CWRITE( BUF, 0, ICON )
942:           else
943:             call CWRITE( BUF2, LN, ICON )
944:           end if
945:           I1      = I
946:         end if
947:       100 continue
948:       return
949:       end
950: C
951: C=====
952: C
953:       subroutine EWRITE( EDATA, NL,      IDIGIT,IWST )
954: C
955:       real EDATA(*)
956: C
957:       common /CWRT1/  LINE
958:       character*256 LINE
959:       common /CWRT2/  IUOUT,  NBUF,  IPOS
960: C
961:       character*32 BUF
962:       character*32 BUF2
963: C
964:       if ( NL.le.0 ) NL  = 1
965: C
966:       if ( NL.eq.1 ) then
967:         call CEOPT( BUF, EDATA(1), IDIGIT, LN )
968:         call CWRITE( BUF, LN, IWST )
969:         return
970:       end if
971: C
972:       IWST  = 0
973:       I1    = 1
974:       do 100 I = 1, NL + 1
975:         if ( I.gt.NL .or. (I.le.NL.and.EDATA(I1).ne.EDATA(I)) ) then

```

```

976:         call CEOPT( BUF2, EDATA(I1), IDIGIT, LN )
977:         NN      = I - I1
978:         if ( NN.gt.1 ) then
979:           write(BUF,'(I7,','*',A)') NN, BUF2(1:LN)
980:           call CWRITE( BUF, 0, ICON )
981:         else
982:           call CWRITE( BUF2, LN, ICON )
983:         end if
984:         I1      = I
985:       end if
986:     100 continue
987:     return
988:     end
989: C
990: C=====
991: C   CONVERT A FLOATING POINT NUMBER TO 'OPTIOMIZED'
992: C   FORM HAVING IDIGIT DECIMAL PRECISION AND STORE IN BUF.
993: C
994:       subroutine CEOPT( BUF,  DATA,  IDIGIT,LN )
995: C
996:       real DATA
997:       character*(*) BUF
998:       character*20 FORMT
999:       character*80 BUF2
1000: C
1001:       if ( IDIGIT.gt.9 .or. IDIGIT.le.0 ) then
1002:         write(6,*) ' XXX INVALID OR TOO LONG DIGIT (', IDIGIT, ')'
1003:         stop 666
1004:       end if
1005: C
1006:       if ( DATA.eq.0.0 ) then
1007:         BUF  = '0.'
1008:         LN   = 2
1009:         return
1010:       end if
1011: C
1012:       if ( IDIGIT.eq.8 ) then
1013:         write(BUF,'(1p,e15.7)') DATA
1014:         LN      = 15
1015:       else
1016:         FORMT   = ' '
1017:         LN      = 20
1018:         write(FORMT,'(''(1P,E'',I2,',''',I1,',''E2)''')' ) LN, IDIGIT - 1
1019: C
1020:         write(BUF,fmt =FORMT) DATA
1021:       end if
1022: C
1023:       K      = INDEX(BUF(:LN),'E')
1024: C
1025:       if ( K.ne.0 ) then
1026:         do 100 I = K - 1, 1, -1
1027:           if ( BUF(I:I).eq.'0' ) then
1028:             BUF(I:I)  = ' '
1029:           else
1030:             go to 110
1031:           end if
1032:         100 continue
1033:       110 if ( I.lt.K-1 ) call CCOMP( BUF(:LN), LL )
1034: C
1035:       K      = INDEX(BUF(:LN),'E')
1036:       NDIGIT = K - INDEX(BUF(:LN),',')
1037:       read(BUF(K+1:K+3),'(I3)') NE
1038: C
1039:       if ( NE.eq.0 ) then
1040:         BUF(K:K+3) = ' '

```

src/tools/gmvplbcv.f

```
1041:         else if ( NE.ge.-1.and.NE.lt.NDIGIT ) then
1042:             KK      = INDEX(BUF(:LN),'.')
1043:             BUF2     = BUF(1:KK-1) //BUF(KK+1:K-1)
1044:             BUF      = ' '
1045:             BUF      = BUF2(1:KK-1+NE) //'.'//BUF2(KK+NE:K)
1046:             ***      BUF(KK:KK) = BUF(KK-1:KK-1)
1047:             ***      BUF(KK-1:KK-1) = '.'
1048:             ***      BUF(K:K+3) = ' '
1049:         else if ( ABS(NE).lt.10 ) then
1050:             BUF(K+1:K+3) = ' '
1051:             if ( NE.gt.0 ) write(BUF(K+1:K+1),'(I1)') NE
1052:             if ( NE.lt.0 ) write(BUF(K+1:K+2),'(I2)') NE
1053:         end if
1054:     end if
1055: C
1056: L = LN
1057: call CCOMP( BUF(:L), LN )
1058: return
1059: end
```

src/tools/incinc.c

```

1: #include <stdlib.h>
2: #include <stdio.h>
3: #include <string.h>
4: #include <ctype.h>
5:
6: /*****
7: *   Perform include expansion for various type of INCLUDE commands
8: *
9: *   Programmed by M.Sasaki      ( 11 March 1992 )
10: *   Update :
11: *
12: *       12 June 1992
13: *
14: *       Include type specification
15: *
16: *       14 February 1993
17: *
18: *       Files included can be given with absolute path, otherwise
19: *       pathname is taken relative to that of the inputfile.
20: *       (obey the convention of UNIX fortran's INCLUDE rule)
21: *       The -Idir option is preserved for standard input.)
22: *       (12 June 1992)
23: *****/
24: /*****
25: * char Usage[] =
26: * " ----- \n"
27: * "   Usage : incinc [-Idir] [-Hhead -Ttail] infile outfile \n"
28: * " ----- " ;
29: *****/
30: /*****
31: * Does not distinct uppper/lower case for "head" and "tail".
32: *****/
33:
34: /*****
35: * Restrictions & Cautions *****/
36: *****/
37: * 1. Cannot expand nested include modules.
38: * Users must perform expansion by using this command
39: * more than once by themselves.
40: * 2. Does not expand INCLUDE on lines beginning 'C' or 'c'.
41: * Lines with '*' on the first column can be expanded.
42: * (-INCLUDE of GEM file of FACOM neglects column 1-)
43: * 3. Neglect control parameter N for IBM/VS-FORTRAN.
44: * ( INCLUDE(name) n )
45: *
46: *****/
47:
48: /*****
49: * For : INCLUDE( ...) type ***** IBM,FACOM,NEC/ACOS
50: *****/
51: *
52: * char * Include = "include(" ;
53: * char * Term = ")" ;
54: *****/
55:
56: /**default *****/
57: * For : INCLUDE '...' type **** SUN,HP,VAX
58: *****/
59: char * Include = "include\'" ;
60: char * Term = "\' " ;
61:
62: /*****
63: * For : -INCLUDE ... type **** FACOM/GEM
64: *****/
65: * char * Include = "include" ;
66:
67: * char * Term = " " ;
68: *****/
69: #define COLMMAX 132
70:
71: int IncludeLine() ;
72: int IncludeExpansion() ;
73:
74: main( argc, argv )
75: int argc ;
76: char **argv ;
77: {
78: FILE *infile, *outfile ;
79: char infile_name[256], outfile_name[256] ;
80: char *idir , *cp ;
81: char line[COLMMAX+1] ;
82: int i,j ;
83:
84: idir = NULL ;
85: if ( argc < 1 )
86: {
87: /*** fprintf( stderr, "%s\n", Usage ) ; ***/
88: fprintf(stderr, "%s\n%s\n%s\n",
89: " ----- ",
90: "   Usage : incinc [-Idir] [-Hhead -Ttail] infile outfile",
91: " ----- " ) ;
92: } ;
93: exit( 1 ) ;
94: }
95:
96:
97: /***** argumentnts *****/
98:
99: for( i=1 ,j=0 ; i < argc ; i++ )
100: {
101: if( !strcmp( argv[i] , "-I" , 2 ) )
102: {
103: idir = argv[i] + 2 ;
104: }
105: else if( !strcmp( argv[i] , "-H" , 2 ) )
106: {
107: Include = argv[i]+2 ;
108: }
109: else if( !strcmp( argv[i] , "-T" , 2 ) )
110: {
111: Term = argv[i]+2 ;
112: }
113: else if( j == 0 )
114: {
115: j++ ;
116: strcpy( infile_name, argv[i] ) ;
117: if( (infile = fopen( argv[i] , "r" ) ) == NULL )
118: {
119: fprintf(stderr, " *** Failed to open <%s> as input \n", argv[i])
120:
121: exit( 2 ) ;
122: }
123:
124: /***** set idir to that of input file name *****/
125:
126: if( idir == NULL )
127: {
128: if( (cp = strrchr( argv[i], '/' ) ) != NULL )
129: {
130: *(cp+1) = '\0' ;

```

src/tools/incinc.c

```

130:         idir = argv[i] ;
131:     }
132: }
133: }
134: else if( j==1 )
135: {
136:     j++ ;
137:     strcpy( outfile_name, argv[i] ) ;
138:     if( (outfile = fopen( argv[i] , "w" ) )==NULL )
139:     {
140:         fprintf(stderr," ++ Failed to open <%=s> as output \n",argv[i]
);
141:         exit( 2 ) ;
142:     }
143: }
144: }
145:
146: printf( "C Include header: <%=s> \n", Include ) ;
147: printf( "C Include tail : <%=s> \n", Term ) ;
148: printf( "C directory : <%=s> \n", idir ) ;
149:
150: if ( j <= 1 )
151: {
152:     if( j == 0 ) infile = stdin ;
153:     outfile = stdout ;
154: }
155:
156: /***** Beware strings taken by fgets may include new-line character */
157:
158: while( fgets( line, COLMMAX, infile ) != NULL )
159: {
160:     char incname[256] ;
161:
162:     if ( IncludeLine( incname, line ) )
163:     {
164:         if( !IncludeExpansion( incname, idir, line, outfile ) )
165:             exit(3) ;
166:     }
167:     /**** Beware strings taken by fgets may include new-line character */
168:     else
169:         fprintf( outfile, "%s", line ) ;
170: }
171: } /* End of main() */
172:
173:
174: int IncludeLine( incname, line )
175: char incname[] ;
176: char line[COLMMAX+1] ;
177: {
178:     char *cp ,*cp2 ;
179:     int i,n , len ,leni ;
180:     char buffer[256] ;
181:
182:     extern char *Include ; /* Prefix for INCLUDE module */
183:     extern char *Term ; /* Termination character of INCLUDE */
184:
185:     leni = strlen( Include ) ;
186:
187: /**** comment line ****/
188:     if( line[0] == 'C' ||
189:         line[0] == 'c' )
190:         return 0 ;
191: /** if( line[0] == 'C' ||
192: * line[0] == 'c' ||
193: * line[0] == '*' )

```

```

194: * return 0 ;
195: */
196:
197: /**** continuation ****/
198:     if( (strncmp( line,"", 5) == 0) && (line[5] != ' ') )
199:         return 0 ;
200:
201: /**** remove blank characters ****/
202:
203:     for( cp = line, len = 0 ; ; ++cp )
204:     {
205:         if( *cp != ' ' && *cp != '\n' ) buffer[len++] = *cp ;
206:
207:         if( *cp == '\0' )
208:         {
209:             if ( len < leni )
210:                 return 0 ;
211:             else
212:                 break ;
213:         }
214:     }
215: /*****
216:     if( strncasecmp( buffer, Include , leni ) != 0 )
217:         return 0 ;
218:     *****/
219:     for( i = 0 ; i < leni ; ++i )
220:     {
221:         if( toupper( buffer[i] ) != toupper( Include[i] ) )
222:             return 0 ;
223:     }
224:
225:     if( Term[0] != '\0' )
226:     {
227:         if( strlen( Term ) == 1 )
228:         {
229:             if( ( cp2 = strchr( &buffer[leni], Term[0] ) ) == NULL )
230:                 return 0 ;
231:         }
232:         else
233:         {
234:             /**** if( ( cp2 = strstr( &buffer[leni], Term ) ) == NULL ) ****/
235:             cp2 = NULL ;
236:             for ( cp = &buffer[leni] ; *cp != '\0' ; ++cp )
237:             {
238:                 if( strncmp( cp, Term, strlen(Term)) == 0 )
239:                 {
240:                     cp2 = cp ;
241:                     break ;
242:                 }
243:             }
244:             if( cp2 == NULL ) return 0 ;
245:         }
246:         /***** Put null character at the position of terminator ****/
247:         *cp2 = '\0' ;
248:     }
249:
250:     strcpy( incname, &buffer[leni] ) ;
251:
252:     return 1 ;
253:
254: } /* End of IncludeLine */
255:
256:
257:
258: int IncludeExpansion( incname, idir, line, outfile )

```

src/tools/incinc.c

```
259:     char incname[] ;
260:     char idir[] ;
261:     char line[COLMMAX+1] ;
262:     FILE *outfile ;
263: {
264:     char fname[256] ;
265:     FILE *incfile ;
266:     int nl ;
267:
268:     if( incname[0] == '/' )
269:         strcpy( fname, incname ) ;
270:     else
271:     {
272:         if ( idir != NULL )
273:         {
274:             strcpy( fname, idir ) ;
275:             if ( idir[strlen(idir)-1] != '/' ) strcat( fname, "/" ) ;
276:         }
277:         else
278:         {
279:             fname[0] = '\0' ;
280:         }
281:
282:         strcat( fname, incname ) ;
283:     }
284:
285:     if ( ( incfile = fopen( fname, "r" ) ) == NULL )
286:     {
287:         fprintf(stderr, " Failed to open include file < %s>\n", fname ) ;
288:         return 0 ;
289:     }
290:
291:     fprintf( outfile, "-----< %s>: %s %s %s Expanded \n",
292:             incname, Include, incname , Term ) ;
293:
294:     for ( nl = 0 ; fgets( line, COLMMAX , incfile ) != NULL ; nl++ )
295:     {
296:         /*** Beware strings taken by fgets may include new-line character **/
297:         fprintf( outfile, "%s", line ) ;
298:     }
299:
300:     fclose( incfile ) ;
301:
302:     fprintf( outfile,
303:             "----- %d LINES INCLUDED ABOVE (FROM %s)\n", nl, incname ) ;
304:
305:     return 1 ;
306: }
307: } /* End of IncludeExpansion **/
```

src/tools/insu.f

```
1:
2: C
3: C   PROGRAM : INSU
4: C
5: C   Defoctorization of an integer
6: C
7: C   PROGRAMMED BY M.SASAKI
8: C
9: C
10:  parameter( KK   = 40000, KL = 10000 )
11:  dimension IS(KK), JS(KL)
12: 100 read(5,*) N
13:  if ( N.eq.0 ) go to 160
14:  J   = 0
15:  M   = 0
16:  NN  = N
17:  LL  = 0
18:  do 140 I = 2, SQRT(N+0.1) + 1
19: CCCCC DO 100 I=2,      N/2+ 1
20:  if ( I.gt.NN ) go to 150
21:  if ( I.gt.4.and.(MOD(I,6).ne.1.and.MOD(I,6).ne.5) ) go to 140
22:  do 110 K = 1, J
23: 110  if ( MOD(I,IS(K)).eq.0 ) go to 140
24:  J   = J + 1
25:  if ( J.gt.KK ) then
26:  write(6,*) ' MEMORY OVER ! IS(' , J, ') = ' , I
27:  go to 100
28:  end if
29:  IS(J) = I
30:  L     = 0
31: 120  if ( MOD(NN,I).ne.0 ) go to 130
32:  LL    = LL + 1
33:  NN    = NN/I
34:  L     = L + 1
35:  go to 120
36: 130  if ( L.ne.0 ) then
37:  write(6,*) I, ' ** ', L
38:  M     = M + 1
39:  JS(M) = I
40:  end if
41: 140 continue
42: C
43: 150 if ( NN.ne.1 ) write(6,*) NN, ' ** 1'
44: C
45:  write(10,*) ' j= ', J
46:  write(10,*) (IS(K),K=1,J)
47:  go to 100
48: 160 stop
49:  end
```

src/tools/mcrdf.f

```

1:      program MCRDF
2: C-----
3: C *****
4: C MCRDF
5: C *****
6: C
7: C Calculate radial distribution function by Monte Carlo
8: C sampling method : collective rearrangement algorithms
9: C (Chemical short-range ordering effects are not
10: C incorporated)
11: C-----
12: C Reference:
13: C
14: C I.Murata, T.Mori, M.Nakagawa (JAERI)
15: C H.Shirai (AMP Technology Japan Ltd.)
16: C
17: C "Packing Simulation Code to Calculate Distribution Function
18: C of Hard Spheres by Monte Carlo Method: MCRDF"
19: C JAERI-Data/Code 96-016 (March 1996)
20: C
21: C-----
22: C OUTPUT FILE LIST
23: C I/O UNIT SUB I/O USE
24: C FT01 MCRDF WRITE RESTART FILE
25: C FT02 MCRDF READ RESTART FILE
26: C FT03 NND WRITE PLOT(NGRPH)
27: C FT04 CRCP2 WRITE PLOT(NGRPH)
28: C FT05 MCRDF WRITE INPUT DATA
29: C FT06 ALL SUB WRITE GENERAL OUTPUT
30: C FT07 CRCP WRITE PLOT(NGRPH)
31: C FT08 CRCP WRITE ANG. DIST.
32: C FT09 CRCP WRITE MCNP-CFP
33: C FT10 XSEC WRITE X-SEC FIGURE
34: C FT11 - WRITE -
35: C FT12 NND WRITE MCNP-CFP
36: C FT13 NND2/R WRITE PLOT(NGRPH)
37: C FT14 NND2/R WRITE MCNP-CFP
38: C FT15 NND3/R WRITE PLOT(NGRPH)
39: C FT16 NND3/R WRITE MCNP-CFP
40: C FT17 XSEC WRITE 2-D RDF(NGRPH)
41: C FT20 NNDMVP WRITE NND's in NND-library form of MVP/GMVP
42: C FT25 NNDMVP WRITE NND's in MVP/GMVP input form
43: C-----
44: C
45: C parameter( PIE = 3.141592, MPX = 1000 )
46: C
47: C COMMON BLOCK
48: C
49: C common /INPDT/ PFC, AAR, IBOX
50: C
51: C PFC : PACKING FRACTION
52: C AAR : DIMENSION OF CUBIC BOX
53: C IBOX: 0:RANDOM, 1:FCC, 2:BCC, 3:UCL
54: C
55: C common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
56: C & RDF(MPX), RCP(3,MPX), RCP2(3,MPX), SIGT,
57: C & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
58: C & NNR
59: C
60: C XXX : X-CORDINATE OF PARTICLE
61: C YYY : Y-CORDINATE OF PARTICLE
62: C ZZZ : Z-CORDINATE OF PARTICLE
63: C NNR : THE NUMBER OF PARTICLE BEING PACKED IN THE BOX
64: C RAR : RADIUS OF PARTICLE
65: C RDF : RADIAL DISTRIBUTION FUNCTION
66: C
67: C RCP : RADIAL COLLISION PROBABILITY
68: C
69: C common /RANCM/ IX, NRN
70: C
71: C IX : RANDOM NUMBER INDEX
72: C NRN : NUMBER OF RANDOM NUMBER CALLED
73: C
74: C common /OVLCM/ ANL, DDR, CCC, RMI, RMA,
75: C & NNP(100,MPX), NP(MPX), MMP(100,MPX), MP,
76: C & NOL, IAD, NOM, ICE
77: C
78: C ANL : AVERAGE OVERLAPPED PARTICLES FOR EACH PARTICLE
79: C DDR : AVERAGE OVERLAPPING LENGTH
80: C CCC : OVERLAPPING ADJUSTMENT FACTOR FOR NP(I)=1
81: C RMI : MINIMUM OVERLAPPING LENGTH
82: C RMA : MAXIMUM OVERLAPPING LENGTH
83: C NNP : THE PARTICLE NUMBER OF OVERLAPPED PARTICLE
84: C NP : THE NUMBER OF OVERLAPPED PARTICLES FOR EACH PARTICLE
85: C MMP : THE PARTICLE NUMBER OF NOT-OVERLAPPED PARTICLE
86: C MP : THE NUMBER OF NOT-OVERLAPPED PARTICLES(FIXED)
87: C NOL : THE NUMBER OF PARTICLES HAVING OVERLAPPED ONES
88: C IAD : FLAG TO PROCESS ADJUSTING FOR NOT-OVERLAPPED PARTICLES
89: C NOM : PARTICLE NUMBER WHICH HAS THE LARGEST OVERLAPPING
90: C ICE : FROM THIS CYCLE, SPECIAL TREATMENT IS USED FOR OVLPPED
91: C SIGT: TOTAL MACRO CROSS SECTION OF MATRIX
92: C
93: C common /TRNCM/ XTR(27), YTR(27), ZTR(27)
94: C
95: C XTR : TRANSFORMATION MATRIX X
96: C YTR : TRANSFORMATION MATRIX Y
97: C ZTR : TRANSFORMATION MATRIX Z
98: C
99: C OTHER DIMENSIONS
100: C dimension DIS(MPX), MM(100), IUM(MPX), D1(1000), I1(1000),
101: C & IST(10), DRAR(100), LOPT(5)
102: C
103: C INITIALIZE COMMON CONSTANTS
104: C
105: C ISP = 0
106: C IX = 0
107: C NRN = 0
108: C RAR = .5
109: C MP = 5
110: C IAD = 0
111: C ISTMX = 10
112: C
113: C NUMBER OF PARTICLES
114: C read(5,*) NNR
115: C if ( NNR.le.0 ) NNR = 300
116: C
117: C SKIP COUNT OF RANDOM NUMBER
118: C
119: C read(5,*) ISKIP
120: C
121: C IBOX : 0 ; RANDOM PACKING
122: C 1 ; FACE CENTERED CUBIC LATTICE
123: C 2 ; BODY CENTERED CUBIC LATTICE
124: C 3 ; UNIT CUBIC LATTICE
125: C
126: C read(5,*) IBOX
127: C
128: C A SELECT OF INITIAL CONDITION
129: C
130: C read(5,*) IREAD
131: C if ( IBOX.ne.0 ) then

```


src/tools/mcrdf.f

```

131:      read(5,*) DUMM
132:      else if ( ABS(IREAD).ne.1 ) then
133:        read(5,*) PFC
134:        AAR      = RAR*((4.*PIE*NNR/3./PFC)**.3333333)
135:      else
136:        read(2,'(I6)') NNR
137:        read(2,'(I12)') IX
138:        read(2,'(E11.4)') PFC
139:        read(2,'(E11.4)') RAR
140:        read(2,'(E11.4)') AAR
141:        do 100 I = 1, NNR
142:          read(2,'(3E12.5)') XXX(I), YYY(I), ZZZ(I)
143: 100    continue
144:        read(5,*) DUMM
145:      end if
146: C
147:      read(5,*) IADD
148:      read(5,*) CTME
149:      read(5,*) RCF
150:      read(5,*) IVIB
151:      read(5,*) ICY
152:      read(5,*) DDRTL
153:      read(5,*) DRTTL
154:      read(5,*) LOPT(1)
155:      read(5,*) LOPT(2)
156:      read(5,*) LOPT(3)
157:      read(5,*) LOPT(4)
158:      read(5,*) LOPT(5)
159:      read(5,*) SIGT
160: C
161: C      SET THE INITIAL PARTICLE COORDINATION DISTRIBUTION
162: C
163:      if ( ABS(IREAD).ne.1 ) then
164:        if ( IBOX.ne.0 ) then
165:          if ( IBOX.eq.1 ) call FCC
166:          if ( IBOX.eq.2 ) call BCC
167:          if ( IBOX.eq.3 ) call UCL
168:        else
169:          do 110 I = 1, NNR
170:            XXX(I) = (RANG()-0.5)*AAR
171:            YYY(I) = (RANG()-0.5)*AAR
172:            ZZZ(I) = (RANG()-0.5)*AAR
173: 110      continue
174:          end if
175:        end if
176: C
177: C      PRINT INPUT CONDITION
178: C
179:      write(6,*) ' CALCULATION OPT. : ', IBOX
180:      if ( IBOX.eq.0 ) write(6,*) ' CALCULATE RANDOM PACKING.'
181:      if ( IBOX.eq.1 ) write(6,*) ' CALCULATE FCC PACKING.'
182:      if ( IBOX.eq.2 ) write(6,*) ' CALCULATE BCC PACKING.'
183:      if ( IBOX.eq.3 ) write(6,*) ' CALCULATE UCL PACKING.'
184:      write(6,*) ' CONTINUE OPTION : ', IREAD
185:      if ( ABS(IREAD).eq.1 ) write(6,*)
186:      & ' (READ XXX,YYY,ZZZ FROM FT02.)'
187:      if ( IREAD.eq.-1 ) write(6,*) ' ADJUSTMENT IS NOT DONE.'
188:      write(6,*) ' PACKING FRACTION : ', PFC
189:      write(6,*) ' PARTICLE RADIUS : ', RAR
190:      write(6,*) ' CELL DIMENSION : ', AAR
191:      write(6,*) ' NO. OF PARTICLE : ', NNR
192:      write(6,*) ' RANDOM SEED IX : ', IX
193:      write(6,*) ' RANDOM SKIP NO. : ', ISKIP
194:      write(6,*) ' ADJ. NOT-OVERLAP : ', IADD
195:      write(6,*) ' DDR TOLERANCE : ', DDRTL

```

```

196:      write(6,*) ' LIM. OF DDR DEC. : ', DRTTL
197:      write(6,*) ' NO. OF CYCLE : ', ICY
198:      write(6,*) ' CPU TIME(MIN) : ', CTME
199:      write(6,*) ' BINDER CORRECT. : ', -RCF*100., '(%)'
200:      write(6,*) ' VIB. OPTION : ', IVIB
201:      write(6,*) ' ADJ. OVERLAPPED : ', ICE
202:      write(6,*) ' (TOLERANCE)'
203:      write(6,*) ' TOT. MAC. X-SEC : ', SIGT
204:      write(6,*) ' OPTION CAL.(>1:ON,0:OFF)'
205:      write(6,*) ' CRCP2 : ', LOPT(1)
206:      write(6,*) ' NND : ', LOPT(2)
207:      write(6,*) ' NND2 : ', LOPT(3)
208:      write(6,*) ' NND3 : ', LOPT(4)
209:      write(6,*) ' XSEC : ', LOPT(5)
210: C
211: C      SETTING THE TRANSFORMATION MATRIX
212: C
213:      do 120 I = 1, 27
214:        XTR(I) = XTR(I)*AAR
215:        YTR(I) = YTR(I)*AAR
216:        ZTR(I) = ZTR(I)*AAR
217: 120    continue
218: C
219: C      BYPASS THE ADJUSTMENT IN CASE OF CRYSTAL OR IREAD=-1
220: C
221:      if ( IBOX.ne.0 ) go to 190
222:      if ( IREAD.eq.-1 ) go to 190
223: C
224: C      SKIPPING RANDOM NUMBERS IF ISKIP > 0
225: C
226:      do 130 I = 1, ISKIP
227:        A = RANG()
228: 130    continue
229: C
230: C      DETERMINE PACKING ARRANGEMENT BY COORDINATE ADJUSTMENT
231: C
232:      write(6,'(1H1)')
233:      write(6,*)
234:      & ' HAVING AV.NO. AVERAGE MINIMUM ',
235:      & ' MAXIMUM'
236:      write(6,*)
237:      & ' CYCLE OVLAPD OVL.PAR OVL.LEN OVL.LEN ',
238:      & ' OVL.LEN'
239: C
240: C      MAIN ROUTINE START
241: C
242:      IC = 0
243: 140 CTME9 = CTME*0.9
244:      ICV = 0
245:      ICPU = 0
246: C
247:      do 150 ICC = 1, ICY
248:        IC = IC + 1
249: C
250: C      CHECK CPU TIME
251: C
252:      call CLOCKM( MMM )
253:      CMT = REAL(MMM) /1000./60.
254:      if ( CMT.gt.CTME9 ) ICPU = 1
255:      if ( CMT.gt.CTME ) then
256:        write(6,*) ' CPU TIME IS EXHAUSTED.'
257:        go to 160
258:      end if
259: C
260: C      FIND PARTICLES OVERLAPPED

```

src/tools/mcrdf.f

```

261: C
262:       call OVLP
263: C
264: *       call OVLPB
265: C
266: C       IF(NOL.EQ.0) GOTO 1100
267:       if ( MOD(IC,40).eq.0 ) then
268:           write(6,7000) IC, NOL, ANL, DDR*2.*RAR, RMI, RMA
269:       7000       format(' ',I8,2X,I6,2X,1P3E10.3,2X,E10.3,2X,E10.3,2X,E10.3)
270:       end if
271: C
272: C       CHECK OVERLAPPING LENGTH TOLERANCE & INCREASE RADII
273: C
274:       if ( DDR.lt.DDRTL ) then
275:           write(6,*) ' DDR IS CHECKED.'
276:           write(6,*) ' CALCULATION IS CONVERGED AT CYCLE(' , IC, ') '
277:           RARN = RAR - RMA/2.
278:           V = 4./3.*PIE*(RARN**3)*FLOAT(NNR)
279:           PF = V/(AAR**3)
280: CCC       IF ( PF.LT. PFC .AND. ICPU.EQ.0 ) THEN
281: C           WRITE(6,*) ' CALCULATED PACKING FRACTION ', PF,
282: C           ' IS STILL SMALLER THAN TARGET FRACTION ', PFC
283: C           RAR0 = RAR
284: C           RAR = RAR + DRAR(ISP)
285: C           IF ( ISP.GT.1 ) ISP = ISP - 1
286: C           WRITE(6,*) ' INCREASE RADII FROM ', RAR0, ' TO ', RAR
287: C           ICV = 0
288: C           CALL VIB
289: C           GOTO 950
290: CCC       ELSE
291: C           go to 180
292: CCC       END IF
293:       end if
294: C
295: C       CHECK CONVERGING RATE AND CALL VIB MODULE BY THE RESULT
296: C
297:       ICV = ICV + 1
298:       call DRATE( ICV, DDR, DDRATE )
299:       if ( DDRATE.lt.DRTTL.and.ICPU.eq.0 ) then
300:           ICV = 0
301:           call VIB
302:       end if
303: C
304: C       ADJUST THE COORDINATES OF THE PARTICLES
305: C
306:       call ADJST( IC, IST(1) )
307: C
308: 150 continue
309: C
310: C       TOO MUCH LOOP NEEDED. DECREASE RADII
311: C
312:       write(6,*) ' TOO MUCH LOOP NEEDED '
313:       RAR0 = RAR
314:       RAR = RAR - DDR
315:       ISP = ISP + 1
316:       DRAR(ISP) = DDR
317:       write(6,*) ' DECREASE RADII FROM ', RAR0, ' TO ', RAR
318:       go to 140
319: C
320: C       IN CASE OF NO CONVERGENCE
321: C       STORE PARTICLE ARRANGEMENT TO FILE 01
322: C
323: 160 write(6,*) ' CYCLE(' , IC, ') IS DONE. NOT CONVERGED.'
324:       write(1,'(I6)') NNR
325:       write(1,'(I12)') IX

```

```

326:       write(1,'(1P3E11.4)') PFC
327:       write(1,'(1P3E11.4)') RAR
328:       write(1,'(1P3E11.4)') AAR
329:       do 170 I = 1, NNR
330:           write(1,'(1P3E12.5)') XXX(I), YYY(I), ZZZ(I)
331: 170 continue
332:       write(6,*) ' XXX,YYY,ZZZ ARE WRITTEN IN FT01.'
333:       go to 250
334: C
335: C       REDUCTION OF RADIUS FOR NOT OVERLAPPING EACH OTHER
336: C
337: 180 RATIO = (RAR-RMA/2.) /RAR - 1.
338:       RAR = RAR - RMA/2.
339:       write(6,*) ' NEW RAR = ', RAR
340:       write(6,*) ' -BY ', RATIO*100., '(%)-'
341: C
342: C       COMPUTE PACKING FRACTION
343: C
344: 190 V = 4./3.*PIE*RAR*RAR*RAR*FLOAT(NNR)
345:       PF = V/AAR/AAR/AAR
346:       write(6,'(1H1)')
347:       write(6,*) ' CALCULATED PACKING FRACTION : ', PF
348:       write(6,'(1H0)')
349:       if ( IBOX.eq.0 ) then
350:           write(6,*) ' NO. X Y Z ',
351:           & ' NEAREST NEIGHBOR(1-5) DISTANCE(1-5)'
352:           do 210 I = 1, NNR
353: CCCCC       CALL CKXYZ(XXX(I),YYY(I),ZZZ(I))
354:           do 200 J = 1, NNR
355:               R =
356:               & DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),ZZZ(J),
357:               & III)
358:               DIS(J) = R
359: 200       continue
360:           call SORT( DIS(1), NNR, MM(1), 5 )
361:           write(6,7020) I, XXX(I), YYY(I), ZZZ(I), (MM(J),J=1,5),
362:           & (DIS(MM(K)),K=1,5)
363: 7020       format(' ',I4,2X,1P3E11.3,5I4,2X,5E11.3)
364: 210       continue
365:       end if
366: C
367: C       SPECIAL OUTPUT FOR FCC, BCC AND UCL
368:       if ( IBOX.ne.0 ) then
369:           do 220 J = 1, NNR
370:               R =
371:               & DIST(XXX(1),YYY(1),ZZZ(1),XXX(J),YYY(J),ZZZ(J),III)
372:               DIS(J) = R
373: 220       continue
374:           call SORT( DIS(1), NNR, IUM(1), NNR )
375:           N1 = 1
376:           D1(N1) = DIS(IUM(2))
377:           I1(N1) = 1
378:           do 230 J = 3, NNR
379:               if ( ABS(DIS(IUM(J))-D1(N1))/D1(N1).lt.1.E-3 ) then
380:                   I1(N1) = I1(N1) + 1
381:               else
382:                   N1 = N1 + 1
383:                   I1(N1) = 1
384:                   D1(N1) = DIS(IUM(J))
385:               end if
386: 230       continue
387:           write(6,'(1H0)')
388:           write(6,*) ' DISTANCE FROM THE CENTER'
389:           write(6,'(10F6.3,I4,2X)') (D1(J),I1(J),J=1,N1)
390:       end if

```

src/tools/mcrdf.f

```

391: C
392: C      PARTICLE ARRANGEMENT IS FINISHED.
393: C
394: C      call CLOCKM( MMM )
395: C      write(6,*) ' CPU TIME(STEP 1 END) : ', REAL(MMM) /1000., '(SEC)'
396: C
397: C      STORE PARTICLE ARRANGEMENT TO FILE 01
398: C      write(1,'(I6)') NNR
399: C      write(1,'(I12)') IX
400: C      write(1,'(1PE11.4)') PF
401: C      write(1,'(1PE11.4)') RAR
402: C      write(1,'(1PE11.4)') AAR
403: C      do 240 I = 1, NNR
404: C         write(1,'(1P3E12.5)') XXX(I), YYY(I), ZZZ(I)
405: 240 continue
406: C      write(6,*) ' XXX,YYY,ZZZ ARE WRITTEN IN FT01.'
407: C
408: C      STEP 2
409: C      PARTICLE RADIAL DISTRIBUTION FUNCTION IS CALCULATED.
410: C
411: C      check %%%%%%%%%%%%%
412: C      write(6,*) '%%% PF RAR AAR ', PF, RAR, AAR
413: C      %%%%%%%%%%%%%
414: C      call CRDF
415: C      IF(ibox.ne.0) GOTO 1200
416: C
417: C      SPECIAL TREATMENT FOR HTTR COATED FUEL PARTICLE
418: C      REDUCE RADIUS DUE TO COATED GRAPHITE BINDER
419: C
420: C      PF2 = PF
421: C      if ( RCF.gt.0. ) then
422: C         RAR = 0.5
423: C         RAR0 = RAR
424: C         RAR = RAR*(1.-RCF)
425: C         PF2 = PF*RAR*RAR*RAR/RAR0/RAR0/RAR0
426: C         write(6,*) ' SPECIAL TREATMENT FOR HTTR COATED FUEL PARTICLE'
427: C         write(6,*) ' RAR=', RAR
428: C         write(6,*) ' INPUT PACKING FRACTION =', PFC
429: C         write(6,*) ' CALCLATED PACKING FRACTION =', PF
430: C         write(6,*) ' FINAL PACKING FRACTION =', PF2
431: C
432: C      ... RCF > 0 : |RCF| is final packing fraction
433: C
434: C      ( Added by M.Sasaki : Jan 1998 )
435: C
436: C      else if ( RCF.lt.0.0 ) then
437: C         PF2 = ABS(RCF)
438: C         RAR = (PF2/PF)**3 * RAR
439: C      end if
440: C
441: C      STEP 3
442: C      call CLOCKM( MMM )
443: C      write(6,*) ' CPU TIME(STEP 2 END) : ', REAL(MMM) /1000., '(SEC)'
444: C
445: C      RADIAL COLLISION PROBABILITY DENSITY IS CALCULATED.
446: C
447: C      PARTICLE STARTS FROM THE SURFACE OF THE PARTICLE
448: C      call CRCP
449: C      call CLOCKM( MMM )
450: C      write(6,*) ' CPU TIME(AFTER CRCP) : ', REAL(MMM) /1000., '(SEC)'
451: C
452: C      PARTICLE STARTS FROM THE CENTER OF THE PARTICLE
453: C      if ( LOPT(1).eq.1 ) then
454: C         call CRCP2
455: C         call CLOCKM( MMM )

```

```

456: C      write(6,*) ' CPU TIME(AFTER CRCP2) : ', REAL(MMM) /1000.,
457: C      &      '(SEC)'
458: C      end if
459: C
460: C      CALCULATE NND DISTRIBUTION BY EXTENDED NND CALCULATION OPTION
461: C      PARTICLES START FROM THE SURFACE OF THE CFP
462: C      if ( LOPT(2).eq.1 ) then
463: C         call NND
464: C         call CLOCKM( MMM )
465: C         write(6,*) ' CPU TIME(AFTER NND) : ', REAL(MMM) /1000., '(SEC)'
466: C      end if
467: C
468: C      CALCULATE NND DISTRIBUTION BY EXTENDED NND CALCULATION OPTION
469: C      PARTICLES START FROM THE MATRIX
470: C      if ( LOPT(3).eq.1 .or. LOPT(3).eq.2 ) then
471: C         if ( LOPT(3).eq.1 ) call NND2
472: C         if ( LOPT(3).eq.2 ) call NND2R
473: C         call CLOCKM( MMM )
474: C         write(6,*) ' CPU TIME(AFTER NND2) : ', REAL(MMM) /1000.,
475: C         &      '(SEC)'
476: C      end if
477: C
478: C      CALCULATE NND DISTRIBUTION BY EXTENDED NND CALCULATION OPTION
479: C      PARTICLES START FROM THE PLANE OUT OF THE BOX
480: C      if ( LOPT(4).ge.1.and.LOPT(4).le.5 ) then
481: C         if ( LOPT(4).eq.1 ) call NND3
482: C         if ( LOPT(4).eq.2 ) call NND3R
483: C         if ( LOPT(4).eq.3 ) call NND3R2
484: C         if ( LOPT(4).eq.4 ) call NND3R3
485: C         if ( LOPT(4).eq.5 ) call NND3R4( PF2 )
486: C         call CLOCKM( MMM )
487: C         write(6,*) ' CPU TIME(AFTER NND3) : ', REAL(MMM) /1000.,
488: C         &      '(SEC)'
489: C      end if
490: C
491: C      PAIR CORRELATION FUNCTION IS CALCULATED
492: C      if ( LOPT(5).eq.1 ) then
493: C         call XSEC
494: C         call CLOCKM( MMM )
495: C         write(6,*) ' CPU TIME(AFTER XSEC) : ', REAL(MMM) /1000.,
496: C         &      '(SEC)'
497: C      end if
498: C
499: C      ... output NND's in "NND-library" form for MVP/GMVP ...
500: C
501: C
502: C      if ( LOPT(2).ne.0.and.LOPT(3).ne.0.and.LOPT(4).ne.0 ) then
503: C         call NNDMVP
504: C      end if
505: C
506: 250 call CLOCKM( MMM )
507: C      write(6,'(I10)')
508: C      write(6,*) ' CALLED RANDOM NUMBER : ', NRN
509: C      write(6,*) ' NEXT RANDOM NUMBER SEED : ', IX
510: C      write(6,*) ' CPU TIME : ', REAL(MMM) /1000., '(SEC)'
511: C      stop
512: C      end
513: C
514: C=====
515: C
516: C      subroutine FCC
517: C      MAKE FCC LATTICE
518: C
519: C      parameter( PIE = 3.141592, MPX = 1000 )
520: C

```

src/tools/mcrdf.f

```

521:      common /INPDT/   PFC,      AAR,      IBOX
522:      common /MAINCM/  XXX(MPX),  YYY(MPX),  ZZZ(MPX),
523:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
524:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
525:      &      NNR
526: C
527:      dimension AX(4), AY(4), AZ(4)
528: C
529: C      NAN : NAMBER OF PARTICLES IN A UNIT CELL
530: C      NN  : NUMBER OF UNIT CELLS ALONG THE AXIS
531: C      NLA : NUMBER OF UNIT CELLS
532: C      NNR : NUMBER OF PARTICLES
533: C      A   : UNIT CELL AXIS LENGTH
534: C      AX  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
535: C      AY  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
536: C      AZ  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
537: C
538:      NAN      = 4
539:      NN       = 5
540:      NLA      = NN*NN*NN
541: C      NNR=NLA*NAN
542:      A        = RAR*SQRT(2.)*2.
543:      PFC      = 4./3.*PIE*RAR*RAR*RAR*REAL(NAN) /A/A/A
544:      AAR      = A*REAL(NN)
545: C
546: C      RELATIVE ADDRESS FROM THE CENTER OF UNIT CELL
547:      AX(1)    = -.5*A
548:      AY(1)    = -.5*A
549:      AZ(1)    = -.5*A
550:      AX(2)    = -.5*A
551:      AY(2)    = 0.
552:      AZ(2)    = 0.
553:      AX(3)    = 0.
554:      AY(3)    = -.5*A
555:      AZ(3)    = 0.
556:      AX(4)    = 0.
557:      AY(4)    = 0.
558:      AZ(4)    = -.5*A
559: C
560:      NO       = (NN-1) /2
561:      NNR      = 0
562:      do 130 I1 = -NO, NO
563:      do 120 I2 = -NO, NO
564:      do 110 I3 = -NO, NO
565:      X1       = A*REAL(I1)
566:      Y1       = A*REAL(I2)
567:      Z1       = A*REAL(I3)
568:      do 100 I4 = 1, NAN
569:      XXX(NNR+I4) = X1 + AX(I4)
570:      YYY(NNR+I4) = Y1 + AY(I4)
571:      ZZZ(NNR+I4) = Z1 + AZ(I4)
572: 100      continue
573:      NNR      = NNR + NAN
574: 110      continue
575: 120      continue
576: 130      continue
577: C
578:      return
579:      end
580: C
581: C=====
582: C
583:      subroutine BCC
584:      MAKE BCC LATTICE
585: C

```

```

586:      parameter( PIE = 3.141592, MPX = 1000 )
587: C
588:      common /INPDT/   PFC,      AAR,      IBOX
589:      common /MAINCM/  XXX(MPX),  YYY(MPX),  ZZZ(MPX),
590:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
591:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
592:      &      NNR
593: C
594:      dimension AX(2), AY(2), AZ(2)
595: C
596: C      NAN : NAMBER OF PARTICLES IN A UNIT CELL
597: C      NN  : NUMBER OF UNIT CELLS ALONG THE AXIS
598: C      NLA : NUMBER OF UNIT CELLS
599: C      NNR : NUMBER OF PARTICLES
600: C      A   : UNIT CELL AXIS LENGTH
601: C      AX  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
602: C      AY  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
603: C      AZ  : RELATIVE CORDINATE OF PARTICLE IN A UNIT CELL
604: C
605:      NAN      = 2
606:      NN       = 7
607:      NLA      = NN*NN*NN
608: C      NNR=NLA*NAN
609:      A        = RAR/SQRT(3.)*4.
610:      PFC      = 4./3.*PIE*RAR*RAR*RAR*REAL(NAN) /A/A/A
611:      AAR      = A*REAL(NN)
612: C
613: C      RELATIVE ADDRESS FROM THE CENTER OF UNIT CELL
614:      AX(1)    = -.5*A
615:      AY(1)    = -.5*A
616:      AZ(1)    = -.5*A
617:      AX(2)    = 0.
618:      AY(2)    = 0.
619:      AZ(2)    = 0.
620: C
621:      NO       = (NN-1) /2
622:      NNR      = 0
623:      do 130 I1 = -NO, NO
624:      do 120 I2 = -NO, NO
625:      do 110 I3 = -NO, NO
626:      X1       = A*REAL(I1)
627:      Y1       = A*REAL(I2)
628:      Z1       = A*REAL(I3)
629:      do 100 I4 = 1, NAN
630:      XXX(NNR+I4) = X1 + AX(I4)
631:      YYY(NNR+I4) = Y1 + AY(I4)
632:      ZZZ(NNR+I4) = Z1 + AZ(I4)
633: 100      continue
634:      NNR      = NNR + NAN
635: 110      continue
636: 120      continue
637: 130      continue
638: C
639:      return
640:      end
641: C
642: C=====
643: C
644:      subroutine UCL
645:      MAKE UNIT CUBIC LATTICE
646: C
647:      parameter( PIE = 3.141592, MPX = 1000 )
648: C
649:      common /INPDT/   PFC,      AAR,      IBOX
650:      common /MAINCM/  XXX(MPX),  YYY(MPX),  ZZZ(MPX),

```

src/tools/mcrdf.f

```

651:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
652:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
653:      &      NNR
654: C
655:      dimension AX(1), AY(1), AZ(1)
656: C
657: C      NAN : NUMBER OF PARTICLES IN A UNIT CELL
658: C      NN  : NUMBER OF UNIT CELLS ALONG THE AXIS
659: C      NLA : NUMBER OF UNIT CELLS
660: C      NNR : NUMBER OF PARTICLES
661: C      A   : UNIT CELL AXIS LENGTH
662: C      AX  : RELATIVE COORDINATE OF PARTICLE IN A UNIT CELL
663: C      AY  : RELATIVE COORDINATE OF PARTICLE IN A UNIT CELL
664: C      AZ  : RELATIVE COORDINATE OF PARTICLE IN A UNIT CELL
665: C
666:      NNR = 1
667:      NN  = 9
668:      NLA = NN*NN*NN
669: C      NNR=NLA*NAN
670:      A   = RAR*2.
671:      PFC = 4./3.*PIE*RAR*RAR*RAR*REAL(NAN)/A/A/A
672:      AAR = A*REAL(NN)
673: C
674: C      RELATIVE ADDRESS FROM THE CENTER OF UNIT CELL
675: C      AX(1) = -.5*A
676: C      AY(1) = -.5*A
677: C      AZ(1) = -.5*A
678: C
679:      NO = (NN-1)/2
680:      NNR = 0
681:      do 130 I1 = -NO, NO
682:      do 120 I2 = -NO, NO
683:      do 110 I3 = -NO, NO
684:      X1 = A*REAL(I1)
685:      Y1 = A*REAL(I2)
686:      Z1 = A*REAL(I3)
687:      do 100 I4 = 1, NAN
688:      XXX(NNR+I4) = X1 + AX(I4)
689:      YYY(NNR+I4) = Y1 + AY(I4)
690:      ZZZ(NNR+I4) = Z1 + AZ(I4)
691:      100 continue
692:      NNR = NNR + NAN
693:      110 continue
694:      120 continue
695:      130 continue
696: C
697:      return
698:      end
699: C
700: C=====
701: C
702:      subroutine OVLP
703: C      FIND OVERLAPPED AND NOT-OVERLAPPED PARTICLE
704:      parameter( MPX = 1000 )
705:      common /MAINCM/ XXX(MPX),      YYY(MPX),      ZZZ(MPX),
706:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
707:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
708:      &      NNR
709:      common /OVLCM/ ANL,      DDR,      CCC,      RMI,      RMA,
710:      &      NNP(100,MPX),      NP(MPX),      MMP(100,MPX),      MP,
711:      &      NOL,      IAD,      NOM,      ICE
712:      common /INPDT/ PFC,      AAR,      IBOX
713: C      FIND OVERLAPPED
714:      RMI = 1.E10
715:      RMA = 0.

```

```

716:      DDR = 0.
717:      NOL = 0
718:      ANL = 0.
719:      do 100 I = 1, NNR
720:      NP(I) = 0
721:      100 continue
722:      RAR2 = RAR*2.
723:      AR2 = AAR*0.5
724:      do 120 I = 1, NNR
725:      do 110 J = I + 1, NNR
726:      DX = MIN(ABS(XXX(J)-XXX(I)),AAR-ABS(XXX(J)-XXX(I)))
727:      if ( DX.le.RAR2 ) then
728:      DY = MIN(ABS(YYY(J)-YYY(I)),AAR-ABS(YYY(J)-YYY(I)))
729:      if ( DY.le.RAR2 ) then
730:      DZ =
731:      &      MIN(ABS(ZZZ(J)-ZZZ(I)),AAR-ABS(ZZZ(J)-ZZZ(I)))
732:      if ( DZ.le.RAR2 ) then
733:      R =
734:      &      DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),
735:      &      ZZZ(J),III)
736:      if ( R.lt.RAR2 ) then
737:      R = SQRT(R)
738:      NP(I) = NP(I) + 1
739:      NP(J) = NP(J) + 1
740:      NNP(NP(I),I) = J
741:      NNP(NP(J),J) = I
742:      SDR = 2.*RAR - R
743:      DDR = DDR + SDR
744:      if ( SDR.lt.RMI ) then
745:      RMI = SDR
746:      else if ( SDR.gt.RMA ) then
747:      NOM = I
748:      RMA = SDR
749:      end if
750:      end if
751:      end if
752:      end if
753:      end if
754:      110 continue
755:      ANL = ANL + REAL(NP(I))
756:      if ( NP(I).ne.0 ) NOL = NOL + 1
757:      120 continue
758: C
759:      if ( NOL.ne.0 ) then
760:      DDR = DDR/ANL/2./RAR
761:      ANL = ANL/REAL(NOL)
762: C      WRITE(6,*) ' THE NUMBER OF PARTICLES HAVING OVERLAPPED : ',NOL
763: C      WRITE(6,*) ' THE AVERAGE NUMBER OF OVERLAPPED PARTICLES : ',ANL
764: C      WRITE(6,*) ' THE AVERAGE OVERLAPPED LENGTH : ',DDR,
765: C      1 '*RAR*2.0'
766:      end if
767:
768:      end
769: C
770: C=====
771: C
772:      subroutine OVLPB
773:      subroutine OVLP
774: C
775: C      FIND OVERLAPPED AND NOT-OVERLAPPED PARTICLE
776: C
777: C      ... Modified by M.Sasaki (using block method)
778: C
779:      parameter( MPX = 1000 )
780:      common /MAINCM/ XXX(MPX),      YYY(MPX),      ZZZ(MPX),

```

src/tools/mcrdf.f

```

781:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
782:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
783:      &      NNR
784:      common /OVLCM/  ANL,      DDR,      CCC,      RMI,      RMA,
785:      &      NNP(100,MPX),      NP(MPX),      MMP(100,MPX),      MP,
786:      &      NOL,      IAD,      NOM,      ICE
787:      common /INPDT/  PFC,      AAR,      IBOX
788: C
789:      parameter( MBOX = 7 )
790:      parameter( MNBOX = 300 )
791:      integer LINDX(MNBOX,MBOX,MBOX,MBOX)
792:      integer NINDX(MBOX,MBOX,MBOX)
793:      integer IDXWK(MPX)
794:      common /BWORK/  LINDX,      NINDX,      IDXWK
795: C
796:      data IMESSAGE /0/
797:      data XBOX0 /0.0/
798: C
799: C .... box division size (should be greater than diameter of particle)
800: C
801:      NBOX = INT(AAR/((2.0+0.1)*RAR))
802:      if ( NBOX.gt.MBOX ) NBOX = MBOX
803:      XBOX = AAR/NBOX
804: C
805:      if ( IMESSAGE.eq.0.or.abs(XBOX0/XBOX-1.0).gt.0.01 ) then
806:        write(6,*) ' *** divided into n*n blocks : n = ', NBOX
807:        write(6,*) ' *** block size : xbox ', XBOX, ' XBOX0 ',XBOX0
808:      end if
809:      XBOX0 = XBOX
810: C
811:      do K = 1, NBOX
812:        do J = 1, NBOX
813:          do I = 1, NBOX
814:            NINDX(I,J,K) = 0
815:          end do
816:        end do
817:      end do
818: C
819: C ... coordinates are defined between -0.5*aar and 0.5*aar
820: C
821:      XAMIN = AAR*0.5
822:
823: check %%%
824: c      aa1 = 100.0
825: c      aa2 = -100.0
826: c %%%%%%%%%
827:
828: C
829: C ... register particles in boxes
830: C
831:      do I = 1, NNR
832:        II = MAX(1,MIN(INT((XXX(I)+XAMIN)/XBOX)+1,NBOX))
833:        JJ = MAX(1,MIN(INT((YYY(I)+XAMIN)/XBOX)+1,NBOX))
834:        KK = MAX(1,MIN(INT((ZZZ(I)+XAMIN)/XBOX)+1,NBOX))
835:        NINDX(II,JJ,KK) = NINDX(II,JJ,KK) + 1
836:        if ( NINDX(II,JJ,KK).le.MNBOX ) then
837:          LINDX(NINDX(II,JJ,KK),II,JJ,KK) = I
838:        end if
839:
840: check %%%
841: c      aa1 = min( aa1,xxx(i)/aar,yyy(i)/aar,zzz(i)/aar)
842: c      aa2 = max( aa2,xxx(i)/aar,yyy(i)/aar,zzz(i)/aar)
843: c %%%%%%%%%
844:
845:      end do

```

```

846:
847: check %%%
848: c      write(*,*) ' aa1 aa2 ',aa1,aa2
849: c %%%%%%%%%
850:
851:      MM = 0
852:      do K = 1, NBOX
853:        do J = 1, NBOX
854:          do I = 1, NBOX
855:            MM = MAX(NINDX(I,J,K),MM)
856:          end do
857:        end do
858:      end do
859:      if ( IMESSAGE.lt.10 ) then
860:        write(6,*) ' *** max of particles per block: ', MM
861:        IMESSAGE = IMESSAGE + 1
862:      end if
863:      if ( MM.gt.MNBOX ) then
864:        write(6,*) ' xxx too many particles are in a block!! (limit=',
865:      &      MNBOX
866:      stop 999
867:      end if
868: C
869: C      FIND OVERLAPPED
870:      RMI = 1.E10
871:      RMA = 0.
872:      DDR = 0.
873:      NOL = 0
874:      ANL = 0.
875:      do 100 I = 1, NNR
876:        NP(I) = 0
877: 100 continue
878:      RAR2 = RAR*2.
879:      RAR2SQ = RAR2*RAR2
880:      AR2 = AAR*0.5
881: C
882:      do KK = 1, NBOX
883:        do JJ = 1, NBOX
884:          do 130 II = 1, NBOX
885:            MM1 = NINDX(II,JJ,KK)
886:            if ( MM1.eq.0 ) go to 130
887: C
888: C ... make index list IDWRK (cyclic boundary condition)
889: C
890: C      IDWRK(1:MM1) : particles in box (II,JJ,KK) (current)
891: C      IDWRK(MM1+1:MM) : particles in neiboring boxes
892: C                          having sequential box # less than
893: C                          that of current box.
894: C
895:      do M = 1, MM1
896:        IDXWK(M) = LINDX(M,II,JJ,KK)
897:      end do
898:
899:      MM = MM1
900:      IJK0 = II + NBOX*((JJ-1)+NBOX*(KK-1))
901:
902:      do KKK = KK - 1, KK + 1
903:        KKKK = KKK
904:        if ( KKK.gt.NBOX ) then
905:          KKKK = 1
906:        else if ( KKK.le.0 ) then
907:          KKKK = NBOX
908:        end if
909:        do JJJ = JJ - 1, JJ + 1
910:          JJJJ = JJJ

```

src/tools/mcrdf.f

```

911:         if ( JJJ.gt.NBOX ) then
912:             JJJJ = 1
913:         else if ( JJJ.le.0 ) then
914:             JJJJ = NBOX
915:         end if
916:         do III = II - 1, II + 1
917:             IIII = III
918:             if ( III.gt.NBOX ) then
919:                 IIII = 1
920:             else if ( III.le.0 ) then
921:                 IIII = NBOX
922:             end if
923:             IJK1 = IIII + NBOX*((JJJJ-1)+NBOX*(KKKK-1))
924:             if ( IJK1.lt.IJK0 ) then
925:                 do M = 1, NINDX(IIII,JJJJ,KKKK)
926:                     MM = MM + 1
927:                     IDXWK(MM) = LINDX(M,IIII,JJJJ,KKKK)
928:                 end do
929:             end if
930:         end do
931:     end do
932: end do
933: C
934: C DO 10 I=1,NNR
935: C DO 20 J=I+1,NNR
936: C
937: C do 120 III = 1, MM1
938: C I = IDXWK(III)
939: C do 110 JJJ = III + 1, MM
940: C J = IDXWK(JJJ)
941: C
942: C DLX = XXX(J) - XXX(I)
943: C DLY = YYY(J) - YYY(I)
944: C DLZ = ZZZ(J) - ZZZ(I)
945: C R = DLX*DLX + DLY*DLY + DLZ*DLZ
946: C IF ( R .LT. RAR2SQ ) THEN
947: C R = SQRT(R)
948: C R =
949: C & DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),
950: C & ZZZ(J),IIII)
951: C if ( R.lt.RAR2 ) then
952: C NP(I) = NP(I) + 1
953: C NP(J) = NP(J) + 1
954: C NNP(NP(I),I) = J
955: C NNP(NP(J),J) = I
956: C SDR = 2.*RAR - R
957: C DDR = DDR + SDR
958: C if ( SDR.lt.RMI ) then
959: C RMI = SDR
960: C else if ( SDR.gt.RMA ) then
961: C NOM = I
962: C RMA = SDR
963: C end if
964: C end if
965: C 110 continue
966: C ANL=ANL+REAL(NP(I))
967: C IF(NP(I).NE.0) NOL=NOL+1
968: C 120 continue
969: C
970: C 130 continue
971: C end do
972: C end do
973: C
974: C do I = 1, NNR
975: C ANL = ANL + REAL(NP(I))

```

```

976:         if ( NP(I).ne.0 ) NOL = NOL + 1
977:     end do
978: C
979: C if ( NOL.ne.0 ) then
980: C DDR = DDR/ANL/2./RAR
981: C ANL = ANL/REAL(NOL)
982: C WRITE(6,*) ' THE NUMBER OF PARTICLES HAVING OVERLAPPED : ',NOL
983: C WRITE(6,*) ' THE AVERAGE NUMBER OF OVERLAPPED PARTICLES : ',ANL
984: C WRITE(6,*) ' THE AVERAGE OVERLAPPED LENGTH : ',DDR,
985: C 1 ' *RAR*2.0'
986: C end if
987: C
988: C end
989: C
990: C=====
991: C
992: C function DIST(X1,Y1,Z1,X2,Y2,Z2,II)
993: C CALCULATE DISTANCE BETWEEN (X1,Y1,Z1) AND (X2,Y2,Z2)
994: C MODIFIED TO CONSIDER IN CASE OF FLYING TO THE OUTER REGION
995: C '92.4.8(I.M.)
996: C (X1,Y1,Z1) IS REFERENCE POINT.
997: C (X2,Y2,Z2) MAY BE CHANGED.
998: C
999: C common /INPDT/ PFC, AAR, IBOX
1000: C dimension IS(3), DL(3), P1(3), P2(3)
1001: C
1002: C P1(1) = X1
1003: C P1(2) = Y1
1004: C P1(3) = Z1
1005: C P2(1) = X2
1006: C P2(2) = Y2
1007: C P2(3) = Z2
1008: C AR2 = AAR/2.0
1009: C do 100 I = 1, 3
1010: C IS(I) = 0
1011: C DL(I) = P2(I) - P1(I)
1012: C if ( DL(I).gt.AR2 ) then
1013: C DL(I) = DL(I) - AAR
1014: C IS(I) = -1
1015: C else if ( DL(I).lt.-AR2 ) then
1016: C DL(I) = DL(I) + AAR
1017: C IS(I) = 1
1018: C end if
1019: C 100 continue
1020: C DIST = SQRT(DL(1)*DL(1)+DL(2)*DL(2)+DL(3)*DL(3))
1021: C II = 14 - 3*IS(1) + IS(2) + 9*IS(3)
1022: C
1023: C return
1024: C end
1025: C
1026: C=====
1027: C
1028: C function DIST2(X1,Y1,Z1,X2,Y2,Z2)
1029: C CALCULATE DISTANCE BETWEEN (X1,Y1,Z1) AND (X2,Y2,Z2)
1030: C '93.8.12
1031: C DIST2 = SQRT((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)+(Z1-Z2)*(Z1-Z2))
1032: C
1033: C return
1034: C end
1035: C
1036: C=====
1037: C
1038: C subroutine DRATE( ICV, DDR, DDRATE )
1039: C parameter( NAVE = 10 )
1040: C dimension DDROLD(NAVE)

```

src/tools/mcrdf.f

```

1041: C
1042: C ... DDROLD must be saved (M.Sasaki)
1043: C
1044: C data DDROLD /NAVE*0.0/
1045: C
1046: C if ( ICV.le.NAVE ) then
1047: C   DDROLD(ICV) = DDR
1048: C   DDRATE = 1.E10
1049: C else
1050: C   do 100 I = 1, NAVE - 1
1051: C     DDROLD(I) = DDROLD(I+1)
1052: C   continue
1053: C   DDROLD(NAVE) = DDR
1054: C   DDRATE = (DDROLD(1)-DDROLD(NAVE)) /FLOAT(NAVE)
1055: C end if
1056: C
1057: C return
1058: C end
1059: C
1060: C=====
1061: C
1062: C subroutine VIB
1063: C
1064: C   MOVE PARTICLES AT RANDOM
1065: C
1066: C   parameter( MPX = 1000 )
1067: C   common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
1068: C & RDF(MPX), RCP(3,MPX), RCP2(3,MPX), SIGT,
1069: C & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
1070: C & NNR
1071: C   common /OVLCM/ ANL, DDR, CCC, RMI, RMA,
1072: C & NNP(100,MPX), NP(MPX), MMP(100,MPX), MP,
1073: C & NOL, IAD, NOM, ICE
1074: C
1075: C   write(6,7000)
1076: C   7000 format(' DDR DECREASING RATE IS TOO SMALL. ',
1077: C & ' MOVE PARTICLES AT RANDOM.')
1078: C   DRAR = RAR/10.
1079: C   do 100 I = 1, NNR
1080: C     XXX(I) = XXX(I) + (RANG()-0.5)*DRAR
1081: C     YYY(I) = YYY(I) + (RANG()-0.5)*DRAR
1082: C     ZZZ(I) = ZZZ(I) + (RANG()-0.5)*DRAR
1083: C     call CKXYZ( XXX(I), YYY(I), ZZZ(I) )
1084: C   100 continue
1085: C
1086: C   return
1087: C end
1088: C
1089: C=====
1090: C
1091: C subroutine CKXYZ( X, Y, Z )
1092: C   CHECK THE CORDINATES, AND SETTLE IN THE BOX.
1093: C
1094: C   ... This routine counld not fix coordinates less than -1.5*AAR
1095: C   or greater than 1.5*AAR . Modified ( M.Sasaki )
1096: C
1097: C
1098: C   common /INPDT/ PFC, AAR, IBOX
1099: C   check %%%%%%%%%
1100: C   X0 = X
1101: C   Y0 = Y
1102: C   Z0 = Z
1103: C   %%%%%%%%%
1104: C
1105: C   IF(X.LT.-.5*AAR) X=X+AAR

```

```

1106: C   IF(X.GT..5*AAR) X=X-AAR
1107: C   IF(Y.LT.-.5*AAR) Y=Y+AAR
1108: C   IF(Y.GT..5*AAR) Y=Y-AAR
1109: C   IF(Z.LT.-.5*AAR) Z=Z+AAR
1110: C   IF(Z.GT..5*AAR) Z=Z-AAR
1111: C   AAR2 = AAR*0.5
1112: C   if ( ABS(X).gt.AAR2 ) then
1113: C     X1 = X + AAR2
1114: C     X = MOD(ABS(X1),AAR) - AAR2
1115: C     if ( X1.lt.0.0 ) X = -X
1116: C   end if
1117: C   if ( ABS(Y).gt.AAR2 ) then
1118: C     Y1 = Y + AAR2
1119: C     Y = MOD(ABS(Y1),AAR) - AAR2
1120: C     if ( Y1.lt.0.0 ) Y = -Y
1121: C   end if
1122: C   if ( ABS(Z).gt.AAR2 ) then
1123: C     Z1 = Z + AAR2
1124: C     Z = MOD(ABS(Z1),AAR) - AAR2
1125: C     if ( Z1.lt.0.0 ) Z = -Z
1126: C   end if
1127: C   check %%%%%%%%%
1128: C   if ( MAX(ABS(X/AAR),ABS(Y/AAR),ABS(Z/AAR)).gt.0.5 ) write(*,*)
1129: C & '%% CKXYZ ', X0/AAR, Y0/AAR, Z0/AAR, X/AAR, Y/AAR, Z/AAR
1130: C   %%%%%%%%%
1131: C
1132: C   return
1133: C end
1134: C
1135: C=====
1136: C
1137: C subroutine ADJUST( IC, IST )
1138: C
1139: C   ADJUST THE CORDINATES OF PARTICLES
1140: C
1141: C   parameter( MPX = 1000 )
1142: C
1143: C   common /INPDT/ PFC, AAR, IBOX
1144: C
1145: C   common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
1146: C & RDF(MPX), RCP(3,MPX), RCP2(3,MPX), SIGT,
1147: C & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
1148: C & NNR
1149: C   common /OVLCM/ ANL, DDR, CCC, RMI, RMA,
1150: C & NNP(100,MPX), NP(MPX), MMP(100,MPX), MP,
1151: C & NOL, IAD, NOM, ICE
1152: C   common /TRNCM/ XTR(27), YTR(27), ZTR(27)
1153: C
1154: C   dimension DIS(MPX), MM(100), ISS(MPX), IST(10)
1155: C
1156: C   ISTMX = 10
1157: C   do 100 I = 1, ISTMX
1158: C     IST(I) = 0
1159: C   100 continue
1160: C
1161: C   do 110 I = 1, NNR
1162: C     XNW(I) = XXX(I)
1163: C     YNW(I) = YYY(I)
1164: C     ZNW(I) = ZZZ(I)
1165: C   check %%%%%%%%%
1166: C   if ( MAX(ABS(XNW(I)/AAR),ABS(YNW(I)/AAR),ABS(ZNW(I)/AAR)).gt.
1167: C & 0.5 ) write(*,*) '%% adjst ', XNW(I) /AAR, YNW(I) /AAR,
1168: C & ZNW(I) /AAR
1169: C   %%%%%%%%%
1170: C   110 continue

```


src/tools/mcrdf.f

```

1171: C
1172: C      FOR OVERLAPPED PARTICLES
1173:       do 220 I = 1, NNR
1174:         U      = 0.
1175:         V      = 0.
1176:         W      = 0.
1177: C
1178:       if ( IC.ge.ICE.and.ICE.ne.0 ) then
1179: C
1180: CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC TEMPORALY ROUTINE
1181: CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC NORMALLY NOT USED
1182: C 11 CONTINUE
1183: C      RM=1.E10
1184: C      DO 23 J=1, NP(I)
1185: C      R=DIST(XXX(I),YYY(I),ZZZ(I),XXX(NNP(J,I)),YYY(NNP(J,I)),
1186: C 1 ZZZ(NNP(J,I)),III)
1187: C      IF(R.LT.RM) JR=J
1188: C      IF(R.LT.RM) IJ=III
1189: C 23 IF(R.LT.RM) RM=R
1190: C      U=(2.*RAR/RM-1.)*(XXX(I)-XXX(NNP(JR,I))-XTR(I,IJ))*0.5
1191: C      V=(2.*RAR/RM-1.)*(YYY(I)-YYY(NNP(JR,I))-YTR(I,IJ))*0.5
1192: C      W=(2.*RAR/RM-1.)*(ZZZ(I)-ZZZ(NNP(JR,I))-ZTR(I,IJ))*0.5
1193: C      C=1.
1194: C      IF(NP(I).EQ.1) C=CCC
1195: C      XNW(I)=XXX(I)+U*C
1196: C      YNW(I)=YYY(I)+V*C
1197: C      ZNW(I)=ZZZ(I)+W*C
1198: C      CALL CKXYZ(XNW(I),YNW(I),ZNW(I))
1199: C
1200: C      GOTO 10
1201: CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1202: CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1203: C
1204: C      SPECIAL TREATMENT FOR OVERLAPPED PARTICLE
1205: C
1206:       if ( IC.eq.ICE.and.I.eq.1 ) write(6, '(1H1)')
1207:       if ( IC.eq.ICE.and.I.eq.1 ) write(6,*)
1208:       &      ' ADJUST IS DONE BY SPECIAL TREATMENT.'
1209:       call OVLP2( I, RMI2, RMA2, DD2 )
1210:       if ( NP(I).ne.0. ) then
1211:         do 120 J = 1, NP(I)
1212:           R      = DIST(XNW(I),YNW(I),ZNW(I),XNW(NNP(J,I)),
1213:           &      YNW(NNP(J,I)),ZNW(NNP(J,I)),III)
1214:           U      = U + (2.*RAR/R-1.)*
1215:           &      (XNW(I)-XNW(NNP(J,I))-XTR(III))
1216:           V      = V + (2.*RAR/R-1.)*
1217:           &      (YNW(I)-YNW(NNP(J,I))-YTR(III))
1218:           W      = W + (2.*RAR/R-1.)*
1219:           &      (ZNW(I)-ZNW(NNP(J,I))-ZTR(III))
1220: 120 continue
1221: C      C      = 1.
1222:       if ( NP(I).eq.1 ) C = CCC
1223:       R      = REAL(NP(I))
1224:       U      = U/R
1225:       V      = V/R
1226:       W      = W/R
1227:       XNW(I) = XNW(I) + U*C
1228:       YNW(I) = YNW(I) + V*C
1229:       ZNW(I) = ZNW(I) + W*C
1230:       call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1231: C
1232: C      PARTICLE IS MOVED IN ORDER NOT TO EXCEED THE MAXIMUM OVERLAPPED
1233: C      LENGTH.
1234:       FAC      = 1.
1235:       do 130 JJ = 1, ISTMX - 1

```

```

1236:       call OVLP2( I, RMI3, RMA3, DD3 )
1237:       if ( RMA3.le.RMA2 ) IST(JJ) = IST(JJ) + 1
1238: C
1239: C      PROCESS FOR LOOSE PACKED PARTICLE
1240:       if ( RMA3.le.RMA2.and.JJ.eq.1 ) then
1241:         XNW(I) = XNW(I) - U*C*FAC*0.5
1242:         YNW(I) = YNW(I) - V*C*FAC*0.5
1243:         ZNW(I) = ZNW(I) - W*C*FAC*0.5
1244:         call OVLP2( I, RMI4, RMA4, DD4 )
1245:         if ( RMA3.lt.RMA4 ) then
1246:           XNW(I) = XNW(I) + U*C*FAC*0.5
1247:           YNW(I) = YNW(I) + V*C*FAC*0.5
1248:           ZNW(I) = ZNW(I) + W*C*FAC*0.5
1249:         end if
1250:       check %%%%%%%%%
1251:       c      write(*,*) '%% LOOSE PACKED ', XNW(I) /AAR, YNW(I)
1252:       c      &      /AAR, ZNW(I) /AAR
1253:       c      %%%%%%%%%
1254:       go to 220
1255:     end if
1256: C
1257:       if ( RMA3.le.RMA2 ) go to 220
1258:       FAC      = FAC/2.
1259:       XNW(I) = XNW(I) - U*C*FAC
1260:       YNW(I) = YNW(I) - V*C*FAC
1261:       ZNW(I) = ZNW(I) - W*C*FAC
1262:       call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1263: 130 continue
1264:       IST(ISTMX) = IST(JJ) + 1
1265: C
1266: C      PROCESS FOR CLOSE PACKED PARTICLE
1267:       do 140 J = 1, NP(I)
1268:         R      = DIST(XNW(I),YNW(I),ZNW(I),XNW(NNP(J,I)),
1269:         &      YNW(NNP(J,I)),ZNW(NNP(J,I)),III)
1270:         U      = (2.*RAR/R-1.)*
1271:         &      (XNW(I)-XNW(NNP(J,I))-XTR(III))*(-0.5)
1272:         V      = (2.*RAR/R-1.)*
1273:         &      (YNW(I)-YNW(NNP(J,I))-YTR(III))*(-0.5)
1274:         W      = (2.*RAR/R-1.)*
1275:         &      (ZNW(I)-ZNW(NNP(J,I))-ZTR(III))*(-0.5)
1276:         L      = NNP(J,I)
1277:         XNW(L) = XNW(L) + U
1278:         YNW(L) = YNW(L) + V
1279:         ZNW(L) = ZNW(L) + W
1280:         call CKXYZ( XNW(L), YNW(L), ZNW(L) )
1281: 140 continue
1282: C
1283:       go to 220
1284:     end if
1285: C
1286:       else if ( NP(I).ne.0. ) then
1287: C
1288: C      NORMAL TREATMENT FOR OVERLAPPED PARTICLES
1289: C
1290:       DRMAX      = 0.0
1291:       do 150 J = 1, NP(I)
1292:         R      = DIST(XXX(I),YYY(I),ZZZ(I),XXX(NNP(J,I)),
1293:         &      YYY(NNP(J,I)),ZZZ(NNP(J,I)),III)
1294:         DR      = 2.*RAR/R - 1.
1295: C980107 ..... if ( DR.gt.DRMAX ) DRMAX      = DR
1296:         if ( DR*R.gt.DRMAX ) DRMAX      = DR*R
1297:         U      = U + DR*(XXX(I)-XXX(NNP(J,I))-XTR(III))
1298:         V      = V + DR*(YYY(I)-YYY(NNP(J,I))-YTR(III))
1299:         W      = W + DR*(ZZZ(I)-ZZZ(NNP(J,I))-ZTR(III))
1300: 150 continue

```

src/tools/mcrdf.f

```

1301:      DV      = SQRT(U*U+V*V+W*W)
1302:      if ( DV.ne.0.0 ) then
1303:      C      = DRMAX/DV/2.
1304:      RR      = 0.1
1305:      U      = U*(1.0+(RANG()-0.5)*RR)
1306:      V      = V*(1.0+(RANG()-0.5)*RR)
1307:      W      = W*(1.0+(RANG()-0.5)*RR)
1308:      XNW(I) = XXX(I) + U*C
1309:      YNW(I) = YYY(I) + V*C
1310:      ZNW(I) = ZZZ(I) + W*C
1311:      call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1312:      end if
1313: C
1314:      go to 220
1315:      end if
1316: C
1317: C FOR NOT-OVERLAPPED PARTICLES
1318:      if ( IAD.eq.0 ) then
1319:      C      if ( IAD.eq.1 ) then
1320: C
1321: C      OPTION 1(IAD=1)
1322:      do 160 J = 1, NNR
1323:      C      DIS(J) =
1324:      &      DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),
1325:      &      ZZZ(J),III)
1326:      C      ISS(J) = III
1327:      160      continue
1328:      call SORT( DIS(1), NNR, MM(1), MP )
1329:      U2      = 0.
1330:      V2      = 0.
1331:      W2      = 0.
1332:      do 170 J = 1, MP
1333:      C      U2      = U2 + XXX(MM(J+1)) + XTR(ISS(MM(J+1)))
1334:      C      V2      = V2 + YYY(MM(J+1)) + YTR(ISS(MM(J+1)))
1335:      C      W2      = W2 + ZZZ(MM(J+1)) + ZTR(ISS(MM(J+1)))
1336:      170      continue
1337:      R      = REAL(MP)
1338:      U2      = U2/R
1339:      V2      = V2/R
1340:      W2      = W2/R
1341:      XNW(I) = U2
1342:      YNW(I) = V2
1343:      ZNW(I) = W2
1344:      call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1345:      else if ( IAD.eq.2 ) then
1346: C
1347: C      OPTION 2(IAD=2)
1348:      do 180 J = 1, NNR
1349:      C      DIS(J) =
1350:      &      DIST(XNW(I),YNW(I),ZNW(I),XNW(J),YNW(J),
1351:      &      ZNW(J),III)
1352:      C      ISS(J) = III
1353:      180      continue
1354:      call SORT( DIS(1), NNR, MM(1), MP )
1355:      U2      = 0.
1356:      V2      = 0.
1357:      W2      = 0.
1358:      do 190 J = 1, MP
1359:      C      U2      = U2 + XNW(MM(J+1)) + XTR(ISS(MM(J+1)))
1360:      C      V2      = V2 + YNW(MM(J+1)) + YTR(ISS(MM(J+1)))
1361:      C      W2      = W2 + ZNW(MM(J+1)) + ZTR(ISS(MM(J+1)))
1362:      190      continue
1363:      R      = REAL(MP)
1364:      U2      = U2/R
1365:      V2      = V2/R
1366:      W2      = W2/R
1367:      XNW(I) = U2
1368:      YNW(I) = V2
1369:      ZNW(I) = W2
1370:      call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1371:      else if ( IAD.eq.3 ) then
1372: C
1373: C      OPTION 3(IAD=3)
1374:      RL      = 1.E10
1375:      do 200 J = 1, NNR
1376:      C      if ( I.ne.J ) then
1377:      C      R      =
1378:      &      DIST(XNW(I),YNW(I),ZNW(I),XNW(J),YNW(J),
1379:      &      ZNW(J),III)
1380:      C      if ( R.lt.RL ) JL = J
1381:      C      if ( R.lt.RL ) RL = R
1382:      C      end if
1383:      200      continue
1384:      210      UU      = 2.*(RANG()-0.5)
1385:      VV      = 2.*(RANG()-0.5)
1386:      WW      = 2.*(RANG()-0.5)
1387:      if ( UU*UU+VV*VV+WW*WW.gt.1. ) go to 210
1388:      R      = SQRT(UU*UU+VV*VV+WW*WW)
1389:      UU      = UU/R
1390:      VV      = VV/R
1391:      WW      = WW/R
1392:      XNW(I) = (RL-2.*RAR)*UU + XNW(I)
1393:      YNW(I) = (RL-2.*RAR)*VV + YNW(I)
1394:      ZNW(I) = (RL-2.*RAR)*WW + ZNW(I)
1395:      call CKXYZ( XNW(I), YNW(I), ZNW(I) )
1396:      end if
1397:      end if
1398: C
1399:      220 continue
1400: C
1401: C      END OF ADJUSTMENT
1402: C      SET UP NEW COORDINATES (XNW,YNW,ZNW) TO (XXX,YYY,ZZ)
1403:      do 230 I = 1, NNR
1404:      C      check %%%%%%%%%
1405:      C      if ( MAX(ABS(XNW(I)/AAR),ABS(YNW(I)/AAR),ABS(ZNW(I)/AAR)).gt.
1406:      C      &      0.5 ) write(*,*) '%%% ????' , XNW(I)/AAR, YNW(I)/AAR,
1407:      C      &      ZNW(I)/AAR
1408:      C      %%%%%%%%%
1409:      XXX(I) = XNW(I)
1410:      YYY(I) = YNW(I)
1411:      ZZZ(I) = ZNW(I)
1412:      230 continue
1413:      return
1414:      end
1415: C
1416: C=====
1417: C
1418:      subroutine OVLP2( I,      RMI2, RMA2, DD2 )
1419:      C      FIND OVERLAPPED AND NOT-OVERLAPPED PARTICLE FOR EACH PARTICLE
1420: C
1421:      parameter( MPX = 1000 )
1422:      common /MAINCM/ XXX(MPX),      YYY(MPX),      ZZZ(MPX),
1423:      &      RDF(MPX),      RCP(3,MPX),      RCP2(3,MPX),      SIGT,
1424:      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
1425:      &      NNR
1426:      common /OVLPCM/ ANL,      DDR,      CCC,      RMI,      RMA,
1427:      &      NNP(100,MPX),      NP(MPX),      MMP(100,MPX),      MP,
1428:      &      NOL,      IAD,      NOM,      ICE
1429: C      FIND OVERLAPPED
1430:      RMI2      = 1.E10

```

src/tools/mcrdf.f

```

1431:      RMA2      = 0.
1432:      DD2       = 0.
1433:      NP(I)     = 0
1434:      do 100 J = 1, NNR
1435:        if ( I.ne.J ) then
1436:          R      =
1437:          &      DIST(XNW(I),YNW(I),ZNW(I),XNW(J),YNW(J),ZNW(J),III)
1438:          if ( R.lt.RAR*2. ) then
1439:            NP(I) = NP(I) + 1
1440:            NNP(NP(I),I) = J
1441:            DD2    = DD2 + 2.*RAR - R
1442:            if ( 2.*RAR-R.lt.RMI2 ) RMI2 = 2.*RAR - R
1443:            if ( 2.*RAR-R.gt.RMA2 ) RMA2 = 2.*RAR - R
1444:          end if
1445:        end if
1446:      100 continue
1447:      if ( NP(I).eq.0 ) return
1448:      DD2 = DD2/REAL(NP(I))
1449:      return
1450:      end
1451: C
1452: C=====
1453: C
1454:      subroutine CRDF
1455: C      CALCULATE RADIAL DISTRIBUTION FUNCTION
1456: C      VORONOI CELL STATISTICS CAN BE CALCULATED ('93.7.3)
1457: C
1458:      parameter( MPX = 1000 )
1459:      common /INPDT/ PFC, AAR, IBOX
1460:      common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
1461:      &      RDF(MPX), RCP(3,MPX), RCP2(3,MPX), SIGT,
1462:      &      XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
1463:      &      NNR
1464:      common /CRDFCM/ NNFC(MPX), NFACE(20,MPX), NFFT(20),
1465:      &      NF3(4,1000)
1466: C
1467: C      NNFC :THE NUMBER OF EDGES WHICH EACH PARTICLE HAS
1468: C      NFACE:ALL PARTICLE INDICES WHICH EACH PARTICLE HAS
1469: C      NFFT :DISTRIBUTION OF THE NUMBER OF EDGES PER FACE
1470: C      NF3  :FOUR PARTICLES' INDICES CONTACTING ONE ANOTHER
1471: C
1472:      check %%%%%%%%%%%%%%
1473:      c      write(6,*) '%%% PFC RAR AAR ',PFC,RAR,AAR
1474:      c %%%%%%%%%%%%%%
1475:      NRMX = INT(AAR/2./RAR/2.)
1476:      NGMX = 20
1477:      DEL  = RAR*2./REAL(NGMX)
1478:      FTL  = 0.01
1479:      check %%%%%%%%%%%%%%
1480:      c      write(6,*) '%%% NRMX DEL NNR ',NRMX,DEL,NNR
1481:      c %%%%%%%%%%%%%%
1482:      C      IS TOLERANCE TO DETERMINE CONTACT TWO PARTICLES OR NOT
1483:      C
1484:      do 100 I = 1, 20
1485:        NFFT(I) = 0
1486:      100 continue
1487: C
1488:      NNR2 = NNR
1489:      if ( IBOX.ne.0 ) NNR2 = 1
1490:      do 140 I = 1, NNR2
1491:        NNFC(I) = 0
1492:        do 130 J = 1, NNR
1493:          R =
1494:          &      DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),ZZZ(J),III)
1495:          if ( R.le.RAR*2.*REAL(NRMX) ) then

```

```

1496:      CCCC IF(I.EQ.J) GOTO 40
1497:      C
1498:      C      xxxx jumping to 40 without fixing K !!! (Nov 1998 M.Sasaki)
1499:      C
1500:      C      IF(I.EQ.J) then
1501:      C      K = 1
1502:      C      GOTO 40
1503:      C      endif
1504:      C
1505:      C      xxxx skipping the loop is the real solution ???
1506:      C
1507:      C      if ( I.ne.J ) then
1508:      C
1509:      C      if ( R-RAR*2..lt.RAR*2.*FTL ) then
1510:      C      NNFC(I) = NNFC(I) + 1
1511:      C      NFACE(NNFC(I),I) = J
1512:      C      end if
1513:      C      do 110 K = 1, NRMX*NGMX
1514:      C      if ( R.le.DEL*REAL(K) ) go to 120
1515:      C      110 continue
1516:      C      check %%%%%%
1517:      C      40 continue
1518:      C      write(*,*) '%%% K NRMX NGMX R DEL ', K,NRMX,NGMX,R,DEL
1519:      C      RDF(K)=RDF(K)+1.
1520:      C %%%%%%%%%%%%%%
1521:      C      120      RDF(K) = RDF(K) + 1.
1522:      C      end if
1523:      C      end if
1524:      C      130 continue
1525:      C      NFFT(NNFC(I)) = NFFT(NNFC(I)) + 1
1526:      C      140 continue
1527:      C
1528:      C      FIND FOUR PARTICLES CONTACTING ONE ANOTHER.
1529:      C
1530:      C      INF3 = 0
1531:      C      INF4 = 0
1532:      C      do 230 I = 1, NNR
1533:      C      do 220 J = 1, NNFC(I)
1534:      C      I2 = NFACE(J,I)
1535:      C      do 210 K = 1, NNFC(I2)
1536:      C      I3 = NFACE(K,I2)
1537:      C      if ( I3.ne.I ) then
1538:      C      do 200 L = 1, NNFC(I3)
1539:      C      I4 = NFACE(L,I3)
1540:      C      if ( I4.ne.I2 ) then
1541:      C      if ( I4.eq.I ) then
1542:      C      INF3 = INF3 + 1
1543:      C      do 190 M1 = 1, NNFC(I)
1544:      C      J2 = NFACE(M1,I)
1545:      C      if ( J2.ne.I2.and.J2.ne.I3 ) then
1546:      C      do 150 M2 = 1, NNFC(I2)
1547:      C      J3 = NFACE(M2,I2)
1548:      C      if ( J2.eq.J3 ) go to 160
1549:      C      150 continue
1550:      C      end if
1551:      C      go to 190
1552:      C      160 do 170 M3 = 1, NNFC(I3)
1553:      C      J4 = NFACE(M3,I3)
1554:      C      if ( J2.eq.J4 ) go to 180
1555:      C      170 continue
1556:      C      go to 190
1557:      C      180 INF4 = INF4 + 1
1558:      C      if ( INF4.gt.1000 ) then
1559:      C      INF4 = 1000
1560:      C      else

```

src/tools/mcrdf.f

```

1561:          NF3(1,INF4) = I
1562:          NF3(2,INF4) = I2
1563:          NF3(3,INF4) = I3
1564:          NF3(4,INF4) = J2
1565:          end if
1566: 190          continue
1567:          end if
1568:          end if
1569: 200          continue
1570:          end if
1571: 210          continue
1572: 220          continue
1573: 230          continue
1574: C
1575: C      OUTPUT
1576: write(6, '(1H1)')
1577: write(6,*) ' NO. OF EDGES DISTANCE TOTAL NO. RDF IRDF/NNR '
1578: do 240 I = 1, NRMX*NGMX
1579:   RS = REAL(I-1)/REAL(NGMX)
1580:   RE = REAL(I)/REAL(NGMX)
1581:   IRDF = INT(RDF(I))
1582:   RDF(I) = RDF(I)*RAR*RAR*RAR/REAL(NNR2) /(RE*RE*RE-RS*RS*RS)
1583:   write(6,7000) I, RS, RE, IRDF, RDF(I), REAL(IRDF)/REAL(NNR2)
1584: 240 continue
1585: 7000 format(' ',I4,F6.3,' - ',F6.3,I6,2X,F7.4,2X,F7.4)
1586: C
1587: write(6, '(1H1)')
1588: write(6,*) 'DISTRIBUTION OF PARTICLES WHICH HAVE THE SAME NUMBER',
1589: & ' OF EDGES.'
1590: write(6,*) ' NO. OF EDGES DISTRIBUTION'
1591: do 250 I = 1, 20
1592:   write(6,7020) I, NFFT(I)
1593: 250 continue
1594: 7020 format(' ',5X,I4,7X,I8)
1595: C
1596: write(6, '(1H1)')
1597: write(6,*) 'THREE PAIRS CONTACTING ONE ANOTHER.'
1598: write(6,*) ' (VORONOI CELL STATISTICS)'
1599: write(6,*) ' TOTAL ', INF3
1600: write(6, '(1H0)')
1601: write(6,*) 'FOUR PAIRS CONTACTING ONE ANOTHER.'
1602: write(6,*) ' (VORONOI CELL STATISTICS)'
1603: write(6,*) ' TOTAL ', INF4
1604: write(6,*) ' ALL PAIRS ARE FOLLOWS.'
1605: do 260 I = 1, INF4
1606:   write(6, '(1H ,I4,4I6)') I, NF3(1,I), NF3(2,I), NF3(3,I),
1607: & NF3(4,I)
1608: 260 continue
1609: return
1610: end
1611: C
1612: C=====
1613: C
1614: subroutine CRCP
1615: C      CALCULATE RADIAL COLLISION PROBABILITY DENSITY
1616: C      START PARTICLE FROM ON THE SPHERE
1617: C
1618: C      OUTPUT FILE : FT09; RCP OUTPUT FOR MCNP-CFP
1619: C      FT07; RCP OUTPUT FOR PLOT
1620: C      FT08; ANGULAR DISTRIBUTION
1621: C
1622: parameter( NRY = 1000, MPX = 1000 )
1623: parameter( PI = 3.141592653 )
1624: common /INPDT/ PFC, AAR, IBOX
1625: common /TRNCM/ XTR(27), YTR(27), ZTR(27)

```

```

1626: common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
1627: & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
1628: & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
1629: & NNR
1630: common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
1631: & ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
1632: C
1633: dimension CCN(MPX,20)
1634: C
1635: NDEB = 0
1636: NCO = 0
1637: NRMX = INT(AAR/2./RAR/2.)
1638: NGMX = 20
1639: DEL = RAR*2./REAL(NGMX)
1640: NRYU = NRY
1641: if ( NRY.gt.NNR ) NRYU = NNR
1642: C
1643: C      THE MAXIMUM NUMBER OF SAMPLING
1644: NMAX = 1000000
1645: C
1646: write(6, '(1H1)')
1647: write(6,*) ' SORT START IN CRCP'
1648: call CLOCKM( MMM1 )
1649: do 120 I = 1, NNR
1650:   do 100 J = 1, NNR
1651:     DIS(J) =
1652: & DIST(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),ZZZ(J),III)
1653:     ISS(J) = III
1654: 100 continue
1655: call SORT( DIS(1), NNR, IDD(1), NRYU )
1656: C      WRITE(6, '(1H ,6I6)') I, (IDD(L),L=1,5)
1657: do 110 K = 1, NRYU
1658:   ISIS(K,I) = ISS(IDD(K))
1659:   IDIS(K,I) = IDD(K)
1660: 110 continue
1661: 120 continue
1662: call CLOCKM( MMM2 )
1663: write(6,*) ' SORTING TIME IS ', REAL(MMM2-MMM1)/1000.,
1664: & '(SEC)'
1665: C
1666: do 220 I = 1, NMAX
1667: C      SAMPLE REFERENCE PARTICLE
1668: 130 IP = INT(1.+RANG()*REAL(NNR))
1670: C
1671: C      CHECK IP
1672: if ( IP.lt.1 .or. IP.gt.NNR ) then
1673:   write(6,*) ' IP IS INVALID. REJECT IP. IP=', IP
1674:   go to 130
1675: end if
1676: if ( I.le.NDEB ) write(6, '(1H1)')
1677: if ( I.le.NDEB ) write(6,*) ' IP=', IP
1678: C
1679: C      SAMPLE POINT ON THE SPHERE
1680: 140 UU = RANG()*2. - 1.
1681: VV = RANG()*2. - 1.
1682: WW = RANG()*2. - 1.
1683: R = SQRT(UU*UU+VV*VV+WW*WW)
1684: if ( R.gt.1. ) go to 140
1685: UU = UU/R
1686: VV = VV/R
1687: WW = WW/R
1688: XX = UU*RAR + XXX(IP)
1689: YY = VV*RAR + YYY(IP)
1690: ZZ = WW*RAR + ZZZ(IP)

```

src/tools/mcrdf.f

```

1691:      if ( I.le.NDEB ) write(6,*) ' UU=', UU, ' VV=', VV, ' WW=', WW
1692:      if ( I.le.NDEB ) write(6,*) ' XX=', XX, ' YY=', YY, ' ZZ=', ZZ
1693: C
1694: C      SAMPLE TRACKING DIRECTION
1695: C 40 TH=ASIN(RANG())
1696:      TH = ACOS(SQRT(RANG()))
1697:      PHI = 2.*PI*(RANG()-0.5)
1698:      U = COS(TH)
1699:      V = SIN(TH)*SIN(PHI)
1700:      W = SIN(TH)*COS(PHI)
1701:      R = SQRT(UU*UU+VV*VV+WW*WW)
1702:      E1 = UU/R
1703:      E2 = VV/R
1704:      E3 = WW/R
1705:      S = SQRT(E1*E1+E2*E2)
1706:      if ( S.ne.0.0 ) then
1707:          DXD = E1*U - E2/S*V - E1*E3/S*W
1708:          DYD = E2*U + E1/S*V - E2*E3/S*W
1709:          DZD = E3*U + S*W
1710:      else if ( E3.ge.0. ) then
1711:          DXD = -W
1712:          DYD = V
1713:          DZD = U
1714:      else
1715:          DXD = W
1716:          DYD = V
1717:          DZD = -U
1718:      end if
1719:      U = DXD
1720:      V = DYD
1721:      W = DZD
1722:      if ( I.le.NDEB ) write(6,*) ' UU*U+VV*V+WW*W=', UU*U + VV*V
1723:      & + WW*W
1724:      if ( I.le.NDEB ) write(6,*) ' U=', U, ' V=', V, ' W=', W
1725: C
1726: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
1727:      do 150 J = 1, NRYU
1728:          if ( I.le.NDEB ) write(6,*) ' I=', I, ' J=', J, ' IP=', IP,
1729:          & ' IDIS(J,IP)=', IDIS(J,IP), ' ISIS(J,IP)=', ISIS(J,IP)
1730:          X2 = XXX(IDIS(J,IP)) + XTR(ISIS(J,IP))
1731:          Y2 = YYY(IDIS(J,IP)) + YTR(ISIS(J,IP))
1732:          Z2 = ZZZ(IDIS(J,IP)) + ZTR(ISIS(J,IP))
1733:          T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
1734:          if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2, ' Z2=',
1735:          & Z2
1736:          if ( I.le.NDEB ) write(6,*) ' T=', T
1737:          if ( T.gt.0. ) then
1738:              X = T*U + XX
1739:              Y = T*V + YY
1740:              Z = T*W + ZZ
1741:              D = SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
1742:              if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=', Z
1743:              if ( I.le.NDEB ) write(6,*) ' D=', D
1744:              if ( D.lt.RAR ) go to 160
1745:          end if
1746: 150 continue
1747:      NCO = NCO + 1
1748:      if ( I.le.NDEB ) write(6,*) ' WARNING. NO COLLISION OCCUR.'
1749:      go to 220
1750: C
1751: C      THE TRACKED PARTICLE IS DETERMINED
1752: C
1753: C      P(1) : DISTANCE BETWEEN CENTER AND CENTER
1754: C      P(2) : DISTANCE BETWEEN SURFACE AND CENTER
1755: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE

```

```

1756: C
1757: 160 P(1) = SQRT((XXX(IP)-X2)*(XXX(IP)-X2)+(YYY(IP)-Y2)*
1758:      & (YYY(IP)-Y2)+(ZZZ(IP)-Z2)*(ZZZ(IP)-Z2))
1759:      P(2) = SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*(ZZ-Z2))
1760:      P(3) = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
1761:      & -Z)) - SQRT(RAR*RAR-D*D)
1762:      if ( I.le.NDEB ) write(6,*) ' P(1)=', P(1)
1763:      if ( I.le.NDEB ) write(6,*) ' P(2)=', P(2)
1764:      if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
1765: C
1766: C      CALCULATE COSINE INSERTING THE PARTICLE
1767:      XP = XX + U*P(3)
1768:      YP = YY + V*P(3)
1769:      ZP = ZZ + W*P(3)
1770:      U1 = X2 - XP
1771:      V1 = Y2 - YP
1772:      W1 = Z2 - ZP
1773:      R1 = SQRT(U1*U1+V1*V1+W1*W1)
1774:      U1 = U1/R1
1775:      V1 = V1/R1
1776:      W1 = W1/R1
1777:      CSN = U*U1 + V*V1 + W*W1
1778:      if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
1779: C
1780:      do 210 J = 1, 3
1781:          R = P(J)
1782:          if ( R.le.RAR*2.*REAL(NRMX) ) then
1783:              do 170 K = 1, NRMX*NGMX
1784:                  if ( R.le.DEL*REAL(K) ) go to 180
1785:                  continue
1786: 180 RCP(J,K) = RCP(J,K) + 1.
1787:                  if ( J.eq.3 ) then
1788:                      do 190 L = 1, 20
1789:                          if ( CSN.le..05*REAL(L) ) go to 200
1790:                          continue
1791: 200 CCN(K,L) = CCN(K,L) + 1.
1792:                      end if
1793:                  end if
1794:                  continue
1795: C
1796: C      END OF HISTORIES
1797: 220 continue
1798: C
1799: C      OUTPUT RADIAL COLLISION PROBABILITY
1800: C      UNIT 9 : FOR MCNP.CFP
1801:      write(9,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
1802: C
1803:      write(6,'(1H1)')
1804:      write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
1805:      write(6,*) ' REJECTION RATE : ', REAL(NCO) /REAL(NMAX)
1806:      write(6,*) ' NO. : ', NCO
1807:      write(6,'(1H0)')
1808:      write(6,*) ' NO. DISTANCE IRCP(1) IRCP(2) IRCP(3)',
1809:      & ' RCP(1) RCP(2) RCP(3)'
1810:      do 230 I = 1, NRMX*NGMX
1811:          RS = REAL(I-1) /REAL(NGMX)
1812:          RE = REAL(I) /REAL(NGMX)
1813:          if ( I.eq.1 ) then
1814:              RCP1 = RCP(1,I)
1815:              RCP2 = RCP(2,I)
1816:              RCP3 = RCP(3,I)
1817:          else
1818:              RCP1 = RCP(1,I) + RCP1
1819:              RCP2 = RCP(2,I) + RCP2
1820:              RCP3 = RCP(3,I) + RCP3

```

src/tools/mcrdf.f

```

1821:      end if
1822:      FAC      = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
1823:      RCP1J    = RCP(1,I)*FAC
1824:      RCP2J    = RCP(2,I)*FAC
1825:      RCP3J    = RCP(3,I)*FAC
1826:      RCP1I    = RCP1/REAL(NMAX)
1827:      RCP2I    = RCP2/REAL(NMAX)
1828:      RCP3I    = RCP3/REAL(NMAX)
1829:      write(7,'(1P2E12.5)') (RS+RE) /2., RCP3I
1830:      write(6,7000) I, RS, RE, INT(RCP1), INT(RCP2), INT(RCP3),
1831:      &      RCP1I, RCP2I, RCP3I, INT(RCP(1,I)), INT(RCP(2,I)),
1832:      &      INT(RCP(3,I)), RCP1J, RCP2J, RCP3J
1833: 7000    format(' ',I4,F6.3,' - ',F6.3,4X,3I6,4X,F7.4,2X,F7.4,2X,F7.4,
1834:      &      4X,3I6,4X,F7.4,2X,F7.4,2X,F7.4)
1835: C
1836:      write(9,'(1PE12.5)') RCP3I
1837: 230 continue
1838: C
1839: C      OUTPUT ANGULAR DISTRIBUTION
1840:      write(6,'(1H1)')
1841:      write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
1842:      write(6,*)
1843:      write(6,*) '          COSINE BINS'
1844:      write(6,*) ' GROUP      COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
1845:      do 240 I1 = 1, 20
1846:          write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
1847: 240 continue
1848: 7020 format(' ',I6,6X,2F10.7)
1849:      write(6,'(1H0)')
1850:      write(6,*) ' NO.          DISTANCE      ANG. DIST.(DP/DT)'
1851:      do 250 I = 1, NRMX*NGMX
1852:          RS      = REAL(I-1) /REAL(NGMX)
1853:          RE      = REAL(I) /REAL(NGMX)
1854:          if ( RCP(3,I).le.0 ) then
1855:              write(6,*) I, ' : RCP IS 0'
1856:              go to 250
1857:          end if
1858:          write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
1859:          write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
1860: 250 continue
1861: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
1862: 7060 format(' ',23X,1P10E10.3)
1863: C
1864: C      OUTPUT OF ANGULAR DISTRIBUTION TO FILE 8
1865:      II      = (NRMX*NGMX+4) /5
1866:      do 260 L = 1, NRMX*NGMX
1867:          if ( RCP(3,L).eq.0 ) RCP(3,L) = 1.
1868: 260 continue
1869:      do 280 L = 1, II
1870:          L1      = 5
1871:          if ( L.eq.II ) L1 = MIN(5,NRMX*NGMX-5*(II-1))
1872:          do 270 M = 1, 20
1873:              write(8,'(1P6E10.3)') .025 + .05*REAL(M-1),
1874:              &      (CCN(M1,M)/RCP(3,M1)/.05,M1=5*(L-1)+1,5*L)
1875: 270 continue
1876: 280 continue
1877: C
1878:      return
1879:      end
1880: C
1881: C=====
1882: C
1883:      subroutine CRCP2
1884: C      CALCULATE RADIAL COLLISION PROBABILITY DENSITY
1885: C      START PARTICLE FROM THE CENTER

```

```

1886: C
1887: C      OUTPUT FILE : FT04; RCP2 OUTPUT FOR PLOT
1888: C
1889:      parameter( NRY = 1000, MPX = 1000 )
1890:      common /INPDT/ PFC, AAR, IBOX
1891:      common /TRNCM/ XTR(27), YTR(27), ZTR(27)
1892:      common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
1893:      &      RDF(MPX), RCP(3,MPX), RCP2(3,MPX), SIGT,
1894:      &      XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
1895:      &      NNR
1896:      common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
1897:      &      ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
1898: C
1899:      NDEB      = 0
1900:      NCO      = 0
1901:      NRMX      = INT(AAR/2./RAR/2.)
1902:      NGMX      = 20
1903:      DEL      = RAR*2./REAL(NGMX)
1904:      NRYU      = NRY
1905:      if ( NRY.gt.NNR ) NRYU = NNR
1906: C
1907: C      MAXIMUM SAMPLING TIMES
1908:      NMAX      = 100000
1909: C
1910:      do 170 I = 1, NMAX
1911: C
1912: C      SAMPLE REFERENCE PARTICLE
1913: 100      IP      = INT(1.+RANG()*REAL(NNR))
1914: C
1915: C      CHECK IP
1916:      if ( IP.lt.1 .or. IP.gt.NNR ) then
1917:          write(6,*) ' IP IS INVALID. REJECT IP. IP=', IP
1918:          go to 100
1919:      end if
1920:      if ( I.le.NDEB ) write(6,'(1H1)')
1921:      if ( I.le.NDEB ) write(6,*) ' IP=', IP
1922: C
1923: C      SAMPLE POINT OF THE CENTER OF THE PARTICLE
1924:      XX      = XXX(IP)
1925:      YY      = YYY(IP)
1926:      ZZ      = ZZZ(IP)
1927:      if ( I.le.NDEB ) write(6,*) ' XX=', XX, ' YY=', YY, ' ZZ=', ZZ
1928: C
1929: C      SAMPLE TRACKING DIRECTION
1930: 110      U      = RANG()*2. - 1.
1931:      V      = RANG()*2. - 1.
1932:      W      = RANG()*2. - 1.
1933:      R      = SQRT(U*U+V*V+W*W)
1934:      if ( R.gt.1. ) go to 110
1935:      U      = U/R
1936:      V      = V/R
1937:      W      = W/R
1938:      if ( I.le.NDEB ) write(6,*) ' U=', U, ' V=', V, ' W=', W
1939: C
1940: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
1941:      do 120 J = 1, NRYU
1942:          if ( I.le.NDEB ) write(6,*) ' I=', I, ' J=', J, ' IP=', IP,
1943:          &      ' IDIS(J,IP)=', IDIS(J,IP), ' ISIS(J,IP)=', ISIS(J,IP)
1944:          X2      = XXX(IDIS(J,IP)) + XTR(ISIS(J,IP))
1945:          Y2      = YYY(IDIS(J,IP)) + YTR(ISIS(J,IP))
1946:          Z2      = ZZZ(IDIS(J,IP)) + ZTR(ISIS(J,IP))
1947:          T      = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
1948:          if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2, ' Z2=',
1949:          &      Z2
1950:          if ( I.le.NDEB ) write(6,*) ' T=', T

```

src/tools/mcrdf.f

```

1951:         if ( T.gt.0. ) then
1952:             X       = T*U + XX
1953:             Y       = T*V + YY
1954:             Z       = T*W + ZZ
1955:             D       = SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
1956:             if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=', Z
1957:             if ( I.le.NDEB ) write(6,*) ' D=', D
1958:             if ( D.lt.RAR ) go to 130
1959:         end if
1960: 120    continue
1961:         NCO       = NCO + 1
1962:         if ( I.le.NDEB ) write(6,*) ' WARNING. NO COLLISION OCCUR.'
1963:         go to 170
1964: C
1965: C      THE TRACKED PARTICLE IS DETERMINED
1966: C
1967: C      P(1) : DISTANCE BETWEEN CENTER AND CENTER
1968: C      P(2) : DISTANCE BETWEEN CENTER AND SURFACE
1969: C
1970: 130    P(1)       = SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*(ZZ-Z2))
1971:         P(2)       = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
1972: &                -Z)) - SQRT(RAR*RAR-D*D)
1973:         if ( I.le.NDEB ) write(6,*) ' P(1)=', P(1)
1974:         if ( I.le.NDEB ) write(6,*) ' P(2)=', P(2)
1975: C
1976:         do 160 J = 1, 2
1977:             R       = P(J)
1978:             if ( R.le.RAR*2.*REAL(NRMX) ) then
1979:                 do 140 K = 1, NRMX*NGMX
1980:                     if ( R.le.DEL*REAL(K) ) go to 150
1981: 140                continue
1982: 150                RCP2(J,K) = RCP2(J,K) + 1.
1983:             end if
1984: 160    continue
1985: C
1986: C      END OF HISTORIES
1987: 170    continue
1988: C
1989: C      OUTPUT RADIAL COLLISION PROBABILITY
1990:         write(6,'(1H1)')
1991:         write(6,*) ' PARTICLES START FROM THE CENTER.'
1992:         write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
1993:         write(6,*) ' REJECTION RATE : ', REAL(NCO) /REAL(NMAX)
1994:         write(6,*) ' NO. : ', NCO
1995:         write(6,'(1H0)')
1996:         write(6,*) ' NO.      DISTANCE      IRCP2(1)  IRCP2(2) ',
1997: &                ' RCP2(1)  RCP2(2) '
1998:         do 180 I = 1, NRMX*NGMX
1999:             RS       = REAL(I-1) /REAL(NGMX)
2000:             RE       = REAL(I) /REAL(NGMX)
2001:             if ( I.eq.1 ) then
2002:                 RDP1  = RCP2(1,I)
2003:                 RDP2  = RCP2(2,I)
2004:             else
2005:                 RDP1  = RCP2(1,I) + RDP1
2006:                 RDP2  = RCP2(2,I) + RDP2
2007:             end if
2008:             FAC      = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
2009:             RCP1J    = RCP2(1,I)*FAC
2010:             RCP2J    = RCP2(2,I)*FAC
2011:             RCP1I    = RDP1/REAL(NMAX)
2012:             RCP2I    = RDP2/REAL(NMAX)
2013:             write(4,'(1P2E12.5)') (RS+RE) /2., RCP2I
2014:             write(6,7000) I, RS, RE, INT(RDP1), INT(RDP2), RCP1I, RCP2I,
2015: &                INT(RCP2(1,I)), INT(RCP2(2,I)), RCP1J, RCP2J

```

```

2016: 180    continue
2017: 7000 format(' ',I4,F6.3,' - ',F6.3,4X,2I6,4X,F7.4,2X,F7.4,4X,2I6,4X,
2018: &          F7.4,2X,F7.4)
2019: C
2020:         return
2021:         end
2022: C
2023: C=====
2024: C
2025:         subroutine NND
2026: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
2027: C      START PARTICLE FROM ON THE SPHERE
2028: C
2029: C      EXTENDED NND CALCULATION OPTION WAS ADDED.
2030: C      THIS PROGRAM IS ALMOST THE SAME AS CRCP EXCEPT EXTENDED
2031: C      NND CALCULATION OPTION.
2032: C
2033: C      OUTPUT FILE : FT03; NND OUTPUT FOR PLOT
2034: C      FT12; NND OUTPUT FOR MCNP-CFP
2035: C
2036:         parameter( NRY = 1000, MPX = 1000 )
2037:         parameter( PI = 3.141592653 )
2038:         common /INPDT/ PFC, AAR, IBOX
2039:         common /TRNCM/ XTR(27), YTR(27), ZTR(27)
2040:         common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
2041: &                RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
2042: &                XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
2043: &                NNR
2044:         common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
2045: &                ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
2046: C
2047:         dimension CCN(MPX,100)
2048: C      DIMENSION FOR EXTENDED NND CALCULATION
2049:         dimension ISSS(NRY), IDSS(NRY), NCO(100)
2050: C      DIMENSION IDSS(NRY),NCO(100)
2051: C
2052:         NDEB      = 0
2053:         NC         = 0
2054:         do 100 I = 1, 100
2055:             NCO(I) = 0
2056: 100    continue
2057:         do 110 I = 1, MPX
2058:             RCP(3,I) = 0.
2059: 110    continue
2060: C      MAXIMUM LENGTH OF NND
2061:         DDDT      = 30.0
2062:         NRMX      = INT(DDDT/RAR/2.+0.001)
2063: C      NRMX=INT(AAR/2./RAR/2.)
2064:         NGMX      = 20
2065:         DEL       = RAR*2./REAL(NGMX)
2066:         NRYU      = NRY
2067:         if ( NRY.gt.NNR ) NRYU = NNR
2068: C
2069: C      THE MAXIMUM NUMBER OF SAMPLING
2070:         NMAX      = 200000
2071: C
2072: C      CALCULATE THE LENGTH AND SORTING
2073: C      DO 10 I=1,NNR
2074: C      DO 20 J=1,NNR
2075: C 20 DIS(J)=DIST2(XXX(I),YYY(I),ZZZ(I),XXX(J),YYY(J),ZZZ(J))
2076: C 20 ISS(J)=III
2077: C      CALL SORT(DIS(1),NNR,IDD(1),NRYU)
2078: C      WRITE(6,'(1H ,6I6)') I,(IDD(L),L=1,5)
2079: C      DO 130 K=1,NRYU
2080: C      ISIS(K,I)=ISS(IDD(K))

```

src/tools/mcrdf.f

```

2081: C 130 IDIS(K,I)=IDD(K)
2082: C 10 CONTINUE
2083: C
2084:      do 260 I = 1, NMAX
2085: C
2086:          DDD      = 0.
2087:          ILV      = 1
2088: C
2089: C      SAMPLE REFERENCE PARTICLE
2090: 120      IP      = INT(1.+RANG()*REAL(NNR))
2091: C
2092: C      CHECK IP
2093:      if ( IP.lt.1 .or. IP.gt.NNR ) then
2094:          write(6,*) ' IP IS INVALID REJECT IP. IP=', IP
2095:          go to 120
2096:      end if
2097:      if ( I.le.NDEB ) write(6, '(1H1)')
2098:      if ( I.le.NDEB ) write(6,*) ' IP=', IP
2099: C
2100: C      SAMPLE POINT ON THE SPHERE
2101: 130      UU      = RANG()*2. - 1.
2102:          VV      = RANG()*2. - 1.
2103:          WW      = RANG()*2. - 1.
2104:          R      = SQRT(UU*UU+VV*VV+WW*WW)
2105:      if ( R.gt.1. ) go to 130
2106:          UU      = UU/R
2107:          VV      = VV/R
2108:          WW      = WW/R
2109:          XX      = UU*RAR + XXX(IP)
2110:          YY      = VV*RAR + YYY(IP)
2111:          ZZ      = WW*RAR + ZZZ(IP)
2112:      if ( I.le.NDEB ) write(6,*) ' UU=', UU, ' VV=', VV, ' WW=', WW
2113:      if ( I.le.NDEB ) write(6,*) ' XX=', XX, ' YY=', YY, ' ZZ=', ZZ
2114: C
2115: C      SAMPLE TRACKING DIRECTION
2116: 40      TH=ASIN(RANG())
2117: C      ANGLE OPTION
2118:          TH      = ACOS(SQRT(RANG()))
2119: C      TH=0.
2120: C      TH=PI*0.4999
2121: C
2122:          PHI      = 2.*PI*(RANG()-0.5)
2123:          U      = COS(TH)
2124:          V      = SIN(TH)*SIN(PHI)
2125:          W      = SIN(TH)*COS(PHI)
2126:          R      = SQRT(UU*UU+VV*VV+WW*WW)
2127:          E1      = UU/R
2128:          E2      = VV/R
2129:          E3      = WW/R
2130:          S      = SQRT(E1*E1+E2*E2)
2131:      if ( S.ne.0.0 ) then
2132:          DXD      = E1*U - E2/S*V - E1*E3/S*W
2133:          DYD      = E2*U + E1/S*V - E2*E3/S*W
2134:          DZD      = E3*U + S*W
2135:      else if ( E3.ge.0. ) then
2136:          DXD      = -W
2137:          DYD      = V
2138:          DZD      = U
2139:      else
2140:          DXD      = W
2141:          DYD      = V
2142:          DZD      = -U
2143:      end if
2144:          U      = DXD
2145:          V      = DYD

```

```

2146:          W      = DZD
2147:      if ( I.le.NDEB ) write(6,*) ' UU*U+VV*V+WW*W=', UU*U + VV*V
2148:          + WW*W
2149:      if ( I.le.NDEB ) write(6,*) ' U=', U, ' V=', V, ' W=', W
2150: C
2151: C      CALCULATE LENGTH TO THE BOUNDARY.
2152:          DDDD      = 1.0E10
2153: C      REACH THE PLANE NORMAL TO THE X AXIS
2154:      if ( U.ne.0. ) then
2155:          XB      = AAR/2. - RAR + XXX(IP)
2156:          TT      = (XB-XX)/U
2157:          if ( TT.lt.0. ) then
2158:              XB      = -AAR/2. + RAR + XXX(IP)
2159:              TT      = (XB-XX)/U
2160:              if ( TT.ge.0. ) then
2161:                  if ( TT.lt.DDDD ) DDDD = TT
2162:              end if
2163:          else
2164:              if ( TT.lt.DDDD ) DDDD = TT
2165:          end if
2166:      end if
2167: C      REACH THE PLANE NORMAL TO THE X AXIS
2168:      if ( V.ne.0. ) then
2169:          YB      = AAR/2. - RAR + YYY(IP)
2170:          TT      = (YB-YY)/V
2171:          if ( TT.lt.0. ) then
2172:              YB      = -AAR/2. + RAR + YYY(IP)
2173:              TT      = (YB-YY)/V
2174:              if ( TT.ge.0. ) then
2175:                  if ( TT.lt.DDDD ) DDDD = TT
2176:              end if
2177:          else
2178:              if ( TT.lt.DDDD ) DDDD = TT
2179:          end if
2180:      end if
2181: C      REACH THE PLANE NORMAL TO THE Z AXIS
2182:      if ( W.ne.0. ) then
2183:          ZB      = AAR/2. - RAR + ZZZ(IP)
2184:          TT      = (ZB-ZZ)/W
2185:          if ( TT.lt.0. ) then
2186:              ZB      = -AAR/2. + RAR + ZZZ(IP)
2187:              TT      = (ZB-ZZ)/W
2188:              if ( TT.ge.0. ) then
2189:                  if ( TT.lt.DDDD ) DDDD = TT
2190:              end if
2191:          else
2192:              if ( TT.lt.DDDD ) DDDD = TT
2193:          end if
2194:      end if
2195: C
2196: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
2197:      do 140 J = 1, NRYU
2198:          if ( I.le.NDEB ) write(6,*) ' I=', I, ' J=', J, ' IP=', IP,
2199:          &      ' IDIS(J,IP)=', IDIS(J,IP), ' ISIS(J,IP)=', ISIS(J,IP)
2200:          X2      = XXX(IDIS(J,IP)) + XTR(ISIS(J,IP))
2201:          Y2      = YYY(IDIS(J,IP)) + YTR(ISIS(J,IP))
2202:          Z2      = ZZZ(IDIS(J,IP)) + ZTR(ISIS(J,IP))
2203: C      X2=XXX(IDIS(J,IP))
2204: C      Y2=YYY(IDIS(J,IP))
2205: C      Z2=ZZZ(IDIS(J,IP))
2206: C
2207: C      CALCULATE THE LENGTH TO THE PARTICLE CENTER: CCC
2208: C      CCC=SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*(ZZ-Z2))
2209: C
2210: C      IF(CCC.GT.DDDD+RAR) GOTO 370

```


src/tools/mcrdf.f

```

2211: C
2212:         T      = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
2213:         if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2, ' Z2=',
2214: &         Z2
2215:         if ( I.le.NDEB ) write(6,*) ' T=', T
2216:         if ( T.gt.0. ) then
2217:             X      = T*U + XX
2218:             Y      = T*V + YY
2219:             Z      = T*W + ZZ
2220:             D      = SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
2221:             if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=', Z
2222:             if ( I.le.NDEB ) write(6,*) ' D=', D
2223: C         IF(D.lt.RAR) GOTO 60
2224:             if ( D.lt.RAR ) then
2225:                 P3      =
2226: &             SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
2227: &             -Z)) - SQRT(RAR*RAR-D*D)
2228:             if ( P3.le.DDDD ) go to 210
2229:             end if
2230:         end if
2231: 140         continue
2232:         if ( I.le.NDEB ) write(6,*) ' WARNING. NO COLLISION OCCUR.'
2233: C
2234: C         NO COLLISION OCCUR IN THE CURRENT BOX.
2235: C         EXTENDED NND CALCULATION OPTION.
2236: C
2237: C         CHECK THE LENGTH OF THE FLIGHT PATH
2238: 150         DDD      = DDD + DDDD
2239:         if ( DDD.lt.DDDT ) then
2240:             ILV      = ILV + 1
2241: C
2242: C         DETERMINE THE INCIDENT PARTICLE CORDINATION.
2243: 160         XX      = (RANG()-0.5)*(AAR-2.*RAR)
2244:             YY      = (RANG()-0.5)*(AAR-2.*RAR)
2245:             ZZ      = (RANG()-0.5)*(AAR-2.*RAR)
2246: C 520 X1=(RANG()-0.5)*AAR
2247: C         Y1=(RANG()-0.5)*AAR
2248: C         SG=SIGN(1.,RANG()-0.5)
2249: C         XX=SG*AAR*0.5
2250: C         YY=XX
2251: C         ZZ=XX
2252: C         IK=INT(1.+RANG()*3.)
2253: C         UU=0.
2254: C         VV=0.
2255: C         WW=0.
2256: C         IF(IK.EQ.1) THEN
2257: C             XX=X1
2258: C             YY=Y1
2259: C             WW=-SG
2260: C         ELSE IF(IK.EQ.2) THEN
2261: C             UU=-SG
2262: C             YY=X1
2263: C             ZZ=Y1
2264: C         ELSE
2265: C             ZZ=X1
2266: C             XX=Y1
2267: C             VV=-SG
2268: C         END IF
2269: C
2270: C         RE-ORDER BY THE LENGTH
2271:         do 170 J = 1, NNR
2272:             DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
2273:             if ( DIS(J).le.RAR ) go to 160
2274:             ISS(J) = III
2275: 170         continue

```

```

2276:         call SORT2( DIS(1), NNR, IDD(1), NRYU )
2277:         do 180 K = 1, NRYU
2278:             ISSS(K) = ISS(IDD(K))
2279:             IDSS(K) = IDD(K)
2280: 180         continue
2281: C
2282: C         DETERMINE INSERTING DIRECTION OF THE PARTICLE
2283: 190         UP      = RANG()*2. - 1.
2284:             VP      = RANG()*2. - 1.
2285:             WP      = RANG()*2. - 1.
2286:             R      = SQRT(UP*UP+VP*VP+WP*WP)
2287:             if ( R.gt.1. ) go to 190
2288: C         RR=UU*UP+VV*VP+WW*WP
2289: C         IF(RR.LE.0.) GOTO 500
2290:             U      = UP/R
2291:             V      = VP/R
2292:             W      = WP/R
2293: C
2294: C         CALCULATE LENGTH TO THE BOUNDARY.
2295:             DDDD      = 1.0E10
2296: C         REACH THE PLANE NORMAL TO THE X AXIS
2297:             if ( U.ne.0. ) then
2298:                 XB      = AAR/2. - RAR
2299:                 TT      = XB/U
2300:                 if ( TT.lt.0. ) then
2301:                     XB      = -AAR/2. + RAR
2302:                     TT      = XB/U
2303:                 if ( TT.ge.0. ) then
2304:                     if ( TT.lt.DDDD ) DDDD = TT
2305:                 end if
2306:             else
2307:                 if ( TT.lt.DDDD ) DDDD = TT
2308:             end if
2309:         end if
2310: C         REACH THE PLANE NORMAL TO THE X AXIS
2311:             if ( V.ne.0. ) then
2312:                 YB      = AAR/2. - RAR
2313:                 TT      = YB/V
2314:                 if ( TT.lt.0. ) then
2315:                     YB      = -AAR/2. + RAR
2316:                     TT      = YB/V
2317:                 if ( TT.ge.0. ) then
2318:                     if ( TT.lt.DDDD ) DDDD = TT
2319:                 end if
2320:             else
2321:                 if ( TT.lt.DDDD ) DDDD = TT
2322:             end if
2323:         end if
2324: C         REACH THE PLANE NORMAL TO THE Z AXIS
2325:             if ( W.ne.0. ) then
2326:                 ZB      = AAR/2. - RAR
2327:                 TT      = ZB/W
2328:                 if ( TT.lt.0. ) then
2329:                     ZB      = -AAR/2. + RAR
2330:                     TT      = ZB/W
2331:                 if ( TT.ge.0. ) then
2332:                     if ( TT.lt.DDDD ) DDDD = TT
2333:                 end if
2334:             else
2335:                 if ( TT.lt.DDDD ) DDDD = TT
2336:             end if
2337:         end if
2338: C
2339: C         DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
2340: C

```

src/tools/mcrdf.f

```

2341: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
2342:       do 200 J = 1, NRYU
2343:         X2 = XXX(IDSS(J)) + XTR(ISSS(J))
2344:         Y2 = YYY(IDSS(J)) + YTR(ISSS(J))
2345:         Z2 = ZZZ(IDSS(J)) + ZTR(ISSS(J))
2346: C      X2=XXX(IDSS(J))
2347: C      Y2=YYY(IDSS(J))
2348: C      Z2=ZZZ(IDSS(J))
2349: C
2350: C      CALCULATE THE LENGTH TO THE PARTICLE CENTER: CCC
2351: C      CCC=SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*(ZZ-Z2))
2352: C
2353: C      IF(CCC.GT.DDDD+RAR) GOTO 270
2354: C
2355:       T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
2356:       if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
2357: &      ' Z2=', Z2
2358:       if ( I.le.NDEB ) write(6,*) ' T=', T
2359:       if ( T.gt.0. ) then
2360:         X = T*U + XX
2361:         Y = T*V + YY
2362:         Z = T*W + ZZ
2363:         D =
2364: &      SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z
2365: &      -ZZ))
2366:       if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z='
2367: &      Z
2368:       if ( I.le.NDEB ) write(6,*) ' D=', D
2369: C      IF(D.LT.RAR) GOTO 60
2370:       if ( D.lt.RAR ) then
2371:         P3 =
2372: &      SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*
2373: &      (ZZ-Z)) - SQRT(RAR*RAR-D*D)
2374:       if ( P3.le.DDDD ) go to 210
2375:       end if
2376:     end if
2377: 200 continue
2378: C
2379: C      RETURN TO THE SURFACE OF THE BOX
2380: C      TRANSPORT CONTINUES
2381:       go to 150
2382:     else
2383:       NC = NC + 1
2384:       go to 260
2385:     end if
2386: C
2387: C
2388: C      THE TRACKED PARTICLE IS DETERMINED
2389: 210 NCO(ILV) = NCO(ILV) + 1
2390: C
2391: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
2392: C
2393:       P(3) = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
2394: &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
2395:       if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
2396: C
2397: C      CALCULATE COSINE INSERTING THE PARTICLE
2398:       XP = XX + U*P3
2399:       YP = YY + V*P3
2400:       ZP = ZZ + W*P3
2401:       U1 = X2 - XP
2402:       V1 = Y2 - YP
2403:       W1 = Z2 - ZP
2404:       R1 = SQRT(U1*U1+V1*V1+W1*W1)
2405:       U1 = U1/R1
2406:       V1 = V1/R1
2407:       W1 = W1/R1
2408:       CSN = U*U1 + V*V1 + W*W1
2409:       if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
2410: C
2411:       J = 3
2412:       R = P(J)
2413:       if ( R.le.RAR*2.*REAL(NRMX) ) then
2414:         do 220 K = 1, NRMX*NGMX
2415:           if ( R.le.DEL*REAL(K) ) go to 230
2416: 220 continue
2417:           RCP(J,K) = RCP(J,K) + 1.
2418:           do 240 L = 1, 20
2419:             if ( CSN.le..05*REAL(L) ) go to 250
2420: 240 continue
2421:           CCN(K,L) = CCN(K,L) + 1.
2422:         end if
2423: C
2424: C      END OF HISTORIES
2425: 260 continue
2426: C
2427: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
2428: C      UNIT 12 : FOR MCNP.CFP
2429:       write(12,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
2430: C
2431:       write(6,'(1H1)')
2432:       write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 1 (EXTENDED NND',
2433: &      ' CALCULATION WAS DONE) OUTPUT FILE : FT12'
2434:       write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
2435:       write(6,*) ' REJECTION NO. : ', NC
2436:       write(6,*) ' NORMAL LEVEL. ', NCO(1)
2437:       write(6,'(1H )')
2438:       write(6,*) ' EXTEND LEVEL. LEVEL NO.'
2439:       I = 1
2440: 270 I = I + 1
2441:       if ( I.ne.100.and.NCO(I).ne.0 ) then
2442:         write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
2443:         go to 270
2444:       end if
2445:       write(6,'(1H0)')
2446:       write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
2447:       do 280 I = 1, NRMX*NGMX
2448:         RS = REAL(I-1) /REAL(NGMX)
2449:         RE = REAL(I) /REAL(NGMX)
2450:         if ( I.eq.1 ) then
2451:           RCP3 = RCP(3,I)
2452:         else
2453:           RCP3 = RCP(3,I) + RCP3
2454:         end if
2455:         FAC = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
2456:         RCP3J = RCP(3,I)*FAC
2457:         RCP3I = RCP3/REAL(NMAX)
2458:         write(3,'(1P2E12.5)') (RS+RE) /2., RCP3I
2459:         write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
2460: 7000 format(' ',6,F7.3, ' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
2461: C
2462:       write(12,'(1PE12.5)') RCP3I
2463: 280 continue
2464:       write(6,'(1H0)')
2465:       write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
2466: C
2467: C      OUTPUT ANGULAR DISTRIBUTION
2468:       write(6,'(1H1)')
2469:       write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
2470:       write(6,*) ' (EXTENDED NND CALCULATION)'

```

src/tools/mcrdf.f

```

2471:      write(6,*)
2472:      write(6,*) '          COSINE BINS'
2473:      write(6,*) ' GROUP  COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
2474:      do 290 I1 = 1, 20
2475:        write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
2476:      290 continue
2477:      7020 format(' ',I6,6X,2F10.7)
2478:      write(6,'(1H0)')
2479:      write(6,*) ' NO.          DISTANCE      ANG. DIST.(DP/DT)'
2480:      do 300 I = 1, NRMX*NGMX
2481:        RS      = REAL(I-1) /REAL(NGMX)
2482:        RE      = REAL(I) /REAL(NGMX)
2483:        if ( RCP(3,I).le.0 ) then
2484:          write(6,*) I, ' : RCP IS 0'
2485:          go to 300
2486:        end if
2487:        write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
2488:        write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
2489:      300 continue
2490:      7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
2491:      7060 format(' ',23X,1P10E10.3)
2492: C
2493:      return
2494:      end
2495: C
2496: C=====
2497: C
2498:      subroutine NND2
2499: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
2500: C      START PARTICLE FROM THE MATRIX
2501: C      THIS DISTRIBUTION IS USED FOR FIRST INSERTING THE CFP.
2502: C
2503: C      OUTPUT FILE : FT13; NND OUTPUT FOR PLOT
2504: C      FT14; NND OUTPUT FOR MCNP-CFP
2505: C
2506:      parameter( NRY  = 1000, MPX = 1000 )
2507:      parameter( PI   = 3.141592653 )
2508:      common /INPDT/  PFC,      AAR,      IBOX
2509:      common /TRNCM/  XTR(27),   YTR(27),   ZTR(27)
2510:      common /MAINCM/ XXX(MPX),   YYY(MPX),   ZZZ(MPX),
2511: &      RDF(MPX),      RCP(3,MPX),   RDP2(3,MPX),   SIGT,
2512: &      XNW(MPX),      YNW(MPX),     ZNW(MPX),      RAR,
2513: &      NNR
2514:      common /CRC/    DIS(MPX),      P(3),      IDIS(NRY,MPX),
2515: &      ISIS(NRY,MPX),   IDD(NRY),      ISS(MPX)
2516: C
2517:      dimension CCN(MPX,100)
2518: C      DIMENSION FOR EXTENDED NND CALCULATION
2519:      dimension ISSS(NRY), IDSS(NRY), NCO(100)
2520: C      DIMENSION IDSS(NRY),NCO(100)
2521: C
2522:      NDEB      = 0
2523:      NC        = 0
2524:      do 100 I = 1, 100
2525:        NCO(I) = 0
2526:      100 continue
2527:      do 110 I = 1, MPX
2528:        RCP(3,I) = 0.
2529:      110 continue
2530: C      MAXIMUM LENGTH OF NND
2531:      DDDT      = 30.0
2532:      NRMX      = INT(DDDT/RAR/2.+ .001)
2533: C      NRMX=INT(AAR/2./RAR/2.)
2534:      NGMX      = 20
2535:      DEL       = RAR*2./REAL(NGMX)

```

```

2536:      NRYU      = NRY
2537:      if ( NRY.gt.NNR ) NRYU = NNR
2538: C
2539: C      THE MAXIMUM NUMBER OF SAMPLING
2540:      NMAX      = 100000
2541: C
2542:      do 230 I = 1, NMAX
2543: C
2544:        DDD      = 0.
2545:        DDDD     = 0.
2546:        ILV      = 0
2547: C
2548: C      CHECK THE LENGTH OF THE FLIGHT PATH
2549:      120      DDD = DDD + DDDD
2550:        if ( DDD.lt.DDDD ) then
2551:          ILV = ILV + 1
2552: C
2553: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.
2554:      130      XX = (RANG()- .5)*(AAR-2.*RAR)
2555:              YY = (RANG()- .5)*(AAR-2.*RAR)
2556:              ZZ = (RANG()- .5)*(AAR-2.*RAR)
2557: C
2558: C      RE-ORDER BY THE LENGTH
2559:        do 140 J = 1, NNR
2560:          DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
2561:          if ( DIS(J).le.RAR ) go to 130
2562:          ISS(J) = III
2563:      140      continue
2564:        call SORT2( DIS(1), NNR, IDD(1), NRYU )
2565:        do 150 K = 1, NRYU
2566:          ISSS(K) = ISS(IDD(K))
2567:          IDSS(K) = IDD(K)
2568:      150      continue
2569: C
2570: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
2571:      160      UP = RANG()*2. - 1.
2572:              VP = RANG()*2. - 1.
2573:              WP = RANG()*2. - 1.
2574:              R  = SQRT(UP*UP+VP*VP+WP*WP)
2575:              if ( R.gt.1. ) go to 160
2576:              U  = UP/R
2577:              V  = VP/R
2578:              W  = WP/R
2579: C
2580: C      CALCULATE LENGTH TO THE BOUNDARY.
2581:      DDDD      = 1.0E10
2582: C      REACH THE PLANE NORMAL TO THE X AXIS
2583:        if ( U.ne.0. ) then
2584:          XB      = AAR/2. - RAR
2585:          TT      = XB/U
2586:          if ( TT.lt.0. ) then
2587:            XB     = -AAR/2. + RAR
2588:            TT     = XB/U
2589:          if ( TT.ge.0. ) then
2590:            if ( TT.lt.DDDD ) DDDD = TT
2591:          end if
2592:        else
2593:          if ( TT.lt.DDDD ) DDDD = TT
2594:        end if
2595:      end if
2596: C      REACH THE PLANE NORMAL TO THE X AXIS
2597:        if ( V.ne.0. ) then
2598:          YB      = AAR/2. - RAR
2599:          TT      = YB/V
2600:          if ( TT.lt.0. ) then

```

src/tools/mcrdf.f

```

2601:          YB      = -AAR/2. + RAR
2602:          TT      = YB/V
2603:          if ( TT.ge.0. ) then
2604:            if ( TT.lt.DDDD ) DDDD = TT
2605:          end if
2606:        else
2607:          if ( TT.lt.DDDD ) DDDD = TT
2608:        end if
2609:      end if
2610: C      REACH THE PLANE NORMAL TO THE Z AXIS
2611:      if ( W.ne.0. ) then
2612:        ZB      = AAR/2. - RAR
2613:        TT      = ZB/W
2614:        if ( TT.lt.0. ) then
2615:          ZB      = -AAR/2. + RAR
2616:          TT      = ZB/W
2617:          if ( TT.ge.0. ) then
2618:            if ( TT.lt.DDDD ) DDDD = TT
2619:          end if
2620:        else
2621:          if ( TT.lt.DDDD ) DDDD = TT
2622:        end if
2623:      end if
2624: C
2625: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
2626: C
2627: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR
2628:      do 170 J = 1, NRYU
2629:        X2      = XXX(IDSS(J)) + XTR(ISSS(J))
2630:        Y2      = YYY(IDSS(J)) + YTR(ISSS(J))
2631:        Z2      = ZZZ(IDSS(J)) + ZTR(ISSS(J))
2632:        T       = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
2633:        if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
2634: &      ' Z2=', Z2
2635:        if ( I.le.NDEB ) write(6,*) ' T=', T
2636:        if ( T.gt.0. ) then
2637:          X      = T*U + XX
2638:          Y      = T*V + YY
2639:          Z      = T*W + ZZ
2640:          D      =
2641: &      SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z
2642: &      -Z2))
2643:        if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=',
2644: &      Z
2645:        if ( I.le.NDEB ) write(6,*) ' D=', D
2646:        if ( D.lt.RAR ) then
2647:          P3     =
2648: &      SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*
2649: &      (ZZ-Z)) - SQRT(RAR*RAR-D*D)
2650:          if ( P3.le.DDDD ) go to 180
2651:        end if
2652:      end if
2653: 170      continue
2654: C
2655: C      RETURN TO THE SURFACE OF THE BOX
2656: C      TRANSPORT CONTINUES
2657:      go to 120
2658: C
2659: C
2660: C      THE TRACKED PARTICLE IS DETERMINED
2661: 180      NCO(ILV) = NCO(ILV) + 1
2662: C
2663: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
2664: C
2665:          P(3)      = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ-
2666: &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
2667:          if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
2668: C
2669: C      CALCULATE COSINE INSERTING THE PARTICLE
2670:          XP      = XX + U*P3
2671:          YP      = YY + V*P3
2672:          ZP      = ZZ + W*P3
2673:          U1      = X2 - XP
2674:          V1      = Y2 - YP
2675:          W1      = Z2 - ZP
2676:          R1      = SQRT(U1*U1+V1*V1+W1*W1)
2677:          U1      = U1/R1
2678:          V1      = V1/R1
2679:          W1      = W1/R1
2680:          CSN     = U*U1 + V*V1 + W*W1
2681:          if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
2682: C
2683:          J      = 3
2684:          R      = P(J)
2685:          if ( R.le.RAR*2.*REAL(NRMX) ) then
2686:            do 190 K = 1, NRMX*NGMX
2687:              if ( R.le.DEL*REAL(K) ) go to 200
2688:            190      continue
2689:            RCP(J,K) = RCP(J,K) + 1.
2690:            do 210 L = 1, 20
2691:              if ( CSN.le..05*REAL(L) ) go to 220
2692:            210      continue
2693:            220      CCN(K,L) = CCN(K,L) + 1.
2694:          end if
2695:        else
2696:          NC      = NC + 1
2697:        end if
2698: C
2699: C      END OF HISTORIES
2700: 230      continue
2701: C
2702: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
2703: C      UNIT 14 : FOR MCNP.CFP
2704:      write(14,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
2705: C
2706:      write(6,'(1H1)')
2707:      write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 2 (EXTENDED NND',
2708: &      ' CALCULATION WAS DONE) OUTPUT FILE : FT14'
2709:      write(6,*) ' PARTICLES START IN THE MATRIX.'
2710:      write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
2711:      write(6,*) ' REJECTION NO. : ', NC
2712:      write(6,*) ' NORMAL LEVEL. ', NCO(1)
2713:      write(6,'(1H )')
2714:      write(6,*) ' EXTEND LEVEL. LEVEL NO.'
2715:      I      = 1
2716: 240      I      = I + 1
2717:      if ( I.ne.100.and.NCO(I).ne.0 ) then
2718:        write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
2719:        go to 240
2720:      end if
2721:      write(6,'(1H0)')
2722:      write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
2723:      do 250 I = 1, NRMX*NGMX
2724:        RS      = REAL(I-1) /REAL(NGMX)
2725:        RE      = REAL(I) /REAL(NGMX)
2726:        if ( I.eq.1 ) then
2727:          RCP3   = RCP(3,I)
2728:        else
2729:          RCP3   = RCP(3,I) + RCP3
2730:        end if

```

src/tools/mcrdf.f

```

2731:      FAC      = RAR*RAR/RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
2732:      RCP3J     = RCP(3,I)*FAC
2733:      RCP3I     = RCP3/REAL(NMAX)
2734:      write(13, '(1P2E12.5)') (RS+RE) /2., RCP3I
2735:      write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
2736: 7000      format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
2737: C
2738:      write(14, '(1P2E12.5)') RCP3I
2739: 250 continue
2740:      write(6, '(1H0)')
2741:      write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
2742: C
2743: C      OUTPUT ANGULAR DISTRIBUTION
2744:      write(6, '(1H1)')
2745:      write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
2746:      write(6,*) ' (EXTENDED NND CALCULATION)'
2747:      write(6,*) '
2748:      write(6,*) ' COSINE BINS'
2749:      write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
2750:      do 260 I1 = 1, 20
2751:          write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
2752: 260 continue
2753: 7020 format(' ',I6,6X,2F10.7)
2754:      write(6, '(1H0)')
2755:      write(6,*) ' NO. DISTANCE ANG. DIST. (DP/DE)'
2756:      do 270 I = 1, NRMX*NGMX
2757:          RS = REAL(I-1) /REAL(NGMX)
2758:          RE = REAL(I) /REAL(NGMX)
2759:          if ( RCP(3,I).le.0 ) then
2760:              write(6,*) I, ' : RCP IS 0'
2761:              go to 270
2762:          end if
2763:          write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
2764:          write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
2765: 270 continue
2766: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
2767: 7060 format(' ',23X,1P10E10.3)
2768: C
2769:      return
2770:      end
2771: C
2772: C=====
2773: C
2774:      subroutine NND2R
2775: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
2776: C      START PARTICLE FROM THE MATRIX
2777: C      THE LOCATION WHERE PARTICLES START ARE SAMPLED BY CONSIDERING
2778: C      TOTAL CROSS SECTION.
2779: C      EXTENDED NND CALCULATION OPTION WAS ADDED.
2780: C      THIS PROGRAM IS ALMOST THE SAME AS NND
2781: C
2782: C      OUTPUT FILE : FT13; NND OUTPUT FOR PLOT
2783: C      FT14; NND OUTPUT FOR MCNP-CFP
2784: C
2785:      parameter( NRY = 1000, MPX = 1000 )
2786:      parameter( PI = 3.141592653 )
2787:      common /INPDT/ PFC, AAR, IBOX
2788:      common /TRNCM/ XTR(27), YTR(27), ZTR(27)
2789:      common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
2790: & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
2791: & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
2792: & NNR
2793:      common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
2794: & ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
2795: C

```

```

2796:      dimension CCN(MPX,100)
2797: C      DIMENSION FOR EXTENDED NND CALCULATION
2798:      dimension ISSS(NRY), IDSS(NRY), NCO(100)
2799: C      DIMENSION IDSS(NRY),NCO(100)
2800: C
2801:      NDEB = 0
2802:      NC = 0
2803:      do 100 I = 1, 100
2804:          NCO(I) = 0
2805: 100 continue
2806:      do 110 I = 1, MPX
2807:          RCP(3,I) = 0.
2808: 110 continue
2809: C      MAXIMUM LENGTH OF NND
2810:      DDDT = 30.0
2811:      NRMX = INT(DDDT/RAR/2.+0.001)
2812: C      NRMX=INT(AAR/2./RAR/2.)
2813:      NGMX = 20
2814:      DEL = RAR*2./REAL(NGMX)
2815:      NRYU = NRY
2816:      if ( NRY.gt.NNR ) NRYU = NNR
2817: C
2818: C      THE MAXIMUM NUMBER OF SAMPLING
2819:      NMAX = 200000
2820: C
2821:      do 260 I = 1, NMAX
2822: C
2823:          DDD = 0.
2824:          ILV = 1
2825: C
2826: C      SAMPLE REFERENCE PARTICLE
2827: 120      IP = INT(1.+RANG()*REAL(NNR))
2828: C
2829: C      CHECK IP
2830:      if ( IP.lt.1 .or. IP.gt.NNR ) then
2831:          write(6,*) ' IP IS INVALID. REJECT IP. IP=', IP
2832:          go to 120
2833:      end if
2834:      if ( I.le.NDEB ) write(6, '(1H1)')
2835:      if ( I.le.NDEB ) write(6,*) ' IP=', IP
2836: C
2837: C      SAMPLE POINT ON THE SPHERE
2838: 130      UU = RANG()*2. - 1.
2839:          VV = RANG()*2. - 1.
2840:          WW = RANG()*2. - 1.
2841:          R = SQRT(UU*UU+VV*VV+WW*WW)
2842:          if ( R.gt.1. ) go to 130
2843:          UU = UU/R
2844:          VV = VV/R
2845:          WW = WW/R
2846:          XX = UU*RAR + XXX(IP)
2847:          YY = VV*RAR + YYY(IP)
2848:          ZZ = WW*RAR + ZZZ(IP)
2849:          if ( I.le.NDEB ) write(6,*) ' UU=', UU, ' VV=', VV, ' WW=', WW
2850:          if ( I.le.NDEB ) write(6,*) ' XX=', XX, ' YY=', YY, ' ZZ=', ZZ
2851: C
2852: C      SAMPLE TRACKING DIRECTION
2853: C 40 TH=ASIN(RANG())
2854:          TH = ACOS(SQRT(RANG()))
2855:          PHI = 2.*PI*(RANG()-0.5)
2856:          U = COS(TH)
2857:          V = SIN(TH)*SIN(PHI)
2858:          W = SIN(TH)*COS(PHI)
2859:          R = SQRT(UU*UU+VV*VV+WW*WW)
2860:          E1 = UU/R

```

src/tools/mcrdf.f

```

2861:      E2      = VV/R
2862:      E3      = WW/R
2863:      S        = SQRT(E1*E1+E2*E2)
2864:      if ( S.ne.0.0 ) then
2865:          DXD   = E1*U - E2/S*V - E1*E3/S*W
2866:          DYD   = E2*U + E1/S*V - E2*E3/S*W
2867:          DZD   = E3*U + S*W
2868:      else if ( E3.ge.0. ) then
2869:          DXD   = -W
2870:          DYD   = V
2871:          DZD   = U
2872:      else
2873:          DXD   = W
2874:          DYD   = V
2875:          DZD   = -U
2876:      end if
2877:      U        = DXD
2878:      V        = DYD
2879:      W        = DZD
2880:      if ( I.le.NDEB ) write(6,*) ' UU*U+VV*V+WW*W=', UU*U + VV*V
2881:      &        + WW*W
2882:      if ( I.le.NDEB ) write(6,*) ' U=', U, ' V=', V, ' W=', W
2883: C
2884: C      SAMPLE THE POINT WHERE THE COLLISION OCCUR.
2885:      RR       = -ALOG(1.-RANG()) /SIGT
2886:      XS       = XX + RR*U
2887:      YS       = YY + RR*V
2888:      ZS       = ZZ + RR*W
2889:      DDS      = SQRT((XX-XS)*(XX-XS)+(YY-YS)*(YY-YS)+(ZZ-ZS)*(ZZ-ZS))
2890: C
2891: C      CALCULATE LENGTH TO THE BOUNDARY.
2892:      DDDD     = 1.0E10
2893: C      REACH THE PLANE NORMAL TO THE X AXIS
2894:      if ( U.ne.0. ) then
2895:          XB    = AAR/2. - RAR + XXX(IP)
2896:          TT    = (XB-XX) /U
2897:          if ( TT.lt.0. ) then
2898:              XB    = -AAR/2. + RAR + XXX(IP)
2899:              TT    = (XB-XX) /U
2900:              if ( TT.ge.0. ) then
2901:                  if ( TT.lt.DDDD ) DDDD = TT
2902:              end if
2903:          else
2904:              if ( TT.lt.DDDD ) DDDD = TT
2905:          end if
2906:      end if
2907: C      REACH THE PLANE NORMAL TO THE X AXIS
2908:      if ( V.ne.0. ) then
2909:          YB    = AAR/2. - RAR + YYY(IP)
2910:          TT    = (YB-YY) /V
2911:          if ( TT.lt.0. ) then
2912:              YB    = -AAR/2. + RAR + YYY(IP)
2913:              TT    = (YB-YY) /V
2914:              if ( TT.ge.0. ) then
2915:                  if ( TT.lt.DDDD ) DDDD = TT
2916:              end if
2917:          else
2918:              if ( TT.lt.DDDD ) DDDD = TT
2919:          end if
2920:      end if
2921: C      REACH THE PLANE NORMAL TO THE Z AXIS
2922:      if ( W.ne.0. ) then
2923:          ZB    = AAR/2. - RAR + ZZZ(IP)
2924:          TT    = (ZB-ZZ) /W
2925:          if ( TT.lt.0. ) then
2926:              ZB    = -AAR/2. + RAR + ZZZ(IP)
2927:              TT    = (ZB-ZZ) /W
2928:              if ( TT.ge.0. ) then
2929:                  if ( TT.lt.DDDD ) DDDD = TT
2930:              end if
2931:          else
2932:              if ( TT.lt.DDDD ) DDDD = TT
2933:          end if
2934:      end if
2935: C
2936:      if ( DDS.gt.DDDD ) go to 120
2937: C
2938: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
2939:      do 140 J = 1, NRYU
2940:          if ( I.le.NDEB ) write(6,*) ' I=', I, ' J=', J, ' IP=', IP,
2941:      &        ' IDIS(J,IP)=', IDIS(J,IP), ' ISIS(J,IP)=', ISIS(J,IP)
2942:          X2    = XXX(IDIS(J,IP)) + XTR(ISIS(J,IP))
2943:          Y2    = YYY(IDIS(J,IP)) + YTR(ISIS(J,IP))
2944:          Z2    = ZZZ(IDIS(J,IP)) + ZTR(ISIS(J,IP))
2945: C      X2=XXX(IDIS(J,IP))
2946: C      Y2=YYY(IDIS(J,IP))
2947: C      Z2=ZZZ(IDIS(J,IP))
2948: C
2949: C      CALCULATE THE LENGTH TO THE PARTICLE CENTER: CCC
2950: C      CCC=SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*(ZZ-Z2))
2951: C
2952: C      IF(CCC.GT.DDDD+RAR) GOTO 370
2953: C
2954:      T        = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
2955:      if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2, ' Z2=',
2956:      &        Z2
2957:      if ( I.le.NDEB ) write(6,*) ' T=', T
2958:      if ( T.gt.0. ) then
2959:          X      = T*U + XX
2960:          Y      = T*V + YY
2961:          Z      = T*W + ZZ
2962:          D      = SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
2963:          if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=', Z
2964:          if ( I.le.NDEB ) write(6,*) ' D=', D
2965:          if ( D.lt.RAR ) then
2966:              P3    =
2967:      &          SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
2968:      &          -Z)) - SQRT(RAR*RAR-D*D)
2969:          if ( P3.le.DDDD ) then
2970:              if ( DDS.lt.P3 ) then
2971:                  DDD    = -DDS
2972:                  go to 210
2973:              end if
2974:              go to 120
2975:          end if
2976:      end if
2977:      end if
2978: 140      continue
2979:      DDDD     = DDDD - DDS
2980:      if ( I.le.NDEB ) write(6,*) ' WARNING. NO COLLISION OCCUR.'
2981: C
2982: C      NO COLLISION OCCUR IN THE CURRENT BOX.
2983: C      EXTENDED NND CALCULATION OPTION.
2984: C
2985: C      CHECK THE LENGTH OF THE FLIGHT PATH
2986: 150      DDD     = DDD + DDDD
2987:          if ( DDD.lt.DDDT ) then
2988:              ILV    = ILV + 1
2989: C
2990: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.

```

src/tools/mcrdf.f

```

2991: 160      XX      = (RANG()-0.5)*(AAR-2.*RAR)
2992:         YY      = (RANG()-0.5)*(AAR-2.*RAR)
2993:         ZZ      = (RANG()-0.5)*(AAR-2.*RAR)
2994: C 520 X1=(RANG()-0.5)*AAR
2995: C      Y1=(RANG()-0.5)*AAR
2996: C      SG=SIGN(1.,RANG()-0.5)
2997: C      XX=SG*AAR*0.5
2998: C      YY=XX
2999: C      ZZ=XX
3000: C      IK=INT(1.+RANG()*3.)
3001: C      UU=0.
3002: C      VV=0.
3003: C      WW=0.
3004: C      IF(IK.EQ.1) THEN
3005: C      XX=X1
3006: C      YY=Y1
3007: C      WW=-SG
3008: C      ELSE IF(IK.EQ.2) THEN
3009: C      UU=-SG
3010: C      YY=X1
3011: C      ZZ=Y1
3012: C      ELSE
3013: C      ZZ=X1
3014: C      XX=Y1
3015: C      VV=-SG
3016: C      END IF
3017: C
3018: C      RE-ORDER BY THE LENGTH
3019: C      do 170 J = 1, NNR
3020: C      DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
3021: C      if ( DIS(J).le.RAR ) go to 160
3022: C      ISS(J) = III
3023: C      continue
3024: C      call SORT2( DIS(1), NNR, IDD(1), NRYU )
3025: C      do 180 K = 1, NRYU
3026: C      ISSS(K) = ISS(IDD(K))
3027: C      IDSS(K) = IDD(K)
3028: C      continue
3029: C      180
3030: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
3031: C      190      UP      = RANG()*2. - 1.
3032: C      VP      = RANG()*2. - 1.
3033: C      WP      = RANG()*2. - 1.
3034: C      R      = SQRT(UP*UP+VP*VP+WP*WP)
3035: C      if ( R.gt.1. ) go to 190
3036: C      RR=UU*UP+VV*VP+WW*WP
3037: C      IF(RR.LE.0.) GOTO 500
3038: C      U      = UP/R
3039: C      V      = VP/R
3040: C      W      = WP/R
3041: C
3042: C      CALCULATE LENGTH TO THE BOUNDARY.
3043: C      DDDD      = 1.0E10
3044: C      REACH THE PLANE NORMAL TO THE X AXIS
3045: C      if ( U.ne.0. ) then
3046: C      XB      = AAR/2. - RAR
3047: C      TT      = XB/U
3048: C      if ( TT.lt.0. ) then
3049: C      XB      = -AAR/2. + RAR
3050: C      TT      = XB/U
3051: C      if ( TT.ge.0. ) then
3052: C      if ( TT.lt.DDDD ) DDDD = TT
3053: C      end if
3054: C      else
3055: C      if ( TT.lt.DDDD ) DDDD = TT

```

```

3056:         end if
3057:         end if
3058: C      REACH THE PLANE NORMAL TO THE X AXIS
3059: C      if ( V.ne.0. ) then
3060: C      YB      = AAR/2. - RAR
3061: C      TT      = YB/V
3062: C      if ( TT.lt.0. ) then
3063: C      YB      = -AAR/2. + RAR
3064: C      TT      = YB/V
3065: C      if ( TT.ge.0. ) then
3066: C      if ( TT.lt.DDDD ) DDDD = TT
3067: C      end if
3068: C      else
3069: C      if ( TT.lt.DDDD ) DDDD = TT
3070: C      end if
3071: C      end if
3072: C      REACH THE PLANE NORMAL TO THE Z AXIS
3073: C      if ( W.ne.0. ) then
3074: C      ZB      = AAR/2. - RAR
3075: C      TT      = ZB/W
3076: C      if ( TT.lt.0. ) then
3077: C      ZB      = -AAR/2. + RAR
3078: C      TT      = ZB/W
3079: C      if ( TT.ge.0. ) then
3080: C      if ( TT.lt.DDDD ) DDDD = TT
3081: C      end if
3082: C      else
3083: C      if ( TT.lt.DDDD ) DDDD = TT
3084: C      end if
3085: C      end if
3086: C
3087: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
3088: C
3089: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
3090: C      do 200 J = 1, NRYU
3091: C      X2      = XXX(IDSS(J)) + XTR(ISSS(J))
3092: C      Y2      = YYY(IDSS(J)) + YTR(ISSS(J))
3093: C      Z2      = ZZZ(IDSS(J)) + ZTR(ISSS(J))
3094: C      X2=XXX(IDSS(J))
3095: C      Y2=YYY(IDSS(J))
3096: C      Z2=ZZZ(IDSS(J))
3097: C
3098: C      CALCULATE THE LENGTH TO THE PARTICLE CENTER: CCC
3099: C      CCC=SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
3100: C
3101: C      IF(CCC.GT.DDDD+RAR) GOTO 370
3102: C
3103: C      T      = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
3104: C      if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
3105: C      ' Z2=', Z2
3106: C      if ( I.le.NDEB ) write(6,*) ' T=', T
3107: C      if ( T.gt.0. ) then
3108: C      X      = T*U + XX
3109: C      Y      = T*V + YY
3110: C      Z      = T*W + ZZ
3111: C      D      =
3112: C      &      SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z
3113: C      &      -Z2))
3114: C      if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=',
3115: C      &      Z
3116: C      if ( I.le.NDEB ) write(6,*) ' D=', D
3117: C      IF(D.LT.RAR) GOTO 60
3118: C      if ( D.lt.RAR ) then
3119: C      P3      =
3120: C      &      SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*

```

src/tools/mcrdf.f

```

3121:      &              (ZZ-Z)) - SQRT(RAR*RAR-D*D)
3122:      if ( P3.le.DDDD ) go to 210
3123:      end if
3124:      end if
3125: 200      continue
3126: C
3127: C      RETURN TO THE SURFACE OF THE BOX
3128: C      TRANSPORT CONTINUES
3129:      go to 150
3130:      else
3131:      NC      = NC + 1
3132:      go to 260
3133:      end if
3134: C
3135: C
3136: C      THE TRACKED PARTICLE IS DETERMINED
3137: 210      NCO(ILV)      = NCO(ILV) + 1
3138: C
3139: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
3140: C
3141:      P(3)      = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
3142: &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
3143:      if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
3144: C
3145: C      CALCULATE COSINE INSERTING THE PARTICLE
3146:      XP      = XX + U*P3
3147:      YP      = YY + V*P3
3148:      ZP      = ZZ + W*P3
3149:      U1      = X2 - XP
3150:      V1      = Y2 - YP
3151:      W1      = Z2 - ZP
3152:      R1      = SQRT(U1*U1+V1*V1+W1*W1)
3153:      U1      = U1/R1
3154:      V1      = V1/R1
3155:      W1      = W1/R1
3156:      CSN      = U*U1 + V*V1 + W*W1
3157:      if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
3158: C
3159:      J      = 3
3160:      R      = P(J)
3161:      if ( R.le.RAR*2.*REAL(NRMX) ) then
3162:      do 220 K = 1, NRMX*NGMX
3163:      if ( R.le.DEL*REAL(K) ) go to 230
3164: 220      continue
3165: 230      RCP(J,K)      = RCP(J,K) + 1.
3166:      do 240 L = 1, 20
3167:      if ( CSN.le..05*REAL(L) ) go to 250
3168: 240      continue
3169: 250      CCN(K,L)      = CCN(K,L) + 1.
3170:      end if
3171: C
3172: C      END OF HISTORIES
3173: 260 continue
3174: C
3175: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
3176: C      UNIT 12 : FOR MCNP.CFP
3177: write(14,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
3178: C
3179: write(6,'(1H1)')
3180: write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 2R (EXTENDED NND',
3181: &      ' CALCULATION WAS DONE)      OUTPUT FILE : FT12'
3182: write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
3183: write(6,*) ' REJECTION NO. : ', NC
3184: write(6,*) '      NORMAL LEVEL.      ', NCO(1)
3185: write(6,'(1H )')

```

```

3186: write(6,*) '      EXTEND LEVEL.      LEVEL      NO.'
3187: I      = 1
3188: 270 I      = I + 1
3189: if ( I.ne.100.and.NCO(I).ne.0 ) then
3190:      write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
3191:      go to 270
3192: end if
3193: write(6,'(1H0)')
3194: write(6,*) ' NO.      DISTANCE      IRCP(3)', '      RCP(3)'
3195: do 280 I = 1, NRMX*NGMX
3196:      RS      = REAL(I-1) /REAL(NGMX)
3197:      RE      = REAL(I) /REAL(NGMX)
3198:      if ( I.eq.1 ) then
3199:      RCP3      = RCP(3,I)
3200:      else
3201:      RCP3      = RCP(3,I) + RCP3
3202:      end if
3203:      FAC      = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
3204:      RCP3J      = RCP(3,I)*FAC
3205:      RCP3I      = RCP3/REAL(NMAX)
3206:      write(13,'(1P2E12.5)') (RS+RE) /2., RCP3I
3207:      write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
3208: 7000      format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
3209: C
3210:      write(14,'(1PE12.5)') RCP3I
3211: 280 continue
3212: write(6,'(1H0)')
3213: write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
3214: C
3215: C      OUTPUT ANGULAR DISTRIBUTION
3216: write(6,'(1H1)')
3217: write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
3218: write(6,*) '      (EXTENDED NND CALCULATION)'
3219: write(6,*)
3220: write(6,*) '      COSINE BINS'
3221: write(6,*) ' GROUP      COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
3222: do 290 I1 = 1, 20
3223:      write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
3224: 290 continue
3225: 7020 format(' ',I6,6X,2F10.7)
3226: write(6,'(1H0)')
3227: write(6,*) ' NO.      DISTANCE      ANG. DIST.(DP/DT)'
3228: do 300 I = 1, NRMX*NGMX
3229:      RS      = REAL(I-1) /REAL(NGMX)
3230:      RE      = REAL(I) /REAL(NGMX)
3231:      if ( RCP(3,I).le.0 ) then
3232:      write(6,*) I, ' : RCP IS 0'
3233:      go to 300
3234:      end if
3235:      write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
3236:      write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
3237: 300 continue
3238: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
3239: 7060 format(' ',23X,1P10E10.3)
3240: C
3241:      return
3242:      end
3243: C
3244: C=====
3245: C
3246:      subroutine NND3
3247: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
3248: C      START PARTICLES FROM THE PLANE OUT OF THE BOX
3249: C
3250: C      EXTENDED NND CALCULATION OPTION WAS ADDED.

```


src/tools/mcrdf.f

```

3251: C      THIS PROGRAM IS ALMOST THE SAME AS CRCP EXCEPT EXTENDED
3252: C      NND CALCULATION OPTION.
3253: C
3254: C      OUTPUT FILE : FT15; NND OUTPUT FOR PLOT
3255: C      FT16; NND OUTPUT FOR MCNP-CFP
3256: C
3257: C      parameter( NRY = 1000, MPX = 1000 )
3258: C      PARAMETER(PI=3.141592653)
3259: C      common /INPDT/  PFC,      AAR,      IBOX
3260: C      common /TRNCM/  XTR(27),    YTR(27),    ZTR(27)
3261: C      common /MAINCM/ XXX(MPX),    YYY(MPX),    ZZZ(MPX),
3262: C      &      RDF(MPX),      RCF(3,MPX),      RDP2(3,MPX),      SIGT,
3263: C      &      XNW(MPX),      YNW(MPX),      ZNW(MPX),      RAR,
3264: C      &      NNR
3265: C      common /CRC/    DIS(MPX),      P(3),      IDIS(NRY,MPX),
3266: C      &      ISIS(NRY,MPX),      IDD(NRY),      ISS(MPX)
3267: C
3268: C      dimension CCN(MPX,100)
3269: C      DIMENSION FOR EXTENDED NND CALCULATION
3270: C      dimension ISSS(NRY), IDSS(NRY), NCO(100)
3271: C
3272: C      NDEB = 0
3273: C      NC = 0
3274: C      do 100 I = 1, 100
3275: C      NCO(I) = 0
3276: C 100 continue
3277: C      do 110 I = 1, MPX
3278: C      RCP(3,I) = 0.
3279: C 110 continue
3280: C      MAXIMUM LENGTH OF NND
3281: C      DDDT = 30.0
3282: C      NRMX = INT(DDDT/RAR/2.+0.001)
3283: C      NRMX=INT(AAR/2./RAR/2.)
3284: C      NGMX = 20
3285: C      DEL = RAR*2./REAL(NGMX)
3286: C      NRYU = NRY
3287: C      if ( NRY.gt.NNR ) NRYU = NNR
3288: C
3289: C      THE MAXIMUM NUMBER OF SAMPLING
3290: C      NMAX=30
3291: C      NMAX = 200000
3292: C
3293: C
3294: C      do 260 I = 1, NMAX
3295: C
3296: C      DDD = 0.
3297: C      ILV = 1
3298: C      IOPT=0
3299: C
3300: C      SAMPLE POINT ON THE PLANE
3301: C      YY = -AAR*.5 - RAR
3302: C      XX = (RANG()*2.-1.)*(AAR*.5-RAR)
3303: C      ZZ = (RANG()*2.-1.)*(AAR*.5-RAR)
3304: C
3305: C      SAMPLE TRACKING DIRECTION
3306: C      U = 0.
3307: C      V = 1.
3308: C      W = 0.
3309: C
3310: C      RE-ORDER BY THE LENGTH
3311: C      do 120 J = 1, NNR
3312: C      DIS(J) = DIST2(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J))
3313: C 120 continue
3314: C      DIS(J)=DIST2(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J))
3315: C      IF(DIS(J).LT.2.*RAR) THEN
3316: C      WRITE(6,*) ' RE-ORDER DEBUG, DIS(J).LT.2*RAR'
3317: C      WRITE(6,*) ' I=',I
3318: C      WRITE(6,*) ' J=',J
3319: C      WRITE(6,*) ' XX=',XX
3320: C      WRITE(6,*) ' YY=',YY
3321: C      WRITE(6,*) ' ZZ=',ZZ
3322: C      WRITE(6,*) ' XXX=',XXX(J)
3323: C      WRITE(6,*) ' YYY=',YYY(J)
3324: C      WRITE(6,*) ' ZZZ=',ZZZ(J)
3325: C      WRITE(6,*) ' DIS=',DIS(J)
3326: C      IOPT=1
3327: C      END IF
3328: C 391 CONTINUE
3329: C      call SORT2( DIS(1), NNR, IDD(1), NRYU )
3330: C      do 130 K = 1, NRYU
3331: C      IDSS(K) = IDD(K)
3332: C 130 continue
3333: C      IF(IOPT.EQ.1) THEN
3334: C      WRITE(6,*) ' *****SORT RESULTS'
3335: C      WRITE(6,*) ' IOPT=',IOPT
3336: C      WRITE(6,*) (IDSS(II),II=1,10)
3337: C      WRITE(6, '(1H ,10F6.3)') (DIS(IDSS(II)),II=1,10)
3338: C      END IF
3339: C
3340: C      CALCULATE LENGTH TO THE BOUNDARY.
3341: C      DDDD = AAR
3342: C
3343: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
3344: C      do 140 J = 1, NRYU
3345: C      X2=XXX(IDIS(J,IP))+XTR(ISIS(J,IP))
3346: C      Y2=YYY(IDIS(J,IP))+YTR(ISIS(J,IP))
3347: C      Z2=ZZZ(IDIS(J,IP))+ZTR(ISIS(J,IP))
3348: C      X2 = XXX(IDSS(J))
3349: C      Y2 = YYY(IDSS(J))
3350: C      Z2 = ZZZ(IDSS(J))
3351: C      T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
3352: C      if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2, ' Z2=',
3353: C      &      Z2
3354: C      if ( I.le.NDEB ) write(6,*) ' T=', T
3355: C      if ( T.gt.0. ) then
3356: C      X = T*U + XX
3357: C      Y = T*V + YY
3358: C      Z = T*W + ZZ
3359: C      D = SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
3360: C      IF(T.LT.2.*RAR.AND.IOPT.EQ.1) THEN
3361: C      WRITE(6,*) ' T.LT.2*RAR'
3362: C      WRITE(6,*) ' I=',I
3363: C      WRITE(6,*) ' J=',J
3364: C      WRITE(6,*) ' IDSS(J)=' ,IDSS(J)
3365: C      WRITE(6,*) ' XX=',XX
3366: C      WRITE(6,*) ' YY=',YY
3367: C      WRITE(6,*) ' ZZ=',ZZ
3368: C      WRITE(6,*) ' X2=',X2
3369: C      WRITE(6,*) ' Y2=',Y2
3370: C      WRITE(6,*) ' Z2=',Z2
3371: C      WRITE(6,*) ' T =',T
3372: C      WRITE(6,*) ' X =',X
3373: C      WRITE(6,*) ' Y =',Y
3374: C      WRITE(6,*) ' Z =',Z
3375: C      WRITE(6,*) ' D =',D
3376: C      END IF
3377: C      if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=', Z
3378: C      if ( I.le.NDEB ) write(6,*) ' D=', D
3379: C      IF(D.LT.RAR) GOTO 60
3380: C      if ( D.lt.RAR ) then

```

src/tools/mcrdf.f

```

3381:          P3      =
3382:      &          SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
3383:      &          -Z)) - SQRT(RAR*RAR-D*D)
3384: C      WRITE(6,*) ' D.LT.RAR D=',D
3385: C      WRITE(6,*) ' I=',I
3386: C      WRITE(6,*) ' J=',J
3387: C      WRITE(6,*) ' P3=',P3
3388: C      WRITE(6,*) ' DDDD=',DDDD
3389:          if ( P3.le.DDDD ) go to 210
3390:      end if
3391:      end if
3392: 140      continue
3393:      if ( I.le.NDEB ) write(6,*) ' WARNING. NO COLLISION OCCUR.'
3394: C      NO COLLISION OCCUR IN THE CURRENT BOX.
3395: C      EXTENDED NND CALCULATION OPTION.
3396: C
3397: C      CHECK THE LENGTH OF THE FLIGHT PATH
3398: C      DDD = DDD + DDD
3399: 150      if ( DDD.lt.DDDT ) then
3400:          ILV = ILV + 1
3401:      end if
3402: C
3403: C      DETERMINE THE INCIDENT PARTICLE COORDINATION
3404: 160      XX = (RANG()- .5)*(AAR-2.*RAR)
3405:          YY = (RANG()- .5)*(AAR-2.*RAR)
3406:          ZZ = (RANG()- .5)*(AAR-2.*RAR)
3407: C
3408: C      RE-ORDER BY THE LENGTH
3409:      do 170 J = 1, NNR
3410:          DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
3411:          if ( DIS(J).le.RAR ) go to 160
3412:          ISS(J) = III
3413: 170      continue
3414:          call SORT2( DIS(1), NNR, IDD(1), NRYU )
3415:          do 180 K = 1, NRYU
3416:              ISSS(K) = ISS(IDD(K))
3417:              IDSS(K) = IDD(K)
3418: 180      continue
3419: C
3420: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
3421: 190      UP = RANG()*2. - 1.
3422:          VP = RANG()*2. - 1.
3423:          WP = RANG()*2. - 1.
3424:          R = SQRT(UP*UP+VP*VP+WP*WP)
3425:          if ( R.gt.1. ) go to 190
3426: C      RR=UU*UP+VV*VP+WW*WP
3427: C      IF(RR.LE.0.) GOTO 500
3428:          U = UP/R
3429:          V = VP/R
3430:          W = WP/R
3431: C
3432: C      CALCULATE LENGTH TO THE BOUNDARY.
3433:          DDDD = 1.0E10
3434: C      REACH THE PLANE NORMAL TO THE X AXIS
3435:          if ( U.ne.0. ) then
3436:              XB = AAR/2. - RAR
3437:              TT = XB/U
3438:          if ( TT.lt.0. ) then
3439:              XB = -AAR/2. + RAR
3440:              TT = XB/U
3441:          if ( TT.ge.0. ) then
3442:              if ( TT.lt.DDDD ) DDDD = TT
3443:          end if
3444:      else
3445:          if ( TT.lt.DDDD ) DDDD = TT

```

```

3446:      end if
3447:      end if
3448: C      REACH THE PLANE NORMAL TO THE X AXIS
3449:          if ( V.ne.0. ) then
3450:              YB = AAR/2. - RAR
3451:              TT = YB/V
3452:          if ( TT.lt.0. ) then
3453:              YB = -AAR/2. + RAR
3454:              TT = YB/V
3455:          if ( TT.ge.0. ) then
3456:              if ( TT.lt.DDDD ) DDDD = TT
3457:          end if
3458:      else
3459:          if ( TT.lt.DDDD ) DDDD = TT
3460:      end if
3461:      end if
3462: C      REACH THE PLANE NORMAL TO THE Z AXIS
3463:          if ( W.ne.0. ) then
3464:              ZB = AAR/2. - RAR
3465:              TT = ZB/W
3466:          if ( TT.lt.0. ) then
3467:              ZB = -AAR/2. + RAR
3468:              TT = ZB/W
3469:          if ( TT.ge.0. ) then
3470:              if ( TT.lt.DDDD ) DDDD = TT
3471:          end if
3472:      else
3473:          if ( TT.lt.DDDD ) DDDD = TT
3474:      end if
3475:      end if
3476: C
3477: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
3478: C
3479: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
3480:      do 200 J = 1, NRYU
3481:          X2 = XXX(IDSS(J)) + XTR(ISSS(J))
3482:          Y2 = YYY(IDSS(J)) + YTR(ISSS(J))
3483:          Z2 = ZZZ(IDSS(J)) + ZTR(ISSS(J))
3484: C      X2=XXX(IDSS(J))
3485: C      Y2=YYY(IDSS(J))
3486: C      Z2=ZZZ(IDSS(J))
3487:          T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
3488:          if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
3489:          &          ' Z2=', Z2
3490:          if ( I.le.NDEB ) write(6,*) ' T=', T
3491:          if ( T.gt.0. ) then
3492:              X = T*U + XX
3493:              Y = T*V + YY
3494:              Z = T*W + ZZ
3495:              D =
3496:              &          SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z
3497:              &          -Z2))
3498:              if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y, ' Z=',
3499:              &          Z
3500:              if ( I.le.NDEB ) write(6,*) ' D=', D
3501: C      IF(D.LT.RAR) GOTO 60
3502:          if ( D.lt.RAR ) then
3503:              P3 =
3504:              &          SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*
3505:              &          (ZZ-Z)) - SQRT(RAR*RAR-D*D)
3506:              if ( P3.le.DDDD ) go to 210
3507:          end if
3508:      end if
3509: 200      continue
3510: C

```

src/tools/mcrdf.f

```

3511: C      RETURN TO THE SURFACE OF THE BOX
3512: C      TRANSPORT CONTINUES
3513:         go to 150
3514:     else
3515:         NC = NC + 1
3516:         go to 260
3517:     end if
3518: C
3519: C
3520: C      THE TRACKED PARTICLE IS DETERMINED
3521: 210 NCO(ILV) = NCO(ILV) + 1
3522: C
3523: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
3524: C
3525: C      P(3) = SQRT((XX-K)*(XX-K)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
3526: &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
3527: C      if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
3528: C
3529: C      CALCULATE COSINE INSERTING THE PARTICLE
3530: C      XP = XX + U*P3
3531: C      YP = YY + V*P3
3532: C      ZP = ZZ + W*P3
3533: C      U1 = X2 - XP
3534: C      V1 = Y2 - YP
3535: C      W1 = Z2 - ZP
3536: C      R1 = SQRT(U1*U1+V1*V1+W1*W1)
3537: C      U1 = U1/R1
3538: C      V1 = V1/R1
3539: C      W1 = W1/R1
3540: C      CSN = U*U1 + V*V1 + W*W1
3541: C      if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
3542: C
3543: C      J = 3
3544: C      R = P(J)
3545: C      if ( R.le.RAR*2.*REAL(NRMX) ) then
3546: C          do 220 K = 1, NRMX*NGMX
3547: C              if ( R.le.DEL*REAL(K) ) go to 230
3548: 220 continue
3549: 230 RCP(J,K) = RCP(J,K) + 1.
3550: C          do 240 L = 1, 20
3551: C              if ( CSN.le..05*REAL(L) ) go to 250
3552: 240 continue
3553: 250 CCN(K,L) = CCN(K,L) + 1.
3554: C          end if
3555: C
3556: C      END OF HISTORIES
3557: 260 continue
3558: C
3559: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
3560: C      UNIT 16 : FOR MCNP.CFP
3561: C      write(16,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
3562: C
3563: C      write(6,'(1H1)')
3564: C      write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 3 (EXTENDED NND',
3565: &      ' CALCULATION WAS DONE) OUTPUT FILE : FT16'
3566: C      write(6,*) ' PARTICLES START FROM THE PLANE OUT OF THE BOX.'
3567: C      write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
3568: C      write(6,*) ' REJECTION NO. : ', NC
3569: C      write(6,*) ' NORMAL LEVEL. ', NCO(1)
3570: C      write(6,'(1H )')
3571: C      write(6,*) ' EXTEND LEVEL. LEVEL NO.'
3572: C      I = 1
3573: 270 I = I + 1
3574: C      if ( I.ne.100.and.NCO(I).ne.0 ) then
3575: C          write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
3576: C          go to 270
3577: C      end if
3578: C      write(6,'(1H0)')
3579: C      write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
3580: C      do 280 I = 1, NRMX*NGMX
3581: C          RS = REAL(I-1) /REAL(NGMX)
3582: C          RE = REAL(I) /REAL(NGMX)
3583: C          if ( I.eq.1 ) then
3584: C              RCP3 = RCP(3,I)
3585: C          else
3586: C              RCP3 = RCP(3,I) + RCP3
3587: C          end if
3588: C          FAC = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
3589: C          RCP3J = RCP(3,I)*FAC
3590: C          RCP3I = RCP3/REAL(NMAX)
3591: C          write(15,'(1P2E12.5)') (RS+RE) /2., RCP3I
3592: C          write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
3593: C          format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
3594: C
3595: C          write(16,'(1P2E12.5)') RCP3I
3596: 280 continue
3597: C          write(6,'(1H0)')
3598: C          write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
3599: C
3600: C      OUTPUT ANGULAR DISTRIBUTION
3601: C      write(6,'(1H1)')
3602: C      write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
3603: C      write(6,*) ' (EXTENDED NND CALCULATION)'
3604: C      write(6,*)
3605: C      write(6,*) ' COSINE BINS'
3606: C      write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
3607: C      do 290 I1 = 1, 20
3608: C          write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
3609: 290 continue
3610: 7020 format(' ',I6,6X,2F10.7)
3611: C      write(6,'(1H0)')
3612: C      write(6,*) ' NO. DISTANCE ANG. DIST.(DP/DT)'
3613: C      do 300 I = 1, NRMX*NGMX
3614: C          RS = REAL(I-1) /REAL(NGMX)
3615: C          RE = REAL(I) /REAL(NGMX)
3616: C          if ( RCP(3,I).le.0 ) then
3617: C              write(6,*) I, ' : RCP IS 0'
3618: C              go to 300
3619: C          end if
3620: C          write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
3621: C          write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
3622: 300 continue
3623: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
3624: 7060 format(' ',23X,1P10E10.3)
3625: C
3626: C      return
3627: C      end
3628: C
3629: C=====
3630: C
3631: C      subroutine NND3R
3632: C          CALCULATE NEAREST NEIGHBOR DISTRIBUTION
3633: C          START PARTICLE FROM THE PLANE OUT OF THE BOX.
3634: C          INCIDENT PARTICLES ARE SAMPLED AT RANDOM IN THE BOX.
3635: C
3636: C          OUTPUT FILE : FT15; NND OUTPUT FOR PLOT
3637: C          FT16; NND OUTPUT FOR MCNP-CFP
3638: C
3639: C          parameter( NRY = 1000, MPX = 1000 )
3640: C          parameter( PI = 3.141592653 )

```

src/tools/mcrdf.f

```

3641:      common /INPDT/   PFC,      AAR,      IBOX
3642:      common /TRNCM/   XTR(27),    YTR(27),    ZTR(27)
3643:      common /MAINCM/   XXX(MPX),   YYY(MPX),   ZZZ(MPX),
3644:      &      RDF(MPX),    RCP(3,MPX),    RDP2(3,MPX),    SIGT,
3645:      &      XNW(MPX),    YNW(MPX),    ZNW(MPX),    RAR,
3646:      &      NNR
3647:      common /CRC/      DIS(MPX),    P(3),    IDIS(NRY,MPX),
3648:      &      ISIS(NRY,MPX),    IDD(NRY),    ISS(MPX)
3649: C
3650:      dimension CCN(MPX,100)
3651: C      DIMENSION FOR EXTENDED NND CALCULATION
3652:      dimension ISSS(NRY), IDSS(NRY), NCO(100)
3653: C      DIMENSION IDSS(NRY),NCO(100)
3654: C
3655:      NDEB = 0
3656:      NC = 0
3657:      do 100 I = 1, 100
3658:          NCO(I) = 0
3659: 100 continue
3660:      do 110 I = 1, MPX
3661:          RCP(3,I) = 0.
3662: 110 continue
3663: C      MAXIMUM LENGTH OF NND
3664:      DDDT = 30.0
3665:      NRMX = INT(DDDT/RAR/2. +.001)
3666: C      NRMX=INT(AAR/2./RAR/2. +.001)
3667:      NGMX = 20
3668:      DEL = RAR*2./REAL(NGMX)
3669:      NRYU = NRY
3670:      if ( NRY.gt.NNR ) NRYU = NNR
3671: C
3672: C      THE MAXIMUM NUMBER OF SAMPLING
3673:      NMAX = 100000
3674: C
3675:      do 230 I = 1, NMAX
3676: C
3677:          DDD = 0.
3678:          DDDD = 0.
3679:          ILV = 0
3680: C
3681: C      CHECK THE LENGTH OF THE FLIGHT PATH
3682: 120      DDD = DDD + DDDD
3683:          if ( DDD.lt.DDDT ) then
3684:              ILV = ILV + 1
3685: C
3686: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.
3687: 130      XX = (RANG()- .5)*(AAR-2.*RAR)
3688:          YY = (RANG()- .5)*(AAR-2.*RAR)
3689:          ZZ = (RANG()- .5)*(AAR-2.*RAR)
3690: C
3691: C      RE-ORDER BY THE LENGTH
3692:      do 140 J = 1, NNR
3693:          DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
3694: C
3695: C      MODIFIED ON 10-AUG-94
3696: C      IF(DIS(J).LE.RAR) GOTO 520
3697:          if ( ILV.ne.1.and.DIS(J).le.RAR ) go to 130
3698: C
3699:          ISS(J) = III
3700: 140      continue
3701:      call SORT2( DIS(1), NNR, IDD(1), NRYU )
3702:      do 150 K = 1, NRYU
3703:          ISSS(K) = ISS(IDD(K))
3704:          IDSS(K) = IDD(K)
3705: 150      continue

```

```

3706: C
3707: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
3708: 160      UP = RANG()*2. - 1.
3709:      VP = RANG()*2. - 1.
3710:      WP = RANG()*2. - 1.
3711:      R = SQRT(UP*UP+VP*VP+WP*WP)
3712:      if ( R.gt.1. ) go to 160
3713:      U = UP/R
3714:      V = VP/R
3715:      W = WP/R
3716: C
3717: C      CALCULATE LENGTH TO THE BOUNDARY.
3718:      DDDD = 1.0E10
3719: C      REACH THE PLANE NORMAL TO THE X AXIS
3720:      if ( U.ne.0. ) then
3721:          XB = AAR/2. - RAR
3722:          TT = XB/U
3723:          if ( TT.lt.0. ) then
3724:              XB = -AAR/2. + RAR
3725:              TT = XB/U
3726:          if ( TT.ge.0. ) then
3727:              if ( TT.lt.DDDD ) DDDD = TT
3728:          end if
3729:      else
3730:          if ( TT.lt.DDDD ) DDDD = TT
3731:      end if
3732:      end if
3733: C      REACH THE PLANE NORMAL TO THE X AXIS
3734:      if ( V.ne.0. ) then
3735:          YB = AAR/2. - RAR
3736:          TT = YB/V
3737:          if ( TT.lt.0. ) then
3738:              YB = -AAR/2. + RAR
3739:              TT = YB/V
3740:          if ( TT.ge.0. ) then
3741:              if ( TT.lt.DDDD ) DDDD = TT
3742:          end if
3743:      else
3744:          if ( TT.lt.DDDD ) DDDD = TT
3745:      end if
3746:      end if
3747: C      REACH THE PLANE NORMAL TO THE Z AXIS
3748:      if ( W.ne.0. ) then
3749:          ZB = AAR/2. - RAR
3750:          TT = ZB/W
3751:          if ( TT.lt.0. ) then
3752:              ZB = -AAR/2. + RAR
3753:              TT = ZB/W
3754:          if ( TT.ge.0. ) then
3755:              if ( TT.lt.DDDD ) DDDD = TT
3756:          end if
3757:      else
3758:          if ( TT.lt.DDDD ) DDDD = TT
3759:      end if
3760:      end if
3761: C
3762: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
3763: C
3764: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
3765:      do 170 J = 1, NRYU
3766:          X2 = XXX(IDSS(J)) + XTR(ISSS(J))
3767:          Y2 = YYY(IDSS(J)) + YTR(ISSS(J))
3768:          Z2 = ZZZ(IDSS(J)) + ZTR(ISSS(J))
3769:          T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
3770:          if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,

```

src/tools/mcrdf.f

```

3771:      &      ' Z2=', Z2
3772:      if ( I.le.NDEB ) write(6,*) ' T=', T
3773:      if ( T.gt.0. ) then
3774: C
3775: C      MODIFIED(ADDED) ON 10-AUG-94
3776:      if ( ILV.ne.1 .or. T.ge.RAR ) then
3777: C
3778:      X      = T*U + XX
3779:      Y      = T*V + YY
3780:      Z      = T*W + ZZ
3781:      D      =
3782:      &      SQR((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*(Z-Z2))
3783:      &
3784:      if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y,
3785:      &      ' Z=', Z
3786:      if ( I.le.NDEB ) write(6,*) ' D=', D
3787:      if ( D.lt.RAR ) then
3788:      P3      =
3789:      &      SQR((X-X)*(X-X)+(Y-Y)*(Y-Y)+(Z-Z)*(Z-Z)) - SQR(RAR*RAR-D*D)
3790:      &
3791:      if ( P3.le.DDDD ) go to 180
3792:      end if
3793:      end if
3794:      end if
3795: 170      continue
3796: C
3797: C      RETURN TO THE SURFACE OF THE BOX
3798: C      TRANSPORT CONTINUES
3799:      go to 120
3800: C
3801: C
3802: C      THE TRACKED PARTICLE IS DETERMINED
3803: 180      NCO(ILV) = NCO(ILV) + 1
3804: C
3805: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
3806: C
3807:      P(3) = SQR((X-X)*(X-X)+(Y-Y)*(Y-Y)+(Z-Z)*(Z-Z)) - SQR(RAR*RAR-D*D) + DDD
3808:      &
3809:      if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
3810: C
3811: C      CALCULATE COSINE INSERTING THE PARTICLE
3812:      XP      = XX + U*P3
3813:      YP      = YY + V*P3
3814:      ZP      = ZZ + W*P3
3815:      U1      = X2 - XP
3816:      V1      = Y2 - YP
3817:      W1      = Z2 - ZP
3818:      R1      = SQR(U1*U1+V1*V1+W1*W1)
3819:      U1      = U1/R1
3820:      V1      = V1/R1
3821:      W1      = W1/R1
3822:      CSN      = U*U1 + V*V1 + W*W1
3823:      if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
3824: C
3825:      J      = 3
3826:      R      = P(J)
3827:      if ( R.le.RAR*2.*REAL(NRMX) ) then
3828:      do 190 K = 1, NRMX*NGMX
3829:      if ( R.le.DEL*REAL(K) ) go to 200
3830: 190      continue
3831: 200      RCP(J,K) = RCP(J,K) + 1.
3832:      do 210 L = 1, 20
3833:      if ( CSN.le..05*REAL(L) ) go to 220
3834: 210      continue
3835: 220      CCN(K,L) = CCN(K,L) + 1.

```

```

3836:      end if
3837:      else
3838:      NC      = NC + 1
3839:      end if
3840: C
3841: C      END OF HISTORIES
3842: 230      continue
3843: C
3844: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
3845: C      UNIT 14 : FOR MCNP.CFP
3846:      write(16,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
3847: C
3848:      write(6,'(1H1)')
3849:      write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 3R (EXTENDED NND',
3850:      &      ' CALCULATION WAS DONE) OUTPUT FILE : FT14'
3851:      write(6,*) ' PARTICLES START IN THE MATRIX.'
3852:      write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
3853:      write(6,*) ' REJECTION NO. : ', NC
3854:      write(6,*) ' NORMAL LEVEL. ', NCO(1)
3855:      write(6,'(1H )')
3856:      write(6,*) ' EXTEND LEVEL. LEVEL NO.'
3857:      I      = 1
3858: 240      I      = I + 1
3859:      if ( I.ne.100.and.NCO(I).ne.0 ) then
3860:      write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
3861:      go to 240
3862:      end if
3863:      write(6,'(1H0)')
3864:      write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
3865:      do 250 I = 1, NRMX*NGMX
3866:      RS      = REAL(I-1) /REAL(NGMX)
3867:      RE      = REAL(I) /REAL(NGMX)
3868:      if ( I.eq.1 ) then
3869:      RCP3     = RCP(3,I)
3870:      else
3871:      RCP3     = RCP(3,I) + RCP3
3872:      end if
3873:      FAC      = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
3874:      RCP3J     = RCP(3,I)*FAC
3875:      RCP3I     = RCP3/REAL(NMAX)
3876:      write(15,'(1P2E12.5)') (RS+RE) /2., RCP3I
3877:      write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
3878: 7000      format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
3879: C
3880:      write(16,'(1PE12.5)') RCP3I
3881: 250      continue
3882:      write(6,'(1H0)')
3883:      write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
3884: C
3885: C      OUTPUT ANGULAR DISTRIBUTION
3886:      write(6,'(1H1)')
3887:      write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
3888:      write(6,*) ' (EXTENDED NND CALCULATION)'
3889:      write(6,*)
3890:      write(6,*) ' COSINE BINS'
3891:      write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
3892:      do 260 I1 = 1, 20
3893:      write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
3894: 260      continue
3895: 7020      format(' ',I6,6X,2F10.7)
3896:      write(6,'(1H0)')
3897:      write(6,*) ' NO. DISTANCE ANG. DIST.(DP/DT)'
3898:      do 270 I = 1, NRMX*NGMX
3899:      RS      = REAL(I-1) /REAL(NGMX)
3900:      RE      = REAL(I) /REAL(NGMX)

```

src/tools/mcrdf.f

```

3901:      if ( RCP(3,I).le.0 ) then
3902:        write(6,*) I, ' : RCP IS 0'
3903:        go to 270
3904:      end if
3905:      write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
3906:      write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
3907: 270 continue
3908: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
3909: 7060 format(' ',23X,1P10E10.3)
3910: C
3911:      return
3912:      end
3913: C
3914: C=====
3915: C
3916:      subroutine NND3R2
3917: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
3918: C      START PARTICLE FROM THE PLANE OUT OF THE BOX.
3919: C      INCIDENT PARTICLES ARE SAMPLED AT RANDOM IN THE BOX.
3920: C      THIS SUBROUTINE IS ALMOST THE SAME AS NND3R.
3921: C
3922: C      OUTPUT FILE : FT15; NND OUTPUT FOR PLOT
3923: C      FT16; NND OUTPUT FOR MCNP-CFP
3924: C
3925:      parameter( NRY = 1000, MPX = 1000 )
3926:      parameter( PI = 3.141592653 )
3927:      common /INPDT/ PFC, AAR, IBOX
3928:      common /TRNCM/ XTR(27), YTR(27), ZTR(27)
3929:      common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX)
3930:      & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
3931:      & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
3932:      & NNR
3933:      common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
3934:      & ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
3935: C
3936:      dimension CCN(MPX,100)
3937: C      DIMENSION FOR EXTENDED NND CALCULATION
3938:      dimension ISSS(NRY), IDSS(NRY), NCO(100)
3939: C      DIMENSION IDSS(NRY),NCO(100)
3940: C
3941:      NDEB = 0
3942:      NC = 0
3943:      do 100 I = 1, 100
3944:        NCO(I) = 0
3945: 100 continue
3946:      do 110 I = 1, MPX
3947:        RCP(3,I) = 0.
3948: 110 continue
3949: C      MAXIMUM LENGTH OF NND
3950:      DDDT = 30.0
3951:      NRMX = INT(DDDT/RAR/2.+0.001)
3952: C      NRMX=INT(AAR/2./RAR/2.)
3953:      NGMX = 20
3954:      DEL = RAR*2./REAL(NGMX)
3955:      NRYU = NRY
3956:      if ( NRY.gt.NNR ) NRYU = NNR
3957: C
3958: C      THE MAXIMUM NUMBER OF SAMPLING
3959:      NMAX = 100000
3960: C
3961:      do 230 I = 1, NMAX
3962: C
3963:        DDD = 0.
3964:        DDDD = 0.
3965:        ILV = 0
3966: C
3967: C      CHECK THE LENGTH OF THE FLIGHT PATH
3968: 120      DDD = DDD + DDDD
3969:        if ( DDD.lt.DDDT ) then
3970:          ILV = ILV + 1
3971: C
3972: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.
3973: 130      XX = (RANG()-0.5)*(AAR-2.*RAR)
3974:          YY = (RANG()-0.5)*(AAR-2.*RAR)
3975:          ZZ = (RANG()-0.5)*(AAR-2.*RAR)
3976: C
3977: C      RE-ORDER BY THE LENGTH
3978:          do 140 J = 1, NNR
3979:            DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
3980: C
3981: C      MODIFIED ON 14-SEP-94
3982: C      RETURN TO NND2'S ROUTINE
3983:          if ( DIS(J).le.RAR ) go to 130
3984: C      IF(ILV.NE.1.AND.DIS(J).LE.RAR) GOTO 520
3985: C
3986:          ISS(J) = III
3987: 140      continue
3988:          call SORT2( DIS(1), NNR, IDD(1), NRYU )
3989:          do 150 K = 1, NRYU
3990:            ISSS(K) = ISS(IDD(K))
3991:            IDSS(K) = IDD(K)
3992: 150      continue
3993: C
3994: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
3995: 160      UP = RANG()*2. - 1.
3996:          VP = RANG()*2. - 1.
3997:          WP = RANG()*2. - 1.
3998:          R = SQRT(UP*UP+VP*VP+WP*WP)
3999:          if ( R.gt.1. ) go to 160
4000:          U = UP/R
4001:          V = VP/R
4002:          W = WP/R
4003: C
4004: C      CALCULATE LENGTH TO THE BOUNDARY.
4005:          DDDD = 1.0E10
4006: C      REACH THE PLANE NORMAL TO THE X AXIS
4007:          if ( U.ne.0. ) then
4008:            XB = AAR/2. - RAR
4009:            TT = XB/U
4010:            if ( TT.lt.0. ) then
4011:              XB = -AAR/2. + RAR
4012:              TT = XB/U
4013:            if ( TT.ge.0. ) then
4014:              if ( TT.lt.DDDD ) DDDD = TT
4015:            end if
4016:          else
4017:            if ( TT.lt.DDDD ) DDDD = TT
4018:          end if
4019:        end if
4020: C      REACH THE PLANE NORMAL TO THE X AXIS
4021:          if ( V.ne.0. ) then
4022:            YB = AAR/2. - RAR
4023:            TT = YB/V
4024:            if ( TT.lt.0. ) then
4025:              YB = -AAR/2. + RAR
4026:              TT = YB/V
4027:            if ( TT.ge.0. ) then
4028:              if ( TT.lt.DDDD ) DDDD = TT
4029:            end if
4030:          else

```

src/tools/mcrdf.f

```

4031:          if ( TT.lt.DDDD ) DDDD = TT
4032:        end if
4033:      end if
4034: C      REACH THE PLANE NORMAL TO THE Z AXIS
4035:      if ( W.ne.0. ) then
4036:        ZB = AAR/2. - RAR
4037:        TT = ZB/W
4038:        if ( TT.lt.0. ) then
4039:          ZB = -AAR/2. + RAR
4040:          TT = ZB/W
4041:        if ( TT.ge.0. ) then
4042:          if ( TT.lt.DDDD ) DDDD = TT
4043:        end if
4044:      else
4045:        if ( TT.lt.DDDD ) DDDD = TT
4046:      end if
4047:    end if
4048: C
4049: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
4050: C
4051: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
4052:      do 170 J = 1, NRYU
4053:        X2 = XXX(IDSS(J)) + XTR(ISSS(J))
4054:        Y2 = YYY(IDSS(J)) + YTR(ISSS(J))
4055:        Z2 = ZZZ(IDSS(J)) + ZTR(ISSS(J))
4056:        T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
4057:        if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
4058: &      ' Z2=', Z2
4059:        if ( I.le.NDEB ) write(6,*) ' T=', T
4060:        if ( T.gt.0. ) then
4061: C
4062: C      MODIFIED(ADDED) ON 10-AUG-94
4063:        if ( ILV.ne.1 .or. T.ge.RAR ) then
4064: C
4065:          X = T*U + XX
4066:          Y = T*V + YY
4067:          Z = T*W + ZZ
4068:          D =
4069: &      SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*
4070: &      (Z-Z2))
4071:          if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y,
4072: &      ' Z=', Z
4073:          if ( I.le.NDEB ) write(6,*) ' D=', D
4074:          if ( D.lt.RAR ) then
4075:            P3 =
4076: &      SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ
4077: &      -Z)*(ZZ-Z)) - SQRT(RAR*RAR-D*D)
4078:          if ( P3.le.DDDD ) go to 180
4079:        end if
4080:      end if
4081:    end if
4082: 170 continue
4083: C
4084: C      RETURN TO THE SURFACE OF THE BOX
4085: C      TRANSPORT CONTINUES
4086:      go to 120
4087: C
4088: C
4089: C      THE TRACKED PARTICLE IS DETERMINED
4090: 180 NCO(ILV) = NCO(ILV) + 1
4091: C
4092: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
4093: C
4094:      P(3) = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ
4095: &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
4096:
4097: C
4098: C      CALCULATE COSINE INSERTING THE PARTICLE
4099: C      XP = XX + U*P3
4100: C      YP = YY + V*P3
4101: C      ZP = ZZ + W*P3
4102: C      U1 = X2 - XP
4103: C      V1 = Y2 - YP
4104: C      W1 = Z2 - ZP
4105: C      R1 = SQRT(U1*U1+V1*V1+W1*W1)
4106: C      U1 = U1/R1
4107: C      V1 = V1/R1
4108: C      W1 = W1/R1
4109: C      CSN = U*U1 + V*V1 + W*W1
4110: C      if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
4111: C
4112: C      J = 3
4113: C      R = P(J)
4114: C      if ( R.le.RAR*2.*REAL(NRMX) ) then
4115: C        do 190 K = 1, NRMX*NGMX
4116: C          if ( R.le.DEL*REAL(K) ) go to 200
4117: 190 continue
4118: C        RCP(J,K) = RCP(J,K) + 1.
4119: C        do 210 L = 1, 20
4120: C          if ( CSN.le..05*REAL(L) ) go to 220
4121: 210 continue
4122: C        CCN(K,L) = CCN(K,L) + 1.
4123: C      end if
4124: C      else
4125: C        NC = NC + 1
4126: C      end if
4127: C
4128: C      END OF HISTORIES
4129: 230 continue
4130: C
4131: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
4132: C      UNIT 14 : FOR MCNP.CFP
4133: C      write(16,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
4134: C
4135: C      write(6,'(1H1)')
4136: C      write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 3R (EXTENDED NND',
4137: &      ' CALCULATION WAS DONE) OUTPUT FILE : FT14'
4138: C      write(6,*) ' PARTICLES START IN THE MATRIX.'
4139: C      write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
4140: C      write(6,*) ' REJECTION NO. : ', NC
4141: C      write(6,*) ' NORMAL LEVEL. ', NCO(1)
4142: C      write(6,'(1H )')
4143: C      write(6,*) ' EXTEND LEVEL. LEVEL NO.'
4144: C      I = 1
4145: 240 I = I + 1
4146: C      if ( I.ne.100.and.NCO(I).ne.0 ) then
4147: C        write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
4148: C        go to 240
4149: C      end if
4150: C      write(6,'(1H0)')
4151: C      write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
4152: C      do 250 I = 1, NRMX*NGMX
4153: C        RS = REAL(I-1) /REAL(NGMX)
4154: C        RE = REAL(I) /REAL(NGMX)
4155: C        if ( I.eq.1 ) then
4156: C          RCP3 = RCP(3,I)
4157: C        else
4158: C          RCP3 = RCP(3,I) + RCP3
4159: C        end if
4160: C        FAC = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE-RE*RS*RS)

```

src/tools/mcrdf.f

```

4161:      RCP3J  = RCP(3,I)*FAC
4162:      RCP3I  = RCP3/REAL(NMAX)
4163:      write(15,'(1P2E12.5)') (RS+RE) /2., RCP3I
4164:      write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
4165:      format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
4166: C
4167:      write(16,'(1P2E12.5)') RCP3I
4168:      250 continue
4169:      write(6,'(1H0)')
4170:      write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
4171: C
4172: C      OUTPUT ANGULAR DISTRIBUTION
4173: C      write(6,'(1H1)')
4174: C      write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
4175: C      write(6,*) ' (EXTENDED NND CALCULATION)'
4176: C      write(6,*)
4177: C      write(6,*) ' COSINE BINS'
4178: C      write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
4179: C      do 260 I1 = 1, 20
4180: C      write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
4181: C      260 continue
4182: C      7020 format(' ',I6,6X,2F10.7)
4183: C      write(6,'(1H0)')
4184: C      write(6,*) ' NO. DISTANCE ANG. DIST.(DPADT)'
4185: C      do 270 I = 1, NRMX*NGMX
4186: C      RS = REAL(I-1)/REAL(NGMX)
4187: C      RE = REAL(I)/REAL(NGMX)
4188: C      if ( RCP(3,I).le.0 ) then
4189: C      write(6,*) I, ' : RCP IS 0'
4190: C      go to 270
4191: C      end if
4192: C      write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
4193: C      write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
4194: C      270 continue
4195: C      7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
4196: C      7060 format(' ',23X,1P10E10.3)
4197: C
4198:      return
4199:      end
4200: C
4201: C=====
4202: C
4203:      subroutine NND3R3
4204: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
4205: C      START PARTICLE FROM THE PLANE OUT OF THE BOX.
4206: C      INCIDENT PARTICLES ARE SAMPLED AT RANDOM IN THE BOX.
4207: C      THIS SUBROUTINE IS ALMOST THE SAME AS NND3R.
4208: C
4209: C      OUTPUT FILE : FT15; NND OUTPUT FOR PLOT
4210: C      FT16; NND OUTPUT FOR MCNP-CFP
4211: C
4212:      parameter( NRY = 1000, MPX = 1000 )
4213:      parameter( PI = 3.141592653 )
4214:      common /INPDT/ PFC, AAR, IBOX
4215:      common /TRNCM/ XTR(27), YTR(27), ZTR(27)
4216:      common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
4217:      & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
4218:      & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
4219:      & NNR
4220:      common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
4221:      & ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
4222: C
4223:      dimension CCN(MPX,100)
4224: C      DIMENSION FOR EXTENDED NND CALCULATION
4225:      dimension ISSS(NRY), IDSS(NRY), NCO(100)

```

```

4226: C      DIMENSION IDSS(NRY),NCO(100)
4227: C
4228:      NDEB = 0
4229:      NC = 0
4230:      do 100 I = 1, 100
4231:      NCO(I) = 0
4232:      100 continue
4233:      do 110 I = 1, MPX
4234:      RCP(3,I) = 0.
4235:      110 continue
4236: C      MAXIMUM LENGTH OF NND
4237:      DDDT = 30.0
4238:      NRMX = INT(DDDT/RAR/2.+0.001)
4239: C      NRMX=INT(AAR/2./RAR/2.)
4240:      NGMX = 20
4241:      DEL = RAR*2./REAL(NGMX)
4242:      NRYU = NRY
4243:      if ( NRY.gt.NNR ) NRYU = NNR
4244: C
4245: C      THE MAXIMUM NUMBER OF SAMPLING
4246:      NMAX = 100000
4247: C
4248:      do 230 I = 1, NMAX
4249: C
4250:      DDD = 0.
4251:      DDDD = 0.
4252:      ILV = 0
4253: C
4254: C      CHECK THE LENGTH OF THE FLIGHT PATH
4255: C      120 DDD = DDD + DDDD
4256: C      if ( DDD.lt.DDDT ) then
4257: C      ILV = ILV + 1
4258: C
4259: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.
4260: C      130 XX = (RANG()-0.5)*(AAR-2.*RAR)
4261: C      YY = (RANG()-0.5)*(AAR-2.*RAR)
4262: C      ZZ = (RANG()-0.5)*(AAR-2.*RAR)
4263: C
4264: C      RE-ORDER BY THE LENGTH
4265: C      do 140 J = 1, NNR
4266: C      DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
4267: C
4268: C      MODIFIED ON 14-SEP-94
4269: C      RETURN TO NND3R'S ROUTINE
4270: C      IF(DIS(J).LE.RAR) GOTO 520
4271: C      if ( ILV.ne.1.and.DIS(J).le.RAR ) go to 130
4272: C
4273: C      ISS(J) = III
4274: C      140 continue
4275: C      call SORT2( DIS(1), NNR, IDD(1), NRYU )
4276: C      do 150 K = 1, NRYU
4277: C      ISSS(K) = ISS(IDD(K))
4278: C      IDSS(K) = IDD(K)
4279: C      150 continue
4280: C
4281: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
4282: C      160 UP = RANG()*2. - 1.
4283: C      VP = RANG()*2. - 1.
4284: C      WP = RANG()*2. - 1.
4285: C      R = SQRT(UP*UP+VP*VP+WP*WP)
4286: C      if ( R.gt.1. ) go to 160
4287: C      U = UP/R
4288: C      V = VP/R
4289: C      W = WP/R
4290: C

```


src/tools/mcrdf.f

```

4291: C      CALCULATE LENGTH TO THE BOUNDARY.
4292:          DDDD = 1.0E10
4293: C      REACH THE PLANE NORMAL TO THE X AXIS
4294:          if ( U.ne.0. ) then
4295:              XB = AAR/2. - RAR
4296:              TT = XB/U
4297:              if ( TT.lt.0. ) then
4298:                  XB = -AAR/2. + RAR
4299:                  TT = XB/U
4300:              if ( TT.ge.0. ) then
4301:                  if ( TT.lt.DDDD ) DDDD = TT
4302:              end if
4303:          else
4304:              if ( TT.lt.DDDD ) DDDD = TT
4305:          end if
4306:      end if
4307: C      REACH THE PLANE NORMAL TO THE X AXIS
4308:          if ( V.ne.0. ) then
4309:              YB = AAR/2. - RAR
4310:              TT = YB/V
4311:              if ( TT.lt.0. ) then
4312:                  YB = -AAR/2. + RAR
4313:                  TT = YB/V
4314:              if ( TT.ge.0. ) then
4315:                  if ( TT.lt.DDDD ) DDDD = TT
4316:              end if
4317:          else
4318:              if ( TT.lt.DDDD ) DDDD = TT
4319:          end if
4320:      end if
4321: C      REACH THE PLANE NORMAL TO THE Z AXIS
4322:          if ( W.ne.0. ) then
4323:              ZB = AAR/2. - RAR
4324:              TT = ZB/W
4325:              if ( TT.lt.0. ) then
4326:                  ZB = -AAR/2. + RAR
4327:                  TT = ZB/W
4328:              if ( TT.ge.0. ) then
4329:                  if ( TT.lt.DDDD ) DDDD = TT
4330:              end if
4331:          else
4332:              if ( TT.lt.DDDD ) DDDD = TT
4333:          end if
4334:      end if
4335: C
4336: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
4337: C
4338: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
4339:          do 170 J = 1, NRYU
4340:              X2 = XXX(IDSS(J)) + XTR(ISSS(J))
4341:              Y2 = YYY(IDSS(J)) + YTR(ISSS(J))
4342:              Z2 = ZZZ(IDSS(J)) + ZTR(ISSS(J))
4343: C
4344: C      MODIFIED FOR NNR3R3 ON 16-SEP-94
4345:          T =
4346:          &      SQRT((XX-X2)*(XX-X2)+(YY-Y2)*(YY-Y2)+(ZZ-Z2)*
4347:          &      (ZZ-Z2))
4348:          if ( T.gt.RAR ) then
4349: C
4350:              T = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
4351:              if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
4352:              &      ' Z2=', Z2
4353:              if ( I.le.NDEB ) write(6,*) ' T=', T
4354:              if ( T.gt.0. ) then
4355:                  X = T*U + XX
4356:                  Y = T*V + YY
4357:                  Z = T*W + ZZ
4358:                  D =
4359:                  &      SQRT((X-X2)*(X-X2)+(Y-Y2)*(Y-Y2)+(Z-Z2)*
4360:                  &      (Z-Z2))
4361:                  if ( I.le.NDEB ) write(6,*) ' X=', X, ' Y=', Y,
4362:                  &      ' Z=', Z
4363:                  if ( I.le.NDEB ) write(6,*) ' D=', D
4364:                  if ( D.lt.RAR ) then
4365:                      P3 =
4366:                      &      SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ
4367:                      &      -Z)*(ZZ-Z)) - SQRT(RAR*RAR-D*D)
4368:                      if ( P3.le.DDDD ) go to 180
4369:                  end if
4370:              end if
4371:          end if
4372:          170 continue
4373: C
4374: C      RETURN TO THE SURFACE OF THE BOX
4375: C      TRANSPORT CONTINUES
4376:          go to 120
4377: C
4378: C
4379: C      THE TRACKED PARTICLE IS DETERMINED
4380:          180 NCO(ILV) = NCO(ILV) + 1
4381: C
4382: C      P(3) : DISTANCE BETWEEN SURFACE AND SURFACE
4383: C
4384:          P(3) = SQRT((XX-X)*(XX-X)+(YY-Y)*(YY-Y)+(ZZ-Z)*(ZZ
4385:          &      -Z)) - SQRT(RAR*RAR-D*D) + DDD
4386:          if ( I.le.NDEB ) write(6,*) ' P(3)=', P(3)
4387: C
4388: C      CALCULATE COSINE INSERTING THE PARTICLE
4389:          XP = XX + U*P3
4390:          YP = YY + V*P3
4391:          ZP = ZZ + W*P3
4392:          U1 = X2 - XP
4393:          V1 = Y2 - YP
4394:          W1 = Z2 - ZP
4395:          R1 = SQRT(U1*U1+V1*V1+W1*W1)
4396:          U1 = U1/R1
4397:          V1 = V1/R1
4398:          W1 = W1/R1
4399:          CSN = U*U1 + V*V1 + W*W1
4400:          if ( I.le.NDEB ) write(6,*) ' CSN=', CSN
4401: C
4402:          J = 3
4403:          R = P(J)
4404:          if ( R.le.RAR*2.*REAL(NRMX) ) then
4405:              do 190 K = 1, NRMX*NGMX
4406:                  if ( R.le.DEL*REAL(K) ) go to 200
4407:              190 continue
4408:              RCP(J,K) = RCP(J,K) + 1.
4409:              do 210 L = 1, 20
4410:                  if ( CSN.le..05*REAL(L) ) go to 220
4411:              210 continue
4412:              220 CCN(K,L) = CCN(K,L) + 1.
4413:              end if
4414:          else
4415:              NC = NC + 1
4416:          end if
4417: C
4418: C      END OF HISTORIES
4419:          230 continue
4420: C

```

src/tools/mcrdf.f

```

4421: C      OUTPUT NEAREST NEIGHBOR DISTRIBUTION
4422: C      UNIT 14 : FOR MCNP.CFP
4423:       write(16,'(I5,F10.4)') NRMX*NGMX, 1./REAL(NGMX)
4424: C
4425:       write(6,'(1H1)')
4426:       write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 3R (EXTENDED NND',
4427: & ' CALCULATION WAS DONE)      OUTPUT FILE : FT14'
4428:       write(6,*) ' PARTICLES START IN THE MATRIX.'
4429:       write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
4430:       write(6,*) ' REJECTION NO. : ', NC
4431:       write(6,*) ' NORMAL LEVEL. ', NCO(1)
4432:       write(6,'(1H)')
4433:       write(6,*) ' EXTEND LEVEL. LEVEL NO.'
4434:       I = 1
4435: 240 I = I + 1
4436:       if ( I.ne.100.and.NCO(I).ne.0 ) then
4437:         write(6,'(1H,20X,2I8)') I - 1, NCO(I)
4438:         go to 240
4439:       end if
4440:       write(6,'(1H0)')
4441:       write(6,*) ' NO. DISTANCE IRCP(3)', RCP(3)
4442:       do 250 I = 1, NRMX*NGMX
4443:         RS = REAL(I-1) /REAL(NGMX)
4444:         RE = REAL(I) /REAL(NGMX)
4445:         if ( I.eq.1 ) then
4446:           RCP3 = RCP(3,I)
4447:         else
4448:           RCP3 = RCP(3,I) + RCP3
4449:         end if
4450:         FAC = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
4451:         RCP3J = RCP(3,I)*FAC
4452:         RCP3I = RCP3/REAL(NMAX)
4453:         write(15,'(1P2E12.5)') (RS+RE) /2., RCP3I
4454:         write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
4455: 7000 format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
4456: C
4457:       write(16,'(1P2E12.5)') RCP3I
4458: 250 continue
4459:       write(6,'(1H0)')
4460:       write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT03 FOR NGRAPH.'
4461: C
4462: C      OUTPUT ANGULAR DISTRIBUTION
4463:       write(6,'(1H1)')
4464:       write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
4465:       write(6,*) ' (EXTENDED NND CALCULATION)'
4466:       write(6,*)
4467:       write(6,*) ' COSINE BINS'
4468:       write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'
4469:       do 260 I1 = 1, 20
4470:         write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
4471: 260 continue
4472: 7020 format(' ',I6,6X,2F10.7)
4473:       write(6,'(1H0)')
4474:       write(6,*) ' NO. DISTANCE ANG. DIST.(DP/DT)'
4475:       do 270 I = 1, NRMX*NGMX
4476:         RS = REAL(I-1) /REAL(NGMX)
4477:         RE = REAL(I) /REAL(NGMX)
4478:         if ( RCP(3,I).le.0 ) then
4479:           write(6,*) I, ' : RCP IS 0'
4480:           go to 270
4481:         end if
4482:         write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
4483:         write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
4484: 270 continue
4485: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)

```

```

4486: 7060 format(' ',23X,1P10E10.3)
4487: C
4488:       return
4489:       end
4490: C
4491: C=====
4492: C
4493:       subroutine NND3R4( PF2 )
4494: C      SUBROUTINE NND2
4495: C      CALCULATE NEAREST NEIGHBOR DISTRIBUTION
4496: C      START PARTICLE FROM THE MATRIX
4497: C      THIS SUBROUTINE IS EXACTLY THE SAME AS NND2.
4498: C
4499: C      OUTPUT FILE : FT15; NND OUTPUT FOR PLOT
4500: C      FT16; NND OUTPUT FOR MCNP-CFP
4501: C
4502:       parameter( NRY = 1000, MPX = 1000 )
4503:       parameter( PI = 3.141592653 )
4504:       common /INPDT/ PFC, AAR, IBOX
4505:       common /TRNCM/ XTR(27), YTR(27), ZTR(27)
4506:       common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
4507: & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
4508: & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
4509: & NNR
4510:       common /CRC/ DIS(MPX), P(3), IDIS(NRY,MPX),
4511: & ISIS(NRY,MPX), IDD(NRY), ISS(MPX)
4512: C
4513:       dimension CCN(MPX,100)
4514: C      DIMENSION FOR EXTENDED NND CALCULATION
4515:       dimension ISSS(NRY), IDSS(NRY), NCO(100)
4516: C      DIMENSION IDSS(NRY),NCO(100)
4517: C
4518:       NDEB = 0
4519:       NC = 0
4520:       do 100 I = 1, 100
4521:         NCO(I) = 0
4522: 100 continue
4523:       do 110 I = 1, MPX
4524:         RCP(3,I) = 0.
4525: 110 continue
4526: C      MAXIMUM LENGTH OF NND
4527:       DDDT = 30.0
4528:       NRMX = INT(DDDT/RAR/2.+0.001)
4529: C      NRMX=INT(AAR/2./RAR/2.)
4530:       NGMX = 20
4531:       DEL = RAR*2./REAL(NGMX)
4532:       NRYU = NRY
4533:       if ( NRY.gt.NNR ) NRYU = NNR
4534: C
4535: C      THE MAXIMUM NUMBER OF SAMPLING
4536:       NMAX = 100000
4537: C
4538:       do 230 I = 1, NMAX
4539: C
4540:         DDD = 0.
4541:         DDDD = 0.
4542:         ILV = 0
4543: C
4544: C      CHECK THE LENGTH OF THE FLIGHT PATH
4545: 120 DDD = DDD + DDDD
4546:       if ( DDD.lt.DDDT ) then
4547:         ILV = ILV + 1
4548: C
4549: C      DETERMINE THE INCIDENT PARTICLE CORDINATION.
4550: 130 XX = (RANG()-0.5)*(AAR-2.*RAR)

```

src/tools/mcrdf.f

```

4551:          YY      = (RANG()-0.5)*(AAR-2.*RAR)
4552:          ZZ      = (RANG()-0.5)*(AAR-2.*RAR)
4553: C
4554: C      RE-ORDER BY THE LENGTH
4555:       do 140 J = 1, NNR
4556:         DIS(J) = DIST(XX,YY,ZZ,XXX(J),YYY(J),ZZZ(J),III)
4557:         if ( DIS(J).le.RAR ) go to 130
4558:         ISS(J) = III
4559: 140       continue
4560:       call SORT2( DIS(1), NNR, IDD(1), NRYU )
4561:       do 150 K = 1, NRYU
4562:         ISSS(K) = ISS(IDD(K))
4563:         IDSS(K) = IDD(K)
4564: 150       continue
4565: C
4566: C      DETERMINE INSERTING DIRECTION OF THE PARTICLE
4567: 160       UP      = RANG()*2. - 1.
4568:       VP      = RANG()*2. - 1.
4569:       WP      = RANG()*2. - 1.
4570:       R      = SQRT(UP*UP+VP*VP+WP*WP)
4571:       if ( R.gt.1. ) go to 160
4572:       U      = UP/R
4573:       V      = VP/R
4574:       W      = WP/R
4575: C
4576: C      CALCULATE LENGTH TO THE BOUNDARY.
4577:       DDDD = 1.0E10
4578: C      REACH THE PLANE NORMAL TO THE X AXIS
4579:       if ( U.ne.0. ) then
4580:         XB      = AAR/2. - RAR
4581:         TT      = XB/U
4582:         if ( TT.lt.0. ) then
4583:           XB      = -AAR/2. + RAR
4584:           TT      = XB/U
4585:         if ( TT.ge.0. ) then
4586:           if ( TT.lt.DDDD ) DDDD = TT
4587:         end if
4588:       else
4589:         if ( TT.lt.DDDD ) DDDD = TT
4590:       end if
4591:     end if
4592: C      REACH THE PLANE NORMAL TO THE X AXIS
4593:       if ( V.ne.0. ) then
4594:         YB      = AAR/2. - RAR
4595:         TT      = YB/V
4596:         if ( TT.lt.0. ) then
4597:           YB      = -AAR/2. + RAR
4598:           TT      = YB/V
4599:         if ( TT.ge.0. ) then
4600:           if ( TT.lt.DDDD ) DDDD = TT
4601:         end if
4602:       else
4603:         if ( TT.lt.DDDD ) DDDD = TT
4604:       end if
4605:     end if
4606: C      REACH THE PLANE NORMAL TO THE Z AXIS
4607:       if ( W.ne.0. ) then
4608:         ZB      = AAR/2. - RAR
4609:         TT      = ZB/W
4610:         if ( TT.lt.0. ) then
4611:           ZB      = -AAR/2. + RAR
4612:           TT      = ZB/W
4613:         if ( TT.ge.0. ) then
4614:           if ( TT.lt.DDDD ) DDDD = TT
4615:         end if

```

```

4616:       else
4617:         if ( TT.lt.DDDD ) DDDD = TT
4618:       end if
4619:     end if
4620: C
4621: C      DETERMINE PARTICLE WHICH IS TRACKED EARLIEST
4622: C
4623: C      FIND THE PARTICLE WHICH CONTACTS WITHIN RAR.
4624:       do 170 J = 1, NRYU
4625:         X2      = XXX(IDSS(J)) + XTR(ISSS(J))
4626:         Y2      = YYY(IDSS(J)) + YTR(ISSS(J))
4627:         Z2      = ZZZ(IDSS(J)) + ZTR(ISSS(J))
4628:         T      = (X2-XX)*U + (Y2-YY)*V + (Z2-ZZ)*W
4629:         if ( I.le.NDEB ) write(6,*) ' X2=', X2, ' Y2=', Y2,
4630:           ' Z2=', Z2
4631:         &
4632:         if ( I.le.NDEB ) write(6,*) ' T=', T
4633:         if ( T.gt.0. ) then
4634:           X      = T*U + XX
4635:           Y      = T*V + YY
4636:           Z      = T*W + ZZ
4637:           D      =
4638:             &
4639:             &
4640:             &
4641:             &
4642:             &
4643:             &
4644:             &
4645:             &
4646:             &
4647:             &
4648:             &
4649:             &
4650:             &
4651:             &
4652:             &
4653:             &
4654:             &
4655:             &
4656:             &
4657:             &
4658:             &
4659:             &
4660:             &
4661:             &
4662:             &
4663:             &
4664:             &
4665:             &
4666:             &
4667:             &
4668:             &
4669:             &
4670:             &
4671:             &
4672:             &
4673:             &
4674:             &
4675:             &
4676:             &
4677:             &
4678:             &
4679:             &
4680:             &

```

src/tools/mcrdf.f

```

4681:         if ( R.le.RAR*2.*REAL(NRMX) ) then
4682:             do 190 K = 1, NRMX*NGMX
4683:                 if ( R.le.DEL*REAL(K) ) go to 200
4684:             190 continue
4685:             200 RCP(J,K) = RCP(J,K) + 1.
4686:                 do 210 L = 1, 20
4687:                     if ( CSN.le..05*REAL(L) ) go to 220
4688:                 210 continue
4689:                 220 CCN(K,L) = CCN(K,L) + 1.
4690:             end if
4691:         else
4692:             NC = NC + 1
4693:         end if
4694: C
4695: C     END OF HISTORIES
4696: 230 continue
4697: C
4698: C     OUTPUT NEAREST NEIGHBOR DISTRIBUTION
4699: C     UNIT 16 : FOR MCNP-CFP
4700: write(16,'(I5,2F10.4)') NRMX*NGMX, 1./REAL(NGMX), PF2
4701: C
4702: write(6,'(1H1)')
4703: write(6,*) ' NEAREST NEIGHBOR DISTRIBUTION 3 (EXTENDED NND',
4704: & ' CALCULATION WAS DONE) OUTPUT FILE : FT16'
4705: write(6,*) ' PARTICLES START IN THE MATRIX'
4706: write(6,*) ' ', NMAX, ' PARTICLE HISTORIES WERE DONE'
4707: write(6,*) ' REJECTION NO. : ', NC
4708: write(6,*) ' NORMAL LEVEL. ', NCO(I)
4709: write(6,'(1H )')
4710: write(6,*) ' EXTEND LEVEL. LEVEL NO.'
4711: I = 1
4712: 240 I = I + 1
4713: if ( I.ne.100.and.NCO(I).ne.0 ) then
4714:     write(6,'(1H ,20X,2I8)') I - 1, NCO(I)
4715:     go to 240
4716: end if
4717: write(6,'(1H0)')
4718: write(6,*) ' NO. DISTANCE IRCP(3)', ' RCP(3)'
4719: do 250 I = 1, NRMX*NGMX
4720:     RS = REAL(I-1) /REAL(NGMX)
4721:     RE = REAL(I) /REAL(NGMX)
4722:     if ( I.eq.1 ) then
4723:         RCP3 = RCP(3,I)
4724:     else
4725:         RCP3 = RCP(3,I) + RCP3
4726:     end if
4727:     FAC = RAR*RAR*RAR/REAL(NMAX) /PFC/(RE*RE*RE-RS*RS*RS)
4728:     RCP3J = RCP(3,I)*FAC
4729:     RCP3I = RCP3/REAL(NMAX)
4730:     write(15,'(1P2E12.5)') (RS+RE) /2., RCP3I
4731:     write(6,7000) I, RS, RE, INT(RCP3), RCP3I, INT(RCP(3,I)), RCP3J
4732: 7000 format(' ',I6,F7.3,' - ',F7.3,4X,I7,4X,F7.4,4X,I7,4X,F7.4)
4733: C
4734:     write(16,'(1P2E12.5)') RCP3I
4735: 250 continue
4736: write(6,'(1H0)')
4737: write(6,*) ' NND FOR MCNP-CFP IS OUTPUT TO FT15 FOR NGRAPH.'
4738: C
4739: C     OUTPUT ANGULAR DISTRIBUTION
4740: write(6,'(1H1)')
4741: write(6,*) ' ANGULAR DISTRIBUTION INSERTING THE PARTICLE'
4742: write(6,*) ' (EXTENDED NND CALCULATION)'
4743: write(6,*)
4744: write(6,*) ' COSINE BINS'
4745: write(6,*) ' GROUP COSINE OF ANGLE BETWEEN NORMAL AND INSETING'

```

```

4746:         do 260 I1 = 1, 20
4747:             write(6,7020) I1, .05*REAL(I1-1), .05*REAL(I1)
4748:         260 continue
4749: 7020 format(' ',I6,6X,2F10.7)
4750: write(6,'(1H0)')
4751: write(6,*) ' NO. DISTANCE ANG. DIST.(DP/DT)'
4752: do 270 I = 1, NRMX*NGMX
4753:     RS = REAL(I-1) /REAL(NGMX)
4754:     RE = REAL(I) /REAL(NGMX)
4755:     if ( RCP(3,I).le.0 ) then
4756:         write(6,*) I, ' : RCP IS 0'
4757:         go to 270
4758:     end if
4759:     write(6,7040) I, RS, RE, (CCN(I,J)/RCP(3,I)/.05,J=1,10)
4760:     write(6,7060) (CCN(I,J)/RCP(3,I)/.05,J=11,20)
4761: 270 continue
4762: 7040 format(' ',I4,F6.3,' - ',F6.3,4X,1P10E10.3)
4763: 7060 format(' ',23X,1P10E10.3)
4764: C
4765:     return
4766: end
4767: C
4768: C=====
4769: C
4770: subroutine XSEC
4771: C     CALCULATE PAIR CORRELATION FUNCTION FOR CROSS SECTION OF COMPACT
4772: C     CUTTING MUST BE DONE BY THE PLANE WHICH IS Z=ZA.
4773: C
4774: parameter( MPX = 1000 )
4775: common /INPDT/ PFC, AAR, IBOX
4776: common /MAINCM/ XXX(MPX), YYY(MPX), ZZZ(MPX),
4777: & RDF(MPX), RCP(3,MPX), RDP2(3,MPX), SIGT,
4778: & XNW(MPX), YNW(MPX), ZNW(MPX), RAR,
4779: & NNR
4780: C
4781: dimension XSC(2), PCF(2,MPX), IXSC(2), NNI(MPX)
4782: C
4783: NRMX = INT(AAR/2./RAR/2.)
4784: NGMX = 20
4785: DEL = RAR*2./REAL(NGMX)
4786: do 110 J = 1, 2
4787:     IXSC(J) = 0
4788:     do 100 I = 1, NRMX*NGMX
4789:         PCF(J,I) = 0.
4790: 100 continue
4791: 110 continue
4792: C
4793: IZA = INT(AAR/RAR/2.)
4794: write(6,'(1H1)')
4795: write(6,*) ' SUB-XSEC'
4796: write(6,*) ' CUTTING PLANE : ', IZA
4797: C
4798: NNTT = 0
4799: do 180 J = 1, IZA
4800:     ZA = -AAR/2. + 2.*RAR*REAL(J-1) + RAR
4801:     write(6,*) ' ', J, ' :ZA=', ZA
4802:     write(10,'(1P2E12.5)') AAR
4803: C
4804: C     FIND PARTICLES CONTAINED IN THE CUTTING PLANE
4805: NN = 0
4806: do 120 I = 1, NNR
4807:     if ( ABS(ZZZ(I)-ZA).le.RAR ) then
4808:         NN = NN + 1
4809:         NNI(NN) = I
4810:     end if

```

src/tools/mcrdf.f

```

4811: 120 continue
4812: write(6,*) ' :NN=', NN
4813: NNTT = NNTT + NN
4814: write(10,'(I4)') NN
4815: C
4816: do 170 I = 1, NN
4817: do 160 K = 1, NN
4818: if ( I.eq.1.and.J.eq.1 ) then
4819: Z1 = ZZZ(NNI(K)) - ZA
4820: Z1 = SQRT(RAR*RAR-Z1*Z1)
4821: write(10,'(1P3E12.5)') XXX(NNI(K)), YYY(NNI(K)), Z1
4822: end if
4823: if ( I.ne.K ) then
4824: XSC(1) =
4825: & DIST(XXX(NNI(I)),YYY(NNI(I)),ZA,XXX(NNI(K)),
4826: & YYY(NNI(K)),ZA,III)
4827: XSC(2) = DIST(XXX(NNI(I)),YYY(NNI(I)),ZZZ(NNI(I)),
4828: & XXX(NNI(K)),YYY(NNI(K)),ZZZ(NNI(K)),III)
4829: C
4830: do 150 J2 = 1, 2
4831: R = XSC(J2)
4832: if ( R.le.RAR*2.*REAL(NRMX) ) then
4833: do 130 K2 = 1, NRMX*NGMX
4834: if ( R.le.DEL*REAL(K2) ) go to 140
4835: 130 continue
4836: 140 PCF(J2,K2) = PCF(J2,K2) + 1.
4837: IXSC(J2) = IXSC(J2) + 1
4838: end if
4839: 150 continue
4840: end if
4841: C
4842: 160 continue
4843: 170 continue
4844: 180 continue
4845: write(6,*) ' TOTAL =', NNTT
4846: C
4847: C OUTPUT PAIR CORRELATION FUNCTION
4848: write(6,'(1H1)')
4849: write(6,*) ' PCF(1) ON THE CUTTING PLANE'
4850: write(6,*) ' PCF(2) CENTER TO CENTER ON THE CUTTING PLANE'
4851: write(6,*) ' TOTAL (1) : ', IXSC(1)
4852: write(6,*) ' TOTAL (2) : ', IXSC(2)
4853: write(6,*)
4854: write(6,*) ' NO. DISTANCE PCF(1) PCF(2)',
4855: & ' PCF(1)/TOT PCF(2)/TOT PCF(1)/DV PCF(2)/DV'
4856: do 190 I = 1, NRMX*NGMX
4857: RS = REAL(I-1) /REAL(NGMX)*2.*RAR
4858: RE = REAL(I) /REAL(NGMX)*2.*RAR
4859: FAC = 2./3.*RAR*RAR/REAL(NNTT) /(RE*RE-RS*RS)
4860: write(17,'(1P2E12.5)') (0.5+REAL(I-1)) /REAL(NGMX), PCF(1,I)*
4861: & FAC
4862: write(6,7000) I, RS, RE, INT(PCF(1,I)), INT(PCF(2,I)), PCF(1,I)
4863: & /REAL(IXSC(1)), PCF(2,I) /REAL(IXSC(2)), PCF(1,I)*FAC,
4864: & PCF(2,I)*FAC
4865: 190 continue
4866: 7000 format(' ',I4,F6.3,' - ',F6.3,4X,2I6,4X,F7.4,2X,F7.4,4X,F7.4,2X,
4867: & F7.4)
4868: C
4869: C
4870: return
4871: end
4872: C
4873: C=====
4874: C
4875: subroutine SORT2( A, NNR, MM, MP )

```

```

4876: C THE FIRST ONE IS NOT OMITTED.
4877: C
4878: parameter( MPX = 1000 )
4879: parameter( ISORT = 1 )
4880: dimension A(1), MM(1), N(MPX)
4881: if ( ISORT.eq.1 ) then
4882: call QSORT2( A, NNR, MM, MP )
4883: go to 140
4884: end if
4885: do 100 I = 1, NNR
4886: N(I) = I
4887: 100 continue
4888: 110 IFLG = 0
4889: IN = 0
4890: 120 IN = IN + 1
4891: if ( A(N(IN+1)).lt.A(N(IN)) ) then
4892: ND = N(IN+1)
4893: N(IN+1) = N(IN)
4894: N(IN) = ND
4895: IFLG = 1
4896: end if
4897: if ( IN.ne.NNR-1 ) go to 120
4898: if ( IFLG.ne.0 ) go to 110
4899: do 130 I = 1, MP
4900: MM(I) = N(I)
4901: 130 continue
4902: C WRITE(6,'(25I4)')(N(I),I=1,NNR)
4903: 140 return
4904: end
4905: C
4906: C=====
4907: C
4908: subroutine SORT( A, NNR, MM, MP )
4909: C
4910: parameter( MPX = 1000 )
4911: parameter( ISORT = 1 )
4912: dimension A(1), MM(1), N(MPX)
4913: if ( ISORT.eq.1 ) then
4914: call QSORT( A, NNR, MM, MP )
4915: go to 140
4916: end if
4917: do 100 I = 1, NNR
4918: N(I) = I
4919: 100 continue
4920: 110 IFLG = 0
4921: IN = 0
4922: 120 IN = IN + 1
4923: if ( A(N(IN+1)).lt.A(N(IN)) ) then
4924: ND = N(IN+1)
4925: N(IN+1) = N(IN)
4926: N(IN) = ND
4927: IFLG = 1
4928: end if
4929: if ( IN.ne.NNR-1 ) go to 120
4930: if ( IFLG.ne.0 ) go to 110
4931: do 130 I = 1, MP
4932: MM(I) = N(I+1)
4933: 130 continue
4934: C WRITE(6,'(25I4)')(N(I),I=1,NNR)
4935: 140 return
4936: end
4937: C
4938: C=====
4939: C
4940: C DIMENSION D(10),MM(5)

```

src/tools/mcrdf.f

```

4941: C      DATA D / 9,8,7,6,5,4,3,2,1,7.5 /
4942: C      CALL QSORT(D,10,MM,5)
4943: C      WRITE(6,*)D
4944: C      WRITE(6,*)MM
4945: C      STOP
4946: C      END
4947: C *****
4948: C *          Q U I C K S O R T          *
4949: C *****
4950: C
4951: C      .... Modified by M.Sasaki not to use DO UNTIL/WHILE structure.
4952: C
4953: C      subroutine QSORT( DSORT, N,      MM,      MP )
4954: C      parameter( MPX = 1000 )
4955: C      DIMENSION DSORT(N),MM(1),NN(MPX) ... modified M.Sasaki
4956: C      dimension DSORT(N), MM(1), NN(MPX+1)
4957: C      dimension NSTACK(0:50)
4958: C
4959: C      do 100 I = 1, N
4960: C          NN(I) = I
4961: C      100 continue
4962: C      M0 = 1
4963: C      M1 = N
4964: C      M00 = M0
4965: C      M11 = M1
4966: C      IPRT = 0
4967: C      NSTACK(IPRT) = -1
4968: C
4969: C      DO 100 UNTIL ( IPRT .EQ. 0 )
4970: C
4971: C      DO 200 WHILE ( M1-M0 .GT. 10 )
4972: C      110 if ( M1-M0.gt.10 ) then
4973: C          M2 = (M0+M1) /2
4974: C          XKEY = DSORT(NN(M2))
4975: C          I = M0 - 1
4976: C          J = M1 + 1
4977: C          IFLG = 1
4978: C
4979: C      CCCCC DO 300 WHILE ( IFLG .EQ. 1 )
4980: C      120 if ( IFLG.eq.1 ) then
4981: C      130 I = I + 1
4982: C          if ( DSORT(NN(I)).lt.XKEY ) go to 130
4983: C      140 J = J - 1
4984: C          if ( DSORT(NN(J)).gt.XKEY ) go to 140
4985: C          if ( I.lt.J ) then
4986: C              ND = NN(I)
4987: C              NN(I) = NN(J)
4988: C              NN(J) = ND
4989: C          else
4990: C              IFLG = 0
4991: C              if ( I.eq.J ) J = J - 1
4992: C          end if
4993: C          go to 120
4994: C      end if
4995: C      300 CONTINUE
4996: C
4997: C      IPRT = IPRT + 1
4998: C      NSTACK(IPRT) = M1
4999: C      M1 = J
5000: C      go to 110
5001: C      end if
5002: C      200 CONTINUE
5003: C
5004: C      DO 600 WHILE ( NSTACK(IPRT) .EQ. 0 )
5005: C          IPRT = IPRT - 1

```

```

5006: C      600 CONTINUE
5007: C      150 if ( NSTACK(IPRT).eq.0 ) then
5008: C          IPRT = IPRT - 1
5009: C          go to 150
5010: C      end if
5011: C
5012: C      M0 = M1 + 1
5013: C      M1 = NSTACK(IPRT)
5014: C      NSTACK(IPRT) = 0
5015: C      if ( M0.eq.M00.and.M1.eq.M11 ) M0 = M0 + 1
5016: C      M00 = M0
5017: C      M11 = M1
5018: C      if ( IPRT.ne.0 ) go to 110
5019: C      100 CONTINUE
5020: C
5021: C      SIMPLE INSERTION METHOD
5022: C
5023: C      do 170 K = N - 1, 1, -1
5024: C          NN(N+1) = NN(K)
5025: C          M = K + 1
5026: C      160 if ( DSORT(NN(M)).lt.DSORT(NN(N+1)) ) then
5027: C          NN(M-1) = NN(M)
5028: C          M = M + 1
5029: C          go to 160
5030: C      end if
5031: C      NN(M-1) = NN(N+1)
5032: C      170 continue
5033: C
5034: C      do 180 I = 1, MP
5035: C          MM(I) = NN(I+1)
5036: C      180 continue
5037: C
5038: C      return
5039: C      end
5040: C
5041: C=====
5042: C
5043: C      subroutine QSORT2( DSORT, N,      MM,      MP )
5044: C          THE FIRST ONE IS NOT OMITTED.
5045: C
5046: C      .... Modified by M.Sasaki not to use DO UNTIL/WHILE structure.
5047: C
5048: C      parameter( MPX = 1000 )
5049: C      CCCC DIMENSION DSORT(N),MM(1),NN(MPX) ... modified M.Sasaki
5050: C      dimension DSORT(N), MM(1), NN(MPX+1)
5051: C      dimension NSTACK(0:50)
5052: C
5053: C      do 100 I = 1, N
5054: C          NN(I) = I
5055: C      100 continue
5056: C      M0 = 1
5057: C      M1 = N
5058: C      M00 = M0
5059: C      M11 = M1
5060: C      IPRT = 0
5061: C      NSTACK(IPRT) = -1
5062: C
5063: C      DO 100 UNTIL ( IPRT .EQ. 0 )
5064: C
5065: C      DO 200 WHILE ( M1-M0 .GT. 10 )
5066: C      110 if ( M1-M0.gt.10 ) then
5067: C          M2 = (M0+M1) /2
5068: C          XKEY = DSORT(NN(M2))
5069: C          I = M0 - 1
5070: C          J = M1 + 1

```

src/tools/mcrdf.f

```

5071:      IFLG      = 1
5072: C
5073: CCCC      DO 300 WHILE ( IFLG .EQ. 1 )
5074: 120      if ( IFLG.eq.1 ) then
5075: 130          I      = I + 1
5076:          if ( DSORT(NN(I)).lt.XKEY ) go to 130
5077: 140          J      = J - 1
5078:          if ( DSORT(NN(J)).gt.XKEY ) go to 140
5079:          if ( I.lt.J ) then
5080:              ND      = NN(I)
5081:              NN(I)    = NN(J)
5082:              NN(J)    = ND
5083:          else
5084:              IFLG      = 0
5085:              if ( I.eq.J ) J = J - 1
5086:          end if
5087:          go to 120
5088:      end if
5089: C 300      CONTINUE
5090: C
5091:          IPRT      = IPRT + 1
5092:          NSTACK(IPRT) = M1
5093:          M1        = J
5094:          go to 110
5095:      end if
5096: C 200      CONTINUE
5097: C
5098:
5099: C      DO 600 WHILE ( NSTACK(IPRT) .EQ. 0 )
5100: C          IPRT = IPRT - 1
5101: C 600      CONTINUE
5102:
5103: 150 if ( NSTACK(IPRT).eq.0 ) then
5104:     IPRT = IPRT - 1
5105:     go to 150
5106: end if
5107: C
5108:     M0      = M1 + 1
5109:     M1      = NSTACK(IPRT)
5110:     NSTACK(IPRT) = 0
5111:     if ( M0.eq.M00.and.M1.eq.M11 ) M0 = M0 + 1
5112:     M00     = M0
5113:     M11     = M1
5114:
5115:     if ( IPRT.ne.0 ) go to 110
5116: C 100      CONTINUE
5117: C
5118: C      SIMPLE INSERTION METHOD
5119: C
5120:     do 170 K = N - 1, 1, -1
5121:         NN(N+1) = NN(K)
5122:         M      = K + 1
5123: 160     if ( DSORT(NN(M)).lt.DSORT(NN(N+1)) ) then
5124:         NN(M-1) = NN(M)
5125:         M      = M + 1
5126:         go to 160
5127:     end if
5128:     NN(M-1) = NN(N+1)
5129: 170 continue
5130: C
5131:     do 180 I = 1, MP
5132:         MM(I) = NN(I)
5133: 180 continue
5134: C
5135:     return

```

```

5136:      end
5137: C
5138: C=====
5139: C
5140: *      FUNCTION RANG()
5141: C          SAMPLING UNIFORM RANDOM NUMBER
5142: *      COMMON /RANCM/ IX,NRN
5143: *      CALL RANU2(IX,A,1,ICON)
5144: *      NRN=NRN+1
5145: *      RANG=A
5146: C      IF(NRN.LE.50) WRITE(6,*) ' NRN=',NRN,' RANG=',A
5147: *      RETURN
5148: *      END
5149: C
5150: C      ... Modified by M.Sasaki (call RANU2 every NRAN call)
5151: C
5152: function RANG()
5153: C      SAMPLING UNIFORM RANDOM NUMBER
5154: parameter( NRAN = 2048 )
5155: common /RANCM/ IX, NRN, A(NRAN)
5156: data NUSED /NRAN/
5157: data NINIT /0/
5158: C
5159: if ( NINIT.eq.0 ) then
5160:     call RNINIT( 1 )
5161:     NINIT = 1
5162: end if
5163: if ( NUSED.eq.NRAN ) then
5164:     call RANU2( IX, A, NRAN, ICON )
5165:     NUSED = 0
5166: end if
5167: NRN = NRN + 1
5168: NUSED = NUSED + 1
5169: RANG = A(NUSED)
5170: CCCC IF(NRN.LE.50) WRITE(6,*) ' NRN=',NRN,' RANG=',RANG
5171: return
5172: end
5173: C
5174: C=====
5175: C
5176: subroutine NNDMVP
5177: C
5178: parameter (MAXNND=2000)
5179: real XNND(MAXNND,3)
5180: C
5181: C      .... read NND1
5182: C
5183: rewind(12)
5184: read(12,*) N1, RD1
5185: read(12,*) (XNND(I,1),I=1,N1)
5186: C
5187: C      .... read NND2
5188: C
5189: rewind(14)
5190: read(14,*) N2, RD2
5191: read(14,*) (XNND(I,2),I=1,N2)
5192: C
5193: C      .... read NND3
5194: C
5195: rewind(16)
5196: read(16,*) N3, RD3, PF
5197: read(16,*) (XNND(I,3),I=1,N3)
5198: C
5199: C ***** output format *****
5200: C

```

src/tools/mcrdf.f

```
5201: C      NND-library of MVP : in free form
5202: C
5203: C (0) Version /** version string of file form
5204: C      Currently this is "version1.0"
5205: C (1) Description /* arbitrary message upto 80 characters
5206: C (2) NPF /* number of NND sets by packing fractions
5207: C (3) PF(I),I=1,NPF /** list of packing fractions
5208: C (4) repetition of NND sets for each packing fraction
5209: C
5210: C      * record having string "###" on the head (separator)
5211: C      * PF, RD, N /** fraction, radii division unit, number of division
5212: C
5213: C      And for I = 1 to N
5214: C
5215: C      * NND1(I),NND2(I),NND3(I) /** NND-1, NND-2, NND-3
5216: C
5217: C *****
5218: C
5219: C      This routine outputs record group (4) only
5220: C
5221: C
5222: C      write(20,7000) PF, PF, RD1, N1
5223: C      7000 format('## PF =',e15.7/2x,e15.7,2x,e15.7,2x,i6)
5224: C      write(20,7200) (XNND(I,1),XNND(I,2),XNND(I,3),I=1,N1)
5225: C      7200 format(3(1x,e15.7))
5226: C
5227: C
5228: C      .... output as MVP/GMVP input form
5229: C
5230: C      write(25,7100) PF, RD1, RD1, RD1
5231: C      7100 format(2x,'FP(',e15.7,' )'/
5232: C      &      2x,'WNND1(',e15.7,' )'/
5233: C      &      2x,'WNND2(',e15.7,' )'/
5234: C      &      2x,'WNND3(',e15.7,' )'/
5235: C
5236: C      7120 format(2x,a,'('/(1x,5(1x,1p,e13.5:)))
5237: C
5238: C      write(25,7120) 'FNND1',(XNND(I,1),I=1,N1)
5239: C      write(25,'('' )''')
5240: C      write(25,7120) 'FNND2',(XNND(I,2),I=1,N1)
5241: C      write(25,'('' )''')
5242: C      write(25,7120) 'FNND3',(XNND(I,3),I=1,N1)
5243: C      write(25,'('' )''')
5244: C
5245: C      return
5246: C      end
5247: C
5248: C=====
5249: C
5250: C      ... CPU time monitor here uses CPUTM routine in MVP's src/utills.
5251: C
5252: C      subroutine CLOCKM( MM )
5253: C      call CPUTM( X )
5254: C      MM      = 1000.0*X
5255: C      return
5256: C      end
```


src/tools/mvpdump.f

```

1: C
2: C*****
3: C MVPDUMP :
4: C   A utility program to printout contents of MVP neutron library.
5: C
6: C   Reference: JAERI-Data/Code 96-018
7: C
8: C*****
9: C
10: C Control data from standard input device:
11: C
12: C   All items are input by list-directed read statement (read(*,*)).
13: C
14: C (0) File name of the library file (up to 80 characters, sorry!!)
15: C
16: C   or pair of two lines ;
17: C
18: C   PATH=directory-path-to-library
19: C   filename
20: C
21: C   e.g.
22: C
23: C       PATH=/home/MVPlib/longpath1/longpath2/longpath3/longpath4/
24: C       U08003J3.this-is-a-silly-long-file-name-for-sample
25: C
26: C   Specifies a file (too long, "... " means abbreviation) ;
27: C
28: C       /home/MVPlib/.../longpath4/U08003J3.this-is-a-...
29: C
30: C   "PATH=..." is effective until overridden by next "PATH=...".
31: C
32: C   ((This record is added in modification in 1999, and undocumented
33: C   in JAERI-Data/Code 96-018.))
34: C
35: C (1) NIN,IDUMP /
36: C
37: C   NIN : Fortran I/O unit to which input library data is connected
38: C   (ineffective in 1999 modification)
39: C   IDUMP : printout mode (from page62 of JAERI-Data/Code 96-018)
40: C
41: C -----
42: C           IDUMP      Neutron data      gamma-production data
43: C           -----
44: C                   rec #1,#2      rec #3      rec #1,#2      rec #3
45: C -----
46: C                   0              No          No          No          No
47: C                   1              Yes         Yes         No          No
48: C                   11             Yes         Yes         Yes         Yes
49: C                   13             Yes         Yes         Yes         No
50: C                   -1             Yes         No          Yes         Yes
51: C                   -3             Yes         No          Yes         No
52: C -----
53: C
54: C (2) EHI, ELOW /
55: C
56: C   Range of energy   EHI > E > ELOW
57: C
58: C
59: C (3) IPRF1,IPRX3,IPRX4,IPRX5,IPRUNR /
60: C
61: C   Detailed print control (undocumented in JAERI-Data/Code 96-018)
62: C
63: C   Repeat (0) to (3).
64: C
65: C   A blank record for (0) or (1) with NIN = IDUMP = 0 terminates

```

```

66: C   the program.
67: C
68: C*****
69: C
70: C   An example of input:
71: C
72: C col .....
73: C   PATH=/home/long/and/winding/road/MVPlib/MVPLIB94/
74: C   n300j3/U08W03J3.j31
75: C   11 11 /
76: C   1.0E7 0.1 /
77: C   1 1 1 1 1 /
78: C   n300j3/H01003J3.j31
79: C   11 11 /
80: C   1.0E7 0.1 /
81: C   1 1 1 1 1 /
82: C   n300j3/H01P03J3.j31
83: C   11 11 /
84: C   1.0E7 0.1 /
85: C   1 1 1 1 1 /
86: C
87: C col .....
88: C
89: C*****
90: C
91: C=====
92: C
93: C *****
94: C CONTROL ROUTINE FOR MVP MATERIAL LIBRARY DUMP
95: C *****
96: C
97: C   PARAMETER      ( MTMAX = 120 , NKMAX = 90 )
98: C   PARAMETER      ( MEMORY = 300 0000 )
99: C
100: C
101: C   INTEGER OTAPE
102: C
103: C   COMMON /UNITS / OTAPE
104: C
105: C   COMMON /HEADER/ C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
106: C
107: C   COMMON /MAINIO/ NSYSI,NSYSO,NOUT,NWRK
108: C   COMMON /CONTRL/ MATD,ZA,ELUNR,EHUNR,CRSMIN,JEMORY,IUNRES,INTUNR,
109: C   + IDUMP,IDBG
110: C   COMMON /THMCNT/ METHDT,NATOM,MODF4T
111: C
112: C   COMMON /HEAD1 / NPTS,NTDATA,NEUNR,NUNR,NLEV,NBINA,NBINE,NNU,NMT,
113: C   + NNK,IGFLAG,LFI,LNU,ITHERM,ISTU,IENDU,LSUNR,LSNU,
114: C   + NBINA1,NBINE1,KDUMMY(10),ATW,TLAB,ETHERM,ESAB,
115: C   + ELOW,EHI,RDUMMY(13),LNUD,NNUD,NNF,LSNUD
116: C   CMOD + ELOW,EHI,RDUMMY(13)
117: C   COMMON /HEAD2 / MTINFO(MTMAX),MTPAR(MTMAX),ISTMT(MTMAX),
118: C   + IENDMT(MTMAX),NEUMT(MTMAX),LCTMT(MTMAX),
119: C   + NEANG(MTMAX),NKF5(MTMAX),NEF5(MTMAX),
120: C   + MTHR(MTMAX),LSTF3(MTMAX),LSTF4(MTMAX),
121: C   + LSTF5(MTMAX),
122: C   + NEF5S(NKMAX,MTMAX),LFF5S(NKMAX,MTMAX),
123: C   + LSTF5S(NKMAX,MTMAX),QVAL(MTMAX),
124: C   + QVAL2(MTMAX),IORDMT(MTMAX)
125: C
126: C   CADDED NCAP and after FOR VERSION-2 BY JAIS K.KANEKO 10/30/1991
127: C   COMMON /HEAD3 / NOGAM,NOGAM3,MTGAM(MTMAX),NGTYPE(MTMAX),
128: C   @ NKGAM(MTMAX),N14GAM(MTMAX),N15GAM(MTMAX),
129: C   @ LSTGAM(MTMAX),MTTOG(MTMAX),NEGYLD(MTMAX),
130: C   @ NEF5G(MTMAX),NKF5G(MTMAX),LCTMTG(MTMAX)
131: C   @ ,NCAP,NCAPG,MTCAP(MTMAX),MTCAPG(MTMAX),
132: C   @ EG1L(MTMAX),EG2U(MTMAX)

```

src/tools/mvpdump.f

```
131: C
132:      COMMON /F6CONT/ LF6,LSTF6,MTTOF6(MTMAX),EF61L(MTMAX),
133:      +          EF62U(MTMAX)
134: C
135: C
136:      REAL*4          A(MEMORY)
137: C
138: Cadded.M.S. July 1999
139: C
140: C ... some f77 compilers panics when large local array is placed even
141: C ... in main routine.
142: C
143: C ... Some versions of g77 claim about argumet type mismatch and don't
144: C ... generate objects (this should be a bug) when compiled as
145: C ... all-in-one source code. So IA(*) is declared and used for
146: C ... real arguments declared as integer on dummy arguments.
147: C
148:      common /mbuffer/ A
149:      integer      IA(MEMORY)
150:      equivalence( A(1),IA(1) )
151: Cend.M.S.
152: C
153:      INTEGER*4      HDATE(2)
154:      CHARACTER*8     MATT
155:      CHARACTER*120   PTITLE
156: C
157: Cadded.M.S. July 1999
158:      character*256 filen
159:      character*80  filen0
160:      character*80  dpath
161:      logical opend
162: Cend.M.S.
163: C
164: C-----INITIAL SET
165: C
166:      NSYSI   = 5
167:      NSYSO   = 9
168:      NWRK    = 1
169:      NATOM   = 0
170:      MODF4T  = 0
171: CDEL  OTAPE = 2
172:      OTAPE   = 0
173: C
174:      NOUT    = 10
175:      filen   = ' '
176:      filen0  = ' '
177:      dpath   = ' '
178: C
179:      IF(OTAPE.GT.0) THEN
180:          REWIND OTAPE
181:          WRITE(OTAPE,11) 777,0,0,0
182:          NOSEQ = 1
183:      ENDIF
184: C
185:      11 FORMAT(50H ENDF OUTPUT FROM MVPDUMP CODE
186:      +          16X,I4,I2,I3,I5)
187: C
188: C      LOOP OF NUCLIDE
189: C
190: C 101 WRITE(6,*) ' ** ENTER NOUT IDUMP IN FREE FORMAT **'
191: C 101 WRITE(6,*) ' ** ENTER Filename or PATH=... **'
192:      filen0 = ' '
193:      READ (NSYSI,'(a)') filen0
194: C
195:      if(filen0.eq.' ') goto 999
```

```
196: C
197:      if ( filen0(1:5).eq.'PATH=' ) then
198:          dpath = filen0(6:)
199:          WRITE(6,*) ' >>> Directory path : <',dpath(:iclen2(dpath)),'>'
200:          WRITE(6,*) ' ** ENTER Filename **'
201:          READ (NSYSI,'(a)') filen0
202:      end if
203: C
204:      if ( dpath.eq.' ' ) then
205:          filen = filen0
206:      else
207:          filen = dpath(:iclen2(dpath))//filen0
208:      end if
209: C
210:      WRITE(6,*) ' >>> File name : <',filen(:iclen2(filen)),'>'
211: C
212:      inquire(unit=nout, opened=open)
213:      if ( open ) close(nout)
214: C
215:      open(unit=nout,form='UNFORMATTED',status='OLD',IOSTAT=ios,
216:      &      file=filen(:iclen2(filen)) )
217: C
218:      if ( IOS.ne.0 ) then
219:          write(6,*) 'XXX Failed to open library file : code ',ios
220:          write(6,*) ' filename=<',filen(:iclen2(filen)),'>'
221:          stop 888
222:      end if
223: C
224:      WRITE(6,*) ' ** ENTER NOUT IDUMP IN FREE FORMAT **'
225: C
226:      READ (NSYSI,*) Ndumy,IDUMP
227:      IF(Ndumy .LE.0) GO TO 999
228:      IF(IDUMP.EQ.0) GO TO 101
229: C
230:      ESAB   = 0
231:      TSTAR  = 0.0
232:      IRC    = 0
233:      NPTS   = 0
234:      NTDATA = 0
235:      NEUNR  = 0
236:      NUNR   = 0
237:      NLEV   = 0
238:      NBINA  = 0
239:      NBINE  = 0
240:      NNU    = 0
241:      NMT    = MTMAX
242:      NNK    = NKMAX
243:      IGFLAG = 0
244:      LFI    = 0
245:      LNU    = 0
246:      ITERM  = 0
247:      ISTU   = 0
248:      IENDU  = 0
249:      LSUNR  = 0
250:      LSNU   = 0
251:      NBINA1 = 0
252:      NBINE1 = 0
253:      ATW    = 0.0
254:      TLAB   = 0.0
255:      ETHERM = 0.0
256:      ESAB   = 0.0
257:      ELOW   = 0.0
258:      EHI    = 0.0
259: CCMOD
260:      LSNUD  = 0
```

src/tools/mvpdump.f

```

261:      NNUD      = 0
262:      LNUD      = 0
263:      NNF       = 0
264:      NOGAM     = 0
265:      NOGAM3    = 0
266:      NCAP      = 0
267:      NCAPG     = 0
268:      LF6       = 0
269:      LSTF6     = 0
270: CEND
271:      CALL CLEA( RDUMMY , 13 , 0.0 )
272:      CALL ICLEA( KDUMMY , 10 , 0 )
273:      LENG = MTMAX*(13 + NKMAX*3)
274:      CALL ICLEA( MTINFO , LENG , 0 )
275:      CALL CLEA( QVAL , MTMAX , 0.0 )
276:      CALL CLEA( QVAL2 , MTMAX , 0.0 )
277:      LENG = MTMAX*10
278:      CALL ICLEA( MTGAM , LENG , 0 )
279:      CALL ICLEA( LCTMTG , MTMAX , 1 )
280:      CALL ICLEA( MTCAP , MTMAX , 0 )
281:      CALL ICLEA( MTCAPG , MTMAX , 0 )
282:      CALL CLEA( EG1L , MTMAX , 0.0 )
283:      CALL CLEA( EG2U , MTMAX , 0.0 )
284:      CALL ICLEA( MTTOF6 , MTMAX*3 , 0 )
285: C
286: CCC      REWIND NOUT
287:      call rwind(NOUT)
288:      READ(NOUT) MATT,HDATE,PTITLE,NPTS,NTDATA
289:      REWIND NOUT
290: C
291:      WRITE(6,*) ' *** NPTS = ',NPTS
292: C
293:      LOC1 = 1
294:      LOC2 = LOC1 + NPTS
295:      LOC3 = LOC2 + NTDATA
296:      LOC4 = LOC3 + NPTS
297:      LOC5 = LOC4 + NPTS
298:      LEMORY = MEMORY - LOC5 + 1
299:      CALL DIMCHK('MAIN ' , LOC5 , MEMORY )
300: C
301:      CALL MDUMP( A(LOC1) , A(LOC2) ,IA(LOC2) , A(LOC5) ,IA(LOC5) ,
302:      +          A(LOC3) , A(LOC4) , LEMORY )
303: C
304:      IF(OTAPE.GT.0) THEN
305:          CALL FEND
306:          CALL MEND
307:          NOSEQ = 0
308:          ENDIF
309:      GO TO 101
310: C
311: C *** END OF PROCESS
312: C
313: 999 CONTINUE
314: C
315:      IF(OTAPE.GT.0) THEN
316:          CALL TEND
317:          REWIND OTAPE
318:          ENDIF
319: C
320:      STOP
321:      END
322: C
323: C=====
324: C
325:      SUBROUTINE CARDO(C1H,C2H,L1H,L2H,N1H,N2H)

```

```

326: C
327: C      WRITE ENDF/B CARD.
328: C
329: C***** CHARACTER *****
330: C      CHARACTER*4 KSIGN
331: C***** CHARACTER *****
332: C      INTEGER OTAPE
333: C
334: C      COMMON/UNITS/OTAPE
335: C      COMMON/NORMF/XNORM(6),KEXP(6)
336: C      COMMON/NORMC/KSIGN(6)
337: C      COMMON/HEADER/DUMMY(6),MATH,MFH,MTH,NOSEQ
338: C
339: C      IF(OTAPE.LE.0) RETURN
340: C
341: C      IF(MTH.GT.0) GO TO 10
342: C      CALL SENDO
343: C      RETURN
344: C
345: C-----CONVERT FLOATING POINT NUMBERS TO STANDARD OUTPUT FORM.
346: C      10 CALL NORMX(C1H,XNORM(1),KSIGN(1),KEXP(1))
347: C      CALL NORMX(C2H,XNORM(2),KSIGN(2),KEXP(2))
348: C-----ELIMINATE -0
349: C      IF(IABS(L1H).LE.0) L1H=0
350: C      IF(IABS(L2H).LE.0) L2H=0
351: C      IF(IABS(N1H).LE.0) N1H=0
352: C      IF(IABS(N2H).LE.0) N2H=0
353: C-----OUTPUT CARD IMAGE.
354: C      WRITE(OTAPE,5010) (XNORM(I),KSIGN(I),KEXP(I),I=1,2),
355: C      1 L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
356: C      NOSEQ=NXTSEQ(NOSEQ)
357: C      RETURN
358: C
359: C      5010 FORMAT(2(F8.5,A1,I2),4I11,I4,I2,I3,I5)
360: C      END
361: C
362: C=====
363: C
364: C=====
365: C Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
366: C-----
367: C      ... REWIND OPERATION IN SAFE :
368: C
369: C      History: Programmed by M.Sasaki (J.R.I.)
370: C
371: C      Rewind operation for an un-opened file may cause error-stop
372: C      in some systems.
373: C=====
374: C      subroutine RWIND( IUNIT )
375: C
376: C      logical OPD
377: C      inquire(unit =IUNIT,opened =OPD)
378: C      if ( OPD ) rewind(IUNIT)
379: C      return
380: C      end
381: C
382: C      function ICLEN2(CH)
383: C=====
384: C Utility routine for MVP/GMVP system (JAERI/JRI 1987-1994)
385: C-----
386: C      Get position of the last non-blank character in a character string.
387: C HISTORY: PROGRAMMED BY M.SASAKI (Jul 1994)
388: C=====
389: C      character*(*) CH
390: C      do 100 I = LEN(CH), 1, -1

```

src/tools/mvpdump.f

```

391:         if ( CH(I:I).ne.' ' ) then
392:             ICLEN2 = I
393:             return
394:         end if
395:     100 continue
396:         ICLEN2 = 0
397:         return
398:     end
399: C
400: C=====
401: C
402: C SUBROUTINE CHKWRT(MEMBER,IA,A,LENG,IOPT,IPRINT)
403: C *****
404: C CHECK WRITER ROUTINE FOR 1-D ARRAY DATA.
405: C =====
406: C MEMBER : MEMNER NAME OR COMMENT
407: C IA      : INTEGER OR CHARACTER ARRAY TO BE PRINT
408: C A       : REAL ARRAY
409: C LENG    : LENGTH OF 1-D ARRAY (WORD UNIT)
410: C IOPT     : OPTION OF CHECK WRITER
411: C           =1/2/3:INTERGER/REAL/CAHRACTER
412: C IPRINT   : WRITER UNIT NUMBER
413: C *****
414: C COMMON /CONTRL/ MATD,ZA,ELUNR,EHUNR,CRSMIN,JEMORY,IUNRES,INTUMR,
415: C + IDUMP,IDBG
416: C
417: C REAL*4      A(1)
418: C INTEGER*4    IA(1)
419: C CHARACTER*8  MEMBER
420: C
421: C IF(IDBG.NE.1) RETURN
422: C
423: C LMAX = 200
424: C IDEL = LMAX/2
425: C
426: C IF(IOPT.EQ.2) GO TO 101
427: C IF(IOPT.EQ.3) GO TO 201
428: C ***IOPT=1*** INTEGER TYPE
429: C IF(LENG.LT.LMAX) THEN
430: C     WRITE(IPRINT,10) MEMBER,LENG
431: C     WRITE(IPRINT,11) (IA(I),I=1,LENG)
432: C     ELSE
433: C     IST = 1
434: C     IEND = IDEL
435: C     WRITE(IPRINT,12) MEMBER,IST,IEND
436: C     WRITE(IPRINT,11) (IA(I),I=IST,IEND)
437: C     IST = LENG - IDEL + 1
438: C     IEND = LENG
439: C     WRITE(IPRINT,12) MEMBER,IST,IEND
440: C     WRITE(IPRINT,11) (IA(I),I=IST,IEND)
441: C     ENDIF
442: C     GO TO 301
443: C ***IOPT=2*** REAL TYPE
444: C 101 IF(LENG.LT.LMAX) THEN
445: C     WRITE(IPRINT,20) MEMBER,LENG
446: C     WRITE(IPRINT,21) (A(I),I=1,LENG)
447: C     ELSE
448: C     IST = 1
449: C     IEND = IDEL
450: C     WRITE(IPRINT,22) MEMBER,IST,IEND
451: C     WRITE(IPRINT,21) (A(I),I=IST,IEND)
452: C     IST = LENG - IDEL + 1
453: C     IEND = LENG
454: C     WRITE(IPRINT,22) MEMBER,IST,IEND
455: C     WRITE(IPRINT,21) (A(I),I=IST,IEND)

```

```

456:                                     ENDIF
457:                                     GO TO 301
458: C ***IOPT=3*** CHARACTER TYPE
459: 201 CONTINUE
460:     WRITE(IPRINT,30) MEMBER,LENG
461:     WRITE(IPRINT,31) (IA(I),I=1,LENG)
462: C ***CHKWRT FINISHED
463: 301 CONTINUE
464:     RETURN
465: C
466: 10 FORMAT(/1H ,10X,' ***** ',
467: * /1H ,10X,' * INTEGER 1-D ARRAY PRINT * ',
468: * /1H ,10X,' ***** ',
469: * /1H ,10X,' * MEMBER NAME = ' ,A8, ' * ',
470: * /1H ,10X,' * LNEGTH = ' ,I6, ' * ',
471: * /1H ,10X,' ***** '/')
472: 11 FORMAT(1H ,5X,10I6)
473: 12 FORMAT(/1H ,10X,' ***** ',
474: * /1H ,10X,' * INTEGER 1-D ARRAY PRINT * ',
475: * /1H ,10X,' ***** ',
476: * /1H ,10X,' * MEMBER NAME = ' ,A8, ' * ',
477: * /1H ,10X,' * FROM ' ,I6, ' TO ' ,I6, ' * ',
478: * /1H ,10X,' ***** '/')
479: 20 FORMAT(/1H ,10X,' ***** ',
480: * /1H ,10X,' * REAL 1-D ARRAY PRINT * ',
481: * /1H ,10X,' ***** ',
482: * /1H ,10X,' * MEMBER NAME = ' ,A8, ' * ',
483: * /1H ,10X,' * LNEGTH = ' ,I6, ' * ',
484: * /1H ,10X,' ***** '/')
485: 21 FORMAT(1H ,5X,1P5E12.5)
486: 22 FORMAT(/1H ,10X,' ***** ',
487: * /1H ,10X,' * REAL 1-D ARRAY PRINT * ',
488: * /1H ,10X,' ***** ',
489: * /1H ,10X,' * MEMBER NAME = ' ,A8, ' * ',
490: * /1H ,10X,' * FROM ' ,I6, ' TO ' ,I6, ' * ',
491: * /1H ,10X,' ***** '/')
492: C
493: 30 FORMAT(/1H ,10X,' ***** ',
494: * /1H ,10X,' * CHARACTER 1-D ARRAY PRINT * ',
495: * /1H ,10X,' ***** ',
496: * /1H ,10X,' * MEMBER NAME = ' ,A8, ' * ',
497: * /1H ,10X,' * LNEGTH = ' ,I6, ' * ',
498: * /1H ,10X,' ***** '/')
499: 31 FORMAT(1H ,5X,17A4)
500: C
501:     END
502: C
503: C=====
504: C
505: C SUBROUTINE CLEA( A , LENG , ASET )
506: C
507: C REAL*4      A(1)
508: C INTEGER*4    IA(1)
509: C
510: C IF(LENG.LE.0) RETURN
511: C
512: C DO 100 I = 1 , LENG
513: C     A(I) = ASET
514: C 100 CONTINUE
515: C     RETURN
516: C
517: C ENTRY      ICLEA(IA ,LENG , IASET)
518: C
519: C IF(LENG.LE.0) RETURN
520: C

```

src/tools/mvpdump.f

```

521:      IF(LENG.LE.0) RETURN
522:      DO 200 I = 1 , LENG
523:        IA(I) = IASET
524:      200 CONTINUE
525:      RETURN
526:      END
527: C
528: C=====
529: C
530:      SUBROUTINE DIMCHK(SUBNAM,LMAX,LIMIT)
531: C
532: C  DEMENSION SIZE CHECK FOR VARIABLE DIMENSION METHOD
533: C
534:      COMMON /MAINIO/ NSYSI,NSYSO,NOUT,NWRK
535:      COMMON /CONTRL/ MATD,ZA,ELUNR,EHUNR,CRSMIN,MEMORY,IUNRES,INTUNR
536:      + IDUMF,IDBG
537: C
538:      CHARACTER*6 SUBNAM
539: C
540:      IF(LMAX.LE.LIMIT) RETURN
541: C
542:      MNEED = MEMORY + (LMAX-LIMIT)
543:      WRITE(NSYSO,100) SUBNAM,MEMORY,MNEED
544:      STOP 999
545: C
546: 100 FORMAT(/1H , ' ERROR STOP AT SUBROUTINE(' ,A6,') ' ,
547: * /1H , ' MEMORY OVER ' ,
548: * //1H , ' MEMORY RESERVED ----- ',I8,' WORDS ' ,
549: * /1H , ' MEMORY REQUESTED ----- ',I8,' WORDS ' //)
550: C
551:      END
552: C
553: C=====
554: C
555:      SUBROUTINE FEND
556: C
557:      INTEGER OTAPE
558: C
559:      COMMON/HEADER/C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
560:      COMMON/UNITS /OTAPE
561: C
562:      MTH = 0
563:      MFH = 0
564:      WRITE(OTAPE,10) MATH,MFH,MTH,NOSEQ
565:      NOSEQ = NXTSEQ(NOSEQ)
566: C
567: 10 FORMAT(66X,I4,I2,I3,I5)
568: C
569:      RETURN
570:      END
571: C
572: C=====
573: C
574:      SUBROUTINE LENGTH(EMESH,INTE ,PA ,NE ,NSYSO,NWRK ,IPROP )
575: C
576: C  PRINT THERMAL INELASTIC ANGULAR
577: C
578:      REAL*4 EMESH(NE) ,PA(5,NE,NE)
579:      INTEGER*4 INTE (NE)
580: C
581:      READ (NWRK) ( ( (PA(L,J,I),L=1,5 ),J=1,I ),I=1,NE )
582: C
583:      WRITE (NSYSO,101)
584:      WRITE (NSYSO,102) EMESH
585:      WRITE (NSYSO,103) INTE

```

```

586:      WRITE (NSYSO,104) (L,L=1,5)
587: C
588:      DO 100 I = 1 , NE
589:        ISW = 1
590:        IF(IPROP.LE.1) THEN
591:          ISW = 0
592:          IF(I.LE.10) ISW = 1
593:          IF(I.GT.(NE-15)) ISW = 1
594:        ENDIF
595:        IF(ISW.EQ.1) THEN
596:          DO 90 J = 1 , I
597:            SUM = 0.0
598:            DO 80 L = 1 , 5
599:              SUM = SUM + PA(L,J,I)
600:            80 CONTINUE
601:            IF(SUM.GT.0) WRITE(NSYSO,105) J,I,(PA(L,J,I),L=1,5)
602:            90 CONTINUE
603:          ENDIF
604:        100 CONTINUE
605:      RETURN
606: C
607: 101 FORMAT(1H1,5X,' ***** ',
608: * /1H ,5X,' * THERMAL INELASTIC ANGULAR TABLE * ',
609: * /1H ,5X,' ***** ' //)
610: 102 FORMAT(1H , ' ENERGY MESH (EV) : ',1P5E12.5)
611: 103 FORMAT(1H , ' INTERPOLATION CODE : ',10I6)
612: 104 FORMAT(/1H ,5X,'##### ',
613: * /1H ,5X,'## <<THERMAL INELASTIC ANGULAR TABLES>> ## ',
614: * /1H ,5X,'## CUMULATIVE PROBABILITY FOR SCATTERING ## ',
615: * /1H ,5X,'## BY ANGULAR PROCESS L GIVEN A TRANSITION## ',
616: * /1H ,5X,'## FROM ENERGY POINTS I TO J OR J TO I ; ## ',
617: * /1H ,5X,'## FIVE PROCESSES ARE ALLOWED. << PA >> ## ',
618: * /1H ,5X,'## ***** ',
619: * /1H ,5X,'## L=1: MU.GE.0.0 , ISOTROPIC ## ',
620: * /1H ,5X,'## L=2: MU.LT.0.0 , ISOTROPIC ## ',
621: * /1H ,5X,'## L=3: MU.EQ.+1.0 ## ',
622: * /1H ,5X,'## L=4: MU.GE.-1.0 ## ',
623: * /1H ,5X,'## L=5: MU.EQ.0.0 & MU>0 ## ',
624: * /1H ,5X,'##### ',
625: * //1H ,5X,' ',5(7H L = ,I3,2X) )
626: 105 FORMAT(1H ,5X,'PA(L, ',I2,1H, ',I2,') : ',1P5E12.5)
627: C
628:      END
629: C
630: C=====
631: C
632:      SUBROUTINE LGLF61(EMESH ,INTE ,LSE ,A ,IA ,
633:      + NE ,NSYSO ,MT ,PLOW ,PHI , LAST )
634: C
635: C-----LIST LF=67 KALBACH-87 FORMULATION
636: C
637: C
638:      REAL*4 EMESH(NE) ,A(1)
639:      INTEGER*4 INTE (NE) ,LSE(NE) ,IA(1)
640: C
641:      WRITE(NSYSO,101) MT
642: C
643:      DO 100 LOP = 1 , NE
644:        NEP = IA( LSE(LOP) )
645:        ND = IA( LSE(LOP) + 1 )
646:        LOC1 = LSE(LOP) + 2
647:        LOC2 = LOC1 + NEP
648:        LOC3 = LOC2 + NEP*2
649:        LOC4 = LOC3 + NEP + 1
650:        LOC5 = LOC4 + NEP + 1

```

src/tools/mvdpdump.f

```

651:          LOC6 = LOC5 + NEP + 1
652: C
653:          IF(EMESH(LOP).GT.PHI ) GO TO 100
654:          IF(EMESH(LOP).LT.PLOW) GO TO 100
655: C
656:          WRITE(NSYSO,102) LOP,INTE(LOP),LSE(LOP),EMESH(LOP),NEP-ND
657:          WRITE(NSYSO,104)
658:          WRITE(NSYSO,105)
659:          DO 50 J = 1, NEP
660:          WRITE(NSYSO,103) J,A(LOC3+J-1),A(LOC4+J-1),A(LOC1+J-1),
661:             @ IA(LOC2+2*J-2),IA(LOC2+2*J-1),
662:             @ A(LOC5+J-1),A(LOC6+J-1)
663: 50          CONTINUE
664:          WRITE(NSYSO,106) J,A(LOC3+NEP),A(LOC4+NEP),
665:             @ A(LOC5+J-1),A(LOC6+J-1)
666:          WRITE(NSYSO,105)
667: 100 CONTINUE
668:          LAST = LOC6 + NEP
669: C
670: 101 FORMAT(1H,15X,' << ENERGY DISTRIBUTION OF GAMMA RAYS >>',
671: * //1H,15X,' --- THESE DATA ARE EXPRESSED BY KALBACH-87 --- ',
672: * //1H,12X,' MT NUMBER : ',I3,' (LF=61) '//
673: 102 FORMAT(1H,12X,' NO = ',I3,' INTERPOLATION CODE =',I2,
674: * ' LOC = ',I7,
675: * //1H,12X,' INCIDENT ENERGY : ',1PE12.5,' EV',
676: * ' 6X,' & NO OF DISCRETE LEVELS =',I2)
677: 103 FORMAT(1H,5X,I3,3X,1P3E12.5,3X,2I5,0P,1P2E12.5)
678: 106 FORMAT(1H,5X,I3,3X,1P2E12.5,24X,0P,1P2E12.5)
679: 104 FORMAT(1H0,5X,'NO.',3X,' E-2ND ',',PROBABILITY ',
680: @ ' Q ',',IANS1 IANS2',
681: @ ' R-VALUE ',', A-VALUE ')
682: 105 FORMAT(1H,3X,10(10H-----))
683: C
684:          RETURN
685:          END
686: C
687: C=====
688: C
689:          SUBROUTINE LILF1B(EMESH,INTE,LSE,A,IA,
690: + NE,NSYSO,MT,NK,LF,PLOW,PHI, LAST )
691: C
692: C-----LIST LF=1 OR 6 OF ENERGY DISTRIBUTION DATA(ENERGY BINS + BMC)
693: C----- LF=5
694: C
695:          REAL*4          EMESH(NE),A(1)
696:          INTEGER*4        INTE (NE),LSE(NE),IA(1)
697: C
698:          IF(LF.EQ.1) WRITE(NSYSO,101) MT,NE
699:          IF(LF.EQ.6) WRITE(NSYSO,201) MT,NK,NE
700: C
701:          DO 100 LOP = 1, NE
702:             NEP = IA( LSE(LOP) )
703:             LOC1 = LSE(LOP) + 1
704:             LOC2 = LOC1 + NEP
705:             LOC3 = LOC2 + NEP*2
706:             LOC4 = LOC3 + NEP + 1
707:             IF(LF.EQ.1.AND.EMESH(LOP).GT.PHI ) GO TO 100
708:             IF(LF.EQ.1.AND.EMESH(LOP).LT.PLOW) GO TO 100
709: C
710:             IF(LF.NE.5)
711:             @ WRITE(NSYSO,102) LOP,INTE(LOP),LSE(LOP),EMESH(LOP)
712:             WRITE(NSYSO,104)
713:             WRITE(NSYSO,105)
714:             DO 50 J = 1, NEP
715:             WRITE(NSYSO,103) J,A(LOC3+J-1),A(LOC4+J-1),A(LOC1+J-1),

```

```

716:             @ IA(LOC2+2*J-2),IA(LOC2+2*J-1)
717: 50          CONTINUE
718:             WRITE(NSYSO,103) J,A(LOC3+NEP),A(LOC4+NEP)
719:             WRITE(NSYSO,105)
720: 100 CONTINUE
721:             LAST = LOC4 + NEP
722: C
723: 101 FORMAT(1H1,10X,' << ENERGY DISTRIBUTION OF SECONDARY NEUTRON >>',
724: * //1H,12X,' MT NUMBER : ',I3,' (LF=1) ',
725: * ' & NE : ',I3//)
726: 201 FORMAT(1H1,10X,' << ENERGY DISTRIBUTION OF GAMMA RAY >>',
727: * //1H,12X,' MT NUMBER : ',I3,' (LF=6) ',
728: * ' SUB-SECTION NO : ',I3,' & NE : ',I3//)
729: 102 FORMAT(1H,12X,' NO = ',I3,' INTERPOLATION CODE =',I2,
730: * ' LOC = ',I7,
731: * //1H,12X,' INCIDENT ENERGY : ',1PE12.5,' EV')
732: 103 FORMAT(1H,5X,I3,3X,1P3E12.5,2X,2I5)
733: 104 FORMAT(1H0,5X,'NO.',3X,' E-2ND ',',PROBABILITY ',
734: @ ' Q ',',IANS1 IANS2')
735: 105 FORMAT(1H,3X,7(10H-----))
736: C
737:          RETURN
738:          END
739: C
740: C=====
741: C
742:          SUBROUTINE LILF61(EMESH,INTE,LSE,A,IA,
743: + NE,NSYSO,MT,PLOW,PHI, LAST )
744: C
745: C-----LIST LF=67 KALBACH-87 FORMULATION
746: C
747: C
748:          REAL*4          EMESH(NE),A(1)
749:          INTEGER*4        INTE (NE),LSE(NE),IA(1)
750: C
751:          WRITE(NSYSO,101) MT
752: C
753:          DO 100 LOP = 1, NE
754:             NEP = IA( LSE(LOP) )
755:             ND = IA( LSE(LOP) + 1 )
756:             LOC1 = LSE(LOP) + 2
757:             LOC2 = LOC1 + NEP
758:             LOC3 = LOC2 + NEP*2
759:             LOC4 = LOC3 + NEP + 1
760:             LOC5 = LOC4 + NEP + 1
761:             LOC6 = LOC5 + NEP + 1
762: C
763:             IF(EMESH(LOP).GT.PHI ) GO TO 100
764:             IF(EMESH(LOP).LT.PLOW) GO TO 100
765: C
766:             WRITE(NSYSO,102) LOP,INTE(LOP),LSE(LOP),EMESH(LOP),NEP-ND
767:             WRITE(NSYSO,104)
768:             WRITE(NSYSO,105)
769:             DO 50 J = 1, NEP
770:             WRITE(NSYSO,103) J,A(LOC3+J-1),A(LOC4+J-1),A(LOC1+J-1),
771:             @ IA(LOC2+2*J-2),IA(LOC2+2*J-1),
772:             @ A(LOC5+J-1),A(LOC6+J-1)
773: 50          CONTINUE
774:             WRITE(NSYSO,106) J,A(LOC3+NEP),A(LOC4+NEP),
775:             @ A(LOC5+J-1),A(LOC6+J-1)
776:             WRITE(NSYSO,105)
777: 100 CONTINUE
778:             LAST = LOC6 + NEP
779: C
780: 101 FORMAT(1H1,15X,' << ENERGY DISTRIBUTION OF SECONDARY NEUTRON >>',

```

src/tools/mvpdump.f

```

781:      *      /1H ,15X,' --- THESE DATA ARE EXPESSED BY KALBACH-87 --- ',
782:      *      //1H ,12X,'          MT      NUMBER      : ',I3,' (LF=61) '//'
783: 102 FORMAT(1H ,12X,'          NO = ',I3,'      ITERPOLATION CODE =',I2,
784:      *      '      LOC = ',I7,
785:      *      /1H ,12X,'          INCIDENT ENERGY : ',1PE12.5,' EV',
786:      *      6X,' & NO OF DISCRETE LEVELS = ',I2)
787: 103 FORMAT(1H ,5X,I3,3X,1P3E12.5,2X,2I5,0P,1P2E12.5)
788: 106 FORMAT(1H ,5X,I3,3X,1P2E12.5,24X ,0P,1P2E12.5)
789: 104 FORMAT(1H0,5X,'NO.',3X,' E-2ND      ','PROBABILITY ',
790:      @      '      Q      ','IANS1 IANS2',
791:      @      '      R-VALUE      ','A-VALUE      ')
792: 105 FORMAT(1H ,3X,10(10H-----))
793: C
794: RETURN
795: END
796: C
797: C=====
798: C
799: SUBROUTINE LILF66(EMESH ,INT ,APSX,NE ,NSYSO ,MT ,LF ,NPSX)
800: C
801: C----- PRINT LF=66 N-BODY PHASE-SPACE DISTRIBUTION DATA
802: C
803: REAL*4      EMESH(NE),APSX(NE)
804: INTEGER*4    INT (NE)
805: C
806: WRITE(NSYSO,101) MT,LF,NE,NPSX
807: WRITE(NSYSO,102)
808: WRITE(NSYSO,103)
809: DO 100 I = 1 , NE
810: WRITE(NSYSO,104) I,INT(I),EMESH(I),APSX(I)
811: 100 CONTINUE
812: WRITE(NSYSO,103)
813: C
814: 101 FORMAT(1H1,20X,' *****
815:      *      /1H ,20X,' *      LF=66 (FILE-6 LAW=6) DATA      * ',
816:      *      /1H ,20X,' *****
817:      *      /1H ,20X,' * N-BODY PHASE-SPACE DISTTIBUTION * ',
818:      *      /1H ,20X,' *****
819:      *      //1H ,10X,' MT = ',I3,'      LF = ',I3,'      NE = ',I3,'
820:      *      '      NO OF BODIES IN TH PHASE SPACE = ',I3//)
821: 102 FORMAT(1H ,5X,' NO INT ENERGY(EV)  APSX(TOTAL MASS RATIO) ')
822: 103 FORMAT(1H ,3X,'-----)
823: 104 FORMAT(1H ,5X,2I3,1P3E12.5)
824: C
825: RETURN
826: END
827: C
828: C=====
829: C
830: SUBROUTINE LILF67(EMESH ,INTE ,LSE ,A ,IA ,IDUMP,NE ,
831:      +      NBINA1,NSYSO ,MT ,PLOW ,PHI ,LAST )
832: C
833: C-----LIST LF=67 DDX DATA (LF=7 FORMULATION)
834: C
835: C
836: PARAMETER ( MAXMU = 300 )
837: C
838: REAL*4      EMESH(NE),A(1)
839: INTEGER*4    INTE (NE),LSE(NE),IA(1)
840: REAL*4      WORK (MAXMU) , XMU (MAXMU )
841: INTEGER*4    LST2A(MAXMU)
842: C
843: WRITE(NSYSO,101) MT
844: LIMIT      = NE-1
845: C

```

```

846: DO 100 LOP = 1 , NE
847: LOC1 = LSE(LOP)
848: INTMU= IA(LOC1+NBINA1)
849: KSW = LOC1 + NBINA1 + 1
850: NMU = IA(KSW)
851: CM WRITE(NSYSO,*) ' LOP LOC1 INTMU NMU : ',LOP,LOC1,INTMU,NMU
852: IF(NMU.GT.MAXMU) WRITE(NSYSO,*) ' NMU > 300 AT SUBR(LILF67) '
853: IF(NMU.GT.MAXMU) STOP ' NMU > 300 AT SUBR(LILF67) '
854: C
855: CALL CLEA ( XMU , MAXMU , 0.0 )
856: CALL ICLEA ( LST2A , MAXMU , 0 )
857: ISW = KSW + NMU
858: DO 10 J = 1 , NMU
859: XMU (J) = A(KSW + J )
860: LST2A(J) = IA(ISW + J )
861: 10 CONTINUE
862: C
863: IF(EMESH(LOP).GT.PHI.OR.EMESH(LOP).LT.PLOW) GO TO 90
864: IF(IDUMP.LE.1.AND.LOP.GT.2.AND.LOP.LT.LIMIT) GO TO 90
865: C
866: WRITE(NSYSO,102) LOP,INTE(LOP),LSE(LOP),EMESH(LOP),NMU,INTMU
867: WRITE(NSYSO,103) (XMU (J),J=1,NMU)
868: WRITE(NSYSO,104) (LST2A(J),J=1,NMU)
869: CALL LISTAN ( EMESH(LOP) , A(LOC1) , WORK , NBINA1 , NSYSO )
870: C
871: C *** PRINT SECONDARY NEUTRON ENERGY DISTRIBUTION FOR EACH ANGLE
872: C
873: DO 50 JOP = 1 , NMU
874: WRITE(NSYSO,105) JOP,XMU(JOP),EMESH(LOP),MT
875: C
876: NEP = IA(LST2A(JOP))
877: KOC1 = LST2A(JOP) + 1
878: KOC2 = KOC1 + NEP
879: KOC3 = KOC2 + NEP*2
880: KOC4 = KOC3 + NEP + 1
881: CALL LILF67A( NEP ,A(KOC1) ,IA(KOC2) ,A(KOC3) ,A(KOC4) ,NSYSO )
882: 50 CONTINUE
883: C
884: 90 CONTINUE
885: NEP = IA(LST2A(NMU) )
886: LAST = LST2A(NMU) + NEP*3 + 2*(NEP+1) - 1
887: C
888: 100 CONTINUE
889: C
890: 101 FORMAT(1H1,10X,' << NEUTRON DOUBLE DIFFERENTIAL X-SECTION DATA >>',
891:      *      //1H ,12X,'          MT      NUMBER      : ',I3,' (LF=67) '//'
892: 102 FORMAT(1H ,12X,'          NO = ',I3,'      ITERPOLATION CODE =',I2,
893:      *      '      LOC = ',I7,
894:      *      /1H ,12X,'          INCIDENT ENERGY : ',1PE12.5,' EV',
895:      *      /1H ,12X,' NO OF ANGULAR MESH = ',I3,' & INTERPOLATION CODE =',I2)
896: 103 FORMAT(1H ,' COSINE OF SCATTERING ANGLE OF SECONDARY NEUTRON : '
897:      *      ,/(1H ,5X,10F9.5,:))
898: 104 FORMAT(1H ,' START POSITION OF SECONDARY NEUTRON FOR EACH ANGLE : '
899:      *      ,/(1H ,5X,10I9,:))
900: C
901: 105 FORMAT(/1H ,10X,' << SECONDARY ENERGY DISTRIBUTION OF ',I2,
902:      *      '-TH ANGLE (' ,F7.4,' ) >> ',
903:      *      /1H ,10X,' INCIDENT NEUTRON ENERGY IS ',E12.5,
904:      *      ' FOR MT = ',I3,' REACTION '//)
905: C
906: RETURN
907: END
908: C
909: C=====
910: C

```

src/tools/mvpdump.f

```

911:      SUBROUTINE LISAEY(EMESH ,INTE ,TYIELD,YIELD ,MT ,
912:      @      NE      ,NK      ,NSYSO )
913: C
914: C-----PRINT DDX YIELD DATA
915: C
916:      REAL*4      EMESH(NE) ,YIELD(NE,NK) ,TYIELD(NE)
917:      INTEGER*4    INTE (NE)
918: C
919: C
920: C
921: C
922:      WRITE(NSYSO,1006) MT,NK
923:      WRITE(NSYSO,1009)
924:      WRITE(NSYSO,1007)
925: C
926:      DO 1000 I = 1 , NE
927:      WRITE(NSYSO,1008) J,EMESH(J) ,INTE(J) ,TYIELD(J) ,(YIELD(J,K),K=1,NK)
928: 1000 CONTINUE
929:      WRITE(NSYSO,1007)
930: C
931:      RETURN
932: C
933: 1006 FORMAT(///1H ,10X,'*****
934: *      /1H ,10X,'*      YIELD DATA FOR ANGULAR-ENERGY DATA *',
935: *      /1H ,10X,'*****
936: *      /1H ,10X,'*      MT = ',I3,' *',
937: *      /1H ,10X,'*****
938: *      /1H ,15X,' NO OF SUBSECTION = ',I3/)
939: 1009 FORMAT(/1H ,5X,' NO ENERGY(EV) INT TOTAL EACH YIELD ')
940: 1007 FORMAT(1H ,5X,'-----',8(8H-----) )
941: 1008 FORMAT(1H ,5X,I3,1PE12.5,2X,0P,I3,8F8.5,: ,/
942: *      (1H ,33X,7F8.5,:))
943: C
944:      END
945: C
946: C-----
947: C
948:      SUBROUTINE LISLFI(EMESH ,INTE ,EBIN ,PBIN ,NE ,NBIN ,
949:      +      NWRK ,NSYSO ,MT ,PLOW ,PHI )
950: C
951: C-----LIST LF=1 OF ENERGY DISTRIBUTION DATA
952: C
953:      REAL*4      EMESH(NE) ,EBIN(NBIN,NE) ,PBIN(NBIN,NE)
954:      INTEGER*4    INTE (NE)
955: C
956:      READ(NWRK) ( ( EBIN(J,I) ,J=1,NBIN) ,(PBIN(J,I) ,J=1,NBIN) ,I=1,NE)
957: C
958:      WRITE(NSYSO,101) MT
959:      DEL      = 1.0 /FLOAT(NBIN-1)
960: C
961:      DO 100 LOP = 1 , NE
962:      IF(EMESH(LOP).GT.PHI ) GO TO 100
963:      IF(EMESH(LOP).LT.PLOW) GO TO 100
964: C
965:      WRITE(NSYSO,102) LOP,INTE(LOP) ,EMESH(LOP)
966:      SUM      = 0.0
967:      DO 50 J = 1,NBIN
968:      WRITE(NSYSO,103) J,EBIN(J,LOP) ,SUM,PBIN(J,LOP)
969:      SUM      = SUM + DEL
970: 50 CONTINUE
971: 100 CONTINUE
972: C
973: 101 FORMAT(1H1,10X,' << ENERGY DISTRIBUTION OF SECONDARY NEUTRON >>',
974: *      /1H ,12X,' MT      NUMBER      : ',I3,' (LF=1) '///)
975: 102 FORMAT(1H ,12X,' NO      = ',I3,' ITERPOLATION CODE =',I2,

```

```

976: *      /1H ,12X,'      INCIDENT ENERGY : ',1PE12.5,' EV',
977: *      /1H ,12X,'NO      ENERGY(EV) CUMU. PROB. PROBILITY(X)')
978: 103 FORMAT(1H ,11X,I3,3X,1P3E12.5)
979: C
980:      RETURN
981:      END
982: C
983: C=====
984: C
985:      SUBROUTINE LISLFI(EBIN ,PBIN ,NBIN ,NSYSO ,MT )
986: C
987: C-----LIST LF=1 OF ENERGY DISTRIBUTION DATA
988: C
989:      REAL*4      EBIN(NBIN) ,PBIN(NBIN)
990: C
991:      WRITE(NSYSO,101) MT
992:      DEL      = 1.0 /FLOAT(NBIN-1)
993: C
994:      SUM      = 0.0
995:      DO 50 J = 1,NBIN
996:      WRITE(NSYSO,102) J,EBIN(J) ,SUM,PBIN(J)
997:      SUM      = SUM + DEL
998: 50 CONTINUE
999: C
1000: 101 FORMAT(1H1,10X,' << ENERGY DISTRIBUTION OF SECONDARY NEUTRON >>',
1001: *      /1H ,12X,' MT      NUMBER      : ',I3,' (LF=5 ) ',
1002: *      /1H ,12X,'NO      ENERGY(EV) CUMU. PROB. PROBILITY(X)')
1003: 102 FORMAT(1H ,11X,I3,3X,1P3E12.5)
1004: C
1005:      RETURN
1006:      END
1007: C
1008: C=====
1009: C
1010:      SUBROUTINE LISLGI(EMESH ,INTE ,EBIN ,PBIN ,NE ,NBIN ,
1011:      +      NWRK ,NSYSO ,MT ,NK )
1012: C
1013: C-----LIST LF=1 OF ENERGY DISTRIBUTION DATA
1014: C
1015:      REAL*4      EMESH(NE) ,EBIN(NBIN,NE) ,PBIN(NBIN,NE)
1016:      INTEGER*4    INTE (NE)
1017: C
1018:      READ(NWRK) ( ( EBIN(J,I) ,J=1,NBIN) ,(PBIN(J,I) ,J=1,NBIN) ,I=1,NE)
1019: C
1020:      WRITE(NSYSO,101) MT,NK,NE
1021:      DEL      = 1.0 /FLOAT(NBIN-1)
1022: C
1023:      DO 100 LOP = 1 , NE
1024:      WRITE(NSYSO,102) LOP,INTE(LOP) ,EMESH(LOP)
1025:      SUM      = 0.0
1026:      DO 50 J = 1,NBIN
1027:      WRITE(NSYSO,103) J,EBIN(J,LOP) ,SUM,PBIN(J,LOP)
1028:      SUM      = SUM + DEL
1029: 50 CONTINUE
1030: 100 CONTINUE
1031: C
1032: 101 FORMAT(1H1,10X,' << ENERGY DISTRIBUTION OF GAMMA RAY >>',
1033: *      /1H ,12X,' MT      NUMBER      : ',I3,' (LF=6) ',
1034: *      ' SUB-SECTION NO : ',I3,' & NE : ',I3/)
1035: 102 FORMAT(1H ,12X,' NO      = ',I3,' ITERPOLATION CODE =',I2,
1036: *      /1H ,12X,'      INCIDENT ENERGY : ',1PE12.5,' EV',
1037: *      /1H ,12X,'NO      ENERGY(EV) CUMU. PROB. PROBILITY(X)')
1038: 103 FORMAT(1H ,11X,I3,3X,1P3E12.5)
1039: C
1040:      RETURN

```


src/tools/mvdpdump.f

```

1041:      END
1042: C
1043: C=====
1044: C
1045:      SUBROUTINE LISTAN( EANG , AMUDIS , ANGDIS , NPANG , NSYSO )
1046: C
1047: C-----ANGULAR DISTRIBUTION PRINT OUT
1048: C
1049:      DIMENSION AMUDIS(NPANG),ANGDIS(NPANG)
1050: C
1051:      WRITE(NSYSO,101) EANG
1052:      WRITE(NSYSO,103)
1053: C
1054:      NDIV = NPANG - 1
1055:      DEL = 1.0 / FLOAT(NDIV)
1056:      ANGDIS(1) = 0.0
1057:      DO 10 I = 2 , NPANG
1058:      ANGDIS(I) = ANGDIS(I-1) + DEL
1059: 10 CONTINUE
1060:      ANGDIS(NPANG) = 1.00000
1061: C
1062:      WRITE(NSYSO,102) AMUDIS(1),ANGDIS(1)
1063: C
1064:      LOOP = NPANG/2
1065:      DO 100 J = 1,LOOP
1066:      LOP1 = J + 1
1067:      LOP2 = LOOP + LOP1
1068:      IF(LOP2.LE.NPANG)
1069:      *WRITE(NSYSO,102)
1070:      * AMUDIS(LOP1),ANGDIS(LOP1),AMUDIS(LOP2),ANGDIS(LOP2)
1071:      IF(LOP2.GT.NPANG)
1072:      *WRITE(NSYSO,102) AMUDIS(LOP1),ANGDIS(LOP1)
1073: 100 CONTINUE
1074:      WRITE(NSYSO,103)
1075: C
1076:      RETURN
1077: C
1078: 101 FORMAT(/1H ,15X,' ## ANGULAR DISTRIBUTION DATA ## ',
1079:      *//1H ,10X,' INCIDENT NEUTRON ENERGY : ',1PE12.5,' EV',/,
1080:      */1H ,4X,' COSINE CUMULATIVE ANGULAR | COSINE CUMULATIVE ANGULAR',
1081:      */1H ,4X,' (THETA) DISTRIBUTION | (THETA) DISTRIBUTION ')
1082: CM102 FORMAT(1H ,5X,F5.2,F12.7,10X,2H| ,F5.2,F12.7)
1083: 102 FORMAT(1H ,5X,F10.6,2X,F8.5, 7X,2H| ,F10.6,2X,F8.5)
1084: 103 FORMAT(1H ,4X,
1085:      * '-----+-----')
1086: C
1087:      END
1088: C
1089: C=====
1090: C
1091:      SUBROUTINE LISTBC(EMESH ,INTE ,Q ,IEPOS ,MT ,
1092:      @ NE ,NBIN ,NSYSO ,NWRK ,ITYPE )
1093: C
1094: C-----PRINT BMC SAMPLING DATA
1095: C
1096: CMOD PARAMETER ( MTMAX = 120 , NKMAX = 10 )
1097: PARAMETER ( MTMAX = 120 , NKMAX = 70 )
1098: COMMON /HEAD2 / MTINFO(MTMAX),MTPAR(MTMAX),ISTMT(MTMAX),
1099: + IENDMT(MTMAX),NEUMT(MTMAX),LCTMT(MTMAX),
1100: + NEANG(MTMAX),NKF5(MTMAX),NEF5(MTMAX),
1101: + MTTHR(MTMAX),LSTF3(MTMAX),LSTF4(MTMAX),
1102: + LSTF5(MTMAX),
1103: + NEF5S(NKMAX,MTMAX),LFF5S(NKMAX,MTMAX),
1104: + LSTF5S(NKMAX,MTMAX),QVAL(MTMAX),
1105: + QVAL2(MTMAX),IORDMT(MTMAX)

```

```

1106: C
1107:      REAL*4 EMESH(NE),Q(NBIN,NE)
1108:      INTEGER*4 INTE (NE),IEPOS(2,NBIN,NE)
1109: C
1110: C
1111: C
1112:      READ(NWRK) ( ( Q(J,I) ,J=1,NBIN ) ,
1113:      + ( IEPOS(1,J,I) ,IEPOS(2,J,I) ,J=1,NBIN ) ,
1114:      + I=1,NE )
1115: C
1116:      IF(ITYPE.GT.0) GO TO 2001
1117: C
1118:      WRITE(NSYSO,1001) MT
1119: C
1120:      DO 1000 J = 1 , NE
1121:      WRITE(NSYSO,1003) J,EMESH(J),INTE(J)
1122:      WRITE(NSYSO,1004)
1123:      DO 1500 I = 1 , NBIN
1124:      WRITE(NSYSO,1005) I,LFF5S(I,MT),Q(I,J),(IEPOS(K,I,J),K=1,2)
1125: 1500 CONTINUE
1126:      WRITE(NSYSO,1004)
1127: 1000 CONTINUE
1128: C
1129:      RETURN
1130: C
1131: 1001 FORMAT(1H1,10X,'*****',
1132:      * /1H ,10X,'* BMC SAMPLING DATA FOR SUB-SECTION * ',
1133:      * /1H ,10X,'* SELECTION OF ENERGY DISTRIBUTION * ',
1134:      * /1H ,10X,'*****',
1135:      * /1H ,10X,'* MT = ',I3,' * ',
1136:      * /1H ,10X,'*****')
1137: 1003 FORMAT(/1H ,5X,
1138:      *' LOP = ',I3,5X,'INCIDENT ENERGY= ',1PE12.5,' EV',5X,' INT =',I5,
1139:      * //1H ,5X,'NO LF Q-ANSWER IANS1 IANS2')
1140: 1004 FORMAT(1H ,3X,4(10H-----))
1141: 1005 FORMAT(1H ,4X,2I3,1PE12.5,2X,2I6)
1142: C
1143: C
1144: C
1145: 2001 CONTINUE
1146:      WRITE(NSYSO,2002) MT
1147: C
1148:      DO 2000 J = 1 , NE
1149:      WRITE(NSYSO,2003) J,EMESH(J),INTE(J)
1150:      WRITE(NSYSO,2004)
1151:      DO 2500 I = 1 , NBIN
1152:      WRITE(NSYSO,2005) I,EMESH(I),Q(I,J),(IEPOS(K,I,J),K=1,2)
1153: 2500 CONTINUE
1154:      WRITE(NSYSO,2004)
1155: 2000 CONTINUE
1156: C
1157:      RETURN
1158: C
1159: 2002 FORMAT(1H1,10X,'*****',
1160:      * /1H ,10X,'* BMC SAMPLING DATA FOR SCODARY * ',
1161:      * /1H ,10X,'*ENERGY DISTRIBUTION OF THERMAL INELASTIC* ',
1162:      * /1H ,10X,'*****',
1163:      * /1H ,10X,'* MT = ',I3,' * ',
1164:      * /1H ,10X,'*****')
1165: 2003 FORMAT(/1H ,5X,
1166:      *' LOP = ',I3,5X,'INCIDENT ENERGY= ',1PE12.5,' EV',5X,' INT =',I5,
1167:      * //1H ,5X,'NO ENERGY(EV) Q-ANSWER IANS1 IANS2')
1168: 2004 FORMAT(1H ,3X,5(10H-----))
1169: 2005 FORMAT(1H ,4X,I3,1P2E12.5,2X,2I6)
1170: C

```

src/tools/mvpdump.f

```

1171:      END
1172: C
1173: C=====
1174: C
1175:      SUBROUTINE LISTGC(EMESH ,INTE ,Q ,IEPOS ,MT ,
1176:      @      NE ,NBIN ,NSYSO ,NWRK ,ESK , IDUMP )
1177: C
1178: C-----PRINT BMC SAMPLING DATA
1179: C
1180:      REAL*4      EMESH(NE),Q(NBIN,NE),ESK(NBIN)
1181:      INTEGER*4    INTE (NE),IEPOS(2,NBIN,NE)
1182: C
1183: C
1184: C
1185:      CMOD READ(NWRK) ( ( Q(J,I) , J=1,NBIN ) ,
1186:      CMOD +      ( IEPOS(1,J,I) ,IEPOS(2,J,I) , J=1,NBIN ) ,
1187:      CMOD +      I=1,NE )
1188: C
1189:      READ(NWRK) ( ( Q(J,I) , I = 1 , NE ) , J=1,NBIN )
1190: C
1191:      CM WRITE(NSYSO,1001) MT
1192:      CM WRITE(NSYSO,1006) MT,(ESK(I),I=1,NBIN)
1193:      CM WRITE(NSYSO,1009)
1194:      CM WRITE(NSYSO,1007)
1195: C
1196:      NE1 = NE - 1
1197:      DO 1000 J = 1 , NE
1198:      CM IF(IDUMP.EQ.1.AND.J.GT.6.AND.J.LT.NE1) GO TO 1000
1199: C
1200:      CM WRITE(NSYSO,1003) J,EMESH(J),INTE(J)
1201:      CM WRITE(NSYSO,1004)
1202:      CM DO 1500 I = 1 , NBIN
1203:      CM WRITE(NSYSO,1005) I,Q(I,J),(IEPOS(K,I,J),K=1,2),
1204:      CM @      ESK(I)
1205:      CM1500 CONTINUE
1206:      CM WRITE(NSYSO,1004)
1207:      CM WRITE(NSYSO,1008) J,EMESH(J),INTE(J),(Q(I,J),I=1,NBIN)
1208:      CM1000 CONTINUE
1209:      CM WRITE(NSYSO,1007)
1210: C
1211:      RETURN
1212: C
1213:      1001 FORMAT(///1H ,10X,'*****
1214:      *      /1H ,10X,' BMC SAMPLING DATA FOR SUB-SECTION
1215:      *      /1H ,10X,' SELECTION OF GAMMA YEILD DATA
1216:      *      /1H ,10X,'*****
1217:      *      /1H ,10X,' MT = ' ,I3,'
1218:      *      /1H ,10X,'*****
1219:      1003 FORMAT(//1H ,5X,
1220:      *' LOP = ' ,I3,5X,' INCIDENT ENERGY= ' ,1P2E12.5,' EV',5X,' INT =',I5,
1221:      *' //1H ,5X,' NO Q-ANSWER IANS1 IANS2 ESK(EV)')
1222:      1004 FORMAT(1H ,3X,5(10H-----))
1223:      1005 FORMAT(1H ,4X, I6,1P2E12.5,2X,2I6,0P,4X,1P2E12.5)
1224: C
1225:      1006 FORMAT(///1H ,10X,'*****
1226:      *      /1H ,10X,' CUMULATIVE GAMMA YIELD DATA * ' ,
1227:      *      /1H ,10X,'*****
1228:      *      /1H ,10X,' MT = ' ,I3,' * ' ,
1229:      *      /1H ,10X,'*****
1230:      *      /1H ,5X,'ESK(EV) : ' ,1P8E11.4,:/,
1231:      *      (1H ,5X,' : ' ,1P8E11.4,:))
1232:      1009 FORMAT(1H ,5X,' NO ENERGY(EV) INT CUMULATIVE YIELD ')
1233:      1007 FORMAT(1H ,5X,'-----',8(8H-----) )
1234:      1008 FORMAT(1H ,5X,I3,1P2E12.5,2X,0P,I3,8F8.5,:/,
1235:      *      (1H ,25X,8F8.5,:))

```

```

1236: C
1237:      END
1238: C
1239: C=====
1240: C
1241:      SUBROUTINE LISTGX( XX , YY , NN , ELOW , EHI , NSYSO )
1242: C
1243: C-----X-SECTION PRINT-OUT ROUTINE
1244: C
1245:      DIMENSION XX(1) , YY(1)
1246: C
1247:      IF(XX( 1).LT.ELOW) RETURN
1248:      IF(XX(NN).GT.EHI ) RETURN
1249: C
1250:      IST = 1
1251:      IEND = NN
1252: C
1253:      DO 100 J = 1 , NN-1
1254:      E1 = XX(J )
1255:      E2 = XX(J+1)
1256:      IF(EHI.GT.E2.AND.EHI .LE.E1) IST = J
1257:      100 CONTINUE
1258:      IF(EHI.EQ.XX(NN)) IST = NN
1259: C
1260:      DO 200 J = 1 , NN-1
1261:      E1 = XX(J )
1262:      E2 = XX(J+1)
1263:      IF(ELOW.GE.E2.AND.ELOW.LT.E1) IEND = J+1
1264:      200 CONTINUE
1265:      IF(ELOW.EQ.XX(1)) IEND = 1
1266: C
1267:      NP = IEND - IST + 1
1268: C
1269:      CM WRITE(NSYSO,10) IST,IEND,NP
1270:      CM WRITE(NSYSO,11) EHI,ELOW
1271:      CM WRITE(NSYSO,12) XX(IST),XX(IEND)
1272:      CM10 FORMAT(1H ,' ## IST IEND NP ## ',3I6)
1273:      CM11 FORMAT(1H ,' ## EHI ELOW ## ',1P2E12.5)
1274:      CM12 FORMAT(1H ,' ##X(IST) X(IEND)# ',1P2E12.5)
1275: C
1276:      CALL LISTX2( XX(IST) , YY(IST) , NP , IST , NSYSO )
1277: C
1278:      RETURN
1279:      END
1280: C
1281: C=====
1282: C
1283:      SUBROUTINE LISTHS( NSYSO ,NTTHSC ,NPTHE ,NRETHE ,NETHE ,
1284:      *      NBNTHE ,NPPTHIN ,NIGTH ,NFGTH ,MXNGTH ,
1285:      *      NPFCUT ,ENERGY ,NPTS ,IOPT ,
1286:      *      TMPTHS ,EMESH3 ,CRSTH3 ,NBTH4 ,INTTH4 ,
1287:      *      ANGTH4 ,EMESH7 ,CRSTH7 ,EMESH5 ,ETRAN7 ,
1288:      *      ANGTH7 ,CRSMT2 )
1289: C
1290: C *** PRINT TEMP. DEPENDENT THERMAL SCATTERING DATA
1291: C
1292:      REAL*4      ENERGY(NPTS)
1293: C
1294:      REAL*4      TMPTHS(NTTHSC)
1295:      REAL*4      EMESH3(NPTHE)
1296:      REAL*4      CRSTH3(NTTHSC,NPTHE)
1297:      INTEGER*4    NBTH4(NRETHE)
1298:      INTEGER*4    INTTH4(NRETHE)
1299:      REAL*4      ANGTH4(NTTHSC,NBNTHE,NETHE)
1300:      REAL*4      EMESH7(NPTHIN)

```

src/tools/mvpdump.f

```

1301:      REAL*4      CRSTH7(NTTHSC,NPTHIN)
1302:      REAL*4      EMESH5(NFGTH)
1303:      REAL*4      ETRAN7(NTTHSC,NFGTH,NIGTH)
1304:      REAL*4      ANGTH7(NTTHSC,5,MXNGTH)
1305:      REAL*4      CRSMT2(NPTCUT)
1306: C
1307: C *** START OF PROCESS
1308: C
1309: C
1310: C *** PRINT THERMAL ELASTIC CROSS SECTION
1311: C
1312:      IF(NPTHE.GT.0) THEN
1313: C
1314:      WRITE(NSYSO,101)
1315:      WRITE(NSYSO,102)                (TMPHS(K),K=1,NTTHSC)
1316:      WRITE(NSYSO,103)
1317:      DO 110 I = 1 , NPTHE
1318:      WRITE(NSYSO,104) I,EMESH5(I),(CRSTH3(K,I),K=1,NTTHSC)
1319: 110 CONTINUE
1320:      WRITE(NSYSO,103)
1321: C
1322: 101 FORMAT(1H1,10X,'*****
1323:      *      /1H ,10X,'*      THERMAL ELASTIC CROSS SECTION      * ',
1324:      *      /1H ,10X,'*****
1325: 102 FORMAT(1H ,5X,'NO ENERGY(EV) ',12(F7.1,2H K,:))
1326: 103 FORMAT(1H ,1X,13(10H-----))
1327: 104 FORMAT(1H ,3X,I4,F9.6,2X,12F9.4)
1328: C
1329:      ENDIF
1330: C
1331: C *** PRINT THERMAL ELASTIC ANGULAR DATA
1332: C
1333:      IF(NETHE.GT.0) THEN
1334: C
1335:      WRITE(NSYSO,201)
1336:      WRITE(NSYSO,205) (NBTH4(I),I=1,NRETHE)
1337:      WRITE(NSYSO,206) (INTTH4(I),I=1,NRETHE)
1338: C
1339:      DO 220 J = 1 , NETHE
1340: CDEL IF(J.GT.10.AND.J.LT.(NETHE-10)) GO TO 220
1341: C
1342:      WRITE(NSYSO,202) J,ANGTH4(1,1,J),(TMPHS(K),K=1,NTTHSC)
1343:      WRITE(NSYSO,203)
1344:      PROB      = 0.0
1345:      DEL      = 1.00 / FLOAT(NBNTHE)
1346:      DO 210 I = 2 , NBNTHE
1347:      PROB      = PROB + DEL
1348:      WRITE(NSYSO,204) I,PROB,(ANGTH4(K,I,J),K=1,NTTHSC)
1349: 210 CONTINUE
1350:      WRITE(NSYSO,203)
1351: 220 CONTINUE
1352: C
1353:      ENDIF
1354: C
1355: 201 FORMAT(1H1,10X,'*****
1356:      *      /1H ,10X,'*      THERMAL ELASTIC ANGULAR DATA LISTING * ',
1357:      *      /1H ,10X,'*****
1358: 202 FORMAT(/1H ,10X,'ENERGY NO = ',I3,5X,
1359:      *      'INCIDENT ENERGY= ',F10.6,' EV'
1360:      *      //1H , 5X,'NO CUM. PROB ',12(F7.1,2H K,:))
1361: 203 FORMAT(1H ,1X,13(10H-----))
1362: 204 FORMAT(1H ,3X,I4,F9.6,2X,12F9.6)
1363: 205 FORMAT(1H ,5X,'NBT :',20I6)
1364: 206 FORMAT(1H ,5X,'INT :',20I6)
1365: C

1366: C *** PRINT THERMAL INELASTIC X-SECTION
1367: C
1368:      IF(NPTHIN.GT.0) THEN
1369: C
1370:      WRITE(NSYSO,301)
1371:      WRITE(NSYSO,302)                (TMPHS(K),K=1,NTTHSC)
1372:      WRITE(NSYSO,303)
1373:      DO 310 I = 1 , NPTHIN
1374:      WRITE(NSYSO,304) I,EMESH7(I),(CRSTH7(K,I),K=1,NTTHSC)
1375: 310 CONTINUE
1376:      WRITE(NSYSO,303)
1377: C
1378:      ENDIF
1379: C
1380: 301 FORMAT(1H1,30X,'*****
1381:      *      /1H ,30X,'*      THERMAL INELASTIC CROSS SECTION      * ',
1382:      *      /1H ,30X,'*****
1383: 302 FORMAT(1H ,5X,'NO ENERGY(EV) ',12(F7.1,2H K,:))
1384: 303 FORMAT(1H ,1X,13(10H-----))
1385: 304 FORMAT(1H ,3X,I4,F9.6,2X,12F9.4)
1386: C
1387: C *** PRINT THERMAL INELASTIC MATRIX DATA
1388: C
1389:      IF(NFGTH .GT.0) THEN
1390: C
1391:      WRITE(NSYSO,401)
1392:      DO 430 J = 1 , NIGTH
1393: CDEL IF(J.GT.10.AND.J.LT.(NIGTH-10)) GO TO 430
1394:      IF(IOPT.LE.1) THEN
1395:          IF(J.GT.10.AND.J.LT.(NIGTH-10)) GO TO 430
1396:          ENDIF
1397: C
1398:      WRITE(NSYSO,402) J,EMESH5(J),(TMPHS(K),K=1,NTTHSC)
1399:      WRITE(NSYSO,403)
1400:      DO 420 I = 1 , NFGTH
1401:      SUM      = 0.0
1402:      DO 410 K = 1 , NTTHSC
1403:      SUM      = SUM + ETRAN7(K,I,J)
1404: 410 CONTINUE
1405:      IF(SUM.GT.0.0) THEN
1406:          WRITE(NSYSO,404) I,EMESH5(I),(ETRAN7(K,I,J),K=1,NTTHSC)
1407:          ENDIF
1408: 420 CONTINUE
1409:      WRITE(NSYSO,403)
1410: 430 CONTINUE
1411:      ENDIF
1412: C
1413: 401 FORMAT(1H1,10X,'*****
1414:      *      /1H ,10X,'*      THERMAL INELASTIC ENERGY TRANSFER      * ',
1415:      *      /1H ,10X,'*****
1416: 402 FORMAT(/1H ,10X,'SOURCE ENERGY NO = ',I3,5X,
1417:      *      'INCIDENT ENERGY= ',F10.6,' EV'
1418:      *      //1H ,5X,'NO SINK E.(EV)',12(F7.1,2H K,:))
1419: 403 FORMAT(1H ,1X,13(10H-----))
1420: 404 FORMAT(1H ,4X,I3,F9.6,2X,12F9.6)
1421: C
1422: C *** PRINT THERMAL INELASTIC ANGULAR DATA
1423: C
1424:      IF(MXNGTH.GT.0) THEN
1425: C
1426:      WRITE(NSYSO,501) MXNGTH
1427:      J      = 0
1428:      DO 530 JJ = 1 , NFGTH
1429: C
1430:      IF(IOPT.LE.1) THEN

```

src/tools/mvpdump.f

```

1431:      IF(JJ.GT.10.AND.JJ.LE.(NFGTH-9)) THEN
1432:          J = J + JJ
1433:          GO TO 530
1434:      ENDIF
1435:
1436: C
1437:      WRITE(NSYSO,502) JJ,EMESH5(JJ),(TMPHS(K),K=1,NTTHSC)
1438:      WRITE(NSYSO,503)
1439: C
1440:      DO 520 II= 1 , JJ
1441:          J = J + 1
1442:          DO 520 I = 1 , 5
1443:              SUM = 0.0
1444:              DO 510 K = 1 , NTTHSC
1445:                  SUM = SUM + ANGTH7(K,I,J)
1446:      510 CONTINUE
1447:      IF(SUM.GT.0.0) THEN
1448:          WRITE(NSYSO,504) J,I,(ANGTH7(K,I,J),K=1,NTTHSC)
1449:      ENDIF
1450:      520 CONTINUE
1451:      WRITE(NSYSO,503)
1452:      530 CONTINUE
1453: C
1454: CDEL WRITE(NSYSO,*) ' ** J MXNGTH = ',J,MXNGTH
1455: ENDIF
1456: C
1457: 501 FORMAT(1H1,10X,'*****
1458: * /1H ,10X,'* THERMAL INELASTIC ANGULAR DATA * '
1459: * /1H ,10X,' < NO OF DATA = ',I6,' > * '
1460: * /1H ,10X,'*****
1461: 502 FORMAT(/1H ,10X,'SINK ENERGY NO = ',I3,5X,
1462: * 'SINK ENERGY= ',F10.6,' EV'
1463: * //1H ,2X,' J K',12(F9.1,2H K,:))
1464: 503 FORMAT(1H ,1X,13(10H-----))
1465: 504 FORMAT(1H ,1X,I4,I2,1P12E11.4)
1466: C
1467: C *** PRINT THERMAL ELASTIC CROSS SECTION FOR FREE-GAS
1468: C
1469:      IF(NPTCUT.GT.0) THEN
1470: C
1471:          NN = NPTCUT
1472:          LOOP = (NN+1)/2
1473:          WRITE(NSYSO,601)
1474: C
1475:          DO 600 LOP = 1 , LOOP
1476:              LOP1 = (LOP-1)*2 + 1
1477:              LOP2 = LOP *2
1478:              KOP1 = NPTS + 1 - LOP1
1479:              KOP2 = NPTS + 1 - LOP2
1480: C
1481:              IF(LOP2.LE.NN) THEN
1482:                  WRITE(NSYSO,602) LOP1,ENERGY(KOP1),CRSMT2(LOP1),
1483: *                               LOP2,ENERGY(KOP2),CRSMT2(LOP2)
1484:              ELSE
1485:                  WRITE(NSYSO,602) LOP1,ENERGY(KOP1),CRSMT2(LOP1)
1486:              ENDIF
1487:      600 CONTINUE
1488:      ENDIF
1489: C
1490: 601 FORMAT(1H1,10X,'*****
1491: * /1H ,10X,'* THERMAL ELASTIC X-SECTION FOR FREE-GAS * '
1492: * /1H ,10X,'*****
1493: * //1H ,2X,' NO ENERGY(EV)',2X,'X-SECTION'
1494: * ,4X,' NO ENERGY(EV)',2X,'X-SECTION')
1495: 602 FORMAT(1H , I6,2X,1P2E12.5,0P,I6,2X,1P2E12.5)

```

```

1496: C
1497: C *** END OF PROCESS
1498: C
1499:      RETURN
1500:      END
1501: C
1502: C=====
1503: C
1504:      SUBROUTINE LISTUN(EMESH ,INTE ,Q ,IUPOS ,SIGT ,SIGE ,
1505: @ SIGF ,NE ,NBIN ,NSYSO ,NWRK )
1506: C
1507: C PRINT UNRESOLVED BMC SAMPLING DATA
1508: C
1509:      REAL*4 EMESH(NE)
1510:      REAL*4 Q(NBIN,NE),SIGT(NBIN,NE),SIGE(NBIN,NE),SIGF(NBIN,NE)
1511:      INTEGER*4 INTE (NE),IUPOS(2,NBIN,NE)
1512: C
1513: C
1514: C
1515:      READ(NWRK) (( Q(J,I) ,J=1,NBIN ) ,
1516: + ( IUPOS(1,J,I) ,IUPOS(2,J,I) ,J=1,NBIN ) ,
1517: + ( SIGT(J,I),SIGE(J,I),SIGF(J,I),J=1,NBIN ) , I=1,NE )
1518: C
1519:      WRITE(NSYSO,1001)
1520:      DO 1000 J = 1 , NE
1521:          WRITE(NSYSO,1002) J,EMESH(J),INTE(J)
1522:          WRITE(NSYSO,1003)
1523:          DO 1500 I = 1 , NBIN
1524:              WRITE(NSYSO,1004) I,SIGT(I,J),SIGE(I,J),SIGF(I,J),
1525: + Q(I,J),(IUPOS(K,I,J),K=1,2)
1526: + ,SIGT(I,J)-SIGE(I,J)-SIGF(I,J)
1527:      1500 CONTINUE
1528:          WRITE(NSYSO,1003)
1529:      1000 CONTINUE
1530: C
1531:      RETURN
1532: C
1533: 1001 FORMAT(1H1,10X,'*****
1534: * /1H ,10X,'* UNRESOLVED RESONANCE PROBABILITY TABLE * '
1535: * /1H ,10X,' ( BMC SAMPLING) * '
1536: * /1H ,10X,'*****
1537: 1002 FORMAT(/1H ,5X,
1538: *' LOP = ',I3,5X,'INCIDENT ENERGY= ',1P12.5,' EV',5X,' INT =',I5,
1539: * //1H ,5X,'NO SIG-T(BARN) SIG-E(BARN) SIG-F(BARN) Q-ANSWER ' ,
1540: * ' IANS1 IANS2 SIG-C(BARN) ')
1541: C1003 FORMAT(1H ,3X,7(10H-----))
1542: 1003 FORMAT(1H ,3X,8(10H-----))
1543: C1004 FORMAT(1H ,4X,I3,1P4E12.5,4X,2I6)
1544: 1004 FORMAT(1H ,4X,I3,1P4E12.5,4X,2I6,1P12.5)
1545: C
1546:      END
1547: C
1548: C=====
1549: C
1550:      SUBROUTINE LISTX1( XX , YY , NN , IST , NSYSO )
1551: C
1552: C----X-SECTION PRINT-OUT ROUTINE
1553: C
1554:      DIMENSION XX(1) , YY(1)
1555: C
1556:      LOOP = (NN+1)/2
1557:      WRITE(NSYSO,101)
1558: C
1559:      DO 100 LOP = 1 , LOOP
1560:          LOP1 = (LOP-1)*2 + 1

```

src/tools/mvdpdump.f

```

1561:      LOP2 = LOP * 2
1562:      J1 = LOP1 + IST - 1
1563:      J2 = LOP2 + IST - 1
1564:      IF(LOP2.LE.NN) THEN
1565:          WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1),
1566:          *                J2,XX(LOP2),YY(LOP2)
1567:      ELSE
1568:          WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1)
1569:      ENDIF
1570: 100 CONTINUE
1571: C
1572: 101 FORMAT(1H ,2X,' NO ENERGY(EV)',2X,'X-SECTION'
1573: *        ,4X,' NO ENERGY(EV)',2X,'X-SECTION')
1574: 102 FORMAT(1H , I6,2X,1P2E12.5,0P,I6,2X,1P2E12.5)
1575: C
1576: RETURN
1577: END
1578: C
1579: C-----
1580: C
1581: SUBROUTINE LISTX2( XX , YY , NN , IST , NSYSO )
1582: C
1583: C-----X-SECTION PRINT-OUT ROUTINE
1584: C
1585: DIMENSION XX(1) , YY(1)
1586: C
1587: LOOP = (NN+1)/2
1588: WRITE(NSYSO,101)
1589: C
1590: DO 100 LOP = 1 , LOOP
1591: LOP1 = (LOP-1)*2 + 1
1592: LOP2 = LOP * 2
1593: J1 = LOP1 + IST - 1
1594: J2 = LOP2 + IST - 1
1595: IF(LOP2.LE.NN) THEN
1596:     WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1),
1597:     *                J2,XX(LOP2),YY(LOP2)
1598: ELSE
1599:     WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1)
1600: ENDIF
1601: 100 CONTINUE
1602: C
1603: 101 FORMAT(1H ,2X,' NO ENERGY(EV)',2X,' YIELD '
1604: *        ,4X,' NO ENERGY(EV)',2X,' YIELD ')
1605: 102 FORMAT(1H , I6,2X,1P2E12.5,0P,I6,2X,1P2E12.5)
1606: C
1607: RETURN
1608: END
1609: C
1610: C-----
1611: C
1612: SUBROUTINE LISTX3( XX , YY , NN , IST , NSYSO )
1613: C
1614: C-----X-SECTION PRINT-OUT ROUTINE
1615: C
1616: DIMENSION XX(1) , YY(1)
1617: C
1618: LOOP = (NN+1)/2
1619: WRITE(NSYSO,101)
1620: C
1621: DO 100 LOP = 1 , LOOP
1622: LOP1 = (LOP-1)*2 + 1
1623: LOP2 = LOP * 2
1624: J1 = LOP1 + IST - 1
1625: J2 = LOP2 + IST - 1

```

```

1626:      IF(LOP2.LE.NN) THEN
1627:          WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1),
1628:          *                J2,XX(LOP2),YY(LOP2)
1629:      ELSE
1630:          WRITE(NSYSO,102) J1,XX(LOP1),YY(LOP1)
1631:      ENDIF
1632: 100 CONTINUE
1633: C
1634: 101 FORMAT(1H ,2X,' NO ENERGY(EV)',2X,'NU-BAR '
1635: *        ,4X,' NO ENERGY(EV)',2X,'NU-BAR ')
1636: 102 FORMAT(1H , I6,2X,1P2E12.5,0P,I6,2X,1P2E12.5)
1637: C
1638: RETURN
1639: END
1640: C
1641: C-----
1642: C
1643: SUBROUTINE LISTXS( XX , YY , NN , ELOW , EHI , NSYSO )
1644: C
1645: C-----X-SECTION PRINT-OUT ROUTINE
1646: C
1647: DIMENSION XX(1) , YY(1)
1648: C
1649: IF(XX( 1).LT.ELOW) RETURN
1650: IF(XX(NN).GT.EHI ) RETURN
1651: C
1652: IST = 1
1653: IEND = NN
1654: C
1655: DO 100 J = 1 , NN-1
1656: E1 = XX(J )
1657: E2 = XX(J+1)
1658: IF(EHI.GT.E2.AND.EHI .LE.E1) IST = J
1659: 100 CONTINUE
1660: IF(EHI.EQ.XX(NN)) IST = NN
1661: C
1662: DO 200 J = 1 , NN-1
1663: E1 = XX(J )
1664: E2 = XX(J+1)
1665: IF(ELOW.GE.E2.AND.ELOW.LT.E1) IEND = J+1
1666: 200 CONTINUE
1667: IF(ELOW.EQ.XX(1)) IEND = 1
1668: C
1669: NP = IEND - IST + 1
1670: C
1671: CM WRITE(NSYSO,10) IST,IEND,NP
1672: CM WRITE(NSYSO,11) EHI,ELOW
1673: CM WRITE(NSYSO,12) XX(IST),XX(IEND)
1674: CM 10 FORMAT(1H , ' ## IST IEND NP ## ',3I6)
1675: CM 11 FORMAT(1H , ' ## EHI ELOW ## ',1P2E12.5)
1676: CM 12 FORMAT(1H , ' ##X(IST) X(IEND)# ',1P2E12.5)
1677: C
1678: CALL LISTX1( XX(IST) , YY(IST) , NP , IST , NSYSO )
1679: C
1680: RETURN
1681: END
1682: C
1683: C-----
1684: C
1685: SUBROUTINE LISU3R( NSYSO , NURT , NBNU3R , NTU3R , IFISUN ,
1686: *                TMPU3R , EUNRTB , SIGEAV , SIGFAV , SIGTAV ,
1687: *                PROBUN , SIGETB , SIGFTB , SIGTTB )
1688: C
1689: C *** PRINT TEMP. DEPENDENT UNRESOLVED PROBABILITY DATA
1690: C

```

src/tools/mvpdump.f

```

1691:      REAL*4      TMPU3R(NTU3R)
1692:      REAL*4      EUNRTB(NURT)
1693:      REAL*4      SIGEAV(NURT)
1694:      REAL*4      SIGFAV(NURT)
1695:      REAL*4      SIGTAV(NURT)
1696:      REAL*4      PROBUN(NBNU3R,NURT)
1697:      REAL*4      SIGETB(NTU3R,NBNU3R,NURT)
1698:      REAL*4      SIGFTB(NTU3R,NBNU3R,NURT)
1699:      REAL*4      SIGTTB(NTU3R,NBNU3R,NURT)
1700: C
1701:      CHARACTER*12  IDREAC(3)
1702: C
1703: COM DATA IDREAC/ 12H TOTAL      , 12H ELASTIC      , 12H FISSION      /
1704: DATA IDREAC/ 12H CAPTURE      , 12H ELASTIC      , 12H FISSION      /
1705: C
1706: C *** START OF PRINTING TO NSYSO DEVICE
1707: C
1708:      WRITE(NSYSO,1001)
1709:      WRITE(NSYSO,1002)
1710:      WRITE(NSYSO,1003)
1711:      DO 1000 J = 1 , NURT
1712:      WRITE(NSYSO,1004) J,EUNRTB(J),SIGTAV(J),SIGEAV(J),SIGFAV(J)
1713:      1000 CONTINUE
1714:      WRITE(NSYSO,1003)
1715: C
1716:      1001 FORMAT(1H1,12X,'*****
1717:      *      /1H ,12X,' * UNRESOLVED RESONANCE AVERAGE X-SECTION *
1718:      *      /1H ,12X,'*****
1719:      1002 FORMAT(/1H ,5X,'NO ENERGY(EV) SIG-T(BARN) SIG-E(BARN)
1720:      *      ' SIG-F(BARN) ')
1721:      1003 FORMAT(1H ,3X,5(11H-----))
1722:      1004 FORMAT(1H ,4X,I3,1P3E12.5)
1723: C
1724: C *** PRINT TEMPERATURE DEPENDENT X-SECTION TABLE
1725: C
1726:      DO 2000 LOP = 1 , 3
1727:      IF(LOP.EQ.3.AND.IFISUN.EQ.0) GO TO 2000
1728: C
1729:      WRITE(NSYSO,2001) IDREAC(LOP)
1730:      DO 1200 J = 1 , NURT
1731:      WRITE(NSYSO,2002) J,EUNRTB(J),(TMPU3R(K),K=1,NTU3R)
1732:      WRITE(NSYSO,2003)
1733:      DO 1100 I = 1 , NBNU3R
1734:      IF(LOP.EQ.1)
1735:      *      WRITE(NSYSO,2004) I,PROBUN(I,J),(SIGTTB(K,I,J),K=1,NTU3R)
1736:      IF(LOP.EQ.2)
1737:      *      WRITE(NSYSO,2004) I,PROBUN(I,J),(SIGETB(K,I,J),K=1,NTU3R)
1738:      IF(LOP.EQ.3)
1739:      *      WRITE(NSYSO,2004) I,PROBUN(I,J),(SIGFTB(K,I,J),K=1,NTU3R)
1740:      1100 CONTINUE
1741:      WRITE(NSYSO,2003)
1742:      1200 CONTINUE
1743:      2000 CONTINUE
1744: C
1745:      2001 FORMAT(1H1,20X,'*****
1746:      *      /1H ,20X,' * UNRESOLVED RESONANCE PROBABILITY TABLE *
1747:      *      /1H ,20X,' (REACTION:' ,A12,' ) *
1748:      *      /1H ,20X,'*****
1749:      2002 FORMAT(/1H ,10X,'ENERGY NO = ',I3,5X,
1750:      *      ' INCIDENT ENERGY= ',1PE12.5,' EV',0P
1751:      *      //1H ,5X,'NO PROBABILITY',12(F7.1,2H K,:))
1752:      2003 FORMAT(1H ,1X,13(10H-----))
1753:      2004 FORMAT(1H ,4X,I3,1PE11.4,0P,12F9.4)
1754: C
1755: C *** END OF PRINTING

```

```

1756: C
1757:      RETURN
1758:      END
1759: C
1760: C=====
1761: C
1762:      SUBROUTINE LLF67A( NEP , Q , IPOS , EBIN , EPROB , NSYSO )
1763: C
1764: C-----SECONDARY NERGY DISTRIBUTION DATA FOR LF=67 (ENERGY BINS + BMC)
1765: C
1766:      REAL*4      Q(NEP),EBIN(NEP),EPROB(NEP)
1767:      INTEGER*4      IPOS(2,NEP)
1768: C
1769: C
1770: C
1771:      WRITE(NSYSO,104)
1772:      WRITE(NSYSO,105)
1773:      DO 50 J = 1 , NEP
1774:      WRITE(NSYSO,103) J,EBIN(J),EPROB(J),Q(J),IPOS(1,J),IPOS(2,J)
1775:      50 CONTINUE
1776:      WRITE(NSYSO,103) J,EBIN(J),EPROB(J)
1777:      WRITE(NSYSO,105)
1778: C
1779:      103 FORMAT(1H ,5X,I3,3X,1P3E12.5,2X,2I5)
1780:      104 FORMAT(1H0,5X,'NO.',3X,' E-2ND ', 'PROBABILITY ',
1781:      @      ' Q ', ' IANS1 IANS2')
1782:      105 FORMAT(1H ,3X,7(10H-----))
1783: C
1784:      RETURN
1785:      END
1786: C
1787: C=====
1788: C
1789:      SUBROUTINE LTHETA(EMESH ,INT ,THETA,BTHETA,NE ,NSYSO ,MT ,LF ,U)
1790: C
1791: C----- PRINT ENERGY DEPENDENT THETA VALUE
1792: C
1793:      REAL*4      EMESH(NE),THETA(NE),BTHETA(NE)
1794:      INTEGER*4      INT (NE)
1795: C
1796:      WRITE(NSYSO,101) MT,LF,NE,U
1797:      WRITE(NSYSO,102)
1798:      WRITE(NSYSO,103)
1799:      DO 100 I = 1 , NE
1800:      IF(LF.LT.10) THEN
1801:      WRITE(NSYSO,104) I,INT(I),EMESH(I),THETA(I)
1802:      ELSE
1803:      WRITE(NSYSO,104) I,INT(I),EMESH(I),THETA(I),BTHETA(I)
1804:      ENDIF
1805:      100 CONTINUE
1806:      WRITE(NSYSO,103)
1807: C
1808:      101 FORMAT(1H1,20X,' *****
1809:      *      /1H ,20X,' * THETA VALUE INFORMATION *
1810:      *      /1H ,20X,' *****
1811:      *      ///1H ,10X,' MT = ',I3,' LF = ',I3,' NE = ',I3,
1812:      *      ' U-PRIME = ',1PE12.5//)
1813:      102 FORMAT(1H ,5X,' NO INT ENERGY(EV) THETA1 THETA2 ')
1814:      103 FORMAT(1H ,3X,'-----')
1815:      104 FORMAT(1H ,5X,2I3,1P3E12.5)
1816: C
1817:      RETURN
1818:      END
1819: C
1820: C=====

```

src/tools/mvdpdump.f

```

1821: C
1822: SUBROUTINE MDUMP(ENERGY,ADATA,IDATA,A,IA,XMESH,YMESH,MEMORY)
1823: C
1824: PARAMETER ( MTMAX = 120 , NKMAX = 90 )
1825: PARAMETER ( MXREC = 1000 )
1826: C
1827: INTEGER*4 OTAPE
1828: C
1829: COMMON /HEADER/ C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
1830: COMMON /UNITS / OTAPE
1831: COMMON /LEADER/ C1,Q,L1,L2,N1,N2
1832: C
1833: COMMON /MAINIO/ NSYSI,NSYSO,NOUT,NWRK,NWRK2
1834: COMMON /CONTRL/ MATT,ZA,ELUNR,EHUNR,CRSMIN,LEMORY,IUNRES,INTUNR,
1835: + IDUMP,IDBG
1836: C
1837: COMMON /HEAD1 / NPTS,NTDATA,NUNR,NUNR2,NLEV,NBINA,NBINE,NNU,NMT,
1838: + NNK,IGFLAG,LFI,LNU,ITHERM,ISTU,IENDU,LSUNR,LSNU,
1839: + NBINA1,NBINE1,KDUMMY(10),ATW,TLAB,ETHERM,ESAB,
1840: + ELOW,EHI,TSTAR,RDUMMY(12),LNUD,NNF,LSNUD
1841: CM + ELOW,EHI,RDUMMY(13)
1842: COMMON /HEAD2 / MTINFO(MTMAX),MTPAR(MTMAX),ISTMT(MTMAX),
1843: + IENDMT(MTMAX),NEUMT(MTMAX),LCTMT(MTMAX),
1844: + NEANG(MTMAX),NEF5(MTMAX),NEF5(MTMAX),
1845: + MTTHR(MTMAX),LSTF3(MTMAX),LSTF4(MTMAX),
1846: + LSTF5(MTMAX),
1847: + NEF5S(NKMAX,MTMAX),LFF5S(NKMAX,MTMAX),
1848: + LSTF5S(NKMAX,MTMAX),QVAL(MTMAX),
1849: + QVAL2(MTMAX)
1850: C
1851: CADDED NCAP and after FOR VERSION-2 BY JAIS K.KANEKO 10/30/1991
1852: COMMON /HEAD3 / NOGAM,NOGAM3,MTGAM(MTMAX),NGTYPE(MTMAX),
1853: @ NKGAM(MTMAX),N14GAM(MTMAX),N15GAM(MTMAX),
1854: @ LSTGAM(MTMAX),MTTOG(MTMAX),NEGYLD(MTMAX),
1855: @ NEF5G(MTMAX),NKF5G(MTMAX),LCTMTG(MTMAX),
1856: @ NGREC,LENGAM(MXREC)
1857: @ ,NCAP,NCAPG,MTCAP(MTMAX),MTCAPG(MTMAX),
1858: @ EG1L(MTMAX),EG2U(MTMAX),EGRANG(2,MTMAX)
1859: C
1860: COMMON /F6CONT/ LF6,LSTF6,MTTOF6(MTMAX),EF61L(MTMAX),
1861: + EF62U(MTMAX),F6RANG(2,MTMAX)
1862: CEND
1863: C
1864: REAL*4 A(MEMORY)
1865: INTEGER*4 IA(MEMORY)
1866: REAL*4 ADATA(NTDATA)
1867: INTEGER*4 IDATA(NTDATA)
1868: REAL*4 ENERGY(NPTS)
1869: REAL*4 XMESH (NPTS)
1870: REAL*4 YMESH (NPTS)
1871: INTEGER*4 INT(10),NBT(10)
1872: C
1873: INTEGER*4 LSTF5G(MTMAX)
1874: INTEGER*4 LSTF4G(MTMAX)
1875: INTEGER*4 NEGANG(MTMAX)
1876: REAL*4 ESK (MTMAX)
1877: C
1878: CHARACTER*8 MATT
1879: CHARACTER*4 TITLE(30)
1880: INTEGER*4 HDATE(2)
1881: C
1882: INTEGER*4 GDUMP,TDUMP
1883: C
1884: C-----READ INPUT DATA
1885: C

```

```

1886: CMS WRITE(NSYSO,10)
1887: WRITE(6,10)
1888: READ (NSYSI, *) PHI,PLOW
1889: C
1890: CMS WRITE(NSYSO,11)
1891: WRITE(6,11)
1892: IPRF1 = 1
1893: IPRX3 = 1
1894: IPRX4 = 0
1895: IPRX5 = 0
1896: IPRUNR = 0
1897: C
1898: READ (NSYSI, *) IPRF1,IPRX3,IPRX4,IPRX5,IPRUNR
1899: C
1900: 10 FORMAT(/1H , ' * ENTER ENERGY RAGNGE TO BE DUMP IN FREE FORMAT * '
1901: * /1H , ' *** EHI ELOW *** ' )
1902: 11 FORMAT(/1H , ' * ENTER FILE-WISE PRINT OPTION IN FREE FORMAT * '
1903: * /1H , ' *** IPRF1 IPRX3,IPRX4,IPRX5,IPRUNR *** ' )
1904: C
1905: C----- READ LIBRARY
1906: C
1907: REWIND NOUT
1908: READ(NOUT) MATT,HDATE,TITLE,
1909: + NPTS,NTDATA,NUNR,NUNR2,NLEV,NBINA,NBINE,NNU,NMT,
1910: + NNK,IGFLAG,LFI,LNU,ITHERM,ISTU,IENDU,LSUNR,LSNU,
1911: + NBINA1,NBINE1,NNUD,LNUD,NNF,LSNUD,
1912: + NCAP,NCAPG,LF6,LSTF6,LLSSF,LSTDOP,(KDUMMY(I),I=1,4),
1913: + ATW,TLAB,ETHERM,ESAB,ELOW,EHI,TSTAR,RDUMMY
1914: C
1915: READ(NOUT) MTINFO,MTPAR,ISTMT,IENDMT,NEUMT,LCTMT,
1916: + NEANG,NKF5,NEF5,MTTHR,LSTF3,LSTF4,LSTF5,
1917: + QVAL,QVAL2,LSTGAM,MTTOG,NEGYLD,NEF5G,NKF5G,LCTMTG
1918: + ,MTCAP,MTCAPG,EG1L,EG2U
1919: READ(NOUT) ADATA
1920: REWIND NOUT
1921: C
1922: ZA = RDUMMY(1)
1923: MATH = RDUMMY(2)
1924: MFH = 3
1925: C
1926: IF(PHI.EQ.0.0) PHI = EHI
1927: IF(PLOW.EQ.0.0) PLOW = ELOW
1928: IF(PHI.GT.EHI) PHI = EHI
1929: IF(PLOW.LT.ELOW) PLOW = ELOW
1930: CM IF(PHI.LE.PLOW) CALL ERROR('MDUMP ',999)
1931: IF(PHI.LT.PLOW) PHI = PLOW
1932: C
1933: JDUMP = IDUMP/10
1934: NDUMP = IDUMP - 10*JDUMP
1935: C
1936: IDUMP = IDUMP/10
1937: JDUMP = IDUMP/10
1938: GDUMP = IDUMP - 10*JDUMP
1939: C
1940: IDUMP = IDUMP/10
1941: JDUMP = IDUMP/10
1942: TDUMP = IDUMP - 10*JDUMP
1943: C
1944: IST = 0
1945: IEND = 0
1946: NP1 = NPTS + 1
1947: DO 30 I = 1 , NPTS
1948: ENERGY(I) = ADATA(I)
1949: XMESH (I) = ADATA(NP1-I)
1950: IF(I.EQ.1) GO TO 30

```

src/tools/mvdpdump.f

```

1951: C
1952:     IF(PHI .GT.ENERGY(I).AND.PHI .LE.ENERGY(I-1)) IST = I-1
1953:     IF(PLOW.GE.ENERGY(I).AND.PLOW.LT.ENERGY(I-1)) IEND = I
1954: 30 CONTINUE
1955: C
1956:     WRITE(NSYSO,*) ' ** PLOW PHI IEND IST ** ',PLOW,PHI,IEND,IST
1957: C
1958:     IF(LF6.EQ.1) THEN
1959:         ISW = LSTF6 - 1
1960:         DO 31 I = 1 , NMT
1961:             ISW = ISW + 1
1962:             MTOF6(I) = IDATA(ISW)
1963: 31 CONTINUE
1964: C
1965:         DO 32 I = 1 , NMT
1966:             ISW = ISW + 1
1967:             EF61U(I) = ADATA(ISW)
1968: 32 CONTINUE
1969: C
1970:         DO 33 I = 1 , NMT
1971:             ISW = ISW + 1
1972:             EF62U(I) = ADATA(ISW)
1973: 33 CONTINUE
1974: C
1975:     ENDIF
1976: C
1977: C ***** ADDED FOR I-FREE VERSION
1978: C *****
1979: C
1980:     ITFREE = 0
1981:     LENTHS = 0
1982:     LENU3R = 0
1983:     LSTTHS = 0
1984:     LSTU3R = 0
1985: C
1986:     IF(LSTDOP.GT.0) THEN
1987:         ITFREE = 1
1988:         LENTHS = IDATA(LSTDOP)
1989:         LENU3R = IDATA(LSTDOP+1)
1990:         LSTTHS = 0
1991:         IF(LENTHS.GT.0) LSTTHS = LSTDOP + 2
1992:         LSTU3R = 0
1993:         IF(LENU3R.GT.0) THEN
1994:             LSTU3R = LSTDOP + 2
1995:             IF(LENTHS.GT.0) LSTU3R = LSTTHS + LENTHS
1996:         ENDIF
1997:     ENDIF
1998: C
1999:     NTU3R = 0
2000:     NURT = 0
2001:     NBNU3R = 0
2002:     IFISUN = 0
2003:     IF(LSTU3R.GT.0) THEN
2004:         NTU3R = IDATA(LSTU3R)
2005:         NURT = IDATA(LSTU3R+1)
2006:         NBNU3R = IDATA(LSTU3R+2)
2007:         IFISUN = IDATA(LSTU3R+3)
2008:     ENDIF
2009: C
2010:     NTTHSC = 0
2011:     NPTHE = 0
2012:     LCTTHE = 0
2013:     LTC THE = 0
2014:     NRETHE = 0
2015:     NETHE = 0

```

```

2016:     NBN THE = 0
2017:     NPTHIN = 0
2018:     NIGTH = 0
2019:     NFGTH = 0
2020:     MXNGTH = 0
2021:     NATOMT = 0
2022:     MPCUT = 0
2023:     EMAXTH = 0.0
2024:     EHITHE = 0.0
2025:     IF(LSTTHS.GT.0) THEN
2026:         NTTHSC = IDATA(LSTTHS)
2027:         NPTHE = IDATA(LSTTHS + 1 )
2028:         LCTTHE = IDATA(LSTTHS + 2 )
2029:         NRETHE = IDATA(LSTTHS + 3 )
2030:         NETHE = IDATA(LSTTHS + 4 )
2031:         NBN THE = IDATA(LSTTHS + 5 )
2032:         NPTHIN = IDATA(LSTTHS + 6 )
2033:         NIGTH = IDATA(LSTTHS + 7 )
2034:         NFGTH = IDATA(LSTTHS + 8 )
2035:         MXNGTH = IDATA(LSTTHS + 9 )
2036:         NATOMT = IDATA(LSTTHS + 10 )
2037:         NPTCUT = IDATA(LSTTHS + 11 )
2038:         EMAXTH = ADATA(LSTTHS + 12 )
2039:         EHITHE = ADATA(LSTTHS + 13 )
2040:     ENDIF
2041: C
2042: C PRINT OUT LIBRARY INFORMATION
2043: C
2044:     WRITE(NSYSO,101) MATT,HDATE,TITLE
2045:     WRITE(NSYSO,102) NPTS,NTDATA,NUNR,NUNR2,NLEV,NBINA,NBINE,NNU,NMT
2046:     WRITE(NSYSO,116) NNK,IGFLAG,LFI,LNU,ITHERM,ISTU,IENDU,LSUNR,LSNU,
2047: + NBINA1,NBINE1,LF6,LSTF6
2048:     WRITE(NSYSO,117) NNUD,LNUD,NNF,LSNUD,NCAP,NCAPG
2049:     WRITE(NSYSO,103) ATW,TLAB,ETHERM,ESAB,ELow,EHI,TSTAR,ZA,MATH
2050:     IF(NLEV .GT.0) WRITE(NSYSO,104) (MTTHR(I),I=1,NLEV)
2051:     IF(NCAP .GT.0) WRITE(NSYSO,124) (MTCAP(I),I=1,NCAP)
2052:     IF(NCAPG.GT.0) WRITE(NSYSO,125) (MTCAPG(I),I=1,NCAPG)
2053: C
2054: 101 FORMAT(1H1,20X,'*****',
2055: @ /1H,20X,'* MVP MATERIAL LIBRARY INFORMATION * ',
2056: @ /1H,20X,'*****',
2057: @ /1H,20X,'* MATERIAL ID = ',A8,' * ',
2058: @ /1H,20X,'*****',
2059: @ /1H,20X,'* PRODUCTION DATE : ',2A4,' * ',
2060: @ /1H,20X,'*****',
2061: @ //1H,10X,' COMMENT : ',15A4,
2062: @ /1H,10X,' : ',15A4//)
2063: 102 FORMAT(1H,15X,
2064: @ 'NPTS : NO OF SMOOTH DATA GRID RECORD.....',I10,
2065: @ /1H,15X,'NTDATA: TOTAL RECORD OF #3 RECORD (WORDS)...',I10,
2066: @ /1H,15X,'NUNR : NO OF UNRESOLVED PROB. TABLES .....',I10,
2067: @ /1H,15X,'NUNR2 : LENGTH OF UNRESOLVED PROB. TABLE ...',I10,
2068: @ /1H,15X,'NLEV : NO OF THRESHOLD REACTIONS .....',I10,
2069: @ /1H,15X,'NBINA : NO OF BINS IN ANGULAR DIS. TABLE...',I10,
2070: @ /1H,15X,'NBINE : NO OF BINS IN ENERGY DIS. TABLE...',I10,
2071: @ /1H,15X,'NNU : LENGTH OF NU-DATA .....',I10,
2072: @ /1H,15X,'NMT : TOTAL NO OF REACTION CONSIDERED.....',I10)
2073: 116 FORMAT(1H,15X,
2074: @ 'NNK : MAX NO OF SUB-SECTIONS IN FILE-5....',I10,
2075: @ /1H,15X,'IGFLAG: GAMMA PRODUCTION DATA FLAG .....',I10,
2076: @ /1H,15X,'LFI : FLAG OF FISSION REACTION .....',I10,
2077: @ /1H,15X,'LNU : NU-DATA FLAG (1/2=POLY/TAB).....',I10,
2078: @ /1H,15X,'ITHERM: FLAG OF THERMAL SCATTERING .....',I10,
2079: @ /1H,15X,'ISTU : UPPER ENERGY MESH POINT OF U.R. ....',I10,
2080: @ /1H,15X,'IENDU : LOWER ENERGY MESH POINT OF U.R. ....',I10,

```


src/tools/mvpdump.f

```

2081:      @ /1H ,15X,'LSUNR : START POSITION OF U.R. TABLE .....',I10,
2082:      @ /1H ,15X,'LSNU : START POSITION OF NU-DATA.....',I10,
2083:      @ /1H ,15X,'NBINA1: NBINA + 1 .....',I10,
2084:      @ /1H ,15X,'NBINE1: NBINE + 1 .....',I10,
2085:      @ /1H ,15X,'LF6 : FLAG OF DDX DATA (FILE6) .....',I10,
2086:      @ /1H ,15X,'LSTF6 : START POSIOTN FOR DDX CONTROL .....',I10)
2087: 117 FORMAT(1H ,15X,
2088:      @ 'NNUD : LENGTH OF DELAYED NU-DATA.....',I10,
2089:      @ /1H ,15X,'LNUD : DELAYED NU-DATA FLAG(1/2=POLY/TAB)...',I10,
2090:      @ /1H ,15X,'NNF : NO OF PRECURSOR FAMILIES .....',I10,
2091:      @ /1H ,15X,'LSNUD : START POSITION OF DELAYED NU .....',I10,
2092:      @ /1H ,15X,'NCAP : NO OF CAPTURE REACTIONS .....',I10,
2093:      @ /1H ,15X,'NCAPG : NO OF GAMMA YIELD REAC. BY CAPTURE..',I10)
2094: 103 FORMAT(1H ,15X,
2095:      @ 'ATW : ATOMIC WEIGHT IN N.M.U. ....',F12.3,
2096:      @ /1H ,15X,'TLAB : LABORATORY TEMPERATURE IN KELVIN ...',F12.3,
2097:      @ /1H ,15X,'ETHERM: UPPER ENERGY FOR THERMAL ELA. MOD...',F12.3,
2098:      @ /1H ,15X,'ESAB : HIGHEST ENERGY FOR THERMAL INELA...',F12.3,
2099:      @ /1H ,15X,'ELOW : LOWER ENERGY DEFINED (EV) .....',F12.5,
2100:      @ /1H ,15X,'EHI : UPPER ENERGY DEFINED (EV) .....',F12.1,
2101:      @ /1H ,15X,'TSTAR : EFFECTIVE TEMPERATURE IN KELVIN ....',F12.3,
2102:      @ /1H ,15X,'ZA : THE (Z&A) DESIGNATION .....',F12.1,
2103:      @ /1H ,15X,'MATNO : MATERIAL NUMBER IN ENDF .....',I12/)
2104: 104 FORMAT(1H ,15X,
2105:      @ 'MTHR : THRESHOLD REACTION MT ID .....',I10I4)
2106: 124 FORMAT(1H ,15X,
2107:      @ 'MTCAP : REACTION ID FOR CAPTURE .....',I10I4)
2108: 125 FORMAT(1H ,15X,
2109:      @ 'MTCAPG: REACTION ID FOR (CAPTURE,GAMMA).....',I10I4)
2110: C
2111:      WRITE(NSYSO,105)
2112:      WRITE(NSYSO,106)
2113:      WRITE(NSYSO,107)
2114:      DO 50 MT = 1 , NMT
2115:      IF(MTINFO(MT).LE.0) GO TO 50
2116:      WRITE(NSYSO,108) MT,MTINFO(MT),ISTMT(MT),IENDMT(MT),MTPAR(MT),
2117:      + NEUMT(MT),QVAL(MT),QVAL2(MT)
2118: C
2119: C *** OUTPUT FILE-3 DATA
2120: C
2121:      IF(OTAPE.LE.0) GO TO 50
2122: C
2123:      C1 = TLAB
2124:      Q = QVAL(MT)
2125:      MTH = MT
2126:      NP = MTINFO(MT)
2127:      IST0 = NPTS + 1 - IENDMT(MT)
2128:      NBT(1) = NP
2129:      INT(1) = 2
2130:      N1 = 1
2131:      N2 = NP
2132:      L1 = 0
2133:      L2 = 0
2134: C
2135:      MST = LSTF3(MT) - 1
2136:      NP1 = NP + 1
2137:      DO 40 I = 1 , NP
2138:      YMESH(NP1-I) = ADATA(MST+I)
2139: 40 CONTINUE
2140: C
2141:      WRITE(6,*) ' ** MT NP MST IST0 : ',MT,NP,MST,IST0
2142:      WRITE(6,*) ' ** EMESH(IST0) E1 : ',XMESH(IST0),ADATA(IENDMT(MT))
2143: C
2144:      CALL CARDO ( ZA , ATW , 0 , 99 , 0 , 0 )
2145:      CALL CARDO ( C1 , Q , L1, L2 , N1 , N2 )

```

```

2146:      CALL TERPO ( NBT , INT , 1 )
2147:      CALL POINTV( XMESH(IST0),YMESH,NP)
2148:      CALL SENDO
2149: C
2150:      50 CONTINUE
2151:      WRITE(NSYSO,107)
2152: C
2153: 105 FORMAT(1H1,20X,'*****',
2154:      @ /1H ,20X,'* #2 RECORD INFROMATION (PART 1) * ',
2155:      @ /1H ,20X,'*****'//)
2156: 106 FORMAT(1H ,10X,' MT MTINFO ISTMT IENDMT MTPAR NEUMT ',
2157:      @ ' QVAL(EV) QVAL2(EV) ')
2158: 107 FORMAT(1H , 9X,7(10H-----))
2159: 108 FORMAT(1H ,10X,I3,5I8,1P2E12.5)
2160: C
2161:      WRITE(NSYSO,109)
2162:      WRITE(NSYSO,110)
2163:      WRITE(NSYSO,111)
2164: C
2165:      DO 60 MT = 1 , NMT
2166:      IF(MTINFO(MT).LE.0) GO TO 60
2167:      CB6 IF(MTINFO(MT).LE.0.AND.NKF5(MT).LE.0) GO TO 60
2168:      ISAVE = IABS(NKF5(MT))
2169:      IF(MTINFO(MT).LE.0.AND.ISAVE.EQ.0) GO TO 60
2170:      WRITE(NSYSO,112) MT,LSTF3(MT),LCTMT(MT),NEANG(MT),LSTF4(MT),
2171:      + NKF5(MT),NEF5(MT),LSTF5(MT)
2172:      CM
2173:      CM IF(ISAVE.GT.0) THEN
2174:      CM JSAVE = ISAVE + 1
2175:      CM WRITE(NSYSO,113) (LFF5S(J,MT),J=1,JSAVE)
2176:      CM WRITE(NSYSO,114) (NEF5S(J,MT),J=1,JSAVE)
2177:      CM WRITE(NSYSO,115) (LSTF5S(J,MT),J=1,JSAVE)
2178:      CM ENDF
2179: 60 CONTINUE
2180:      WRITE(NSYSO,111)
2181: C
2182: 109 FORMAT(1H1,20X,'*****',
2183:      @ /1H ,20X,'* #2 RECORD INFROMATION (PART 2) * ',
2184:      @ /1H ,20X,'*****'//)
2185: 110 FORMAT(1H ,10X,' MT LSTF3 LCTMT NEANG LSTF4 NKF5 ',
2186:      @ ' NEF5 LSTF5 ')
2187: 111 FORMAT(1H , 9X,7(10H-----))
2188: 112 FORMAT(1H ,10X,I3,8I8)
2189: 113 FORMAT(1H ,10X,' LFF5S : ',10I8)
2190: 114 FORMAT(1H ,10X,' NEF5S : ',10I8)
2191: 115 FORMAT(1H ,10X,' LSTF5S : ',10I8)
2192: C
2193:      IF(LF6 .EQ.1) THEN
2194:      WRITE(NSYSO,161)
2195:      WRITE(NSYSO,162)
2196:      DO 65 MT = 1 , NMT
2197:      IF(MTTOF6(MT).LE.0) GO TO 65
2198:      WRITE(NSYSO,163) MT,MTTOF6(MT),EF61L(MT),EF62U(MT)
2199: 65 CONTINUE
2200:      WRITE(NSYSO,162)
2201:      ENDF
2202: C
2203: 161 FORMAT(1H1,20X,'*****',
2204:      @ /1H ,20X,'* #2 RECORD INFROMATION (DDX DATA) * ',
2205:      @ /1H ,20X,'*****',
2206:      @ ///1H ,10X,' MT MTTOF6 EF61L(EV) EF62U(EV)')
2207: 162 FORMAT(1H , 9X,4(10H-----))
2208: 163 FORMAT(1H ,10X,I3,I8,3X,1P2E12.5)
2209: C
2210: C *** PRINT TFREE CONTROL DATA

```

src/tools/mvpdump.f

```

2211: C
2212:     IF(ITFREE.GT.0) THEN
2213:         WRITE(NSYSO,141) LSTDOP,LSTTHS,LSTU3R,LENTHS,LENU3R
2214: C
2215:         IF(LSTU3R.GT.0) THEN
2216:             WRITE(NSYSO,142) NTU3R,NURT,NBNU3R,IFISUN
2217:             LSTTU3 = LSTU3R + 3
2218:             ISW0 = LSTTU3 + NTU3R
2219:             WRITE(NSYSO,143) (ADATA(LSTTU3+I),I=1,NTU3R)
2220:             WRITE(NSYSO,148) (ADATA(ISW0 +I),I=1,NTU3R)
2221:             ENDIF
2222:         IF(LSTTHS.GT.0) THEN
2223:             WRITE(NSYSO,144) NTHSC,NPTHE ,LCTTHE,NRETHE,
2224: +             NETHE ,NBNTHE,NPTHIN,NIGTH ,NFGTH,
2225: +             MXNGTH,NATOMT,NPTCUT,EMAXTH,EHITHE
2226:             LSTTSC = LSTTHS + 13
2227:             ISW1 = LSTTSC + NTHSC
2228:             ISW2 = LSTTSC + NTHSC*2
2229:             WRITE(NSYSO,145) (ADATA(LSTTSC+I),I=1,NTHSC)
2230:             WRITE(NSYSO,146) (ADATA(ISW1 +I),I=1,NTHSC)
2231:         IF(NPTHE.GT.0) WRITE(NSYSO,147) (IDATA(ISW2 +I),I=1,NTHSC)
2232:             ENDIF
2233:         ENDIF
2234: C
2235: C
2236: 141 FORMAT(1H1,20X,'*****
2237: 1 /1H ,15X,'* TFREX CONTROL DATA INFORMATION *
2238: 2 /1H ,20X,'*****
2239: 3//1H ,15X,'LSTDOP: START ADDRESS OF TEMP. DEP. DATA ...',I10,
2240: 4 /1H ,15X,'LSTTHS: START ADDRESS OF THERMAL SCATTERING...',I10,
2241: 5 /1H ,15X,'LSTU3R: START ADDRESS OF U.R.P. TABLE ...',I10,
2242: 6 /1H ,15X,'LENTHS: RECORD LENGTH OF THERMAL SCATTERING...',I10,
2243: 7 /1H ,15X,'LENU3R: RECORD LENGTH OF U.R.P. TABLE ...',I10//)
2244: 142 FORMAT(1H0,15X,'***** UNRESOLVED PROBABILITY DATA CONTROL *****',
2245: 1 /1H ,15X,'NTU3R : NO OF TEMPERATURE IN U.R.P. DATA ...',I10,
2246: 2 /1H ,15X,'NURT : NO OF ENERGY IN FILE-2 U.R. EVAL...',I10,
2247: 3 /1H ,15X,'NBNU3R: NO OF BROBAILITY BIN ...',I10,
2248: 4 /1H ,15X,'IFISUN: FISSION REACTION FLAG(0/1:NO/YES)...',I10)
2249: 143 FORMAT(19H TEMPERATURES .... ,11F10.3)
2250: 148 FORMAT(19H LOG(TEMP.) .... ,11F10.6)
2251: C
2252: 144 FORMAT(1H0,15X,'***** THERMAL SCATTERING DATA CONTROL *****',
2253: 1 /1H ,15X,'NTHSC: NO OF TEMPERATURE IN TH.SC. DATA ...',I10,
2254: 2 /1H ,15X,'NPTHE : NO OF THERMAL ELASTIC ENERGY MESH...',I10,
2255: 3 /1H ,15X,'LCTTHE: FLAME OF ELASTIC SCATTERING ANGLE...',I10,
2256: 4 /1H ,15X,'NRETHE: INTERPOLATION SCHEME FOR INCI. ENG...',I10,
2257: 5 /1H ,15X,'NETHE : NO OF INCIDENT ENERGY IN ELS. SCAT...',I10,
2258: 6 /1H ,15X,'NBNTHE: NO OF ANGLE BINS F9R ELASTIC SCAT...',I10,
2259: 7 /1H ,15X,'NPTHIN: NO OF THERMAL INEL. ENERGY MESH 1...',I10,
2260: 8 /1H ,15X,'NIGTH : NO OF THERMAL INEL. ENERGY MESH 2...',I10,
2261: 9 /1H ,15X,'NFGTH: NO OF THERMAL INEL. ENERGY MESH 3...',I10,
2262: A /1H ,15X,'MXNGTH: =(NFG)*(NFG+1)/2 .....',I10,
2263: C /1H ,15X,'NATOMT: NO OF ATOMS IN MOLCULE .....',I10,
2264: B /1H ,15X,'NPTCUT: NO OF CUT DATA IN ORIGIANL MT=2 ....',I10,
2265: D /1H ,15X,'EMAXTH: = ESAB .....',F10.5,
2266: E /1H ,15X,'EHITHE: HIGHTEST ENERGY OF ISOTROPIC ELS. SC',F10.5//)
2267: C
2268: 145 FORMAT(19H TEMPERATURES .... ,11F10.3)
2269: 146 FORMAT(19H EFFECTIVE TEMP... ,11F10.3)
2270: 147 FORMAT(19H INTERPOL. METHOD. ,11I10 )
2271: C
2272: C
2273: C     OUTPUT PROCESS
2274: C
2275:     IF(NDUMP.LE.0) GO TO 463

```

```

2276: C
2277:     IF(IPRF1.GT.0) THEN
2278: C
2279:     IF(LNU.EQ.1) THEN
2280:         LOC1 = LSNU - 1
2281:         WRITE(NSYSO,151) NNU
2282:         WRITE(NSYSO,152) (ADATA(LOC1+I),I=1,NNU)
2283:         ENDIF
2284:     IF(LNU.EQ.2) THEN
2285: CM     LOC1 = LSNU - 1
2286: CM     LOC2 = LOC1 + NNU
2287:         WRITE(NSYSO,153)
2288: C         DO 160 I = 1, NNU
2289: C         WRITE(NSYSO,154) I,ADATA(LOC1+I),ADATA(LOC2+I)
2290: C 160     CONTINUE
2291:         LOC1 = LSNU
2292:         LOC2 = LOC1 + NNU
2293:         CALL LISTX3( ADATA(LOC1), ADATA(LOC2) , NNU , 1 , NSYSO )
2294:         ENDIF
2295: C
2296: 151 FORMAT(//1H ,5X,' ## NUBAR POLYNOMIAL COEFFICIENT DATA ## ',
2297: * //1H ,10X,' NO OF COMPONENT ==> ',I6//)
2298: 152 FORMAT(1H , ' COEFFICIENT : ',1P5E12.5)
2299: 153 FORMAT(//1H ,15X,' ## NUBAR TABULATION DATA ## ')
2300: CM153 FORMAT(//1H ,5X,' ## NUBAR TABULATION DATA ## ',
2301: CM * //1H , ' NO ENERGY(EV) NU-BAR ')
2302: 154 FORMAT(1H ,I4,2X,1P2E12.5)
2303: C
2304:     IF(LNUD.EQ.1) THEN
2305:         LOC1 = LSNUD - 1
2306:         WRITE(NSYSO,171) NNUD
2307:         WRITE(NSYSO,172) (ADATA(LOC1+I),I=1,NNUD)
2308:         LOC3 = LOC1 + NNUD
2309:         ENDIF
2310:     IF(LNUD.EQ.2) THEN
2311: CM     LOC1 = LSNUD - 1
2312: CM     LOC2 = LOC1 + NNUD
2313: CM     WRITE(NSYSO,173)
2314: CM     DO 170 I = 1, NNUD
2315: CM     WRITE(NSYSO,174) I,ADATA(LOC1+I),ADATA(LOC2+I)
2316: CM170     CONTINUE
2317:         LOC1 = LSNUD
2318:         LOC2 = LOC1 + NNUD
2319:         CALL LISTX3( ADATA(LOC1), ADATA(LOC2) , NNUD, 1 , NSYSO)
2320:         LOC3 = LOC2 + NNUD - 1
2321:         ENDIF
2322: C
2323:     IF(NNF.GT.0) THEN
2324:         WRITE(NSYSO,175)
2325:         DO 180 I = 1, NNF
2326:             WRITE(NSYSO,174) I,ADATA(LOC3+I)
2327: C 180     CONTINUE
2328:             ENDIF
2329: C
2330:     ENDIF
2331: C
2332: 171 FORMAT(//1H ,5X,' ## DELAYED NUBAR POLYNOMIAL COEFFICIENT DATA ##'
2333: * //1H ,10X,' NO OF COMPONENT ==> ',I6//)
2334: 172 FORMAT(1H , ' COEFFICIENT : ',1P5E12.5)
2335: 173 FORMAT(//1H ,15X,' ## DELAYED NUBAR TABULATION DATA ## ')
2336: CM173 FORMAT(//1H ,5X,' ## DELAYED NUBAR TABULATION DATA ## ',
2337: CM * //1H , ' NO ENERGY(EV) NU-BAR ')
2338: 174 FORMAT(1H ,I4,2X,1P2E12.5)
2339: 175 FORMAT(//1H ,5X,' ## DECAY CONSTANT OF PRECURSOR FAMILY ## ',
2340: * //1H , ' NO DECAY CONSTANT(1/SEC) ')

```

src/tools/mvpdump.f

```

2341: C
2342: C *** PRINT NEUTRON CROSS SECTION
2343: C
2344:     IF(IPRX3.GT.0) THEN
2345:         DO 200 MT = 1 , NMT
2346:             NP      = MTINFO(MT)
2347:             IF(NP.LE.0) GO TO 200
2348:             WRITE(NSYSO,201) MT,QVAL(MT)
2349:             LOCX    = LSTF3(MT)
2350:             LOCE    = ISTMT(MT)
2351:             CALL LISTXS(ENERGY(LOCE),ADATA(LOCX),NP,PLOW,PHI,NSYSO )
2352: 200 CONTINUE
2353:             ENDIF
2354: C
2355: 201 FORMAT(1H1,15X, ' ## SMOOTH CROSS SECTION DUMP ## ',
2356: * //1H ,10X,' MT = ',I3,10X,' Q-VALUE = ',1PE12.5,' EV'//)
2357: C
2358: C *** PRINT NEUTRON ANGULAR DISTRIBUTION DATA
2359: C
2360:     IF(IPRX4.LE.0) GO TO 333
2361: C
2362:     DO 300 MT = 1 , NMT
2363:         IF(MT.EQ.92) GO TO 300
2364:         NP      = NEANG(MT)
2365:         IF(NP.LE.0) GO TO 300
2366:         LOCX    = LSTF4(MT)
2367:         LOC1    = LOCX
2368:         LOC2    = LOC1 + NP
2369:         LOC3    = LOC2 + NP
2370:         WRITE(NSYSO,301) MT,NP,LCTMT(MT)
2371:         WRITE(NSYSO,302) (ADATA(LOC1+I-1),I=1,NP)
2372:         WRITE(NSYSO,303) (IDATA(LOC2+I-1),I=1,NP)
2373: C
2374:     DO 250 LOP= 1 , NP
2375:         EINT    = ADATA(LOC1)
2376:         IF(EINT.GE.PLOW.AND.EINT.LE.PHI)
2377:         +CALL LISTAN( EINT , ADATA(LOC3), A , NBINA1 , NSYSO )
2378:         LOC1    = LOC1 + 1
2379:         LOC2    = LOC2 + 1
2380:         LOC3    = LOC3 + NBINA1
2381: 250 CONTINUE
2382: 300 CONTINUE
2383: C
2384: 301 FORMAT(1H1,15X, ' ## ANGULAR DITRIBUTIN DUMP ## ',
2385: * //1H ,10X,' MT = ',I3,10X,' NO OF ENERGY = ',I3,
2386: * //1H ,10X,' LCT = ',I3,' (1/2:LAB/CM) '/')
2387: 302 FORMAT(1H ,5X,'INCIDENT ENERGY (EV): ',1PE12.5)
2388: 303 FORMAT(1H ,5X,'INTERPOLATION CODE : ',10I6)
2389: C
2390:     IF(NEANG(92).GT.0) THEN
2391:         NE      = NEANG(92)
2392:         LOC1    = LSTF4(92)
2393:         LOC2    = LOC1 + NE
2394:         LOC3    = LOC2 + NE
2395:         LENG    = 5*NE*(NE+1)/2
2396:         LAST    = LOC3 + LENG - 1
2397:         LENG    = 5*NE*NE
2398:         CALL DIMCHK('MDUMP ',LENG,MEMORY)
2399: CMS ..... REWIND NWRK
2400:         call rwind(NWRK)
2401:         WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2402:         REWIND NWRK
2403:         CALL LENGTH(ADATA(LOC1),IDATA(LOC2),
2404: 1 A(1) ,NE ,NSYSO ,NWRK ,IPRX4 )
2405: 2

```

```

2406:                                     REWIND NWRK
2407:                                     ENDIF
2408: C
2409: C *** PRINT NEUTRON ENERGY DISTRIBUTION DATA
2410: C
2411:     333 CONTINUE
2412:     IF(IPRX5.LE.0) GO TO 463
2413: C
2414:     DO 400 MT = 1 , NMT
2415:         NK      = IABS(NKF5(MT))
2416: CB6 IF(NKF5(MT).LE.0) GO TO 400
2417:         IF(NK    .LE.0) GO TO 400
2418:         ISWF6    = 0
2419:         IF(NKF5(MT).LT.0) ISWF6 = 1
2420: C
2421: CM NE      = NEF5(MT)
2422: CM NK      = NKF5(MT)
2423:         LOC    = LSTF5(MT)
2424:         NE      = IDATA(LOC)
2425:         LOC1    = LOC + 1
2426: CB6 NK      = IDATA(LOC1)
2427:         NK      = IABS( IDATA(LOC1) )
2428:         NEF5(MT) = NE
2429:         NKF5(MT) = NK
2430: C
2431:         DO 310 LOP = 1 , NK
2432:             LOC1    = LOC1 + 1
2433:             LSTF5S(LOP,MT) = IDATA(LOC1)
2434: 310 CONTINUE
2435:             LOC1    = LOC1 + 1
2436: C
2437:         IF(NE.GT.0) THEN
2438:             IF(ISWF6.EQ.0) THEN
2439: CMOD LOC1 = LSTF5(MT)
2440:             LOC2 = LOC1 + NE
2441:             LOC3 = LOC2 + NE
2442:             LAST = LSTF5S(1,MT) - 1
2443:             KOC1 = 1
2444:             KOC2 = KOC1 + NK*NE
2445:             LMAX = 3*NK*NE
2446:             CALL DIMCHK('LISTUN',LMAX,MEMORY)
2447:             REWIND NWRK
2448:             call rwind(NWRK)
2449:             WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2450:             REWIND NWRK
2451:             CALL LISTBC(ADATA(LOC1),IDATA(LOC2),
2452: @ A(KOC1) ,IA(KOC2) ,
2453: @ MT ,NE ,NK ,NSYSO ,NWRK , 0 )
2454:             REWIND NWRK
2455: C
2456:             ELSE
2457:             LOC2 = LOC1 + NE
2458:             LOC3 = LOC2 + NE
2459:             LOC4 = LOC3 + NE
2460:             CALL LISAEY(ADATA(LOC1),IDATA(LOC2),ADATA(LOC3),
2461: + ADATA(LOC4),MT ,NE ,NK ,NSYSO )
2462:             ENDIF
2463:             ENDIF
2464: C
2465:         DO 390 LOP = 1 , NK
2466:             LOC1    = LSTF5S(LOP,MT)
2467:             LFF5S(LOP,MT) = IDATA(LOC1)
2468:             LOC1    = LOC1 + 1
2469:             NEF5S(LOP,MT) = IDATA(LOC1)
2470: C

```

src/tools/mvpdump.f

```

2471:      LF      = LFF5S(LOP,MT)
2472:      NE      = NEF5S(LOP,MT)
2473: CMOD  LOC1   = LSTF5S(LOP,MT)
2474:      LOC1    = LOC1 + 1
2475: C
2476:      WRITE(NSYSO,*) ' ## MT NK LOP LF NE (FILE-5) : ',MT,NK,LOP,LF,NE
2477: C
2478:      IF(LF.EQ.1) THEN
2479:          LOC2 = LOC1 + NE
2480:          LOC3 = LOC2 + NE
2481: CM1992-4-13
2482:          IF(NBINE.GT.0) THEN
2483:              LAST = LOC3 + 2*NE*NBINE1 - 1
2484:              KOC1 = 1
2485:              KOC2 = KOC1 + NE*NBINE1
2486:              LMAX = 2*NE*NBINE1
2487:              CALL DIMCHK('LISTUN',LMAX,MEMORY)
2488: CCCCCC
2489:              REWIND NWRK
2490:              call rwind(NWRK)
2491:              WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2492:              REWIND NWRK
2493:              CALL LISLF1(ADATA(LOC1),IDATA(LOC2),
2494: @              A(KOC1),A(KOC2),
2495: @              NE,NBINE1,NWRK,NSYSO,MT,PLOW,PHI)
2496:              REWIND NWRK
2497: CM1992-4-13
2498:          ELSE
2499:              CALL LILF1B(ADATA(LOC1),IDATA(LOC2),IDATA(LOC3),
2500: @              ADATA(1),IDATA(1),
2501: @              NE,NSYSO,MT,LOP,LF,PLOW,PHI, LAST )
2502:          ENDIF
2503: CM1992-4-13
2504:          GO TO 390
2505:          ENDIF
2506: C
2507:      IF(LF.EQ.4) THEN
2508:          LOC2 = LOC1 + NE
2509:          LOC3 = LOC2 + NE
2510:          LAST = LOC3 + 3*NE*NE - 1
2511:          KOC1 = 1
2512:          KOC2 = KOC1 + NE*NE
2513:          LMAX = 3*NE*NE
2514:          CALL DIMCHK('LISTUN',LMAX,MEMORY)
2515: CCCCCC
2516:          REWIND NWRK
2517:          call rwind(NWRK)
2518:          WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2519:          REWIND NWRK
2520:          CALL LISTBC(ADATA(LOC1),IDATA(LOC2),
2521: @              A(KOC1),IA(KOC2),
2522: @              MT,NE,NE,NSYSO,NWRK,LF)
2523:          GO TO 390
2524:          ENDIF
2525: C
2526:      IF(LF.EQ.2) THEN
2527:          GCOEF1 = ADATA(LOC1)
2528:          GCOEF2 = ADATA(LOC1+1)
2529:          WRITE(NSYSO,321) MT,LF,GCOEF1,GCOEF2
2530:          GO TO 390
2531:          ENDIF
2532: C
2533:      321 FORMAT(/1H ,20X,' ***** ',
2534: *              /1H ,20X,' * MONO-ENERGY DISTRIBUTION DATA * ',
2535: *              /1H ,20X,' ***** ',
2536: *              //1H ,10X,' MT = ',I3,5X,' LF = ',I2,
2537: *              5X,' GCOEF1 = ',1PE12.5,5X,' GCOEF2 = ',1PE12.5//)

```

```

2536: C
2537:      IF(LF.EQ.61) THEN
2538:          LOC2 = LOC1 + NE
2539:          LOC3 = LOC2 + NE
2540:          CALL LILF61(ADATA(LOC1),IDATA(LOC2),IDATA(LOC3),
2541: @              ADATA(1),IDATA(1),NE,NSYSO,
2542: @              MT,PLOW,PHI, LAST )
2543:          GO TO 390
2544:          ENDIF
2545: C
2546:      IF(LF.EQ.66) THEN
2547:          LOC2 = LOC1 + NE
2548:          LOC3 = LOC2 + NE
2549:          LOC4 = LOC3 + NE
2550:          CALL LILF66(ADATA(LOC1),IDATA(LOC2),ADATA(LOC3),NE,
2551: *              NSYSO,MT,LF,IDATA(LOC4))
2552:          GO TO 390
2553:          ENDIF
2554: C
2555:      IF(LF.EQ.67) THEN
2556:          LOC2 = LOC1 + NE
2557:          LOC3 = LOC2 + NE
2558:          CALL LILF67(ADATA(LOC1),IDATA(LOC2),IDATA(LOC3),
2559: @              ADATA(1),IDATA(1),
2560: @              IPRX5,NE,NBINA1,NSYSO,
2561: @              MT,PLOW,PHI, LAST )
2562:          GO TO 390
2563:          ENDIF
2564: C
2565:          LOC2 = LOC1 + NE
2566:          LOC3 = LOC2 + NE
2567:          LOC4 = LOC3 + NE
2568:          LOC5 = LOC4 + NE
2569:          UVAL = ADATA(LOC4)
2570:          IF(LF.GE.11) UVAL = ADATA(LOC5)
2571:          IF(LF.EQ.3) UVAL = 0.0
2572:          IF(LF.EQ.5) THEN
2573:              IF(NBINE.GT.0) THEN
2574:                  LOC5 = LOC4 + NBINE1
2575:                  LOC6 = LOC5 + NBINE1
2576:                  UVAL = ADATA(LOC6)
2577:              ELSE
2578:                  NEP = IDATA(LOC4)
2579:                  LOC6 = LOC4 + 1 + NEP*3 + (NEP+1)*2
2580:                  UVAL = ADATA(LOC6)
2581:              ENDIF
2582:          ENDIF
2583:          CALL LTHETA(ADATA(LOC1),IDATA(LOC2),ADATA(LOC3),ADATA(LOC4),
2584: *              NE,NSYSO,MT,LF,UVAL)
2585:          IF(LF.EQ.5) THEN
2586:              IF(NBINE.GT.0) THEN
2587:                  CALL LISLF5(ADATA(LOC4),ADATA(LOC5),NBINE1,NSYSO,MT)
2588:              ELSE
2589:                  CALL LILF1B(ADATA(LOC1),IDATA(LOC2),1,
2590: @              ADATA(LOC4),IDATA(LOC4),
2591: @              1,NSYSO,MT,LOP,LF,PLOW,PHI, LAST )
2592:              ENDIF
2593:          ENDIF
2594:          390 CONTINUE
2595:          400 CONTINUE
2596: C
2597:          WRITE(NSYSO,461)
2598:          WRITE(NSYSO,110)
2599:          WRITE(NSYSO,111)
2600:          DO 460 MT = 1, NMT

```

src/tools/mvpdump.f

```

2601: CMOD IF(NKF5(MT).GT.0) THEN
2602: IF(IABS(NKF5(MT)).GT.0) THEN
2603: WRITE(NSYSO,112) MT,LSTF3(MT),LCTMT(MT),NEANG(MT),LSTF4(MT),
2604: + NKF5(MT),NEF5(MT),LSTF5(MT)
2605: CM WRITE(NSYSO,113) (LFF5S(J,MT),J=1,NKF5(MT)+1)
2606: CM WRITE(NSYSO,114) (NEF5S(J,MT),J=1,NKF5(MT)+1)
2607: CM WRITE(NSYSO,115) (LSTF5S(J,MT),J=1,NKF5(MT)+1)
2608: JSAVE = 1 + IABS(NKF5(MT))
2609: WRITE(NSYSO,113) (LFF5S(J,MT),J=1,JSAVE)
2610: WRITE(NSYSO,114) (NEF5S(J,MT),J=1,JSAVE)
2611: WRITE(NSYSO,115) (LSTF5S(J,MT),J=1,JSAVE)
2612: ENDIF
2613: 460 CONTINUE
2614: WRITE(NSYSO,111)
2615: C
2616: 461 FORMAT(1H1,20X,'*****
2617: @ /1H ,20X,'* #3 ENERGY DISTRIBUTION INDEX LIST *
2618: @ /1H ,20X,'*****
2619: C
2620: C
2621: C
2622: 463 CONTINUE
2623: IF(NUNR.GT.0.AND.IPRUNR.GT.0) THEN
2624: LOC1 = LSUNR
2625: LOC2 = LOC1 + NUNR
2626: LOC3 = LOC2 + NUNR
2627: LAST = NUNR
2628: KOC1 = 1
2629: KOC2 = KOC1 + NUNR*NUNR2
2630: KOC3 = KOC2 + NUNR*NUNR2*2
2631: KOC4 = KOC3 + NUNR*NUNR2
2632: KOC5 = KOC4 + NUNR*NUNR2
2633: LMAX = 6*NUNR*NUNR2
2634: CALL DIMCHK('LISTUN',LMAX,MEMORY)
2635: C
2636: CCCC
2637: call rwind(NWRK)
2638: WRITE(NWRK) (ADATA(I),I=LOC3,LOC5)
2639: REWIND NWRK
2640: CALL LISTUN(ADATA(LOC1),IDATA(LOC2),
2641: @ A(KOC1),IA(KOC2),A(KOC3),A(KOC4),
2642: @ A(KOC5),NUNR ,NUNR2 ,NSYSO ,NWRK)
2643: REWIND NWRK
2644: ENDIF
2645: C
2646: C **** PRINT OUT GAMMA DATA
2647: C
2648: IF(GDUMP.LE.0) GO TO 899
2649: IF(IGFLAG.EQ.0) GO TO 899
2650: C
2651: C ** PRINT GAMMA DATA
2652: C
2653: PHI = EHI*10.0
2654: PLOW = ELOW
2655: C
2656: ISW = LSTF5(115) - 1
2657: DO 465 I = 1 ,NMT
2658: ISW = ISW + 1
2659: LSTGAM(I) = IDATA(ISW)
2660: 465 CONTINUE
2661: C
2662: DO 466 I = 1 ,NMT
2663: ISW = ISW + 1
2664: MTOG(I) = IDATA(ISW)
2665: 466 CONTINUE

```

```

2666: C
2667: WRITE(NSYSO,471)
2668: WRITE(NSYSO,472)
2669: DO 470 MT = 1 , NMT
2670: IF(LSTGAM(MT).LE.0.AND.MTOG(MT).LE.0) GO TO 470
2671: WRITE(NSYSO,473) MT,LSTGAM(MT),NEGYLD(MT),NEF5G(MT),NKF5G(MT),
2672: + MTOG(MT),LCTMTG(MT),EG1L(MT),EG2U(MT)
2673: 470 CONTINUE
2674: WRITE(NSYSO,472)
2675: C
2676: 471 FORMAT(1H1,20X,'*****
2677: @ /1H ,20X,'* #2 RECORD INFORMATION (GAMMA DATA) *
2678: @ /1H ,20X,'*****
2679: @ ///1H ,10X,' MT LSTGAM NEGYLD NEF5G NKF5G MTOG
2680: @ 'LCTMTG EG1L(EV) EG2U(EV) '
2681: 472 FORMAT(1H , 9X,8(10H-----))
2682: 473 FORMAT(1H ,10X,I3,6I8,3X,1P2E12.5)
2683: C
2684: C *** LOOP OF REACTION
2685: C
2686: C
2687: DO 800 MT = 1 , NMT
2688: ISW = LSTGAM(MT)
2689: IF(ISW.LE.0) GO TO 800
2690: C
2691: NE = IDATA(ISW)
2692: C
2693: IF(IPRX3.GT.0) THEN
2694: WRITE(NSYSO,701) MT,QVAL(MT)
2695: LOCE = ISW + 1
2696: LOCX = ISW + NE + 1
2697: CALL LISTGX(ADATA(LOCE),ADATA(LOCX),NE,PLOW,PHI,NSYSO)
2698: ENDIF
2699: C
2700: 701 FORMAT(1H1,15X,' ## TOTAL GAMMA YIELD DATA DUMP ##
2701: * //1H ,10X,' MT = ',I3,10X,' Q-VALUE = ',1PE12.5,' EV'//)
2702: C
2703: ISW = ISW + 2*NE + 1
2704: NE = IDATA(ISW)
2705: NK = IDATA(ISW+1)
2706: CALL ICLEA( LSTF4G , MTMAX , 0 )
2707: CALL ICLEA( LSTF5G , MTMAX , 0 )
2708: CALL ICLEA( NEGANG , MTMAX , 0 )
2709: CALL CLEA( ESK , MTMAX , 0.0 )
2710: C
2711: ISW = ISW + 1
2712: DO 710 K = 1 , NK
2713: ISW = ISW + 1
2714: LSTF5G(K) = IDATA(ISW)
2715: 710 CONTINUE
2716: DO 715 K = 1 , NK
2717: ISW = ISW + 1
2718: ESK (K) = ADATA(ISW)
2719: 715 CONTINUE
2720: LOC1 = ISW + 1
2721: C
2722: *
2723: C
2724: IF(NE.GT.0.AND.IPRX3.GT.0) THEN
2725: LOC2 = LOC1 + NE
2726: LOC3 = LOC2 + NE
2727: LAST = LSTF5G(1) - 1
2728: KOC1 = 1
2729: KOC2 = KOC1 + NK*NE
2730: LMAX = NK*NE*3

```

src/tools/mvpdump.f

```

2731:      CALL DIMCHK('LISTBC',LMAX,MEMORY)
2732:      CCCCCC      REWIND NWRK
2733:      call rwind(NWRK)
2734:      WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2735:      REWIND NWRK
2736:      CALL LISTGC(ADATA(LOC1),IDATA(LOC2),
2737:      @          A(KOC1) ,IA(KOC2) ,MT ,NE ,
2738:      @          NK ,NSYSO ,NWRK ,ESK ,IPRX3 )
2739:      REWIND NWRK
2740:      ENDIF
2741: C
2742: IF(IPRX5.LE.0.AND.IPRX4.LE.0) GO TO 800
2743: C
2744: DO 790 LOP = 1 , NK
2745:   ISW      = LSTF5G(LOP)
2746:   LF       = IDATA(ISW)
2747: C
2748: WRITE(6,*) ' ** MT NK LOP ISW LF : ',MT,NK,ISW,LF
2749: IF(IPRX5.LE.0) GO TO 765
2750: C
2751: IF(LF.EQ.2) THEN
2752:   GCOEF1   = ADATA(ISW + 2)
2753:   GCOEF2   = ADATA(ISW + 3)
2754:   LSTF4G(LOP) = IDATA(ISW + 4)
2755:   NEGANG(LOP) = IDATA(ISW + 5)
2756:   WRITE(NSYSO,721) MT,LF,LOP,GCOEF1,GCOEF2
2757:   ENDIF
2758: C
2759: IF(LF.EQ.61) THEN
2760:   NE       = IDATA(ISW+1)
2761:   LOC1     = ISW + 2
2762:   LOC2    = LOC1 + NE
2763:   LOC3    = LOC2 + NE
2764:   CALL LGLF61(ADATA(LOC1),IDATA(LOC2),IDATA(LOC3),
2765:   @          ADATA(1) ,IDATA(1) ,NE ,NSYSO ,
2766:   @          MT ,PLOW ,PHI ,LAST )
2767:   LSTF4G(LOP) = IDATA( LAST + 1 )
2768:   NEGANG(LOP) = IDATA( LAST + 2 )
2769: C
2770: WRITE(6,*) ' ** LOP LF LSTF4G NEGANG : ',LOP,LF,
2771: *          LSTF4G(LOP),NEGANG(LOP)
2772: C
2773:   ENDIF
2774: C
2775: IF(LF.EQ.6) THEN
2776:   NE       = IDATA(ISW+1)
2777:   LOC1     = ISW + 2
2778:   LOC2    = LOC1 + NE
2779:   LOC3    = LOC2 + NE
2780:   IF(NBINE.GT.0) THEN
2781:     LAST   = LOC3 + 2*NE*NBINE1 - 1
2782:     CM      LSTF4G(LOP) = IDATA( LAST + 1 )
2783:     CM      NEGANG(LOP) = IDATA( LAST + 2 )
2784:     KOC1    = 1
2785:     KOC2    = KOC1 + NE*NBINE1
2786:     LMAX    = 2*NE*NBINE1
2787:     CALL DIMCHK('LISTEN',LMAX,MEMORY)
2788:     CCCCCC      REWIND NWRK
2789:     call rwind(NWRK)
2790:     WRITE(NWRK) (ADATA(I),I=LOC3,LAST)
2791:     REWIND NWRK
2792:     CALL LISLGI(ADATA(LOC1),IDATA(LOC2),
2793:     @          A(KOC1) ,A(KOC2) ,
2794:     @          NE ,NBINE1,NWRK ,NSYSO ,MT,LOP)
2795:     REWIND NWRK

```

```

2796:      ELSE
2797:      CALL LILF1B(ADATA(LOC1),IDATA(LOC2),IDATA(LOC3),
2798:      @          ADATA(1) ,IDATA(1) ,
2799:      @          NE ,NSYSO ,MT ,LOP ,LF ,PLOW ,PHI , LAST )
2800:      ENDIF
2801:      LSTF4G(LOP) = IDATA( LAST + 1 )
2802:      NEGANG(LOP) = IDATA( LAST + 2 )
2803:      ENDIF
2804: C
2805: 765 IF(NEGANG(LOP).LE.0) GO TO 785
2806: IF(IPRX4 .LE.0) GO TO 785
2807: C
2808: C *** PRINT GAMMA ANGULAR DATA
2809: C
2810:   NP       = NEGANG(LOP)
2811:   LOCX     = LSTF4G(LOP)
2812: C
2813:   WRITE(NSYSO,*) ' ** MT LOP NP LOCX ** ',MT,LOP,NP,LOCX
2814: C
2815:   LOC1     = LOCX
2816:   LOC2     = LOC1 + NP
2817:   LOC3     = LOC2 + NP
2818:   WRITE(NSYSO,731) MT,NP,LCTMTG(MT),LOP
2819:   WRITE(NSYSO,302) (ADATA(LOC1+I-1),I=1,NP)
2820:   WRITE(NSYSO,303) (IDATA(LOC2+I-1),I=1,NP)
2821:   LIMIT    = NP - 1
2822: C
2823:   DO 750 JOP= 1 , NP
2824:     EINT    = ADATA(LOC1)
2825:     IF(EINT.GE.PLOW.AND.EINT.LE.PHI)
2826:     +CALL LISTAN( EINT , ADATA(LOC3), A , NBINA1 , NSYSO )
2827:     LOC1    = LOC1 + 1
2828:     LOC2    = LOC2 + 1
2829:     LOC3    = LOC3 + NBINA1
2830:   750 CONTINUE
2831: C
2832: 785 CONTINUE
2833:   WRITE(6,*) ' ** LOP LF LSTF4G NEGANG : ',LOP,LF,
2834: *          LSTF4G(LOP),NEGANG(LOP)
2835: C
2836: 790 CONTINUE
2837: 800 CONTINUE
2838: C
2839: 721 FORMAT(/1H ,20X,' ***** ' ,
2840: *          /1H ,20X,' * GAMMA MONO-ENERGY DISTRIBUTION * ' ,
2841: *          /1H ,20X,' ***** ' ,
2842: *          ///1H ,10X,' MT = ',I3,5X,' LF = ',I2,' SUB-SECTION NO = ',I3,
2843: *          /1H ,10X,' GCOEF1 = ',1PE12.5,5X,' GCOEF2 = ',1PE12.5//)
2844: C
2845: 731 FORMAT(1H1,15X,' ## GAMMA ANGULAR DITRIBUTION DUMP ## ' ,
2846: *          /1H ,10X,' MT = ',I3,5X,' NO OF ENERGY = ',I3,
2847: *          5X,' LCT = ',I3,' (1/2:LAB/CM) ' ,
2848: *          5X,' SUB-SECTION NO = ',I3/)
2849: C
2850: C *** PRINT TEMPERATURE DEPENDENT DATA OF U.R.P.T & THERMAL DATA
2851: C
2852: 899 CONTINUE
2853: IF(ITFREE.EQ.0) RETURN
2854: C
2855: IF(TDUMP .LE.0) RETURN
2856: C
2857: IF(LSTU3R.LE.0) GO TO 999
2858: C
2859: C *** PRINT TEMP. DEPENDENT UNRESOLVED PROBABILITY DATA
2860: C

```

src/tools/mvpdump.f

```

2861: C      TMPU3R(NTU3R)
2862: C      LOC1 = LSTTU3 + 1
2863: C      EUNRTB(NURT)
2864: C      LOC2 = LOC1 + 2*NTU3R
2865: C      SIGEAV(NURT)
2866: C      LOC3 = LOC2 + NURT
2867: C      SIGFAV(NURT)
2868: C      LOC4 = LOC3 + NURT
2869: C      SIGTAV(NURT)
2870: C      LOC5 = LOC4 + NURT
2871: C      PROBUN(NBNU3R,NURT)
2872: C      LOC6 = LOC5 + NURT
2873: C      SIGETB(NTU3R,NBNU3R,NURT)
2874: C      LOC7 = LOC6 + NURT*NBNU3R
2875: C      SIGFTB(NTU3R,NBNU3R,NURT)
2876: C      LOC8 = LOC7 + NURT*NBNU3R*NTU3R
2877: C      SIGITB(NTU3R,NBNU3R,NURT)
2878: C      LOC9 = LOC8 + NURT*NBNU3R*NTU3R
2879: C
2880: CM      WRITE(NSISO,*) ' ** LOC1 TO LOC9 FOR SUBR.(LISU3R) ** '
2881: CM      WRITE(NSYSO,'(9I10)') LOC1,LOC2,LOC3,LOC4,LOC5,LOC6,LOC7,LOC8,LOC9
2882: C
2883: C      CALL LISU3R( NSYSO , NURT , NBNU3R , NTU3R , IFISUN ,
2884: C      *          ADATA(LOC1), ADATA(LOC2), ADATA(LOC3),
2885: C      *          ADATA(LOC4), ADATA(LOC5), ADATA(LOC6),
2886: C      *          ADATA(LOC7), ADATA(LOC8), ADATA(LOC9) )
2887: C
2888: C      999 CONTINUE
2889: C      IF(LSTTHS.LE.0) GO TO 1099
2890: C
2891: C *** PRINT TEMP. DEPENDENT THERMAL SCATTERING DATA
2892: C
2893: C      TMPTHS(NTTHSC)
2894: C      LOC01 = LSTTSC + 1
2895: C      EMESH3(NPTHE)
2896: C      LOC02 = LOC01 + 3*NTTHSC
2897: C      CRSTH3(NTTHSC,NPTHE)
2898: C      LOC03 = LOC02 + NPTHE
2899: C      NBTH4(NRETHE)
2900: C      LOC04 = LOC03 + NPTHE*NTTHSC
2901: C      INTTH4(NRETHE)
2902: C      LOC05 = LOC04 + NRETHE
2903: C      ANGTH4(NTTHSC,NBNTHE,NETHE)
2904: C      LOC06 = LOC05 + NRETHE
2905: C      EMESH7(NPTHIN)
2906: C      LOC07 = LOC06 + NTTHSC*NBNTHE*NETHE
2907: C      IF(NPTHE.EQ.0) LOC07 = LOC01 + 2*NTTHSC
2908: C      CRSTH7(NTTHSC,NPTHIN)
2909: C      LOC08 = LOC07 + NPTHIN
2910: C      EMESH5(NFGTH)
2911: C      LOC09 = LOC08 + NPTHIN*NTTHSC
2912: C      ETRAN7(NTTHSC,NFGTH,NIGTH)
2913: C      LOC10 = LOC09 + NFGTH
2914: C      ANGTH7(NTTHSC,5,MXNGTH)
2915: C      LOC11 = LOC10 + NTTHSC*NFGTH*NIGTH
2916: C      CRSMT2(NPTCUT)
2917: C      LOC12 = LOC11 + NTTHSC*5*MXNGTH
2918: C
2919: C      WRITE(NSYSO,*) ' ** LOC01 TO LOC12 FOR SUBR.(LISTHS) ** '
2920: C      WRITE(NSYSO,'(9I10)') LOC01,LOC02,LOC03,LOC04,LOC05,LOC06
2921: C      WRITE(NSYSO,'(9I10)') LOC07,LOC08,LOC09,LOC10,LOC11,LOC12
2922: C
2923: C      CALL LISTHS( NSYSO , NTTHSC , NPTHE , NRETHE , NETHE ,
2924: C      *          NBNTHE , NPTHIN , NIGTH , NFGTH , MXNGTH ,
2925: C      *          NPTCUT , ENERGY , NPTS , TDUMP ,

```

```

2926: C      *          ADATA(LOC01), ADATA(LOC02), ADATA(LOC03),
2927: C      *          IDATA(LOC04), IDATA(LOC05), ADATA(LOC06),
2928: C      *          ADATA(LOC07), ADATA(LOC08), ADATA(LOC09),
2929: C      *          ADATA(LOC10), ADATA(LOC11), ADATA(LOC12) )
2930: C
2931: C *** END OF PROCESS
2932: C
2933: C      1099 CONTINUE
2934: C
2935: C      RETURN
2936: C      END
2937: C
2938: C=====
2939: C
2940: C      SUBROUTINE MEND
2941: C
2942: C      INTEGER OTAPE
2943: C
2944: C      COMMON/HEADER/C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
2945: C      COMMON/UNITS/OTAPE
2946: C
2947: C      MATH = 0
2948: C      MTH = 0
2949: C      MFH = 0
2950: C      WRITE(OTAPE,10) MATH,MFH,MTH,NOSEQ
2951: C      NOSEQ = NXTSEQ(NOSEQ)
2952: C
2953: C      10 FORMAT(66X,I4,I2,I3,I5)
2954: C
2955: C      RETURN
2956: C      END
2957: C
2958: C=====
2959: C
2960: C      SUBROUTINE NORMX(X,XNORM,KSIGN,KXPLUS)
2961: C
2962: C      CONVERT FLOATING POINT NUMBER TO NORMAL FORM FOR OUTPUT.
2963: C      OUTPUT WILL BE IN THE FORM X.XXXXX+/-NN, WHICH GIVES
2964: C      AN ADDITIONAL DIGIT OF ACCURACY WHEN COMPARED TO E11.4
2965: C      FORTRAN OUTPUT. IN ADDITION THE OUTPUT WILL BE COMPUTER
2966: C      INDEPENDENT (I.E. SAME ON IBM, CDC, UNIVAC ETC.)
2967: C
2968: C      INTEGER PLUS
2969: C      C***** CHARACTER *****
2970: C      CHARACTER*4 PLUS,MINUS,KSIGN
2971: C      C***** CHARACTER *****
2972: C      C***** INTEGER *****
2973: C      INTEGER PLUS
2974: C      C***** INTEGER *****
2975: C      C-----DEFINE TWO POSSIBLE SIGNS FOR EXPONENT.
2976: C      DATA PLUS/'+' '/'
2977: C      DATA MINUS/'-' '/'
2978: C      DATA ZERO/0.0E+00/
2979: C      DATA ONE/1.0E+00/
2980: C      DATA TEN/1.0E+01/
2981: C      DATA XLOW/1.0E+00/
2982: C      DATA XHIGH/9.999995E+00/
2983: C      C-----COMPUTE SIGNED EXPONENT FROM ABSOLUTE VALUE.
2984: C      XABS=ABS(X)
2985: C      C-----IF NUMBER IS 0.0 DEFINE 0.00000+ 0
2986: C      IF(XABS) 50,50,10
2987: C      C-----DEFINE EXPONENT TO NORMALIZE MANTISSA.
2988: C      CC 10 KXREAL=ALOG10(XABS)
2989: C      10 KXREAL=LOG10(XABS)
2990: C      IF(XABS.LT.ONE) KXREAL=KXREAL-1

```

src/tools/mvpdump.f

```

2991:      SHIFT=TEN**KXREAL
2992:      XNORM=XABS/SHIFT
2993: C-----ADJUST MANTISSA FOR ROUNDING TO EXACT POWERS OF 10 DURING OUTPUT.
2994:      IF(XNORM.GE.XLOW) GO TO 20
2995:      XNORM=ONE
2996:      GO TO 30
2997: 20 IF(XNORM.LT.XHIGH) GO TO 30
2998:      KXREAL=KXREAL+1
2999:      XNORM=ONE
3000: C-----COMPUTE SIGNED MANTISSA.
3001: 30 IF(X.LT.ZERO) XNORM=-XNORM
3002: C-----DEFINE SIGN OF EXPONENT AND INSURE EXPONENT IS POSITIVE.
3003:      IF(KXREAL) 40,60,70
3004: 40 KSIGN=MINUS
3005:      KXPLUS=-KXREAL
3006:      RETURN
3007: C-----X IS 0.0. DEFINE AS 0.000000+ 0
3008: 50 XNORM=ZERO
3009: 60 KXREAL=0
3010: C-----EXPONENT IS ZERO OR POSITIVE. THEREFORE SIGN IS POSITIVE.
3011: 70 KSIGN=PLUS
3012:      KXPLUS=KXREAL
3013:      RETURN
3014:      END
3015: C
3016: C=====
3017: C
3018:      FUNCTION NXTSEQ(NOSEQ)
3019: C
3020: C  DEFINE NEXT SEQUENCE NUMBER FOR ENDF/B OUTPUT. ALLOW FOR
3021: C  MORE THAN 100000 CARDS PER EVALUATION BY RESETING NUMBER
3022: C  TO 1 EVERY TIME 100000 IS REACHED.
3023: C
3024:      NN=NOSEQ+1
3025:      IF(NN.EQ.100000) NN=1
3026:      NXTSEQ=NN
3027:      RETURN
3028:      END
3029: C
3030: C=====
3031: C
3032: CM      SUBROUTINE OUTE(E,FIELD)
3033:      SUBROUTINE OUTE(E4,FIELD)
3034: C
3035: C  FORMAT ENERGY INTO HOLLERITH FORM FOR OUTPUT.
3036: C  ENERGY MUST BE BETWEEN 1 MILLI-EV AND 100 MEV.
3037: C
3038: C***** CHARACTER *****
3039:      CHARACTER*4 BLANK,ZERO,DOT,DIGIT,FIELD
3040: C***** CHARACTER *****
3041: C***** INTEGER *****
3042: C      INTEGER BLANK,ZERO,DOT,DIGIT,FIELD
3043: C***** INTEGER *****
3044:      REAL*4 E4
3045: C***** DOUBLE *****
3046:      DOUBLE PRECISION E,HALF,TEN,TEN0,TEN1,TEN2,TEN3,TEN4,TEN5,TEN6,
3047: 1 TEN7
3048: C***** DOUBLE *****
3049:      DIMENSION FIELD(11),TEN(11),DIGIT(10)
3050:      EQUIVALENCE (TEN(1),TEN0),(TEN(2),TEN1),(TEN(3),TEN2),
3051: 1 (TEN(4),TEN3),(TEN(5),TEN4),(TEN(6),TEN5),(TEN(7),TEN6),
3052: 2 (TEN(8),TEN7)
3053:      DATA BLANK/' '/
3054:      DATA ZERO/'0'/
3055:      DATA DOT/'.'/

```

```

3056:      DATA DIGIT/'0','1','2','3','4','5','6','7','8','9'/
3057: C***** DOUBLE *****
3058:      DATA HALF/0.5D+00/
3059:      DATA TEN/1.0D+00,1.0D+01,1.0D+02,1.0D+03,1.0D+04,1.0D+05,
3060: 1 1.0D+06,1.0D+07,1.0D+08,1.0D+09,1.0D+10/
3061: C***** DOUBLE *****
3062: C***** SINGLE *****
3063: C      DATA HALF/0.5E+00/
3064: C      DATA TEN/1.0E+00,1.0E+01,1.0E+02,1.0E+03,1.0E+04,1.0E+05,
3065: C 1 1.0E+06,1.0E+07,1.0E+08,1.0E+09,1.0E+10/
3066: C***** SINGLE *****
3067: C-----FIRST COLUMN WILL ALWAYS BE BLANK.
3068: C
3069:      E      = DBLE(E4)
3070: C
3071:      FIELD(1)=BLANK
3072: C-----SELECT FULL 9 DIGIT OUTPUT (E AT LEAST 1) OR PARTIAL DIGIT OUTPUT
3073: C----- (E LESS THAN 1).
3074:      IF(E-TEN0) 10,90,20
3075: C-----PARTIAL DIGIT OUTPUT. INITIALIZE CHARACTERS AND DEFINE FIXED POINT
3076: C-----OUTPUT FIELD.
3077: 10 FIELD(2)=ZERO
3078:      FIELD(3)=DOT
3079:      FIELD(4)=ZERO
3080:      FIELD(5)=ZERO
3081:      IN=TEN(9)*E+HALF
3082:      IEXP=0
3083:      GO TO 180
3084: C-----FULL 9 DIGIT OUTPUT. PERFORM BINARY SEARCH TO DEFINE COLUMN
3085: C-----POSITION OF DECIMAL POINT.
3086: 20 IF(E-TEN4) 30,130,40
3087: 30 IF(E-TEN2) 50,110,60
3088: 40 IF(E-TEN6) 70,150,80
3089: 50 IF(E-TEN1) 90,100,100
3090: 60 IF(E-TEN3) 110,120,120
3091: 70 IF(E-TEN5) 130,140,140
3092: 80 IF(E-TEN7) 150,160,160
3093: 90 IEXP=3
3094:      GO TO 170
3095: 100 IEXP=4
3096:      GO TO 170
3097: 110 IEXP=5
3098:      GO TO 170
3099: 120 IEXP=6
3100:      GO TO 170
3101: 130 IEXP=7
3102:      GO TO 170
3103: 140 IEXP=8
3104:      GO TO 170
3105: 150 IEXP=9
3106:      GO TO 170
3107: 160 IEXP=10
3108: C-----DEFINE FIXED POINT OUTPUT FIELD AND INITIALIZE OUTPUT COLUMNS.
3109: 170 INDEX=12-IEXP
3110:      IN=(E*TEN(INDEX))+HALF
3111:      FIELD(IEXP)=DOT
3112: C-----CONVERT FIXED POINT OUTPUT FIELD TO HOLLERITH.
3113: 180 II=11
3114:      DO 190 I=1,11
3115:      IF(II.EQ.IEXP) GO TO 190
3116:      IN2=IN/10
3117:      IN3=(IN-10*IN2)+1
3118:      FIELD(II)=DIGIT(IN3)
3119:      IF(IN2.LE.0) GO TO 200
3120:      IN=IN2

```


src/tools/mvpdump.f

```

3121: 190 II=II-1
3122: 200 CONTINUE
3123: C
3124: RETURN
3125: END
3126: C
3127: C=====
3128: C
3129: SUBROUTINE POINTV(X,Y,IXY)
3130: C
3131: C WRITE IXY DATA POINTS. FORMAT OF ENERGIES WILL VARY TO ALLOW
3132: C MAXIMUM PRECISION OF BETWEEN 6 AND 9 DIGITS ACCURACY.
3133: C
3134: C CROSS SECTIONS WILL ALWAYS BE OUTPUT E11.4 FORMAT.
3135: C
3136: C ENERGIES WILL BE OUTPUT IN EITHER STANDARD E11.4 FORMAT OR A
3137: C VARIABLE F FORMAT (VARIABLE FROM F11.8 TO F11.0) TO GIVE THE
3138: C MAXIMUM NUMBER OF DIGITS OF ACCURACY. AS OUTPUT BY THIS ROUTINE
3139: C STANDARD FORM E11.4 FORMAT GIVES 6 DIGITS OF ACCURACY. THE
3140: C VARIABLE FORM F FORMAT WILL GIVE 6 TO 9 DIGITS ACCURACY. AS
3141: C LONG AS THE EXPONENT OF AN ENERGY IN E11.4 FORMAT IS BETWEEN -3
3142: C AND +8, MORE DIGITS WILL BE INCLUDED IF THE NUMBER IS OUTPUT IN
3143: C VARIABLE F FORMAT. IN PARTICULAR A FULL 9 DIGITS WILL BE OUTPUT
3144: C FOR ALL ENERGIES BETWEEN 1 EV AND 100 MEV. BETWEEN 1 MILLI-EV
3145: C AND 1 EV THE NUMBER OF DIGITS WILL VARY FROM 6 TO 8.
3146: C
3147: C INTEGER OTAPE
3148: C***** CHARACTER *****
3149: C CHARACTER*4 KSIGN,FIELD
3150: C***** CHARACTER *****
3151: C***** INTEGER *****
3152: C INTEGER FIELD
3153: C***** INTEGER *****
3154: C***** DOUBLE *****
3155: CDEL DOUBLE PRECISION X,ELOW,EHIGH
3156: C***** DOUBLE *****
3157: C COMMON/UNITS/OTAPE
3158: C COMMON/NORMF/XNORM(6),KEXP(6)
3159: C COMMON/NORMC/KSIGN(6)
3160: C COMMON/HEADER/DUMMY(6),MATH,MFH,MTH,NOSEQ
3161: C COMMON/FLAGS/MINUS3
3162: C COMMON/FIELDC/FIELD(11,3)
3163: C
3164: C DIMENSION X(IXY),Y(IXY)
3165: C-----DEFINE LOWER AND UPPER ENERGY LIMITS FOR VARIABLE F OUTPUT.
3166: C***** DOUBLE *****
3167: C DATA ELOW/1.0D-03/
3168: C DATA EHIGH/1.0D+08/
3169: C***** DOUBLE *****
3170: C***** SINGLE *****
3171: C DATA ELOW/1.0E-03/
3172: C DATA EHIGH/1.0E+08/
3173: C***** SINGLE *****
3174: C-----SET UP LOOP OVER CARDS.
3175: C DO 100 I=1,IXY,3
3176: C IP2=I+2
3177: C IF(IP2.GT.IXY) IP2=IXY
3178: C IOUT=IP2-I+1
3179: C-----SELECT VARIABLE F OR E OUTPUT FORMAT.
3180: C IF(X(I).LE.ELOW.OR.X(IP2).GE.EHIGH) GO TO 50
3181: C
3182: C OUTPUT ONE CARD WITH ENERGY IN F FORMAT AND CROSS SECTION IN E
3183: C FORMAT.
3184: C
3185: C-----CONVERT DATA TO NORMAL FORM.

```

```

3186: I3=0
3187: DO 10 II=I,IP2
3188: I3=I3+1
3189: CALL OUTE(X(II),FIELD(1,I3))
3190: C-----SET FLAG IF CROSS SECTION IS NEGATIVE.
3191: IF(Y(II).LT.0.0) MINUS3=1
3192: 10 CALL NORMX(Y(II),XNORM(I3),KSIGN(I3),KEXP(I3))
3193: C-----OUTPUT RECORD.
3194: GO TO (20,30,40),IOUT
3195: 20 WRITE(OTAPE,5040) (FIELD(J,1),J=1,11),XNORM(1),KSIGN(1),KEXP(1),
3196: 1 MATH,MFH,MTH,NOSEQ
3197: GO TO 100
3198: 30 WRITE(OTAPE,5050) ((FIELD(J,M),J=1,11),XNORM(M),KSIGN(M),KEXP(M),
3199: 1 M=1,2),MATH,MFH,MTH,NOSEQ
3200: GO TO 100
3201: 40 WRITE(OTAPE,5060) ((FIELD(J,M),J=1,11),XNORM(M),KSIGN(M),KEXP(M),
3202: 1 M=1,3),MATH,MFH,MTH,NOSEQ
3203: GO TO 100
3204: C
3205: C OUTPUT ONE CARD IN STANDARD E FORMAT.
3206: C
3207: C-----CONVERT DATA TO NORMAL FORM.
3208: 50 I3=0
3209: DO 60 II=I,IP2
3210: I3=I3+1
3211: ES=X(II)
3212: CALL NORMX(ES,XNORM(I3),KSIGN(I3),KEXP(I3))
3213: I3=I3+1
3214: C-----SET FLAG IF CROSS SECTION IS NEGATIVE.
3215: IF(Y(II).LT.0.0) MINUS3=1
3216: 60 CALL NORMX(Y(II),XNORM(I3),KSIGN(I3),KEXP(I3))
3217: C-----OUTPUT RECORD.
3218: GO TO (70,80,90),IOUT
3219: 70 WRITE(OTAPE,5010) (XNORM(II),KSIGN(II),KEXP(II),II=1,2),
3220: 1 MATH,MFH,MTH,NOSEQ
3221: GO TO 100
3222: 80 WRITE(OTAPE,5020) (XNORM(II),KSIGN(II),KEXP(II),II=1,4),
3223: 1 MATH,MFH,MTH,NOSEQ
3224: GO TO 100
3225: 90 WRITE(OTAPE,5030) (XNORM(II),KSIGN(II),KEXP(II),II=1,6),
3226: 1 MATH,MFH,MTH,NOSEQ
3227: 100 NOSEQ=NXTSEQ(NOSEQ)
3228: RETURN
3229: C
3230: 5010 FORMAT(2(F8.5,A1,I2),44X,I4,I2,I3,I5)
3231: 5020 FORMAT(4(F8.5,A1,I2),22X,I4,I2,I3,I5)
3232: 5030 FORMAT(6(F8.5,A1,I2),I4,I2,I3,I5)
3233: 5040 FORMAT(11A1,F8.5,A1,I2,44X,I4,I2,I3,I5)
3234: 5050 FORMAT(2(11A1,F8.5,A1,I2),22X,I4,I2,I3,I5)
3235: 5060 FORMAT(3(11A1,F8.5,A1,I2),I4,I2,I3,I5)
3236: END
3237: C
3238: C=====
3239: C
3240: C SUBROUTINE SENDO
3241: C
3242: C OUTPUT SEND, FEND, MEND OR TEND CARD IN STANDARD FORMAT.
3243: C
3244: C INTEGER OTAPE
3245: C COMMON/UNITS/OTAPE
3246: C COMMON/HEADER/C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
3247: C
3248: C IF(OTAPE.LE.0) RETURN
3249: C WRITE(OTAPE,5030) MATH,MFH,NOSEQ
3250: C NOSEQ=NXTSEQ(NOSEQ)

```

src/tools/mvpdump.f

```

3251: C
3252: 5030 FORMAT(66X,I4,I2,3H 0,I5)
3253: C
3254: RETURN
3255: END
3256: C
3257: C=====
3258: C
3259: SUBROUTINE TEND
3260: C
3261: INTEGER OTAPE
3262: C
3263: COMMON/HEADER/C1H,C2H,L1H,L2H,N1H,N2H,MATH,MFH,MTH,NOSEQ
3264: COMMON/UNITS /OTAPE
3265: C
3266: NOSEQ = 1
3267: MATH = -1
3268: MTH = 0
3269: MFH = 0
3270: WRITE(OTAPE,10) MATH,MFH,MTH,NOSEQ
3271: C
3272: 10 FORMAT(66X,I4,I2,I3,I5)
3273: C
3274: RETURN
3275: END
3276: C
3277: C=====
3278: C
3279: SUBROUTINE TERPO(NBT,INT,N1)
3280: C
3281: C WRITE TAB1 INTERPOLATION LAW.
3282: C
3283: INTEGER OTAPE
3284: COMMON/UNITS/OTAPE
3285: COMMON/HEADER/DUMMY(6),MATH,MFH,MTH,NOSEQ
3286: C
3287: DIMENSION NBT(N1),INT(N1)
3288: C
3289: IF(OTAPE.LE.0) RETURN
3290: C
3291: DO 40 I=1,N1,3
3292: IP2=I+2
3293: IF(IP2.GT.N1) IP2=N1
3294: IOUT=IP2-I+1
3295: GO TO (10,20,30),IOUT
3296: 10 WRITE(OTAPE,5020) NBT(I),INT(I),MATH,MFH,MTH,NOSEQ
3297: GO TO 40
3298: 20 WRITE(OTAPE,5010) (NBT(II),INT(II),II=I,IP2),MATH,MFH,MTH,NOSEQ
3299: GO TO 40
3300: 30 WRITE(OTAPE,5000) (NBT(II),INT(II),II=I,IP2),MATH,MFH,MTH,NOSEQ
3301: 40 NOSEQ=NXTSEQ(NOSEQ)
3302: RETURN
3303: C
3304: 5000 FORMAT(6I11,I4,I2,I3,I5)
3305: 5010 FORMAT(4I11,22X,I4,I2,I3,I5)
3306: 5020 FORMAT(2I11,44X,I4,I2,I3,I5)
3307: END
3308: C
3309: C=====
3310: C
3311: SUBROUTINE WOT(X,NCOL,LTBL,LG,N6,TOP1,TOP2,TOP3,TITLE )
3312: C*****
3313: C OUTPUT WRITES 1,2, OR 3-D ARRAYS
3314: C*****
3315: C X(LTBL,NCOL,LG) : 1,2 OR 3-D ARRARY

```

```

3316: C TOP1 : COMMENT FOR FIRST SUFFIX OF X ARRAY (A4)
3317: C TOP2 : COMMENT FOR SECOND SUFFIX OF X ARRAY (A4)
3318: C TOP3 : COMMENT FOR THIRD SUFFIX OF X ARRAY (A4)
3319: C*****
3320: DIMENSION X(LTBL,NCOL,1 )
3321: CHARACTER*4 TOP1,TOP2,TOP3,BLK
3322: CHARACTER*20 TITLE
3323: C
3324: DATA BLK / 4H /
3325: C
3326: IF(N6.LE. 0) RETURN
3327: IF(N6.GT.99) RETURN
3328: IF(N6.EQ. 5) RETURN
3329: C
3330: WRITE(N6,7) TITLE
3331: 7 FORMAT(1H1,20X,'*****',
3332: + /1H ,20X,'* ',A20,' *',
3333: + /1H ,20X,'*****'/)
3334: C
3335: DO 170 L=1,LG
3336: I02=0
3337: I03=(NCOL+9)/10
3338: IF (LG.GT.1) WRITE(N6,10000) TOP3,L
3339: DO 160 I=1,I03
3340: I01=I02+1
3341: I02=MIN0(I01+9,NCOL)
3342: WRITE(N6,10200) TOP1,(TOP2,J,J=I01,I02)
3343: DO 150 K=1,LTBL
3344: IF (K.EQ.1) GO TO 140
3345: DO 100 J=I01,I02
3346: 100 IF (X(K,J,L).NE.X(K-1,J,L)) GO TO 130
3347: IF (K.EQ.KS) GO TO 110
3348: KE=K
3349: IF (K.EQ.LTBL) GO TO 130
3350: GO TO 150
3351: 110 IF (K.EQ.LTBL) GO TO 140
3352: DO 120 J=I01,I02
3353: 120 IF (X(K,J,L).NE.X(K+1,J,L)) GO TO 140
3354: KE=KE+1
3355: GO TO 150
3356: 130 IF (K.EQ.KS) GO TO 140
3357: WRITE(N6,10100) TOP1,KS,TOP1,KE
3358: IF (K.EQ.LTBL.AND.KE.EQ.K) GO TO 150
3359: C 140 WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3360: 140 CONTINUE
3361: IF(LG.EQ.1.AND.K.EQ.LTBL) THEN
3362: IF(TOP3.EQ.BLK) THEN
3363: WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3364: ELSE
3365: WRITE(N6,10400) TOP3,(X(K,J,L),J=I01,I02)
3366: ENDIF
3367: ELSE
3368: WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3369: ENDIF
3370: C
3371: KS=K+1
3372: KE=KS
3373: 150 CONTINUE
3374: 160 CONTINUE
3375: 170 CONTINUE
3376: RETURN
3377: C
3378: 10000 FORMAT(/1H0,2X,A4,I5)
3379: 10100 FORMAT(8X,A4,I5,6H THRU ,A4,I5,14H SAME AS ABOVE)
3380: 10200 FORMAT(/2X,A4,2X,A4,I3,9(5X,A4,I3))

```

src/tools/mvpdump.f

```
3381: 10300 FORMAT(I6,1P10E12.5)
3382: 10400 FORMAT(2H  ,A4,1P10E12.5)
3383: C
3384:      END
3385: C
3386: C=====
3387: C
3388:      SUBROUTINE WOT6(X,NCOL,LTBL,LG,TOP1,TOP2,TOP3,IPRINT)
3389: C*****
3390: C      OUTPUT WRITES 1,2, OR 3-D ARRAYS
3391: C*****
3392: C      X(LTBL,NCOL,LG) : 1,2 OR 3-D ARRAY
3393: C      TOP1             : COMMENT FOR FIRST SUFFIX OF X ARRAY (A4)
3394: C      TOP2             : COMMENT FOR SECOND SUFFIX OF X ARRAY (A4)
3395: C      TOP3             : COMMENT FOR THIRD SUFFIX OF X ARRAY (A4)
3396: C*****
3397: C
3398: C      DIMENSION      X(LTBL,NCOL,1 )
3399: C      CHARACTER*4     TOP1,TOP2,TOP3,BLK
3400: C
3401: C      DATA          BLK / 4H      /
3402: C
3403: C      N6=IPRINT
3404: C      DO 170 L=1,LG
3405: C          I02=0
3406: C          I03=(NCOL+9)/10
3407: C          IF (LG.GT.1) WRITE(N6,10000) TOP3,L
3408: C          DO 160 I=1,I03
3409: C              I01=I02+1
3410: C              I02=MIN0(I01+9,NCOL)
3411: C              WRITE(N6,10200) TOP1,(TOP2,J,J=I01,I02)
3412: C              DO 150 K=1,LTBL
3413: C                  IF (K.EQ.1) GO TO 140
3414: C                  DO 100 J=I01,I02
3415: C                      IF (X(K,J,L).NE.X(K-1,J,L)) GO TO 130
3416: C                      IF (KE.EQ.KS) GO TO 110
3417: C                      KE=K
3418: C                      IF (K.EQ.LTBL) GO TO 130
3419: C                      GO TO 150
3420: C                      IF (K.EQ.LTBL) GO TO 140
3421: C                      DO 120 J=I01,I02
3422: C                          IF (X(K,J,L).NE.X(K+1,J,L)) GO TO 140
3423: C                          KE=KE+1
3424: C                          GO TO 150
3425: C                          IF (KE.EQ.KS) GO TO 140
3426: C                          WRITE(N6,10100) TOP1,KS,TOP1,KE
3427: C                          IF (K.EQ.LTBL.AND.KE.EQ.K) GO TO 150
3428: C                          WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3429: C                          CONTINUE
3430: C                          IF (LG.EQ.1.AND.K.EQ.LTBL) THEN
3431: C                              IF (TOP3.EQ.BLK) THEN
3432: C                                  WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3433: C                                  ELSE
3434: C                                  WRITE(N6,10400) TOP3,(X(K,J,L),J=I01,I02)
3435: C                                  ENDIF
3436: C                              ELSE
3437: C                                  WRITE(N6,10300) K,(X(K,J,L),J=I01,I02)
3438: C                                  ENDIF
3439: C
3440: C                      KS=K+1
3441: C                      KE=KS
3442: C                      150 CONTINUE
3443: C                      160 CONTINUE
3444: C                      170 CONTINUE
3445: C                      RETURN
```

```
3446: C
3447: 10000 FORMAT(/1H0,2X,A4,I5)
3448: 10100 FORMAT(8X,A4,I5,6H THRU ,A4,I5,14H SAME AS ABOVE)
3449: 10200 FORMAT(/2X,A4,2X,A4,I3,9(5X,A4,I3))
3450: 10300 FORMAT(I6,1P10E12.5)
3451: 10400 FORMAT(2H  ,A4,1P10E12.5)
3452: C
3453:      END
```

src/tools/myfsplit.f

```

1:      program SPLIT
2:      C
3:      C =====
4:      C   Routine splitter for FORTRAN source file
5:      C
6:      C * Made by M.Sasaki (by modification of PSTOPO program by himself):
7:      C
8:      C -----
9:      C   Motivated by intolerable awkwardness of the 'fsplit' of HP-UX !!!
10:     C -----
11:     C
12:     C $Id: myfsplit.f,v 1.8 1999/10/07 08:41:05 sasaki Exp sasaki $
13:     C
14:     C =====
15:     C
16:     C Rules & Comments:
17:     C
18:     C
19:     C   * Splitted into current directory if '-d directory' option
20:     C     is not specified.
21:     C   * Splitted routines are named as NAME.f.
22:     C     NAME is upper case by default. (-i : option for names
23:     C     in lower case )
24:     C   * Block data routine are named as BLOCKn.f as default,
25:     C     where 'n' is sequence number of appearance of block data's.
26:     C     ( -b blk : options to change default BLOCK prefix to blk.)
27:     C   * Output message on the top of splitted files.
28:     C     (Original file, date & time ).
29:     C     Can be suppressed by -n option.
30:     C
31:     C Restrictions :
32:     C
33:     C   * Executable on HP-UX or other sysytems having command line
34:     C     parameter passing methods to FORTRAN program (by minor
35:     C     modification, I hope ).
36:     C   * SUBROUTINE, FUNCTION or BLOCKDATA symbol must be in a line.
37:     C     Subprogram name must be in the same line as SUBROUTINE
38:     C     etc. (continuations for arguments are possible)
39:     C   * Number of lines contained in a routine must be less than
40:     C     MAXL (parameter).
41:     C
42:     C Bug :
43:     C
44:     C   * Why this program is written by Fortran .....
45:     C
46:     C -----
47:     C < History >
48:     C
49:     C First version : 25 June 1992
50:     C
51:     C Second version : 10 January 1993
52:     C   *treat cases in which program name is written in a line
53:     C   other than SUBROUTINE,FUNCTION or BLOCKDATA line.
54:     C
55:     C Third version : 14,25 November 1997
56:     C
57:     C   Make default of splitted filename to lower case.
58:     C   Do not add split message to files ("-n" is changed).
59:     C
60:     C -----
61:     C
62:     C parameter( LLINE      = 80 )
63:     C parameter( MLEN = 16 )
64:     C character IDSN*256, ODSN*256, XLINE*(LLINE)
65:     C character*16 MEMBER, TEMPC

```

```

66:     C
67:     C maxl : line number maximum allowed for a sub-program
68:     C maxmem : subprogram number maximum
69:     C
70:     C   parameter( MAXL = 5000, MAXMEM = 600 )
71:     C
72:     C   common /BUFFER/ LINE(MAXL),      MLIST(MAXMEM)
73:     C   character*(LLINE) LINE
74:     C   character*(MLEN) MLIST
75:     C
76:     C
77:     C
78:     C   character*64 BLOCK
79:     C   character*128 DIR
80:     C   character*256 STRNG
81:     C   character*12 DT, TM
82:     C
83:     C   data IUNT /5/
84:     C   data IDSN //'stdin'/
85:     C
86:     C   data BLOCK //'BLOCK'/
87:     C   data DIR //' '
88:     C
89:     C   .... Routine names in upper case if upcs = 1.
90:     C
91:     C 1/0 :output name in upper/lower case
92:     C
93:     C   data UPCS /0/
94:     C   data JMESSG /1/
95:     C   data JMESSG /0/
96:     C
97:     C cut trailing blanks
98:     C
99:     C   data JCUT /0/
100:    C   data JCUT /1/
101:    C
102:    C cut after 72'th column
103:    C
104:    C   data JCUTNM /0/
105:    C
106:    C overwrite existing file.
107:    C
108:    C   data JOVW /1/
109:    C
110:    C   write(6,*) '==== FORTRAN source file splitter ===='
111:    C   write(6,*) '==== Version 2.1 6 June 1995 ===='
112:    C   write(6,*) '==== Version 3.1 25 November 1997 ===='
113:    C   write(6,*) ' myfsplit [-d dir] [-b blk] [-U] [-n] [-s] [-t]',
114:    C   & ' [filename]'
115:    C   write(6,*) ' '
116:    C   write(6,*) '      dir: output directory.'
117:    C   write(6,*) '      blk: prefix to unnamed block data routine.'
118:    C   write(6,*) '      ( file name is blk + appearance # .f )'
119:    C   write(6,*) '      -U : output file names in uppercase.'
120:    C   write(6,*) '      (lowercase by default)'
121:    C   write(6,*) '      -n : add output of splitting message'
122:    C   write(6,*) '      on output files.'
123:    C   write(6,*) '      -s : do not cut unnecessary trailing blanks.'
124:    C   write(6,*) '      -t : cut anything after 73'th column.'
125:    C   write(6,*) '      -k : does not destroy existing file.'
126:    C   write(6,*) '      (similar treatment as duplicate name)'
127:    C   write(6,*) ' '
128:    C   write(6,*) '      filename: input file. (stdin if omitted)'
129:    C   write(6,*) '=====
130:    C

```

src/tools/myfsplit.f

```

131:      KKJ      = IARGC()
132: C/#IF SYSTEM(HP800 HIOSF)
133: *      kkj = kkj - 1
134: C/#ENDIF
135:      IOS      = 0
136: C
137:      if ( KKJ.eq.0 ) then
138:          write(6,*) ' *** no parameters given ! ***'
139:          go to 120
140:      end if
141: C
142:      NC      = 0
143: C
144:      100 NC      = NC + 1
145:      if ( NC.gt.KKJ ) go to 120
146: C
147: C/#IF SYSTEM(HP700)
148: *      LL      = IGETARG(NC,STRNG,LEN(STRNG))
149: C/#ELSE
150: C/#IF SYSTEM(HP800 HIOSF*)
151: *      call getarg(nc+1, strng)
152: C/#ELSE
153:      call getarg(nc, strng)
154: C/#ENDIF
155:      LL = MIN( INDEX(STRNG,' ') - 1, LEN(STRNG))
156:      IF( LL.EQ.0 ) LL = LEN(STRNG)
157: C/#ENDIF
158: Check %%
159: C      write(6,*) ' nc ',nc, ' <',strng(:ll),' >'
160: C
161: C      .... file name ....
162: C
163:      if ( STRNG(1:1).ne.'-' ) then
164:          IUNT      = 20
165:          IDSN      = STRNG(:LL)
166:          open( IUNT, file =IDSN(:LL), status = 'OLD', form = 'FORMATTED',
167:          &      iostat =IOS )
168:          if ( IOS.ne.0 ) write(6,*) ' Input file open error <',
169:          &      STRNG(:LL), ' >', ' IOSTAT=', IOS
170: C
171: C      .... options ....
172: C
173:      else
174:          if ( STRNG(2:2).eq.'d' ) then
175:              LP      = 3
176:              if ( LL.le.2 ) then
177:                  NC      = NC + 1
178: C/#IF SYSTEM(HP700)
179: *                  LL      = IGETARG(NC,STRNG,LEN(STRNG))
180: C/#ELSE
181: C/#IF SYSTEM(HP800 HIOSF*)
182: *                  call getarg(nc+1, strng)
183: C/#ELSE
184:                  call getarg(nc, strng)
185: C/#ENDIF
186:                  LL = MIN( INDEX(STRNG,' ') - 1, LEN(STRNG))
187:                  IF( LL.EQ.0 ) LL = LEN(STRNG)
188: C/#ENDIF
189:                  LP      = 1
190:          end if
191:          DIR      = STRNG(LP:LL)
192:          LDIR      = INDEX(DIR//' ',' ') - 1
193:          if ( DIR(LDIR:LDIR).ne.'/' ) then
194:              LDIR      = LDIR + 1
195:              DIR(LDIR:LDIR) = '/'
196:          end if
197:          else if ( STRNG(2:2).eq.'b' ) then
198:              LP      = 3
199:              if ( LL.le.2 ) then
200:                  NC      = NC + 1
201: C/#IF SYSTEM(HP700)
202: *                  LL      = IGETARG(NC,STRNG,LEN(STRNG))
203: C/#ELSE
204: C/#IF SYSTEM(HP800 HIOSF*)
205: *                  call getarg(nc+1, strng)
206: C/#ELSE
207:                  call getarg(nc, strng)
208: C/#ENDIF
209:                  LL = MIN( INDEX(STRNG,' ') - 1, LEN(STRNG))
210:                  IF( LL.EQ.0 ) LL = LEN(STRNG)
211: C/#ENDIF
212:                  LP      = 1
213:          end if
214:          BLOCK      = STRNG(LP:LL)
215:          else if ( INDEX('Unskt',STRNG(2:2)).ne.0 ) then
216:              do 110 K = 2, LL
217:                  if ( STRNG(K:K).eq.'U' ) then
218:                      UPCS      = 1
219:                  else if ( STRNG(K:K).eq.'n' ) then
220:                      JMESSG      = 0
221:                      JMESSG      = 1
222:                  else if ( STRNG(K:K).eq.'s' ) then
223:                      JCUT      = 0
224:                  else if ( STRNG(K:K).eq.'t' ) then
225:                      JCUTNM      = 1
226:                  else if ( STRNG(K:K).eq.'k' ) then
227:                      JOVW      = 0
228:                  end if
229:              110 continue
230:          end if
231:          end if
232:          go to 100
233: C
234:      120 continue
235: C
236:      if ( IOS.ne.0 ) stop 1
237: C
238:      if ( IUNT.eq.5 ) write(6,*) ' *** From standard input ',
239:      &      ' ( CNTL-D to stop) ** '
240: C
241:      LBLK      = INDEX(BLOCK//' ',' ') - 1
242:      LIDSN      = INDEX(IDSN//' ',' ') - 1
243: C
244:      NL      = 0
245:      NBLKD      = 0
246:      ISTOP      = 0
247:      NMEM      = 0
248:      NRT      = 0
249:      NGET      = 0
250: C
251: C      .... Start New routine ....
252: C
253:      130 NLM      = 0
254:      MEMBER      = ' '
255: C
256: C      .... Check one line ....
257: C
258:      140 call GETLIN( IUNT, LINE, NL, NLM, MAXL, ISTOP )
259: C
260:      if ( ISTOP.ne.0 ) go to 200

```

src/tools/myfsplit.f

```

261: C
262:   if ( NLM.gt.MAXL ) then
263:     write(6,*) '== Too long routine (< ', MAXL, ' lines)'
264:     stop 4
265:   end if
266: C
267: C .... comment & continuation line ....
268: C
269:   if ( LINE(NLM)(1:1).eq.'C'
270: & .or. LINE(NLM)(1:1).eq.'c'
271: & .or. LINE(NLM)(1:1).eq.'*' ) go to 140
272:   if ( LINE(NLM)(1:5).eq.' ' .and.LINE(NLM)(6:6).ne.' ' ) go to 140
273: C
274: C
275: C
276: C ... remove blank ...
277: C
278:   call COMP( LINE(NLM), XLINE, 72, LL )
279: C
280: C .... to upper case ....
281: C
282:   call UPPER( XLINE(1:LL) )
283: C
284:   if ( INDEX(XLINE(1:LL),'PROGRAM').eq.1 ) then
285:     call SEARCH( XLINE, LEN('PROGRAM')+1, MEMBER, IUNT, LINE, NL,
286: & NLM, MAXL, ISTOP )
287: C
288:   else if ( XLINE(1:10).eq.'SUBROUTINE' ) then
289:     call SEARCH( XLINE, LEN('SUBROUTINE')+1, MEMBER, IUNT, LINE,
290: & NL, NLM, MAXL, ISTOP )
291: Check %%%
292: C   write(6,*) 'SUBROUTINE ',member
293: C
294: C
295:   else if ( INDEX(XLINE(1:LL),'FUNCTION').ne.0 ) then
296:     KK = INDEX(XLINE(1:LL),'FUNCTION')
297:     if ( KK.gt.1 ) then
298:       LK = MAX( INDEX(XLINE(:KK-1),'REAL'),
299: & INDEX(XLINE(:KK-1),'INTEGER'),
300: & INDEX(XLINE(:KK-1),'CHARACTER'),
301: & INDEX(XLINE(:KK-1),'LOGICAL'),
302: & INDEX(XLINE(:KK-1),'COMPLEX'),
303: & INDEX(XLINE(:KK-1),'SINGLEPRECISION'),
304: & INDEX(XLINE(:KK-1),'DOUBLEPRECISION'))
305:     if ( LK.ne.1 ) go to 140
306:   end if
307:   call SEARCH( XLINE, KK+LEN('FUNCTION'), MEMBER, IUNT, LINE, NL,
308: & NLM, MAXL, ISTOP )
309: C
310: C
311:   else if ( XLINE(1:9).eq.'BLOCKDATA' ) then
312:     call SEARCH( XLINE, LEN('BLOCKDATA')+1, MEMBER, IUNT, LINE, NL,
313: & NLM, MAXL, ISTOP )
314:     if ( MEMBER.eq.' ' ) then
315:       NBLKD = NBLKD + 1
316:       write(TEMPC,'(a,i5)') BLOCK(:LBLK), NBLKD
317:       call COMP( TEMPC, MEMBER, LEN(TEMPC), LL )
318:       if ( LL.lt.LEN(MEMBER) ) MEMBER(LL+1:) = ' '
319:     end if
320: C
321:   else if ( LL.eq.3.and.XLINE(1:3).eq.'END' ) then
322:     go to 150
323:   end if
324:   go to 140
325: C

```

```

326: C
327: C .... END of a routine ....
328: C
329:   150 continue
330:     NRT = NRT + 1
331: Check %%%
332: C   write(6,*) '<',member,'>', nlm, nl
333:   if ( MEMBER.eq.' ' ) then
334:     MEMBER = 'MAIN'
335:   end if
336: C
337:   LM = INDEX(MEMBER// ' ', ' ') - 1
338: C
339: C --- filename : upper case / lower case
340: C
341:   if ( UPCS.ne.0 ) then
342:     call UPPER( MEMBER(1:LM) )
343:   else
344:     call LOWER( MEMBER(1:LM) )
345:   end if
346: C
347: Check %%%
348:   write(6,*) '<', MEMBER, '>'
349: C
350:   call CHKMEN( MEMBER, MLIST, MAXMEM, NMEN, LM, ODSN, JOVW, DIR,
351: & LDIR )
352: Check %%%
353:   write(6,*) '<', MEMBER, '>'
354: C
355:   LF = INDEX(ODSN// ' ', ' ') - 1
356:   write(6,*) 'File ', NRT, ' <<', ODSN(:LF), '>> ( ', NLM,
357: & ' lines ) '
358: C
359: C
360: C
361:   open( 10, file =ODSN(:LF), status ='UNKNOWN', access =
362: & 'SEQUENTIAL', form ='FORMATTED', iostat =IOS )
363:   if ( IOS.ne.0 ) then
364:     write(6,*) ' file <', ODSN(:LF), '> open error. iostat=', IOS
365:     stop 2
366:   end if
367: C
368: C .... message ...
369: C
370:   if ( JMESSG.ne.0 ) then
371: C/#IF SYSTEM(HP700)
372: *   call TIME( TM )
373: *   call DATE( DT )
374: C/#ELSE
375:     tm = ' '
376:     dt = ' '
377: C/#ENDIF
378:   write(10,('C== FSPLIT: ORIGINAL FILE: ',a)) IDSN(:LIDSN)
379:   write(10,('C== DATE: ',a,' TIME: ',a)) DT(1:9),
380: & TM(1:8)
381:   end if
382: C
383: C
384:   if ( JCUTNM.eq.1 ) then
385:     do 192 K = 1, NLM
386:       LINE(K)(73:) = ' '
387:   192 continue
388:   endif
389: C
390:   if ( JCUT.eq.1 ) then

```

src/tools/myfsplit.f

```

391:      do 190 K = 1, NLM
392: ccccc      do 170 IK = 72, 1, -1
393:            do 170 IK = len(LINE(1)), 1, -1
394:              if ( LINE(K)(IK:IK).ne.' ' ) then
395:                go to 180
396:              end if
397: 170        continue
398: 180        continue
399:            if ( IK.gt.0 ) then
400:              write(10,'(a)') LINE(K) (:IK)
401:            else
402:              write(10,'()')
403:            end if
404: 190        continue
405:      else
406:        write(10,'(a)') (LINE(K),K=1,NLM)
407:      end if
408: C
409:      close( 10, iostat =IOS, status = 'KEEP' )
410: C
411:      go to 130
412: 200 write(6,*) ' === ', NL, ' lines splitted into ', NRT, ' files.'
413: 210 stop
414:      end
415: C
416: C-----
417: C
418: C Remove blanks from LINE(1:NN) if any.
419: C Return compressed length as L.
420: C
421:      subroutine COMP( LINE, CLINE, NN, L )
422:      character LINE*(*), CLINE*(*)
423:      L = 0
424:      CLINE = ' '
425:      do 100 I = 1, NN
426:        if ( LINE(I:I).eq.' ' ) go to 100
427:        L = L + 1
428:        CLINE(L:L) = LINE(I:I)
429: 100 continue
430:      return
431:      end
432: C
433: C-----
434: C
435: C Search program name from XLINE(IS:IS)
436: C Search succeeding continuation lines if necessary.
437: C
438:      subroutine SEARCH( XLINE, IS, MEMBER,IUNT, LINE, NL, NLM,
439:      & MAXL, ISTOP )
440:      parameter( LLINE = 80 )
441:      parameter( MLINE = 72 )
442:      character XLINE*(*)
443:      character LINE(MAXL)*(LLINE), MEMBER*(*)
444:      M = 0
445:      MEMBER = ' '
446:      IE = INDEX(XLINE,'(') - 1
447:      if ( IE.lt.0 ) IE = LEN(XLINE)
448:      do 100 I = IS, IE
449:        if ( XLINE(I:I).ne.' ' ) then
450:          M = M + 1
451:          MEMBER(M:M) = XLINE(I:I)
452:        end if
453: 100 continue
454:      if ( M.gt.0 ) go to 120
455: C

```

```

456: C ... program name may be in the next line ....
457: C
458: 110 call GETLIN( IUNT, LINE, NL, NLM, MAXL, ISTOP )
459:      if ( ISTOP.ne.0 ) return
460: C
461:      if ( LINE(NLM)(6:6).eq.' ' ) return
462: C
463:      if ( LINE(NLM)(7:MLINE).ne.' ' ) then
464:        call COMP( LINE(NLM)(7:MLINE), XLINE,
465:      &          LEN(LINE(NLM)(:MLINE))-6, LL )
466:        IE = INDEX(XLINE,'(')
467:        if ( IE.gt.0 ) then
468:          MEMBER = XLINE(:IE-1)
469:        else
470:          MEMBER = XLINE
471:        end if
472:        go to 120
473:      end if
474:      go to 110
475: C
476: C ... for REAL FUNCTION xxx*8(...) etc.
477: C
478: 120 continue
479: C
480:      do 130 L = 1, LEN(MEMBER)
481:        if ( MEMBER(L:L).eq.'*' ) then
482:          MEMBER(L:) = ' '
483:          return
484:        end if
485: 130 continue
486:      return
487:      end
488: C
489: C-----
490: C
491: C Get a line from input file
492: C
493:      subroutine GETLIN( IUNT, LINE, NL, NLM, MAXL, ISTOP )
494:      parameter( LLINE = 80 )
495:      character*(LLINE) LINE(MAXL)
496: C
497:      data IEND /0/
498: C
499:      if ( IEND.eq.1 ) go to 100
500:      ISTOP = 0
501: C
502:      if ( NLM.ge.MAXL ) then
503:        write(6,*) 'xxx Too many lines in a routine (max=', MAXL, ') '
504:        stop 99
505:      end if
506: C
507:      read(IUNT,fmt ='(a)',end =100) LINE(NLM+1)
508:      NLM = NLM + 1
509:      NL = NL + 1
510:      return
511: C
512: 100 ISTOP = 1
513:      IEND = 1
514:      return
515:      end
516: C
517: C-----
518: C
519: C Compose output filename from subnm(1:lm) & dir(1:ldir)
520: C

```

src/tools/myfsplit.f

```

521: C      Return odsn(:lf) as output filename
522: C
523:       subroutine OFILEN( ODSN, SUBNM, LM, LF, DIR, LDIR )
524: C
525:       character*(*) ODSN, DIR, SUBNM
526:       ODSN = ' '
527:       if ( DIR.eq.' ' ) then
528:         ODSN = SUBNM(:LM) //'.'f'
529:       else
530:         ODSN = DIR(:LDIR) //SUBNM(:LM) //'.'f'
531:       end if
532:       LF = INDEX(ODSN,' ') - 1
533:       if ( LF.lt.0 ) LF = LEN(ODSN)
534:       return
535:     end
536: C
537: C-----
538: C
539: C Check duplicate name & assign alternate name if necessary
540: C
541:       subroutine CHKMEN( MEMBER,MLIST, MAXMEM,NMEM, LM, ODSN, JOVW,
542: & DIR, LDIR )
543:       parameter( MLEN = 16 )
544:       character*(MLEN) MEMBER, MLIST(MAXMEM)
545:       character*(*) ODSN, DIR
546:       logical EXIST
547: C
548:       character TNAME*64, SYM*34
549:       data SYM /'23456789ABCDEFGHIJKLMNPOQRSTUVWXYZ'/
550: C
551: C
552: C .... check file existence ....
553: C
554:       call OFILEN( ODSN, MEMBER, LM, LF, DIR, LDIR )
555: C
556:       if ( JOVW.eq.0 ) then
557:         inquire(file =ODSN(:LF),exist =EXIST)
558:         if ( EXIST ) then
559:           write(6,*) '!! file ', ODSN(:LF), ' exist '
560:           go to 110
561:         end if
562:       end if
563: C
564:       do 100 I = 1, NMEM
565:         if ( MEMBER.eq.MLIST(I) ) go to 110
566:       100 continue
567:       NMEM = NMEM + 1
568:       if ( NMEM.gt.MAXMEM ) then
569:         write(6,*) '=== Too many routines !! (max=', MAXMEM, ') '
570:         stop 8
571:       end if
572:       MLIST(NMEM) = MEMBER
573:       return
574: C
575: C
576: C
577:       110 if ( LM.lt.LEN(MEMBER)-2 ) then
578:         do 130 I = 1, LEN(SYM)
579:           TNAME = MEMBER(1:LM) //'_'//SYM(I:I)
580:           do 120 J = 1, NMEM
581:             if ( TNAME.eq.MLIST(J) ) go to 130
582:           120 continue
583:
584:           if ( JOVW.eq.0 ) then
585:             call OFILEN( ODSN, TNAME, LM+2, LF, DIR, LDIR )

```

```

586:             inquire(file =ODSN(:LF),exist =EXIST)
587:             if ( EXIST ) then
588:               write(6,*) '!! file ', ODSN(:LF), ' exist '
589:               go to 130
590:             end if
591:           end if
592:
593:           go to 170
594:         130 continue
595:       end if
596: C
597:       do 160 I = 2, LM
598:         do 150 J = 1, LEN(SYM)
599:           if ( I.lt.LM ) TNAME = MEMBER(1:I-1) //SYM(J:J) //
600:             & MEMBER(I+1:LM)
601:           if ( I.eq.LM ) TNAME = MEMBER(1:I-1) //SYM(J:J)
602:           do 140 K = 1, NMEM
603:             if ( TNAME.eq.MLIST(K) ) go to 150
604:           140 continue
605:
606:           if ( JOVW.eq.0 ) then
607:             call OFILEN( ODSN, TNAME, LM+2, LF, DIR, LDIR )
608:             inquire(file =ODSN(:LF),exist =EXIST)
609:             if ( EXIST ) then
610:               write(6,*) '!! file ', ODSN, ' exist '
611:               go to 150
612:             end if
613:           end if
614:           go to 170
615:         150 continue
616:       160 continue
617: C
618:       write(6,*) MEMBER, ' --- Failed to make a unique file name.'
619:       stop
620: C
621:       170 LM1 = INDEX(TNAME//' ',' ') - 1
622:       write(6,*) 'SUBPROGRAM <', MEMBER(1:LM), '> NAMED AS <',
623:         & TNAME(:LM1), '>'
624:       NMEM = NMEM + 1
625:       if ( NMEM.gt.MAXMEM ) then
626:         write(6,*) '=== Too many routines !! (max=', MAXMEM, ') '
627:         stop 8
628:       end if
629:       MLIST(NMEM) = TNAME
630:       MEMBER = TNAME
631:       LM = LM1
632: C
633:       call OFILEN( ODSN, MEMBER, LM1, LF, DIR, LDIR )
634: C
635:       return
636:     end
637: C
638: C-----
639: C
640: C Make lowercase character to uppercase
641: C
642:       subroutine UPPER( STR )
643:       character*(*) STR
644:       character*26 UP
645:       character*26 LOW
646:       data UP/'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
647:       data LOW/'abcdefghijklmnopqrstuvwxyz'/
648:       do 100 I = 1, LEN(STR)
649:         K = INDEX(LOW,STR(I:I))
650:         if ( K.ne.0 ) STR(I:I) = UP(K:K)

```


src/tools/myfsplit.f

```
651:      100 continue
652:      return
653:      end
654: C
655: C-----
656: C
657: C      Make uppercase character to lowercase
658: C
659:      subroutine LOWER( STR )
660:      character*(*) STR
661:      character*26 UP
662:      character*26 LOW
663:      data UP/'ABCDEFGHIJKLMNOPQRSTUVWXYZ' /
664:      data LOW/'abcdefghijklmnopqrstuvwxyz' /
665:      do 100 I = 1, LEN(STR)
666:         K      = INDEX(UP,STR(I:I))
667:         if ( K.ne.0 ) STR(I:I) = LOW(K:K)
668:      100 continue
669:      return
670:      end
```



src/tools/nlb2txt.f

```

1: C
2:      program BIN2TX
3: C
4: C
5: C=====
6: C MVP UTILITY: NEUTRON/PHOTON/ELECTRON/PHOTONUCLEAR LIBRARY FORM
7: C CONVERTER 1.
8: C BINARY --> TEXT
9: C=====
10: C
11: C=====
12: C
13: C
14: C << INPUT >> (From standard input. START ON COLUMN 1)
15: C
16: C 0. KIND OF LIBRARY ( NEUTRON/PHOTON/ELECTRON/PHOTONUCLEAR )
17: C (Character string must start on the first column.)
18: C
19: C 1. THE NAME OF INDEX FILE , OR BLANK LINE.
20: C
21: C 2. IF 1. IS NOT BLANK;
22: C Repeat [NUCLIDE ID'S AND] NAME OF OUTPUT FILE
23: C
24: C [NUC=nuclide-name-matching-pattern] [OUTPUT-FILE]
25: C
26: C When NUC=... is not specified, all nuclides in the index
27: C file are converted.
28: C When no output-file name is specified, a filename in the
29: C last input line will be used and append data to the file.
30: C
31: C If the name includes '*', files whose names are composed by
32: C replacing '*' with nuclide-id are output files for each
33: C nuclide;
34: C
35: C Example: LIB.*.MVP --> LIB.U05003J3.MVP
36: C
37: C else the name is output file name itself.
38: C
39: C When the output file name is given as ">nn" , where "nn"
40: C is a two digit integer, data are output on fortran I/O
41: C unit nn.
42: C
43: C IF 1. IS BLANK
44: C REPEAT PAIRS OF INPUT-FILE & OUTPUT-FILE NAMES IN ONE LINE
45: C (72 CHAR).
46: C A '+' INDICATES NEXT LINE HAS OUTPUT FILE NAME.
47: C IF NO-OUTPUT FILE SPECIFIED, THAT MEANS APPENDING OUPUT TO
48: C THE LAST OUTPUT FILE
49: C
50: C When the output file name is given as ">nn" , where "nn"
51: C is a two digit integer, data are output on fortran I/O
52: C unit nn.
53: C
54: C * When an output file is specified as ">nn" , I/O unit nn is not
55: C explicitly opened in this program. this is for use with JCL in
56: C mainframe computers.
57: C
58: C=====
59: C
60: C
61: C
62: C .... LIMIT : Allowable maximum size of working area.
63: C Modify this parameter and recompile when an memory error
64: C occurs.
65: C

```

```

66:      parameter( LIMIT = 10000000 )
67: C
68:      real ADATA(LIMIT)
69:      integer IEF(LIMIT)
70:      common /BUFFER/ ADATA, IEF
71: C
72:      common /IEFSIZ/ MXIEF
73: C
74:      character*256 INFILE
75:      character*256 TMPSTR
76:      character*72 PATH
77:      character*128 OTF
78:      character*128 INFL
79:      character*128 OTFILE
80:      character*128 FINDEX
81:      character*32 NUCID
82:      character*72 LINE
83: C
84:      character*16 PARTCL
85: CMODV3character*8 NUCNM
86:      character*16 NUCNM
87:      logical OPD
88: C
89: C ... nout00 is the default output I/O unit.
90: C
91:      parameter( NOUT00 = 20 )
92:      parameter( NIN00 = 10 )
93: C
94: C
95:      MXIEF = LIMIT
96: C
97: C
98:      write(6,*) ' '
99:      write(6,*) ' == MVP LIBRARY CONVERTER : BINARY --> TEXT '
100:      write(6,*) ' '
101: C
102: C
103: C***** READ PARTICLE TYPE: NEUTRON / PHOTON / ELECTRON / PHOTONUCLEAR ****
104: C
105: C
106:      write(6,*) 'PARTICLE (NEUTRON/PHOTON/ELECTRON/PHOTONUCLEAR) ->'
107: C
108:      100 read(5,'(A)') PARTCL
109:      if ( PARTCL(1:1).eq.' ' ) go to 100
110: C
111:      if ( PARTCL.ne.'NEUTRON' .and.PARTCL.ne.'PHOTON' .and.
112:      & PARTCL.ne.'ELECTRON'.and.PARTCL.ne.'PHOTONUCLEAR' ) then
113:          write(6,*) 'XXX INVALID PARTICLE TYPE :', PARTCL
114:          stop 666
115:      end if
116:      LLLP = INDEX(PARTCL,' ') - 1
117:      write(6,*) '* ', PARTCL(:LLL), ' LIBRARIES ARE CONVERTED.'
118: C
119: C
120: C***** READ INDEX FILE NAME **** IF BLANK, FILE BY FILE CONVERSION ****
121: C
122: C
123:      FINDEX = ' '
124:      write(6,*) 'NAME OF INDEX FILE -> '
125: C
126:      110 read(5,'(A)') FINDEX
127:      if ( FINDEX(1:1).eq.' ' ) go to 110
128: C
129:      write(6,*) 'INDEX: ', FINDEX
130: C

```

src/tools/nlb2txt.f

```

131: C=====
132: C   CONVERT DATA DESCRIBED IN INDEX FILE .
133: C=====
134: C
135:       if ( FINDEX.ne.' ' ) then
136:         NIDX = 25
137: C
138: C/#IF READONLY(DEC)
139: *       open( NIDX, file =FINDEX, iostat =IOS, form ='FORMATTED',
140: *           &         readonly )
141: C/#ELSEIF READONLY(ACTION)
142: *       open( NIDX, file =FINDEX, iostat =IOS, form ='FORMATTED',
143: *           &         action ='READ' )
144: C/#ELSEIF READONLY(MODE)
145: *       open( NIDX, file =FINDEX, iostat =IOS, form ='FORMATTED',
146: *           &         mode ='READ' )
147: C/#ELSE
148: *       open( NIDX, file =FINDEX, iostat =IOS, status ='OLD',
149: *           &         form ='FORMATTED' )
150: C/#ENDIF
151: C
152:       if ( IOS.ne.0 ) then
153:         write(6,*) ' == INDEX FILE OPEN ERROR : CODE ', IOS
154:         stop 666
155:       end if
156: C
157: C   .... READ OUTPUT FILE ....
158: C
159: C   IF OUTPUT FILE NAME HAS A '*' CHARACTER, TEXT FILES ARE
160: C   CREATED FOR EACH NUCLIDE AND NAMED BY REPLACING THE '*' CHARACTER
161: C   BY NUCLIDE-ID .
162: C   ELSE ALL OUTPUT ARE WRITTEN ON THE OUTPUT FILE ITSELF.
163: C
164: C
165:       NIN = NIN00
166:       NOUT = NOUT00
167:       OTF = ' '
168: C
169:       IREUSE = 1
170:       NCOM = 0
171: C
172: 120   NCOM = NCOM + 1
173: C
174:       write(6,*)
175:       &       '[NUC=NUCLIDE-ID] OUTPUT FILE ( WILD CHARACTER * )-> '
176: C
177:       LINE = ' '
178: 130   read(5,fmt ='(A)',end =230) LINE
179:       if ( LINE(1:1).eq.''' ) go to 130
180: C
181: C   ... separate [NUC=nuclide-ID] & OUTPUT FILE
182: C
183:       NUCID = ' '
184:       IREUSE = 1
185:       LNL = 0
186:       IE = 0
187: C
188: 140   IS = IE + 1
189:       call NOBLNK( LINE, IS, IE, LEN(LINE) )
190:       if ( IS.le.LEN(LINE) ) then
191:         K = INDEX(LINE(IS:IE),'NUC=')
192:         if ( K.eq.1 ) then
193:           NUCID = LINE(IS+LEN('NUC='):IE)
194:           LNL = INDEX(NUCID,' ') - 1
195:           if ( LNL.lt.0 ) LNL = LEN(NUCID)

```

```

196:       write(6,'(/' '   NUCLIDE-ID PATTERN : <' ' ,a,' '>' ')')
197:       &       NUCID(:LNL)
198:       else
199: C
200: C   .... ireuse : 1/0 =
201: C       append data to the file on the last line/no
202: C
203:       if ( OTF.ne.LINE(IS:IE) ) then
204:         OTF = LINE(IS:IE)
205:         IREUSE = 0
206:       end if
207:       end if
208:       go to 140
209: end if
210: C
211: if ( NCOM.eq.1.and.OTF.eq.' ' ) then
212:   write(6,*) 'XXX NO OUTPUT-FILE NAME GIVEN FOR THE ',
213:   &       'FIRST "[NUC=NUCLIDE-ID] [OUTPUT-FILE]" LINE.'
214:   stop 666
215: end if
216: C
217: write(6,'(/' '   OUTPUT FILE : ' ',A')') OTF
218: C
219: KA = INDEX(OTF,'*')
220: LO = INDEX(OTF,' ') - 1
221: if ( LO.lt.0 ) LO = LEN(OTF)
222: C
223: C
224: C   ... output filename includes no wildcard character '*' ...
225: C
226: if ( KA.eq.0 ) then
227: C
228: C   .... ">nn" for output file name --> output on I/O unit nn
229: C
230:       JFLNM = 1
231:       if ( OTF(1:1).eq.'>' ) then
232:         read(OTF(2:3),'(i2)',err =150) NOUT
233:         go to 160
234: C
235: 150   write(6,*) ' XXX output file name as ">nn" is ',
236:       &       'specified but invalid form : <', OTF(:LO), '>'
237:       stop 888
238: C
239: 160   continue
240:       JFLNM = 0
241:       else
242:         NOUT = NOUT00
243:       end if
244: C
245:       if ( NOUT.le.0 .or. NOUT.gt.99 ) then
246:         write(6,*) 'XXX Invalid output I/O unit specified. ',
247:         &       NOUT
248:         stop 888
249:       end if
250:       if ( NOUT.eq.NIN00 .or. NOUT.eq.NIDX ) then
251:         write(6,*) 'XXX Attempting to output data on ',
252:         &       'the input I/O unit (unit ', NIN00, ') or ',
253:         &       'on the index file (unit ', NIDX, ') '
254:         stop 888
255:       end if
256: C
257: C
258:       IOS = 0
259:       TMPSTR = ' '
260:       inquire(NOUT,opened =OPD)

```

src/tools/nlb2txt.f

```

261: C
262: C      ... Open output file when a filename was given explicitly.
263: C      Do not open when specified as an I/O unit #.
264: C
265: C      if ( .not.OPD ) then
266: C
267: C          if ( JFLNM.ne.0 ) then
268: C              open( NOUT, file =OTF, iostat =IOS, status = 'UNKNOWN',
269: C                  &      form = 'FORMATTED' )
270: C          end if
271: C
272: C      ... reopen I/O unit of outputfile.
273: C
274: C      else if ( IREUSE.eq.0 ) then
275: C          if ( JFLNM.ne.0 ) then
276: C              close( NOUT )
277: C              open( NOUT, file =OTF, iostat =IOS, status = 'UNKNOWN',
278: C                  &      form = 'FORMATTED' )
279: C          end if
280: C      end if
281: C      if ( IOS.ne.0 ) then
282: C          write(6,*) '== OUTPUT FILE OPEN ERROR : CODE ', IOS
283: C          stop 666
284: C      end if
285: C  end if
286: C
287: C
288: C      .... READ INDEX FILE (INPIDX ROUTINE OF MVP CODE (Apr 1998))
289: C
290: C
291: C      PATH      = ' '
292: C      LPATH      = 0
293: C      rewind NIDX
294: C
295: C 170      call INPIDX( NIDX, NUCNM, INFL, PATH, LPATH, 6, IEND )
296: C
297: C      if ( IEND.ne.0 ) then
298: C          write(6, '(/lx,a,/)' )
299: C          &      ' == ALL SPECIFIED NUCLIDE IN INDEX FILE ARE ',
300: C          &      ' CONVERTED SUCCESSFULLY ==='
301: C
302: C      ... input new command
303: C          go to 120
304: C      end if
305: C
306: C      if ( NUCID.ne.' ' ) then
307: C          call MATCH( IM, NUCID(:LNL), NUCNM )
308: C          if ( IM.eq.0 ) go to 170
309: C      end if
310: C
311: C      INFILE = INFL
312: C      LL      = INDEX(INFILE, ' ') - 1
313: C      if ( LL.lt.0 ) LL = LEN(INFILE)
314: C
315: C      write(6,*) 'INPUT: <', INFILE(:LL), '>'
316: C
317: C      inquire(NIN,opened =OPD)
318: C      if ( OPD ) close( NIN )
319: C
320: C      C/#IF READONLY(DEC)
321: C      *      open( NIN, file =INFILE(:LL), iostat =IOS, form = 'UNFORMATTED',
322: C      *      &      readonly )
323: C      C/#ELSEIF READONLY(ACTION)
324: C      *      open( NIN, file =INFILE(:LL), iostat =IOS, form = 'UNFORMATTED',
325: C      *      &      action = 'READ' )

```

```

326: C/#ELSEIF READONLY(MODE)
327: C      *      open( NIN, file =INFILE(:LL), iostat =IOS, form = 'UNFORMATTED',
328: C      *      &      mode = 'READ' )
329: C      C/#ELSE
330: C      *      open( NIN, file =INFILE(:LL), iostat =IOS, status = 'OLD',
331: C      *      &      form = 'UNFORMATTED' )
332: C      C/#ENDIF
333: C
334: C      if ( IOS.ne.0 ) then
335: C          write(6,*) ' == INPUT FILE OPEN ERROR : CODE ', IOS
336: C          stop 666
337: C      end if
338: C
339: C
340: C      ... When a part of output file name should be replaced by nuclide-ID.
341: C
342: C      if ( KA.ne.0 ) then
343: C          OTFILE = ' '
344: C          if ( KA.eq.1 ) then
345: C              if ( LO.eq.1 ) then
346: C                  OTFILE = NUCNM
347: C              else
348: C                  LL0 = INDEX(NUCNM, ' ') - 1
349: C                  if ( LL0.lt.0 ) LL0 = LEN(NUCNM)
350: C                  OTFILE = NUCNM(:LL0) //OTF(KA+1:LO)
351: C              end if
352: C          else if ( KA.eq.LO ) then
353: C              OTFILE = OTF(:KA-1) //NUCNM
354: C          else
355: C              LL0 = INDEX(NUCNM, ' ') - 1
356: C              if ( LL0.lt.0 ) LL0 = LEN(NUCNM)
357: C              OTFILE = OTF(:KA-1) //NUCNM(:LL0) //OTF(KA+1:LO)
358: C          end if
359: C
360: C          LL = INDEX(OTFILE, ' ') - 1
361: C          if ( LL.lt.0 ) LL = LEN(OTFILE)
362: C          write(6,*) ' OUTPUT FILE <', OTFILE(:LL), '>'
363: C
364: C          inquire(NOUT,opened =OPD)
365: C          if ( OPD ) close( NOUT )
366: C
367: C          open( NOUT, file =OTFILE, iostat =IOS, status = 'UNKNOWN',
368: C              &      form = 'FORMATTED' )
369: C
370: C          if ( IOS.ne.0 ) then
371: C              write(6,*) ' == OUTPUT FILE OPEN ERROR : CODE ', IOS
372: C              stop 666
373: C          end if
374: C      end if
375: C
376: C
377: C      if ( PARTCL.eq.'NEUTRON' ) then
378: C          call NMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
379: C      else if ( PARTCL.eq.'PHOTON' ) then
380: C          call PMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
381: C      else if ( PARTCL.eq.'ELECTRON' ) then
382: C          call EMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
383: C      else if ( PARTCL.eq.'PHOTONUCLEAR' ) then
384: C          call PNMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
385: C      end if
386: C
387: C      go to 170
388: C
389: C
390: C=====

```

src/tools/nlb2txt.f

```

391: C   FILE-BY-FILE CONVERSION.
392: C=====
393: C
394: C
395: C   else
396: C       NIN      = NIN00
397: C       NOUT     = NOUT00
398: C
399: C
400: C   180   write(6,*) '   # INPUT & OUTPUT FILE --> '
401: C
402: C   190   read(5,'(A72)',end =230) LINE
403: C       if ( LINE(1:1).eq.'''') go to 190
404: C
405: C   C..... INPUT FILE (BINARY) ....
406: C
407: C       IS      = 1
408: C       call NOBLNK( LINE, IS, IE, 72 )
409: C       INFILE  = LINE(IS:IE)
410: C       write(6,*) ' INPUT BINARY FILE: ', INFILE
411: C
412: C /#IF READONLY(DEC)
413: C *       open( NIN, file =INFILE, iostat =IOS, form ='UNFORMATTED',
414: C *           &         readonly )
415: C /#ELSEIF READONLY(ACTION)
416: C *       open( NIN, file =INFILE, iostat =IOS, form ='UNFORMATTED',
417: C *           &         action ='READ' )
418: C /#ELSEIF READONLY(MODE)
419: C *       open( NIN, file =INFILE, iostat =IOS, form ='UNFORMATTED',
420: C *           &         mode   ='READ' )
421: C /#ELSE
422: C *       open( NIN, file =INFILE, iostat =IOS, status ='OLD',
423: C *           &         form   ='UNFORMATTED' )
424: C /#ENDIF
425: C
426: C       if ( IOS.ne.0 ) then
427: C           write(6,*) ' == INPUT FILE OPEN ERROR : CODE ', IOS
428: C           stop 666
429: C       end if
430: C
431: C   C..... OUTPUT FILE .....
432: C
433: C       IS      = IE + 1
434: C       call NOBLNK( LINE, IS, IE, 72 )
435: C       if ( IS.gt.72 ) then
436: C           write(6,*) '== OUTPUT DATA ARE ADDED TO ', OTF
437: C           inquire(NOUT,opened =OPD)
438: C           if ( .not.OPD ) then
439: C               write(6,*) 'XXX OUTPUT FILE NOT OPENED !!! '
440: C               stop 666
441: C           end if
442: C       else
443: C           if ( LINE(IS:IS).eq.'+' ) then
444: C   200       read(5,'(A72)') LINE
445: C           if ( LINE(1:1).eq.'''') go to 200
446: C           IS      = 1
447: C           call NOBLNK( LINE, IS, IE, 72 )
448: C       end if
449: C       OTF      = LINE(IS:IE)
450: C       LO      = INDEX(OTF,' ') - 1
451: C       if ( LO.lt.0 ) LO = LEN(OTF)
452: C       write(6,*) ' OUTPUT FILE (TEXT) : ', OTF(:LO)
453: C
454: C
455: C   .... ">nn" for output file name --> output on I/O unit nn

```

```

456: C
457: C   JFLNM      = 1
458: C   if ( OTF(1:1).eq.'>' ) then
459: C       read(OTF(2:3),'(i2)',err =210) NOUT
460: C       go to 220
461: C
462: C   210   write(6,*) ' XXX output file name as ">nn" is ',
463: C   &       'specified but invalid form : <', OTF(:LO), '>'
464: C       stop 888
465: C
466: C   220   JFLNM      = 0
467: C   else
468: C       NOUT      = NOUT00
469: C   end if
470: C
471: C   if ( NOUT.le.0 .or. NOUT.gt.99 ) then
472: C       write(6,*) 'XXX Invalid output I/O unit specified. ',
473: C   &       NOUT
474: C       stop 888
475: C   end if
476: C   if ( NOUT.eq.NIN ) then
477: C       write(6,*) 'XXX Attempting to output data on ',
478: C   &       'the input I/O unit (unit ', NIN, ' )'
479: C       stop 888
480: C   end if
481: C
482: C   if ( JFLNM.ne.0 ) then
483: C       inquire(NOUT,opened =OPD)
484: C       if ( OPD ) close( NOUT )
485: C
486: C       open( NOUT, file =OTF, iostat =IOS, status ='UNKNOWN',
487: C   &       form   ='FORMATTED' )
488: C
489: C   if ( IOS.ne.0 ) then
490: C       write(6,*) ' == OUTPUT FILE OPEN ERROR : CODE ', IOS
491: C       stop 666
492: C   end if
493: C   end if
494: C
495: C   end if
496: C
497: C   if ( PARTCL.eq.'NEUTRON' ) then
498: C       call NMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
499: C   else if ( PARTCL.eq.'PHOTON' ) then
500: C       call PMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
501: C   else if ( PARTCL.eq.'PHOTONUCLEAR' ) then
502: C       call PNMDUMP( NIN, NOUT, ADATA, ADATA, IEF, LIMIT )
503: C   end if
504: C
505: C   close( NIN )
506: C
507: C   go to 180
508: C   end if
509: C
510: C   230 stop
511: C   end
512: C
513: C   subroutine INPIDX( INDX, IDS, FILE, PATH, LPATH, IOPR, IEND
514: C   &
515: C   )
516: C=<MVP>=====
517: C   PURPOSE: GET A NUCID & FILE NAME DATA FROM CROSS SECTION INDEX FILE.
518: C   CALLED FROM: FALOC, FALOCF
519: C   HISTORY: PROGRAMMED BY M.SASAKI (3 APRIL 1992)
520: C   UPDATE:

```

src/tools/nlb2txt.f

```

521: C 10 JUN 1993 : skip blank lines in index file.
522: C 13 AUG 1993 : get pathname of index file for use as default 'PATH'
523: C           for UNIX.
524: C 20 Apr 1994 : Correction for file name composition procedure (do
525: C           not add path to filenames with absolute pathname).
526: C           The head of filename in indexfile is LINE(IS:IS) and is not
527: C           file(1:1).
528: C 22 Jul 1996: enable MS-DOS type directory separation character '\'
529: C           as CHAR(92).
530: C 18 May 1997: enable MacOS type directory separation character ':'
531: C 14 Jul 1997: enable relative path name for "PATH" line.
532: C           Subroutine GTPATH is created to extract directory path of
533: C           a file opened on an I/O unit.
534: C 5 Apr 1998: a "C/#ELSEIF FILENAME(DOS)" should be
535: C           "C/#ELSEIF FILENAME(MACOS)"
536: C 20 Apr 1998: add print I/O unit to argument, and expand "$xxxx" in
537: C           PATH by environment variable.
538: C 26 Nov 2002: Extend dimensions of IDAT1 and EDAT1. (M.Sasaki)
539: C-----
540: C   INDX : I/O UNIT OF INDEX FILE
541: C   IDS : ID   FETCHED.
542: C   FILE : FILE NAME FETCHED.
543: C   PATH : CURRENT PATH.
544: C   LPATH : LENGTH OF CURRENT PATH NAME.
545: C   IOPR : message printout I/O unit.
546: C   IEND : 0 = DATA FETCHER
547: C           1 = END OF INDEX-FILE
548: C=====
549: C   character*(*) IDS, FILE, PATH
550: C
551: C   character*72 LINE
552: C   character*256 CWRK
553: C   logical NMD
554: C
555: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
556: C
557: C   ... get path name of index file ...
558: C
559: C   if ( PATH.eq.' ' ) then
560: C     call GTPATH( INDX, PATH, LPATH )
561: C   end if
562: C
563: C/#ENDIF
564: C
565: C
566: C 100 LINE = ' '
567: C   read(INDX,fmt ='(A)',end =110) LINE
568: C   if ( LINE.eq.' ' ) go to 100
569: C   if ( LINE(1:1).eq.'*' ) go to 100
570: C
571: C   .... GET ID CHARACTER OR 'PATH' STRING ....
572: C
573: C   IS = 1
574: C   call NOBLNK( LINE, IS, IE, LEN(LINE) )
575: C   IE = IS + MIN(LEN(IDS)-1,IE-IS)
576: C
577: C
578: C   .... PATH NAME ....
579: C
580: C   if ( LINE(IS:IE).eq.'PATH' ) then
581: C
582: C     .... GET PATH NAME ...
583: C
584: C     IS = IE + 1
585: C     call NOBLNK( LINE, IS, IE, LEN(LINE) )

```

```

586: C     IE = IS + MIN(LEN(PATH)-1,IE-IS)
587: C     if ( IE.ge.IS ) then
588: C       PATH = LINE(IS:IE)
589: C       LPATH = IE - IS + 1
590: C
591: C/#IF .NOT.NOGETENV
592: C
593: C   .... environment variable expansion is not supported in this program.
594: C
595: C       KK = INDEX(PATH,'$')
596: C       if ( KK.ne.0 ) then
597: C         write(IOPR,7000) PATH(:ICLEN2(PATH))
598: C         7000 format(1X,'XXX(INPIDX) Environment variable expansion',
599: C           & ' in PATH is currently not supported in NLB2TXT ...'/
600: C           & 1X,' <PATH ',A,'>')
601: C         stop 888
602: C       end if
603: C/#ENDIF
604: C
605: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
606: C
607: C   ... when PATH is given by relative path ...
608: C
609: C   UNIX : not start with '/'
610: C   MS-DOS(Windows) : not have disk drive letter ( "[a-z]:" )
611: C   MacOS : beginning with ":"
612: C
613: C/#IF UNIX
614: C   if ( LPATH.gt.0.and.PATH(1:1).ne.'/' ) then
615: C/#ELSEIF FILENAME(DOS)
616: C   *   if ( LPATH.gt.0.and.PATH(2:2).ne.':' ) then
617: C/#ELSEIF FILENAME(MACOS)
618: C   *   if ( LPATH.gt.0.and.PATH(1:1).eq.':' ) then
619: C/#ENDIF
620: C
621: C   ... get absolute path of index file again ...
622: C   call GTPATH( INDX, FILE, LPP )
623: C
624: C   PATH = FILE(:LPP) //PATH(:LPATH)
625: C
626: C   end if
627: C
628: C/#ENDIF
629: C   end if
630: C   go to 100
631: C   else
632: C
633: C   .... ID ....
634: C
635: C   IDS = ' '
636: C   IDS = LINE(IS:IE)
637: C
638: C   .... GET FILENAME
639: C
640: C   IS = IE + 1
641: C   call NOBLNK( LINE, IS, IE, LEN(LINE) )
642: C   IE = IS + MIN(LEN(FILE)-1,IE-IS)
643: C
644: C   FILE = ' '
645: C
646: C   if ( LPATH.gt.0 ) then
647: C/#IF UNIX
648: C
649: C   ... filename with relative directory path ..
650: C   if ( LINE(IS:IS).ne.'/' ) then

```

src/tools/nlb2txt.f

```

651:          FILE      = PATH(1:LPATH) //LINE(IS:IE)
652:      else
653:          FILE      = LINE(IS:IE)
654:      end if
655: C
656: C/#ELSE
657: *          FILE      = PATH(1:LPATH) //LINE(IS:IE)
658: C/#ENDIF
659:      else
660:          FILE      = LINE(IS:IE)
661:      end if
662: C
663: C      ... SHIP DUMMY LINE (FOR FUTURE USE) ....
664: C
665: Ccccccccccccc READ(INDX,'()')
666:      end if
667: C
668:      IEND      = 0
669:      return
670: C
671: C      ..... END OF INDEX FILE ...
672: C
673: 110 IEND      = 1
674:      return
675:      end
676: C
677: C =====
678: C subroutine gtpath gets directory path name of file opened on I/O
679: C unit IOU
680: C
681:      subroutine GTPATH( IOU,  PATH,  LPATH )
682: C
683:      character*(*) PATH
684: C
685: C ... local data ...
686: C
687:      character*256 FILE
688:      logical NMD
689: C
690: C/#IF UNIX .OR. FILENAME(DOS) .OR. FILENAME(MACOS)
691: C
692:      PATH      = ' '
693:      LPATH      = 0
694:      FILE      = ' '
695:      inquire(IOU,name =FILE,named =NMD,iostat =IOS)
696: C
697:      if ( IOS.eq.0 ) then
698:          if ( NMD ) then
699:              K      = 0
700:              do 100 K = LEN(FILE), 1, -1
701: C/#IF FILENAME(DOS)
702: C
703: C      CHAR(92) is '\ ' in ASC-II character code.
704: C      '\ ' may be taken as "escape code" characters in character string
705: C      on some Fortran compilers. so such an expression is used.
706: C
707: *          if ( FILE(K:K).eq.CHAR(92) ) go to 110
708: C/#ELSEIF FILENAME(MACOS)
709: *          if ( FILE(K:K).eq.':' ) go to 110
710: C/#ELSE
711:          if ( FILE(K:K).eq.'/' ) go to 110
712: C/#ENDIF
713: 100      continue
714:      K      = 0
715: 110      continue

```

```

716:          if ( K.gt.0 ) then
717:              PATH      = FILE(:K)
718:              LPATH      = K
719:          end if
720:      end if
721:      end if
722: C/#ENDIF
723:      return
724:      end
725: C
726: C
727: C
728: C
729:      subroutine NMDUMP( NIN,  NOUT,  ADATA, IDATA, IEF,  NTDAT3 )
730: C=====
731: C      NEUTRON LIBRARY CONVERTER (MVP)
732: C
733: C=====
734:      parameter( MTMAX      = 120, NKMAX      = 90 )
735:      parameter( MXREC      = 1000 )
736: C
737:      common /MAINIO/ NSYSI,  NSYSO,  NWRK,  NWRK2
738:      common /CONTRL/ MATD,  ZA,  ELUNR,  EHUNR,  CRSMIN, LEMORY,
739: &          IUNRES,  INTUNR, IDUMP,  IDBG
740: C
741:      common /HEAD1/  NPTS,  NTDATA,  NUNR,  NUNR2,  NLEV,  NBINA,
742: &          NBINE,  NNU,  NNK,  IGFLAG, LFI,  LNU,
743: &          I THERM,  ISTU,  IENDU,  LSUNR,  LSNU,  NBINA1, NBINE1,
744: &          NNUD,  LNNUD,  NNF,  LSNUD,  NCAP,  NCAPG,
745: &          KDUMMY(74),  ATW,  TLAB,  ETHERM, ESAB,  ELOW,
746: &          EHI,  TSTAR,  RDUMMY(43)
747: C      integer IDAT1(1)
748: C      real EDAT1(1)
749: C ... for Intel Fortran ...
750:      integer IDAT1(101)
751:      real EDAT1(101)
752:      equivalence(NPTS,IDAT1(1)), (NPTS,EDAT1(1))
753: C
754:      common /HEAD2/  MTINFO(MTMAX),  MTPAR(MTMAX),  ISTMT(MTMAX),
755: &          IENDMT(MTMAX),  NEUMT(MTMAX),  LCTMT(MTMAX),
756: &          NEANG(MTMAX),  NKF5(MTMAX),  NEF5(MTMAX),
757: &          MTTHR(MTMAX),  LSTF3(MTMAX),  LSTF4(MTMAX),
758: &          LSTF5(MTMAX),  QVAL(MTMAX),  QVAL2(MTMAX),
759: &          LSTGAM(MTMAX),  MTTOG(MTMAX),  NEGYLD(MTMAX),
760: &          NEF5G(MTMAX),  NKF5G(MTMAX),  LCTMTG(MTMAX),
761: &          MTCAP(MTMAX),  MTCAPG(MTMAX),  EG1L(MTMAX),
762: &          EG2U(MTMAX),  LSTF5S(NKMAX,MTMAX)
763:      integer IDAT2(1)
764:      real EDAT2(1)
765:      equivalence(MTINFO(1),IDAT2(1)), (MTINFO(1),EDAT2(1))
766: C
767:      real ADATA(NTDAT3)
768:      integer IDATA(NTDAT3)
769:      integer IEF(NTDAT3)
770: C
771:      integer LSTF4G(MTMAX)
772:      real ESK(MTMAX)
773: C
774:      character*16 MATT
775:      character*8 TITLE(15)
776:      character*8 HDATE
777: C
778: C
779: C ... local data ...
780: C

```

src/tools/nlb2txt.f

```

781:      character CTEMP*136
782: C
783: C-----READ INPUT DATA
784: C
785: C
786: C===== RECORD 1 =====
787: C
788:      rewind NIN
789:      read(NIN) MATT, HDATE, TITLE, NPTS, NTDATA, NUNR, NUNR2, NLEV,
790:      CMOD &      NBINA, NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITERM,
791:      CMOD &      ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF,
792:      CMOD &      LSNUD, NCAP, NCAPG, (KDUMMY(I),I=1,8), ATW, TLAB, ETHERM,
793:      CMOD &      ESAB, ELOW, EHI, TSTAR, RDUMMY
794:      read(NIN) CTEMP
795: C
796:      rewind NIN
797:      LIBTYP = 2
798:      IF(CTEMP(17:25).eq.'VERSION 3') LIBTYP = 3
799:      MATT = ' '
800:      TITLE(15) = ' '
801: C
802:      if( LIBTYP.eq.2 ) then
803:          read(NIN) MATT(1:8), HDATE,(TITLE(I),I=1,15),
804:          &      NPTS, NTDATA, NUNR, NUNR2, NLEV,
805:          &      NBINA, NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITERM,
806:          &      ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF,
807:          &      LSNUD, NCAP, NCAPG, (KDUMMY(I),I=1,8), ATW, TLAB, ETHERM,
808:          &      ESAB, ELOW, EHI, TSTAR, (RDUMMY(I),I=1,12)
809: C
810:      else
811:          read(NIN) MATT(1:16), (TITLE(I),I=1,14),HDATE,
812:          &      NPTS, NTDATA, NUNR, NUNR2, NLEV,
813:          &      NBINA, NBINE, NNU, NMT, NNK, IGFLAG, LFI, LNU, ITERM,
814:          &      ISTU, IENDU, LSUNR, LSNU, NBINA1, NBINE1, NNUD, LNUD, NNF,
815:          &      LSNUD, NCAP, NCAPG, (KDUMMY(I),I=1,74), ATW, TLAB, ETHERM,
816:          &      ESAB, ELOW, EHI, TSTAR, (RDUMMY(I),I=1,43)
817:      end if
818: Caddb6
819:      LF6 = KDUMMY(1)
820:      LSTF6 = KDUMMY(2)
821:      LLSSF = KDUMMY(3)
822:      LSTDOP = KDUMMY(4)
823: Caddb6th
824:      LASYM = KDUMMY(5)
825:      NOTDOP = KDUMMY(6)
826: C
827:      ZA = RDUMMY(1)
828:      MATH = RDUMMY(2)
829: Cend
830: C
831: C
832: C===== RECORD 2 =====
833: C
834:      read(NIN) MTINFO, MTPAR, ISTMT, IENDMT, NEUMT, LCTMT, NEANG, NKF5,
835:      &      NEF5, MTHR, LSTF3, LSTF4, LSTF5, QVAL, QVAL2, LSTGAM,
836:      &      MTTOG, NEGYLD, NEF5G, NKF5G, LCTMTG, MTCAP, MTCAPG, EGLL,
837:      &      EG2U
838: C
839:      if ( NTDATA.gt.NTDAT3 ) then
840:          write(6,*) ' XXX TOO LARGE #3 DATA (', NTDATA, ' ) '
841:          write(6,*) ' LIMIT (NTDAT3) = ', NTDAT3
842:          stop 999
843:      end if
844: C
845:      NSYSO = 6

```

```

846: C
847: C
848: C===== RECORD 3 =====
849: C
850:      read(NIN) (ADATA(I),I=1,NTDATA)
851: C
852: C ***** added for t-ftree version
853: C
854:      ITFREE = 0
855:      LENTHS = 0
856:      LENU3R = 0
857:      LSTTHS = 0
858:      LSTU3R = 0
859:      ISWB6T = 0
860: C
861:      if ( LSTDOP.gt.0 ) then
862:          ITFREE = 1
863:          LENTHS = IDATA(LSTDOP)
864:          LENU3R = IDATA(LSTDOP+1)
865:          LSTTHS = 0
866:          if ( LENTHS.gt.0 ) LSTTHS = LSTDOP + 2
867:          LSTU3R = 0
868:          if ( LENU3R.gt.0 ) then
869:              LSTU3R = LSTDOP + 2
870:              if ( LENTHS.gt.0 ) LSTU3R = LSTTHS + LENTHS
871:          end if
872:      end if
873: C
874:      NTU3R = 0
875:      NURT = 0
876:      NBNU3R = 0
877:      IFISUN = 0
878:      if ( LSTU3R.gt.0 ) then
879:          NTU3R = IDATA(LSTU3R)
880:          NURT = IDATA(LSTU3R+1)
881:          NBNU3R = IDATA(LSTU3R+2)
882:          IFISUN = IDATA(LSTU3R+3)
883:      end if
884: C
885:      NTTHSC = 0
886:      NPTHE = 0
887:      LCTTHE = 0
888:      LTCTHE = 0
889:      NRETHE = 0
890:      NETHE = 0
891:      NBNTHE = 0
892:      NPPTHIN = 0
893:      NIGTH = 0
894:      NFGTH = 0
895:      MXNGTH = 0
896:      NATOMT = 0
897:      MPCUT = 0
898:      EMAXTH = 0.0
899:      EHITHE = 0.0
900: Caddb6th
901:      NTHELS = 0
902:      LTHR = 0
903:      NEBRG = 0
904:      NPDBY = 0
905:      EMXTH6 = 0.0
906:      ELWTHE = 0.0
907:      SBMT2 = 0.0
908: Cend
909:      if ( LSTTHS.gt.0 ) then
910:          NTTHSC = IDATA(LSTTHS)

```


src/tools/nlb2txt.f

```

911:      NPTHE = IDATA(LSTTHS+1)
912:      LCTTHE = IDATA(LSTTHS+2)
913:      NRETHE = IDATA(LSTTHS+3)
914:      NETHE = IDATA(LSTTHS+4)
915:      NBNTHE = IDATA(LSTTHS+5)
916:      NPTHIN = IDATA(LSTTHS+6)
917:      NIGTH = IDATA(LSTTHS+7)
918:      NFGTH = IDATA(LSTTHS+8)
919:      MXNGTH = IDATA(LSTTHS+9)
920:      NATOMT = IDATA(LSTTHS+10)
921:      NPTCUT = IDATA(LSTTHS+11)
922:      EMXTH = ADATA(LSTTHS+12)
923:      EHTHE = ADATA(LSTTHS+13)
924: caddb6th
925:      if(LCTTHE.lt.0) ISWB6T = 1
926: c
927:      if( ISWB6T.eq.1 ) then
928:          NTHELS = IDATA(LSTTHS+14)
929:          LTHR = IDATA(LSTTHS+15)
930:          NEBRG = IDATA(LSTTHS+16)
931:          NPDBY = IDATA(LSTTHS+17)
932:          EMXTH6 = ADATA(LSTTHS+18)
933:          ELWTHE = ADATA(LSTTHS+19)
934:          SBMT2 = ADATA(LSTTHS+20)
935:      end if
936:      end if
937: C
938: CHECK %%
939: c      WRITE(6,*) ' A(LSTF5(1 ... ) ',
940: c      & (IDATA(I),I=LSTF5(1),LSTF5(1)+110)
941: c      WRITE(6,*) ' A(LSTF5(1 ... ) ',
942: c      & (ADATA(I),I=LSTF5(1),LSTF5(1)+110)
943: C
944:      do 110 K = 1, 15
945:          if ( INDEX(TITLE(K),CHAR(0)).ne.0 ) then
946:              do 100 I = 1, LEN(TITLE(K))
947:                  if ( TITLE(K)(I:I).eq.CHAR(0) ) TITLE(K)(I:I) = ' '
948:              100 continue
949:          end if
950:      110 continue
951: C
952: C
953: C
954: C      PRINT OUT LIBRARY INFORMATION
955: C
956:      write(NSYSO,7000) MATT, HDATE, TITLE
957:      write(NSYSO,7020) NPTS, NTDATA, NUNR, NUNR2, NLEV, NBINA, NBINE,
958: c      & NNU, NMT
959:      write(NSYSO,7040) NNK, IGFLAG, LFI, LNU, IOTHERM, ISTU, IENDU,
960: c      & LSUNR, LSNU, NBINAL, NBINE1
961:      write(NSYSO,7080) NNUD, LNUD, NNF, LSNU, NCAP, NCAPG
962:      write(NSYSO,7100) ATW, TLAB, ETHERM, ESAB, ELOW, EHI, TSTAR, ZA,
963: c      & MATH
964: Caddb6
965: Cmod write(NSYSO,7060) LF6, LLSSF, LSTF6
966: Caddb6th
967:      write(NSYSO,7060) LF6, LLSSF, LSTF6,ISWB6T,LASYM,NOTDOP
968: Cend
969:
970: c      IF(NLEV.GT.0)      WRITE(NSYSO,104) (MTTHR(I),I=1,NLEV)
971: c      IF(NCAP.GT.0)      WRITE(NSYSO,124) (MTCAP(I),I=1,NCAP)
972: c      IF(NCAPG.GT.0)     WRITE(NSYSO,125) (MTCAPG(I),I=1,NCAPG)
973: C
974:      7000 format(/1X,20X,'*****'/'1X,
975:      &      20X,'* MVP MATERIAL LIBRARY INFORMATION *'/'1X,20X,
```

```

976:      &      '*****'/'1X,20X,
977:      &      '* MATERIAL ID = ',A16,' *'/'1X,20X,
978:      &      '*****'/'1X,20X,
979:      &      '* PRODUCTION DATE : ',A8,' *'/'1X,20X,
980:      &      '*****'/'1X,10X,
981:      &      ' COMMENT : ',8A8/1X,10X,' : ',7A8//)
982:      7020 format(1X,15X,'NPTS : NO OF SMOOTH DATA GRID RECORD.....',I10/
983:      &      1X,15X,'NTDATA: TOTAL RECORD OF #3 RECORD (WORDS)...',I10/
984:      &      1X,15X,'NUNR : NO OF UNRESOLVED PROB. TABLES .....',I10/
985:      &      1X,15X,'NUNR2 : LENGTH OF UNRESOLVED PROB. TABLE ...',I10/
986:      &      1X,15X,'NLEV : NO OF THRESHOLD REACTIONS .....',I10/
987:      &      1X,15X,'NBINA : NO OF BINS IN ANGULAR DIS. TABLE....',I10/
988:      &      1X,15X,'NBINE : NO OF BINS IN ENERGY DIS. TABLE....',I10/
989:      &      1X,15X,'NNU : LENGTH OF NU-DATA .....',I10/
990:      &      1X,15X,'NMT : TOTAL NO OF REACTION CONSIDERED.....',I10)
991:      7040 format(1X,15X,'NNK : MAX NO OF SUB-SECTIONS IN FILE-5....',I10/
992:      &      1X,15X,'IGFLAG: GAMMA PRODUCTION DATA FLAG .....',I10/
993:      &      1X,15X,'LFI : FLAG OF FISSION REACTION .....',I10/
994:      &      1X,15X,'LNU : NU-DATA FLAG (1/2=POLY/TAB).....',I10/
995:      &      1X,15X,'ITHERM: FLAG OF THERMAL SCATTERING .....',I10/
996:      &      1X,15X,'ISTU : UPPER ENERGY MESH POINT OF U.R. ....',I10/
997:      &      1X,15X,'IENDU : LOWER ENERGY MESH POINT OF U.R. ....',I10/
998:      &      1X,15X,'LSUNR : START OF POSITION OF U.R. TABLE ....',I10/
999:      &      1X,15X,'LSNU : START OF POSITION OF NU-DATA.....',I10/
1000:      &      1X,15X,'NBINAL: NBINA + 1 .....',I10/
1001:      &      1X,15X,'NBINE1: NBINE + 1 .....',I10)
1002: Caddb6
1003:      7060 format(1X,15X,'LF6 : Angle-energy data flag(0/1).....',I10/
1004:      &      1X,15X,'LLSSF : Unresolved infinite flaf(0/1).....',I10/
1005:      &      1X,15X,'LSTF6 : Start postion of angle-energy data....',I10/
1006:      &      1X,15X,'ISWB6T: A FLAG OF ENDF/B6 THERNAL DATA ....',I10/
1007:      &      1X,15X,'LASYM : A FLAG OF S(A,B) ASYMMETRY.....',I10/
1008:      &      1X,15X,'NOTDOP: A FLAG OF DOPPLER BROADENING.....',I10)
1009: Cend
1010:      7080 format(1X,15X,'NNUD : LENGTH OF DELAYED NU-DATA.....',I10/
1011:      &      1X,15X,'LNUD : DELAYED NU-DATA FLAG(1/2=POLY/TAB)...',I10/
1012:      &      1X,15X,'NNF : NO OF PRECURSOR FAMILIES .....',I10/
1013:      &      1X,15X,'LSNU : START OF POSITION OF DELAYED NU ....',I10/
1014:      &      1X,15X,'NCAP : NO OF CAPTURE REACTIONS .....',I10/
1015:      &      1X,15X,'NCAPG : NO OF GAMMA YIELD REAC. BY CAPTURE....',I10)
1016:      7100 format(1X,15X,'ATW : ATOMIC WEIGHT IN N.M.U. ....',F12.3
1017:      &      /1X,15X,'TLAB : LABORATORY TEMPERATURE IN KELVIN ...',
1018:      &      F12.3/1X,15X,
1019:      &      'ETHERM: UPPER ENERGY FOR THERMAL ELA. MOD...',F12.3/1X,
1020:      &      15X,'ESAB : HIGHEST ENERGY FOR THERMAL INELA ...',F12.3/
1021:      &      1X,15X,'ELOW : LOWER ENERGY DEFINED (EV) .....',
1022:      &      F12.5/1X,15X,
1023:      &      'EHI : UPPER ENERGY DEFINED (EV) .....',F12.1/1X,
1024:      &      15X,'TSTAR : EFFECTIVE TEMPERATURE IN KELVIN ....',F12.3/
1025:      &      1X,15X,'ZA : The (Z&A) designation .....',
1026:      &      F12.1/1X,15X,
1027:      &      'MATNO : Material number in ENDF .....',I12//)
1028: Caddb6th
1029:      7110 format(1X,15X,'NTHELS: NUMBER OF TEMP. IN THERMAL ELASTIC..',I12,
1030:      &      /1X,15X,'NEBRG : NUMBER OF BRAGG EDGES .....',I12,
1031:      &      /1X,15X,'NPDBY : NUMBER OF DEBYE-WALLER INTEGRAL.....',I12,
1032:      &      /1X,15X,'EMXTH6: MAX. ENERGY OF ENDF/B6 THERMAL DATA.',F12.4,
1033:      &      /1X,15X,'ELWTHE: MIN. ENERGY OF THERMAL ELASTIC DATA.',F12.4,
1034:      &      /1X,15X,'SBMT2 : BOUND XS FOR INCOHERENT ELASTIC ....',F12.4)
1035: Cend
1036:      7120 format(1X,15X,'MTTHR : THRESHOLD REACTION MT ID .....',I10I4)
1037:      7140 format(1X,15X,'MTCAP : REACTION ID FOR CAPTURE .....',I10I4)
1038:      7160 format(1X,15X,'MTCAPG: REACTION ID FOR (CAPTURE,GAMMA).....',I10I4)
1039: C
1040:      write(NSYSO,7180)
```

src/tools/nlb2txt.f

```

1041:      write(NSYSO,7200)
1042:      write(NSYSO,7220)
1043:      do 120 MT = 1, NMT
1044:        if ( MTINFO(MT).le.0 ) go to 120
1045:        write(NSYSO,7240) MT, MTINFO(MT), ISTMT(MT), IENDMT(MT),
1046:      &      MTPAR(MT), NEUMT(MT), QVAL(MT), QVAL2(MT)
1047:      120 continue
1048:      write(NSYSO,7220)
1049: C
1050: 7180 format(/1X,20X,'*****'/'1X,
1051:      &      20X,'* #2 RECORD INFORMATION (PART 1) *'/'1X,20X,
1052:      &      '*****'/'//)
1053: 7200 format(1X,10X,' MT MTINFO ISTMT IENDMT MTPAR NEUMT ',
1054:      &      ' QVAL(EV) QVAL2(EV) ')
1055: 7220 format(1X,9X,7('-----'))
1056: 7240 format(1X,10X,I3,5I8,1P2E12.5)
1057: C
1058:      write(NSYSO,7260)
1059:      write(NSYSO,7280)
1060:      write(NSYSO,7300)
1061:      do 130 MT = 1, NMT
1062:        if ( MTINFO(MT).le.0.and.NKF5(MT).le.0 ) go to 130
1063:        write(NSYSO,7320) MT, LSTF3(MT), LCTMT(MT), NEANG(MT),
1064:      &      LSTF4(MT), NKF5(MT), NEF5(MT), LSTF5(MT)
1065:      130 continue
1066:      write(NSYSO,7300)
1067: C
1068: 7260 format(/1X,20X,'*****'/'1X,
1069:      &      20X,'* #2 RECORD INFORMATION (PART 2) *'/'1X,20X,
1070:      &      '*****'/'//)
1071: 7280 format(' ',10X,' MT LSTF3 LCTMT NEANG LSTF4 NKF5 ',
1072:      &      'NEF5 LSTF5 ')
1073: 7300 format(' ',9X,7('-----'))
1074: 7320 format(' ',10X,I3,8I8)
1075: 7340 format(' ',10X,' LFF5S : ',10I8)
1076: 7360 format(' ',10X,' NEF5S : ',10I8)
1077: 7380 format(' ',10X,' LSTF5S : ',10I8)
1078: C
1079: C *** print tfree control data
1080: C
1081:      if ( ITFREE.gt.0 ) then
1082:        write(NSYSO,7400) LSTDOP, LSTTHS, LSTU3R, LENTHS, LENU3R
1083: C
1084:        if ( LSTU3R.gt.0 ) then
1085:          write(NSYSO,7420) NTU3R, NURT, NBNU3R, IFISUN
1086:          LSTTU3 = LSTU3R + 3
1087:          ISW0 = LSTTU3 + NTU3R
1088:          write(NSYSO,7440) (ADATA(LSTTU3+I),I=1,NTU3R)
1089:          write(NSYSO,7460) (ADATA(ISW0+I),I=1,NTU3R)
1090:        end if
1091: C
1092:        if ( LSTTHS.gt.0 ) then
1093:          write(NSYSO,7480) NTHSC, NPTHE, LCTTHE, NRETHE, NETHE,
1094:      &      NBNTHE, NPPTHIN, NIGTH, NFGTH, MXNGTH, NATOMT,
1095:      &      NPTCUT, EMAXTH, EHITHE
1096: Caddb6th
1097:          if ( ISWB6T.NE.1 ) then
1098:            LSTTSC = LSTTHS + 13
1099:            ISW1 = LSTTSC + NTHSC
1100:            ISW2 = LSTTSC + NTHSC*2
1101:            write(NSYSO,7500) (ADATA(LSTTSC+I),I=1,NTHSC)
1102:            write(NSYSO,7520) (ADATA(ISW1+I),I=1,NTHSC)
1103:            if ( NPTHE.gt.0 )
1104:      &      write(NSYSO,7540) (IDATA(ISW2+I),I=1,NTHSC)
1105: C

```

```

1106:      else
1107:        write(NSYSO,7110) NTHELS,NEBRG,NPDBY,EMXTH6,ELWTHE,SBMT2
1108:        LSTTSC = LSTTHS + 20
1109:        ISW1 = LSTTSC + NTHSC
1110:        ISW2 = LSTTSC + NTHSC*2
1111:        write(NSYSO,7500) (ADATA(LSTTSC+I),I=1,NTHSC)
1112:        write(NSYSO,7520) (ADATA(ISW1+I),I=1,NTHSC)
1113:        write(NSYSO,7540) (IDATA(ISW2+I),I=1,NTHSC)
1114:      end if
1115: C
1116:      end if
1117:      end if
1118: C
1119: C
1120: 7400 format('1',20X,'*****'/'1X,
1121:      &      20X,'* tfree control data information *'/'1X,20X,
1122:      &      '*****'/'//1X,15X,
1123:      &      'lstdop: start address of temp. dep. data ...',I10/1X,15X,
1124:      &      'lstths: start address of thermal scattering.',I10/1X,15X,
1125:      &      'lstu3r: start address of u.r.p. table .....',I10/1X,15X,
1126:      &      'lenths: record length of thermal scattering.',I10/1X,15X,
1127:      &      'lenu3r: record length of u.r.p. table .....',I10/)
1128: 7420 format(/1X,15X,'***** unresolved probability data control ****'/
1129:      &      1X,15X,'ntu3r : no of temperature in u.r.p. data ...',I10/
1130:      &      1X,15X,'nurt : no of energy in file-2 u.r. eval....',I10/
1131:      &      1X,15X,'nbnu3r: no of brobaility bin .....',I10/
1132:      &      1X,15X,'ifisun: fission reaction flag(0/1:no/yes)...',I10)
1133: 7440 format(' temperatures .... ',11F10.3)
1134: 7460 format(' log(temp.) .... ',11F10.6)
1135: C
1136: 7480 format(/1X,15X,'***** thermal scattering data control *****/
1137:      &      1X,15X,'ntthsc: no of temperature in th.sc. data ...',I10/
1138:      &      1X,15X,'npthe : no of thermal elastic energy mesh...',I10/
1139:      &      1X,15X,'lctthe: flame of elastic scattering angle...',I10/
1140:      &      1X,15X,'nrethe: interpolation scheme for inci. eng..',I10/
1141:      &      1X,15X,'nethe : no of incident energy in els. scat..',I10/
1142:      &      1X,15X,'nbnthe: no of angle bins for elastic scat...',I10/
1143:      &      1X,15X,'npthin: no of thermal inel. energy mesh 1...',I10/
1144:      &      1X,15X,'nigth : no of thermal inel. energy mesh 2...',I10/
1145:      &      1X,15X,'nfgth : no of thermal inel. energy mesh 3...',I10/
1146:      &      1X,15X,'mxngth: =(nfg)*(nfg+1)/2 or nfg*nfg .....',I10/
1147:      &      1X,15X,'natomt: no of atoms in molecule .....',I10/
1148:      &      1X,15X,'nptcut: no of cut data in originl mt=2 ....',I10/
1149:      &      1X,15X,'emaxth: = esab .....',
1150:      &      F10.5/1X,15X,
1151:      &      'ehithe: highest energy of thermal elastic .',F10.5)
1152: C
1153: 7500 format(' temperatures .... ',11F10.3)
1154: 7520 format(' effective temp... ',11F10.3)
1155: 7540 format(' interpol. method. ',11I10)
1156: C
1157: C-----
1158: C
1159: C      OUTPUT TO TEXT FILE
1160: C
1161: C-----
1162: C
1163: C .... SET RECORD LENGTH OF OUTPUT FILE ...(80 CHARACTER)
1164: C
1165:      call POPEN( NOUT, 80 )
1166: C
1167: C .... @NUC=NUCLIDE-ID .....
1168: C
1169:      CTEMP(1:30) = '
1170: C

```

src/tools/nlb2txt.f

```

1171:      if ( LIBTYP.eq.2 ) then
1172:         CTEMP  = '@NUC="//MATT(1:8)
1173:      else
1174:         CTEMP  = '@NUC="//MATT(1:16)
1175:      end if
1176: c
1177:      LCT      = INDEX(CTEMP,' ') - 1
1178:      if ( LCT.lt.0 ) LCT = LEN(CTEMP)
1179:      call CWRITE( CTEMP(:LCT), 0, IRT )
1180:      call NXTLIN( IRT )
1181:      call CWRITE( '***** TEXT IMAGE MVP LIBRARY (NEUTRON) ', 0, IRT )
1182:      call NXTLIN( IRT )
1183: c
1184: c
1185: c
1186: c --- RECORD #1 ---
1187: c
1188: c
1189: c ..... 17 CHARACTER STRINGS EACH OF THEM HAS 8-CHARACTERS ...
1190: c
1191:      call CWRITE( '@CHAR 17 8 ', 0, IRT )
1192: c
1193:      if ( LIBTYP.eq.2 ) then
1194:         call SWRITE( MATT, 8, IRT )
1195:         call SWRITE( HDATE, 8, IRT )
1196:         do 140 I = 1, 15
1197:            call SWRITE( TITLE(I), 8, IRT )
1198:         140 continue
1199:      else
1200:         call SWRITE( MATT(1:8) , 8, IRT )
1201:         call SWRITE( MATT(9:16) , 8, IRT )
1202:         do 141 I = 1, 14
1203:            call SWRITE( TITLE(I), 8, IRT )
1204:         141 continue
1205:         call SWRITE( HDATE, 8, IRT )
1206:      end if
1207:      call NXTLIN( IRT )
1208: c
1209:      if ( LIBTYP.eq.2 ) then
1210: c
1211: c ... NPTS -- KDUMMY(8) ... (INTEGER DATA)
1212: c
1213:         LSS      = 1
1214:         ND       = 34
1215:         call IDOUT( IDAT1(LSS), ND, IRT )
1216:         LSS      = LSS + ND
1217: c
1218: c ... ATW -- RDUMMY(12) ... (REAL DATA)
1219: c
1220:         ND       = 19
1221:         call EDOUT( EDAT1(101), ND, IRT )
1222: c
1223:         else
1224: c
1225: c ... NPTS -- KDUMMY(74) ... (INTEGER DATA)
1226: c
1227:         LSS      = 1
1228:         ND       = 100
1229:         call IDOUT( IDAT1(LSS), ND, IRT )
1230:         LSS      = LSS + ND
1231: c
1232: c ... ATW -- RDUMMY(43) ... (REAL DATA)
1233: c
1234:         ND       = 50
1235:         call EDOUT( EDAT1(LSS), ND, IRT )

```

```

1236: c
1237:      endif
1238: c
1239:      call NXTLIN( IRT )
1240:      call CWRITE( 'END-OF-RECORD-1', 0, IRT )
1241:      call NXTLIN( IRT )
1242: c
1243: c
1244: c --- RECORD #2 ---
1245: c
1246: c ... MTINFO -- LSTF5 ...
1247: c
1248:         LSS      = 1
1249:         ND       = 13*MTMAX
1250:         call IDOUT( IDAT2(LSS), ND, IRT )
1251:         LSS      = LSS + ND
1252: c
1253: c ... QVAL & QVAL2 .....
1254: c
1255:         ND       = 2*MTMAX
1256:         call EDOUT( EDAT2(LSS), ND, IRT )
1257:         LSS      = LSS + ND
1258: c
1259: c ... LSTGAM -- MTCAPG ...
1260: c
1261:         ND       = 8*MTMAX
1262:         call IDOUT( IDAT2(LSS), ND, IRT )
1263:         LSS      = LSS + ND
1264: c
1265: c ... EG1L & EG2U .....
1266: c
1267:         ND       = 2*MTMAX
1268:         call EDOUT( EDAT2(LSS), ND, IRT )
1269:         LSS      = LSS + ND
1270: c
1271:         call NXTLIN( IRT )
1272:         call CWRITE( 'END-OF-RECORD-2', 0, IRT )
1273:         call NXTLIN( IRT )
1274: c
1275: c
1276:         write(6,*) ' == RECORD-2: NUMBER OF DATA =', LSS - 1
1277: c
1278: c
1279: c --- RECORD #3 ---
1280: c
1281: c IEF(I) : FLAG THAT INDICATES DATA TYPE (INTEGER/REAL) OF ADATA(I)
1282: c
1283: c
1284: c
1285:         LSS      = 1
1286: c*
1287: c*
1288: c***** EMESH, XNU, XNUD, RAMDA *****
1289: c*
1290: c*
1291:         L0       = LSS
1292:         LSS      = LREALF( IEF,NPTS,LSS)
1293:         LSS      = LREALF( IEF,LNU*NNU,LSS)
1294:         LSS      = LREALF( IEF,LNUD*NNUD,LSS)
1295:         LSS      = LREALF( IEF,NNF,LSS)
1296:         call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1297: c*
1298: c*
1299: c*
1300:         call NXTLIN( IRT )

```

src/tools/nlb2txt.f

```

1301:      call CWRITE( '# -- SMOOTH CROSS SECTION ---- ', 0, IRT )
1302:      call NXTLIN( IRT )
1303: C
1304:      L0      = LSS
1305:      LSS      = LREALF(IEF,LSTF4(1)-LSTF3(1),LSS)
1306:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1307: C
1308: C
1309:      call NXTLIN( IRT )
1310:      call CWRITE( '# -- ANGULAR DISTRIBUTION ---- ', 0, IRT )
1311:      call NXTLIN( IRT )
1312:      L0      = LSS
1313: C
1314:      write(6,*) ' L0  LSTF4(1) ', L0, LSTF4(1)
1315:      if ( L0.ne.LSTF4(1) ) then
1316:        write(6,*) ' INVALID POSITION : (ANGULAR DIST.DATA) ', L0,
1317:      & ' ( LSTF4(1)= ', LSTF4(1), ' ) '
1318:      end if
1319: C*
1320: C*
1321: C***** ANGULAR DISTRIBUTION DATA *****
1322: C*
1323: C*
1324:      do 160 M = 1, NMT
1325: C
1326: C
1327: C*
1328: C***** EANG,INTANG,UMTAB *****
1329: C*
1330: C
1331: C
1332: C      ... SPECIAL TREATEMENT FOR MT=92 (THERMAL INELASTIC) ...
1333: C
1334:      if ( M.eq.92 ) then
1335:        LSS      = LREALF(IEF,NEANG(92),LSS)
1336:        LSS      = LINTF(IEF,NEANG(92),LSS)
1337:        do 150 J = 1, NEANG(92)
1338: Cmodb6th      LSS      = LREALF(IEF,5*J,LSS)
1339:        NBIN      = J*5
1340:        IF(LASYM.EQ.1) NBIN = NEANG(92)*5
1341:        LSS      = LREALF(IEF,NBIN,LSS)
1342: Cend
1343:      150      continue
1344:      else
1345:        LSS      = LREALF(IEF,NEANG(M),LSS)
1346:        LSS      = LINTF(IEF,NEANG(M),LSS)
1347:        LSS      = LREALF(IEF,NBINAL*NEANG(M),LSS)
1348:      end if
1349:      160 continue
1350:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1351: C*
1352: C***** angle-energy control data ****mttof6,ef611,ef62u*****
1353: C*
1354: C*
1355:      if ( LF6.gt.0 ) then
1356: C
1357:        write(6,*) ' lss lstf6 ', LSS, LSTF6
1358:        if ( LSS.ne.LSTF6 ) then
1359:          write(6,*) ' invalid position : (angle-energy control) ',
1360:        & ' LSS, ' lstf6 = ', LSTF6, ' ) '
1361:        end if
1362: C
1363:      call NXTLIN( IRT )
1364:      call CWRITE( '# -- mttof6 ef611 ef62u ----- ', 0, IRT )
1365:      call NXTLIN( IRT )

```

```

1366: C
1367:      L0      = LSS
1368:      LSS      = LINTF(IEF,NMT,LSS)
1369:      LSS      = LREALF(IEF,NMT*2,LSS)
1370:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1371:      end if
1372: C
1373: Cend
1374: C
1375:      write(6,*) ' LSS  LSTF5(1) ', LSS, LSTF5(1)
1376:      if ( LSS.ne.LSTF5(1) ) then
1377:        write(6,*) ' INVALID POSITION : (ENRGY DIST.DATA) ', LSS,
1378:      & ' ( LSTF5(1)= ', LSTF5(1), ' ) '
1379:      end if
1380: C*
1381: C*
1382: C***** ENERGY DISTRIBUTION DATA *****
1383: C*
1384: C*
1385:      call NXTLIN( IRT )
1386:      call CWRITE( '# -- ENERGY DISTRIBUTION ---- ', 0, IRT )
1387:      call NXTLIN( IRT )
1388:      L0      = LSS
1389: C
1390:      do 260 M = 1, NMT
1391:      if ( NKF5(M).ne.0 ) then
1392:        write(6,*) ' M LSS  LSTF5(M) ', M, LSS, LSTF5(M)
1393:        write(6,*) ' LNUD ', LNUD
1394:        if ( NEF5(M).ne.IDATA(LSS) .or. NKF5(M).ne.IDATA(LSS+1) )
1395:        then
1396:          write(6,*) ' !!! SIZE UNMATCH BETWEEN RECORD #2 & #3 '
1397:          write(6,*) ' MT=', M, ' NEF5 ', NEF5(M), ' (#2) ',
1398:        & IDATA(LSS), ' (#3) '
1399:          write(6,*) ' MT=', M, ' NKF5 ', NKF5(M), ' (#2) ',
1400:        & IDATA(LSS+1), ' (#3) '
1401:          stop 888
1402:        end if
1403: C
1404: Caddb6
1405:      ISWF6      = 0
1406:      if ( NKF5(M).lt.0 ) ISWF6      = 1
1407:      NKF5(M) = IABS(NKF5(M))
1408: Cend
1409: C
1410: C ... NEF5,NKF5,LSTF5S ...
1411:      LSS      = LINTF(IEF,2+NKF5(M),LSS)
1412: C ... EPROB ...
1413:      LSS      = LREALF(IEF,NEF5(M),LSS)
1414: C ... INTF5 ...
1415:      LSS      = LINTF(IEF,NEF5(M),LSS)
1416: C
1417: C ... QF5, INK1, INK2 ... if file-5 data
1418: C
1419:      if ( ISWF6.eq.0 ) then
1420:        do 170 NK = 1, NEF5(M)
1421:          write(6,*) ' nk= ', NK, ' nkf5 ', NKF5(M)
1422:          LSS      = LREALF(IEF,NKF5(M),LSS)
1423:          LSS      = LINTF(IEF,2*NKF5(M),LSS)
1424:      170      continue
1425: C
1426:      else
1427: C ... tyeild ...
1428:      LSS      = LREALF(IEF,NEF5(M),LSS)
1429:      do 180 NK = 1, NKF5(M)
1430: C ... subsection yield ...

```

src/tools/nlb2txt.f

```

1431:          write(6,*) '      nk=', NK, ' nef5 ', NEF5(M)
1432:          LSS      = LREALF(IEF,NEF5(M),LSS)
1433: 180          continue
1434: C
1435:          end if
1436: C
1437: C ... ENERGY DISTRIBUTION CLASSIFIED BY LF VALUE ...
1438: C
1439:          do 250 NK = 1, NK5(M)
1440:          LF      = IDATA(LSS)
1441: C
1442:          write(6,*) ' LF = ', LF, ' NK = ', NK
1443:          write(6,*) ' MT = ', M, ' LSS= ', LSS
1444: C
1445:          if ( LF.eq.1 ) then
1446:          NEF5S   = IDATA(LSS+1)
1447:          LSS      = LINTF(IEF,2,LSS)
1448:          LSS      = LREALF(IEF,NEF5S,LSS)
1449:          LSS      = LINTF(IEF,NEF5S,LSS)
1450: C** 28 OCT 1992 ***
1451: C TWO TYPE FOR LF=1 CASE
1452:          if ( NBINE.gt.0 ) then
1453:          LSS      = LREALF(IEF,2*NBINE1*NEF5S,LSS)
1454:          else if ( NBINE.eq.0 ) then
1455:          LSS      = LINTF(IEF,NEF5S,LSS)
1456:          do 190 J = 1, NEF5S
1457:          NEPI     = IDATA(LSS)
1458:          LSS      = LINTF(IEF,1,LSS)
1459:          LSS      = LREALF(IEF,NEPI,LSS)
1460:          LSS      = LINTF(IEF,2*NEPI,LSS)
1461:          LSS      = LREALF(IEF,2*(NEPI+1),LSS)
1462: 190          continue
1463:          else
1464:          write(6,*) 'XXX INVALID NBINE VALUE !! ', NBINE
1465:          stop
1466:          end if
1467: C*** 28 OCT 1992 *** END OF MODIFICATION ***
1468:          else if ( LF.eq.2 ) then
1469:          LSS      = LINTF(IEF,2,LSS)
1470:          LSS      = LREALF(IEF,2,LSS)
1471:          else if ( LF.eq.3 ) then
1472:          LSS      = LINTF(IEF,2,LSS)
1473:          LSS      = LREALF(IEF,2,LSS)
1474:          LSS      = LINTF(IEF,2,LSS)
1475:          LSS      = LREALF(IEF,2,LSS)
1476:          else if ( LF.eq.4 ) then
1477:          NEF5S   = IDATA(LSS+1)
1478:          LSS      = LINTF(IEF,2,LSS)
1479:          LSS      = LREALF(IEF,NEF5S,LSS)
1480:          LSS      = LINTF(IEF,NEF5S,LSS)
1481:          do 200 KK = 1, NEF5S
1482:          LSS      = LREALF(IEF,NEF5S,LSS)
1483:          LSS      = LINTF(IEF,2*NEF5S,LSS)
1484: 200          continue
1485:          else if ( LF.eq.5 ) then
1486:          NEF5S   = IDATA(LSS+1)
1487:          LSS      = LINTF(IEF,2,LSS)
1488:          LSS      = LREALF(IEF,NEF5S,LSS)
1489:          LSS      = LINTF(IEF,NEF5S,LSS)
1490: C** 28 OCT 1992 ***
1491: C TWO TYPE FOR LF=5 CASE
1492:          if ( NBINE.gt.0 ) then
1493:          LSS      = LREALF(IEF,NEF5S+2*NBINE1+1,LSS)
1494:          else if ( NBINE.eq.0 ) then
1495:          LSS      = LREALF(IEF,NEF5S,LSS)
1496:          NEP      = IDATA(LSS)
1497:          LSS      = LINTF(IEF,1,LSS)
1498:          LSS      = LREALF(IEF,NEP,LSS)
1499:          LSS      = LINTF(IEF,2*NEP,LSS)
1500:          LSS      = LREALF(IEF,2*(NEP+1)+1,LSS)
1501:          else
1502:          write(6,*) 'XXX INVALID NBINE VALUE !! ', NBINE
1503:          stop 666
1504:          end if
1505: C** 28 OCT 1992 *** END OF MODIFICATION
1506:          else if ( LF.eq.6 ) then
1507:          NEF5S   = IDATA(LSS+1)
1508:          LSS      = LINTF(IEF,2,LSS)
1509:          LSS      = LREALF(IEF,NEF5S,LSS)
1510:          LSS      = LINTF(IEF,NEF5S,LSS)
1511: C** 28 OCT 1992 ***
1512: C TWO TYPE FOR LF=6 CASE
1513:          if ( NBINE.gt.0 ) then
1514:          LSS      = LREALF(IEF,2*NBINE1*NEF5S,LSS)
1515:          else if ( NBINE.eq.0 ) then
1516:          LSS      = LINTF(IEF,NEF5S,LSS)
1517:          do 210 J = 1, NEF5S
1518:          NEPI     = IDATA(LSS)
1519:          LSS      = LINTF(IEF,1,LSS)
1520:          LSS      = LREALF(IEF,NEPI,LSS)
1521:          LSS      = LINTF(IEF,2*NEPI,LSS)
1522:          LSS      = LREALF(IEF,2*(NEPI+1),LSS)
1523: 210          continue
1524:          else
1525:          write(6,*) 'XXX INVALID NBINE VALUE !! ', NBINE
1526:          stop 666
1527:          end if
1528: C** 28 OCT 1992 *** END OF MODIFICATION
1529:          else if ( LF.eq.7 ) then
1530:          NEF5S   = IDATA(LSS+1)
1531:          LSS      = LINTF(IEF,2,LSS)
1532:          LSS      = LREALF(IEF,NEF5S,LSS)
1533:          LSS      = LINTF(IEF,NEF5S,LSS)
1534:          LSS      = LREALF(IEF,NEF5S+1,LSS)
1535:          else if ( LF.eq.9 ) then
1536:          NEF5S   = IDATA(LSS+1)
1537:          LSS      = LINTF(IEF,2,LSS)
1538:          LSS      = LREALF(IEF,NEF5S,LSS)
1539:          LSS      = LINTF(IEF,NEF5S,LSS)
1540:          LSS      = LREALF(IEF,NEF5S+1,LSS)
1541:          else if ( LF.eq.11 ) then
1542:          NEF5S   = IDATA(LSS+1)
1543:          LSS      = LINTF(IEF,2,LSS)
1544:          LSS      = LREALF(IEF,NEF5S,LSS)
1545:          LSS      = LINTF(IEF,NEF5S,LSS)
1546:          LSS      = LREALF(IEF,2*NEF5S+1,LSS)
1547: Caddb6 new lf = 61
1548:          else if ( LF.eq.61 ) then
1549:          NEF5S   = IDATA(LSS+1)
1550:          LSS      = LINTF(IEF,2,LSS)
1551:          LSS      = LREALF(IEF,NEF5S,LSS)
1552:          LSS      = LINTF(IEF,NEF5S,LSS)
1553: C
1554:          if ( NBINE.gt.0 ) then
1555:          write(NSYSO,*) ' ** nbine must be 0 for lf=61 !! '
1556:          stop 916
1557: C
1558:          else if ( NBINE.eq.0 ) then
1559:          LSS      = LINTF(IEF,NEF5S,LSS)
1560:          do 220 J = 1, NEF5S

```

src/tools/nlb2txt.f

```

1561:          NEPI    = IDATA(LSS)
1562:          LSS      = LINTF(IEF,2,LSS)
1563:          LSS      = LREALF(IEF,NEPI,LSS)
1564:          LSS      = LINTF(IEF,2*NEPI,LSS)
1565:          LSS      = LREALF(IEF,4*(NEPI+1),LSS)
1566: 220      continue
1567:      else
1568:          write(6,*) 'XXX Invalid NBINE value !! ', NBINE
1569:          stop
1570:      end if
1571: Caddb6 new lf = 66
1572:      else if ( LF.eq.66 ) then
1573:          LSS      = LINTF(IEF,2,LSS)
1574:          LSS      = LREALF(IEF,2,LSS)
1575:          LSS      = LINTF(IEF,2,LSS)
1576:          LSS      = LREALF(IEF,2,LSS)
1577:          LSS      = LINTF(IEF,1,LSS)
1578: Caddb6 new lf = 67
1579:      else if ( LF.eq.67 ) then
1580:          NEF5S    = IDATA(LSS+1)
1581:          LSS      = LINTF(IEF,2,LSS)
1582:          LSS      = LREALF(IEF,NEF5S,LSS)
1583:          LSS      = LINTF(IEF,NEF5S,LSS)
1584: C
1585:          if ( NBINE.ne.0 ) then
1586:              write(NSYSO,*) ' ** NBINE must be 0 for LF=67 !!!'
1587:              write(NSYSO,*) ' ** but NBINWE is ', NBINE, ' !!!'
1588:              stop 906
1589:          end if
1590: C
1591:          LSS      = LINTF(IEF,NEF5S,LSS)
1592:          do 240 J = 1, NEF5S
1593:              LSS    = LREALF(IEF,NBINA1,LSS)
1594:              NMU     = IDATA(LSS+1)
1595:              LSS     = LINTF(IEF,2,LSS)
1596:              LSS     = LREALF(IEF,NMU,LSS)
1597:              LSAVE   = LSS
1598:              LSS     = LINTF(IEF,NMU,LSS)
1599:              do 230 K = 1, NMU
1600:                  if ( LSS.ne.IDATA(LSAVE+K-1) ) then
1601:                      write(6,*)
1602:                          & ' invalid position : (LF=67 data) ',
1603:                          & LSS, ' ( LST2A(k)= ',
1604:                          & IDATA(LSAVE+K-1), ' )'
1605:                      write(6,*) ' Eloop j & Mu-loop k is ', J, K,
1606:                          & ' & NMU is ', NMU
1607:                  end if
1608:                  NEPI    = IDATA(LSS)
1609:                  LSS     = LINTF(IEF,1,LSS)
1610:                  LSS     = LREALF(IEF,NEPI,LSS)
1611:                  LSS     = LINTF(IEF,2*NEPI,LSS)
1612:                  LSS     = LREALF(IEF,2*(NEPI+1),LSS)
1613:              230      continue
1614:          240      continue
1615: Cend
1616: Caddb6th new lf = 71
1617:          else if ( LF.eq.71 ) then
1618:              NEF5S    = IDATA(LSS+1)
1619:              write(6,*) ' ** LF NK NEF5S LSS : ',LF,NK,NEF5S,LSS
1620:              LSS      = LINTF(IEF,2,LSS)
1621:              LSS      = LREALF(IEF,NEF5S,LSS)
1622:              LSS      = LREALF(IEF,NEF5S,LSS)
1623:              LSS      = LINTF(IEF,NEF5S,LSS)
1624:              write(6,*) ' ** LF NK NEF5S LSS : ',LF,NK,NEF5S,LSS
1625: C
1626:          do 245 J = 1, NEF5S
1627:              NBIN     = IDATA(LSS)
1628:              NBIN2    = NBIN*2
1629:              LSS      = LINTF(IEF,1,LSS)
1630:              write(6,*) ' ** LF LOP NBIN LSS : ',LF,J,NBIN,LSS
1631:              LSS      = LREALF(IEF,NBIN,LSS)
1632:              LSS      = LINTF(IEF,NBIN2,LSS)
1633:          245      continue
1634: C*** 28 OCT 1992 : ADD LF TYPE CHECK ****
1635:          else
1636:              write(6,*) 'XXX UNDEFINED LF ', LF, ' FOR ',
1637:                  & 'ENERGY DISTRIBUTION DATA !!!'
1638:              stop 666
1639:          end if
1640:          continue
1641: C
1642:          end if
1643:          260 continue
1644: C
1645:          call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1646: C
1647:          write(6,*) ' LSS LSUNR ', LSS, LSUNR
1648: C
1649:          if ( LSS.ne.LSUNR ) then
1650:              write(6,*) ' POSITION OF UNRS. IS INVALID !! ', LSS,
1651:                  & ' ( LSUNR = ', LSUNR, ' ) '
1652:          end if
1653: C*
1654: C*
1655: C***** UNRESOLVED RESONANCE DATA *****
1656: C*
1657: C*
1658:          call NXTLIN( IRT )
1659:          call CWRITE( '# -- UNRESOLVED RESONANCE DATA ---- ', 0, IRT )
1660:          call NXTLIN( IRT )
1661:          L0      = LSS
1662: C
1663: C ... EUNR      ...
1664: C
1665:          LSS     = LREALF(IEF,NUNR,LSS)
1666: C
1667: C ... INTUNR    ...
1668: C
1669:          LSS     = LINTF(IEF,NUNR,LSS)
1670: Caddb6
1671:          if ( LLSSF.eq.0 ) then
1672:              LENUN  = 3*NUNR2
1673:          else
1674:              LENUN  = 3*NUNR2 + 3
1675:          end if
1676: Cend
1677:          do 270 K = 1, NUNR
1678:              LSS    = LREALF(IEF,NUNR2,LSS)
1679:              LSS    = LINTF(IEF,2*NUNR2,LSS)
1680:              LSS    = LREALF(IEF,LENUN,LSS)
1681:          270      continue
1682: C
1683:          call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1684: C*
1685: C*
1686: C***** GAMMA GENERATION DATA *****
1687: C*
1688: C*
1689:          if ( IGFLAG.eq.1 ) then
1690: C

```

src/tools/nlb2txt.f

```

1691:      call NXTLIN( IRT )
1692:      call CWRITE( '# -- GAMMA GENERATION DATA ---- ', 0, IRT )
1693:      call NXTLIN( IRT )
1694:      L0      = LSS
1695: C
1696:      do 280 M = 1, NMT
1697:          if ( LSTGAM(M).ne.IDATA(LSS+M-1)
1698:      &      .or. MTTOG(M).ne.IDATA(LSS+NMT+M-1) ) then
1699:              write(6,*) '!!! SIZE UNMATCH BETWEEN RECORD #2 & #3'
1700:              write(6,*) 'MT=', M, ' LSTGAM ', LSTGAM(M), ' (#2) ',
1701:      &      IDATA(LSS+M-1), ' (#3) '
1702:              write(6,*) 'MT=', M, ' MTTOG ', MTTOG(M), ' (#2) ',
1703:      &      IDATA(LSS+NMT+M-1), ' (#3) '
1704:          end if
1705:      280 continue
1706: C
1707: C ... LSTGAM , MTTOG ...
1708: C
1709:      LSS      = LINTF(IEF,2*NMT,LSS)
1710:      LSSM      = 0
1711: C
1712:      do 320 M = 1, NMT
1713:          if ( LSTGAM(M).ne.0 ) then
1714: C
1715:              write(6,*) ' MT ', M, ' LSS ', LSS, ' LSTGAM ', LSTGAM(M)
1716: C
1717:              LSS      = LSTGAM(M)
1718:              NE        = NEGYLD(M)
1719:              NEGP       = NEF5G(M)
1720:              NKG        = NKF5G(M)
1721: C ... NEG ...
1722:              if ( NE.ne.IDATA(LSS) ) then
1723:                  write(6,*) '!!! SIZE UNMATCH BETWEEN RECORD #2 & #3'
1724:                  write(6,*) 'MT=', M, ' NEGYLD ', NE, ' (#2) ',
1725:      &      IDATA(LSS), ' (#3) '
1726:              end if
1727:              LSS      = LINTF(IEF,1,LSS)
1728: C
1729: C ... EMESHG, TYIELD ...
1730: C
1731:              LSS      = LREALF(IEF,2*NE,LSS)
1732: C
1733: C ... NEGP, NKG, LSTF5G ...
1734: C
1735:              if ( NEGP.ne.IDATA(LSS) ) then
1736:                  write(6,*) '!!! SIZE UNMATCH BETWEEN RECORD #2 & #3'
1737:                  write(6,*) 'MT=', M, ' NEGP ', NE, ' (#2) ',
1738:      &      IDATA(LSS), ' (#3) '
1739:              end if
1740:              if ( NKG.ne.IDATA(LSS+1) ) then
1741:                  write(6,*) '!!! SIZE UNMATCH BETWEEN RECORD #2 & #3'
1742:                  write(6,*) 'MT=', M, ' NKG ', NE, ' (#2) ',
1743:      &      IDATA(LSS+1), ' (#3) '
1744:              end if
1745: CM1994-9-26
1746:              LSTF5G     = LSS + 1
1747: C
1748:              LSS      = LINTF(IEF,2+NKG,LSS)
1749: C
1750: C ... ESK, EPROBG ...
1751: C
1752:              LSS      = LREALF(IEF,NKG+NEGP,LSS)
1753: C
1754: C ... INTF5G ...
1755: C

```

```

1756:      LSS      = LINTF(IEF,NEGP,LSS)
1757: C
1758: C ... CY ...
1759: C
1760:      LSS      = LREALF(IEF,NKG+NEGP,LSS)
1761: C
1762:      do 310 NK = 1, NKG
1763:          LSS      = IDATA(LSTF5G+NK)
1764:          LF        = IDATA(LSS)
1765: C
1766:          if ( LF.eq.2 ) then
1767:              LSS      = LINTF(IEF,2,LSS)
1768:              LSS      = LREALF(IEF,2,LSS)
1769:          else if ( LF.eq.6 ) then
1770:              NEF5S     = IDATA(LSS+1)
1771:              LSS      = LINTF(IEF,2,LSS)
1772:              LSS      = LREALF(IEF,NEF5S,LSS)
1773:              LSS      = LINTF(IEF,NEF5S,LSS)
1774:              if ( NBINE.gt.0 ) then
1775:                  LSS      = LREALF(IEF,2*NBINE1*NEF5S,LSS)
1776:              else if ( NBINE.eq.0 ) then
1777:                  LSS      = LINTF(IEF,NEF5S,LSS)
1778:                  do 290 J = 1, NEF5S
1779:                      NEPI    = IDATA(LSS)
1780:                      LSS      = LINTF(IEF,1,LSS)
1781:                      LSS      = LREALF(IEF,NEPI,LSS)
1782:                      LSS      = LINTF(IEF,2*NEPI,LSS)
1783:                      LSS      = LREALF(IEF,2*(NEPI+1),LSS)
1784:                  290 continue
1785:              end if
1786: C % % % % %
1787:
1788:      &
1789: Caddb6 lf=61
1790:
1791:          else if ( LF.eq.61 ) then
1792:              NEF5S     = IDATA(LSS+1)
1793:              LSS      = LINTF(IEF,2,LSS)
1794:              LSS      = LREALF(IEF,NEF5S,LSS)
1795:              LSS      = LINTF(IEF,NEF5S,LSS)
1796: C
1797:              if ( NBINE.gt.0 ) then
1798:                  write(NSYSO,*)
1799:                  &      ' ** nbine must be 0 for lf=61 ]]'
1800:                  stop 916
1801: C
1802:              else if ( NBINE.eq.0 ) then
1803:                  LSS      = LINTF(IEF,NEF5S,LSS)
1804:                  do 300 J = 1, NEF5S
1805:                      NEPI    = IDATA(LSS)
1806:                      LSS      = LINTF(IEF,2,LSS)
1807:                      LSS      = LREALF(IEF,NEPI,LSS)
1808:                      LSS      = LINTF(IEF,2*NEPI,LSS)
1809:                      LSS      = LREALF(IEF,4*(NEPI+1),LSS)
1810:                  300 continue
1811:              else
1812:                  write(6,*) 'xxx invalid nbine value !! ', NBINE
1813:                  stop
1814:              end if
1815: C
1816: C*** lf type check ****
1817:      &
1818:              write(6,*) 'XXX Undefined LF ', LF, ' for ',
1819:      &      ' gamma energy distribution data !!!'
1820:              write(6,*) 'XXX MT NKG NK : ', M, NKG, NK
1821:              stop 777

```

src/tools/nlb2txt.f

```

1821: C
1822:           end if
1823:           write(6,*) ' LSS ', LSS, ' LSTGAN ', IDATA(LSS),
1824:           &           ' NEGANG ', IDATA(LSS+1)
1825:           LSTGAN = IDATA(LSS)
1826:           NEGANG = IDATA(LSS+1)
1827: C
1828: C ... LSTGAN, NEGANG
1829:           LSS = LINTF(IEF,2,LSS)
1830: C
1831:           LSS = LSTGAN
1832: C ... EGANG ...
1833:           LSS = LREALF(IEF,NEGANG,LSS)
1834: C ... INTGAN ...
1835:           LSS = LINTF(IEF,NEGANG,LSS)
1836: C ... UMG TAB ...
1837:           LSS = LREALF(IEF,NBINA1*NEGANG,LSS)
1838:           310 continue
1839: C
1840: C*** 28 OCT 1992 *** END OF MODIFICATION
1841: C
1842:           LSSM = MAX(LSS,LSSM)
1843:           end if
1844:           320 continue
1845:           LSS = LSSM
1846: C
1847:           write(6,*) ' LSSM ', LSSM, ' NTDATA ', NTDATA
1848:           call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1849:           end if
1850: Cadd
1851: C **** process for t-free version ****
1852: C
1853:           if ( ITFREE.ne.0 ) then
1854: C ....
1855:           call NXTLIN( IRT )
1856:           call CWRITE( '# -- temperature dependent data --- ', 0, IRT )
1857:           call NXTLIN( IRT )
1858: C ....
1859:           L0 = LSS
1860: C ... lenth & lenu3r
1861:           LSS = LINTF(IEF,2,LSS)
1862: C ... thermal scattering data
1863:           if ( LENTHS.gt.0 ) then
1864: caddb6th
1865:           if(ITHERM.eq.0 ) then
1866:           LSS = LREALF(IEF,2*NTTHSC,LSS)
1867:           LSS = LINTF(IEF, NTTHSC,LSS)
1868:           endif
1869: C
1870:           if(ITHERM.ne.0) then
1871:           if(ISWB6T.eq.1) then
1872:           LSS = LINTF(IEF,12,LSS)
1873:           LSS = LREALF(IEF, 2,LSS)
1874:           LSS = LINTF(IEF, 4,LSS)
1875:           LSS = LREALF(IEF, 3,LSS)
1876:           LSS = LREALF(IEF,2*NTTHSC,LSS)
1877:           LSS = LINTF(IEF, NTTHSC,LSS)
1878: C .... thermal elastic data
1879:           if ( NTHELS.gt.0 ) then
1880:           LENGWK = NTHELS + NPTHE
1881:           LSS = LREALF(IEF,LENGWK,LSS)
1882:           LSS = LINTF(IEF,NTHELS,LSS)
1883:           end if
1884:           if ( NEBRG .gt.0 ) then
1885:           LENGWK = NEBRG*(NTHELS+1)

```

```

1886:           LSS = LREALF(IEF,LENGWK,LSS)
1887:           end if
1888:           if ( NPDBY .gt.0 ) then
1889:           LSS = LREALF(IEF,NPDBY ,LSS)
1890:           end if
1891:           if ( NPTHIN.gt.0 ) then
1892:           LENGWK = NPTHIN*(NTTHSC+1)
1893:           LSS = LREALF(IEF,LENGWK,LSS)
1894:           end if
1895:           if ( NFGTH.gt.0 ) then
1896:           LENGWK = NFGTH + NTTHSC*(NFGTH*NIGTH+4*MXNGTH)+MXNGTH
1897:           LSS = LREALF(IEF,LENGWK,LSS)
1898:           end if
1899:           if ( NPTCUT.gt.0 ) then
1900:           LSS = LREALF(IEF,NPTCUT,LSS)
1901:           end if
1902: C
1903:           else
1904: Cend
1905:           LSS = LINTF(IEF,12,LSS)
1906:           LSS = LREALF(IEF,2,LSS)
1907:           LSS = LREALF(IEF,2*NTTHSC,LSS)
1908: C .... thermal elastic data
1909:           if ( NPTHE.gt.0 ) then
1910:           LSS = LINTF(IEF,NTTHSC,LSS)
1911:           LENGWK = NPTHE*(NTTHSC+1)
1912:           LSS = LREALF(IEF,LENGWK,LSS)
1913:           end if
1914:           if ( NRETHE.gt.0 ) then
1915:           LSS = LINTF(IEF,2*NRETHE,LSS)
1916:           end if
1917:           if ( NBNTHE.gt.0 ) then
1918:           LENGWK = NBNTHE*NETHE*NTTHSC
1919:           LSS = LREALF(IEF,LENGWK,LSS)
1920:           end if
1921:           if ( NPTHIN.gt.0 ) then
1922:           LENGWK = NPTHIN*(NTTHSC+1)
1923:           LSS = LREALF(IEF,LENGWK,LSS)
1924:           end if
1925:           if ( NFGTH.gt.0 ) then
1926:           LENGWK = NFGTH + NTTHSC*(NFGTH*NIGTH+5*MXNGTH)
1927:           LSS = LREALF(IEF,LENGWK,LSS)
1928:           end if
1929:           if ( NPTCUT.gt.0 ) then
1930:           LSS = LREALF(IEF,NPTCUT,LSS)
1931:           end if
1932: C
1933:           end if
1934:           end if
1935: cend
1936:           end if
1937: C .... temperature dependent u.r. probability data
1938:           if ( LENU3R.gt.0 ) then
1939:           LSS = LINTF(IEF,4,LSS)
1940:           LENGWK = LENU3R - 4
1941:           LSS = LREALF(IEF,LENGWK,LSS)
1942:           end if
1943: C
1944: C
1945:           write(6,*) ' lss ', LSS, ' ntdata ', NTDATA
1946:           call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
1947:           end if
1948: C ....
1949: C ....
1950:           if ( LSS-1.ne.0) then

```


src/tools/nlb2txt.f

```

1951:      write(6,*) ' DATA NUMBER MISMATCH (RECORD #3 )'
1952:      write(6,*) ' NTDATA= ', NTDATA, ' TRACED = ', LSS - 1
1953:      end if
1954: C ....
1955: C ....
1956:      call NXTLIN( IRT )
1957:      call CWRITE( 'END-OF-RECORD-3', 0, IRT )
1958:      call NXTLIN( IRT )
1959:      end
1960: C
1961: C
1962: C
1963: C
1964: C
1965: C
1966:      subroutine PMDUMP( MIN, NMT, ADATA, IDATA, IEF, NTDAT3 )
1967: C=====
1968: C PHOTON LIBRARY CONVERTER (MVP)
1969: C
1970: C=====
1971: C
1972:      parameter( MTMAX = 60 )
1973: C
1974:      common /HEAD1/ NPTS, NTDATA, NMT, NSUBS, NSF, NFF,
1975: & NFFE, LSTSF, LSTFF, NPEF, LSPEF, IDUMMY(20),
1976: & AWR, Z, C2M, AK0, R0, R02PA, ELO,
1977: & EHI, ADUMMY(10)
1978:      integer IDAT1(1)
1979:      real EDAT1(1)
1980:      equivalence(NPTS,IDAT1(1)), (NPTS,EDAT1(1))
1981: C
1982:      common /HEAD2/ IDINFO(MTMAX), MT(MTMAX), ISTID(MTMAX),
1983: & IENDID(MTMAX), LSTF3(MTMAX), INT(MTMAX),
1984: & EPE(MTMAX), EFL(MTMAX)
1985:      integer IDAT2(1)
1986:      real EDAT2(1)
1987:      equivalence(IDINFO(1),IDAT2(1)), (IDINFO(1),EDAT2(1))
1988: C
1989: C
1990:      real ADATA(NTDAT3)
1991:      integer IDATA(NTDAT3)
1992:      integer IEF(NTDAT3)
1993: C
1994:      character*8 MATT
1995:      character*8 HDATE
1996:      character*8 TITLE(15)
1997: C
1998: C-----READ INPUT DATA
1999: C
2000:      rewind NIN
2001:      read(NIN) MATT, HDATE, TITLE, NPTS, NTDATA, NMT, NSUBS, NSF, NFF,
2002: & NFFE, LSTSF, LSTFF, NPEF, LSPEF, (IDUMMY(I),I=1,20), AWR,
2003: & Z, C2M, AK0, R0, R02PA, ELO, EHI
2004: C
2005:      read(NIN) IDINFO, MT, ISTID, IENDID, LSTF3, INT, EPE, EFL
2006: C
2007:      if ( NTDATA.gt.NTDAT3 ) then
2008:        write(6,*) ' XXX TOO LARGE #3 DATA (', NTDATA, ') '
2009:        write(6,*) ' LIMIT (NTDAT3) = ', NTDAT3
2010:        stop 999
2011:      end if
2012: C
2013: C
2014:      read(NIN) (ADATA(I),I=1,NTDAT3)
2015: C

```

```

2016:      rewind NIN
2017: C
2018:      do 110 K = 1, 15
2019:        if ( INDEX(TITLE(K),CHAR(0)).ne.0 ) then
2020:          do 100 I = 1, LEN(TITLE(K))
2021:            if ( TITLE(K)(I:I).eq.CHAR(0) ) TITLE(K) (I:I) = ' '
2022:          100 continue
2023:        end if
2024:      110 continue
2025: C
2026: C PRINT OUT LIBRARY INFORMATION
2027: C
2028:      NSYSO = 6
2029:      write(NSYSO,7000) MATT, HDATE, TITLE
2030: C
2031:      7000 format('1',20X,'*****'/'1X,
2032: & 20X,'* MVP MATERIAL LIBRARY INFORMATION *'/'1X,20X,
2033: & '*****'/'1X,20X,
2034: & '* MATERIAL ID = ',A8,' *'/'1X,20X,
2035: & '*****'/'1X,20X,
2036: & '* PRODUCTION DATE : ',A8,' *'/'1X,20X,
2037: & '*****'/'1X,10X,
2038: & ' COMMENT : ',8A8/1X,10X,' : ',7A8//)
2039: C
2040:      write(NSYSO,7020) NPTS, NTDATA, NMT, NSUBS, NSF, NFF, NFFE, LSTSF,
2041: & LSTFF, NPEF, LSPEF, AWR, Z, C2M, AK0, R0, R02PA, ELO, EHI
2042:      7020 format(1X,15X,'NPTS : NO OF SMOOTH DATA GRID RECORD.....',I10/
2043: & 1X,15X,'NTDATA: TOTAL RECORD OF #3 RECORD (WORDS)...',I10/
2044: & 1X,15X,'NMT : NO OF PHOTON RECATIONS .....',I10/
2045: & 1X,15X,'NSUBS : PHOTECTLIC SUBSHELL ABSORPTION ...',I10/
2046: & 1X,15X,'NSF : NO OF SCATTERING FACTOR V (INC.SC.)...',I10/
2047: & 1X,15X,'NFF : NO OF FORM FACTOR ( COH.SC.) .....',I10/
2048: & 1X,15X,'NFFE : NO OF ENERGY POINT FOR FORM FACTOR...',I10/
2049: & 1X,15X,'LSTSF : START OF POSITION OF SCAT.FACTOR....',I10/
2050: & 1X,15X,'LSTFF : START OF POSITION OF FORM FACTOR ...',I10/
2051: & 1X,15X,'NPEF : NO OF FLUORESCENCE DATA .....',I10/
2052: & 1X,15X,'LSPEF : START OF FLUORESCENCE DATA .....',I10/
2053: & 1X,15X,'ATR : ATOMIC WEIGHT IN N.M.U. ....',
2054: & F10.3/1X,15X,
2055: & 'Z : ATOMIC NUMBER .....',F10.3/1X,
2056: & 15X,'C2M : REST MASS OF ELECTRON .....',F10.3/
2057: & 1X,15X,'AK0 : FACTOR (M0C**2 --> 1/ANGSTROM).....',
2058: & F10.3/1X,15X,
2059: & 'R0 : RADIUS OF ELECTRON .....',F10.5/1X,
2060: & 15X,'R02PA : PAI* R**2 .....',F10.3/
2061: & 1X,15X,'ELO : LOWEST ENERGY .....',
2062: & F10.5/1X,15X,
2063: & 'EHI : HIGTEST ENERGY .....',F10.5)
2064: C
2065:      write(NSYSO,7040)
2066:      write(NSYSO,7060)
2067:      write(NSYSO,7080)
2068: C
2069:      7040 format('1',20X,'*****'/'1X,
2070: & 20X,'* #2 RECORD INFORMATION *'/'1X,20X,
2071: & '*****'//)
2072:      7060 format(1X,' MT IDINFO MT ISTID IENDID LSTF3 INT ',
2073: & ' EPE EPL ')
2074:      7080 format(1X,7('-----'))
2075:      7100 format(' ',I3,6I8,1P2E12.5)
2076: C
2077:      do 120 M = 1, NMT
2078:        write(NSYSO,7100) M, IDINFO(M), MT(M), ISTID(M), IENDID(M),
2079: & LSTF3(M), INT(M), EPE(M), EFL(M)
2080:      120 continue

```

src/tools/nlb2txt.f

```

2081:      write(NSYSO,7080)
2082: C
2083: C
2084: C-----
2085: C
2086: C      OUTPUT TO TEXT FILE
2087: C
2088: C-----
2089: C
2090: C
2091: C      call POPEN( NOUT, 80 )
2092: C
2093: C      ... @NUC=NUCLIDE-ID      ....
2094: C
2095: C      call CWRITE( '@NUC= '//MATT, 0, IRT )
2096: C      call NXTLIN( IRT )
2097: C      call CWRITE( '##### TEXT IMAGE MVP LIBRARY (PHOTON) ', 0, IRT )
2098: C      call NXTLIN( IRT )
2099: C
2100: C
2101: C
2102: C --- RECORD #1 ---
2103: C
2104: C
2105: C      .... 17 CHARACTER STRINGS EACH OF THEM HAS 8-CHARACTERS ...
2106: C
2107: C      call CWRITE( '@CHAR 17 ', 0, IRT )
2108: C      call SWRITE( MATT, 8, IRT )
2109: C      call SWRITE( HDATE, 8, IRT )
2110: C      do 130 I = 1, 15
2111: C          call SWRITE( TITLE(I), 8, IRT )
2112: C 130 continue
2113: C
2114: C      call NXTLIN( IRT )
2115: C
2116: C
2117: C      ... NPTS -- IDUMMY(20) ..
2118: C
2119: C      LSS      = 1
2120: C      ND       = 31
2121: C      call IDOUT( IDAT1(LSS), ND, IRT )
2122: C      LSS      = LSS + ND
2123: C
2124: C      ... AWR -- ADUMMY(10)
2125: C
2126: C      ND       = 18
2127: C      call EDOUT( EDAT1(LSS), ND, IRT )
2128: C
2129: C      call NXTLIN( IRT )
2130: C      call CWRITE( 'END-OF-RECORD-1', 0, IRT )
2131: C      call NXTLIN( IRT )
2132: C
2133: C
2134: C --- RECORD #2 ---
2135: C
2136: C      ... IDINFO -- INT      ...
2137: C
2138: C      LSS      = 1
2139: C      ND       = 6*MTMAX
2140: C      call IDOUT( IDAT2(LSS), ND, IRT )
2141: C      LSS      = LSS + ND
2142: C
2143: C      ... EPE & EFL      ....
2144: C
2145: C      ND       = 2*MTMAX

```

```

2146:      call EDOUT( EDAT2(LSS), ND, IRT )
2147: C      LSS      = LSS + ND
2148: C
2149: C      call NXTLIN( IRT )
2150: C      call CWRITE( 'END-OF-RECORD-2', 0, IRT )
2151: C
2152: C
2153: C      write(6,*) ' == RECORD-2: NUMBER OF DATA == ', LSS - 1
2154: C
2155: C
2156: C --- RECORD #3 ---
2157: C
2158: C
2159: C
2160: C      call NXTLIN( IRT )
2161: C      call CWRITE( '# -- ENERGY MESH DATA      ---- ', 0, IRT )
2162: C      call NXTLIN( IRT )
2163: C
2164: C      LSS      = 1
2165: C      L0       = LSS
2166: C      LSS      = LREALF(IEF,NPTS,LSS)
2167: C      LSS      = LREALF(IEF,NPTS,LSS)
2168: C      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2169: C
2170: C      call NXTLIN( IRT )
2171: C      call CWRITE( '# -- SMOOTH CROSS SECTION ---- ', 0, IRT )
2172: C      call NXTLIN( IRT )
2173: C
2174: C      L0       = LSS
2175: C      LSTC     = 0
2176: C      LSS      = LREALF(IEF,LSTSF-L0,LSS)
2177: C
2178: C      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2179: C
2180: C
2181: C      call NXTLIN( IRT )
2182: C      call CWRITE( '# -- SCATTERING FACTOR ----- ', 0, IRT )
2183: C      call NXTLIN( IRT )
2184: C      L0       = LSS
2185: C
2186: C      write(6,*) ' L0 LSTSF      ', L0, LSTSF
2187: C
2188: C      LSS      = LREALF(IEF,2*NSF,LSS)
2189: C      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2190: C
2191: C      write(6,*) ' LSS LSTFF      ', LSS, LSTFF
2192: C**      IF( LSS .NE. LSTFF ) THEN
2193: C**          WRITE(6,*) ' INVALID POSITION : (ENRGY DIST.DATA) ',
2194: C**      &          LSS, ' ( LSTF5(1)= ',LSTF5(1), ' )'
2195: C**      ENDIF
2196: C
2197: C
2198: C      call NXTLIN( IRT )
2199: C      call CWRITE( '# -- FORM FACTOR ----- ', 0, IRT )
2200: C      call NXTLIN( IRT )
2201: C      L0       = LSS
2202: C
2203: C      LSS      = LREALF(IEF,NFFE+2*NFF+2*NFFE*NFF,LSS)
2204: C
2205: C      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2206: C
2207: C      write(6,*) ' LSS LSPEF ', LSS, LSPEF
2208: C      if ( LSS.ne.LSPEF ) then
2209: C          write(6,*) ' INVALID POSITION : (BEFORE FLUORESCENCE) ', LSS,
2210: C      &          ' ( LSPEF = ', LSPEF, ' )'

```

src/tools/nlb2txt.f

```

2211:      end if
2212: C
2213: C
2214:      call NXTLIN( IRT )
2215:      call CWRITE( '# -- FLUORESCENCE DATA ----- ', 0, IRT )
2216:      call NXTLIN( IRT )
2217:      L0      = LSS
2218: C
2219:      NEDGE   = IDATA(LSS)
2220: C
2221:      LSS     = LINTF(IEF,1,LSS)
2222:      LSS     = LREALF(IEF,NEDGE,LSS)
2223:      ND1     = IDATA(LSS)
2224:      ND2     = IDATA(LSS+1)
2225:      LSS     = LINTF(IEF,2*NEDGE+1,LSS)
2226: C
2227:      LSS     = LREALF(IEF,ND1,LSS)
2228:      LSS     = LINTF(IEF,2*ND1,LSS)
2229:      LSS     = LREALF(IEF,ND2,LSS)
2230:      LSS     = LINTF(IEF,2*ND2,LSS)
2231: C
2232:      LSS     = LREALF(IEF,4*NPEF,LSS)
2233: C
2234:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2235: C
2236: C ....
2237: C ....
2238:      if ( LSS-1.ne.NTDATA ) then
2239:        write(6,*) ' DATA NUMBER MISMATCH (RECORD #3 )'
2240:        write(6,*) ' NTDATA= ', NTDATA, ' TRACED = ', LSS - 1
2241:      end if
2242: C ....
2243: C ....
2244:      call NXTLIN( IRT )
2245:      call CWRITE( 'END-OF-RECORD-3', 0, IRT )
2246:      call NXTLIN( IRT )
2247:      end
2248: C
2249: C
2250: C
2251: C
2252: C
2253: C
2254:      subroutine EMDUMP( NIN,   NOUT,  ADATA, IDATA, IEF,   NTDAT3 )
2255: C
2256: C=====
2257: C ELECTRON LIBRARY CONVERTER (MVP)
2258: C
2259: C=====
2260: C
2261: C < CROSS SECTION DATA FOR EACH ATOMIC ELEMENT >
2262: C
2263: C KLBE1(20)      I4  RECORD #1 (SPECIFICATION RECORD)
2264: C
2265: C      1:  NTDATA  LENGTH OF RECORD #2
2266: C      2:  NRAD   LENGTH OF TR AND RR TABLES (USUALLY 49)
2267: C      3:  NMOT   LENGTH OF ER AND RS TABLES (USUALLY 18)
2268: C      4:  NCOR   LENGTH OF TC AND CP TABLES (USUALLY 32)
2269: C      5:  NHFL   LENGTH OF TH AND FL TABLES (USUALLY 39)
2270: C      6:  NEE    NUMBER OF ENERGY GRIDS FOR ELECTRON CROSS SECTION
2271: C              DATA (USUALLY 134)
2272: C      7:  MTOP   NUMBER OF BREMSSTRAHLUNG PHOTON/ELECTRON (K/T)
2273: C              RATIOS (USUALLY 49)
2274: C      8:  NEK    NUMBER OF ENERGY GRIDS FOR EK TABLE (USUALLY 9)
2275: C      9:  NPL    NUMBER OF THE LEGENDRE SERIES FOR GK TABLE

```

```

2276: C              (USUALLY 240)
2277: C      10: LSTC   STARTING POSITION OF DATA FOR AN ATOMIC ELEMENT
2278: C      11: LSTTR  STARTING POSITION OF TR TABLE (LSTC+4)
2279: C      12: LSTER  STARTING POSITION OF ER TABLE (LSTTR+NRAD*2)
2280: C      13: LSTEC  STARTING POSITION OF EK TABLE (LSTER+NMOT*6)
2281: C      14: LSTTC  STARTING POSITION OF TC TABLE (LSTER+NEK*14)
2282: C      15: LSTTH  STARTING POSITION OF TH TABLE (LSTTC+NCOR*2)
2283: C      16: LSTEEL STARTING POSITION OF ENERGY GRIDS FOR ELECTRON
2284: C              CROSS SECTION DATA, IN EV (LSTTH+NHFL*2)
2285: C      17: LSTPBR STARTING POSITION OF BREMSSTRAHLUNG CROSS SECTIONS
2286: C              (LSTEEL+NEE)
2287: C      18: LSTRKT STARTING POSITION OF K/T RATIOS (LSTPBR+NEE)
2288: C      19: LSTEB  STARTING POSITION OF BREMSSTRAHLUNG CUMULATIVE
2289: C              PROBABILITIES, EBA (LSTRKT+MTOP)
2290: C      20: LSTGK  STARTING POSITION OF GK TABLE (LSTEB+NEE*MTOP)
2291: C
2292: C XLBE1(8)      R4  ATOMIC PARAMETERS
2293: C
2294: C      1: AWR    ATOMIC WEIGHT
2295: C      2: Z      ATOMIC NUMBER
2296: C      3: Z13    Z**(1/3)
2297: C      4: ZLN    LOG(Z)
2298: C      5: CCL    BREMSSTRAHLUNG COULOMB CORRECTION FACTOR
2299: C      6: TZ     PROCESSING TEMPERATURE OF CROSS SECTIONS, IN EV
2300: C      7: ETOPE  UPPER ENERGY LIMIT IN ELECTRON LIBRARY
2301: C      8* EBOTE  LOWER ENERGY LIMIT IN ELECTRON LIBRARY
2302: C=====
2303: C
2304: C
2305: C      common /HEAD1/  NTDATA, NRAD,  NMOT,  NCOR,  NHFL,  NEE,
2306: C      &               MTOP,  NEK,    NPL,   IDUM(2),  LSTC,  LSTTR,
2307: C      &               LSTER,  LSTEC,  LSTTC,  LSTTH,  LSTEEL,  LSTPBR,  LSTRKT,
2308: C      &               LSTEB,  LSTGK,  JDUM(5),  AWR,  Z,  Z13,
2309: C      &               ZLN,   CCL,   TZ,    ELOWE,  EHIGHE
2310: C
2311: C      real EDAT1(1)
2312: C      integer IDAT1(1)
2313: C      equivalence(NTDATA,IDAT1(1)), (NTDATA,EDAT1(1))
2314: C
2315: C      real ADATA(NTDAT3)
2316: C      integer IDATA(NTDAT3)
2317: C      integer IEF(NTDAT3)
2318: C
2319: C      character*8 MATT
2320: C      character*8 PDATE
2321: C      character*8 TITLE(15)
2322: C
2323: C-----
2324: C SET INITIAL DATA
2325: C-----
2326: C
2327: C      rewind NIN
2328: C
2329: C-----
2330: C READ SPECIFICATION RECORD #1
2331: C-----
2332: C
2333: C      read(NIN) MATT, PDATE, TITLE, NTDATA, NRAD, NMOT, NCOR, NHFL, NEE,
2334: C      &         MTOP, NEK, NPL, (IDUM(I),I=1,2), LSTC, LSTTR, LSTER,
2335: C      &         LSTEC, LSTTC, LSTTH, LSTEEL, LSTPBR, LSTRKT, LSTEB,
2336: C      &         LSTGK, (JDUM(I),I=1,5), AWR, Z, Z13, ZLN, CCL, TZ, ELOWE,
2337: C      &         EHIGHE
2338: C
2339: C
2340: C      if ( NTDATA.gt.NTDAT3 ) then

```

src/tools/nlb2txt.f

```

2341:      write(6,*) ' XXX TOO LARGE #2 DATA (', NTDATA, ')'
2342:      write(6,*) ' LIMIT (NTDAT3) = ', NTDAT3
2343:      stop 999
2344:  end if
2345: C
2346:      NSYSO      = 6
2347: C
2348: C-----
2349: C  READ RECORD #2 (CROSS SECTION DATA)
2350: C-----
2351: C
2352:      read(NIN) (ADATA(I),I=1,NTDATA)
2353: C
2354: C
2355: C-----
2356: C  PRINT OUT OF INFORMATION FOR ELECTRON CROSS SECTION LIBRARY
2357: C-----
2358: C
2359:      write(NSYSO,7000) MATT, PDATE, TITLE
2360:      write(NSYSO,7020) NTDATA, NRAD, NMOT, NCOR, NHFL, NEE, MTOP, NEK,
2361:      &      NPL
2362:      write(NSYSO,7040) LSTC, LSTTR, LSTER, LSTTK, LSTTC, LSTTH, LSTEEL,
2363:      &      LSTPBR, LSTRKT, LSTEB, LSTGK
2364:      write(NSYSO,7060) AWR, Z, Z13, ZLN, CCL, TZ, BLOWE, EHIGHE
2365: 7000 format(/1X,20X,'*****'/'1X,
2366:      &      20X,' MVP MATERIAL LIBRARY INFORMATION '/*'/1X,20X,
2367:      &      '*****'/'1X,20X,
2368:      &      '* MATERIAL ID = ',A8,' '/*'/1X,20X,
2369:      &      '*****'/'1X,20X,
2370:      &      '* PRODUCTION DATE : ',A8,' '/*'/1X,20X,
2371:      &      '*****'/'1X,10X,
2372:      &      ' COMMENT : ',8A8/1X,10X,' : ',7A8//)
2373: 7020 format(1X,15X,'NTDATA: LENGTH OF RECORD #2 .....',I10/
2374:      &      1X,15X,'NRAD : LENGTH OF TR AND RR TABLES .....',I10/
2375:      &      1X,15X,'NMOT : LENGTH OF ER AND RS TABLES .....',I10/
2376:      &      1X,15X,'NCOR : LENGTH OF TC AND CP TABLES .....',I10/
2377:      &      1X,15X,'NHFL : LENGTH OF TH AND FL TABLES .....',I10/
2378:      &      1X,15X,'NEE : NO OF ENERGY GRIDS FOR EL. CX .....',I10/
2379:      &      1X,15X,'MTOP : NO OF BREMSSTRAHLUNG (K/T) RATIOS .....',I10/
2380:      &      1X,15X,'NEK : NO OF ENERGY GRIDS FOR EK TABLE .....',I10/
2381:      &      1X,15X,'NPL : NO OF THE PL SERIES FOR GK TABLE .....',I10)
2382: 7040 format(1X,15X,'LSTC : STARTING POSITION OF ATOMIC DATA .....',I10/
2383:      &      1X,15X,'LSTTR : STARTING POSITION OF TR TABLE .....',I10/
2384:      &      1X,15X,'LSTER : STARTING POSITION OF ER TABLE .....',I10/
2385:      &      1X,15X,'LSTTK : STARTING POSITION OF EK TABLE .....',I10/
2386:      &      1X,15X,'LSTTC : STARTING POSITION OF TC TABLE .....',I10/
2387:      &      1X,15X,'LSTTH : STARTING POSITION OF TH TABLE .....',I10/
2388:      &      1X,15X,'LSTEEL: STARTING POSITION OF ENERGY GRIDS .....',I10/
2389:      &      1X,15X,'LSTPBR: STARTING POSITION OF BREMS CX .....',I10/
2390:      &      1X,15X,'LSTRKT: STARTING POSITION OF K/T RATIOS .....',I10/
2391:      &      1X,15X,'LSTEB: STARTING POSITION OF EBA DATA .....',I10/
2392:      &      1X,15X,'LSTGK : STARTING POSITION OF GK TABLE .....',I10)
2393: 7060 format(/1X,15X,'AWR : ATOMIC WEIGHT .....',F10.3
2394:      &      /1X,15X,'Z : ATOMIC NUMBER .....',
2395:      &      F10.3/1X,15X,
2396:      &      'Z13 : Z**(1/3) .....',F10.3/1X,
2397:      &      15X,'ZLN : LOG(Z) .....',F10.3/
2398:      &      1X,15X,'CCL : BREMS COULOMB CORRECTION FACTOR .....',F10.3
2399:      &      /1X,15X,'TZ : PROCESSING TEMPERATURE IN EV .....',
2400:      &      F10.3/1X,15X,
2401:      &      'ETOPE: UPPER ENERGY LIMIT IN LIBRARY .....',F10.3/1X,
2402:      &      15X,'EBOTE: LOWER ENERGY LIMIT IN LIBRARY .....',F10.3)
2403: C
2404: C
2405: C

```

```

2406: C-----
2407: C
2408: C      OUTPUT TO TEXT FILE
2409: C
2410: C-----
2411: C
2412: C
2413:      call POPEN( NOUT, 80 )
2414: C
2415: C .... @NUC=NUCLIDE-ID .....
2416: C
2417:      call CWRITE( '@NUC='//MATT, 0, IRT )
2418:      call NXTLIN( IRT )
2419:      call CWRITE( '***** TEXT IMAGE MVP LIBRARY (ELECTRON)', 0, IRT )
2420:      call NXTLIN( IRT )
2421: C
2422: C
2423: C
2424: C --- RECORD #1 ---
2425: C
2426: C
2427: C ..... 17 CHARACTER STRINGS EACH OF THEM HAS 8-CHARACTERS ...
2428: C
2429:      call CWRITE( '@CHAR 17 8 ', 0, IRT )
2430:      call SWRITE( MATT, 8, IRT )
2431:      call SWRITE( PDATE, 8, IRT )
2432:      do 100 I = 1, 15
2433:          call SWRITE( TITLE(I), 8, IRT )
2434:      100 continue
2435: C
2436:      call NXTLIN( IRT )
2437: C
2438: C
2439: C ... NTDATA - JDUM(5) ....
2440: C
2441:      LSS      = 1
2442:      ND       = 27
2443:      call IDOUT( IDAT1(LSS), ND, IRT )
2444:      LSS      = LSS + ND
2445: C
2446: C ... AWR -- EHIGHE
2447: C
2448:      ND       = 8
2449:      call EDOUT( EDAT1(LSS), ND, IRT )
2450: C
2451:      call NXTLIN( IRT )
2452:      call CWRITE( 'END-OF-RECORD-1', 0, IRT )
2453:      call NXTLIN( IRT )
2454: C
2455: C
2456: C --- RECORD #2 ---
2457: C
2458:      LSS      = 1
2459:      ND       = NTDATA
2460:      call EDOUT( ADATA(LSS), ND, IRT )
2461:      LSS      = LSS + ND
2462: C
2463:      call NXTLIN( IRT )
2464:      call CWRITE( 'END-OF-RECORD-2', 0, IRT )
2465: C
2466: C
2467:      write(6,*) ' == RECORD-2: NUMBER OF DATA =', LSS - 1
2468: C
2469: C --- RECORD #3 ---
2470: C

```

src/tools/nlb2txt.f

```

2471: C      NO DATA
2472: C
2473: C ....
2474:       call NXTLIN( IRT )
2475:       call CWRITE( 'END-OF-RECORD-3', 0, IRT )
2476:       call NXTLIN( IRT )
2477: C
2478:       return
2479:       end
2480: C
2481: C      subroutine PNMDUMP( NIN, MOUT, ADATA, IDATA, IEF, NTDAT3 )
2482: C=====
2483: C      PHOTONUCLEAR LIBRARY CONVERTER (MVP)
2484: C=====
2485: C      parameter( MTMAX = 135 )
2486: C
2487: C      common /HEAD1G/ NPTS, NTDATA, NMT2G, NBINA, NNU, NMT,
2488: & LFI, LNU, LSNU, NNUD, LNUD, NNF, LSNUD,
2489: & LSTPEA, KDUMMY(20),
2490: & ATW, TLAB, ELOW, EHI, RDUMMY(15)
2491: C
2492:       integer IDAT1(1)
2493:       real EDAT1(1)
2494:       equivalence(NPTS,IDAT1(1)), (NPTS,EDAT1(1))
2495: C
2496:       common /HEAD2G/ MTINFO(MTMAX), MTPAR(MTMAX), ISTMT(MTMAX),
2497: & IENDMT(MTMAX), LCTMT(MTMAX), NEANG(MTMAX),
2498: & NK2G(MTMAX), NE2G(MTMAX), MT2G(MTMAX),
2499: & LSTF3(MTMAX), LSTF4(MTMAX), LST2G(MTMAX),
2500: & QVAL(MTMAX), QVAL2(MTMAX)
2501: C
2502:       integer IDAT2(1)
2503:       real EDAT2(1)
2504:       equivalence(MTINFO(1),IDAT2(1)), (MTINFO(1),EDAT2(1))
2505: C
2506:       real ADATA(NTDAT3)
2507:       integer IDATA(NTDAT3), IEF(NTDAT3)
2508: C
2509:       character*16 MATT
2510:       character*8 TITLE(14), HDATE
2511:       character CTEMP*128
2512: C
2513: C-----READ INPUT DATA
2514: C===== RECORD 1 =====
2515: C
2516:       rewind NIN
2517:       read(NIN) MATT, TITLE, HDATE, NPTS, NTDATA, NMT2G, NBINA, NNU,
2518: & NMT, LFI, LNU, LSNU, NNUD, LNUD, NNF, LSNUD, LSTPEA,
2519: & KDUMMY,
2520: & ATW, TLAB, ELOW, EHI, RDUMMY
2521: C
2522:       ZA = RDUMMY(1)
2523:       MATH = RDUMMY(2)
2524: C
2525: C===== RECORD 2 =====
2526: C
2527:       read(NIN) MTINFO, MTPAR, ISTMT, IENDMT, LCTMT, NEANG, NK2G, NE2G,
2528: & MT2G, LSTF3, LSTF4, LST2G, QVAL, QVAL2
2529: C
2530:       if ( NTDATA.gt.NTDAT3 ) then
2531:         write(6,*) ' XXX TOO LARGE #3 DATA (', NTDATA, ')'
2532:         write(6,*) ' LIMIT (NTDAT3) = ', NTDAT3
2533:         stop 999
2534:       end if
2535:       NSYSO = 6

```

```

2536: C
2537: C===== RECORD 3 =====
2538: C
2539:       read(NIN) (ADATA(I),I=1,NTDATA)
2540: C
2541:       do K = 1, 15
2542:         if ( INDEX(TITLE(K),CHAR(0)).ne.0 ) then
2543:           do I = 1, LEN(TITLE(K))
2544:             if ( TITLE(K)(I:I).eq.CHAR(0) ) TITLE(K) (I:I) = ' '
2545:           end do
2546:         end if
2547:       end do
2548: C
2549: C .... PRINT OUT LIBRARY INFORMATION
2550: C
2551:       write(NSYSO,7000) MATT, HDATE, TITLE
2552:       write(NSYSO,7020) NPTS, NTDATA, NMT2G, NBINA, NNU, NMT, LFI, LNU,
2553: & LSNU, NNUD, LNUD, NNF, LSNUD, LSTPEA
2554:       write(NSYSO,7100) ATW, TLAB, ELOW, EHI, ZA, MATH
2555: C
2556: 7000 format(
2557: &/21X,'*****'
2558: &/21X,' MVP MATERIAL LIBRARY INFORMATION '
2559: &/21X,'*****'
2560: &/21X,' MATERIAL ID = ',A16, ' '
2561: &/21X,'*****'
2562: &/21X,' PRODUCTION DATE : ',A8, ' '
2563: &/21X,'*****'
2564: &/11X,' COMMENT : ',8A8 /11X,' : ',6A8 //)
2565: 7020 format(
2566: & 16X,'NPTS : NO. OF SMOOTH DATA GRID RECORD.....',I10
2567: &/16X,'NTDATA: TOTAL RECORD OF #3 RECORD (WORDS).....',I10
2568: &/16X,'NMT2G : NO. OF REACTIONS PROD. 2ND PARTICLE....',I10
2569: &/16X,'NBINA : NO. OF BINS IN ANGULAR DIS. TABLE.....',I10
2570: &/16X,'NNU : LENGTH OF NU-DATA.....',I10
2571: &/16X,'NMT : TOTAL NO OF REACTION CONSIDERED.....',I10
2572: &/16X,'LFI : FLAG OF FISSION REACTION.....',I10
2573: &/16X,'LNU : NU-DATA FLAG (1/2=POLY/TAB).....',I10
2574: &/16X,'LSNU : START OF POSITION OF NU-DATA.....',I10
2575: &/16X,'NNUD : LENGTH OF DELAYED NU-DATA.....',I10
2576: &/16X,'LNUD : DELAYED NU-DATA FLAG(1/2=POLY/TAB).....',I10
2577: &/16X,'NNF : NO OF PRECURSOR FAMILIES.....',I10
2578: &/16X,'LSNUD : START OF POSITION OF DELAYED NU.....',I10
2579: &/16X,'LSTPEA: START OF POSITION OF PRODUCT ENG.-ANG..',I10)
2580: 7100 format(
2581: & 16X,'ATW : ATOMIC WEIGHT IN N.M.U. ....',F12.3
2582: &/16X,'TLAB : LABORATORY TEMPERATURE IN KELVIN.....',F12.3
2583: &/16X,'ELOW : LOWER ENERGY DEFINED (EV).....',1PE12.5
2584: &/16X,'EHI : UPPER ENERGY DEFINED (EV).....',E12.5
2585: &/16X,'ZA : The (Z&A) designation.....',0PF12.1
2586: &/16X,'MATNO : Material number in ENDF.....',I12/)
2587: C
2588:       write(NSYSO,7180)
2589:       write(NSYSO,7200)
2590:       write(NSYSO,7220)
2591:       do MT = 1, NMT
2592:         if ( MTINFO(MT).gt.0 ) then
2593:           write(NSYSO,7240) MT, MTINFO(MT), ISTMT(MT), IENDMT(MT),
2594: & MTPAR(MT), QVAL(MT), QVAL2(MT)
2595:         end if
2596:       end do
2597:       write(NSYSO,7220)
2598: C
2599: 7180 format(
2600: &/21X,'*****'

```

src/tools/nlb2txt.f

```

2601:      &/21X,'*   #2 RECORD INFORMATION (PART 1)   *'
2602:      &/21X,'*****'
2603:      7200 format(11X,' MT   MTINFO ISTMT   IENDMT MTPAR   ',
2604:      &      ' QVAL(EV)   QVAL2(EV)   ')
2605:      7220 format(10X,6('-----'))
2606:      7240 format(11X,I3,4I8,1P2E12.5)
2607: C
2608:      write(NSYSO,7260)
2609:      write(NSYSO,7280)
2610:      write(NSYSO,7300)
2611:      do MT = 1, NMT
2612:      if ( MTINFO(MT).gt.0.or.NK2G(MT).re.0 ) then
2613:      write(NSYSO,7320) MT, LSTF3(MT), LCTMT(MT), NEANG(MT),
2614:      &      LSTF4(MT), NK2G(MT), NE2G(MT), LST2G(MT)
2615:      end if
2616:      end do
2617:      write(NSYSO,7300)
2618: C
2619:      7260 format(
2620:      &/21X,'*****'
2621:      &/21X,'*   #2 RECORD INFORMATION (PART 2)   *'
2622:      &/21X,'*****'
2623:      7280 format(' ',10X,' MT   LSTF3   LCTMT   NEANG   LSTF4   NK2G
2624:      &      'NE2G   LST2G   ')
2625:      7300 format(' ',9X,7('-----'))
2626:      7320 format(' ',10X,I3,8I8)
2627: C
2628: C-----
2629: C
2630: C      OUTPUT TO TEXT FILE
2631: C
2632: C-----
2633: C
2634: C .... SET RECORD LENGTH OF OUTPUT FILE ...(80 CHARACTER)
2635: C
2636:      call POPEN( NOUT, 80 )
2637: C
2638: C .... @NUC=NUCLIDE-ID          .....
2639: C
2640:      CTEMP   = '@NUC='//MATT
2641:      LCT     = INDEX(CTEMP,' ') - 1
2642:      if ( LCT.lt.0 ) LCT = LEN(CTEMP)
2643:      call CWRITE( CTEMP(:LCT), 0, IRT )
2644:      call NXTLIN( IRT )
2645:      call CWRITE( '#**** TEXT IMAGE MVP LIBRARY (PHOTONUCLEAR)', 0, IRT )
2646:      call NXTLIN( IRT )
2647: C
2648: C-----
2649: C --- RECORD #1 ---
2650: C-----
2651: C
2652: C      .... 17 CHARACTER STRINGS EACH OF THEM HAS 8-CHARACTERS ...
2653: C
2654:      call CWRITE( '@CHAR 17 8 ', 0, IRT )
2655:      call SWRITE( MATT, 16, IRT )
2656:      call SWRITE( MATT(1:8), 8, IRT )
2657:      call SWRITE( MATT(9:16), 8, IRT )
2658:      do I = 1, 14
2659:      call SWRITE( TITLE(I), 8, IRT )
2660:      end do
2661:      call SWRITE( HDATE, 8, IRT )
2662:      call NXTLIN( IRT )
2663: C
2664: C ... NPTS -- KDUMMY(8) ... (INTEGER DATA)
2665: C

```

```

2666:      LSS     = 1
2667:      ND      = 34
2668:      call IDOUT( IDAT1(LSS), ND, IRT )
2669:      LSS     = LSS + ND
2670: C
2671: C ... ATW -- RDUMMY(12) ... (REAL DATA)
2672: C
2673:      ND      = 19
2674:      call EDOUT( EDAT1(LSS), ND, IRT )
2675:      call NXTLIN( IRT )
2676:      call CWRITE( 'END-OF-RECORD-1', 0, IRT )
2677:      call NXTLIN( IRT )
2678: C
2679: C-----
2680: C --- RECORD #2 ---
2681: C-----
2682: C
2683: C ... MTINFO -- LST2G ...
2684: C
2685:      LSS     = 1
2686:      ND      = 12*MTMAX
2687:      call IDOUT( IDAT2(LSS), ND, IRT )
2688:      LSS     = LSS + ND
2689: C
2690: C ... QVAL & QVAL2 .....
2691: C
2692:      ND      = 2*MTMAX
2693:      call EDOUT( EDAT2(LSS), ND, IRT )
2694:      LSS     = LSS + ND
2695: C
2696:      call NXTLIN( IRT )
2697:      call CWRITE( 'END-OF-RECORD-2', 0, IRT )
2698:      call NXTLIN( IRT )
2699:      write(6,*) ' == RECORD-2: NUMBER OF DATA =', LSS - 1
2700: C
2701: C-----
2702: C --- RECORD #3 ---
2703: C-----
2704: C
2705: C IEF(I) : FLAG THAT INDICATES DATA TYPE (INTEGER/REAL) OF ADATA(I)
2706: C
2707: C***** ENERGY MESH & NU *****
2708: C
2709:      call CWRITE( '# -- ENERGY MESH & NU ---- ', 0, IRT )
2710:      call NXTLIN( IRT )
2711:      LSS     = 1
2712:      L0      = LSS
2713:      LSS     = LREALF(IEF,NPTS,LSS) ! EMESH
2714:      LSS     = LREALF(IEF,LNU*NNU,LSS) ! XNU
2715:      LSS     = LREALF(IEF,LNUD*NNUD,LSS) ! XNUD
2716:      LSS     = LREALF(IEF,NNF,LSS) ! RAMDA
2717:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2718: C
2719: C***** CROSS SECTION *****
2720: C
2721:      call NXTLIN( IRT )
2722:      call CWRITE( '# -- SMOOTH CROSS SECTION ---- ', 0, IRT )
2723:      call NXTLIN( IRT )
2724:      L0      = LSS
2725:      LSS     = LREALF(IEF,LSTF4(1)-LSTF3(1),LSS) ! CROSS
2726:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2727: C
2728: C***** ANGULAR DISTRIBUTION DATA *****
2729: C
2730:      call NXTLIN( IRT )

```

src/tools/nlb2txt.f

```

2731:      call CWRITE( '# -- ANGULAR DISTRIBUTION ---- ', 0, IRT )
2732:      call NXTLIN( IRT )
2733:      NBINA1 = NBINA + 1
2734:      L0     = LSS
2735: C
2736:      do M = 1, NMT
2737:        if ( NEANG(M).gt.0 ) then
2738:          LSS = LREALF(IEF,NEANG(M),LSS)      ! EANG
2739:          LSS = LINTF(IEF,NEANG(M),LSS)        ! INTANG
2740:          LSS = LREALF(IEF,NBINA1*NEANG(M),LSS) ! UMTAB
2741:        end if
2742:      end do
2743:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2744: C
2745: C***** ENERGY DISTRIBUTION DATA *****
2746: C
2747:      call NXTLIN( IRT )
2748:      call CWRITE( '# -- ENERGY DISTRIBUTION ---- ', 0, IRT )
2749:      call NXTLIN( IRT )
2750:      L0 = LSS
2751:      LSS = LINTF(IEF,NMT,LSS)      ! LST2G
2752:      LSS = LINTF(IEF,NMT+2,LSS)    ! MT2G, NMT2G & NMT2GP
2753: C
2754:      do M = 1, NMT
2755:        if ( NK2G(M).ne.0 ) then
2756:          write(6,*) ' M = LSS LST2G(M) ', M, LSS, LST2G(M)
2757:          if ( NE2G(M).ne.IDATA(LSS) ) .or.
2758:            & abs(NK2G(M)).ne.IDATA(LSS+1) ) then
2759:            write(6,*) '!!! SIZE UNMATCH BETWEEN RECORD #2 & #3'
2760:            write(6,*) 'MT=', M, ' NE2G ', NE2G(M), ' (#2) ',
2761:              & IDATA(LSS), ' (#3)'
2762:            write(6,*) 'MT=', M, ' NK2G ', NK2G(M), ' (#2) ',
2763:              & IDATA(LSS+1), ' (#3)'
2764:          stop 888
2765:        end if
2766: C
2767:        ISWF6 = 0
2768:        if ( NK2G(M).lt.0 ) ISWF6 = 1
2769:        NK2G(M) = IABS(NK2G(M))
2770:        LSS = LINTF(IEF,2,LSS)      ! NE2G & NK2G
2771:        LSS = LINTF(IEF,60,LSS)    ! NK2GG & KSHBS
2772:        LSS = LINTF(IEF,NK2G(M),LSS) ! LST2GS
2773:        LSS = LREALF(IEF,2*NK2G(M),LSS) ! ZAP & RWP
2774:        LSS = LINTF(IEF,2*NK2G(M),LSS) ! LIP & LCT
2775:        if ( ISWF6.eq.0 ) then
2776:          LSS = LREALF(IEF,NE2G(M),LSS) ! EPROB
2777:          LSS = LINTF(IEF,NE2G(M),LSS) ! INTF5
2778:          do NK = 1, NE2G(M)
2779:            write(6,*) ' nk=', NK, ' nk2g ', NK2G(M)
2780:            LSS = LREALF(IEF,NK2G(M),LSS) ! QF5
2781:            LSS = LINTF(IEF,2*NK2G(M),LSS) ! INK1 & INK2
2782:          end do
2783:          LSS = LINTF(IEF,1,LSS)      ! Y5
2784:        else
2785:          LSS = LREALF(IEF,NE2G(M),LSS) ! EPRO2G
2786:          LSS = LINTF(IEF,NE2G(M),LSS) ! INT2G
2787:          LSS = LREALF(IEF,NE2G(M),LSS) ! TY2G
2788:          do NK = 1, NK2G(M)
2789:            write(6,*) ' nk=', NK, ' ne2g ', NE2G(M)
2790:            LSS = LREALF(IEF,NE2G(M),LSS) ! Y2G
2791:          end do
2792:        end if
2793: C
2794: C ... ENERGY DISTRIBUTION CLASSIFIED BY LF VALUE ...
2795: C
2796:      do NK = 1, NK2G(M)
2797:        LF = IDATA(LSS)
2798:        write(6,*) ' LF = ', LF, ' NK = ', NK
2799:        if ( LF.eq.1 ) then
2800:          NEF5S = IDATA(LSS+1)
2801:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2802:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2803:          LSS = LINTF(IEF,2*NEF5S,LSS) ! INTENG & LSTF5E
2804:          do J = 1, NEF5S
2805:            NEPI = IDATA(LSS)
2806:            LSS = LINTF(IEF,1,LSS)    ! NEP
2807:            LSS = LREALF(IEF,NEPI,LSS) ! Q
2808:            LSS = LINTF(IEF,2*NEPI,LSS) ! IEPOS1 & IEPOS2
2809:            LSS = LREALF(IEF,2*(NEPI+1),LSS) ! EBINF & EPRB
2810:          end do
2811: C
2812:        else if ( LF.eq.5 ) then
2813:          NEF5S = IDATA(LSS+1)
2814:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2815:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2816:          LSS = LINTF(IEF,NEF5S,LSS) ! INTENG
2817:          LSS = LREALF(IEF,NEF5S,LSS) ! THETA5
2818:          NEP = IDATA(LSS)
2819:          LSS = LINTF(IEF,1,LSS)      ! NEP
2820:          LSS = LREALF(IEF,NEP,LSS)  ! Q
2821:          LSS = LINTF(IEF,2*NEP,LSS) ! IEPOS1 & IEPOS2
2822:          LSS = LREALF(IEF,2*(NEP+1)+1,LSS) ! EBINF, EPRBF & U5
2823: C
2824:        else if ( LF.eq.7 ) then
2825:          NEF5S = IDATA(LSS+1)
2826:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2827:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2828:          LSS = LINTF(IEF,NEF5S,LSS) ! INTENG
2829:          LSS = LREALF(IEF,NEF5S+1,LSS) ! THETA7 & U7
2830: C
2831:        else if ( LF.eq.9 ) then
2832:          NEF5S = IDATA(LSS+1)
2833:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2834:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2835:          LSS = LINTF(IEF,NEF5S,LSS) ! INTENG
2836:          LSS = LREALF(IEF,NEF5S+1,LSS) ! THETA9 & U9
2837: C
2838:        else if ( LF.eq.11 ) then
2839:          NEF5S = IDATA(LSS+1)
2840:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2841:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2842:          LSS = LINTF(IEF,NEF5S,LSS) ! INTENG
2843:          LSS = LREALF(IEF,2*NEF5S+1,LSS) ! THETA, THETAB & UA
2844: C
2845:        else if ( LF.eq.60 ) then
2846:          LSS = LINTF(IEF,1,LSS)      ! LF
2847: C
2848:        else if ( LF.eq.61 ) then
2849:          NEF5S = IDATA(LSS+1)
2850:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2851:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2852:          LSS = LINTF(IEF,NEF5S,LSS) ! INTENG
2853:          LSS = LINTF(IEF,NEF5S,LSS) ! LSTF5E
2854:          do J = 1, NEF5S
2855:            NEPI = IDATA(LSS)
2856:            LSS = LINTF(IEF,2,LSS)    ! NEP & ND
2857:            LSS = LREALF(IEF,NEPI,LSS) ! Q
2858:            LSS = LINTF(IEF,2*NEPI,LSS) ! IEPOS1 & IEPOS2
2859:            LSS = LREALF(IEF,4*(NEPI+1),LSS) ! EBINF, EPRBF

```

src/tools/nlb2txt.f

```

, PRECF & ANGDS
2860:      end do
2861: C
2862:      else if ( LF.eq.62 ) then
2863:          NEF5S = IDATA(LSS+1)
2864:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2865:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2866:          LSS = LINTF(IEF,NEF5S,LSS)  ! INTENG
2867:          LSS = LINTF(IEF,NEF5S,LSS)  ! LSTF5E
2868:          do J = 1, NEF5S
2869:              LSS = LREALF(IEF,2*NBINA1,LSS) ! UTAB & PROBA
2870:          end do
2871: C
2872:      else if ( LF.eq.64 ) then
2873:          LSS = LINTF(IEF,1,LSS)      ! LF
2874: C
2875:      else if ( LF.eq.66 ) then
2876:          LSS = LINTF(IEF,2,LSS)      ! LF & 2
2877:          LSS = LREALF(IEF,2,LSS)     ! ELOW & EHI
2878:          LSS = LINTF(IEF,2,LSS)      ! 2 & 2
2879:          LSS = LREALF(IEF,2,LSS)     ! APSX & APSX
2880:          LSS = LINTF(IEF,1,LSS)      ! NPSX
2881: C
2882:      else if ( LF.eq.67 ) then
2883:          NEF5S = IDATA(LSS+1)
2884:          LSS = LINTF(IEF,2,LSS)      ! LF & NEF5S
2885:          LSS = LREALF(IEF,NEF5S,LSS) ! EENG
2886:          LSS = LINTF(IEF,2*NEF5S,LSS) ! INTENG & LSTF5E
2887:          do J = 1, NEF5S
2888:              LSS = LREALF(IEF,NBINA1,LSS) ! UTAB
2889:              NMU = IDATA(LSS+1)
2890:              LSS = LINTF(IEF,2,LSS)      ! INTMU & NMU
2891:              LSS = LREALF(IEF,NMU,LSS)   ! XMU
2892:              LSAVE = LSS
2893:              LSS = LINTF(IEF,NMU,LSS)    ! LST2A
2894:              do K = 1, NMU
2895:                  if ( LSS.ne.IDATA(LSAVE+K-1) ) then
2896:                      write(6,*) ' invalid position : (LF=67 data)',
2897:                      &      LSS, ' ( LST2A(k)= ', IDATA(LSAVE+K-1), ' )'
2898:                      write(6,*) ' Eloop j & Mu-loop k is ', J, K,
2899:                      &      ' & NMU is ', NMU
2900:                  end if
2901:                  NEPI = IDATA(LSS)
2902:                  LSS = LINTF(IEF,1,LSS)  ! NEPI
2903:                  LSS = LREALF(IEF,NEPI,LSS) ! Q
2904:                  LSS = LINTF(IEF,2*NEPI,LSS) ! IEPOS1 & IEPOS2
2905:                  LSS = LREALF(IEF,2*(NEPI+1),LSS) ! ERINF & ERFB
2906:              end do
2907:          end do
2908: C
2909:      else
2910:          write(6,*) 'XXX UNDEFINED LF ', LF, ' FOR ',
2911:          &      'ENERGY DISTRIBUTION DATA !!!'
2912:          stop 666
2913:      end if
2914:      end do
2915:      end if
2916:      end do
2917: C
2918:      call DOUT( ADATA(L0), IDATA(L0), IEF(L0), LSS-L0 )
2919: C
2920: C ....
2921:      if ( LSS-1.ne.NTDATA ) then
2922:          write(6,*) ' DATA NUMBER MISMATCH (RECORD #3 )'

```

```

2923:          write(6,*) ' NTDATA= ', NTDATA, ' TRACED = ', LSS - 1
2924:      end if
2925: C ....
2926:      call NXTLIN( IRT )
2927:      call CWRITE( 'END-OF-RECORD-3', 0, IRT )
2928:      call NXTLIN( IRT )
2929:      end
2930: C
2931: C
2932: C=====
2933: C MARK INTEGER DATA ( IEF = 0 )
2934: C=====
2935: C
2936: C
2937:      function LINTF(IEF,N,LSS)
2938:          integer IEF(*)
2939:          common /IEFSIZ/ MXIEF
2940: C
2941:          if ( LSS+N-1.gt.MXIEF ) then
2942:              write(6,*) '!!! TOO LONG DATA DETECTED IN "LINTF"'
2943:              write(6,*) ' LAST = ', LSS + N - 1, ' LSS =', LSS
2944:              write(6,*) ' MAX = ', MXIEF
2945:              stop 999
2946:          end if
2947: C
2948:          do 100 I = LSS, LSS + N - 1
2949:              IEF(I) = 0
2950:          100 continue
2951:          LINTF = LSS + N
2952:          return
2953:      end
2954: C
2955: C=====
2956: C MARK REAL DATA ( IEF = 1 )
2957: C=====
2958: C
2959:      function LREALF(IEF,N,LSS)
2960:          integer IEF(*)
2961:          common /IEFSIZ/ MXIEF
2962: C CHECK
2963: C WRITE(6,*) ' LREALF : LSS ',LSS,' N ',N
2964: C IF(LSS+N-1.GT.MXIEF) THEN
2965: C WRITE(6,*) '!!! TOO LONG DATA DETECTED IN "LREALF"'
2966: C WRITE(6,*) ' LAST = ',LSS+N-1,' LSS =',LSS
2967: C WRITE(6,*) ' MAX = ',MXIEF
2968: C STOP 999
2969: C ENDIF
2970: C
2971:          do 100 I = LSS, LSS + N - 1
2972:              IEF(I) = 1
2973:          100 continue
2974:          LREALF = LSS + N
2975:          return
2976:      end
2977: C
2978: C
2979: C=====
2980: C OUTPUT INTEGER DATA IN FREE FORMAT WITH DATA NUMBER.
2981: C=====
2982: C
2983: C
2984:      subroutine IDOUT( IDATA, ND, IRT )
2985:          integer IDATA(ND)
2986:          character*80 LABEL
2987: C

```


src/tools/nlb2txt.f

```

2988: C ---- DATA NUMBER : '@INT ND' ----
2989: C
2990: LABEL = '@INT'
2991: I = INDEX(LABEL,' ') + 1
2992: write(LABEL(I+1:LEN(LABEL)),'(I8)') ND
2993: call CCOMP( LABEL(I+1:LEN(LABEL)), LN )
2994: call CWRITE( LABEL, 0, IRT )
2995: C
2996: if ( ND.eq.0 ) return
2997: call IWRITE( IDATA, ND, IRT )
2998: return
2999: end
3000: C
3001: C
3002: C=====
3003: C OUTPUT REAL DATA IN FREE FORMAT WITH DATA NUMBER.
3004: C=====
3005: C
3006: C
3007: subroutine EDOUT( EDATA, ND, IRT )
3008: real EDATA(ND)
3009: character*80 LABEL
3010: C
3011: data IDIGIT /8/
3012: C
3013: C
3014: LABEL = '@FLOAT'
3015: I = INDEX(LABEL,' ') + 1
3016: write(LABEL(I:LEN(LABEL)),'(I8)') ND
3017: call CCOMP( LABEL(I:LEN(LABEL)), LN )
3018: call CWRITE( LABEL, 0, IRT )
3019: C
3020: if ( ND.eq.0 ) return
3021: C
3022: call EWRITE( EDATA, ND, IDIGIT, IRT )
3023: return
3024: end
3025: C
3026: C
3027: C=====
3028: C OUTPUT INTEGER/REAL MIXED DATA IN FREE FORMAT.
3029: C ( IEF = 0 / 1 : INTEGER/REAL )
3030: C=====
3031: C
3032: C
3033: subroutine DOUT( ADATA, IDATA, IEF, NTDATA )
3034: C
3035: real ADATA(NTDATA)
3036: integer IDATA(NTDATA)
3037: integer IEF(NTDATA)
3038: C
3039: character*80 LABEL
3040: C
3041: data IDIGIT /8/
3042: C
3043: C
3044: if ( NTDATA.eq.0 ) return
3045: C
3046: C ... if all data are the same type use IDOUT or EDOUT
3047: C
3048: JTYPE = -1
3049: C
3050: C (Suppressed)
3051: C do 10 I=2,NTDATA
3052: C if ( IEF(I).ne.IEF(1) ) goto 20

```

```

3053: C 10 continue
3054: C JTYPE = IEF(1)
3055: C 20 continue
3056: C
3057: C ... integer only ....
3058: C
3059: if ( JTYPE.eq.0 ) then
3060: call IDOUT( IDATA, NTDATA, IRT )
3061: C
3062: C ... float only ....
3063: C
3064: else if ( JTYPE.eq.1 ) then
3065: call EDOUT( ADATA, NTDATA, IRT )
3066: C
3067: C ... integer float mixed ....
3068: C
3069: else
3070: LABEL = '@FREE'
3071: I = INDEX(LABEL,' ') + 1
3072: write(LABEL(I:LEN(LABEL)),'(I8)') NTDATA
3073: call CCOMP( LABEL(I:LEN(LABEL)), LN )
3074: C
3075: call CWRITE( LABEL, 0, IRT )
3076: call NXTLIN( IRT )
3077: C
3078: I1 = 1
3079: C
3080: do 100 I = 1, NTDATA + 1
3081: if ( I.gt.NTDATA .or. (I.le.NTDATA.and.IEF(I).ne.IEF(I1)) )
3082: & then
3083: C
3084: if ( IEF(I1).eq.0 ) then
3085: call IWRITE( IDATA(I1), I-I1, IRT )
3086: C
3087: else if ( IEF(I1).eq.1 ) then
3088: call EWRITE( ADATA(I1), I-I1, IDIGIT, IRT )
3089: end if
3090: C
3091: I1 = I
3092: end if
3093: 100 continue
3094: end if
3095: C
3096: return
3097: end
3098: C
3099: C
3100: C
3101: C
3102: C=====
3103: C OUTPUT REAL DATA IN FIXED FORMAT.
3104: C=====
3105: C
3106: C
3107: subroutine FFOUT( ADATA, NTDATA )
3108: C
3109: real ADATA(NTDATA)
3110: C
3111: character*80 LABEL
3112: C
3113: if ( NTDATA.eq.0 ) return
3114: C
3115: LABEL = '@FF'
3116: I = INDEX(LABEL,' ') + 1
3117: write(LABEL(I:LEN(LABEL)),'(I8)') NTDATA

```

src/tools/nlb2txt.f

```

3118:      call CCOMP( LABEL(I:LEN(LABEL)), LN )
3119: C
3120:      call CWRITE( LABEL, 0, IRT )
3121:      call NXTLIN( IRT )
3122: C
3123:      NN      = 5
3124:      do 100 I = 1, NTDATA, NN
3125:          I2      = MIN(NTDATA,I+NN-1)
3126:          LABEL   = ' '
3127:          write(LABEL,'(5(1P,E15.7:))') (ADATA(J),J=I,I2)
3128:          call CWRITE( LABEL, 0, IRT )
3129:          call NXTLIN( IRT )
3130: 100 continue
3131: C
3132:      return
3133: end
3134: C
3135: C
3136: C
3137: C
3138: C
3139: C
3140: C --- 'FREE-FORM' OUTPUT ROUTINES ---
3141: C
3142: C PROGRAMMED BY M.SASAKI ( JAN 1992 )
3143: C
3144: C
3145: C OUTPUT DATA (FLOAT, INT., CHARACTER ) INPUTTABLE BY
3146: C USING 'CREAD' SUBROUTINE PACKAGE MADE BY M.SASAKI.
3147: C
3148: C === CONTENTS ===
3149: C
3150: C -----
3151: C SUBROUTINE  POPEN( IUOUT, NBUF )
3152: C -----
3153: C      ASSIGN OUTPUT I/O UNIT TO IUOUT ASSUMING
3154: C      NBUF CHARACTERS IN ONE LINE.
3155: C
3156: C -----
3157: C SUBROUTINE  EWRITE( EDATA, N, IDIGIT, IWST )
3158: C SUBROUTINE  DWRITE( DDATA, N, IDIGIT, IWST )
3159: C -----
3160: C      OUTPUT N FLOATING POINT DATA (SINGLE/DOUBLE PRECISION
3161: C      RESPECTIVELY) ON UNIT IUOUT IN IDIGIT OF PRECISION
3162: C      IN DECIMAL.
3163: C
3164: C -----
3165: C SUBROUTINE CWRITE( STR, N, IWST )
3166: C SUBROUTINE SWRITE( STR, N, IWST )
3167: C -----
3168: C
3169: C      OUTPUT CHARACTER STRING STR ( N CHARACTERS. IF N=0
3170: C      LEN(STR) CHARACTERS ). 'SWRITE' ADDS TERMINATION CHARACTERS
3171: C      READABLE BY SREAD ROUTINE OF CREAD PACKAGE.
3172: C
3173: C -----
3174: C SUBROUTINE  IWRITE( IDATA, N , IWST)
3175: C -----
3176: C      OUTPUT N INTEGER DATA IN ARRAY.
3177: C
3178: C -----
3179: C SUBROUTINE  NXTLIN( IWST )
3180: C -----
3181: C      FLUSH LINE INPUT BUFFER & GO TO NEXT LINE.
3182: C

```

```

3183: C=====
3184:      subroutine POPEN( IO,      NDG )
3185: C
3186:      common /CWRT1/  LINE
3187:      character*256 LINE
3188:      common /CWRT2/  IUOUT,  NBUF,  IPOS
3189: C
3190:      IUOUT  = IO
3191: C      REWIND IUOUT
3192:      IPOS   = 1
3193: C
3194:      if ( NDG.gt.LEN(LINE) ) then
3195:          write(6,*) ' = OUTPUT BUFFER TOO LONG ! ( > ', LEN(LINE), ')='
3196:          stop 666
3197:      end if
3198:      NBUF   = NDG
3199:      LINE   = ' '
3200: C
3201:      return
3202: end
3203: C
3204: C=====
3205:      subroutine CWRITE( STR,      NL,      IWST )
3206: C
3207:      character*(*) STR
3208: C
3209:      common /CWRT1/  LINE
3210:      character*256 LINE
3211:      common /CWRT2/  IUOUT,  NBUF,  IPOS
3212: C
3213: C
3214:      M      = 1
3215:      N      = NL
3216:      if ( NL.eq.0 ) then
3217:          M      = 0
3218:          do 100 I = 1, LEN(STR)
3219:              if ( STR(I:I).ne.' ' ) go to 110
3220: 100 continue
3221:          M      = I
3222:          do 120 I = LEN(STR), 1, -1
3223:              if ( STR(I:I).ne.' ' ) go to 130
3224: 120 continue
3225:          130 N      = I
3226:      end if
3227: C
3228:      IP2     = IPOS + N - M
3229:      if ( IP2.gt.NBUF ) then
3230:          call NXTLIN( ICON )
3231:          IP2   = IPOS + N - M
3232:      end if
3233:      LINE(IP2:IP2) = STR(M:N)
3234: C
3235:      IPOS     = IP2 + 2
3236:      return
3237: end
3238: C
3239: C=====
3240: C
3241:      subroutine NXTLIN( IWST )
3242: C
3243:      common /CWRT1/  LINE
3244:      character*256 LINE
3245:      common /CWRT2/  IUOUT,  NBUF,  IPOS
3246: C
3247:      write(IUOUT,'(A)') LINE(1:MIN(NBUF,IPOS-1))

```

src/tools/nlb2txt.f

```

3248:      LINE(1:NBUF)   = ' '
3249:      IPOS             = 1
3250:      return
3251:      end
3252: C
3253: C=====
3254: C
3255:      subroutine SWRITE( STR,   NL,   IWST )
3256: C
3257:      character*(*) STR
3258: C
3259:      common /CWRT1/   LINE
3260:      character*256 LINE
3261:      common /CWRT2/   IUOUT, NBUF, IPOS
3262: C
3263:      character*512 STR2
3264:      character*10 TM
3265:      character*1 T
3266:      data TM /' ' "&:;?#@%-./
3267: C
3268:      N             = NL
3269:      if ( NL.eq.0 ) N      = LEN(STR)
3270: C
3271:      if ( N.gt.LEN(STR2)-2 ) then
3272:        write(6,*) ' XXX TOO LONG STRAING! (SWRITE) XXX'
3273:        write(6,*) STR(1:N)
3274:        stop 666
3275:      end if
3276: C
3277:      do 100 K = 1, 6
3278:        if ( INDEX(STR(1:N),TM(K:K)).eq.0 ) then
3279:          T          = TM(K:K)
3280:          go to 110
3281:        end if
3282:      100 continue
3283:      write(6,*) 'XX CANNOT FIND APPROPREATE TERMINATION',
3284:      & ' CHARACTER WITHIN<', TM, '>'
3285:      write(6,*) ' STRING = <', STR(1:N), '> '
3286:      stop 666
3287: C
3288: C
3289:      110 STR2      = T//STR(1:N) //T
3290:      call CWRITE( STR2, N+2, ICON )
3291:      return
3292:      end
3293: C
3294:      subroutine NOBLNK( LINE, IS,   IE,   NL )
3295: C
3296: C  JAERI MONTE CARLO CODE UTILITY
3297: C
3298: C=====
3299: C  PURPOSE:  FIND NONBLANK CHARACTER STRING IN 'LINE'.
3300: C
3301: C  SEARCH LINE(IS:NL)
3302: C  AND RETURNS STARTING POSITION AND END POSITION OF NON-BLANK STRING.
3303: C  IN IS & IE RESPECTIVELY.
3304: C
3305: C  CALLED IN:  ZONEIN  ETC.
3306: C=====
3307:      character*(*) LINE
3308:      if ( IS.gt.NL ) then
3309:        IE      = IS
3310:        return
3311:      end if
3312:      ISS      = 0

```

```

3313:      do 100 I = IS, NL
3314:        if ( LINE(I:I).ne.' ' .and.ISS.eq.0 ) ISS      = I
3315:        if ( LINE(I:I).eq.' ' .and.ISS.ne.0 ) go to 110
3316:      100 continue
3317:      if ( ISS.eq.0 ) then
3318:        IS      = I
3319:        IE      = I
3320:        return
3321:      end if
3322:      110 IS      = ISS
3323:      IE      = I - 1
3324:      return
3325:      end
3326: C
3327: C
3328: C
3329:      subroutine MATCH( IM,   PATTRN,STR )
3330: C
3331: C  MVP/GMVP UTILITY
3332: C
3333: C=====
3334: C  PURPOSE:  STRING PATTERN MATCHING
3335: C  HISTORY:  PROGRAMMED BY M.SASAKI (30 APR 1992)
3336: C  UPDATE:
3337: C  7 JUL 1993:  '*' matches any string including null string.
3338: C               More sophisticated matching logic that allows any
3339: C               combinations of wildcard meta-characters.
3340: C=====
3341: C
3342: C  PATTRN & STR CAN INCLUDE BLANKS BUT THEY MUST BE PLACED AFTER ANY
3343: C  NON-BLANK CHARACERS. ( ignore characters after blank !! )
3344: C
3345: C      IM = 1 / 0 ... match / unmatched
3346: C
3347: C
3348: C  << WILDCARD CHARACTERS >>
3349: C
3350: C  '?' : ANY ONE CHARACTER ON THE PLACE OF '?'.
3351: C  '*' : ANY CHARACTER STRING ( length >= 0 ).
3352: C
3353: C=====
3354:      character*(*) PATTRN
3355:      character*(*) STR
3356: C
3357: C  ... default is unmatched ...
3358: C
3359: C      IM      = 0
3360: C
3361: C      LP      = INDEX(PATTRN,' ') - 1
3362: C      if ( LP.lt.0 ) LP      = LEN(PATTRN)
3363: C
3364: C      LS      = INDEX(STR,' ') - 1
3365: C      if ( LS.lt.0 ) LS      = LEN(STR)
3366: C      if ( LS.eq.0 ) return
3367: C
3368: C
3369: C
3370: C  ... pattern has no '*' .....
3371: C
3372: C
3373: C
3374: C      if ( INDEX(PATTRN(:LP),'*').eq.0 ) then
3375: C        if ( LP.ne.LS ) return
3376: C        do 100 I = 1, LP
3377: C          if ( PATTRN(I:I).ne.'?' .and.PATTRN(I:I).ne.STR(I:I) ) return

```

src/tools/nlb2txt.f

```

3378: 100 continue
3379: IM = 1
3380: return
3381: end if
3382: C
3383: C
3384: C .... pattern has '*' ....
3385: C
3386: C
3387: C << Logic of pattern matching with wild-card character '*' >>
3388: C
3389: C
3390: C
3391: C 0. Set "current pattern" to the whole pattern string.
3392: C And set "object string" to the string checked.
3393: C
3394: C 1. Split the current-pattern string to two sub-patterns by '*'.
3395: C
3396: C pattern = L * R
3397: C for L ( if the current pattern has no '*' )
3398: C
3399: C L : left-side sub-pattern (does not include '*'. can be
3400: C null string)
3401: C R : right-side sub-pattern (may include '*')
3402: C
3403: C 2. If the left-side sub-pattern is not null string,
3404: C check matching with the left-side sub-pattern.
3405: C
3406: C If the sub-pattern does not match any substring of the object
3407: C string (the leftmost substring for the first time), terminate as
3408: C 'unmatching'.
3409: C
3410: C Else cut off the matching substring and characters on the left
3411: C of the substring from the object string and proceed to step 4.
3412: C
3413: C 3. If the pattern has no '*',
3414: C check matching between the current pattern and the rightmost
3415: C substring of the object string.
3416: C
3417: C If matched ,terminate matching as 'matching',else terminate
3418: C matching as 'unmatching'.
3419: C
3420: C 4. Set the right-side sub-pattern as the current pattern and
3421: C repeat the process from 1.
3422: C
3423: C
3424: C IP = 1
3425: C IS = 1
3426: C
3427: C 110 continue
3428: C
3429: C
3430: C K = INDEX(PATTRN(IP:LP),'*')
3431: C
3432: C
3433: C ==== check matching with the left side of current '*'
3434: C
3435: C
3436: C if ( K.gt.1 ) then
3437: C
3438: C IS1 = LS - K + 2
3439: C
3440: C .... for the first matching, the sub-pattern must match
3441: C at the leftmost position.
3442: C

```

```

3443: if ( IP.eq.1 ) IS1 = IS
3444: do 130 ISS = IS, IS1
3445: do 120 I = 0, K - 2
3446: if ( PATTRN(IP+I:IP+I).ne.'?'
3447: & .and.PATTRN(IP+I:IP+I).ne.STR(ISS+I:ISS+I) ) go to 130
3448: 120 continue
3449: C
3450: C ----- left side matches -----
3451: C
3452: C IS = ISS + I
3453: C go to 150
3454: C
3455: C 130 continue
3456: C
3457: C ----- unmatched -----
3458: C
3459: C return
3460: C
3461: C
3462: C ==== check the rightmost string if no '*' remains in pattern ====
3463: C
3464: C
3465: C else if ( K.eq.0 ) then
3466: C if ( LS-IS.lt.LP-IP ) return
3467: C do 140 I = IP, LP
3468: C if ( PATTRN(I:I).ne.'?'
3469: C & .and.PATTRN(I:I).ne.STR(LS+I-LP:LS+I-LP) ) return
3470: C 140 continue
3471: C IM = 1
3472: C return
3473: C end if
3474: C
3475: C (if the left-side sub-pattern is null, do only the following step.)
3476: C
3477: C 150 IP = IP + K
3478: C if ( IP.le.LP ) go to 110
3479: C
3480: C ---- matched !! ---
3481: C
3482: C IM = 1
3483: C
3484: C return
3485: C end
3486: C
3487: C=====
3488: C
3489: C subroutine CCOMP( STR, LN )
3490: C character*(*) STR
3491: C LN = 0
3492: C do 100 I = 1, LEN(STR)
3493: C if ( STR(I:I).ne.' ' ) then
3494: C LN = LN + 1
3495: C STR(LN:LN) = STR(I:I)
3496: C end if
3497: C 100 continue
3498: C if ( LN.lt.LEN(STR) ) STR(LN+1:LEN(STR)) = ' '
3499: C return
3500: C end
3501: C
3502: C=====
3503: C
3504: C function ICLEN2(CH)
3505: C=====
3506: C Get position of the last non-blank character in a character string.
3507: C HISTORY: PROGRAMMED BY M.SASAKI (Jul 1994)

```

src/tools/nlb2txt.f

```

3508: C=====
3509:   character*(*) CH
3510:   do 100 I = LEN(CH), 1, -1
3511:     if ( CH(I:I).ne.' ' ) then
3512:       ICLEN2 = I
3513:       return
3514:     end if
3515:   100 continue
3516:   ICLEN2 = 0
3517:   return
3518: end
3519: C
3520: C=====
3521: C
3522:   subroutine IWRITE( IDATA, NL,   IWST )
3523: C
3524:   integer IDATA(*)
3525: C
3526:   common /CWRT1/   LINE
3527:   character*256 LINE
3528:   common /CWRT2/   IUOUT, NBUF,   IPOS
3529: C
3530:   character*32 BUF
3531:   character*16 BUF2
3532: C
3533:   if ( NL.le.0 ) NL   = 1
3534: C
3535:   if ( NL.eq.1 ) then
3536:     write(BUF,'(I12)') IDATA(1)
3537:     call CWRITE( BUF, 0, IWST )
3538:     return
3539:   end if
3540: C
3541:   IWST   = 0
3542:   I1     = 1
3543:   do 100 I = 1, NL + 1
3544:     if ( I.gt.NL .or. (I.le.NL.and.IDATA(I1).ne.IDATA(I)) ) then
3545:       write(BUF2,'(I16)') IDATA(I1)
3546:       call CCOMP( BUF2, LN )
3547:       NN      = I - I1
3548:       if ( NN.gt.1 ) then
3549:         write(BUF,'(I7,'''',A)') NN, BUF2(1:LN)
3550:         call CWRITE( BUF, 0, ICON )
3551:       else
3552:         call CWRITE( BUF2, LN, ICON )
3553:       end if
3554:       I1      = I
3555:     end if
3556:   100 continue
3557:   return
3558: end
3559: C
3560: C=====
3561: C
3562:   subroutine EWRITE( EDATA, NL,   IDIGIT,IWST )
3563: C
3564:   real EDATA(*)
3565: C
3566:   common /CWRT1/   LINE
3567:   character*256 LINE
3568:   common /CWRT2/   IUOUT, NBUF,   IPOS
3569: C
3570:   character*32 BUF
3571:   character*32 BUF2
3572: C

```

```

3573:   if ( NL.le.0 ) NL   = 1
3574: C
3575:   if ( NL.eq.1 ) then
3576:     call CEOPT( BUF, EDATA(1), IDIGIT, LN )
3577:     call CWRITE( BUF, LN, IWST )
3578:     return
3579:   end if
3580: C
3581:   IWST   = 0
3582:   I1     = 1
3583:   do 100 I = 1, NL + 1
3584:     if ( I.gt.NL .or. (I.le.NL.and.EDATA(I1).ne.EDATA(I)) ) then
3585:       call CEOPT( BUF2, EDATA(I1), IDIGIT, LN )
3586:       NN      = I - I1
3587:       if ( NN.gt.1 ) then
3588:         write(BUF,'(I7,'''',A)') NN, BUF2(1:LN)
3589:         call CWRITE( BUF, 0, ICON )
3590:       else
3591:         call CWRITE( BUF2, LN, ICON )
3592:       end if
3593:       I1      = I
3594:     end if
3595:   100 continue
3596:   return
3597: end
3598: C
3599: C=====
3600: C   CONVERT A FLOATING POINT NUMBER TO 'OPTIOMIZED'
3601: C   FORM HAVING IDIGIT DECIMAL PRECISION AND STORE IN BUF.
3602: C
3603:   subroutine CEOPT( BUF,   DATA, IDIGIT,LN )
3604: C
3605:   real DATA
3606:   character*(*) BUF
3607:   character*20 FORMT
3608:   character*80 BUF2
3609: C
3610:   if ( IDIGIT.gt.9 .or. IDIGIT.le.0 ) then
3611:     write(6,*) ' XXX INVALID OR TOO LONG DIGIT (', IDIGIT, ')'
3612:     stop 666
3613:   end if
3614: C
3615:   if ( DATA.eq.0.0 ) then
3616:     BUF   = '0.'
3617:     LN    = 2
3618:     return
3619:   end if
3620: C
3621:   if ( IDIGIT.eq.8 ) then
3622:     write(BUF,'(1p,e15.7)') DATA
3623:     LN      = 15
3624:   else
3625:     FORMT   = ' '
3626:     LN      = 20
3627:     write(FORMT,'(''(1P,E'',I2,'''',I1,'''E2)''')' ) LN, IDIGIT - 1
3628: C
3629:     write(BUF,fmt =FORMT) DATA
3630:   end if
3631: C
3632:   K      = INDEX(BUF(:LN),'E')
3633: C
3634:   if ( K.ne.0 ) then
3635:     do 100 I = K - 1, 1, -1
3636:       if ( BUF(I:I).eq.'0' ) then
3637:         BUF(I:I)   = ' '

```

src/tools/nlb2txt.f

```
3638:         else
3639:             go to 110
3640:         end if
3641:     100     continue
3642:     110     if ( I.lt.K-1 ) call CCOMP( BUF(:LN), LL )
3643: C
3644:         K         = INDEX(BUF(:LN),'E')
3645:         NDIGIT    = K - INDEX(BUF(:LN),'.')
3646:         read(BUF(K+1:K+3),'(I3)') NE
3647: C
3648:         if ( NE.eq.0 ) then
3649:             BUF(K:K+3) = ' '
3650:         else if ( NE.ge.-1.and.NE.lt.NDIGIT ) then
3651:             KK      = INDEX(BUF(:LN),' ')
3652:             BUF2    = BUF(1:KK-1) //BUF(KK+1:K-1)
3653:             BUF     = ' '
3654:             BUF     = BUF2(1:KK-1+NE) //'.'//BUF2(KK+NE:K)
3655:         else if ( ABS(NE).lt.10 ) then
3656:             BUF(K+1:K+3) = ' '
3657:             if ( NE.gt.0 ) write(BUF(K+1:K+1),'(I1)') NE
3658:             if ( NE.lt.0 ) write(BUF(K+1:K+2),'(I2)') NE
3659:         end if
3660:     end if
3661: C
3662:     L      = LN
3663:     call CCOMP( BUF(:L), LN )
3664:     return
3665: end
```

src/tools/ntxt2lb.f

```

1: C
2: C=====
3: C   MVP UTILITY:  NEUTRON (PHOTON) LIBRARY FORM CONVERTER 2.
4: C       TEXT --> BINARY
5: C=====
6: C
7: C=====
8: C
9: C   << INPUT >>
10: C
11: C   1. INDEX-FILE NAME TO BE CREATED
12: C
13: C   << IF 1. IS NOT BLANK; MULTI-NUCLIDE TREATMENT CREATING INDEX-FILE.>>
14: C
15: C       2. INPUT TEXT FILE NAME
16: C       3. OUTPUT BINARY FILE
17: C
18: C       IF THIS NAME INCLUDES '*', '*' IS REPLACED BY NUCLIDE-ID,
19: C       ELSE FILENAME COCATINATED BY USING NUCLIDE-NAME AS REAL
20: C       OUTPUT-FILE NAME.
21: C
22: C   4. NUCLIDE NAMES CONVERTED BY NAME MATCHING PATTERN.
23: C       ( REPEAT 4. UNTIL BLANK INPUT ENCOUNTERED )
24: C
25: C   << IF 1. IS BLANK; FILE-BY-FILE CONVERSION >>
26: C
27: C       REPEAT INPUT-FILE & OUTPUT-FILE
28: C
29: C       form of this line is;
30: C
31: C=====
32: C
33: C
34: C   program TXT2LB
35: C
36: C   parameter( LIMIT      = 12000000 )
37: C   real ADATA(LIMIT)
38: C   common /BUFFER/  ADATA
39: C
40: C   character*80 FINDEX
41: C   character*80 INFILE
42: C   character*80 OTFILE
43: C   character*80 OTF
44: C
45: C   ..... nuclide name patterns .....
46: C
47: C   CV3  parameter( MATNL      = 8 )
48: C   parameter( MATNL      = 16 )
49: C   parameter( MAXNP      = 64 )
50: C   character*(MATNL) NUCNAM(MAXNP)
51: C   integer NNUCP
52: C
53: C   CCCCC/IF SYSTEM( HP* )
54: C   C/IF FLOATHANDLER( ONSTMT )
55: C   *   on real*4 underflow call underf4
56: C   *   on real*8 underflow call underf8
57: C   C/ENDIF
58: C
59: C
60: C   write(6,*) ' '
61: C   write(6,*) ' == MVP LIBRARY CONVERTER :TEXT --> BINARY '
62: C   write(6,*) ' Version 2.0 (July 1998)'
63: C   write(6,*) ' '
64: C
65: C

```

```

66: C***** READ INDEX FILE NAME **** IF BLANK, FILE BY FILE CONVERSION ****
67: C
68: C
69: C   FINDEX = ' '
70: C   100 read(5,'(A)') FINDEX
71: C   if ( FINDEX(1:1).eq.' ' ) go to 100
72: C
73: C   write(6,*) 'INDEX: ', FINDEX
74: C
75: C=====
76: C   CONVERT DATA DESCRIBED IN INDEX FILE .
77: C=====
78: C
79: C   if ( FINDEX.ne.' ' ) then
80: C       NIN      = 10
81: C       NOUT     = 20
82: C
83: C   .... READ INPUT FILE ....
84: C
85: C   IF OUTPUT FILE NAME HAS A '*' CHARACTER, TEXT FILES ARE
86: C   CREATED FOR EACH NUCLIDE AND NAMED BY REPLACING THE '*' CHARACTER
87: C   BY NUCLIDE-ID .
88: C
89: C   INFILE = ' '
90: C   110 read(5,'(A)') INFILE
91: C   if ( INFILE(1:1).eq.' ' ) then
92: C       INFILE = ' '
93: C       go to 110
94: C   end if
95: C
96: C   write(6,'(//' INPUT FILE : '(A)') INFILE
97: C
98: C   if ( INFILE.eq.'-' ) then
99: C       NIN      = 5
100: C   else
101: C
102: C/IF READONLY(DEC)
103: C
104: C   ... READONLY parameter : DEC VMS-Fortran & many UNIX fortrons
105: C
106: C       open( NIN, file =INFILE, iostat =IOS, form ='FORMATTED',
107: C           &         readonly )
108: C
109: C/ELSEIF READONLY(ACTION)
110: C
111: C   ... ACTION='READ':  IBM VS-fortran or Fortran90
112: C
113: C       open( NIN, file =INFILE, iostat =IOS, form ='FORMATTED',
114: C           &         action ='READ' )
115: C
116: C/ELSEIF READONLY(SHR)
117: C
118: C   ... file sharing mode ( FACOM-M series, etc.)
119: C
120: C
121: C       open( NIN, file =INFILE, iostat =IOS, status ='SHR',
122: C           &         form ='FORMATTED', action ='READ' )
123: C
124: C/ELSEIF READONLY(MODE)
125: C
126: C   ... MODE='READ':  Microsoft Fortran (after v5.1)
127: C
128: C
129: C       open( NIN, file =INFILE, iostat =IOS, form ='FORMATTED',
130: C           &         mode ='READ' )

```

src/tools/ntxt2lb.f

```

131:
132: C/#ELSE
133:
134:         open( NIN, file =INFILE, iostat =IOS, status = 'OLD',
135:             &         form = 'FORMATTED' )
136:
137: C/#ENDIF
138: C
139:         if ( IOS.ne.0 ) then
140:             write(6,*) ' == INPUT FILE OPEN ERROR : CODE ', IOS
141:             stop 666
142:         end if
143:     end if
144: C
145:         OTF = ' '
146: 120 read(5,'(A)') OTF
147:         if ( OTF(1:1).eq.''' ) go to 120
148: C
149:         write(6,'(//' OUTPUT FILE : '//,A)') OTF
150:         NIDX = 25
151: C
152:         open( NIDX, file =FINDEX, iostat =IOS, status = 'UNKNOWN',
153:             &         form = 'FORMATTED' )
154: C
155:         if ( IOS.ne.0 ) then
156:             write(6,*) ' == INDEX FILE OPEN ERROR : CODE ', IOS
157:             stop 666
158:         end if
159: C
160: C
161: C ===== specify name patterns of nuclides converted =====
162: C
163: C         ( terminate input by blank line )
164: C
165:         NNUCP = 0
166: 130 NNUCP = NNUCP + 1
167:         if ( NNUCP.gt.MAXNP ) then
168:             write(6,*) 'XXX TOO MANY NUCLIDE NAME PATTERNS '
169:             stop 999
170:         end if
171: C
172:         NUCNAM(NNUCP) = ' '
173: 140 read(5,'(A)') NUCNAM(NNUCP)
174:         if ( NUCNAM(NNUCP)(1:1).eq.''' ) go to 140
175: C
176:         if ( NUCNAM(NNUCP).ne.' ' ) go to 130
177: C
178:         NNUCP = NNUCP - 1
179: C
180: C         .... SELECT ALL NUCLIDES IF NUCNAM NOT SPECIFIED ....
181: C
182:         if ( NNUCP.eq.0 ) then
183:             NUCNAM(1) = '*'
184:             NNUCP = 1
185:         end if
186: C
187:         write(6,'(//' == NUCLIDE CONVERTED == '//{(3x,'<' ,A,'>'))'
188:             &         ) (NUCNAM(I),I=1,NNUCP)
189: C
190: C
191: C
192: C
193:         IFMODE = 0
194: C
195: 150 call CONV( NIN, NOUT, IEND, IFMODE, OTF, NIDX, NUCNAM, NNUCP,

```

```

196:         &         ADATA, ADATA, LIMIT )
197: C
198:         if ( IEND.eq.1 ) then
199:             write(6,*) '=== ALL NUCLIDE PROCESSED '
200:             go to 190
201:         end if
202:         go to 150
203: C
204: C
205: C=====
206: C FILE-BY-FILE CONVERSION.
207: C=====
208: C
209: C
210:         else
211: C
212: 160 continue
213: C
214:         write(6,*) ' # NAME OF INPUT TEXT LIBRARY --> '
215: C
216:         INFILE = ' '
217: 170 read(5,'(A)',end =190) INFILE
218:         if ( INFILE(1:1).eq.''' ) go to 170
219: C
220:         write(6,*) ' NAME: ', INFILE
221: C
222: C
223:         write(6,*) ' # NAME OF OUTPUT BINARY LIBRARY --> '
224: C
225:         OTFILE = ' '
226: 180 read(5,'(A)') OTFILE
227:         if ( OTFILE(1:1).eq.''' ) go to 180
228: C
229:         write(6,*) ' NAME: ', OTFILE
230: C
231: C
232:         NIN = 10
233:         NOUT = 20
234: C
235: C/#IF READONLY( DEC )
236:         open( NIN, file =INFILE, iostat =IOS, form = 'FORMATTED',
237:             &         readonly )
238: C/#ELSEIF READONLY( SHR )
239:         open( NIN, file =INFILE, iostat =IOS, status = 'SHR',
240:             &         form = 'FORMATTED', action = 'READ' )
241: C/#ELSEIF READONLY( ACTION )
242:         open( NIN, file =INFILE, iostat =IOS, form = 'FORMATTED',
243:             &         action = 'READ' )
244: C/#ELSEIF READONLY( MODE )
245:         open( NIN, file =INFILE, iostat =IOS, form = 'FORMATTED',
246:             &         mode = 'READ' )
247: C/#ELSE
248:         open( NIN, file =INFILE, iostat =IOS, status = 'OLD',
249:             &         form = 'FORMATTED' )
250: C/#ENDIF
251: C
252:         if ( IOS.ne.0 ) then
253:             write(6,*) ' == INPUT FILE OPEN ERROR : CODE ', IOS
254:             stop 666
255:         end if
256: C
257: C
258:         open( NOUT, file =OTFILE, iostat =IOS, status = 'UNKNOWN',
259:             &         form = 'UNFORMATTED' )
260: C

```


src/tools/ntxt2lb.f

```

261:      if ( IOS.ne.0 ) then
262:        write(6,*) ' == OUTPUT FILE OPEN ERROR : CODE ', IOS
263:        stop 666
264:      end if
265: C
266:      IFMODE = 1
267: C
268:      NNUCP = 1
269:      NUCNAM(NNUCP) = '*'
270: C
271: C
272:      call CONV( NIN, NOUT, IEND, IFMODE, OTF, NIDX, NUCNAM, NNUCP,
273: &            ADATA, ADATA, LIMIT )
274: C
275:      close( NIN )
276:      close( NOUT )
277: C
278:      go to 160
279:    end if
280: C
281:    190 stop
282:  end
283: C
284: C ... underflow error handling routine for HP-fortran
285: C ( April 1994 )
286: C/#IF FLOATHANDLER( ONSTMT )
287: C
288: C ... underflow handling ...
289: C
290:   subroutine UNDERF4( R4 )
291:     real R4
292:     R4 = 0.0
293:     return
294:   end
295: C
296: C
297:   subroutine UNDERF8( R8 )
298:     real*8 R8
299:     R8 = 0.0D0
300:     return
301:   end
302: C/#ENDIF
303: C
304: C
305:   subroutine CONV( NIN, NOUT, IEND, IFMODE,OTF, NIDX, NUCNAM,
306: &                NNUCP, ADATA, IDATA, NTDAT3 )
307: C
308: C=====
309: C FILE CONVERSION ROUTINE :
310: C
311: C IEND : =1 IF NO MORE DATA ON INPUT FILE ELSE 0
312: C IFMODE : =0 OUTPUT FILE NOT OPENED YET
313: C PREPARED IN THIS ROUTINE. DOES NOT CLOSE INPUT FILE
314: C =1 OUTPUT FILE OPENED ALREADY
315: C
316: C Update:
317: C 17 July 1995: make temporary variable "aa" used for floating point
318: C data input to double precision to avoid underflow error.
319: C=====
320: C
321: C
322:   character*(*) OTF
323:   integer IDATA(NTDAT3)
324:   real ADATA(NTDAT3)
325: C

```

```

326: C
327: C ..... nuclide name pattern .....
328: C
329: CV3 parameter( MATNL = 8 )
330: parameter( MATNL = 16 )
331: character*(MATNL) NUCNAM(*)
332: C
333: C
334: C
335:   character*8 COMMT(32)
336:   character*(MATNL) MATT
337: C*****CHARACTER*72 BUF
338:   character*80 BUF
339:   character*81 LINE2
340:   character*32 BUF2
341:   integer ISS(100), IEE(100)
342:   character*256 OTFILE
343:   real*8 AA
344:   logical OPD
345: C
346: C
347: C ... temporary buffer for internal input ...
348: C format 701 & 702 are for internal input.
349: C
350:   character*16 CTEMP
351:   7000 format(BN,I16)
352:   7020 format(BN,E16.0)
353: C
354:   NBUF = LEN(BUF)
355:   call SETUNT( NIN, IRT )
356: C
357:   IEND = 0
358: C
359: C ---- @NUC=MATT ---
360: C
361:   call CREAD( BUF, IRST )
362: C
363:   if ( IRST.eq.-1 ) then
364:     write(6,*) ' == END OF INPUT-FILE '
365:     IEND = 1
366:     return
367:   end if
368: C
369:   100 JOUT = 0
370: C
371:   continue
372: C
373:   if ( BUF(:5).eq.'@NUC=' ) then
374:     LL = INDEX(BUF,' ') - 1
375:     if ( LL.lt.0 ) LL = LEN(BUF)
376:     MATT = BUF(6:LL)
377:     write(6,*) '==== NUCLEIDE NAME : ', MATT
378: C
379:     do 110 I = 1, NNUCP
380:       call MATCH( JOUT, NUCNAM(I), MATT )
381:       if ( JOUT.eq.1 ) go to 120
382:     110 continue
383:     120 continue
384: C
385:     if ( JOUT.ne.0 ) then
386:       write(6,*) '==== ', MATT, ' is converted '
387:     else
388:       write(6,*) '==== ', MATT, ' is not converted '
389:     end if
390:   else

```

src/tools/ntxt2lb.f

```

391:      write(6,*) 'XXX INVALID RECORD !! MUST BE @NUC=NUC-ID '
392:      write(6,*) BUF
393:      stop 666
394:      end if
395: C
396: C .... SKIP DATA for unconverted nuclides ....
397: C
398:      if ( JOUT.eq.0 ) then
399:      130      call NEWLIN( IENDF )
400: C
401:      if ( IENDF.eq.1 ) then
402:      write(6,*) ' == END OF INPUT-FILE '
403:      IEND = 1
404:      return
405:      end if
406: C
407:      call CREAD( BUF, IRST )
408: C
409:      if ( BUF(:5).eq.'@NUC=' ) go to 100
410:      go to 130
411:      end if
412: C
413: C
414: C .... OPEN OUTPUT FILE HERE IF IFMODE = 0
415: C      AND ADD DATA TO INDEX FILE
416: C
417:      if ( IFMODE.eq.0 ) then
418:      KA = INDEX(OTF,'*')
419:      LO = INDEX(OTF,' ') - 1
420:      if ( LO.lt.0 ) LO = LEN(OTF)
421: caddv3
422:      MO = INDEX(MATT,' ') - 1
423:      if ( MO.lt.0 ) MO = LEN(MATT)
424: cend
425:      if ( KA.eq.0 ) then
426:      cmodv3
427:      OTFILE = OTF//MATT
428:      else if ( KA.eq.1.and.KA.eq.LO ) then
429:      cmodv3
430:      OTFILE = MATT
431:      else if ( KA.eq.LO ) then
432:      cmodv3
433:      OTFILE = OTF:(KA-1) //MATT
434:      OTFILE = OTF:(KA-1) //MATT:(MO)
435:      else
436:      cmodv3
437:      OTFILE = OTF:(KA-1) //MATT//OTF(KA+1:LO)
438:      OTFILE = OTF:(KA-1) //MATT:(MO)//OTF(KA+1:LO)
439:      end if
440: C
441:      LL = INDEX(OTFILE,' ') - 1
442:      if ( LL.lt.0 ) LL = LEN(OTFILE)
443: C
444:      write(6,*) ' KA LO : <', KA, LO, OTF(:LO), '>'
445: C
446:      write(6,*) ' OUTPUT FILE: <', OTFILE(:LL), '>'
447: C
448:      inquire(NOUT,opened =OPD)
449:      if ( OPD ) close( NOUT )
450: C
451:      open( NOUT, file =OTFILE(:LL), iostat =IOS, status = 'UNKNOWN',
452:      &      form = 'UNFORMATTED' )
453: C
454:      if ( IOS.ne.0 ) then
455:      write(6,*) ' == OUTPUT FILE OPEN ERROR : CODE ', IOS
456:      stop 666
457:      end if

```

```

456: C
457: C      .... INDEX FILE .....
458: C
459: Ccccc WRITE(NIDX,'(A,2X,A/)') MATT, OTFILE(:LL)
460:      write(NIDX,'(A,2X,A/)') MATT, OTFILE(:LL)
461: C
462:      end if
463: C
464: C ---- COMMENT DATA ---
465: C
466:      call CREAD( BUF, IRST )
467: C
468:      if ( BUF.ne.'@CHAR' ) then
469:      write(6,*) ' INVALID DATA : ', BUF
470:      stop
471:      end if
472: C
473:      call IREAD( NC, IRST )
474:      call IREAD( NCL, IRST )
475: C
476:      write(6,*) ' == COMMENTS : ', NC, '*', NCL, ' CHARACTERS == '
477:      if ( NCL.ne.8 ) then
478:      write(6,*) ' == CHARACTER STRINGS MUST BE 8-CHARACTERS ',
479:      &      'IN LENGTH '
480:      stop 999
481:      end if
482: C
483: C ... input title characters as 8 byte character strings.
484: C
485:      do 140 I = 1, NC
486:      call SREAD( COMMT(I), NCL, IRST )
487: 140 continue
488: C
489: C
490:      write(6,*) (COMMT(I),I=1,NC)
491: C
492: C ----- OTHER DATA OF RECORD 1 -----
493: C
494:      ND = 1
495: 150 BUF = ' '
496:      call CREAD( BUF, IRST )
497:      if ( BUF(1:15).ne.'END-OF-RECORD-1' ) then
498:      if ( BUF.eq.'@INT' ) then
499:      call IREAD( N, IRST )
500:      do 160 I = ND, ND + N - 1
501:      call IREAD( IDATA(I), IRST )
502: 160      continue
503:      else if ( BUF.eq.'@FLOAT' ) then
504:      call IREAD( N, IRST )
505:      do 170 I = ND, ND + N - 1
506:      call EREAD( ADATA(I), IRST )
507: 170      continue
508:      end if
509:      ND = ND + N
510:      go to 150
511:      end if
512: C
513:      ND = ND - 1
514:      write(6,*) ' RECORD-1 : ', ND, ' DATA '
515: C
516:      write(NOUT) (COMMT(I),I=1,NC), (ADATA(I),I=1,ND)
517: C
518: C ---- RECORD 2 ----
519: C
520:      ND = 1

```

src/tools/ntxt2lb.f

```

521: 180 BUF = ' '
522: call CREAD( BUF, IRST )
523: if ( BUF(1:15).ne.'END-OF-RECORD-2' ) then
524:   if ( BUF.eq.'@INT' ) then
525:     call IREAD( N, IRST )
526:     do 190 I = ND, ND + N - 1
527:       call IREAD( IDATA(I), IRST )
528: 190   continue
529:   else if ( BUF.eq.'@FLOAT' ) then
530:     call IREAD( N, IRST )
531:     do 200 I = ND, ND + N - 1
532:       call EREAD( ADATA(I), IRST )
533: 200   continue
534:   end if
535:   ND = ND + N
536:   go to 180
537: end if
538: C
539: ND = ND - 1
540: write(6,*) ' RECORD-2 : ', ND, ' DATA '
541: C
542: write(NOUT) (ADATA(I),I=1,ND)
543: C
544: C
545: C --- RECORD 3 -----
546: C
547: C Assuming that all RECORD #3 are written in @FREE type,
548: C @FREE string must start on a newline
549: C and data lines has no termination caharacter ('/' or
550: C commnet character (maybe '"').
551: C
552: C
553: C NDATA = 1
554: 210 BUF = ' '
555: call CREAD( BUF, IRST )
556: C
557: if ( BUF(1:15).ne.'END-OF-RECORD-3' ) then
558: C
559: C   if ( BUF.ne.'@FREE' ) then
560: C     if ( BUF.ne.'@FREE'.and.BUF.ne.'@INT'.and.BUF.ne.'@FLOAT'
561: C       & .and.BUF.ne.'@FF' ) then
562: C       write(6,*) ' ERROR IN READING RECORD-3 :', BUF
563: C       stop
564: C     end if
565: C
566: C   call IREAD( NDS, IRST )
567: C
568: C   write(6,*) ' GOING TO READ ', NDS, ' DATA ',BUF(1:10)
569: C   if ( NDS.eq.0 ) go to 210
570: C
571: C   if ( NDATA+NDS.gt.NTDAT3 ) then
572: C     write(6,*) 'XXX TOO MANY DATA !!!(', NDATA + NDS, ')'
573: C     write(6,*) 'XXX limit is ', NTDAT3
574: C     stop 999
575: C   end if
576: C
577: C   ND = NDATA
578: C
579: C   if ( BUF.eq.'@FREE' ) then
580: C
581: C ... input tokens on a line per a call ...
582: C
583: C iss(i) : starting position of i'th token in 'string'
584: C iee(i) : end position of i'th token in 'string'
585: C

```

```

586: C
587: 220 call CREADS( LINE2, NTOKEN, ISS, IEE, IRST )
588: C
589: if ( IRST.eq.2 ) then
590:   write(6,*) 'XXX too small character buffer <', LINE2, '>'
591:   stop 999
592: end if
593: C
594: do 250 KK = 1, NTOKEN
595:   IS = ISS(KK)
596:   IE = IEE(KK)
597:   NR = 1
598: C
599: C ... repetition as "nr*data"
600: C
601:   K = INDEX(LINE2(IS:IE),'*')
602:   if ( K.ne.0 ) then
603:     CTEMP = LINE2(IS:IS+K-2)
604:     read(CTEMP,fmt =7000,err =260) NR
605:   end if
606: C
607: C
608: C ... floating point data
609: C
610:   if ( INDEX(LINE2(IS:IE),'.').ne.0 ) then
611:     CTEMP = LINE2(IS+K:IE)
612:     read(CTEMP,fmt =7020,err =270) AA
613: C
614:     if ( NR.eq.1 ) then
615:       ADATA(ND) = AA
616:     else
617:       do 230 I = ND, ND + NR - 1
618:         ADATA(I) = AA
619:       230   continue
620:     end if
621: C
622: C ... integer data
623: C
624:   else
625:     CTEMP = LINE2(IS+K:IE)
626:     read(CTEMP,fmt =7000,err =260) II
627: C
628:     if ( NR.eq.1 ) then
629:       IDATA(ND) = II
630:     else
631:       do 240 I = ND, ND + NR - 1
632:         IDATA(I) = II
633:       240   continue
634:     end if
635:   end if
636: C
637:   ND = ND + NR
638: 250 continue
639: C
640:   if ( ND.lt.NDATA+NDS ) go to 220
641: C ----- END LOOP -----
642: C
643:   else if ( BUF.eq.'@INT' ) then
644:     do 162 I = ND, ND + NDS - 1
645:       call IREAD( IDATA(I), IRST )
646:     162   continue
647:     ND = ND + NDS
648:   else if ( BUF.eq.'@FLOAT' ) then
649:     do 172 I = ND, ND + NDS - 1
650:       call EREAD( ADATA(I), IRST )

```

src/tools/ntxt2lb.f

```

651: 172      continue
652:          ND      = ND + NDS
653: C
654: C      ... temporary fixed format
655: C
656:          else if ( BUF.eq.'@FF' ) then
657: CCC      read(NIN,'(6f15.0)') (ADATA(I),I = ND, ND + NDS - 1)
658:          N2 = ND + NDS - 1
659:          NN = 5
660:          do 174 I = ND, N2, NN
661:              i2 = min(N2,I+NN-1)
662:              call getlin(line2,irst)
663:              read(line2,7111) (ADATA(J),J=I,I2)
664: 7111      format(5f15.0)
665: 174      continue
666:          ND      = ND + NDS
667:          end if
668: C
669:          NDATA = ND
670:          go to 210
671:      end if
672: C
673:          NDATA = NDATA - 1
674:          write(6,*) ' RECORD-3 : ', NDATA, ' DATA '
675: C
676:          write(NOUT) (ADATA(I),I=1,NDATA)
677: C
678: C
679:          return
680: C
681: 260 write(6,*) 'xxx integer data read error <', CTEMP, '>'
682:      stop 791
683: 270 write(6,*) 'xxx real data read error <', CTEMP, '>'
684:      stop 792
685:      end
686: C
687: C
688: C
689: C
690: C
691: C      FREE FORM INPUT ROUTINES:
692: C
693: C      < HISTORY >
694: C
695: C      ORIGINALLY MADE BY M.SASAKI TO REPLACE KENO-IV INPUT
696: C      ROUTINES ON VAX ( OCTOBER 1991 )
697: C
698: C      MODIFIED FOR MESH GENERATOR FOR FLEX-SN CODE.
699: C      ( DECEMBER 1991 )
700: C
701: C =====
702: C < GENERAL USAGE >
703: C
704: C      USERS CAN GET A DATA-ITEM BY CALLING ONE OF *READ ROUTINES ONCE.
705: C
706: C      EX.      CALL IREAD( IN, IRST ) -> GET AN INTEGER DATA IN FROM
707: C               STANDARD INPUT.
708: C
709: C      DATA IN INPUT FILE SHOULD BE SEPARATED BY MORE THAN ONE BLANK
710: C      OR BY A COMMA.
711: C
712: C      EX.      1.0  2.0  3.0, 5.0  , , 10.0  /
713: C
714: C      INPUT DATA CAN INCLUDE COMMENT INDICATOR ( '!' AS DEFAULT ) AND
715: C      TERMINATION CHARACTER ( '/' AS DEFAULT ). DATA AFTER THE COMMENT

```

```

716: C      INDICATOR IN A INPUT LINE IS IGNORED. IF INPUT
717: C
718: C
719: C
720: C =====
721: C
722: C      SUBROUTINES & FUNCTIONS:
723: C
724: C      SUBROUTINE  SETUNT( NIN, IRST )
725: C
726: C          SET I/O UNIT NUMBER OF INPUT DATA.
727: C
728: C      SUBROUTINE  CREAD(STRING, IRST) :
729: C
730: C          GET A CHARACTER STRING THAT DOES NOT INCLUDE
731: C          ANY BLANKS FROM THE STANDARD INPUT FILE.
732: C          NON-BLANK CHARACTERS THAT SURPASS THE SIZE OF
733: C          ARGUMENT 'STRING' ARE IGNORED.
734: C
735: C          IRST = 0      : NORMAL INPUT
736: C                   -1   : END OF FILE
737: C                   1   : ALREADY ENCOUNTERED THE TERMINATION CHARACTER
738: C                       NO INPUT. TO ENABLE INPUT AGAIN, USERS MUST
739: C                       PROCEED TO THE NEXT LINE BY CALLING NEWLIN
740: C                       ROUTINE.
741: C                   2   : LENGTH OF THE ENCOUNTERED STRING IS GREATER
742: C                       THAN LEN(STRING).
743: C
744: C      SUBROUTINE  ISKIP(IRST)      :
745: C
746: C          SKIP A CHARACTER STRING(GET A CHARACTER STRING LIKE CREAD
747: C          BUT DOES NOT RETURN IT.
748: C
749: C      SUBROUTINE  SREAD( STRING, NCHAR, IRST ) :
750: C
751: C          GET A CHARACTER STRING BY TAKING THE FIRST NON-BLANK
752: C          CHARACTER AS TERMINATION CHARACTER (THIS MUST NOT BE
753: C          THE TERMINATION CHARACTER).
754: C
755: C      EX.      ' COMMENTS A ' -> COMMENTS A
756: C               "STRINGS 1 2 345 " -> STRINGS 1 2 345
757: C
758: C          NCHAR      : STRING LENGTH ( <= LEN(STRING) )
759: C          IRST = 0    : NORMAL INPUT
760: C                   -1 : END OF FILE
761: C                   1  : ALREADY ENCOUNTERED THE TERMINATION CHARACTER
762: C                   2  : LENGTH OF THE ENCOUNTERED STRING IS GREATER
763: C                       THAN LEN(STRING).
764: C
765: C      SUBROUTINES:
766: C      IREAD(IN,IRST),EREAD(EIN, IRST), DREAD(DIN, IRST) :
767: C
768: C          GET A 4-BYTE INTEGER      -> IN
769: C          A SINGLE PRECISION REAL NUMBER -> EIN
770: C          A DOUBLE PRECISION REAL NUMBER -> DIN
771: C          FROM THE STANDARD INPUT RESPECTIVELY.
772: C
773: C      ENTRIES FOR ARRAY INPUTS;
774: C
775: C      IREADA(INA,N,IRST),EREADA(EINA,N,IRST),DREADA(DINA,N,IRST) :
776: C
777: C          GET N-ITEMS SUCCESSIVELY ACCORDING TO THE DATA TYPE
778: C          OF SINGLE -ITEM ENTRIES ABOVE.
779: C
780: C      SPECIAL NOTATIONS FOR IREAD, EREAD & DREAD :

```

src/tools/ntxt2lb.f

```

781: C
782: C   < REPETITION >
783: C
784: C   '*, 'R', '$' 'OR' \ SANDWICHED BY TWO NUMBERS.
785: C
786: C   EX.    10*1.2    --> REPEAT 1.2  10 TIMES
787: C         3R101     --> REPEAT 101  3 TIMES
788: C
789: C   < ZEROS >
790: C
791: C   <AN INTEGER>Z MEANS REPEATE ZERO SPECIFIED TIMES.
792: C
793: C
794: C =====
795: C AUXILIARY ROUTINES & FUNCTIONS :
796: C
797: C   NEWLIN(IST) : GET NEWLINE. ( IST = 1 : END OF FILE )
798: C
799: C   E(D)AGETJ  : TRANSLATE A CHARACTER STRING ASSUMED TO
800: C               EXPRESS A FLOATING POINT NUMBER TO A FLOATING
801: C               POINT DATA OF SINGLE (DOUBLE) PRECISION.
802: C
803: C   IAGETJ     : TRANSLATE A CHARACTER STRING ASSUMED TO EXPRESS
804: C               AN INTEGER NUMBER TO AN INTEGER DATA OF 4-BYTES.
805: C
806: C
807: C =====
808: C STATE VARIABLES : ( SHOULD BE STATIC VARIABLES )
809: C
810: C   IPOS : CURRENT POSITION INDEX FOR AN INPUT LINE ( 1 TO 80 )
811: C   NREPI : REPETION COUNTER
812: C          TO BE SET WHEN ENCOUNTERED REPETITION AND
813: C          DECREMENTED TO BE ZERO.
814: C   NREPZ : REPETION COUNTER FOR ZERO REPETITION.
815: C   DREP  : DATA MEMORY FOR REPETITION MODE.
816: C          (DOUBLE PRECISION)
817: C   CARD  : BUFFER OF 80 BYTES TO INPUT A LINE
818: C
819: C =====
820: C <<  MODIFICATION HISTORY >>
821: C
822: C =====
823: C
824: C
825: C   subroutine SETUNT( NIN, IRST )
826: C
827: C
828: C   --- DATA INPT /5/, OUTP /6/ ETC. FOR USE IN OTHER THAN KENO-IV
829: C
830: C   common /READC1/ CARD,  COMMT,  TERM
831: C   character CARD*80, COMMT*1, TERM*1
832: C   common /READC2/ DREP,  IPOS,  IEND,  NREPI, NREPZ, IREP,
833: C   &      INPT,  IOUTP
834: C   real*8 DREP
835: C
836: C   INPT  = NIN
837: C   return
838: C   end
839: C
840: C -----
841: C
842: C   subroutine getlin(LINE,IST)
843: C
844: C   character*(*) LINE
845: C

```

```

846: C   parameter( MLCARD  = 80, ML1  = MLCARD + 1 )
847: C
848: C
849: C   --- DATA INPT /5/, OUTP /6/ ETC. FOR USE IN OTHER THAN KENO-IV
850: C
851: C
852: C   common /READC1/ CARD,  COMMT,  TERM
853: C   character CARD*80, COMMT*1, TERM*1
854: C   common /READC2/ DREP,  IPOS,  IEND,  NREPI, NREPZ, IREP,
855: C   &      INPT,  IOUTP
856: C   real*8 DREP
857: C
858: C   IRST = 0
859: C   call NEWLIN(IENDF)
860: C   if ( IENDF.ne.0 ) then
861: C
862: C   ***** END OF FILE *****
863: C
864: C       IRST  = -1
865: C       LINE  = ' '
866: C       return
867: C   end if
868: C   LINE = CARD
869: C   IPOS = 0
870: C   return
871: C   end
872: C
873: C -----
874: C
875: C   subroutine CREAD( STRING,IRST )
876: C
877: C   character*(*) STRING
878: C
879: C   parameter( MLCARD  = 80, ML1  = MLCARD + 1 )
880: C
881: C
882: C   --- DATA INPT /5/, OUTP /6/ ETC. FOR USE IN OTHER THAN KENO-IV
883: C
884: C
885: C   common /READC1/ CARD,  COMMT,  TERM
886: C   character CARD*80, COMMT*1, TERM*1
887: C   common /READC2/ DREP,  IPOS,  IEND,  NREPI, NREPZ, IREP,
888: C   &      INPT,  IOUTP
889: C   real*8 DREP
890: C
891: C   ---- BEGINNING OF INPUT ----
892: C
893: C       IRST  = 0
894: C
895: C   100 continue
896: C
897: C   ---- NEW LINE ----
898: C
899: C       if ( IPOS.eq.0 ) then
900: C   110       call NEWLIN( IENDF )
901: C           if ( IENDF.ne.0 ) then
902: C
903: C   ***** END OF FILE *****
904: C
905: C           IRST  = -1
906: C           STRING = ' '
907: C           return
908: C       end if
909: C   end if
910: C

```

src/tools/ntxt2lb.f

```

911: C
912: C
913:   if ( IEND.eq.1 ) then
914:     STRING = ' '
915: C
916: C ***** ENCOUNTERED TERMINATION CHARACTER *****
917: C
918:  IRST = 1
919:   return
920: end if
921: C
922:   IS = 0
923:   IE = 0
924: C
925:   ICOMMA = 0
926:   do 120 I = IPOS, MLCARD
927: C
928:     if ( CARD(I:I).ne.' ' ) then
929:       if ( CARD(I:I).eq.TERM ) then
930:         IEND = 1
931:         go to 130
932:       else if ( CARD(I:I).eq.COMMT ) then
933:         IPOS = 0
934:         go to 130
935:       else if ( IS.eq.0 ) then
936:         IS = I
937:       end if
938: C
939: C ---- SEPARATOR IS ' ' ----
940: C
941:       else if ( IS.ne.0 ) then
942:         IE = I - 1
943:         IPOS = I + 1
944: C
945:         if ( IPOS.gt.MLCARD ) IPOS = 0
946:         go to 130
947:       end if
948: 120 continue
949:       IPOS = 0
950:       if ( IS.ne.0 ) IE = I - 1
951: C
952: C
953: 130 continue
954: C
955:   if ( IS.eq.0.and.CARD(I:I).eq.COMMT ) go to 100
956: C
957: C
958:   if ( IS.ne.0.and.IE.ne.0 ) then
959:     STRING = CARD(IS:IE)
960:    IRST = 0
961:     if ( IE-IS+1.gt.LEN(STRING) )IRST = 2
962:   else
963:     STRING = ' '
964:    IRST = 1
965:   end if
966:   return
967: end
968: C
969: C -----
970: C
971: C ... simple and faster version of cread ...
972: C   input all tokens in a line with a call.
973: C   assuming no comment or termination characters
974: C   return number of token and token positions in string
975: C

```

```

976: C   length of string must be greater than mlc card
977: C
978: C   subroutine CREADS( STRING,NTOKEN,ISS,   IEE,  IRST )
979: C
980: C
981: C   character*(*) STRING
982: C   integer ISS(*), IEE(*)
983: C
984: C
985: C   parameter( MLCARD = 80, ML1 = MLCARD + 1 )
986: C
987: C
988: C   --- DATA INPT /5/, OUTP /6/ ETC. FOR USE IN OTHER THAN KENO-IV
989: C
990: C
991: C   common /READC1/ CARD, COMMT, TERM
992: C   character CARD*80, COMMT*1, TERM*1
993: C   common /READC2/ DREP, IPOS, IEND, NREPI, NREPZ, IREP,
994: C   & INPT, IOUTP
995: C   real*8 DREP
996: C
997: C ----- BEGINNING OF INPUT -----
998: C
999:  IRST = 0
1000:  NTOKEN = 0
1001: C
1002: C ---- NEW LINE ----
1003: C
1004:   call NEWLIN( IENDF )
1005:   if ( IENDF.ne.0 ) then
1006: C
1007: C ***** END OF FILE *****
1008: C
1009:  IRST = -1
1010:   return
1011: end if
1012: C
1013:   STRING(:MLCARD) = CARD
1014:   STRING(ML1:ML1) = ' '
1015: C
1016:   NTOKEN = 0
1017:   JS = 0
1018: C
1019: C
1020: C
1021:   do 100 I = IPOS, ML1
1022: C
1023:     if ( JS.eq.0 ) then
1024:       if ( STRING(I:I).ne.' ' ) then
1025:         NTOKEN = NTOKEN + 1
1026:         ISS(NTOKEN) = I
1027:         JS = 1
1028:       end if
1029:     else if ( STRING(I:I).eq.' ' ) then
1030:       IEE(NTOKEN) = I - 1
1031:       JS = 0
1032:     end if
1033: 100 continue
1034:   IPOS = 0
1035: C
1036:   return
1037: end
1038: C
1039: C =====
1040: C

```

src/tools/ntxt2lb.f

```

1041:      subroutine SREAD( STRING,NCHAR, IRST )
1042: C
1043: C
1044:      character*(*) STRING
1045: C
1046:      parameter( MLCARD   = 80, ML1   = MLCARD + 1 )
1047: C
1048: C
1049: C --- DATA INPT /5/, OUTP /6/ ETC. FOR USE IN OTHER THAN KENO-IV
1050: C
1051: C
1052:      common /READC1/  CARD,   COMMT,  TERM
1053:      character CARD*80, COMMT*1, TERM*1
1054:      common /READC2/  DREP,   IPOS,   IEND,  NREPI, NREPZ, IREP,
1055:      & INPT,  IOUTP
1056:      real*8 DREP
1057: C
1058:      character*80 BUF
1059:      character*1 TPTERM
1060: C
1061: C ----- BEGINNING OF INPUT -----
1062: C
1063:      IRST   = 0
1064:      LS     = LEN(STRING)
1065:      NCHAR  = 0
1066:      TPTERM = ' '
1067: C
1068: C .... END OF FILE OR NEWLINE CALL NECESSARY ....
1069: *      CALL CREAD( BUF, IRST )
1070: *      IF( IRST.EQ.-1 .OR. IRST.EQ. 1 ) THEN
1071: *          NCHAR = 0
1072: *          RETURN
1073: *      ELSE
1074: *          TPTERM = BUF(1:1)
1075: *          K = INDEX(BUF,' ')
1076: C
1077:      100 continue
1078: C
1079: C----- NEW LINE -----
1080: C
1081:      if ( IPOS.eq.0 ) then
1082:          call NEWLIN( IENDF )
1083:          if ( IENDF.ne.0 ) then
1084:              IRST   = -1
1085:              return
1086:          end if
1087:      end if
1088: C
1089: C
1090: C
1091:      if ( IEND.eq.1 ) then
1092:          IRST   = 1
1093:          return
1094:      end if
1095: C
1096:      IS      = 0
1097:      IE      = 0
1098: C
1099:      do 110 I = IPOS, MLCARD
1100: C
1101:          if ( TPTERM.eq.' ' .and.CARD(I:I).ne.' ' ) then
1102:              if ( CARD(I:I).ne.TERM ) then
1103:                  TPTERM = CARD(I:I)
1104:              else
1105:                  IEND   = 1

```

```

1106:                  IRST   = 1
1107:                  go to 120
1108:              end if
1109: C
1110:          else if ( TPTERM.ne.' ' ) then
1111:              if ( CARD(I:I).ne.TPTERM ) then
1112:                  NCHAR   = NCHAR + 1
1113:                  if ( NCHAR.le.LS ) STRING(NCHAR:NCHAR) = CARD(I:I)
1114:              else
1115:                  IPOS     = I + 1
1116:                  go to 120
1117:              end if
1118:          end if
1119: C
1120:      110 continue
1121:      IPOS   = 0
1122:      go to 100
1123: C
1124: C
1125:      120 continue
1126:      if ( IPOS.gt.MLCARD ) IPOS = 0
1127:      if ( NCHAR.gt.LS ) then
1128:          NCHAR = LS
1129:          IRST  = 2
1130:      end if
1131: C
1132:      return
1133: end
1134: C
1135: C
1136: C =====
1137: C
1138:      subroutine IREAD( IN,  IRST )
1139: C
1140:      parameter( MLCARD   = 80, ML1   = MLCARD + 1 )
1141: C
1142:      character*10 TYPE
1143:      character*80 TOKEN
1144: C
1145:      common /READC1/  CARD,   COMMT,  TERM
1146:      character CARD*80, COMMT*1, TERM*1
1147:      common /READC2/  DREP,   IPOS,   IEND,  NREPI, NREPZ, IREP,
1148:      & INPT,  IOUTP
1149:      real*8 DREP
1150: C
1151:      real*8 DIN, DRD
1152:      real*8 DAGETJ
1153: C
1154:      IENT   = 1
1155:      TYPE   = 'INTEGER'
1156:      go to 100
1157: C
1158:      entry EREAD(EIN,IRST)
1159:      IENT   = 2
1160:      TYPE   = 'E-FLOAT'
1161:      go to 100
1162: C
1163:      entry DREAD(DIN,IRST)
1164:      IENT   = 3
1165:      TYPE   = 'D-FLOAT'
1166:      go to 100
1167: C
1168:      100 continue
1169: C
1170:      IRST   = 0

```

src/tools/ntxt2lb.f

```

1171: C
1172: C
1173: C<<<< IN REPETITION MODE
1174: C
1175: C
1176:       if ( NREPI.gt.0 ) then
1177:         if ( IENT.eq.1 ) IN = INT(DREP)
1178:         if ( IENT.eq.2 ) EIN = REAL(DREP)
1179:         if ( IENT.eq.3 ) DIN = DREP
1180:         NREPI = NREPI - 1
1181:         return
1182:       end if
1183:       if ( NREPZ.gt.0 ) then
1184:         if ( IENT.eq.1 ) IN = 0
1185:         if ( IENT.eq.2 ) EIN = 0.0E0
1186:         if ( IENT.eq.3 ) DIN = 0.0D0
1187:         NREPZ = NREPZ - 1
1188:         return
1189:       end if
1190: C
1191: C
1192: C
1193: C<<<< GET TOKEN FROM INPUT FILE
1194: C
1195: C
1196:       110 call CREAD( TOKEN, ISTAT )
1197: C
1198: C ---- DATA END -----
1199: C
1200:       if ( ISTAT.ne.0 ) then
1201:         IRST = ISTAT
1202:         return
1203:       end if
1204: C
1205: C
1206: C
1207: C<<<< CHECK REPETITION :
1208: C
1209: C
1210:       IR = 0
1211: C
1212:       LENGTH = INDEX(TOKEN,' ') - 1
1213: C
1214:       do 120 I = 1, LENGTH
1215:         if ( INDEX('**RR$Z',TOKEN(I:I)).ne.0 ) then
1216:           IR = I
1217:           go to 130
1218:         end if
1219:       120 continue
1220: C
1221:       130 if ( IR.ne.0 ) then
1222:         NREPI = IAGETJ(TOKEN(1:IR-1),IFLAG)
1223: C
1224: C ..... FOUND AN INVALID CHARACTER ....
1225:       if ( IFLAG.gt.0 ) then
1226:         write(IOUTP,7000) CARD
1227:         write(IOUTP,7020) IPOS + IFLAG - 1, TOKEN(IFLAG:IFLAG), TYPE
1228:         IRST = 1
1229:         return
1230:       end if
1231:       end if
1232: C
1233: C
1234: C<<<< CHECK FOR ZERO REPETITION
1235: C

```

```

1236: C
1237:       if ( IR.ne.0.and.TOKEN(IR:IR).eq.'Z' ) then
1238:         NREPZ = NREPI - 1
1239:         NREPI = 0
1240:         if ( IENT.eq.1 ) IN = 0
1241:         if ( IENT.eq.2 ) EIN = 0.0E0
1242:         if ( IENT.eq.3 ) DIN = 0.0D0
1243:         return
1244:       end if
1245: C
1246: C
1247: C<<<< GET AN VALUE <<<<
1248: C
1249: C
1250: C
1251:       if ( IENT.eq.1 ) IRD = IAGETJ(TOKEN(IR+1:LENGTH),IFLAG)
1252:       if ( IENT.eq.2 ) ERD = EAGETJ(TOKEN(IR+1:LENGTH),IFLAG)
1253:       if ( IENT.eq.3 ) DRD = DAGETJ(TOKEN(IR+1:LENGTH),IFLAG)
1254: C
1255: C
1256: C ..... AN INVALID CHARACTER WAS FOUND ....
1257:       if ( IFLAG.gt.0 ) then
1258:         write(IOUTP,7000) CARD
1259:         write(IOUTP,7020) IPOS + IFLAG - 1, TOKEN(IFLAG:IFLAG), TYPE
1260:         IRST = 1
1261:         return
1262:       end if
1263: C
1264: C
1265: C
1266: C<<<< REPETITION MEMORY
1267: C
1268: C
1269:       if ( NREPI.gt.0 ) then
1270:         if ( IENT.eq.1 ) DREP = IRD
1271:         if ( IENT.eq.2 ) DREP = ERD
1272:         if ( IENT.eq.3 ) DREP = DRD
1273:         NREPI = NREPI - 1
1274:       end if
1275: C
1276:       IRST = 0
1277:       if ( IENT.eq.1 ) IN = IRD
1278:       if ( IENT.eq.2 ) EIN = ERD
1279:       if ( IENT.eq.3 ) DIN = DRD
1280: C
1281:       7000 format('0*****ERROR IN INPUT. CARD IMAGE PRINTED ON NEXT LINE ',
1282:       & '***** '/10X,A80)
1283:       7020 format(' ON THE ABOVE CARD, CHARACTER NUMBER ',I2,' (IMAGE=',A1,
1284:       & ' ) IS NOT VALID IN AN ',A,' FIELD.')
1285:       return
1286:       end
1287: C
1288: C =====
1289: C
1290:       block data
1291: C
1292:       parameter( MLCARD = 80, ML1 = MLCARD + 1 )
1293: C
1294:       common /READC1/ CARD, COMMT, TERM
1295:       character CARD*80, COMMT*1, TERM*1
1296:       common /READC2/ DREP, IPOS, IEND, NREPI, NREPZ, IREP,
1297:       & INPT, IOUTP
1298:       real*8 DREP
1299: C
1300:       data IPOS /0/

```


src/tools/ntxt2lb.f

```

1301:      data NREPI, NREPZ, IREP /0, 0, 0/
1302:      data DREP /0.0/
1303: C
1304:      data INPT /5/, IOUTP /6/
1305: C
1306:      data TERM '//', COMMT '/'#'/
1307:      end
1308: C
1309: C
1310: C ----- GET A NEW LINE -----
1311: C
1312: C
1313:      subroutine NEWLIN( IENDF )
1314: C
1315:      common /READC1/ CARD, COMMT, TERM
1316:      character CARD*80, COMMT*1, TERM*1
1317:      common /READC2/ DREP, IPOS, IEND, NREPI, NREPZ, IREP,
1318:      & INPT, IOUTP
1319:      real*8 DREP
1320: C
1321:      CARD = ' '
1322:      IENDF = 0
1323:      100 read(INPT,fmt = '(A)',end =110) CARD
1324:      if ( CARD(1:1).eq.COMMT ) go to 100
1325:      IPOS = 1
1326:      IEND = 0
1327:      return
1328: C
1329:      110 IENDF = 1
1330:      return
1331:      end
1332: C
1333: C
1334: C
1335: C
1336:      function IAGETJ(TOKEN,IFLAG)
1337: C
1338:      character TOKEN*(*)
1339: C
1340:      character NUMBER*12
1341: C
1342: C CHARACTER FORMT*16
1343:      character CTEMP*16
1344: C
1345:      data NUMBER /'+-0123456789'/
1346: C
1347:      IFLAG = 0
1348: C
1349:      CTEMP = TOKEN
1350:      read(CTEMP,7000,err =100) IAGETJ
1351:      7000 format(BN,I16)
1352:      return
1353: C
1354:      100 continue
1355:      IAGETJ = -999
1356:      do 110 I = 1, LEN(TOKEN)
1357:          if ( INDEX(NUMBER,TOKEN(I:I)).eq.0 ) then
1358:              IFLAG = I
1359:              return
1360:          end if
1361:      110 continue
1362:      IFLAG = -999
1363:      end
1364: C
1365: C -----

```

```

1366: C
1367:      function EAGETJ(TOKEN,IFLAG)
1368: C
1369: C
1370:      character TOKEN*(*)
1371:      character FLOATN*17
1372:      character CTEMP*24
1373: C
1374:      data FLOATN /'0123456789.EeDd+-' /
1375: C
1376:      IFLAG = 0
1377: C
1378:      CTEMP = TOKEN
1379:      read(CTEMP,7000,err =100) EAGETJ
1380:      7000 format(BN,E24.0)
1381:      return
1382: C
1383: C
1384:      100 continue
1385: C
1386:      write(6,*) 'ERROR in EAGETj', TOKEN
1387: C
1388:      do 110 I = 1, LEN(TOKEN)
1389:          if ( TOKEN(I:I).ne.' ' ) then
1390:              if ( INDEX(FLOATN,TOKEN(I:I)).eq.0 ) then
1391:                  IFLAG = I
1392:                  return
1393:              end if
1394:          end if
1395:      110 continue
1396:      IFLAG = -999
1397:      return
1398:      end
1399: C
1400: C -----
1401: C
1402:      function DAGETJ(TOKEN,IFLAG)
1403: Ccccc DOUBLE PRECISION DAGETJ
1404:      real*8 DAGETJ
1405: C
1406: C
1407:      character TOKEN*(*)
1408:      character FLOATN*17
1409:      character CTEMP*32
1410: C
1411:      data FLOATN /'0123456789.Dede+-' /
1412: C
1413: C
1414:      IFLAG = 0
1415: C
1416:      CTEMP = TOKEN
1417:      read(CTEMP,7000,err =100) DAGETJ
1418:      7000 format(BN,D32.0)
1419:      return
1420: C
1421: C
1422:      100 continue
1423:      do 110 I = 1, LEN(TOKEN)
1424:          if ( INDEX(FLOATN,TOKEN(I:I)).eq.0 ) then
1425:              IFLAG = I
1426:              return
1427:          end if
1428:      110 continue
1429:      IFLAG = -999
1430:      end

```

src/tools/ntxt2lb.f

```

1431: C
1432: C
1433: C ===== pattern matching routine ( MVP/GMVP utility )
1434: C
1435: C
1436: C      subroutine MATCH( IM,      PATTRN,STR )
1437: C
1438: C      MVP/GMVP UTILITY
1439: C
1440: C=====
1441: C PURPOSE: STRING PATTERN MATCHING
1442: C HISTORY: PROGRAMMED BY M.SASAKI (30 APR 1992)
1443: C UPDATE:
1444: C 7 JUL 1993: '*' matches any string including null string.
1445: C      More sophisticated matching logic that allows any
1446: C      combinations of wildcard meta-characters.
1447: C=====
1448: C
1449: C      PATTRN & STR CAN INCLUDE BLANKS BUT THEY MUST BE PLACED AFTER ANY
1450: C      NON-BLANK CHARACTERS. ( ignore characters after blank !! )
1451: C
1452: C      IM = 1 / 0 ... match / unmatched
1453: C
1454: C
1455: C << WILDCARD CHARACTERS >>
1456: C
1457: C '?' : ANY ONE CHARACTER ON THE PLACE OF '?'.
1458: C '*' : ANY CHARACTER STRING ( length >= 0 ).
1459: C
1460: C=====
1461: C      character*(*) PATTRN
1462: C      character*(*) STR
1463: C
1464: C      .... default is unmatched ...
1465: C
1466: C      IM      = 0
1467: C
1468: C      LP      = INDEX(PATTRN,' ') - 1
1469: C      if ( LP.lt.0 ) LP = LEN(PATTRN)
1470: C
1471: C      LS      = INDEX(STR,' ') - 1
1472: C      if ( LS.lt.0 ) LS = LEN(STR)
1473: C      if ( LS.eq.0 ) return
1474: C
1475: C
1476: C
1477: C      .... pattern has no '*' .....
1478: C
1479: C
1480: C
1481: C      if ( INDEX(PATTRN(:LP),'*').eq.0 ) then
1482: C      if ( LP.ne.LS ) return
1483: C      do 100 I = 1, LP
1484: C      if ( PATTRN(I:I).ne.'?' .and. PATTRN(I:I).ne.STR(I:I) ) return
1485: C 100 continue
1486: C      IM      = 1
1487: C      return
1488: C      end if
1489: C
1490: C
1491: C      .... pattern has '*' ....
1492: C
1493: C
1494: C << Logic of pattern matching with wild-card character '*' >>
1495: C

```

```

1496: C
1497: C
1498: C
1499: C      0. Set "current pattern" to the whole pattern string.
1500: C      And set "object string" to the string checked.
1501: C
1502: C      1. Split the current-pattern string to two sub-patterns by '*'.
1503: C
1504: C      pattern = L * R
1505: C      or L      ( if the current pattern has no '*' )
1506: C
1507: C      L : left-side sub-pattern (does not include '*'. can be
1508: C      null string)
1509: C      R : right-side sub-pattern (may include '*')
1510: C
1511: C      2. If the left-side sub-pattern is not null string,
1512: C      check matching with the left-side sub-pattern.
1513: C
1514: C      If the sub-pattern does not match any substring of the object
1515: C      string (the leftmost substring for the first time), terminate as
1516: C      'unmatching'.
1517: C
1518: C      Else cut off the matching substring and characters on the left
1519: C      of the substring from the object string and proceed to step 4.
1520: C
1521: C      3. If the pattern has no '*',
1522: C      check matching between the current pattern and the rightmost
1523: C      substring of the object string.
1524: C
1525: C      If matched ,terminate matching as 'matching',else terminate
1526: C      matching as 'unmatching'.
1527: C
1528: C      4. Set the right-side sub-pattern as the current pattern and
1529: C      repeat the process from 1.
1530: C
1531: C      IP      = 1
1532: C      IS      = 1
1533: C
1534: C 110 continue
1535: C
1536: C
1537: C      K      = INDEX(PATTRN(IP:LP),'*')
1538: C
1539: C
1540: C      ==== check matching with the left side of current '*'
1541: C
1542: C
1543: C      if ( K.gt.1 ) then
1544: C
1545: C      IS1     = LS - K + 2
1546: C
1547: C      .... for the first matching, the sub-pattern must match
1548: C      at the leftmost position.
1549: C
1550: C      if ( IP.eq.1 ) IS1 = IS
1551: C      do 130 ISS = IS, IS1
1552: C      do 120 I = 0, K - 2
1553: C      if ( PATTRN(IP+I:IP+I).ne.'?'
1554: C      & .and. PATTRN(IP+I:IP+I).ne.STR(ISS+I:ISS+I) ) go to 130
1555: C 120 continue
1556: C
1557: C      ----- left side matches -----
1558: C
1559: C      IS      = ISS + I
1560: C      go to 150

```

src/tools/ntxt2lb.f

```
1561: C
1562: 130    continue
1563: C
1564: C  ----- unmatched -----
1565: C
1566:         return
1567: C
1568: C
1569: C  ==== check the rightmost string if no '*' remains in pattern ====
1570: C
1571: C
1572:         else if ( K.eq.0 ) then
1573:             if ( LS-IS.lt.LP-IP ) return
1574:             do 140 I = IP, LP
1575:                 if ( PATRN(I:I).ne.'?'
1576:                     & .and.PATRN(I:I).ne.STR(LS+I-LP:LS+I-LP) ) return
1577:             140 continue
1578:             IM = 1
1579:             return
1580:         end if
1581: C
1582: C  (if the left-side sub-pattern is null, do only the following
1583: C  step.)
1584: C
1585: 150 IP = IP + K
1586:     if ( IP.le.LP ) go to 110
1587: C
1588: C  ---- matched !! ----
1589: C
1590:     IM = 1
1591: C
1592:     return
1593: end
```

src/tools/permu.f

```

1: C
2: C   LIST UP PERMUTATION
3: C
4: C   Programmed by M.SASAKI on unknown date in 1980's for unknown
5: C   purpose, and programming logic is unknown even to the programmer
6: C   himself...
7: C
8:     parameter( NN   = 16 )
9:     dimension JF(NN), IP(NN), IJ(NN)
10:    character*(NN) CH, JCH, JK(NN)
11:    character*70 ICH
12:
13: 100 ICH   = ' '
14:    ICH   = ' '
15:    do 110 I = 1, NN
16:      IP(I) = 0
17:      IJ(I) = 0
18:      JK(I) = ' '
19:      JF(I) = 0
20: 110 continue
21: CCC
22: CCC   READ(5,*) M
23:     read(5,'(A)') CH
24:     if ( CH.eq.' ' ) go to 160
25:     M   = INDEX(CH,' ') - 1
26:     if ( M.eq.-1 ) M   = NN
27:     NL  = 70/(M+1)
28: CCC
29:     L   = 1
30:     J   = 0
31:     LL  = 0
32:     IP(1) = 0
33: CCC
34: 120 do 130 I = IP(L) + 1, M
35:     if ( JF(I).eq.0.and.INDEX(JK(L),CH(I:I)).eq.0 ) go to 140
36: 130 continue
37:     JF(IP(L)) = 0
38:     IP(L) = 0
39:     IJ(L) = 0
40:     JK(L) = ' '
41:     L = L - 1
42:     if ( L.eq.0 ) go to 150
43:     go to 120
44: CCC
45: 140 JF(I) = 1
46:     if ( IP(L).ne.0 ) JF(IP(L)) = 0
47:     IP(L) = I
48:     JCH(L:L) = CH(I:I)
49:     IJ(L) = IJ(L) + 1
50:     JK(L) (IJ(L):IJ(L)) = CH(I:I)
51:     L = L + 1
52: C
53:     if ( L.gt.M ) then
54:       J = J + 1
55:       LL = LL + 1
56:       if ( LL.gt.NL ) then
57:         write(6,'(A70)') ICH
58:         ICH = ' '
59:         LL = 1
60:       end if
61:       KK = (M+1)*(LL-1) + 2
62:       ICH(KK:KK+M-1) = JCH(1:M)
63:       L = L - 1
64:     end if
65:     go to 120
66: CCC
67: 150 if ( LL.gt.0 ) write(6,'(A70)') ICH
68:     write(6,*) ' NUMBER OF PERMUTATIONS = ', J
69:     go to 100
70: 160 stop
71:     end

```

src/tools/rstchk.f

```

1: C=====
2: C
3: C Restart file checker of MVP/GMVP
4: C
5: C Check headers of MVP/GMVP restart file and output # of batches etc.
6: C
7: C=====
8: C
9: C      program RSTCHK
10: C
11: C
12: C      CHARACTER*128 CSTRNG
13: C
14: C      NFILE      = 0
15: C-----
16: C Program control information from command line arguments.
17: C-----
18: C
19: C/#IF ARGV
20: C
21: C/# IF ARGV( GETARG2 )
22: *      KJ = IARGC() - 1
23: C/# ELSEIF ARGV( MSF )
24: *      KJ = NARGS() - 1
25: C/# ELSE
26:      KJ      = IARGC()
27: C/# ENDIF
28: C
29: C
30: C      do 100 I = 1, KJ
31: C          IPREIN = IPREIN + 1
32: C
33: C/# IF SYSTEM( CRAY )
34: *      call PXFGETARG(I, CSTRNG, LLLLL, ierr )
35: C/# ELSEIF ARGV( IGETARG )
36: *      LLLLL = IGETARG(I, CSTRNG, len(cstrng) )
37: C/# ELSE
38: C/# IF ARGV( GETARG2 )
39: *      call GETARG(I+1, CSTRNG )
40: C/# ELSEIF ARGV( MSF )
41: C      ... LL2 should be integer*2
42: *      call GETARG(I, CSTRNG, LL2 )
43: C/# ELSE
44:      call GETARG( I, CSTRNG )
45: C/# ENDIF
46: C/# ENDIF
47: C
48: C      NFILE      = NFILE + 1
49: C
50: C      call CHECKR( CSTRNG )
51: C
52: C      100 continue
53: C/#ENDIF
54: C-----
55: C-----
56: C ..... input file name from stdin if no argument is specified or
57: C      command argument handling is disabled ...
58: C-----
59: C
60: C      if ( NFILE.eq.0 ) then
61: C
62: C      110 write(*,*) '>> Restart file name -->'
63: C
64: C      CSTRNG = ' '
65: C      read(*,fmt='(a)',end=120) CSTRNG

```

```

66: C
67: C      call CHECKR( CSTRNG )
68: C
69: C      go to 110
70: C
71: C      120 continue
72: C      end if
73: C
74: C      stop
75: C      end
76: C
77: C=====
78: C
79: C      subroutine CHECKR( FILE )
80: C
81: C      character*(*) FILE
82: C      parameter( IRIN = 20 )
83: C      logical OPENED
84: C
85: C      character*72 TITLE(2)
86: C      character*60 PROGV(40)
87: C      real*8 WSUM, WLEK
88: C
89: C      character*80 HD
90: C
91: C-----
92: C
93: C      LLLLL = MIN(INDEX(FILE,' ')-1,LEN(FILE))
94: C      if ( LLLLL.eq.0 ) LLLLL = LEN(FILE)
95: C
96: C      inquire(unit =IRIN,opened =OPENED)
97: C      if ( OPENED ) close( IRIN )
98: C
99: C      open( unit =IRIN, form = 'UNFORMATTED', file =FILE(:LLLLL), iostat
100: C      &      =ICODE, status = 'OLD' )
101: C
102: C      if ( ICODE.ne.0 ) then
103: C          write(*,*) 'XXX Failed to open file <', FILE(:LLLLL), '>',
104: C      &      ' error code (IOSTAT) = ', ICODE
105: C          return
106: C      end if
107: C
108: C-----
109: C ..... file header (80 CHARACTERS) .....
110: C-----
111: C      read(IRIN) HD
112: C      write(*,7000) HD
113: C      7000 format(/3X,'=== HEADER OF RESTART FILE ==='/1X,' <',A,'>'/)
114: C
115: C-----
116: C ..... problem title .....
117: C-----
118: C      read(IRIN) (TITLE(I),I=1,2)
119: C      write(*,7020) (TITLE(I),I=1,2)
120: C      7020 format(/1X,'==== PROBLEM TITLE ==== '/1X,' <',
121: C      &      A72,'>'/)
122: C
123: C-----
124: C ..... program version description. ....
125: C-----
126: C
127: C      read(IRIN) (PROGV(I),I=1,40)
128: C      write(*,7040) PROGV(40)
129: C      7040 format(/1X,'==== FILE VERSION ==== '/1X,' <',A64,'>'/)
130: C

```

src/tools/rstchk.f

```
131: C=====
132: C Restart file version 3.0 (from augst 1997)
133: C=====
134: C
135: C-----
136: C ... number of histories etc.( sum of all tasks ) ....
137: C
138: C MRBUF: read in buffer size of chopped restrt file records.
139: C-----
140: C if ( INDEX(HD,'MVP RESTART FILE V3.0').eq.1
141: C & .or. INDEX(HD,'GMVP RESTART FILE V3.0').eq.1 ) then
142: C
143: C read(IRIN) NTASK, NTHIST, NBATCH, NRBUF
144: C write(*,7060) NTASK, NTHIST, NBATCH, NRBUF
145: 7060 format(/3X,'=== RESTART FILE version 3.0 ==='/1X,
146: C & ' NUMBER OF TASKS (NTASK) = ',I10/1X,
147: C & ' NUMBER OF HISTORIES (NTHIST) = ',I10/1X,
148: C & ' NUMBER OF BATCHES (NBATCH) = ',I10/1X,
149: C & ' INPUT BUFFER SIZE (NRBUF) = ',I10/1X)
150: C
151: C-----
152: C ..... write non-variable parameters .....
153: C-----
154: C
155: C write(IROT) NGROUP, NPKIND, NREG, NTREG, NRESP, NMEMO, NZONE
156: C & NFBANK, NLATT, NEST, NUC, NEMIC, NENAC, NSTAL,
157: C & NETALY, JEIGN, NRESP, JHLAT, JTLLT
158: C
159: C=====
160: C Restart file version 2.1 (from may 1994)
161: C=====
162: C
163: C else if ( INDEX(HD,'MVP RESTART FILE V2.1').eq.1
164: C & .or. INDEX(HD,'GMVP RESTART FILE V2.1').eq.1 ) then
165: C
166: C read(IRIN) NTHIST, WSUM, IRAND, NBATCH, NFISB, WLEK
167: C
168: C write(*,7080) NTHIST, WSUM, IRAND, NBATCH, NFISB, WLEK
169: 7080 format(/3X,'=== RESTART FILE version 2.1 ==='/1X,
170: C & ' NUMBER OF HISTORIES (NTHIST) = ',I15/1X,
171: C & ' SOURCE WEIGHT (WSUM) = ',D15.7/1X,
172: C & ' RANDOM NUMBER (IRAND) = ',I15/1X,
173: C & ' NUMBER OF BATCHES (NBATCH) = ',I15/1X,
174: C & ' FISSION SOURCE (NFISB) = ',I15/1X,
175: C & ' LEAKED WEIGHT (WLEK) = ',D15.7/1X)
176: C
177: C else
178: C
179: C=====
180: C Unknown restart file version
181: C (or invalid file, binary from incompatible architecture etc.)
182: C=====
183: C
184: C write(*,7100)
185: 7100 format(/3X,'=== Unrecognizable restart file ==='/)
186: C
187: C end if
188: C
189: C return
190: C end
```

src/tools/varlist.f

```

1: C -----
2: C
3: C --- CREATE VARIABLE TABLE FROM COMMENTS FOR COMMONS (GMVP/MVP) ---
4: C
5: C -----
6: C
7: C * PROGRAMMED BY M.SASAKI ( 6 APR 1992 )
8: C * modified not to use FREAD routines ( 20 July 1998 )
9: C
10: C -----
11: C
12: C * REQUIREMENTS FOR INPUT DATA:
13: C
14: C COMMON & COMMENTS AS FOLLOWS:
15: C
16: C INPUT : EXPLANATION
17: C :
18: C C* : 'C*' IN COLUMN 1-2 IGNORED.
19: C C* :
20: C COMMON /XXXX/ ..... : COMMON NAMED XXXX
21: C ..... : (ONLY 1 COMMON STATEMENT)
22: C C* :
23: C C* ... :
24: C C VNAME TYPE COMMENTS $ : VNAME(.....) :
25: C C $ COMMENTS : BEGINS AT COLUMN 2
26: C C COMMENTS : ( ... ) OPTIONAL.
27: C C ..... : $ : MEANS LINE CONNECTION.
28: C C VNAME2 TYPE COMMENTS ... :
29: C C ..... :
30: C (REPEAT ARBITRARY NUMBER OF SETS )
31: C
32: C -----
33: C
34: C * FILES :
35: C
36: C UNIT 10 (INPUT) -----+
37: C |--> UNIT 20 (TABLE)
38: C UNIT 5 (CONTROL INPUT) ----+ (UNIT 6 CHECK)
39: C
40: C -----
41: C
42: C ..... VARIABLE NAME TABLE & COUNTER ...
43: C
44: C parameter( MAXVAR = 1000, LNVAR = 8 )
45: C
46: C integer NVARs
47: C character*(LNVAR) VNAME(MAXVAR)
48: C character*6 TYP(MAXVAR)
49: C character*(LNVAR) COM(MAXVAR)
50: C
51: C ..... VARIABLE DATA POINTER IN SCRATCH FILE (DIRECT ACCESS)
52: C & LENGTH OF DATA
53: C
54: C integer IPVAR(MAXVAR+1)
55: C integer LDATA(MAXVAR)
56: C
57: C ..... BUFFER OF DATA FOR A VARIABLE.
58: C
59: C parameter( MAXBUF = 50 )
60: C parameter( MAXBUF = 150 )
61: C character*(72*MAXBUF) DATA
62: C
63: C ..... CHARACTER STRINGS TO STORE TEMPORARY DATA ...
64: C
65: C character LINE*72, TEMP*72, CMN*(LNVAR)

```

```

66: C
67: C ..... WORKING ARRAY FOR SORTING .....
68: C
69: C integer ISORT(MAXVAR)
70: C
71: C =====
72: C
73: C --- INPUT CONTROL DATA ----
74: C
75: C call FRINIT( 5, 6 )
76: C call RIUNIT( 5 )
77: C
78: C IORD = 1
79: C MARGIN = 8
80: C NCOLMN = 64
81: C
82: C 100 call FREADS( LINE, LN, IEND )
83: C if ( IEND.eq.0 ) then
84: C
85: C if ( LINE(:LN).eq.'VARIABLE' ) then
86: C IORD = 1
87: C else if ( LINE(:LN).eq.'COMMON' ) then
88: C IORD = 2
89: C else if ( LINE(:LN).eq.'MARGIN' ) then
90: C call I4READ( 'MARGIN', MARGIN, 1, NB, IERR )
91: C else if ( LINE(:LN).eq.'COLUMN' ) then
92: C call I4READ( 'COLUMN', NCOLMN, 1, NB, IERR )
93: C end if
94: C
95: C go to 100
96: C end if
97: C 100 line = ' '
98: C READ(5,'(a)',end=105) LINE
99: C
100: C if ( line(1:1).eq.'*' ) goto 100
101: C
102: C if ( LINE.eq.'VARIABLE' ) then
103: C IORD = 1
104: C else if ( LINE.eq.'COMMON' ) then
105: C IORD = 2
106: C else
107: C call ccomp( line, len(line), line, ifl )
108: C if ( LINE(:7).eq.'MARGIN=' ) then
109: C READ(line(:8),'(bn,i16)') MARGIN
110: C else if ( LINE(:7).eq.'COLUMN=' ) then
111: C READ(line(:8),'(bn,i16)') NCOLMN
112: C end if
113: C end if
114: C go to 100
115: C
116: C 105 continue
117: C
118: C ---- OPEN SCRATCH FILE ----
119: C
120: C LRECL = 160
121: C
122: C open( unit =15, access = 'DIRECT', form = 'UNFORMATTED', status =
123: C & 'UNKNOWN', recl = LRECL, iostat = IOS )
124: C
125: C if ( IOS.ne.0 ) then
126: C write(6,*) 'XXX SCRATCH FILE OPEN ERROR !! : IOS=', IOS
127: C stop
128: C end if
129: C
130: C

```

src/tools/varlist.f

```

131: C ==== INPUT LOOP =====
132: C
133: C -----
134: C DATA OUTPUT ON SCRATCH FILE : (UNIT 15)
135: C
136: C      ( CHAR(0) : NULL CHARACTER FOR DATA SEPARATION. )
137: C
138: C
139: C FOR EACH VARIABLE ;
140: C
141: C 1. VARIABLE NAME (WITH DIMENSION SPECIFICATION) + CHAR(0).
142: C 2. TYPE + CHAR(0).
143: C 3. COMMENT PARAGRAPHS SEPARATED BY CHAR(0) ('S).
144: C (SUCCESSIVE CHAR(0)'S ARE TREATED AS ONE CHAR(0).)
145: C -----
146: C
147: C
148: C NVARS = 0
149: C IDATA = 0
150: C JEND = 0
151: C
152: 110 read(10,'(A)',end =120) LINE
153: go to 130
154: C
155: C .... END OF INPUT ENCOUNTERED .....
156: C
157: 120 JEND = 1
158: 130 continue
159: C
160: C
161: C
162: C ..... NEW VARIABLE ....
163: C
164: C if ( JEND.ne.0 .or. LINE(1:2).eq.'C ' .and.LINE(3:3).ne.' ' ) then
165: C
166: C .... PROCESS PREVIOUS DATA ....
167: C
168: C if ( NVARS.ne.0 ) then
169: C
170: C call ADDLIN( DATA, IDATA, CHAR(0) )
171: C if ( NVARS.eq.1 ) then
172: C IPVAR(NVARS) = 1
173: C else
174: C inquire(unit =15,nextrec =IPVAR(NVARS))
175: C end if
176: C
177: C IIREC = IPVAR(NVARS)
178: C do 140 K = 1, IDATA, LRECL
179: C K2 = MIN(IDATA,K+LRECL-1)
180: C write(15,rec =IIREC,iostat =IOS) DATA(K:K2)
181: C if ( IOS.ne.0 ) then
182: C write(6,*) ' XXX ERROR IN OUTPUT ON SCRATCH FILE:',
183: C & ' REC= ', IPVAR(NVARS), ' IOS= ', IOS
184: C & stop 999
185: C end if
186: C IIREC = IIREC + 1
187: 140 continue
188: C
189: C LDATA(NVARS) = IDATA
190: C end if
191: C
192: C if ( JEND.ne.0 ) go to 150
193: C
194: C
195: C

```

```

196: NVARS = NVARS + 1
197: C
198: IDATA = 0
199: C
200: C
201: C if ( NVARS.gt.MAXVAR ) then
202: C write(6,*) 'XXX TOO MANY VARIABLES!! : MAXVAR=', MAXVAR
203: C stop 999
204: C end if
205: C
206: C ... GET VARIABLE NAME AND SIZE DATA IN ( ... ) ...
207: C
208: C VNAME(NVARS) = ' '
209: C
210: C K = INDEX(LINE(3:),' ') + 2
211: C K2 = INDEXR(LINE(K+1:),' ') + K
212: C
213: C .... WITH DIMENSION SPECIFICATION ....
214: C
215: C if ( LINE(K2:K2).eq.'(' ) then
216: C
217: C call CCOMP( VNAME(NVARS), LNVAR, LINE(3:K2-1), IFL )
218: C call ADDLIN( DATA, IDATA, VNAME(NVARS) )
219: C
220: C CHECK % % % % % % % %
221: C WRITE(6,*) 'CHECK 1 VNAME:',VNAME(NVARS)
222: C
223: C ***** KK = INDEX( LINE(K2+1:), ' ') + K2
224: C
225: C call PAIR( LINE(K2:), '( ', ' )', KK )
226: C KK = K2 + KK - 1
227: C
228: C if ( KK.lt.K2 ) then
229: C write(6,*) 'XXX INVALID LINE ! '
230: C write(6,*) ' LINE : ', LINE
231: C stop 666
232: C end if
233: C call CCOMP( TEMP, LEN(TEMP), LINE(K2:KK), IFL )
234: C
235: C LDIM = LENC(TEMP)
236: C
237: C call ADDLIN( DATA, IDATA, TEMP(:LDIM)//CHAR(0) )
238: C K2 = INDEXR(LINE(KK+1:),' ') + KK
239: C else
240: C K3 = INDEX(LINE(3:K),'(' ) + 3 - 1
241: C if ( K3.lt.3 ) then
242: C call CCOMP( VNAME(NVARS), LNVAR, LINE(3:K), IFL )
243: C
244: C CHECK % % % % % % % %
245: C **** WRITE(6,*) 'CHECK 2 VNAME:',VNAME(NVARS)
246: C
247: C LNM = LENC(VNAME(NVARS))
248: C
249: C call ADDLIN( DATA, IDATA, VNAME(NVARS)(1:LNM)//CHAR(0) )
250: C else
251: C
252: C ***** KK = INDEX( LINE(K3+1:), ' ') + K3
253: C
254: C call PAIR( LINE(K3:), '( ', ' )', KK )
255: C KK = K3 + KK - 1
256: C if ( KK.lt.K3 ) then
257: C write(6,*) 'XXX INVALID LINE ! '
258: C write(6,*) ' LINE : ', LINE
259: C stop 666
260: C end if

```


src/tools/varlist.f

```

261:      call CCOMP( TEMP, LEN(TEMP), LINE(3:KK), IFL )
262:      LDIM   = LENC(TEMP)
263:      VNAME(NVARS) = TEMP(1:INDEX(TEMP,'(')-1)
264:
265: CHECK % % % % % % % %
266: ****  WRITE(6,*) 'CHECK 3 VNAME:',VNAME(NVARS)
267:
268:      K2      = INDEXR(LINE(KK+1:),' ') + KK
269:      call ADDLIN( DATA, IDATA, TEMP(:LDIM)//CHAR(0) )
270:    end if
271:  end if
272: C
273:  write(6,*) ' == VARIABLE ', NVARS, ' : VNAME ', VNAME(NVARS)
274: C
275: C ... VARIABLE TYPE ...
276: C
277:      K      = INDEX(LINE(K2:),' ') + K2 - 1
278:      TYP(NVARS) = LINE(K2:K-1)
279:      write(6,*) '      TYPE : ', TYP(NVARS), '  COMMON : ', CMMN
280: C
281: C .... COMMON NAME ....
282: C
283:      COM(NVARS) = CMMN
284:      write(6,*)
285: C
286: C .... COMMENT DATA IN VARIABLE-NAME LINE ....
287: C
288:      K0      = INDEXR(LINE(K:),' ') + K - 1
289:      if ( K0.ge.K ) then
290:        KK      = INDEX(LINE(K0:),'$') + K0 - 1
291:        if ( KK.lt.K0 ) then
292:          call ADDLIN( DATA, IDATA, LINE(K0:)//CHAR(0) )
293:        else
294:          call ADDLIN( DATA, IDATA, LINE(K0:KK-1) )
295:        end if
296:      end if
297: C
298: C ... IGNORE .....
299: C
300:      else if ( LINE(1:2).eq.'C*' .or. LINE(1:1).eq.'*' ) then
301:        go to 110
302: C
303: C
304: C ... CURRENT COMMON NAME ?? ...
305: C
306: C
307:      else if ( LINE(1:2).eq.' ' ) then
308:        if ( INDEX(LINE,'COMMON').ne.0 ) then
309:          K      = INDEX(LINE,'/')
310:          if ( K.eq.0 ) go to 110
311:          K2      = INDEX(LINE(K+1:),'/') + K
312:          call CCOMP( CMMN, LEN(CMMN), LINE(K+1:K2-1), IFL )
313:          LCMMN   = LENC(CMMN)
314:        end if
315:        go to 110
316: C
317: C
318: C ... COMMENT DATA LINE ...
319: C
320: C
321:      else if ( LINE(1:3).eq.'C ' ) then
322: C
323: C
324:          K      = INDEXR(LINE(2:),' ') + 1
325: C

```

```

326:      if ( K.lt.2 ) then
327:        call ADDLIN( DATA, IDATA, CHAR(0)//' '//CHAR(0) )
328: C
329:      else if ( LINE(K:K).eq.'$' ) then
330:        KK      = INDEX(LINE(K+1:),'$') + K
331:        if ( KK.gt.K ) then
332:          call ADDLIN( DATA, IDATA, LINE(K+1:KK-1) )
333:        else
334:          K1      = LENC(LINE(K+1:)) + K
335:          call ADDLIN( DATA, IDATA, LINE(K+1:K1)//CHAR(0) )
336:        end if
337:      else
338:        KK      = INDEX(LINE(2:),'$') + 1
339:        if ( KK.le.1 ) then
340:          K1      = LENC(LINE(K:)) + K - 1
341:          call ADDLIN( DATA, IDATA, LINE(K:K1)//CHAR(0) )
342:        else
343:          call ADDLIN( DATA, IDATA, LINE(K:KK-1) )
344:        end if
345:      end if
346:    end if
347: C
348:      go to 110
349: C
350: C
351: C
352: C ===== OUTPUT VARIABLE DOCUMENTS =====
353: C
354: C
355:      150 inquire(15,nextrec =IPVAR(NVARS+1))
356: C
357: C
358: C ---- SORT VARIABLES IN ALPHABETICAL ORDER ---
359: C
360: C
361:      call DATE( TEMP(1:8) )
362:      write(20,('( ' ** VARIABLE LIST **      DATE ',A') ) TEMP(1:8)
363: C
364: C
365:      do 160 I = 1, NVARS
366:        ISORT(I) = I
367:      160 continue
368: C
369:      if ( IORD.eq.1 ) then
370:        do 180 J = 1, NVARS - 1
371:          IMIN = J
372:          do 170 I = J + 1, NVARS
373:            if ( VNAME(ISORT(I)).lt.VNAME(ISORT(IMIN)) ) IMIN = I
374:          170 continue
375:          if ( IMIN.ne.J ) then
376:            II = ISORT(IMIN)
377:            ISORT(IMIN) = ISORT(J)
378:            ISORT(J) = II
379:          end if
380:        180 continue
381: C
382: C      ELSE
383: C
384:      end if
385: C
386: C
387: C
388:      do 240 I = 1, NVARS
389:        J = ISORT(I)
390:        L = LDATA(J)

```

src/tools/varlist.f

```

391:      IIREC  = IPVAR(J)
392:      do 190 K = 1, L, LRECL
393:         K2   = MIN(L,K+LRECL-1)
394:         read(15,rec =IIREC) DATA(K:K2)
395:         IIREC = IIREC + 1
396:      190    continue
397:      I1      = INDEX(DATA(1:L),CHAR(0))
398:      TEMP(1:I1-1) = DATA(1:I1-1)
399: C
400: C
401: C      .... OUTPUT DECLARATION, TYPE & COMMON .....
402: C
403: C
404: C      ****      LT = LENC(TYP(J))
405: C      ****      LC = LENC(COM(J))
406: C
407: C      write(20,7000) I, TEMP(I1-1), TYP(J), COM(J)
408: C
409: C      7000    format(/I4,'. ',A,T44,':TYPE ',A,': COMMON ',A,'//')
410: C
411: C
412: C      .... OUTPUT COMMENTS .....
413: C
414: C
415: C      200    I2      = INDEXR(DATA(I1:L),CHAR(0)) + I1 - 1
416: C      if ( I2.lt.I1 ) go to 240
417: C
418: C      I1      = INDEX(DATA(I2:L),CHAR(0)) + I2 - 1
419: C      I3      = I1 - 1
420: C
421: C      TEMP(1:MARGIN) = ' '
422: C
423: C      .....
424: C      DATA(I2:I3) :  A PARAGRAPH .
425: C      .....
426: C
427: C      K1      = I2
428: C      210    continue
429: C      if ( K1.le.I3 ) then
430: C         K2   = MIN(K1+NCOLMN-1,I3)
431: C         if ( K2.lt.I3.and.DATA(K2:K2).ne.' '
432: C            & .and.DATA(K2+1:K2+1).ne.' ' ) then
433: C
434: C            do 220 II = K2, K1, -1
435: C               if ( DATA(II:II).eq.' ' ) then
436: C                  K2   = II
437: C                  go to 230
438: C               end if
439: C            220    continue
440: C            end if
441: C
442: C      230    write(20,'(A,A)') TEMP(1:MARGIN), DATA(K1:K2)
443: C      K1      = K2 + 1
444: C      go to 210
445: C      end if
446: C
447: C      go to 200
448: C      240 continue
449: C
450: C      end
451: C
452: C
453: C ===== RETURN POSITION OF FIRST CHARACTER THAT DOES NOT MATCH A
454: C CHARACTER "CHAR".
455: C

```

```

456: C
457: C      function INDEXR(STR,CHAR)
458: C      character*(*) STR, CHAR*(1)
459: C
460: C      do 100 INDEXR = 1, LEN(STR)
461: C         if ( STR(INDEXR:INDEXR).ne.CHAR ) return
462: C      100 continue
463: C
464: C      INDEXR = 0
465: C      return
466: C      end
467: C
468: C
469: C
470: C ===== RETURN LENGTH OF NON-BLANK PART OF A CHARACTER STRING =====
471: C
472: C
473: C      function LENC(STR)
474: C      character*(*) STR
475: C
476: C      do 100 LENC = LEN(STR), 1, -1
477: C         if ( STR(LENC:LENC).ne.' ' ) return
478: C      100 continue
479: C
480: C      LENC = 0
481: C      return
482: C      end
483: C
484: C
485: C ===== ADD CHARACTERS TO A CHARATER STRING "DATA". =====
486: C      CHARACTER "CHAR".
487: C
488: C
489: C      subroutine ADDLIN( DATA, IDATA, STR )
490: C
491: C      character*(*) DATA, STR
492: C
493: C      LL      = IDATA + LEN(STR)
494: C      if ( LL.gt.LEN(DATA) ) then
495: C         write(6,*) ' XXX TOO MANY CHARACTERS!! STOP '
496: C         stop 999
497: C      end if
498: C      DATA(IDATA+1:IDATA+LEN(STR)) = STR
499: C      CHECK % % % % %
500: C      write(6,*) ' CHCEK ADDLIN: ', IDATA, LL
501: C      IDATA = LL
502: C      return
503: C      end
504: C
505: C
506: C ===== FIND END POSITION OF PAIRED CHARACTERS =====
507: C
508: C
509: C      subroutine PAIR( STR, CH1, CH2, KK )
510: C      character STR*(*), CH1*1, CH2*1
511: C      IC      = 0
512: C      JC      = 0
513: C      do 100 I = 1, LEN(STR)
514: C         if ( STR(I:I).eq.CH1 ) then
515: C            JC = 1
516: C            IC = IC + 1
517: C         else if ( STR(I:I).eq.CH2 ) then
518: C            IC = IC - 1
519: C            if ( IC.lt.0 ) then
520: C               KK = 0

```

src/tools/varlist.f

```
521:          return
522:        end if
523:      end if
524:      if ( JC.gt.0.and.IC.eq.0 ) then
525:        KK      = I
526:        return
527:      end if
528:    100 continue
529:      KK      = 0
530:      return
531:    end
532:  c
533:  c  subroutine GTVVAL( X )
534:  c  return
535:  c  end
536:  C
537:  c  subroutine CCOMP( OUTCH, LN,      INCH,  IFL )
538:  C
539:  C  GMVP/MVP UTILITY
540:  C
541:  C=====
542:  C  PURPOSE:  'COMPRESS' CHARACTER STRING INCH BY REMOVING BLANK
543:  C            INTO CHARACTER STRING 'OUTCH' UPTO LN CHARACTERS.
544:  C            ('OUTCH' & 'INCH' CAN OVERLAP IN MEMORY BUT
545:  C            THE STARTING ADDRESS OF 'OUTCH' MUST PRECEED OR BE SAME AS
546:  C            THAT OF 'INCH' )
547:  C
548:  C            IFL : FLAG
549:  C                  0 = NORMAL END
550:  C                  1 = NUMBER OF NON-BLANK CHARACTERS IN INCH EXCEEDS LN.
551:  C
552:  C  HISTORY:  PROGRAMMED BY M.SASAKI ( 18 MAR 1992 )
553:  C=====
554:  c  character*(*) OUTCH, INCH
555:  c  character*1 CH1
556:  C
557:  c  IFL      = 0
558:  c  LL       = LEN(INCH)
559:  c  NN       = 0
560:  c  do 100 I = 1, LL
561:  c    if ( INCH(I:I).ne.' ' ) then
562:  c      NN      = NN + 1
563:  c    if ( NN.le.LN ) then
564:  c      CH1     = INCH(I:I)
565:  c      OUTCH(NN:NN) = CH1
566:  c    end if
567:  c  end if
568:  c  100 continue
569:  c  if ( NN.lt.LN ) OUTCH(NN+1:LN) = ' '
570:  c  if ( NN.gt.LN ) IFL = 1
571:  c  return
572:  c  end
```

src/mtask/mtask.f

```
1: C
2: C --- MVP/GMVP System ----
3: C
4: C Task controller in multitasking on distributed memory processors.
5: C
6: C Feb 1994 by M.Sasaki
7: C
8: C =====
9: C
10: C This program will be run with workhorse tasks of MVP/GMVP
11: C multitasking processes and controls sequence of calculation on
12: C multiple tasks.
13: C
14: C Initial version works on the fixed source problems only;
15: C
16: C * Started by one of the workhorse routine (may be task #0)
17: C   if necessary.
18: C * Gets request from a task that finished assigned histories
19: C   and assign next batch of histories when remaining histories exist
20: C   else sends a termination signal.
21: C
22: C ( In shared memory multi processing, such a control will be done
23: C   with exclusive reference of a shared memory area. )
24: C
25: C -----
26: C update:
27: C 6 Jun 1997: input NTHIST from parent task to enable
28: C   restart calculation on multi task mode.
29: C 06 Mar 2006: Declare NPART, NTHIST, NHASSIGN as integer*8 and
30: C   call PVMFPACK with INTEGER8 for NPART, NTHIST. (Y.Nagaya)
31: C -----
32: C Variables :
33: C
34: C   programs : name of program using this controller (MVP/GMVP)
35: C
36: C
37: C   character*8 PROGRAM
38: C /#IF INTEGER8
39: C   integer*8 NPART, NTHIST, NHASSIGN
40: C /#ELSE
41: C   integer NPART, NTHIST, NHASSIGN
42: C /#ENDIF
43: C
44: C
45: C /#IF PARA( PVM )
46: C   include '../shared/INC/_PVM PARA'
47: C
48: C   parameter( MAXTSK = 4096 )
49: C   integer TASKID(0:MAXTSK-1)
50: C
51: C   call PVMFMYTID( MYTID )
52: C   if ( MYTID.lt.0 ) then
53: C     call PVMFPERROR( 'MTASK controller', MYTID )
54: C     stop 888
55: C   end if
56: C
57: C ... get basic parameters from the parent task ...
58: C
59: C   call PVMFPARENT( MYPARENT )
60: C   if ( MYPARENT.lt.0 ) then
61: C     call PVMFPERROR( 'MTASK parent?', MYPARENT )
62: C     stop 888
63: C   end if
64: C
65: C   call PVMFRECVC( MYPARENT, -1, IBUFID )
66: C
67: C   ... get program name ...
68: C
69: C   call PVMFUNPACK( INTEGER4, NLP, 1, 1, INFO )
70: C   call PVMFUNPACK( STRING, PROGRAM, NLP, 1, INFO )
71: C
72: C /#IF INTEGER8
73: C *   call PVMFUNPACK( INTEGER8, NPART, 1, 1, INFO )
74: C /#ELSE
75: C   call PVMFUNPACK( INTEGER4, NPART, 1, 1, INFO )
76: C /#ENDIF
77: C   call PVMFUNPACK( INTEGER4, NHIST, 1, 1, INFO )
78: C /#IF INTEGER8
79: C *   call PVMFUNPACK( INTEGER8, NTHIST, 1, 1, INFO )
80: C /#ELSE
81: C   call PVMFUNPACK( INTEGER4, NTHIST, 1, 1, INFO )
82: C /#ENDIF
83: C
84: C   call PVMFUNPACK( INTEGER4, NTASK, 1, 1, INFO )
85: C   call PVMFUNPACK( INTEGER4, TASKID, MIN(MAXTSK,NTASK), 1, INFO )
86: C
87: C   IOK = 1
88: C
89: C ... too many task
90: C
91: C   if ( NTASK.gt.MAXTSK ) then
92: C     IOK = 0
93: C   end if
94: C
95: C ... send back OK or not OK signal ...
96: C
97: C   call PVMFINITSEND( PVMDEFAULT, IBUFID )
98: C   call PVMFPACK( INTEGER4, IOK, 1, 1, INFO )
99: C   call PVMFSEND( MYPARENT, 1, INFO )
100: C
101: C   if ( IOK.eq.0 ) then
102: C     call PVMFEXIT( )
103: C     stop 888
104: C   end if
105: C
106: C
107: C
108: C ... wait for requests from workhorse tasks ...
109: C
110: C Cccc NTHIST = 0
111: C
112: C 100 call PVMFRECVC( -1, -1, IBUFID )
113: C
114: C ... confirm sending task & message ID ...
115: C
116: C   call PVMFBUFINFO( IBUFID, NBYTES, MSGID, ITSEND, INFO )
117: C   if ( INFO.lt.0 ) then
118: C     call PVMFPERROR( 'MTASK request?', INFO )
119: C     stop 888
120: C   end if
121: C   Check % % % %
122: C   write(0,*) ' MTASK: tid ',itask, ' msgid ', msgid
123: C   C % % % % %
124: C
125: C ... get task # ...
126: C
127: C   call PVMFUNPACK( INTEGER4, IDTASK, 1, 1, INFO )
128: C   Check % % % %
129: C   write(0,*) ' MTASK: idtask ',idtask
130: C   C % % % % %
```

src/mtask/mtask.f

```
131: C
132: C
133: C === message ID ===
134: C
135: C   999 : terminate this routine ( calculation completed or
136: C       some errors to stop calculation )
137: C       Send back NTHIST (total number of histories assigned)
138: C
139: C   other than 999 : history request.
140: C
141: C   if ( MSGID.ne.999 ) then
142: C
143: C ... assign tasks if remained ...
144: C
145: C   if ( NTHIST.lt.NPART ) then
146: C       NHASSIGN = MIN(NPART-NTHIST,NHIST)
147: C       NTHIST = NTHIST + NHASSIGN
148: C   else
149: C       NHASSIGN = 0
150: C   end if
151: C
152: C ... sendback number of history assigned ...
153: C
154: C       call PVMFINITSEND( PVMDEFAULT, IBUFID )
155: C       call PVMFPACK( INTEGER4, NHASSIGN, 1, 1, INFO )
156: C       call PVMFPACK( INTEGER4, NTHIST, 1, 1, INFO )
157: C       call PVMFSEND( ITSEND, 99, INFO )
158: C Check % % % %
159: C       write(0,*) ' MTASK: nhassign ',nhassign,' nthist ',nthist,
160: C       & ' itsend ', itsend
161: C % % % % %
162: C
163: C       go to 100
164: C
165: C   else
166: C       call PVMFEXIT( INFO )
167: C   end if
168: C
169: C/#ENDIF
170: C       stop
171: C       end
```